

An encoding of Interval Temporal Logic in Isabelle/HOL

Antonio Cau

Ben Moszkowski

David Smallwood

December 18, 2025

Abstract

These Isabelle theories introduce the semantics and syntax of Finite and Infinite Interval Temporal Logic (ITL). The ITL proof system, as introduced in [8, 12], has been encoded and its soundness has been checked. The encoding is shallow using the Intensional Logic technique of [6]. An extensive library of Finite and Infinite ITL theorems, taken from [11], has been checked. Non-empty coinductive lists are used to denote intervals.

Contents

1	Model: Intervals	6
1.1	Extra operations on coinductive lists	6
1.1.1	Auxiliary lemmas	7
1.1.2	lbutlast	8
1.1.3	lfuse	14
1.1.4	ridx and lidx	21
1.1.5	lsub	35
1.1.6	llastlfirst	41
1.1.7	lfusecat	41
1.1.8	kfilter	75
1.1.9	lrev	109
1.1.10	Transfer rules	113
1.2	Non-empty coinductive lists	114
1.2.1	Type definition	114
1.2.2	Code generator setup	116
1.2.3	Connection with 'a llist	118
1.2.4	The nlast element <i>nlast</i>	131
1.2.5	<i>nset</i>	132
1.2.6	<i>nmap</i>	134
1.2.7	Appending two nonempty lazy lists <i>nappend</i>	134
1.2.8	Appending a nonempty lazy list to a lazy list <i>lappendn</i>	136
1.2.9	The length of a nonempty lazy list <i>nlength</i>	137
1.2.10	The nth element of a nonempty lazy list <i>nnth</i>	138
1.2.11	<i>ntake</i>	140
1.2.12	<i>ntaken</i>	144
1.2.13	Concatenating a nonempty lazy list of nonempty lazy lists <i>nconcat</i>	147
1.2.14	<i>nellist-all2</i>	161
1.2.15	From a nonempty lazy list to a lazy list <i>llist-of-nellist</i>	167
1.2.16	<i>ndropn</i>	168
1.2.17	<i>nzip</i>	175
1.2.18	<i>niterates</i>	180
1.2.19	Filtering non-empty lazy lists <i>nfilter</i>	182
1.2.20	Setup for Lifting/Transfer	192
1.3	Extra operations on non-empty conductive lists	196
1.3.1	ndropns	196
1.3.2	Definitions	198
1.3.3	nbutlast	200

1.3.4	nsubn	202
1.3.5	nfuse	207
1.3.6	nridx and nidx	213
1.3.7	nlastnfirst	230
1.3.8	nfusecat	232
1.3.9	nkfilter	247
1.3.10	nrev	288
1.3.11	Transfer rule	294
1.4	Extended Integers	294
1.4.1	Distributive lattice	318
1.4.2	Extended int intervals	320
1.4.3	Relation to <i>enat</i>	324
2	Shallow embedding of Finite and Infinite ITL	326
2.1	Semantics	326
2.1.1	Types of Formulas	326
2.1.2	Semantics of ITL	327
2.1.3	Abbreviations	328
2.1.4	Properties of Operators	335
2.1.5	Soundness Axioms	344
2.1.6	Quantification over State (Flexible) Variables	350
2.1.7	Temporal Quantifiers	350
2.2	Axioms and Rules	350
2.2.1	Rules	351
2.2.2	Axioms	351
2.2.3	Additional Lemmas	352
2.2.4	Quantification	353
2.2.5	Lemmas about <i>current-val</i>	353
2.2.6	Lemmas about <i>next-val</i>	354
2.2.7	Lemmas about <i>fin-val</i>	354
2.2.8	Lemmas about <i>penult-val</i>	355
2.2.9	Basic temporal variables properties	355
2.3	Weak chop operator	356
2.3.1	Propositional reasoning	356
2.3.2	State formulas	358
2.3.3	finite and inf properties	358
2.3.4	Basic Theorems	361
2.3.5	Further Properties Di and Bi	375
2.3.6	Properties of Da and Ba	381
2.3.7	Properties of Fin	389
2.3.8	Properties of Halt	416
2.3.9	Properties of Groups of chops	421
2.4	Strong chop operator	421
2.4.1	Strong Chop axioms	422
2.4.2	ITL operators in terms of SChop	423
2.4.3	Basic Theorems	424
2.4.4	Further Properties Df and Bf	430

2.4.5	Properties of SDa and SBa	438
2.4.6	Properties of SFin	447
2.4.7	Properties of Halt	473
2.4.8	Properties of Groups of strong chops	478
2.5	Chopstar and variants	478
2.5.1	Definitions	479
2.5.2	Semantic lemmas	482
2.5.3	Helper lemmas	493
2.5.4	Properties of Chopstar and Chopplus	493
2.5.5	Kleene Algebra	500
2.5.6	WPowerstar	543
2.5.7	Len	557
2.5.8	Properties of While	560
2.6	Omega and variants	576
2.6.1	Definitions	576
2.6.2	Omega algebra	594
2.6.3	Properties of Omega	616
2.7	Projection operator	627
2.7.1	Definitions	627
2.7.2	Lemmas	629
2.7.3	Soundness of Projection Axioms	714
2.7.4	Axioms	758
2.7.5	Theorems	760
2.8	Until operator	782
2.8.1	Definitions	782
2.8.2	Axioms	783
2.8.3	Theorems	793
2.9	Pi operator	820
2.9.1	Definitions	820
2.9.2	Semantic Lemmas	821
2.9.3	Soundness of Axioms	827
2.9.4	Theorems	858
2.9.5	SChopstar and Pi	872
2.9.6	Omega and Pi	884
2.10	Quantifiers	891
2.11	Time reversal operator	898
2.11.1	Definition	898
2.11.2	Finite theorems	898
2.11.3	Time reversal Rules	901
2.11.4	Properties of Time Reversal	911
2.12	First occurrence operator	935
2.12.1	Definitions	935
2.12.2	Semantic Theorems	937
2.12.3	Bi induction	937
2.12.4	Theorems	942
2.13	Monitors	1012
2.13.1	Syntax	1012

2.13.2	Derived Monitors	1013
2.13.3	Monitor Laws	1016
2.14	Monitor Example	1038
2.15	Interval Temporal Algebra	1044
2.15.1	Definition of Set of intervals and Operations on them	1045
2.15.2	Simplification Lemmas	1047
2.15.3	Algebraic Laws	1051
2.15.4	Derived Laws	1070
2.15.5	Extra Laws	1078
2.15.6	Link between Set of Intervals and ITL	1113
3	Deep embedding of PITL	1120
3.1	Syntax	1120
3.1.1	Primitive formulae	1120
3.1.2	state formula	1120
3.1.3	Derived Boolean Operators	1120
3.1.4	more, empty, skip and wnext	1121
3.1.5	ifinite and inf	1121
3.1.6	weak chop	1121
3.1.7	Box and Diamond Operators	1122
3.1.8	Initial and Final Operators	1122
3.1.9	Big operations	1122
3.2	Semantics	1122
3.2.1	Intervals	1122
3.2.2	Semantics Primitive Operators	1122
3.2.3	Validity	1127
3.3	Guarded Normal Form	1139
3.3.1	Definitions	1139
3.3.2	Mutex lemmas	1142
3.3.3	Add to lemmas	1144
3.3.4	GNF lemmas	1160
4	Deep embedding of QPITL	1192
4.1	Syntax	1192
4.1.1	Primitive formulae	1192
4.1.2	state formula	1192
4.1.3	Derived Boolean Operators	1193
4.1.4	more, empty, skip and wnext	1193
4.1.5	ifinite and inf	1193
4.1.6	weak chop	1194
4.1.7	Box and Diamond Operators	1194
4.1.8	Initial and Final Operators	1194
4.1.9	Forall	1195
4.1.10	Big operations	1195
4.2	Semantics	1195
4.2.1	Intervals	1195
4.2.2	Semantics Primitive Operators	1195

4.2.3	Validity	1232
4.2.4	Quantifier lemmas	1244
4.3	Guarded Normal Form	1254
4.3.1	Definitions	1254
4.3.2	Mutex lemmas	1256
4.3.3	Add to lemmas	1258
4.3.4	GNF lemmas	1274
5	CDT-PITL	1310
5.1	Syntax	1310
5.1.1	Primitive formulae	1310
5.1.2	Derived Boolean Operators	1310
5.1.3	ifinite and inf	1311
5.1.4	weak chop	1311
5.1.5	more, skip, next, wnext, prev and wprev	1311
5.1.6	Box and Diamond Operators	1311
5.1.7	Initial and Final Operators	1312
5.1.8	Horizontal	1312
5.1.9	Compass operators	1312
5.2	Semantics	1314
5.2.1	Intervals	1314
5.2.2	Validity	1328
6	NL-ITL	1340
6.1	Syntax	1340
6.1.1	Primitive formulae	1340
6.1.2	state, introspective, future and past formula	1341
6.1.3	Derived Operators	1342
6.1.4	more, empty, skip and wnext	1343
6.1.5	rfinite, lfinite, rinf and linf	1344
6.1.6	weak chop	1345
6.1.7	Box and Diamond Operators	1345
6.1.8	Initial and Final Operators	1346
6.1.9	Forall	1346
6.1.10	Big operations	1346
6.2	Semantics	1347
6.2.1	Intervals	1347
6.2.2	Semantics Primitive Operators	1347
6.3	Validity	1383

Chapter 1

Model: Intervals

Intervals are defined in terms of non empty coinductive lists.

1.1 Extra operations on coinductive lists

the operations `kfilter`, `lleast`, `lbutlast`, `lidx`, `ridx`, `lfirst`, `lfuse`, `lsub`, `lsubc`, `llastlfirst`, `lltl`, `llbutlast`, `is_lfirst`, `lfusecat` and `lrev` on coinductive lists are defined together with a library of lemmas.

theory *LList-Extras*

imports

Coinductive.Coinductive-List

begin

definition *kfilter* :: ('a \Rightarrow bool) \Rightarrow nat \Rightarrow 'a llist \Rightarrow nat llist
where *kfilter* *P* *n* *xs* = *lmap* *snd* (*lfilter* (*P* \circ *fst*) (*lzip* *xs* (*iterates* *Suc* *n*)))

definition *lleast* :: ('a \Rightarrow bool) \Rightarrow 'a llist \Rightarrow nat
where *lleast* *P* *xs* = (*LEAST* *na*. *na* < *llength* *xs* \wedge *P* (*lnth* *xs* *na*))

primcorec *lbutlast* :: 'a llist \Rightarrow 'a llist
where *lbutlast* *xs* =
 (*case* *xs* of *LNil* \Rightarrow *LNil* |
 (*LCons* *x* *xs'*) \Rightarrow
 (*case* *xs'* of *LNil* \Rightarrow *LNil* |
 (*LCons* *x1* *xs1*) \Rightarrow (*LCons* *x* (*lbutlast* *xs1*))))

simps-of-case *lbutlast-simps* [*simp*, *code*, *nitpick-simp*]: *lbutlast.code*

definition *ridx* :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a llist \Rightarrow bool
where *ridx* *R* *xs* = (\forall *i*. (*Suc* *i*) < *llength* *xs* \longrightarrow *R* (*lnth* *xs* *i*) (*lnth* *xs* (*Suc* *i*)))

definition *lidx* :: nat llist \Rightarrow bool
where *lidx* *xs* \longleftrightarrow (\forall *n*. (*Suc* *n*) < *llength* *xs* \longrightarrow *lnth* *xs* *n* < *lnth* *xs* (*Suc* *n*))

definition $lfirst :: 'a\ llist \Rightarrow 'a$
where $lfirst\ xs = lhd\ xs$

definition $lfuse :: 'a\ llist \Rightarrow 'a\ llist \Rightarrow 'a\ llist$
where $lfuse\ xs\ ys =$
 $(if\ \neg\ lnull\ xs \wedge \neg\ lnull\ ys\ then\ lappend\ xs\ (ltl\ ys)\ else\ lappend\ xs\ ys)$

definition $lsub :: nat \Rightarrow nat \Rightarrow 'a\ llist \Rightarrow 'a\ llist$
where $lsub\ k\ n\ xs = (ltake\ (eSuc\ (n - k))\ (ldrop\ k\ xs))$

definition $lsubc :: nat \Rightarrow nat \Rightarrow 'a\ llist \Rightarrow 'a\ llist$
where $lsubc\ k\ n\ xs = (ltake\ (eSuc\ (n - k))\ (ldrop\ ((min\ k\ (epred\ (llength\ xs))))\ xs))$

definition $llastlfirst :: 'a\ llist\ llist \Rightarrow bool$
where $llastlfirst\ xss = (\forall\ i. (Suc\ i) < llength\ xss \longrightarrow llast\ (lnth\ xss\ i) = lfirst\ (lnth\ xss\ (Suc\ i)))$

definition $lltl :: 'a\ llist\ llist \Rightarrow 'a\ llist\ llist$
where $lltl\ xss = lmap\ ltl\ xss$

definition $llbutlast :: 'a\ llist\ llist \Rightarrow 'a\ llist\ llist$
where $llbutlast\ xss = lmap\ lbutlast\ xss$

abbreviation $is-lfirst \equiv (\lambda xs. \exists b. xs = (LCons\ b\ LNil))$

definition $lrev :: 'a\ llist \Rightarrow 'a\ llist$
where $lrev\ xs = (if\ lfinite\ xs\ then\ llist-of\ (rev\ (list-of\ xs))\ else\ xs)$

1.1.1 Auxiliary lemmas

lemma *enat-min*:

assumes $m \geq enat\ n'$
and $enat\ n < m - enat\ n'$
shows $enat\ n + enat\ n' < m$

using *assms*

by $(metis\ add.commute\ enat.simps(3)\ enat-add-mono\ enat-add-sub-same\ le-iff-add)$

lemma *enat-min-eq*:

assumes $m \geq enat\ n'$
and $enat\ n \leq m - enat\ n'$
shows $enat\ n + enat\ n' \leq m$

using *assms*

by $(metis\ add.commute\ enat.simps(3)\ enat-add-sub-same\ enat-min\ le-iff-add\ le-less)$

lemma *llist-eq-lnth-eq*:

$(xs = ys) \longleftrightarrow (llength\ xs = llength\ ys \wedge (\forall\ i < llength\ xs. lnth\ xs\ i = lnth\ ys\ i))$

proof *auto*

show $llength\ xs = llength\ ys \Longrightarrow \forall i. enat\ i < llength\ ys \longrightarrow lnth\ xs\ i = lnth\ ys\ i \Longrightarrow xs = ys$

proof $(coinduction\ arbitrary:\ xs\ ys)$

```

case (Eq-list xsa ysa)
then show ?case
  proof –
    assume a: llength xsa = llength ysa
    assume b:  $\forall i. \text{enat } i < \text{llength } ysa \longrightarrow \text{lnth } xsa \ i = \text{lnth } ysa \ i$ 
    have 1: lnull xsa = lnull ysa
      using a by auto
    have 2:  $\neg \text{lnull } xsa \longrightarrow$ 
       $\neg \text{lnull } ysa \longrightarrow$ 
      lhd xsa = lhd ysa
      using b lhd-conv-lnth zero-enat-def by force
    have 3:  $\neg \text{lnull } xsa \longrightarrow$ 
       $\neg \text{lnull } ysa \longrightarrow$ 
       $(\exists xs \ ys. \text{ltl } xsa = xs \wedge \text{ltl } ysa = ys \wedge \text{llength } xs = \text{llength } ys \wedge$ 
         $(\forall i. \text{enat } i < \text{llength } ys \longrightarrow \text{lnth } xs \ i = \text{lnth } ys \ i))$ 
      by (metis Extended-Nat.eSuc-mono a b eSuc-enat eSuc-epred llength-eq-0 llength-ltl lnth-ltl)
    show ?thesis using 1 2 3 by blast
  qed
qed
qed

```

lemma *exist-lset-lnth*:

$(\exists x \in \text{lset } xs. P \ x) \longleftrightarrow (\exists i < \text{llength } xs. P \ (\text{lnth } xs \ i))$

by (*metis in-lset-conv-lnth*)

lemma *exist-llength-gr-zero*:

assumes $(\exists x \in \text{lset } xs. P \ x)$

shows $0 < \text{llength } xs$

by (*metis* *assms gr-implies-not-zero gr-zeroI in-lset-conv-lnth*)

1.1.2 lbutlast

lemma *lbutlast-snoc* [*simp*]:

lbutlast (*lappend* *xs* (*LCons* *x* *LNil*)) = *xs*

proof (*cases* *lfinite* *xs*)

case *True*

then show ?*thesis*

proof (*induct* *rule*: *lfinite-induct*)

case (*LNil* *xs*)

then show ?*case* **using** *llist.collapse(1)* **by** *fastforce*

next

case (*LCons* *xs*)

then show ?*case*

proof (*cases* *xs=LNil*)

case *True*

then show ?*thesis* **by** *simp*

next

case *False*

then show ?*thesis*

proof –

```

have 1:  $\exists y \text{ } ys. xs = (LCons \text{ } y \text{ } ys)$ 
  using False llist.exhaust-sel by blast
obtain y ys where 2:  $xs = (LCons \text{ } y \text{ } ys)$ 
  using 1 by blast
have 3:  $lbutlast \text{ } (lappend \text{ } xs \text{ } (LCons \text{ } x \text{ } LNil)) = lbutlast \text{ } (lappend \text{ } (LCons \text{ } y \text{ } ys) \text{ } (LCons \text{ } x \text{ } LNil))$ 
  by (simp add: 2)
have 4:  $lbutlast \text{ } (lappend \text{ } (LCons \text{ } y \text{ } ys) \text{ } (LCons \text{ } x \text{ } LNil)) = lbutlast \text{ } (LCons \text{ } y \text{ } (lappend \text{ } ys \text{ } (LCons \text{ } x \text{ } LNil)))$ 
  by simp
have 5:  $lnull \text{ } ys \implies ?thesis$ 
  by (simp add: 2 lnull-def)
have 6:  $\neg lnull \text{ } ys \implies ?thesis$ 
  by (metis 2 LCons.hyps(3) eq-LConsD lappend-code(2) lbutlast-simps(3) lhd-LCons-ltl)
show ?thesis
  using 5 6 by blast
qed
qed
qed
next
case False
then show ?thesis
  proof (coinduction arbitrary: xs)
    case (Eq-llist xsa)
    then show ?case
      proof -
        have 1:  $lnull \text{ } (lbutlast \text{ } (lappend \text{ } xsa \text{ } (LCons \text{ } x \text{ } LNil))) = lnull \text{ } xsa$ 
          by (metis Eq-llist lappend-inf lbutlast.disc(2) lfinite.simps lhd-LCons-ltl lnull-imp-lfinite)
        have 2:  $\neg lnull \text{ } (lbutlast \text{ } (lappend \text{ } xsa \text{ } (LCons \text{ } x \text{ } LNil))) \longrightarrow$ 
           $\neg lnull \text{ } xsa \longrightarrow$ 
           $lhd \text{ } (lbutlast \text{ } (lappend \text{ } xsa \text{ } (LCons \text{ } x \text{ } LNil))) = lhd \text{ } xsa$ 
          by (metis Eq-llist eq-LConsD lappend-inf lbutlast.disc(1) lbutlast-simps(3) not-llnull-conv)
        have 3:  $\neg lnull \text{ } (lbutlast \text{ } (lappend \text{ } xsa \text{ } (LCons \text{ } x \text{ } LNil))) \longrightarrow$ 
           $\neg lnull \text{ } xsa \longrightarrow$ 
           $(\exists xs. ltl \text{ } (lbutlast \text{ } (lappend \text{ } xsa \text{ } (LCons \text{ } x \text{ } LNil)))) =$ 
           $lbutlast \text{ } (lappend \text{ } xs \text{ } (LCons \text{ } x \text{ } LNil)) \wedge ltl \text{ } xsa = xs \wedge \neg lfinite \text{ } xs$ 
          by (metis Eq-llist eq-LConsD lappend-inf lbutlast-simps(3) lfinite-ltl lhd-LCons-ltl lnull-imp-lfinite)
        show ?thesis using 1 2 3 by auto
      qed
    qed
  qed
qed

lemma lbutlast-ltl:
   $lbutlast \text{ } (ltl \text{ } xs) = ltl \text{ } (lbutlast \text{ } xs)$ 
proof (cases lfinite xs)
case True
then show ?thesis
  proof (induct rule: lfinite-induct)
    case (LNil xs)

```

```

then show ?case by (simp add: lbutlast.ctr(1))
next
case (LCons xs)
then show ?case
  by (simp add: lbutlast.code llist.case-eq-if)
qed
next
case False
then show ?thesis
  proof (coinduction arbitrary: xs)
    case (Eq-llist xsa)
    then show ?case
      proof -
        have 1: lnull (lbutlast (ltl xsa)) = lnull (ltl (lbutlast xsa))
          by (metis Eq-llist lappend-inf lbutlast-snoc lfinite-ltl)
        have 2:  $\neg$  lnull (lbutlast (ltl xsa))  $\longrightarrow$ 
           $\neg$  lnull (ltl (lbutlast xsa))  $\longrightarrow$ 
            lhd (lbutlast (ltl xsa)) = lhd (ltl (lbutlast xsa))
          by (metis Eq-llist lappend-inf lbutlast-snoc lfinite-ltl)
        have 3:  $\neg$  lnull (lbutlast (ltl xsa))  $\longrightarrow$ 
           $\neg$  lnull (ltl (lbutlast xsa))  $\longrightarrow$ 
             $(\exists xs. \text{ltl (lbutlast (ltl xsa))} = \text{lbutlast (ltl xs)} \wedge$ 
               $\text{ltl (ltl (lbutlast xsa))} = \text{ltl (lbutlast xs)} \wedge \neg \text{lfinite xs})$ 
          by (metis Eq-llist lappend-inf lbutlast-snoc lfinite-ltl)
        show ?thesis using 1 2 3 by auto
      qed
    qed
  qed
qed

```

```

lemma lbutlast-not-lfinite:
  assumes  $\neg$  lfinite xs
  shows lbutlast xs = xs
  using assms
  by (coinduction arbitrary: xs)
    (metis lappend-inf lbutlast-snoc lfinite-ltl)

```

```

lemma lbutlast-lfinite:
  lfinite (lbutlast xs)  $\longleftrightarrow$  lfinite xs
proof
  show lfinite (lbutlast xs)  $\implies$  lfinite xs
  proof (induct zs $\equiv$ lbutlast xs rule: lfinite-induct)
    case LNil
    then show ?case using lbutlast-not-lfinite by fastforce
  next
    case LCons
    then show ?case using lbutlast-not-lfinite by fastforce
  qed
show lfinite xs  $\implies$  lfinite (lbutlast xs)
proof (induct rule: lfinite-induct)
  case (LNil xs)

```

```

then show ?case by simp
next
case (LCons xs)
then show ?case by (simp add: lbutlast-ltl)
qed
qed

```

```

lemma llength-lbutlast [simp]:
  llength (lbutlast xs) = epred (llength xs)
by (coinduction arbitrary: xs rule: enat-coinduct)
  (simp,
   metis epred-enat-unfold lbutlast-ltl llength-def llength-eq-0 llength-ltl )

```

```

lemma lbutlast-lappend:
  lbutlast (lappend xs ys) = (if ys = LNil then lbutlast xs else lappend xs (lbutlast ys))
proof (cases lfinite xs)
case True
then show ?thesis
  proof (induct arbitrary: ys rule: lfinite-induct)
  case (LNil xs)
  then show ?case by (simp add: lnull-def)
  next
  case (LCons xs)
  then show ?case
    proof (cases lnull ys )
    case True
    then show ?thesis by (metis lappend-LNil2 lnull-def)
    next
    case False
    then show ?thesis
      proof (cases lnull xs)
      case True
      then show ?thesis
        using LCons.hyps(2) by auto
      next
      case False
      then show ?thesis using LCons by (auto simp add: llist.case-eq-if not-llnull-conv)
        (metis lappend.ctr(2) lbutlast-simps(3) llist.collapse(1) ltl-simps(2))
      qed
    qed
  qed
next
case False
then show ?thesis by (metis lappend-inf lbutlast-snoc)
qed

```

```

lemma lappend-lbutlast-llast-id-lfinite:
  assumes lfinite xs
    ¬ lnull xs
  shows (lappend (lbutlast xs) (LCons (llast xs) LNil)) = xs

```

```

using assms
proof (induct rule: lfinite-induct)
case (LNil xs)
then show ?case by simp
next
case (LCons xs)
then show ?case
  proof (cases xs)
  case lfinite-LNil
  then show ?thesis using LCons by simp
  next
  case (lfinite-LConsI xs x)
  then show ?thesis using LCons
  by (auto simp add: llast-LCons)
    (metis lappend-code(1) lbutlast.ctr(1) llist.collapse(1) ltl-simps(2),
      metis lappend-code(2) lbutlast-simps(3) lhd-LCons-ltl)
qed
qed

```

```

lemma lappend-lbutlast-llast-id-not-lfinite:
  assumes  $\neg$ lfinite xs
     $\neg$  lnull xs
  shows (lappend (lbutlast xs) (LCons (llast xs) LNil)) = xs
using assms
proof (coinduction arbitrary: xs)
case (Eq-llist xsa)
then show ?case
  by (auto simp add: lbutlast-ltl llast-linfinite)
    (metis lfinite-LNil lfinite-ltl llist.collapse(1),
      metis lbutlast.simps(3) lbutlast-not-lfinite)
qed

```

```

lemma lappend-lbutlast-llast-id [simp]:
shows  $\neg$  lnull xs  $\implies$  (lappend (lbutlast xs) (LCons (llast xs) LNil)) = xs
using lappend-lbutlast-llast-id-lfinite lappend-lbutlast-llast-id-not-lfinite by blast

```

```

lemma lbutlast-eq-LNil-conv:
  lbutlast xs = LNil  $\longleftrightarrow$  xs = LNil  $\vee$  ( $\exists x. xs = (LCons x LNil)$ )
by (metis lbutlast.disc-iff(1) lbutlast-simps(2) lhd-LCons-ltl llist.collapse(1) llist.disc(1))

```

```

lemma lbutlast-eq-LCons-conv:
  lbutlast xs = (LCons x ys)  $\longleftrightarrow$  xs = (LCons x (lappend ys (LCons (llast xs) LNil)))
by (metis eq-LConsD lappend-code(2) lappend-lbutlast-llast-id lbutlast.ctr(1) lbutlast-snoc)

```

```

lemma lbutlast-conv-ltake:
  lbutlast xs = ltake (epred (llength xs)) xs
proof (cases lfinite xs)
case True

```

```

then show ?thesis
  proof (induct rule: lfinite-induct)
    case (LNil xs)
    then show ?case by simp
  next
    case (LCons xs)
    then show ?case
      by (metis enat-le-plus-same(2) gen-llength-def lappend-lbutlast-llast-id-lfinite llength-code
        llength-lbutlast ltake-all ltake-lappend1)
  qed
next
case False
then show ?thesis
  proof (coinduction arbitrary: xs)
    case (Eq-llist xsa)
    then show ?case
      by (auto simp add: not-lfinite-llength lbutlast-not-lfinite)
        (metis epred-Infty epred-llength lbutlast-not-lfinite llength-eq-infty-conv-lfinite ltake-epred-ltl)
  qed
qed

```

lemma *lmap-lbutlast*:

```

lmap f (lbutlast xs) = lbutlast (lmap f xs)
by (simp add: lbutlast-conv-ltake)

```

lemma *snocs-eq-iff-lbutlast*:

```

lappend xs (LCons x LNil) = ys  $\longleftrightarrow$ 
  (( lfinite ys  $\wedge$   $\neg$  lnull ys  $\wedge$  lbutlast ys = xs  $\wedge$  llast ys = x)
   $\vee$  ( $\neg$  lfinite ys  $\wedge$  lbutlast xs = ys))
by (metis lappend.disc(2) lappend-inf lappend-lbutlast-llast-id lbutlast.ctr(1) lbutlast-snoc
  lfinite-LNil llast-lappend llast-singleton llist.discI(2))

```

lemma *in-lset-lbutlastD*:

```

x  $\in$  lset(lbutlast xs)  $\implies$  x  $\in$  lset xs
by (metis in-lset-lappend-iff lappend-ltake-ldrop lbutlast-conv-ltake)

```

lemma *in-lset-lbutlast-lappendI*:

```

x  $\in$  lset (lbutlast xs)  $\vee$  (lfinite xs  $\wedge$  x  $\in$  lset(lbutlast ys))  $\implies$ 
  x  $\in$  lset (lbutlast (lappend xs ys))
by (metis empty-iff in-lset-lappend-iff in-lset-lbutlastD lbutlast-lappend lset-LNil)

```

lemma *lnth-lbutlast*:

```

assumes n < llength(lbutlast xs)
shows lnth (lbutlast xs) n = lnth xs n
proof (cases xs)
case LNil
then show ?thesis by simp
next
case (LCons x21 x22)
then show ?thesis by (metis assms lbutlast-conv-ltake llength-lbutlast lnth-ltake)

```

qed

1.1.3 lfuse

lemma *lfuse-conv-lnull*:

$lnull (lfuse\ xs\ ys) \longleftrightarrow lnull\ xs \wedge lnull\ ys$

by (*simp add: lfuse-def*)

lemma *lfuse-LNil-1* [*simp*]:

$lfuse\ LNil\ ys = ys$

by (*simp add: lfuse-def*)

lemma *lfuse-LNil-2* [*simp*]:

$lfuse\ xs\ LNil = xs$

by (*metis lappend-lnull2 lfuse-def llist.discI(1)*)

lemma *lfuse-One-a* [*simp*]:

assumes $\neg lnull\ xs$

shows $lfuse\ (LCons\ (lhd\ xs)\ LNil)\ xs = xs$

using *assms* **by** (*simp add: lfuse-def*)

lemma *lfuse-One-b* [*simp*]:

assumes $\neg lnull\ xs$

shows $lfuse\ xs\ (LCons\ y\ LNil) = xs$

using *assms*

by (*simp add: lfuse-def*)

lemma *lfuse-One-a-var*:

shows $lfuse\ (LCons\ x\ LNil)\ ys = (if\ \neg lnull\ ys\ then\ (LCons\ x\ (ltl\ ys))\ else\ (LCons\ x\ LNil))$

unfolding *lfuse-def* **by** *simp*

lemma *lfuse-One-b-var*:

shows $lfuse\ xs\ (LCons\ y\ LNil) = (if\ \neg lnull\ xs\ then\ xs\ else\ (LCons\ y\ LNil))$

unfolding *lfuse-def* **by** (*simp add: lappend-lnull1*)

lemma *lfuse-LCons-a* [*simp*]:

$lfuse\ (LCons\ x\ xs)\ ys =$

$(if\ lnull\ xs\ then\ (LCons\ x\ (ltl\ ys))\ else\ (LCons\ x\ (lfuse\ xs\ ys)))$

by (*metis lappend-code(2) lappend-lnull1 lfuse-def llist.collapse(1) llist.disc(2) ltl-simps(1)*)

lemma *lfuse-LCons-b* [*simp*]:

$lfuse\ xs\ (LCons\ y\ ys) =$

$(if\ \neg lnull\ xs\ then\ lappend\ xs\ ys\ else\ (LCons\ y\ ys))$

unfolding *lfuse-def* **by** (*simp add: lappend-lnull1*)

lemma *lfuse-simps* [*simp*]:

shows *lhd-lfuse*: $lhd\ (lfuse\ xs\ ys) = (if\ lnull\ xs\ then\ lhd\ ys\ else\ lhd\ xs)$

and *ltl-lfuse*: $ltl\ (lfuse\ xs\ ys) =$

$(if\ lnull\ xs$


```

      then ltl ys
    else (if lnull (ltl ys)
          then ltl xs
          else lappend (ltl xs) (ltl ys)))
by (auto simp add: lfuse-def lappend-lnull2)

```

lemma *lfuse-lbutlast*:

```

assumes llast xs = lfirst ys
shows lfuse xs ys = (if lnull ys then xs else lappend (lbutlast xs) ys)
using assms
by (metis lappend-lbutlast-llast-id lappend-snocL1-conv-LCons2 lbutlast.ctr(1) lfirst-def
      lfuse-LNil-2 lfuse-def lhd-LCons-ltl llist.collapse(1))

```

lemma *lfuse-llength*:

```

shows llength (lfuse xs ys) = (llength xs) + (if lnull xs then llength ys else epred(llength ys))
unfolding lfuse-def by ( simp add: llist.case-eq-if epred-llength)

```

lemma *not-lnull-llength*:

```

  ¬ lnull xs ⟷ 1 ≤ llength xs
by (metis gr-zeroI ileI1 llength-eq-0 not-one-le-zero one-eSuc)

```

lemma *lfuse-llength-atmost-one*:

```

shows llength (lfuse xs ys) ≤ 1 ⟷ llength xs ≤ 1 ∧ llength ys ≤ 1
proof (cases lnull xs)
case True
then show ?thesis
by (metis lfuse-LNil-1 llength-LNil lnull-def zero-le)
next
case False
then show ?thesis unfolding lfuse-llength
by auto
  (meson dual-order.trans enat-le-plus-same(1),
   metis add.commute add-increasing2 co.enat.exhaust-sel not-lnull-llength order-antisym-conv
   order-refl plus-1-eSuc(2) zero-le,
   metis add-decreasing2 epred-1 epred-le-epredI)
qed

```

lemma *lfuse-llength-less-a*:

```

assumes 1 < llength xs
        1 < llength ys
        llast xs = lfirst ys
        lfinite xs
shows llength xs < llength (lfuse xs ys)
unfolding lfuse-def
using assms
by (auto simp add: lfinite-llength-enat)
  (metis co.enat.exhaust-sel epred-llength linorder-neq-iff llength-eq-0 one-eSuc)

```

lemma *lfuse-llength-less-b*:

```

assumes 1 < llength xs

```

```

1 < llength ys
llast xs = lfirst ys
lfinite ys
shows llength ys < llength (lfuse xs ys)
proof -
have 1: lfuse xs ys = lappend (lbutlast xs) ys
by (metis assms(2) assms(3) lfuse-lbutlast llength-lnull not-less-zero)
have 2: llength (lappend (lbutlast xs) ys) = epred(llength xs) + llength ys
by simp
have 3: 0 < epred(llength xs)
by (metis assms(1) co.enat.exhaust-sel gr-zeroI less-numeral-extra(4) not-one-less-zero one-eSuc)
have 4: llength ys < epred(llength xs) + llength ys
using 3 assms(4) lfinite-llength-enat by auto
show ?thesis
using 1 2 4 by presburger
qed

```

```

lemma lfuse-llength-le-a:
llength xs ≤ llength (lfuse xs ys)
by (simp add: lfuse-llength)

```

```

lemma lfuse-llength-le-b:
assumes llast xs = lfirst ys
shows llength ys ≤ llength (lfuse xs ys)
by (simp add: assms lfuse-lbutlast)

```

```

lemma lfuse-lnth-a:
assumes (enat i) < llength xs
shows lnth (lfuse xs ys) i = lnth xs i
using assms
by (simp add: lfuse-def lnth-lappend1)

```

```

lemma lfuse-lnth-b:
assumes llength xs ≤ (enat i)
(enat i) < llength (lfuse xs ys)
shows lnth (lfuse xs ys) i = (lnth ys (i - (the-enat(epred(llength xs)))))
proof (cases ¬lnull xs ∧ ¬lnull ys)
case True
then show ?thesis

```

```

proof -
have 1: lfinite xs
using assms(1) lfinite-ldropn lnull-imp-lfinite lnull-ldropn by blast
obtain n where 15: (enat n) = llength xs using 1 by (metis assms(1) enat-ile)
have 16: (the-enat(epred(llength xs))) = n-1
by (metis 15 epred-enat the-enat.simps)
have 17: 0 < n
by (metis 15 True gr0I llength-eq-0 zero-enat-def)
have 2: lfuse xs ys = lappend xs (ltl ys)
unfolding lfuse-def using assms True by presburger
have 3: n-1 ≤ i

```

```

    by (metis 15 17 Suc-pred' assms(1) enat-ord-simps(2) leD le-imp-less-Suc wlog-linorder-le)
  have 4: (Suc (i - n)) = (i - (n-1))
    by (metis 15 17 Suc-diff-eq-diff-pred Suc-diff-le assms(1) enat-ord-simps(1))
  have 5: lnth (ltl ys) (i - the-enat (llength xs)) = (lnth ys (i - (the-enat (epred (llength xs)))))
    using True 3 lnth-ltl[of ys (i - the-enat (llength xs))] by (metis 15 16 4 the-enat.simps)
  show ?thesis using lnth-lappend[of xs (ltl ys) ] by (simp add: 2 5 assms(1) leD)
qed
next
case False
then show ?thesis using assms unfolding lfuse-def
using lappend-lnull1 by fastforce
qed

```

lemma *lfuse-lnth-c*:

```

assumes epred (llength xs) ≤ (enat i)
        (enat i) < llength (lfuse xs ys)
        llast xs = lfirst ys
shows   lnth (lfuse xs ys) i =
        (if lnull ys then lnth xs (the-enat (epred (llength xs))))
        else lnth ys (i - (the-enat (epred (llength xs)))))
proof (cases ¬ lnull xs ∧ ¬ lnull ys)
case True
then show ?thesis
  using assms
  by (simp add: leD lfuse-lbutlast lnth-lappend)
next
case False
then show ?thesis
  proof (cases lnull xs)
  case True
  then show ?thesis
    using assms
    by (metis epred-0 gr-implies-not-zero lfuse-conv-lnull lfuse-lnth-b llength-lnull)
  next
  case False
  then show ?thesis
    using assms
    by (metis co.enat.exhaust-sel iless-Suc-eq leD lfuse-lbutlast llength-eq-0
        llength-lbutlast lnth-lappend nle-le the-enat.simps)
  qed
qed

```

lemma *lfirst-lfuse-1*:

```

shows   lfirst (lfuse xs ys) = (if ¬ lnull xs then lfirst xs else lfirst ys)
by (simp add: lfirst-def)

```

lemma *llast-lfuse*:

```

assumes ¬ lnull xs
        ¬ lnull ys

```

$lfinite\ xs$
 $lfinite\ ys$
 $llast\ xs = lfirst\ ys$
shows $llast(lfuse\ xs\ ys) = llast\ ys$
using $assms$
by ($metis\ lfirst-def\ lfuse-def\ lhd-LCons-ltl\ llast-LCons\ llast-lappend$)

lemma $lfuse-assoc$:
assumes $\neg lnull\ xs$
 $\neg lnull\ ys$
 $\neg lnull\ zs$
shows $(lfuse\ xs\ (lfuse\ ys\ zs)) = (lfuse\ (lfuse\ xs\ ys)\ zs)$
using $assms$
by ($metis\ lappend-assoc\ lappend-ltl\ lfuse-def\ lfuse-conv-lnull$)

lemma $lfirst-llast$:
assumes $i < llength\ xs$
shows $llast\ (ltake\ (Suc\ i)\ xs) = lfirst\ (ldropn\ i\ xs)$
using $assms$
by ($simp\ add: lfirst-def\ lhd-ldropn\ ltake-Suc-conv-snoc-lnth$)

lemma $ltake-lfuse$:
shows $ltake\ (llength\ xs)\ (lfuse\ xs\ ys) = xs$
by ($metis\ dual-order.refl\ lappend-lnull2\ lfuse-def\ ltake-all\ ltake-lappend1$)

lemma $llast-lfirst-LNil$:
 $llast\ LNil = lfirst\ LNil$
by ($simp\ add: lfirst-def\ lhd-def\ llast-LNil$)

lemma $ldrop-lappend-either-LNil$:
assumes $lnull\ xs \vee lnull\ ys$
shows $ldrop\ (llength\ xs)\ (lappend\ xs\ ys) =$
 $(if\ lfinite\ xs\ then\ ys\ else\ LNil)$
proof –
have $1: lnull\ xs \implies ?thesis$
by ($simp\ add: lappend-lnull1$)
have $2: lnull\ ys \implies ?thesis$
by ($metis\ dual-order.refl\ lappend-LNil2\ ldrop-eq-LNil\ lnull-def$)
show $?thesis$
using $1\ 2\ assms$ **by** $blast$
qed

lemma $ldrop-lfuse$:
assumes $llast\ xs = lfirst\ ys$
shows $ldrop\ (if\ \neg lnull\ xs \wedge \neg lnull\ ys\ then\ epred(llength\ xs)\ else\ (llength\ xs))\ (lfuse\ xs\ ys) =$
 $(if\ lfinite\ xs\ then\ ys\ else\ LNil)$
proof –
have $3: lfuse\ xs\ ys = (if\ \neg lnull\ xs \wedge \neg lnull\ ys$
 $\quad then\ lappend\ (lbutlast\ xs)\ ys$
 $\quad else\ lappend\ xs\ ys)$

by (*meson assms lfuse-def lfuse-lbutlast*)
have 4: *ldrop* ((*if* $\neg \text{lnull } xs \wedge \neg \text{lnull } ys$ then *epred*(*llength* *xs*) else (*llength* *xs*)))
 (*if* $\neg \text{lnull } xs \wedge \neg \text{lnull } ys$
 then *lappend* (*lbutlast* *xs*) *ys*
 else *lappend* *xs* *ys*) = (*if* *lfinite* *xs* then *ys* else *LNil*)
proof (*cases lfinite xs*)
case *True*
then show ?thesis
 by (*simp add: ldrop-lappend lfinite-llength-enat*)
next
case *False*
then show ?thesis
 by *simp*
 (*metis epred-Infty llength-eq-infty-conv-lfinite lnull-imp-lfinite*)
qed
show ?thesis
by (*simp add: 3 4*)
qed

lemma *ldrop-lfuse-a*:
assumes $\neg \text{lnull } xs$
 $\neg \text{lnull } ys$
 $\text{llast } xs = \text{lfirst } ys$
shows *ldrop* (*epred*(*llength* *xs*)) (*lfuse* *xs* *ys*) =
 (*if* *lfinite* *xs* then *ys* else *LNil*)
using *assms*
using *ldrop-lfuse* **by** *force*

lemma *lfuse-ltake-ldrop*:
assumes $i < \text{llength } xs$
shows *lfuse* (*ltake* (*eSuc* *i*) *xs*) (*ldrop* *i* *xs*) = *xs*
using *assms*
by (*metis lappend-ltake-ldrop ldrop-eSuc-conv-ltl ldrop-lnull leD lfuse-def lnull-ldrop*
 ltake.disc(2) zero-ne-eSuc)

lemma *lset-lfuse*:
shows *lset* (*lfuse* *xs* *ys*) =
 (*if* *lfinite* *xs* then
 (*if* $\neg \text{lnull } xs \wedge \neg \text{lnull } ys$ then *lset* *xs* \cup *lset* (*ltl* *ys*) else *lset* *xs* \cup *lset* *ys*)
 else *lset* *xs*)
by (*simp add: lappend-inf lfuse-def*)

lemma *mono-llist-lfuse2* [*partial-function-mono*]:
 $\text{mono-llist } A \implies \text{mono-llist } (\lambda f. \text{lfuse } xs \ (A \ f))$
by (*auto intro!: monotoneI lprefix-lappend-sameI simp add: lfuse-def lnull-lprefix*)

*l*list.case-eq-if fun-ord-def lprefix-ltlI monotone-def dest: monotoneD)
 (metis *l*list.collapse(1) lprefix-ltlI ltl-simps(1))

lemma *mono2mono-lfuse2* [THEN *l*list.mono2mono, cont-intro, simp]:
 shows monotone-lfuse2: monotone (lprefix) (lprefix) (lfuse *xs*)
by (rule monotoneI)
 (metis Coinductive-List.finite-lprefix-def Coinductive-List.finite-lprefix-nitpick-simps(1)
 lfuse-def lprefix-LNil lprefix-lappend-sameI ltl-simps(1) monotoneD monotone-ltl)

lemma *silys*:
 assumes $f = g$
 shows $mcont\ lSup\ (lprefix)\ lSup\ (lprefix)\ f =$
 $mcont\ lSup\ (lprefix)\ lSup\ (lprefix)\ g$
using *assms* **by** auto

lemma *lfuse-LNil-eq-id*:
 $lfuse\ LNil = id$
by (simp add: fun-eq-iff *l*list.case-eq-if)

lemma *mcont2mcont-lfuse2* [THEN *l*list.mcont2mcont, cont-intro, simp]:
 shows *mcont-lfuse2*: $mcont\ lSup\ (lprefix)\ lSup\ (lprefix)\ (lfuse\ xs)$
proof(cases *lfinite xs*)
 case True
 thus ?thesis
proof induct
 case *lfinite-LNil*
 then show ?case **using** *lfuse-LNil-eq-id* **by** simp
 next
 case (*lfinite-LConsI xs x*)
 then show ?case **by** (simp add:monotone-lfuse2)
qed
next
 case False
 hence $lfuse\ xs = (\lambda-. xs)$
by (simp add: fun-eq-iff lappend-inf lfuse-def)
 thus ?thesis **by**(simp add: ccpo.cont-const[OF *l*list-ccpo])
qed

lemma *lfuse-inf*: $\neg\ lfinite\ xs \implies lfuse\ xs\ ys = xs$
by (simp add: lappend-inf lfuse-def)

lemma *l*list-all2-lfuseI:
 assumes 1: *l*list-all2 *P xs ys*
 and 2: $\llbracket\ lfinite\ xs;\ lfinite\ ys\ \rrbracket \implies \textit{llist-all2 *P xs' ys'*
 shows *l*list-all2 *P (lfuse xs xs') (lfuse ys ys')*
proof(cases *lfinite xs*)$

```

case True
with 1 have lfinite ys by(auto dest: llist-all2-lfiniteD)
from 1 2[OF True this] show ?thesis
  proof (coinduction arbitrary: xs ys)
    case LNil
    then show ?case by (simp add: lfuse-conv-lnull llist-all2-lnullD)
    next
    case LCons
    then show ?case
    by (metis (no-types, lifting) lfuse-def llist.rel-sel llist-all2-lappendI)
    qed
next
case False
with 1 have  $\neg$  lfinite ys by(auto dest: llist-all2-lfiniteD)
with False 1 show ?thesis
by (simp add: lfuse-inf)
qed

```

1.1.4 ridx and lidx

lemma *ridx-lidx*:

ridx ($<$) *xs* = *lidx xs*

unfolding *lidx-def ridx-def*

by *simp*

lemma *ridx-expand-1*:

$\text{ridx } R \text{ } xs \longleftrightarrow \text{lnull } xs \vee \text{llength } xs = 1 \vee$
 $(1 < \text{llength } xs \wedge (\forall n. (\text{Suc } n) < \text{llength } xs \longrightarrow R (\text{lnth } xs \text{ } n) (\text{lnth } xs (\text{Suc } n))))$

unfolding *ridx-def*

by (*metis Zero-not-Suc enat-0-iff(1) gr-implies-not-zero iless-eSuc0 linorder-cases llength-eq-0 one-eSuc*)

lemma *lidx-expand-1*:

$\text{lidx } xs \longleftrightarrow \text{lnull } xs \vee \text{llength } xs = 1 \vee$
 $(1 < \text{llength } xs \wedge (\forall n. (\text{Suc } n) < \text{llength } xs \longrightarrow \text{lnth } xs \text{ } n < \text{lnth } xs (\text{Suc } n)))$

using *ridx-lidx ridx-expand-1* **by** *blast*

lemma *ridx-LCons*:

$\text{ridx } R \text{ } (\text{LCons } x \text{ } xs) \longleftrightarrow$
 $\text{lnull } xs \vee$
 $(0 < \text{llength } xs \wedge$
 $(\forall n. (\text{Suc } n) < \text{eSuc } (\text{llength } xs) \longrightarrow R (\text{lnth } (\text{LCons } x \text{ } xs) \text{ } n) (\text{lnth } (\text{LCons } x \text{ } xs) (\text{Suc } n))))$

unfolding *ridx-def*

using *enat-0-iff(1)* **by** *force*

lemma *lidx-LCons*:

$\text{lidx } (\text{LCons } x \text{ } xs) \longleftrightarrow$
 $\text{lnull } xs \vee$
 $(0 < \text{llength } xs \wedge$
 $(\forall n. (\text{Suc } n) < \text{eSuc } (\text{llength } xs) \longrightarrow \text{lnth } (\text{LCons } x \text{ } xs) \text{ } n < \text{lnth } (\text{LCons } x \text{ } xs) (\text{Suc } n))))$

using *ridx-ldx*[of (*LCons* *x xs*)] *ridx-LCons*[of (*<*) *x xs*] by *blast*

lemma *ridx-LCons-conv*:

(*0 < llength xs* \wedge
 $(\forall n. (Suc\ n) < eSuc\ (llength\ xs) \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ (lnth\ (LCons\ x\ xs)\ (Suc\ n))))$
 \longleftrightarrow (*0 < llength xs* \wedge
 $R\ x\ (lhd\ xs) \wedge$
 $(\forall n. 0 \leq n \wedge (Suc\ n) < (llength\ xs) \longrightarrow R\ (lnth\ xs\ n)\ (lnth\ xs\ (Suc\ n))))$)

proof –

have 1: (*0 < llength xs* \wedge
 $(\forall n. (Suc\ n) < eSuc\ (llength\ xs) \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ (lnth\ (LCons\ x\ xs)\ (Suc\ n)))) \longleftrightarrow$
(*0 < llength xs* \wedge
 $(\forall n. 0 \leq n \wedge (Suc\ n) < eSuc\ (llength\ xs) \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ (lnth\ (LCons\ x\ xs)\ (Suc\ n))))$)

by *blast*

have 2: (*0 < llength xs* \wedge
 $(\forall n. 0 \leq n \wedge (Suc\ n) < eSuc\ (llength\ xs) \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ (lnth\ (LCons\ x\ xs)\ (Suc\ n)))) \longleftrightarrow$
(*0 < llength xs* \wedge
 $R\ x\ (lhd\ xs) \wedge$
 $(\forall n. 1 \leq n \wedge (Suc\ n) < eSuc\ (llength\ xs) \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ (lnth\ (LCons\ x\ xs)\ (Suc\ n))))$)

by (*metis Extended-Nat.eSuc-mono One-nat-def eSuc-enat gr-implies-not-zero ldropn-0 less-Suc-eq*

lhd-ldropn lnth-0 lnth-Suc-LCons not-le-imp-less zero-enat-def)

have 3: (*0 < llength xs* \wedge
 $R\ x\ (lhd\ xs) \wedge$
 $(\forall n. 1 \leq n \wedge (Suc\ n) < eSuc\ (llength\ xs) \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ (lnth\ (LCons\ x\ xs)\ (Suc\ n)))) \longleftrightarrow$
(*0 < llength xs* \wedge
 $R\ x\ (lhd\ xs) \wedge$
 $(\forall n. 1 \leq n \wedge (n) < (llength\ xs) \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ (lnth\ (LCons\ x\ xs)\ (Suc\ n))))$)

using *Suc-ile-eq* by *auto*

have 4: (*0 < llength xs* \wedge
 $R\ x\ (lhd\ xs) \wedge$
 $(\forall n. 1 \leq n \wedge (n) < (llength\ xs) \longrightarrow R\ (lnth\ (LCons\ x\ xs)\ n)\ (lnth\ (LCons\ x\ xs)\ (Suc\ n)))) \longleftrightarrow$
(*0 < llength xs* \wedge
 $R\ x\ (lhd\ xs) \wedge$
 $(\forall n. 1 \leq n \wedge (n) < (llength\ xs) \longrightarrow R\ (lnth\ xs\ (n-1))\ (lnth\ xs\ (n))))$)

by (*metis le-add-diff-inverse llist.disc(2) lnth-ltl ltl-simps(2) plus-1-eq-Suc*)

have 5: (*0 < llength xs* \wedge
 $R\ x\ (lhd\ xs) \wedge$
 $(\forall n. 1 \leq n \wedge (n) < (llength\ xs) \longrightarrow R\ (lnth\ xs\ (n-1))\ (lnth\ xs\ (n)))) \longleftrightarrow$
(*0 < llength xs* \wedge
 $R\ x\ (lhd\ xs) \wedge$
 $(\forall n. 0 \leq (n-1) \wedge (Suc\ (n-1)) < (llength\ xs) \longrightarrow R\ (lnth\ xs\ (n-1))\ (lnth\ xs\ (Suc\ (n-1))))$)

by (*metis diff-Suc-1 le0 le-add1 le-add-diff-inverse plus-1-eq-Suc*)

have 6: (*0 < llength xs* \wedge
 $R\ x\ (lhd\ xs) \wedge$
 $(\forall n. 0 \leq (n-1) \wedge (Suc\ (n-1)) < (llength\ xs) \longrightarrow R\ (lnth\ xs\ (n-1))\ (lnth\ xs\ (Suc\ (n-1))))$)

\longleftrightarrow
 $(0 < \text{llength } xs \wedge$
 $R \ x \ (\text{lhd } xs) \wedge$
 $(\forall n. 0 \leq n \wedge (\text{Suc } n) < (\text{llength } xs) \longrightarrow R \ (\text{lnth } xs \ n) \ (\text{lnth } xs \ (\text{Suc } n))))$
by (*metis diff-Suc-1*)
show *?thesis*
using 2 3 4 5 6 **by** *auto*
qed

lemma *lidx-LCons-conv*:
 $(0 < \text{llength } xs \wedge (\forall n. (\text{Suc } n) < \text{eSuc } (\text{llength } xs) \longrightarrow \text{lnth } (LCons \ x \ xs) \ n < \text{lnth } (LCons \ x \ xs) \ (\text{Suc } n)))$
 $\longleftrightarrow (0 < \text{llength } xs \wedge$
 $x < \text{lhd } xs \wedge$
 $(\forall n. 0 \leq n \wedge (\text{Suc } n) < (\text{llength } xs) \longrightarrow \text{lnth } xs \ n < \text{lnth } xs \ (\text{Suc } n)))$
using *ridx-LCons-conv*
by *metis*

lemma *ridx-LCons-1 [simp]*:
 $\text{ridx } R \ (LCons \ x \ xs) \longleftrightarrow \text{lnull } xs \vee (0 < \text{llength } xs \wedge R \ x \ (\text{lhd } xs) \wedge \text{ridx } R \ xs)$
unfolding *ridx-def*
using *ridx-LCons-conv*[*of xs R x*]
by (*auto simp add: enat-0-iff(1)*)

lemma *lidx-LCons-1 [simp]*:
 $\text{lidx } (LCons \ x \ xs) \longleftrightarrow \text{lnull } xs \vee (0 < \text{llength } xs \wedge x < \text{lhd } xs \wedge \text{lidx } xs)$
using *ridx-lidx*[*of LCons x xs*] *ridx-lidx*[*of xs*] *ridx-LCons-1*[*of (<) x xs*] **by** *blast*

lemma *ridx-less*:
assumes *ridx R xs*
 $\text{Suc}(n+k) < \text{llength } xs$
 $\text{transp } R$
shows $R \ (\text{lnth } xs \ n) \ (\text{lnth } xs \ (\text{Suc}(n+k)))$
using *assms* **unfolding** *ridx-def*
proof (*induct k*)
case 0
then show *?case* **by** *simp*
next
case (*Suc k*)
then show *?case*
by (*metis Suc-ile-eq add-Suc-right order-less-imp-le transpE*)
qed

lemma *lidx-less*:
assumes *lidx xs*
 $\text{Suc}(n+k) < \text{llength } xs$
shows $\text{lnth } xs \ n < \text{lnth } xs \ (\text{Suc}(n+k))$
using *assms* *ridx-lidx* *ridx-less*[*of (<) xs n k*] *transp-on-less* **by** *blast*

```

lemma ridx-less-eq:
  assumes ridx R xs
     $k \leq j$ 
     $j < \text{length } xs$ 
    transp R
    reflp R
  shows  $R (\text{lnth } xs \ k) (\text{lnth } xs \ j)$ 
proof (cases k=j)
case True
then show ?thesis using assms by (simp add: reflp-def)
next
case False
then show ?thesis using assms
by (metis less-iff-Suc-add order-neq-le-trans ridx-less)
qed

```

```

lemma lidx-less-eq:
  assumes lidx xs
     $k \leq j$ 
     $j < \text{length } xs$ 
  shows  $\text{lnth } xs \ k \leq \text{lnth } xs \ j$ 
using assms ridx-lidx ridx-less-eq[of ( $<$ ) xs k j ]
by (metis dual-order.order-iff-strict less-iff-Suc-add lidx-less)

```

```

lemma ridx-gr-first:
  assumes ridx R xs
     $0 < i$ 
     $i < \text{length } xs$ 
    transp R
  shows  $R (\text{lnth } xs \ 0) (\text{lnth } xs \ i)$ 
using assms ridx-less[of  $R \ xs \ 0 \ i-1$  ] unfolding ridx-def by simp

```

```

lemma lidx-gr-first:
  assumes lidx xs
     $0 < i$ 
     $i < \text{length } xs$ 
  shows  $(\text{lnth } xs \ 0) < \text{lnth } xs \ i$ 
using assms lidx-less[of  $xs \ 0 \ i-1$  ] unfolding lidx-def by simp

```

```

lemma ridx-ltake-a:
  assumes ridx R xs
     $n \leq \text{length } xs$ 
  shows  $\text{ridx } R (\text{ltake } n \ xs)$ 
using assms
unfolding ridx-def
by simp
  (metis Suc-ile-eq dual-order.strict-trans1 lnth-ltake order-less-imp-le)

```

```

lemma lidx-ltake-a:
  assumes lidx xs

```

```

       $n \leq \text{length } xs$ 
shows    $\text{lidx } (l\text{take } n \text{ } xs)$ 
using  $\text{assms}$ 
using  $\text{ridx-lidx ridx-ltake-a}$  by  $\text{blast}$ 

lemma  $\text{ridx-lappend-lfinite}$ :
assumes  $\text{lfinite } xs$ 
shows    $\text{ridx } R (\text{lappend } xs \text{ } ys) \longleftrightarrow$ 
            $\text{ridx } R \text{ } xs \wedge ((\text{lnull } xs \vee \text{lnull } ys) \vee R (\text{llast } xs) (\text{lhs } ys)) \wedge \text{ridx } R \text{ } ys$ 
using  $\text{assms}$ 
proof ( $\text{induction } xs \text{ arbitrary: } ys$ )
case  $\text{lfinite-LNil}$ 
then show  $?case$  by ( $\text{simp add: ridx-expand-1}$ )
next
case ( $\text{lfinite-LConsI } xs \text{ } x$ )
then show  $?case$ 
  proof ( $\text{cases lnull } xs$ )
    case  $\text{True}$ 
      then show  $?thesis$ 
      by ( $\text{metis lappend-code(1) lappend-code(2) llast-singleton llength-LCons llist.collapse(1)}$ 
           $\text{one-eSuc order-neq-le-trans ridx-LCons-1 ridx-expand-1 zero-le}$ )
    next
      case  $\text{False}$ 
        then show  $?thesis$ 
        by ( $\text{simp add: lfinite-LConsI.IH llast-LCons}$ )
  qed
qed

lemma  $\text{lidx-lappend-lfinite}$ :
assumes  $\text{lfinite } xs$ 
shows    $\text{lidx } (\text{lappend } xs \text{ } ys) \longleftrightarrow$ 
            $\text{lidx } xs \wedge ((\text{lnull } xs \vee \text{lnull } ys) \vee (\text{llast } xs) < (\text{lhs } ys)) \wedge \text{lidx } ys$ 
using  $\text{assms}$  by ( $\text{metis ridx-lappend-lfinite ridx-lidx}$ )

lemma  $\text{ridx-ldrop}$ :
assumes  $\text{ridx } R \text{ } xs$ 
            $n \leq \text{length } xs$ 
shows    $\text{ridx } R (\text{ldrop } n \text{ } xs)$ 
proof –
  have  $1: xs = \text{lappend } (l\text{take } n \text{ } xs) (\text{ldrop } n \text{ } xs)$ 
    by ( $\text{simp add: lappend-ltake-ldrop}$ )
  have  $2: \neg \text{lfinite } (l\text{take } n \text{ } xs) \implies ?thesis$ 
    by ( $\text{simp add: ridx-expand-1}$ )
  have  $3: \text{lfinite } (l\text{take } n \text{ } xs) \implies \text{ridx } R (\text{lappend } (l\text{take } n \text{ } xs) (\text{ldrop } n \text{ } xs)) \longleftrightarrow$ 
            $\text{ridx } R (l\text{take } n \text{ } xs) \wedge$ 
            $((\text{lnull } (l\text{take } n \text{ } xs) \vee \text{lnull } (\text{ldrop } n \text{ } xs)) \vee R (\text{llast } (l\text{take } n \text{ } xs)) (\text{lhs } (\text{ldrop } n \text{ } xs))) \wedge$ 
            $\text{ridx } R (\text{ldrop } n \text{ } xs)$ 
    using  $\text{ridx-lappend-lfinite[of } (l\text{take } n \text{ } xs) \text{ } R (\text{ldrop } n \text{ } xs)]$  by  $\text{blast}$ 
  have  $4: \text{lfinite } (l\text{take } n \text{ } xs) \implies ?thesis$ 
    using  $1 \ 3 \text{ assms}$  by  $\text{metis}$ 

```

show *?thesis* **using** 2 4 **by** *blast*
qed

lemma *lidx-ldrop*:
assumes *lidx xs*
 $n \leq \text{length } xs$
shows *lidx (ldrop n xs)*
using *assms ridx-ldrop ridx-lidx* **by** *blast*

lemma *ridx-ltake-all*:
assumes $\bigwedge n. n \leq \text{length } xs \implies \text{ridx } R \text{ (ltake (enat } n) \text{) } xs$
shows *ridx R xs*
using *assms*
unfolding *ridx-def*
by *auto*
(metis Suc-ile-eq dual-order.refl eSuc-enat ile-eSuc lessI lnth-ltake)

lemma *lidx-ltake-all*:
assumes $\bigwedge n. n \leq \text{length } xs \implies \text{lidx (ltake (enat } n) \text{) } xs$
shows *lidx xs*
using *assms ridx-ltake-all ridx-lidx* **by** *blast*

lemma *ridx-ltake*:
assumes *ridx R (ltake n xs)*
 $n \leq \text{length } xs$
 $k \leq n$
shows *ridx R (ltake (enat k) xs)*
using *assms*
using *ridx-ltake-a* **by** *fastforce*

lemma *lidx-ltake*:
assumes *lidx (ltake n xs)*
 $n \leq \text{length } xs$
 $k \leq n$
shows *lidx (ltake (enat k) xs)*
using *assms ridx-ltake ridx-lidx* **by** *blast*

lemma *lidx-imp-lsorted*:
assumes *lidx xs*
shows *lsorted xs*
using *assms*
by *(metis (no-types, lifting) less-imp-le lhd-LCons-ltl lidx-LCons-1 lsorted-coinduct')*

lemma *lidx-imp-ldistinct*:
assumes *lidx xs*
shows *ldistinct xs*
using *assms*
proof *(coinduction arbitrary: xs)*
case *(ldistinct xsa)*
then show *?case*

```

proof –
  have 1:  $\text{lhd } xsa \notin \text{lset } (\text{ltl } xsa)$ 
    by (metis empty-iff lappend-code(1) lappend-lnull2 ldistinct(1) ldistinct(2) leD lhd-LCons-ltl
      lidx-LCons-1 lidx-imp-lsorted lmember-code(2) lset-LNil lset-lmember lsortedD)
  have 2:  $((\exists xs. \text{ltl } xsa = xs \wedge \text{lidx } xs) \vee \text{ldistinct } (\text{ltl } xsa))$ 
    unfolding lidx-def
    by (metis Extended-Nat.eSuc-mono eSuc-enat ldistinct(1) ldistinct(2) lhd-LCons-ltl lidx-def
      llength-LCons lnth-ltl)
  show ?thesis
    using 1 2 by auto
qed
qed

```

```

lemma ldistinct-Ex1:
assumes ldistinct xs
   $x \in \text{lset } xs$ 
shows  $\exists !i. i < \text{llength } xs \wedge (\text{lnth } xs \ i) = x$ 
using assms
by (metis in-lset-conv-lnth ldistinct-conv-lnth)

```

```

lemma lidx-lset-eq:
assumes lidx xs
  lidx ys
   $\text{lset } xs = \text{lset } ys$ 
shows  $xs = ys$ 
using assms
by (simp add: lidx-imp-ldistinct lidx-imp-lsorted lsorted-ldistinct-lset-unique)

```

```

lemma ridx-lfuse-lfirst-llast:
assumes ridx R ys
   $(\text{lnth } ys \ 0) = (0::\text{nat})$ 
  ridx R zs
   $(\text{lnth } zs \ 0) = 0$ 
  lfinite ys
  lfinite xs
   $\neg \text{lnull } xs$ 
   $\neg \text{lnull } zs$ 
   $\text{llast } ys = cp$ 
shows  $\text{llast } ys = \text{lfirst}(\text{lmap } (\lambda x. x+cp) \ zs)$ 
using assms unfolding lfirst-def
by (simp add: lnth-0-conv-lhd)

```

```

lemma lidx-lfuse-lfirst-llast:
assumes lidx ys
   $(\text{lnth } ys \ 0) = 0$ 
  lidx zs
   $(\text{lnth } zs \ 0) = 0$ 
  lfinite ys
  lfinite xs
   $\neg \text{lnull } xs$ 

```

$\neg \text{lnull } zs$
 $\text{llast } ys = cp$
shows $\text{llast } ys = \text{lfirst}(\text{lmap } (\lambda x. x+cp) \text{ } zs)$
using *assms*
by (*simp add: lfirst-def lnth-0-conv-lhd*)

lemma *ridx-lfuse-lnth-cp*:
assumes *ridx R ys*
 $(\text{lnth } ys \ 0) = 0$
 $\text{ridx } R \text{ } zs$
 $(\text{lnth } zs \ 0) = 0$
 $\text{lfinite } ys$
 $\text{lfinite } zs$
 $\text{lfinite } xs$
 $\neg \text{lnull } xs$
 $\neg \text{lnull } zs$
 $\neg \text{lnull } ys$
 $\text{llast } ys = cp$
 $\text{llast } zs = \text{the-enat}(\text{epred } (\text{llength } xs)) - cp$
 $i < (\text{llength } zs)$
 $cp < \text{llength } xs$
shows $\text{lnth } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x+cp) \text{ } zs)) \ (\text{the-enat}(\text{epred}(\text{llength } ys)) + i) = cp + (\text{lnth } zs \ i)$
proof –
have 1: $\text{llast } ys = \text{lfirst}(\text{lmap } (\lambda x. x+cp) \text{ } zs)$
by (*metis assms(11) assms(4) assms(9) lfirst-def llist.map-sel(1) lnth-0-conv-lhd plus-nat.add-0*)
have 3: $\text{lfirst}(\text{lmap } (\lambda x. x+cp) \text{ } zs) = cp + (\text{lnth } zs \ 0)$
using 1 *assms(11) assms(4)* **by** *force*
have 8: $i \leq \text{epred}(\text{llength } zs)$
using *assms*
by (*metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0*)
have 81: $\text{enat } (\text{the-enat } (\text{epred } (\text{llength } ys)) + i) \leq$
 $\text{enat } (\text{the-enat } (\text{epred } (\text{llength } ys)) + (\text{the-enat } (\text{epred}(\text{llength } zs))))$
by (*metis 8 add-mono-thms-linordered-semiring(2) assms(6) assms(9) co.enat.exhaust-sel*
 $\text{enat-ord-simps(1) enat-ord-simps(4) enat-the-enat ile-eSuc iless-Suc-eq lfinite-llength-enat}$
 $\text{llength-eq-0 order-le-less-trans}$)
have 9: $\text{epred}(\text{llength } zs) < \text{llength } zs$
by (*metis assms(6) assms(9) co.enat.exhaust-sel ile-eSuc iless-Suc-eq lfinite-llength-enat*
 $\text{llength-eq-0 order-neq-le-trans}$)
have 10: $\text{epred } (\text{llength } ys) \leq \text{enat } (\text{the-enat } (\text{epred } (\text{llength } ys)) + (i::\text{nat}))$
by (*metis assms(10) assms(5) co.enat.exhaust-sel enat-ord-simps(1) enat-the-enat ile-eSuc*
 $\text{infinity-ileE le-add1 lfinite-llength-enat llength-eq-0}$)
have 83: $\text{enat } (\text{the-enat } (\text{epred } (\text{llength } ys)) + (\text{the-enat } (\text{epred}(\text{llength } zs)))) =$
 $\text{enat } (\text{the-enat } (\text{epred } (\text{llength } ys)) + \text{epred}(\text{llength } zs))$
by (*metis 10 9 enat-ord-code(4) enat-the-enat leD order-less-imp-le plus-enat-simps(1)*)
have 84: $\text{enat } (\text{the-enat } (\text{epred } (\text{llength } ys)) + \text{epred}(\text{llength } zs)) =$
 $(\text{epred } (\text{llength } ys) + \text{epred}(\text{llength } zs))$
by (*metis 10 9 enat-ord-code(4) enat-the-enat leD order-less-imp-not-less plus-enat-simps(1)*)
have 85: $\text{epred}(\text{llength } ys) < \text{llength } ys$
by (*metis 10 assms(10) co.enat.exhaust-sel enat-ord-code(4) enat-the-enat ile-eSuc iless-Suc-eq*
 $\text{leD llength-eq-0 order-neq-le-trans}$)

```

have 86:  $\text{length } (\text{lfuse } ys \ (\text{lmap } (\lambda x::\text{nat}. x + cp) \ zs)) = \text{length } ys + \text{epred}(\text{length } zs)$ 
  by (simp add: assms(10) lfuse-length)
have 87:  $(\text{epred } (\text{length } ys) + \text{epred}(\text{length } zs)) < \text{length } ys + \text{epred}(\text{length } zs)$ 
  by (metis 83 84 85 add.commute enat-le-plus-same(2) enat-less-enat-plusI2 enat-the-enat
    infinity-ileE)
have 82:  $\text{enat } (\text{the-enat } (\text{epred } (\text{length } ys)) + (\text{the-enat } (\text{epred}(\text{length } zs)))) <$ 
   $\text{length } (\text{lfuse } ys \ (\text{lmap } (\lambda x::\text{nat}. x + cp) \ zs))$ 
  using 83 84 86 87 by presburger
have 11:  $\text{enat } (\text{the-enat } (\text{epred } (\text{length } ys)) + i) < \text{length } (\text{lfuse } ys \ (\text{lmap } (\lambda x::\text{nat}. x + cp) \ zs))$ 
  using 81 82 order-le-less-trans by blast
have 12:  $(\text{the-enat } (\text{epred } (\text{length } ys)) + i - \text{the-enat } (\text{epred } (\text{length } ys))) = i$ 
  by auto
have 4:  $\text{lnth } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x+cp) \ zs)) \ (\text{the-enat}(\text{epred}(\text{length } ys)) + i) =$ 
   $(\text{lnth } (\text{lmap } (\lambda x. x+cp) \ zs) \ i)$ 
  by (simp add: 1 10 11 assms(13) assms(9) lfuse-lnth-c)
have 5:  $(\text{lnth } (\text{lmap } (\lambda x. x+cp) \ zs) \ i) = cp + (\text{lnth } zs \ i)$ 
  by (simp add: assms)
show ?thesis
using 4 5 by presburger
qed

```

lemma *lidx-lfuse-lnth-cp*:

```

assumes lidx ys
   $(\text{lnth } ys \ 0) = 0$ 
  lidx zs
   $(\text{lnth } zs \ 0) = 0$ 
  lfinite ys
  lfinite zs
  lfinite xs
   $\neg \text{lnull } xs$ 
   $\neg \text{lnull } zs$ 
   $\neg \text{lnull } ys$ 
   $\text{llast } ys = cp$ 
   $\text{llast } zs = \text{the-enat}(\text{epred } (\text{length } xs)) - cp$ 
   $i < (\text{length } zs)$ 
   $cp < \text{length } xs$ 
shows  $\text{lnth } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x+cp) \ zs)) \ (\text{the-enat}(\text{epred}(\text{length } ys)) + i) = cp + (\text{lnth } zs \ i)$ 
using assms ridx-lidx ridx-lfuse-lnth-cp by blast

```

lemma *ridx-lfuse-lnth-cp-a*:

```

assumes ridx R ys
   $(\text{lnth } ys \ 0) = 0$ 
  ridx R zs
   $(\text{lnth } zs \ 0) = 0$ 
  lfinite ys
  lfinite zs
  lfinite xs
   $\neg \text{lnull } xs$ 
   $\neg \text{lnull } zs$ 
   $\neg \text{lnull } ys$ 

```

$llast\ ys = cp$
 $llast\ zs = the-enat(epred\ (llength\ xs)) - cp$
 $i < (epred(llength\ ys)) + (llength\ zs)$
 $the-enat(epred(llength\ ys)) \leq i$
 $cp < llength\ xs$
shows $lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x+cp)\ zs))\ i = cp + (lnth\ zs\ (i - the-enat(epred(llength\ ys))))$
proof –
have 1: $i = (the-enat\ (epred\ (llength\ ys)) + (i - the-enat\ (epred\ (llength\ ys))))$
using *assms* **by** *fastforce*
have 2: $enat\ (i - the-enat\ (epred\ (llength\ ys))) < llength\ zs$
using *assms*
by (*metis* 1 *enat-add-mono* *epred-enat* *lfinite-llength-enat* *plus-enat-simps*(1) *the-enat.simps*)
show ?thesis
using 1 2 *assms* *ridx-lfuse-lnth-cp*[of *R* *ys* *zs* *xs* *cp* ($i - the-enat(epred(llength\ ys))$)]
by *presburger*
qed

lemma *lidx-lfuse-lnth-cp-a*:

assumes *lidx* *ys*
 $(lnth\ ys\ 0) = 0$
lidx *zs*
 $(lnth\ zs\ 0) = 0$
lfinite *ys*
lfinite *zs*
lfinite *xs*
 $\neg\ lnull\ xs$
 $\neg\ lnull\ zs$
 $\neg\ lnull\ ys$
 $llast\ ys = cp$
 $llast\ zs = the-enat(epred\ (llength\ xs)) - cp$
 $i < (epred(llength\ ys)) + (llength\ zs)$
 $the-enat(epred(llength\ ys)) \leq i$
 $cp < llength\ xs$
shows $lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x+cp)\ zs))\ (i) = cp + (lnth\ zs\ (i - the-enat(epred(llength\ ys))))$
using *assms* *ridx-lidx* *ridx-lfuse-lnth-cp-a* **by** *blast*

lemma *ridx-lfuse-lnth-cp-llast*:

assumes *ridx* *R* *ys*
 $(lnth\ ys\ 0) = 0$
ridx *R* *zs*
 $(lnth\ zs\ 0) = 0$
lfinite *ys*
lfinite *zs*
lfinite *xs*
 $\neg\ lnull\ xs$
 $\neg\ lnull\ zs$
 $\neg\ lnull\ ys$
 $llast\ ys = cp$
 $llast\ zs = the-enat(epred\ (llength\ xs)) - cp$
 $i < (llength\ zs)$

$cp < \text{length } xs$
shows $\text{llast } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x+cp) \ zs)) = (\text{the-enat } (\text{epred}(\text{length } xs)))$
proof –
have 1: $\text{llast } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x+cp) \ zs)) = \text{llast } (\text{lmap } (\lambda x. x+cp) \ zs)$
using *assms*
by (*metis add-cancel-left-left lfinite-lmap lfirst-def llast-lfuse llist.map-disc-iff llist.map-sel(1) lnth-0-conv-lhd*)
have 2: $\text{llast } (\text{lmap } (\lambda x. x+cp) \ zs) = cp + (\text{llast } zs)$
by (*simp add: assms(6) assms(9) llast-lmap*)
have 3: $cp + (\text{llast } zs) = (\text{the-enat } (\text{epred}(\text{length } xs)))$
using *assms*
by (*metis add-diff-inverse-nat co.enat.exhaust-sel enat-ord-simps(2) enat-the-enat ileI1 ile-eSuc infinity-ileE leD lfinite-conv-length-enat length-eq-0*)
show ?thesis **using** 1 2 3 **by** *auto*
qed

lemma *lidx-lfuse-lnth-cp-llast*:

assumes *lidx ys*
 $(\text{lnth } ys \ 0) = 0$
lidx zs
 $(\text{lnth } zs \ 0) = 0$
lfinite ys
lfinite zs
lfinite xs
 $\neg \text{lnull } xs$
 $\neg \text{lnull } zs$
 $\neg \text{lnull } ys$
 $\text{llast } ys = cp$
 $\text{llast } zs = \text{the-enat}(\text{epred } (\text{length } xs)) - cp$
 $i < (\text{length } zs)$
 $cp < \text{length } xs$
shows $\text{llast } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x+cp) \ zs)) = (\text{the-enat } (\text{epred}(\text{length } xs)))$
using *assms ridx-lidx ridx-lfuse-lnth-cp-llast* **by** *blast*

lemma *ridx-lfuse-lnth-cp-infinite*:

assumes *ridx R ys*
 $(\text{lnth } ys \ 0) = (0::\text{nat})$
ridx R zs
 $(\text{lnth } zs \ 0) = 0$
lfinite ys
 $\neg \text{lfinite } zs$
 $\neg \text{lfinite } xs$
 $\neg \text{lnull } ys$
 $\text{llast } ys = cp$
shows $\text{lnth } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x+cp) \ zs)) (\text{the-enat}(\text{epred}(\text{length } ys)) + i) = cp + (\text{lnth } zs \ i)$
proof –
have 1: $\text{llast } ys = \text{lfirst}(\text{lmap } (\lambda x. x+cp) \ zs)$
by (*metis assms(4) assms(6) assms(9) lfirst-def llist.map-sel(1) lnth-0-conv-lhd lnull-imp-lfinite plus-nat.add-0*)
have 3: $\text{lfirst}(\text{lmap } (\lambda x. x+cp) \ zs) = cp + (\text{lnth } zs \ 0)$

```

using 1 assms by auto
have 8:  $i \leq \text{epred}(\text{llength } zs)$ 
  by (metis assms(6) enat.simps(3) ldrop-eq-LNil lfinite-ldrop lfinite-ltl linorder-le-cases llength-ltl)
have 10:  $\text{epred}(\text{llength } ys) \leq \text{enat}(\text{the-enat}(\text{epred}(\text{llength } ys)) + (i::\text{nat}))$ 
  by (metis add.right-neutral add-le-same-cancel1 assms(5) assms(8) co.enat.exhaust-sel
    enat-ord-simps(1) enat-the-enat ile-eSuc infinity-ileE le-zero-eq lfinite-llength-enat
    linorder-le-cases llength-eq-0)
have 11:  $\text{enat}(\text{the-enat}(\text{epred}(\text{llength } ys)) + i) < \text{llength}(\text{lfuse } ys (\text{lmap } (\lambda x::\text{nat}. x + cp) zs))$ 
  using assms
  by (metis 1 enat-ile ldrop-lfuse lfinite-conv-llength-enat lfinite-ldrop
    lfinite-lmap not-le-imp-less)
have 12:  $(\text{the-enat}(\text{epred}(\text{llength } ys)) + i - \text{the-enat}(\text{epred}(\text{llength } ys))) = i$ 
  by auto
have 4:  $\text{lnth}(\text{lfuse } ys (\text{lmap } (\lambda x. x+cp) zs)) (\text{the-enat}(\text{epred}(\text{llength } ys)) + i) =$ 
   $(\text{lnth}(\text{lmap } (\lambda x. x+cp) zs) i)$ 
  using assms
  by (metis 1 10 11 12 lfuse-lnth-c llist.map-disc-iff lnull-imp-lfinite)
have 5:  $(\text{lnth}(\text{lmap } (\lambda x. x+cp) zs) i) = cp + (\text{lnth } zs i)$ 
  using assms
  by (metis 8 add.commute co.enat.exhaust-sel iless-Suc-eq llength-eq-0 lnth-lmap
    lnull-imp-lfinite)
show ?thesis
using 4 5 by presburger
qed

```

lemma *lidx-lfuse-lnth-cp-infinite*:

```

assumes lidx ys
   $(\text{lnth } ys 0) = 0$ 
  lidx zs
   $(\text{lnth } zs 0) = 0$ 
  lfinite ys
   $\neg \text{lfinite } zs$ 
   $\neg \text{lfinite } xs$ 
   $\neg \text{lnull } ys$ 
   $\text{llast } ys = cp$ 
shows  $\text{lnth}(\text{lfuse } ys (\text{lmap } (\lambda x. x+cp) zs)) (\text{the-enat}(\text{epred}(\text{llength } ys)) + i) = cp + (\text{lnth } zs i)$ 
using assms ridx-lidx ridx-lfuse-lnth-cp-infinite by blast

```

lemma *lidx-lfuse-lidx*:

```

assumes lidx ys
   $\text{lnth } ys 0 = 0$ 
  lidx zs
   $\text{lnth } zs 0 = 0$ 
  lfinite ys
   $\neg \text{lnull } ys$ 
   $\text{llast } ys = cp$ 
  lfinite zs
   $\neg \text{lnull } zs$ 
  lfinite xs
   $\text{llast } zs = \text{the-enat}(\text{epred}(\text{llength } xs)) - cp$ 

```

$cp < \text{llength } xs$
 $i < \text{llength } zs$
 $cp < \text{llength } xs$
shows $\text{lidx } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x + cp) \ zs)) \wedge (\text{lnth } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x + cp) \ zs)) \ 0) = 0$
proof –
have 1: $\text{llast } ys = \text{lfirst}(\text{lmap } (\lambda x. x + cp) \ zs)$
by (metis assms(4) assms(7) assms(9) cancel-comm-monoid-add-class.diff-cancel diff-add dual-order.refl lfirst-def llist.map-sel(1) lnth-0-conv-lhd)
have 2: $\text{lfirst } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x + cp) \ zs)) = \text{lfirst } ys$
by (simp add: assms(6) assms(9) lfirst-lfuse-1)
have 3: $\text{llength } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x + cp) \ zs)) = \text{llength } ys + \text{epred}(\text{llength } zs)$
by (simp add: assms(13) assms(6) lfuse-llength)
have 4: $\bigwedge j. j < \text{llength } ys \implies \text{lnth } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x + cp) \ zs)) \ j = \text{lnth } ys \ j$
using lfuse-lnth-a **by** blast
have 40: $\exists k1. \text{llength } ys = (\text{enat } k1)$
by (simp add: assms(5) lfinite-llength-enat)
obtain k1 **where** 41: $\text{llength } ys = (\text{enat } k1)$
using 40 **by** blast
have 42: $(\text{enat } (k1 - 1)) = \text{epred}(\text{llength } ys)$
using 41 epred-enat **by** presburger
have 43: $0 < k1$
using assms 41
by (metis gr0I llength-eq-0 zero-enat-def)
have 44: $\text{the-enat}(\text{epred}(\text{llength } ys)) = (k1 - 1)$
by (metis 42 the-enat.simps)
have 5: $\bigwedge j. \text{epred}(\text{llength } ys) \leq j \wedge j < \text{epred}(\text{llength } ys) + \text{llength } zs \implies$
 $\text{lnth } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x + cp) \ zs)) \ j =$
 $cp + (\text{lnth } zs \ (j - \text{the-enat}(\text{epred}(\text{llength } ys))))$
using assms lidx-lfuse-lnth-cp-a[of ys zs xs cp]
by (metis enat-ord-simps(1) enat-the-enat gr-implies-not-zero infinity-ileE llength-eq-0)
have 45: $\bigwedge j. k1 - 1 \leq j \wedge j < (\text{enat } (k1 - 1)) + \text{llength } zs \implies$
 $\text{lnth } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x + cp) \ zs)) \ j =$
 $cp + (\text{lnth } zs \ (j - (k1 - 1)))$
using 5 44 **by** (metis 42 enat-ord-simps(1))
have 50: $\text{llength}(\text{lfuse } ys \ (\text{lmap } (\lambda x. x + cp) \ zs)) = \text{epred}(\text{llength } ys) + \text{llength } zs$
using assms
by (metis 3 add.commute epred-iadd1 llength-eq-0)
have 51: $\bigwedge j. \text{enat } (\text{Suc } j) < \text{llength } ys \implies$
 $(\text{lnth } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x + cp) \ zs)) \ j) <$
 $(\text{lnth } (\text{lfuse } ys \ (\text{lmap } (\lambda x. x + cp) \ zs)) \ (\text{Suc } j))$
by (metis assms(1) assms(5) eSuc-enat iless-Suc-eq lfinite-conv-llength-enat lfuse-lnth-a lidx-def not-le-imp-less not-less-iff-gr-or-eq)
have 52: $\bigwedge j. k1 - 1 \leq j \wedge (\text{Suc } j) < \text{enat } (k1 - 1) + \text{llength } zs \implies$
 $cp + (\text{lnth } zs \ (j - (k1 - 1))) <$
 $cp + (\text{lnth } zs \ ((\text{Suc } j) - (k1 - 1)))$
using assms(3) **unfolding** lidx-def
by simp
(metis Suc-diff-le add-Suc-right assms(8) enat-ord-simps(2) leD lfinite-llength-enat nat-add-left-cancel-le not-le-imp-less ordered-cancel-comm-monoid-diff-class.add-diff-inverse)

$plus-enat-simps(1))$
have 6: $\bigwedge j. enat (Suc j) < llength(lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs)) \implies$
 $(lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs))\ j) <$
 $(lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs))\ (Suc\ j))$
proof –
fix j
assume $a: enat (Suc j) < llength(lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs))$
show $(lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs))\ j) < (lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs))\ (Suc\ j))$
proof –
have 61: $enat (Suc j) < llength\ ys \implies$
 $(lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs))\ j) < (lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs))\ (Suc\ j))$
using 51 **by** blast
have 62: $k1 - 1 \leq j \wedge (Suc\ j) < llength(lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs)) \implies$
 $(lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs))\ j) < (lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs))\ (Suc\ j))$
by (metis 42 45 50 52 Suc-ile-eq nle-le not-less-eq-eq order-less-imp-le)
show ?thesis
by (metis 41 43 61 62 Suc-diff-1 Suc-mono a enat-ord-simps(2) leI)
qed
qed
show ?thesis **unfolding** lidx-def
by (simp add: 41 43 6 assms(13) assms(2) lfuse-lnth-a)
qed

lemma lidx-lfuse-lidx-infinite:

assumes lidx ys
 $lnth\ ys\ 0 = 0$
 $lidx\ zs$
 $lnth\ zs\ 0 = 0$
 $lfinite\ ys$
 $\neg lnull\ ys$
 $llast\ ys = cp$
 $\neg lfinite\ zs$
 $\neg lfinite\ xs$
shows $lidx\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs)) \wedge (lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs))\ 0) = 0$
proof –
have 1: $llast\ ys = lfirst(lmap\ (\lambda x. x + cp)\ zs)$
by (metis assms(4) assms(7) assms(8) lfirst-def llist.map-sel(1) lnth-0-conv-lhd
 $lnull-imp-lfinite\ plus-nat.add-0)$
have 2: $lfirst\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs)) = lfirst\ ys$
by (simp add: assms(6) lfirst-lfuse-1)
have 4: $\bigwedge j. j < llength\ ys \implies lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs))\ j = lnth\ ys\ j$
using lfuse-lnth-a **by** blast
have 40: $\exists k1. llength\ ys = (enat\ k1)$
by (simp add: assms(5) lfinite-llength-enat)
obtain $k1$ **where** 41: $llength\ ys = (enat\ k1)$
using 40 **by** blast
have 42: $(enat\ (k1 - 1)) = epred(llength\ ys)$
using 41 epred-enat **by** presburger
have 43: $0 < k1$
using assms 41

by (metis gr0I llength-eq-0 zero-enat-def)
 have 44: the-enat(epred(llength ys)) = (k1 - 1)
 by (metis 42 the-enat.simps)
 have 5: $\bigwedge j. \text{epred}(\text{llength } ys) \leq j \wedge j < \text{epred}(\text{llength } ys) + \text{llength } zs \implies$
 $\text{lnth}(\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) j =$
 $cp + (\text{lnth } zs (j - \text{the-enat}(\text{epred}(\text{llength } ys))))$
 using assms lid_x-lfuse-lnth-cp-infinite[of ys zs xs cp]
 by (metis 42 44 enat-ord-simps(1) ordered-cancel-comm-monoid-diff-class.add-diff-inverse)
 have 45: $\bigwedge j. k1 - 1 \leq j \wedge j < (\text{enat } (k1 - 1)) + \text{llength } zs \implies$
 $\text{lnth}(\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) j =$
 $cp + (\text{lnth } zs (j - (k1 - 1)))$
 using 5 44 by (metis 42 enat-ord-simps(1))
 have 46: $\text{llength } ys + \text{epred } (\text{llength } zs) = \text{epred } (\text{llength } ys) + \text{llength } zs$
 by (metis add.commute assms(6) assms(8) epred-iadd1 llength-eq-0 lnull-imp-lfinite)
 have 50: $\text{llength}(\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) = \text{epred}(\text{llength } ys) + \text{llength } zs$
 using lfuse-llength[of ys (lmap (λx::nat. x + cp) zs)]
 llength-lmap[of (λx::nat. x + cp) zs]
 using 46 assms(6) by presburger
 have 51: $\bigwedge j. \text{enat } (\text{Suc } j) < \text{llength } ys \implies$
 $(\text{lnth}(\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) j) <$
 $(\text{lnth}(\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) (\text{Suc } j))$
 by (metis assms(1) assms(5) eSuc-enat iless-Suc-eq lfinite-conv-llength-enat lfuse-lnth-a
 lid_x-def not-le-imp-less not-less-iff-gr-or-eq)
 have 52: $\bigwedge j. k1 - 1 \leq j \wedge (\text{Suc } j) < \text{enat } (k1 - 1) + \text{llength } zs \implies$
 $cp + (\text{lnth } zs (j - (k1 - 1))) <$
 $cp + (\text{lnth } zs ((\text{Suc } j) - (k1 - 1)))$
 using assms(3) unfolding lid_x-def by simp
 (metis Nat.add-diff-assoc assms(8) enat-ile lfinite-conv-llength-enat not-le-imp-less
 plus-1-eq-Suc)
 have 53: $\bigwedge j. k1 - 1 \leq j \wedge (\text{Suc } j) < \text{enat } (k1 - 1) + \text{llength } zs \implies$
 $\text{lnth}(\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) j <$
 $\text{lnth}(\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) (\text{Suc } j)$
 by (metis 45 52 Suc-ile-eq nle-le not-less-eq-eq order-less-imp-le)
 have 6: $\bigwedge j. \text{enat } (\text{Suc } j) < \text{llength}(\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) \implies$
 $(\text{lnth}(\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) j) <$
 $(\text{lnth}(\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) (\text{Suc } j))$
 by (metis 41 42 43 50 51 53 Suc-diff-1 Suc-eq-plus1 add-less-cancel-right
 enat-ord-simps(2) leI)
 show ?thesis unfolding lid_x-def
 by (simp add: 41 43 6 assms lfuse-lnth-a)
 qed

1.1.5 lsub

lemma lsub-eq-lsubc:

assumes $k \leq n$

$n < \text{llength } xs$

shows $\text{lsub } k n xs = \text{lsubc } k n xs$

using assms

unfolding lsub-def lsubc-def

by (auto simp add: min-def)
 (metis co.enat.exhaust-sel enat-ord-simps(1) iless-Suc-eq ldrop-lnull llength-eq-0
 order-le-less-trans)

lemma *lsub-same*:

assumes $(\text{enat } k) < \text{llength } xs$

shows $\text{lsub } k \ k \ xs = (LCons (\text{lnth } xs \ k) \ LNil)$

using *assms*

unfolding *lsub-def*

by (metis *LNil-eq-ltake-iff diff-is-0-eq' enat-0-iff(1) ldrop-enat ldropn-Suc-conv-ldropn
 ltake-eSuc-LCons order.order-iff-strict*)

lemma *lsubc-same*:

assumes $(\text{enat } k) < \text{llength } xs$

shows $\text{lsubc } k \ k \ xs = (LCons (\text{lnth } xs \ k) \ LNil)$

using *assms*

lsub-eq-lsubc[of k k xs] lsub-same[of k xs]

by *fastforce*

lemma *llength-lsub*:

assumes $k \leq n$

$n < \text{llength } xs$

shows $\text{llength } (\text{lsub } k \ n \ xs) = (eSuc \ (n - k))$

proof –

have 1: $\text{llength } (\text{ldrop } (\text{enat } k) \ xs) = (\text{llength } xs - \text{enat } k)$

by (*simp add: ldrop-enat*)

have 2: $\min (eSuc \ (\text{enat } (n - k))) \ (\text{llength } xs - \text{enat } k) = (eSuc \ (n - k))$

by (*metis 1 Suc-diff-le assms(1) assms(2) eSuc-enat enat-llength-ldropn ileI1 ldrop-enat min.absorb1*)

have 3: $\text{llength } (\text{ltake } (eSuc \ (\text{enat } (n - k))) \ (\text{ldrop } (\text{enat } k) \ xs)) = (eSuc \ (n - k))$

by (*simp add: 1 2*)

show *?thesis*

by (*metis 3 idiff-enat-enat lsub-def*)

qed

lemma *llength-lsubc*:

assumes $k \leq n$

$n < \text{llength } xs$

shows $\text{llength } (\text{lsubc } k \ n \ xs) = (eSuc \ (n - k))$

using *assms lsub-eq-lsubc[of k n xs] llength-lsub[of k n xs] by presburger*

lemma *llength-lsub-a*:

shows $\text{llength } (\text{lsub } k \ n \ xs) =$

$\min (eSuc \ (n - k))$

$(\text{if } k = \infty \text{ then } 0::\text{enat} \text{ else } \text{llength } xs - k)$

unfolding *lsub-def*

using *llength-ltake[of eSuc (n-k) (ldrop k xs)] llength-ldrop[of k xs]*

using *idiff-enat-enat by presburger*

lemma *llength-lsubc-a*:

shows $\text{llength } (\text{lsubc } k \ n \ xs) =$

```

    min (eSuc (n - k))
      (if min k (epred (llength xs)) = ∞
       then 0::enat
       else llength xs - min k (epred (llength xs)))
unfolding lsubc-def
using llength-ldrop[of (min k (epred (llength xs))) xs]
using llength-ltake[of (eSuc (n - k)) (ldrop (min k (epred (llength xs))) xs)]
using idiff-enat-enat by presburger

lemma lsub-not-lnull:
  assumes k ≤ n
           n < llength xs
  shows ¬lnull (lsub k n xs)
using assms
by (metis eSuc-enat idiff-enat-enat llength-LNil llength-lsub llist.collapse(1) zero-ne-eSuc)

lemma lsub-llength-gr-one:
  assumes k < n
           n < llength xs
  shows 1 < llength (lsub k n xs)
using llength-lsub-a[of k n xs] assms
by (auto simp add: min-def one-eSuc zero-enat-def llength-lsub)

lemma lsub-lfinite:
  assumes k ≤ n
           n < llength xs
  shows lfinite (lsub k n xs)
using assms
by (simp add: eSuc-enat llength-eq-enat-lfiniteD llength-lsub)

lemma lnth-lsub:
  assumes n < llength xs
           k + j ≤ n
  shows lnth (lsub k n xs) j = (lnth xs (k + j))
proof -
  have 1: enat (j::nat) < eSuc (enat ((n::nat) - (k::nat)))
    using assms(2) by auto
  have 2: lnth (ltake (eSuc (enat (n - k))) (ldrop (enat k) (xs::'a llist))) j =
    lnth (ldrop (enat k) xs) j
    using 1 lnth-ltake by blast
  have 3: lnth (ldrop (enat k) xs) j = (lnth xs (k + j))
    by (metis add.commute assms(1) assms(2) enat-ord-simps(1) ldrop-enat lnth-ldropn order-le-less-trans)
  show ?thesis unfolding lsub-def
  using 2 3 by presburger
qed

lemma lnth-lsub-a:
  assumes n < llength xs
           m ≤ n

```

$k \leq m$
shows $\text{lnth } (\text{lsub } k \ n \ xs) \ (m-k) = (\text{lnth } xs \ m)$
using *assms* **by** (*simp add: lnth-lsub*)

lemma *ltake-lsub*:

assumes $n < \text{llength } xs$
 $m + k \leq n$
shows $\text{ltake } (eSuc \ m) \ (\text{lsub } k \ n \ xs) = \text{lsub } k \ (m+k) \ xs$
proof –
have 1: $\text{ltake } (eSuc \ m) \ (\text{ltake } (eSuc \ (n - k)) \ (\text{ldrop } k \ xs)) =$
 $\text{ltake } (eSuc \ m) \ (\text{ldrop } k \ xs)$
using *ltake-ltake*[*of* (*eSuc m*) (*eSuc (n - k)*) (*ldrop k xs*)]
using *Nat.le-diff-conv2 assms(2)* **by** *auto*
have 2: $\text{ltake } (eSuc \ (m + k - k)) \ (\text{ldrop } k \ xs) = \text{ltake } (eSuc \ m) \ (\text{ldrop } k \ xs)$
by *simp*
show *?thesis*
unfolding *lsub-def* **using** 1 2 **by** *presburger*
qed

lemma *ldrop-lsub*:

assumes $n < \text{llength } xs$
 $m + k \leq n$
shows $\text{ldrop } m \ (\text{lsub } k \ n \ xs) = \text{lsub } (m+k) \ n \ xs$
proof –
have 1: $(\text{ltake } (eSuc \ (n - k)) \ (\text{ldrop } k \ xs)) =$
 $\text{ldrop } k \ (\text{ltake } (eSuc \ n) \ xs)$
by (*metis Suc-diff-le add-leD2 assms(2) eSuc-enat idiff-enat-enat ldrop-ltake*)
have 2: $\text{ldrop } m \ (\text{ldrop } k \ (\text{ltake } (eSuc \ n) \ xs)) =$
 $\text{ldrop } (m + k) \ (\text{ltake } (eSuc \ n) \ xs)$
by *simp*
show *?thesis* **unfolding** *lsub-def* **using** 1 2
by (*simp add: Suc-diff-le assms(2) eSuc-enat ldrop-ltake*)
qed

lemma *ltl-lsub*:

assumes $n < \text{llength } xs$
 $k \leq n$
shows $\text{ltl}(\text{lsub } k \ n \ xs) = (\text{if } k=n \text{ then } LNil \text{ else } \text{lsub } k \ (n-1) \ (\text{ltl } xs))$
proof –
have 1: $(\text{lsub } k \ n \ xs) = (\text{ltake } (eSuc \ (n - k)) \ (\text{ldrop } k \ xs))$
unfolding *lsub-def* **by** *simp*
have 2: $k=n \implies ?thesis$
by (*simp add: assms(1) lsub-same*)
have 25: $k \neq n \implies (\text{epred } (eSuc \ (n - k))) = (eSuc \ (\text{enat } (n - (1::nat) - k)))$
by (*metis Suc-diff-1 assms(2) co.enat.sel(2) diff-commute eSuc-enat order-neq-le-trans zero-less-diff*)
have 3: $k \neq n \implies ?thesis$
using *assms ltl-ltake*[*of* (*eSuc (n - k)*) (*ldrop k xs*)]
 ltl-ldrop [*of* *k xs*]
unfolding *lsub-def*
using 25 **by** *presburger*

show *?thesis*
using 2 3 **by** *blast*
qed

lemma *ltl-ldrop-one*:
 $l\text{tl } xs = l\text{drop } 1 \text{ } xs$
by (*metis ldrop-0 ldrop-ltl one-eSuc*)

lemma *lappend-lsub-ltl-lsub*:

assumes $k \leq n$

$n \leq m$

$m < l\text{length } xs$

shows $l\text{append } (l\text{sub } k \text{ } n \text{ } xs) (l\text{tl } (l\text{sub } n \text{ } m \text{ } xs)) = l\text{sub } k \text{ } m \text{ } xs$

proof –

have 0: $(l\text{tl } (l\text{sub } n \text{ } m \text{ } xs)) = l\text{drop } 1 \text{ } (l\text{sub } n \text{ } m \text{ } xs)$

by (*metis ldrop-0 ldrop-ltl one-eSuc*)

have 1: $(l\text{sub } k \text{ } n \text{ } xs) = (l\text{take } (e\text{Suc } (n - k)) (l\text{drop } k \text{ } xs))$

unfolding *lsub-def* **by** *simp*

have 2: $(l\text{sub } n \text{ } m \text{ } xs) = (l\text{take } (e\text{Suc } (m - n)) (l\text{drop } n \text{ } xs))$

unfolding *lsub-def* **by** *simp*

have 3: $l\text{sub } k \text{ } m \text{ } xs = (l\text{take } (e\text{Suc } (m - k)) (l\text{drop } k \text{ } xs))$

unfolding *lsub-def* **by** *simp*

have 8: $l\text{take } (e\text{Suc } (e\text{nat } ((n::\text{nat}) - (k::\text{nat}))) + e\text{nat } ((m::\text{nat}) - n)) (l\text{drop } (e\text{nat } k) (xs::'a \text{ llist})) =$
 $(l\text{take } (e\text{Suc } (m - k)) (l\text{drop } k \text{ } xs))$

by (*simp add: assms(1) assms(2) eSuc-enat*)

have 9: $(l\text{take } (e\text{nat } (m - n)) (l\text{drop } (e\text{Suc } (e\text{nat } (n - k))) (l\text{drop } (e\text{nat } k) \text{ } xs))) =$
 $l\text{tl}(l\text{take } (e\text{Suc } (m - n)) (l\text{drop } n \text{ } xs))$

by (*metis 0 2 add commute assms(1) eSuc-minus-1 ldrop-ldrop ldrop-ltake le-add-diff-inverse*
plus-1-eSuc(1) plus-enat-simps(1))

have 10: $l\text{take } (e\text{Suc } (e\text{nat } ((n::\text{nat}) - (k::\text{nat}))) + e\text{nat } ((m::\text{nat}) - n)) (l\text{drop } (e\text{nat } k) (xs::'a \text{ llist})) =$
 $l\text{append } (l\text{take } (e\text{Suc } (e\text{nat } (n - k))) (l\text{drop } (e\text{nat } k) \text{ } xs))$
 $(l\text{take } (e\text{nat } (m - n)) (l\text{drop } (e\text{Suc } (e\text{nat } (n - k))) (l\text{drop } (e\text{nat } k) \text{ } xs)))$

using *ltake-plus-conv-lappend[of (eSuc (n - k)) m-n (ldrop k xs)]* **by** *blast*

show *?thesis*

using 1 10 2 3 8 9 **by** *fastforce*

qed

lemma *lsub-lfuse*:

assumes

$k \leq n$

$n \leq m$

$m < l\text{length } xs$

shows $l\text{fuse } (l\text{sub } k \text{ } n \text{ } xs) (l\text{sub } n \text{ } m \text{ } xs) = l\text{sub } k \text{ } m \text{ } xs$

using *assms*

by (*simp add: lappend-lsub-ltl-lsub lfuse-def lsub-not-lnull order-le-less-subst2*)

lemma *llast-lsub*:

assumes *lfinite xs*

$\neg l\text{null } xs$

$k \leq n$

$n < \text{length } xs$
shows $\text{llast } (\text{lsub } k \ n \ xs) = (\text{lnth } xs \ n)$
using *assms*
using *lnth-lsub[of n xs k]*
by (*metis enat-ord-simps(1) idiff-enat-enat llast-conv-lnth llength-lsub*
not-le-imp-less order-less-irrefl ordered-cancel-comm-monoid-diff-class.add-diff-inverse)

lemma *lfirst-lsub*:
assumes $\neg \text{lnull } xs$
 $k \leq n$
 $n < \text{length } xs$
shows $\text{lfirst } (\text{lsub } k \ n \ xs) = (\text{lnth } xs \ k)$
using *assms*
by (*simp add: ldrops-enat lfirst-def lhd-ldropn lsub-def order-le-less-subst2*)

lemma *lsub-lfuse-lidx*:
assumes *lidx ls*
 $\text{lfinite } ls$
 $\text{lfinite } xs$
 $\neg \text{lnull } xs$
 $\text{llast } ls = \text{epred}(\text{length } xs)$
 $(\text{Suc } i) < (\text{length } ls)$
shows $\text{lfuse } (\text{lsub } (\text{lnth } ls \ i) \ (\text{lnth } ls \ (\text{Suc } i)) \ xs) \ (\text{lsub } (\text{lnth } ls \ (\text{Suc } i)) \ (\text{llast } ls) \ xs) =$
 $(\text{lsub } (\text{lnth } ls \ i) \ (\text{llast } ls) \ xs)$
proof –
have 1: $(\text{lnth } ls \ i) \leq (\text{lnth } ls \ (\text{Suc } i))$
by (*simp add: assms(1) assms(6) lidx-less-eq*)
have 2: $\text{llast } ls = (\text{lnth } ls \ (\text{the-enat}(\text{epred } (\text{length } ls))))$
using *llast-conv-lnth*
by (*metis assms(2) assms(6) co.enat.collapse enat.simps(3) enat-ord-simps(4) enat-the-enat*
gr-implies-not-zero ile-eSuc lfinite-llength-enat not-less-iff-gr-or-eq order-neq-le-trans)
have 3: $1 < \text{length } ls$
by (*metis assms(1) assms(6) enat-ord-simps(1) gr-implies-not-zero leD le-add1 lidx-expand-1*
llength-eq-0 one-enat-def plus-1-eq-Suc)
have 4: $(\text{lnth } ls \ (\text{Suc } i)) \leq (\text{llast } ls)$
using 2 *assms(1) assms(2) assms(6) lfinite-llength-enat lidx-less-eq* **by** *fastforce*
have 5: $\text{enat } (\text{lnth } ls \ (\text{Suc } i)) < \text{length } xs$
by (*metis 4 assms(4) assms(5) co.enat.exhaust-sel eSuc-enat enat-ord-simps(2) le-imp-less-Suc*
llength-eq-0)
have 6: $\text{llast } (\text{lsub } (\text{lnth } ls \ i) \ (\text{lnth } ls \ (\text{Suc } i)) \ xs) = (\text{lnth } xs \ (\text{lnth } ls \ (\text{Suc } i)))$
using *llast-lsub[of xs (lnth ls i) (lnth ls (Suc i))]*
using 1 5 *assms(3) assms(4) enat-ord-simps(1)* **by** *blast*
have 7: $\text{lfirst } (\text{lsub } (\text{lnth } ls \ (\text{Suc } i)) \ (\text{llast } ls) \ xs) = (\text{lnth } xs \ (\text{lnth } ls \ (\text{Suc } i)))$
by (*metis 4 assms(4) assms(5) co.enat.exhaust-sel enat-ord-simps(1) ile-eSuc iless-Suc-eq*
lfirst-lsub llength-eq-0 order-less-le)
show *?thesis*
by (*metis 1 4 assms(4) assms(5) co.enat.exhaust-sel enat-ord-simps(1) ile-eSuc iless-Suc-eq*
llength-eq-0 lsub-lfuse order-neq-le-trans)
qed

1.1.6 llastlfirst

lemma *llastlfirst-ridx*:

llastlfirst xss = ridx (λ a b. llast a = lfirst b) xss

by (*simp add: llastlfirst-def ridx-def*)

lemma *llastlfirst-LNil[simp]*:

llastlfirst LNil

unfolding *llastlfirst-def* **by** *simp*

lemma *llastlfirst-LOne[simp]*:

llastlfirst (LCons xs LNil)

unfolding *llastlfirst-def* **by** (*simp add: zero-enat-def*)

lemma *llastlfirst-LCons[simp]*:

assumes $\neg \text{null } xss$

shows *llastlfirst (LCons xs xss) \longleftrightarrow llast xs = lfirst (lfirst xss) \wedge llastlfirst xss*

using *assms* **unfolding** *llastlfirst-def*

by *auto*

(*metis One-nat-def lfirst-def lhd-LCons-ltl lnth-LCons' not-null-llength one-enat-def,*
metis Suc-ile-eq lnth-Suc-LCons,
metis One-nat-def Suc-diff-le Suc-ile-eq add-diff-cancel-left' le-Suc-eq le-add1 lfirst-def
llist.disc(2) lnth-0 lnth-0-conv-lhd lnth-ltl ltl-simps(2) plus-1-eq-Suc)

lemma *llastlfirst-lappend-lfinite*:

assumes *lfinite xss*

shows *llastlfirst (lappend xss yss) \longleftrightarrow*

*(llastlfirst xss \wedge llastlfirst yss \wedge
 (if null xss \vee null yss then True else llast(llast xss) = lfirst(lfirst yss)))*

using *assms*

by (*metis (mono-tags, lifting) lfirst-def llastlfirst-ridx ridx-lappend-lfinite*)

1.1.7 lfusecat

context notes [*function-internals*]

begin

partial-function (*llist*) *lfusecat* :: 'a llist llist \Rightarrow 'a llist

where *lfusecat xss = (case xss of LNil \Rightarrow LNil | LCons xs xss' \Rightarrow lfuse xs (lfusecat xss'))*

end

lemma *lfusecat-simps [simp, code]*:

shows *lfusecat-LNil: lfusecat LNil = LNil*

and *lfusecat-LCons: lfusecat (LCons xs xss) = lfuse xs (lfusecat xss)*

by (*simp-all add: lfusecat.simps*)

declare *lfusecat.mono[cont-intro]*

lemma *mono2mono-lfusecat[THEN llist.mono2mono, cont-intro, simp]*:

shows *monotone-lfusecat: monotone (lprefix) (lprefix) lfusecat*

by(rule llist.fixp-preserves-mono1[OF lfusecat.mono lfusecat-def]) simp

lemma mcont2mcont-lfusecat[THEN llist.mcont2mcont, cont-intro, simp]:

shows mcont-lfusecat: mcont lSup (lprefix) lSup (lprefix) lfusecat

by (rule llist.fixp-preserves-mcont1[OF lfusecat.mono lfusecat-def])

simp

lemmas lfusecat-fixp-parallel-induct =

parallel-fixp-induct-1-1[OF llist-partial-function-definitions llist-partial-function-definitions

lfusecat.mono lfusecat.mono lfusecat-def lfusecat-def, case-names adm LNil step]

lemma llist-all2-lfusecatI:

llist-all2 (llist-all2 A) xss yss

\implies llist-all2 A (lfusecat xss) (lfusecat yss)

proof(induct arbitrary: xss yss rule: lfusecat-fixp-parallel-induct)

case adm

then show ?case **by** (auto split: llist.split intro: llist-all2-lappendI)

next

case LNil

then show ?case **by** (auto split: llist.split intro: llist-all2-lappendI)

next

case (step f g)

then show ?case

using llist-all2-lfuseI **by** (auto split: llist.split intro: llist-all2-lappendI) blast

qed

lemma not-lnull-lset-conv-a:

$\neg \text{lnull } xss \wedge (\forall xs \in \text{lset } xss. \neg \text{lnull } xs) \longleftrightarrow \text{lset } xss \neq \{\} \wedge \text{LNil} \notin \text{lset } xss$

using llist.collapse(1) llist.disc(1) lset-eq-empty **by** blast

lemma not-lnull-lset-conv-b:

$\neg \text{lnull } xss \wedge (\forall xs \in \text{lset } xss. \neg \text{lnull } xs) \longleftrightarrow \text{lset } xss \neq \{\} \wedge \text{lset } xss \cap \{\text{LNil}\} = \{\}$

using llist.collapse(1) **by** force

lemma lfusecat-eq-LNil:

lfusecat xss = LNil \longleftrightarrow lset xss \subseteq {LNil}

proof (induction xss)

case adm

then show ?case

by simp

next

case LNil

then show ?case

by simp

next

case (LCons xs xss)

then show ?case

by simp
 (metis (no-types, lifting) lfuse-conv-lnull llist.disc(1) llist.expand)
 qed

lemma *lfusecat-lfilter-neq-LNil*:
 lfusecat (lfilter ($\lambda xs. \neg \text{lnull } xs$) xss) = lfusecat xss
proof (induct xss)
case adm
then show ?case **by** simp
next
case LNil
then show ?case **by** simp
next
case (LCons xs xss)
then show ?case **by** (simp add: lappend-lnull1 lfuse-def)
 qed

lemma *lprefix-lfusecatI*:
 lprefix xss yss \implies lprefix (lfusecat xss) (lfusecat yss)
by (meson mcont-lfusecat mcont-monoD)

lemma *lset-lltl-llength*:
 ($\forall xs \in \text{lset } xss. \text{llength } xs \leq 1$) \longleftrightarrow ($\forall xs \in \text{lset } (\text{lltl } xss). \text{lnull } xs$)
unfolding lltl-def
by (auto simp add: ltl-ldrop-one)

lemma *lset-lltl-llength-var*:
 ($\forall xs \in \text{lset } xss. \text{llength } xs \leq 1$) $\longleftrightarrow \text{lset}(\text{lltl } xss) \subseteq \{\text{LNil}\}$
using lset-lltl-llength
by (metis all-not-in-conv insert-iff lnull-def subsetI subset-singletonD)

lemma *lset-lltl-llength-var2*:
 ($\forall xs \in \text{lset } xss. \text{llength } xs > 1$) $\implies \text{lset } (\text{lltl } xss) \cap \{\text{LNil}\} = \{\}$
unfolding lltl-def
by auto
 (metis co.enat.exhaust-sel epred-llength less-numeral-extra(4) llength-LNil not-one-less-zero one-eSuc)

lemma *lfusecat-all-empty-or-LNil-a*:
assumes $\text{lset}(\text{lltl } xss) \subseteq \{\text{LNil}\}$
shows $\text{llength } (\text{lfusecat } xss) \leq 1$
using assms
proof (induction xss)
case adm
then show ?case **unfolding** lltl-def **by** simp
next
case LNil

```

then show ?case by simp
next
case (LCons xs xss)
then show ?case
  unfolding ltl-def
  by simp
  (metis LCons.premis lfuse-length-atmost-one llist.set-intros(1) lset-lltl-length-var)
qed

```

```

lemma lfusecat-all-empty-or-LNil-b:
  assumes llength (lfusecat xss) ≤ 1
  shows lset (lltl xss) ⊆ {LNil}
using assms
proof (induction xss)
case adm
then show ?case unfolding ltl-def by simp
next
case LNil
then show ?case unfolding ltl-def by simp
next
case (LCons xs xss)
then show ?case
unfolding ltl-def
by (simp add: lfuse-length-atmost-one ltl-ldrop-one)
qed

```

```

lemma lfusecat-all-empty-or-LNil:
  shows llength (lfusecat xss) ≤ 1 ⟷ lset (lltl xss) ⊆ {LNil}
using lfusecat-all-empty-or-LNil-a lfusecat-all-empty-or-LNil-b by blast

```

```

lemma lfusecat-not-lnull:
  ¬lnull (lfusecat xss) ⟷ ¬lnull xss ∧ (∃ xs ∈ lset ( xss). ¬lnull ( xs))
by (metis lconcat-eq-LNil lfusecat-LNil lfusecat-eq-LNil lnull-def lnull-lconcat mem-Collect-eq
  subsetD subsetI)

```

```

lemma lset-eq-forall-lnull:
  lnull xss ∨ (∀ xs ∈ lset ( xss). lnull ( xs)) ⟷ lset xss ⊆ {LNil}
by (auto simp add: lset-lnull)

```

```

lemma lfusecat-not-lnull-var:
  assumes ¬lnull xss
  shows ∀ xs ∈ lset xss. ¬lnull xs
  shows ¬lnull (lfusecat xss)
using assms
using lfusecat-not-lnull llist.set-sel(1) by blast

```

```

lemma lfirst-lfusecat-lfirst:
  assumes ¬lnull xss
  shows ¬lnull (lfirst xss)

```

```

  shows lfirst(lfusecat xss) = lfirst(lfirst xss)
proof (cases xss)
case LNil
then show ?thesis
using assms(1) llist.disc(1) by blast
next
case (LCons x21 x22)
then show ?thesis
by (metis assms(2) eq-LConsD lfirst-def lfirst-lfuse-1 lfusecat-LCons)
qed

```

```

lemma lfirst-lfusecat:
  assumes ¬lnull xs
  shows lfirst(lfusecat (LCons xs xss)) = lfirst xs
using assms by (simp add: lfirst-def)

```

```

lemma ltl-lfusecat :
  assumes ¬lnull xss
           ¬lnull (lfirst xss)
  shows ltl(lfusecat xss) = lappend (ltl (lhd xss)) (ltl (lfusecat (ltl xss)))
using assms
unfolding lfirst-def
by (metis lappend-nullable lfusecat-LCons lhd-LCons-ltl ltl-lfuse)

```

```

lemma lfusecat-lappend:
  assumes lfinite xss
  shows lfusecat (lappend xss yss) = lfuse (lfusecat xss) (lfusecat yss)
using assms
proof (induct xss arbitrary: yss)
case lfinite-LNil
then show ?case by auto
next
case (lfinite-LConsI xss xs)
then show ?case
proof -
  have 1: lfusecat (lappend (LCons xs xss) yss) = lfuse xs (lfusecat (lappend xss yss))
  by simp
  have 2: lfinite xss
  by (simp add: lfinite-LConsI.hyps(1))
  have 3: (lfusecat (lappend xss yss)) = lfuse (lfusecat xss) (lfusecat yss)
  by (simp add: lfinite-LConsI.hyps(2))
  have 4: lfuse xs (lfuse (lfusecat xss) (lfusecat yss)) =
           lfuse (lfusecat (LCons xs xss)) (lfusecat yss)
  by (metis lappend-nullable lfuse-LNil-2 lfuse-assoc lfuse-def lfusecat-LCons)
  show ?thesis
  by (simp add: 3 4)
qed
qed

```

lemma *lfusecat-split*:
shows $lfusecat\ xss = lfuse\ (lfusecat\ (ltake\ n\ xss))\ (lfusecat\ (ldrop\ n\ xss))$
proof –
have 1: $xss = lappend\ (ltake\ n\ xss)\ (ldrop\ n\ xss)$
by (*simp add: lappend-ltake-ldrop*)
show ?thesis **using** 1 *lfusecat-lappend*[*of* (*ltake* *n* *xss*) (*ldrop* *n* *xss*)] **by** *force*
qed

lemma *lfuse-lfinite*:
shows $lfinite\ (lfuse\ xs\ ys) \longleftrightarrow lfinite\ xs \wedge lfinite\ ys$
by (*metis lfinite-lappend lfinite-ltl lnull-imp-lfinite ltl-lfuse*)

lemma *lfusecat-lfinite-a*:
assumes *lfinite xss*
 $\forall xs \in lset\ xss.\ lfinite\ xs$
shows $lfinite\ (lfusecat\ xss)$
using *assms*
proof (*induct xss*)
case *lfinite-LNil*
then show ?case **by** *auto*
next
case (*lfinite-LConsI xss xs*)
then show ?case
proof (*cases xss=LNil*)
case *True*
then show ?thesis **by** (*simp add: lfinite-LConsI.prem*s)
next
case *False*
then show ?thesis
proof –
have 1: *lfinite xs*
by (*simp add: lfinite-LConsI.prem*s)
have 5: $\forall xs \in lset\ xss.\ lfinite\ xs$
by (*simp add: lfinite-LConsI.prem*s)
have 6: $lfinite\ (lfusecat\ xss)$
using 5 *lfinite-LConsI.hyps*(2) **by** *blast*
have 7: $lfinite\ (lfuse\ xs\ (lfusecat\ xss))$
by (*simp add: 1 6 lfuse-lfinite*)
show ?thesis **by** (*simp add: 7*)
qed
qed
qed

lemma *lfusecat-repeat-LNil* [*simp*]:
 $lfusecat\ (repeat\ LNil) = LNil$
by (*simp add: lfusecat-eq-LNil*)

lemma *llastfirst-lfusecat-llast*:
assumes *lfinite xss*


```

    ¬lnull xss
    ∀ xs ∈ lset xss. ¬lnull xs
    llastlfirst xss
    ∀ xs ∈ lset xss. lfinite xs
  shows   llast(lfusecat xss) = llast(llast xss)
using   assms
proof (induction xss)
case lfinite-LNil
then show ?case
using llist.disc(1) by blast
next
case (lfinite-LConsI xss xs)
then show ?case
  proof (cases xss = LNil)
  case True
  then show ?thesis
  by simp
  next
  case False
  then show ?thesis
  proof -
    have 1: llastlfirst xss
      using False lfinite-LConsI.prem(3) llastlfirst-LCons llist.collapse(1) by blast
    have 2: ∀ xs ∈ lset xss. lfinite xs
      by (simp add: lfinite-LConsI.prem(4))
    have 3: ∀ xs ∈ lset xss. ¬lnull xs
      using lfinite-LConsI.prem(2) by auto
    have 4: lfinite xss
      by (simp add: lfinite-LConsI.hyps)
    have 5: llast (lfusecat xss) = llast (llast xss)
      using 1 2 3 False lfinite-LConsI.IH llist.collapse(1) by blast
    have 6: llast (llast (LCons xs xss)) = llast (llast xss)
      by (metis False llast-LCons llist.collapse(1))
    have 7: (lfusecat (LCons xs xss)) = lfuse xs (lfusecat xss)
      by simp
    have 8: ¬ lnull xs
      by (simp add: lfinite-LConsI.prem(2))
    have 9: lfinite xs
      by (simp add: lfinite-LConsI.prem(4))
    have 10: ¬ lnull (lfusecat xss)
      using 3 False lfusecat-not-llnull-var llist.collapse(1) by blast
    have 11: lfinite (lfusecat xss)
      using 2 4 lfusecat-lfinite-a by blast
    have 111: lfirst (lfusecat xss) = lfirst (lfirst xss)
      by (metis 3 False lfirst-def lfirst-lfusecat-lfirst llist.collapse(1) llist.set-sel(1))
    have 12: llast xs = lfirst (lfusecat xss)
      using 10 111 lfinite-LConsI.prem(3) lfusecat-not-llnull llastlfirst-LCons by auto
    have 13: llast (lfuse xs (lfusecat xss)) = llast (lfusecat xss)
      using llast-lfuse[of xs (lfusecat xss)] 8 9 10 11 12 by blast
    have 14: llast (lfusecat (LCons xs xss)) = llast (lfusecat xss)

```

qed
qed
qed

by *simp*

using *assms*

```

have 7: llastlfirst (LCons xs xss)
  by (simp add: lfinite-LConsI.premis(4))
have 71: lfirst (lfusecat xss) = lfirst(lfirst xss)
  by (metis False insert-iff lfinite-LConsI.premis(2) lfirst-def lfirst-lfusecat-lfirst
      llist.collapse(1) llist.set-sel(1) llist.simps(19))
have 8: llast xs = lfirst (lfusecat xss)
  using 7 71 False llastlfirst-LCons llist.collapse(1) by auto
have 9: llength (lfuse xs (lfusecat xss)) = llength xs + epred(llength(lfusecat xss))
  by (simp add: lfinite-LConsI.premis(2) lfuse-llength)
have 10: (llength(lfusecat xss)) =
  eSuc( $\sum i = 0 .. (the-enat(epred (llength xss))). epred (llength (lnth xss i))$ )
  using 7 False lfinite-LConsI.IH lfinite-LConsI.premis(2) lfinite-LConsI.premis(3) llastlfirst-LCons
  llist.collapse(1) by auto
have 11: (the-enat(epred(llength (LCons xs xss)))) = (1+the-enat(epred(llength xss)))
  using False
  by simp
  (metis False co-enat.sel(2) eSuc-enat lfinite.cases lfinite-LConsI.hyps lfinite-llength-enat
      llength-LCons the-enat.simps)
have 12: ( $\sum i = 0 .. (the-enat(epred(llength (LCons xs xss))))$ ). epred (llength (lnth (LCons xs xss) i)))
=
  epred(llength(lnth (LCons xs xss) 0)) +
  ( $\sum i = 1 .. (1+the-enat(epred(llength xss)))$ ). epred (llength (lnth (LCons xs xss) i)))
  using 11
  by (simp add: sum.atLeast-Suc-atMost)
have 13: epred(llength(lnth (LCons xs xss) 0)) = epred (llength xs)
  by simp
have 14: ( $\sum i = 1 .. (1+the-enat(epred(llength xss)))$ ). epred (llength (lnth (LCons xs xss) i))) =
  ( $\sum i = 0 .. (the-enat(epred(llength xss)))$ ). epred (llength (lnth (LCons xs xss) (Suc i)))
  using sum.shift-bounds-cl-nat-ivl[of  $\lambda k. epred (llength (lnth (LCons xs xss) k))$ ] 0 1
  (the-enat(epred(llength xss)))
  by simp
have 15: ( $\sum i = 0 .. (the-enat(epred(llength xss)))$ ). epred (llength (lnth (LCons xs xss) (Suc i))) =
  ( $\sum i = 0 .. (the-enat(epred(llength xss)))$ ). epred (llength (lnth (xss) (i)))
  by auto
have 16: epred (llength xs) + ( $\sum i = 0 .. (the-enat(epred(llength xss)))$ ). epred (llength (lnth (xss) (i)))
=
  ( $\sum i = 0 .. (the-enat(epred(llength (LCons xs xss))))$ ). epred (llength (lnth (LCons xs xss) i)))
  using 12 13 14 15 by presburger
have 17: llength xs + epred(llength(lfusecat xss)) =
  eSuc(epred (llength xs) + ( $\sum i = 0 .. (the-enat(epred(llength xss)))$ ). epred (llength (lnth (xss)
(i))))
  by (metis 10 eSuc-epred eSuc-plus epred-eSuc lfinite-LConsI.premis(2) llength-eq-0 lset-intros(1))
show ?thesis
  using 16 17 6 9 by presburger
qed
qed
qed

```

lemma llastlfirst-ltake:
 assumes $n \leq \text{llength } xss$

```

      llastlfirst xss
shows   llastlfirst (ltake (enat n) xss)
using   assms
proof (induction n arbitrary: xss)
case 0
then show ?case
by (simp add: llastlfirst-def)
next
case (Suc n)
then show ?case
unfolding llastlfirst-def
by auto
(metis Suc-ile-eq enat-ord-simps(2) less-Suc-eq lnth-ltake order-le-less-trans)
qed

```

lemma lfusecat-ltake:

```

assumes  $\neg$  lnull xss
         $n \leq \text{llength } xss$ 
         $\forall xs \in \text{lset } xss. \neg \text{lnull } xs$ 
         $\forall xs \in \text{lset } xss. \text{lfinite } xs$ 
        llastlfirst xss
shows   lfusecat (ltake (enat n) xss) =
        ltake (if  $n = 0$  then 0 else eSuc( $\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i)))$ )
        (lfusecat xss)

```

proof –

```

have 1: lfinite (ltake (enat n) xss)
using enat-ord-code(4) lfinite-ltake by blast
have 2: llength (ltake (enat n) xss) = n
by (simp add: assms(2))
have 3: (the-enat(epred(llength (ltake (enat n) xss)))) = n-1
using 2 epred-enat the-enat.simps by presburger
have 4:  $\forall xs \in \text{lset } (\text{ltake } (\text{enat } n) \ xss). \neg \text{lnull } xs$ 
by (meson assms(3) lset-ltake subsetD)
have 5:  $\forall xs \in \text{lset } (\text{ltake } (\text{enat } n) \ xss). \text{lfinite } xs$ 
by (meson assms(4) lset-ltake subsetD)
have 6: llastlfirst (ltake (enat n) xss)
by (simp add: assms(2) assms(5) llastlfirst-ltake)
have 7:  $\bigwedge j. 0 < n \wedge j < n-1 \longrightarrow$ 
        epred(llength (lnth (ltake (enat n) xss) j)) =
        epred(llength (lnth xss j))
by (metis diff-le-self dual-order.strict-trans1 enat-ord-simps(2) lnth-ltake)
have 71: llength (lfusecat (ltake (enat n) xss)) =
        (if  $n = 0$  then 0 else eSuc( $\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } (\text{ltake } (\text{enat } n) \ xss) \ i)))$ )
using lfusecat-llength-lfinite[of (ltake (enat n) xss)]
by (metis 1 2 3 4 5 6 lfusecat-LNil llength-LNil llist.collapse(1) ltake-0
        the-enat.simps zero-enat-def)
have 8: (if  $n = 0$  then 0 else eSuc( $\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } (\text{ltake } (\text{enat } n) \ xss) \ i)))$ ) =
        (if  $n = 0$  then 0 else eSuc( $\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i)))$ )
using 7 atLeastAtMost-iff[of - 0 n-1] sum.cong[of {0..n-1}
        {0..n-1}  $\lambda i. \text{epred}(\text{llength } (\text{lnth } (\text{ltake } (\text{enat } n) \ xss) \ i))$ ]

```

$\lambda i. \text{epred}(\text{llength} (\text{lnth } xss \ i))]$
by (*simp add: Suc-ile-eq dual-order.refl lnth-ltake*)
have 9: $\text{llength} (\text{lfusecat} (\text{ltake} (\text{enat } n) \ xss)) \leq \text{llength} (\text{lfusecat } xss)$
by (*metis lfuse-llength-le-a lfusecat-split*)
have 10: $\text{llength} (\text{ltake} (\text{if } n = 0 \text{ then } 0 \text{ else } \text{eSuc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength} (\text{lnth } xss \ i))))$
 $(\text{lfusecat } xss)) =$
 $(\text{if } n = 0 \text{ then } 0 \text{ else } \text{eSuc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength} (\text{lnth } xss \ i))))$
using 71 8 9 **by** *auto*
have 11: $\text{ltake} (\text{if } n = 0 \text{ then } 0 \text{ else } \text{eSuc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength} (\text{lnth } xss \ i))))$
 $(\text{lfuse} (\text{lfusecat} (\text{ltake } n \ xss)) (\text{lfusecat} (\text{ldrop } n \ xss))) =$
 $(\text{lfusecat} (\text{ltake } n \ xss))$
using 71 8 *ltake-lfuse[of (lfusecat (ltake (enat n) xss)) (lfusecat (ldrop n xss))]*
by *presburger*
have 12: $(\text{lfuse} (\text{lfusecat} (\text{ltake } n \ xss)) (\text{lfusecat} (\text{ldrop } n \ xss))) = \text{lfusecat } xss$
by (*metis lfusecat-split*)
show ?thesis
using 11 12 **by** *auto*
qed

lemma *lfusecat-ldrop*:

assumes $\neg \text{lnull } xss$

$n < \text{llength } xss$

$\forall xs \in \text{lset } xss. \neg \text{lnull } xs$

$\forall xs \in \text{lset } xss. \text{lfinite } xs$

$\text{llastlfirst } xss$

shows $\text{lfusecat} (\text{ldrop} (\text{enat } n) \ xss) =$

$\text{ldrop} (\text{if } n = 0 \text{ then } 0 \text{ else } (\sum i = 0 .. (n-1) . \text{epred}(\text{llength} (\text{lnth } xss \ i))))$
 $(\text{lfusecat } xss)$

proof –

have 1: $\text{lfinite} (\text{ltake} (\text{enat } n) \ xss)$

using *enat-ord-code(4) lfinite-ltake* **by** *blast*

have 2: $\text{llength} (\text{ltake} (\text{enat } n) \ xss) = n$

by (*simp add: assms(2)*)

have 3: $(\text{the-enat}(\text{epred}(\text{llength} (\text{ltake} (\text{enat } n) \ xss)))) = n-1$

using 2 *epred-enat the-enat.simps* **by** *presburger*

have 4: $\forall xs \in \text{lset} (\text{ltake} (\text{enat } n) \ xss). \neg \text{lnull } xs$

by (*meson assms(3) lset-ltake subsetD*)

have 5: $\forall xs \in \text{lset} (\text{ltake} (\text{enat } n) \ xss). \text{lfinite } xs$

by (*meson assms(4) lset-ltake subsetD*)

have 6: $\text{llastlfirst} (\text{ltake} (\text{enat } n) \ xss)$

by (*simp add: assms(2) assms(5) llastlfirst-ltake order.strict-implies-order*)

have 7: $\bigwedge j. 0 < n \wedge j < n-1 \longrightarrow$

$\text{epred}(\text{llength} (\text{lnth} (\text{ltake} (\text{enat } n) \ xss) \ j)) =$

$\text{epred}(\text{llength} (\text{lnth } xss \ j))$

by (*metis diff-le-self dual-order.strict-trans1 enat-ord-simps(2) lnth-ltake*)

have 71: $\text{llength} (\text{lfusecat} (\text{ltake} (\text{enat } n) \ xss)) =$

$(\text{if } n = 0 \text{ then } 0 \text{ else } \text{eSuc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength} (\text{lnth} (\text{ltake} (\text{enat } n) \ xss) \ i))))$

using *lfusecat-llength-lfinite[of (ltake (enat n) xss)]*

by (*metis 1 2 3 4 5 6 lfusecat-LNil llength-LNil llist.collapse(1) ltake-0*

the-enat.simps zero-enat-def)

have 8: (if $n=0$ then 0 else $eSuc(\sum i = 0 .. (n-1) . epred(llength (lnth (ltake (enat n) xss) i))) =$
 (if $n=0$ then 0 else $eSuc(\sum i = 0 .. (n-1) . epred(llength (lnth xss i)))$)
using 7 *atLeastAtMost-iff*[of - 0 $n-1$] *sum.cong*[of {0.. $n-1$ } {0.. $n-1$ }
 $\lambda i. epred(llength (lnth (ltake (enat n) xss) i))$
 $\lambda i. epred(llength (lnth xss i))$]
by (*simp add: Suc-ile-eq dual-order.refl lnth-ltake*)
have 9: $llength (lfusecat (ltake (enat n) xss)) \leq llength (lfusecat xss)$
by (*metis lfuse-llength-le-a lfusecat-split*)
have 10: $llength (ltake (if n=0 then 0 else $eSuc(\sum i = 0 .. (n-1) . epred(llength (lnth xss i)))$))$
 $(lfusecat xss)) =$
 $(if n=0 then 0 else $eSuc(\sum i = 0 .. (n-1) . epred(llength (lnth xss i)))$)$
using 71 8 9 by auto
have 11: $ltake (if n=0 then 0 else $eSuc(\sum i = 0 .. (n-1) . epred(llength (lnth xss i)))$)$
 $(lfuse (lfusecat (ltake n xss)) (lfusecat (ldrop n xss))) =$
 $(lfusecat (ltake n xss))$
using 71 8 ltake-lfuse[of $(lfusecat (ltake (enat n) xss)) (lfusecat (ldrop n xss))]$
by presburger
have 12: $ldrop 0 (lfuse (lfusecat (ltake 0 xss)) (lfusecat (ldrop 0 xss))) =$
 $(lfusecat (ldrop 0 xss))$
by simp
have 13: $0 < n \implies \neg lnull (lfusecat (ltake (enat (n::nat)) (xss::'a llist llist)))$
using 71 8 by force
have 14: $0 < n \implies \neg lnull (lfusecat (ldrop (enat n) xss))$
by (*meson assms(2) assms(3) in-lset-ldropD leD lfusecat-not-llnull-var lnull-ldrop*)
have 15: $0 < n \implies llast (lfusecat (ltake (enat n) xss)) = lfirst (lfusecat (ldrop (enat n) xss))$
proof –
assume a: $0 < n$
have 151: $llast (lfusecat (ltake (enat n) xss)) = llast (llast (ltake (enat n) xss))$
using 1 13 4 5 6 a lfusecat-not-llnull llastfirst-lfusecat-llast by blast
have 152: $lfirst (lfusecat (ldrop (enat n) xss)) = lfirst (lfirst (ldrop (enat n) xss))$
by (*metis assms(2) assms(3) in-lset-conv-lnth ldrop-enat leD lfirst-def lfirst-lfusecat-lfirst*
 $lhd-ldropn lnull-ldrop$)
have 153: $llast (llast (ltake (enat n) xss)) = lfirst (lfirst (ldrop (enat n) xss))$
using assms a
by (*metis 1 13 lappend-ltake-ldrop leD lfusecat-not-llnull llastlfirst-lappend-lfinite*
 $lnull-ldrop$)
show ?thesis
by (*simp add: 151 152 153*)
qed
have 16: $0 < n \implies ldrop (epred (eSuc(\sum i = 0 .. (n-1) . epred(llength (lnth xss i))))$
 $(lfuse (lfusecat (ltake n xss)) (lfusecat (ldrop n xss))) =$
 $(lfusecat (ldrop n xss))$
using 71 8 13 14 15 ldrop-lfuse-a[of $(lfusecat (ltake (enat n) xss)) (lfusecat (ldrop n xss))]$
by (*metis 1 5 less-numeral-extra(3) lfusecat-lfinite-a*)
have 17: $ldrop (if n=0 then 0 else $(\sum i = 0 .. (n-1) . epred(llength (lnth xss i)))$)$
 $(lfuse (lfusecat (ltake n xss)) (lfusecat (ldrop n xss))) =$
 $(lfusecat (ldrop n xss))$
using 11 16 by auto
show ?thesis
by (*metis 17 lfusecat-split*)

qed

lemma *lfusecat-eq-LCons-conv*:

shows $lfusecat\ xss = LCons\ x\ xs \longleftrightarrow$

$(\exists\ xs'\ xss'\ xss''.\ xss = lappend\ (l\text{list-of}\ xss')\ (LCons\ (LCons\ x\ xs')\ xss'') \wedge$
 $xs = lappend\ xs'\ (l\text{tl}\ (lfusecat\ xss'')) \wedge \text{set}\ xss' \subseteq \{xs.\ \text{lnull}\ xs\})$
(is $?lhs \longleftrightarrow ?rhs$ **)**

proof

assume $?rhs$

then obtain $xs'\ xss'\ xss''$

where $xss = lappend\ (l\text{list-of}\ xss')\ (LCons\ (LCons\ x\ xs')\ xss'')$

and $xs = (lappend\ xs'\ (l\text{tl}\ (lfusecat\ xss'')))$

and $\text{set}\ xss' \subseteq \{xs.\ \text{lnull}\ xs\}$ **by** *blast*

moreover from $\langle \text{set}\ xss' \subseteq \{xs.\ \text{lnull}\ xs\} \rangle$

have $\text{lnull}\ (lfusecat\ (l\text{list-of}\ xss'))$

by (*metis lfusecat-not-lnull lset-l\text{list-of mem-Collect-eq subset-eq*)

have 1: $lfusecat\ xss = lfuse\ (lfusecat\ (l\text{list-of}\ xss'))\ (lfusecat\ (LCons\ (LCons\ x\ xs')\ xss''))$

by (*simp add: calculation(1) lfusecat-lappend*)

have 2: $lfuse\ (lfusecat\ (l\text{list-of}\ xss'))\ (lfusecat\ (LCons\ (LCons\ x\ xs')\ xss'')) =$
 $(lfusecat\ (LCons\ (LCons\ x\ xs')\ xss''))$

by (*simp add: <lnull (lfusecat (l\text{list-of} (xss'::'a l\text{list list}))> lfuse-conv-lnull l\text{list.expand}*)

have 3: $(lfusecat\ (LCons\ (LCons\ x\ xs')\ xss'')) = lfuse\ (LCons\ x\ xs')\ (lfusecat\ xss'')$

by *simp*

have 4: $lfuse\ (LCons\ x\ xs')\ (lfusecat\ xss'') =$

$(LCons\ x\ (lappend\ xs'\ (l\text{tl}\ (lfusecat\ xss''))))$

unfolding *lfuse-def*

using *l\text{list.collapse}(1)* **by** *force*

ultimately show $?lhs$

using 1 2 **by** *auto*

next

assume $?lhs$

hence $\neg \text{lnull}\ (lfusecat\ xss)$ **by** *simp*

hence $\neg \text{lset}\ xss \subseteq \{xs.\ \text{lnull}\ xs\}$ **using** *lfusecat-eq-LNil ln\text{ull-def}* **by** *blast*

hence $\neg \text{lnull}\ (l\text{filter}\ (\lambda xs.\ \neg \text{lnull}\ xs)\ xss)$ **by** (*auto*)

then obtain $y\ ys$ **where** $yss: l\text{filter}\ (\lambda xs.\ \neg \text{lnull}\ xs)\ xss = LCons\ y\ ys$

unfolding *not-lnull-conv* **by** *auto*

from *l\text{filter-eq-LConsD}[OF this]*

obtain $us\ vs$ **where** $xss: xss = lappend\ us\ (LCons\ y\ vs)$

and *l\text{finite}\ us*

and $\text{lset}\ us \subseteq \{xs.\ \text{lnull}\ xs\} \neg \text{lnull}\ y$

and $ys: ys = l\text{filter}\ (\lambda xs.\ \neg \text{lnull}\ xs)\ vs$ **by** *blast*

from $\langle \text{l\text{finite}\ us} \rangle$ **obtain** us' **where** [*simp*]: $us = l\text{list-of}\ us'$

unfolding *l\text{finite-eq-range-l\text{list-of}}* **by** *blast*

from $\langle \text{lset}\ us \subseteq \{xs.\ \text{lnull}\ xs\} \rangle$ **have** $us: \text{lnull}\ (lfusecat\ us)$

using *lfusecat-eq-LNil ln\text{ull-def}* **by** *blast*

from $\langle \neg \text{lnull}\ y \rangle$ **obtain** $y'\ ys'$ **where** $y: y = LCons\ y'\ ys'$

unfolding *not-lnull-conv* **by** *blast*

from $\langle ?lhs \rangle$ us **have** [*simp*]: $y' = x$

```

    xs = (lappend ys' (ltl (lfusecat vs)))
unfolding xss y
by (metis <¬ lnull (y::'a llist)> <lfinit (us::'a llist llist)> eq-LConsD lfusecat-LCons
    lfusecat-lappend lhd-lfuse y)
    (metis <¬ lnull (y::'a llist)> <lfusecat (xss::'a llist llist) = LCons (x::'a) (xss::'a llist)>
    eq-LConsD lappend-nullable lfusecat-LCons lfusecat-lfilter-neq-LNil ltl-lfuse y ys yss)
from <lset us ⊆ {xs. lnull xs}> ys show ?rhs unfolding xss y by simp blast
qed

```

lemma not-nullable-expand:

```

    ¬ lnull xs ⟷ (∃ b. xs = (LCons b LNil)) ∨ (∃ b xs'. xs = (LCons b xs') ∧ ¬ lnull xs')
by (metis lhd-LCons-ltl llist.disc(1) llist.disc(2) llist.expand)

```

lemma is-LMore-llength:

```

    (∃ b xs'. xs = (LCons b xs') ∧ ¬ lnull xs') ⟷ 1 < llength xs
by (metis dual-order.strict-implies-not-eq gr-implies-not-zero lhd-LCons-ltl llength-LCons
    llength-eq-0 lstrict-prefix-code(2) lstrict-prefix-code(4) lstrict-prefix-llength-less one-eSuc)

```

lemma is-LEmpty-llength:

```

    (∃ b. xs = (LCons b LNil)) ⟷ llength xs = 1
by (metis epred-llength llength-LCons llength-eq-0 llist.disc(1) ltl-simps(2) not-nullable-expand
    one-eSuc)

```

lemma lfusecat-all-llength-one-lfuse:

```

assumes ∀ ys ∈ lset (lset-of xss'). llength ys = 1
shows lfuse (lfusecat (lset-of xss')) ys =
    lfuse (if ¬ lnull (lset-of xss') then (LCons (lfirst (lfirst (lset-of xss'))) LNil) else LNil) ys

```

proof –

```

have 1: (lfusecat (lset-of xss')) =
    (if ¬ lnull (lset-of xss') then (LCons (lfirst (lfirst (lset-of xss'))) LNil) else LNil)
using assms
proof (induction xss')
case Nil
then show ?case by simp
next
case (Cons as xss')
then show ?case
    proof (cases xss'=Nil)
    case True
    then show ?thesis using Cons
    by simp (metis is-LEmpty-llength lfirst-def lhd-LCons)
    next
    case False
    then show ?thesis using Cons
    by simp
    (metis is-LEmpty-llength lfirst-def lhd-LCons llength-nullable zero-one-enat-neq(1))

```


qed
 qed
 show ?thesis
 using 1 by presburger
 qed

lemma *lfusecat-eq-LCons-conv-all-lset-singleton:*

assumes $\forall \text{ys} \in \text{lset} (\text{llist-of } xss). \text{length ys} = 1$
 $\neg \text{null} (\text{llist-of } xss)$
 $\text{llastlfirst} (\text{llist-of } xss)$
shows $\text{lset} (\text{llist-of } xss) = \{\text{lfirst} (\text{llist-of } xss)\}$
using *assms*
proof (*induction xss*)
case *Nil*
then show ?case **by** *simp*
next
case (*Cons as xss*)
then show ?case
proof –
have 1: $xss = [] \vee as \in \text{lset} (\text{llist-of } xss)$
by (*metis Cons.prem1 Cons.prem3 List.set-insert insert-iff is-LEmpty-length*
le-numeral-extra(4) lfirst-def lhd-LCons-ltl llast-singleton llastlfirst-LCons llist.set-sel(1)
llist-of.simps(2) lnull-llist-of lset-llist-of ltl-simps(2) not-in-set-insert not-llength)
have 2: $xss = [] \implies ?thesis$
by (*simp add: lfirst-def*)
have 3: $as \in \text{lset} (\text{llist-of } xss) \implies ?thesis$
by (*metis 2 Cons.IH Cons.prem1 Cons.prem3 insert-absorb2 insert-iff lfirst-def lhd-LCons*
llastlfirst-LCons llist.simps(19) llist-of.simps(2) lnull-llist-of singletonD)
show ?thesis
using 1 2 3 **by** *linarith*
 qed
 qed

lemma *lfusecat-eq-LCons-conv-all-the-same:*

assumes $\forall \text{ys} \in \text{lset} (\text{llist-of } xss'). \text{length ys} = 1$
 $\neg \text{null} (\text{llist-of } xss')$
 $\text{llastlfirst} (\text{llist-of } xss')$
 $(\text{llast} (\text{llist-of } xss')) = (\text{LCons } x \text{ LNil})$
 $\text{ys} \in \text{lset} (\text{llist-of } xss')$
shows $\text{ys} = (\text{LCons } x \text{ LNil})$
proof –
have 1: $\text{lset} (\text{llist-of } xss') = \{\text{lfirst} (\text{llist-of } xss')\}$
using *assms lfusecat-eq-LCons-conv-all-lset-singleton* **by** *blast*
have 2: $\text{llast} (\text{llist-of } xss') = \text{lnth} (\text{llist-of } xss') (\text{the-enat}(\text{epred}(\text{length} (\text{llist-of } xss'))))$
by (*metis assms(2) co.enat.exhaust-sel enat-the-enat ile-eSuc infinity-ileE llast-conv-lnth*
length-eq-0 length-llist-of)
have 3: $\text{lnth} (\text{llist-of } xss') (\text{the-enat}(\text{epred}(\text{length} (\text{llist-of } xss')))) \in \text{lset} (\text{llist-of } xss')$
by (*metis 2 assms(2) co.enat.exhaust-sel ile-eSuc in-lset-lappend-iff infinity-ileE*
lappend-lbutlast-llast-id length-eq-0 length-eq-inf-conv-lfinite length-lbutlast)

```

    llength-llist-of lset-intros(1))
have 4: (LCons x LNil) ∈ lset (llist-of xss')
  using 2 3 assms(4) by force
show ?thesis
  using 1 4 assms(5) by auto
qed

```

lemma *lfusecat-eq-LCons-conv-all-the-same-a:*

```

assumes lfinite uss
  ∀ ys ∈ lset (uss). llength ys = 1
  ¬ lnull (uss)
  llastlfirst (uss)
  (llast (uss)) = (LCons x LNil)
  ys ∈ lset (uss)
shows   ys = (LCons x LNil)
using assms lfusecat-eq-LCons-conv-all-the-same llist-of-list-of
by (metis (full-types))

```

lemma *lfusecat-eq-LCons-conv-alt:*

```

assumes llastlfirst xss
  ∀ ys ∈ lset xss. ¬ lnull ys
  ¬ lnull xs
shows   lfusecat xss = LCons x xs ⟷
  (∃ xs' xss' xss''. xss = lappend (llist-of xss') (LCons (LCons x xs') xss'') ∧ ¬lnull xs' ∧
    xs = lfuse xs' (lfusecat xss'') ∧ set xss' ⊆ {xs. lnull (ltl xs)})
  (is ?lhs ⟷ ?rhs)

```

proof

```

  assume ?rhs
  then obtain xs' xss' xss''
    where xss = lappend (llist-of xss') (LCons (LCons x xs') xss'')
    and ¬lnull xs'
    and xs = (lfuse xs' (lfusecat xss''))
    and set xss' ⊆ {xs. lnull (ltl xs)} by blast
  moreover from ⟨set xss' ⊆ {xs. lnull (ltl xs)}⟩
  have 00: llength (lfusecat (llist-of xss')) ≤ 1
    by (metis is-LMore-llength lfusecat-all-empty-or-LNil-a lset-llist-of lset-lltl-llength-var
      ltl-simps(2) mem-Collect-eq not-le-imp-less subset-eq)
  have 01: ∀ y ∈ lset (llist-of xss'). llength y = 1
    by (metis assms(2) calculation(1) calculation(4) dual-order.strict-iff-order is-LMore-llength
      lset-lappend1 lset-llist-of ltl-simps(2) mem-Collect-eq not-llength subsetD)
  have 02: lfinite (llist-of xss')
    using lfinite-llist-of by blast
  have 03: llastlfirst (lappend (llist-of xss') (LCons (LCons x xs') xss''))
    using assms calculation(1) by blast
  have 04: llastlfirst (llist-of xss')
    using assms(1) calculation(1) lfinite-llist-of llastlfirst-lappend-lfinite by blast
  have 05: (if lnull (llist-of xss') ∨ lnull (LCons (LCons x xs') xss''))
    then True

```

```

      else llast (llast (llist-of xss')) = lfirst (lfirst (LCons (LCons x xs') xss'))
    using 03 02 llastlfirst-lappend-lfinite[of (llist-of xss') (LCons (LCons x xs') xss')] ]
  by blast
have 06:  $\neg \text{lnull } (LCons (LCons x xs') xss'')$ 
  by simp
have 07:  $\neg \text{lnull } (l\text{list-of } xss') \implies$ 
       $\text{llast } (llast (l\text{list-of } xss')) = \text{lfirst } (l\text{first } (LCons (LCons x xs') xss''))$ 
  using 05 06 by presburger
have 08:  $\neg \text{lnull } (l\text{list-of } xss') \implies (\text{llast } (l\text{list-of } xss')) = (LCons x LNil)$ 
  using 07 lappend-lbutlast-llast-id[of (llist-of xss')] ]
  by (metis 01 02 eq-LConsD in-lset-lappend-iff is-LEmpty-llength lbutlast-lfinite lfirst-def
      llast-singleton lset-intros(1))
have 09:  $\neg \text{lnull } (l\text{list-of } xss') \implies (\forall \text{ys} \in \text{lset } (l\text{list-of } xss'). \text{ys} = (LCons x LNil))$ 
  using lfusecat-eq-LCons-conv-all-the-same
  by (metis 01 04 08)
have 0:  $(\text{lfusecat } (l\text{list-of } xss')) = LNil \vee (\text{lfusecat } (l\text{list-of } xss')) = (LCons x LNil)$ 
  by (metis 00 09 eq-LConsD is-LMore-llength lfirst-def lfirst-lfusecat lfusecat-LNil lhd-LCons-ltl
      linorder-not-le llist.collapse(1) llist.set-sel(1))
have 1:  $\text{lfusecat } xss = \text{lfuse } (\text{lfusecat } (l\text{list-of } xss')) (\text{lfusecat } (LCons (LCons x xs') xss'))$ 
  by (simp add: calculation(1) lfusecat-lappend)
have 2:  $\text{lfuse } (\text{lfusecat } (l\text{list-of } xss')) (\text{lfusecat } (LCons (LCons x xs') xss')) =$ 
       $(\text{lfusecat } (LCons (LCons x xs') xss'))$ 
  using 0 by fastforce
have 3:  $(\text{lfusecat } (LCons (LCons x xs') xss')) = \text{lfuse } (LCons x xs') (\text{lfusecat } xss'')$ 
  by simp
have 4:  $\text{lfuse } (LCons x xs') (\text{lfusecat } xss'') =$ 
       $(LCons x ( \text{lfuse } xs' ( \text{lfusecat } xss'')) )$ 
  by (
    unfolding lfuse-def
    using calculation(2) by auto
  )
ultimately show ?lhs
using 1 2 by auto
next
assume ?lhs
hence  $\neg \text{lnull } (\text{lfusecat } xss)$  by simp
hence  $\neg \text{lset } xss \subseteq \{xs. \text{lnull } (l\text{tl } xs)\}$ 
by (metis  $\langle \text{lfusecat } (xss::'a \text{ llist } llist) = LCons (x::'a) (xss::'a \text{ llist}) \rangle$  assms(3)
    lfusecat-all-empty-or-LNil-a lnull-ldrop lset-ltl-llength-var ltl-ldrop-one ltl-simps(2)
    mem-Collect-eq subsetD)
hence  $\neg \text{lnull } (\text{filter } (\lambda xs. \neg \text{lnull } (l\text{tl } xs)) xss)$  by (auto)
then obtain y ys where yss:  $\text{filter } (\lambda xs. \neg \text{lnull } (l\text{tl } xs)) xss = LCons y ys$ 
  unfolding not-nullable-conv by auto
from lfilt-eq-LConsD[OF this]
obtain us vs where xss:  $xss = \text{lappend } us (LCons y vs)$ 
  and lfinite us
  and  $\text{lset } us \subseteq \{xs. \text{lnull } (l\text{tl } xs)\} \neg \text{lnull } (l\text{tl } y)$ 
  and ys:  $ys = \text{filter } (\lambda xs. \neg \text{lnull } (l\text{tl } xs)) vs$  using lnull-ltlI by blast
from  $\langle \text{lfinite } us \rangle$  obtain us' where [simp]:  $us = \text{llist-of } us'$ 
  unfolding lfinite-eq-range-llist-of by blast
from  $\langle \text{lset } us \subseteq \{xs. \text{lnull } (l\text{tl } xs)\} \rangle$  have us:  $\text{llength } (\text{lfusecat } us) \leq 1$ 

```

```

using lfusecat-eq-LNil
by (metis lfusecat-all-empty-or-LNil-a lnull-ldrop lset-lltl-llength-var ltl-ldrop-one
    mem-Collect-eq subset-eq)
from  $\langle \neg \text{lnull } (\text{ltl } y) \rangle$  obtain  $y' \text{ } ys'$  where  $y: y = LCons \text{ } y' \text{ } ys'$ 
  unfolding not-lnull-conv
  by (metis  $\langle \neg \text{lnull } (\text{ltl } y) \rangle$  lhd-LCons-ltl lnull-ltlI)
have 100: llastlfirst us
  using  $\langle \text{lfinite } us \rangle$  assms(1) llastlfirst-lappend-lfinite xss by blast
have 101:  $\neg \text{lnull } ys'$ 
  using  $\langle \neg \text{lnull } (\text{ltl } y) \rangle$  y by auto
have 102:  $\neg \text{lnull } (LCons \text{ } y \text{ } vs)$ 
  by simp
have 103:  $\neg \text{lnull } us \implies \text{llast } (\text{llast } us) = \text{lfirst } (\text{lfirst } (LCons \text{ } y \text{ } vs))$ 
  using llastlfirst-lappend-lfinite[of us (LCons y vs)]
  using assms(1) xss by auto
have 104:  $(\forall z \in \text{lset } us. \text{llength } z = 1)$ 
  by (metis  $\langle \text{lset } us \subseteq \{xs. \text{lnull } (\text{ltl } xs)\} \rangle$  assms(2)
    dual-order.strict-iff-order eq-LConsD in-lset-lappend-iff is-LMore-llength mem-Collect-eq
    not-lnull-llength subsetD xss)
have 1040:  $\neg \text{lnull } us \implies \text{llast } us \in \text{lset } us$ 
  by simp
have 1041:  $\neg \text{lnull } us \implies \text{llast } us = LCons \text{ } x \text{ } LNil$ 
  using lappend-lbutlast-llast-id[of us] 100 104 1040
  lfusecat-eq-LCons-conv-all-the-same-a[of us x llast us]
  lfusecat-eq-LCons-conv-all-lset-singleton[of us' ]
  by (metis  $\langle \neg \text{lnull } (\text{lfusecat } xss) \rangle$   $\langle \text{lfusecat } xss = LCons \text{ } x \text{ } xs \rangle$   $\langle us = \text{lset-of } us' \rangle$ 
    eq-LConsD is-LEmpty-llength lfirst-def lfirst-lfusecat-lfirst lfusecat-not-lnull
    lhd-lappend lset-eq-empty not-lnull-lset-conv-a singletonD xss)
have 105:  $\neg \text{lnull } us \implies (\forall z \in \text{lset } us. z = (LCons \text{ } x \text{ } LNil))$ 
  using lfusecat-eq-LCons-conv-all-the-same-a[of us x ]
  using 100 104 1041  $\langle \text{lfinite } us \rangle$  by blast
have 106:  $\neg \text{lnull } us \implies \text{lfusecat } us = (LCons \text{ } x \text{ } LNil)$ 
  by (metis 100 104 1041  $\langle \text{lfinite } us \rangle$  dual-order.antisym is-LEmpty-llength lfinite-LConsI
    lfinite-LNil lfusecat-not-lnull-var llast-singleton llastfirst-lfusecat-llast llength-LNil
    lset-eq-empty not-lnull-llength not-lnull-lset-conv-a us zero-one-enat-neq(1))
from  $\langle ?lhs \rangle$  us have [simp]:  $y' = x$ 
  by (metis 103 1041  $\langle \neg \text{lnull } (\text{ltl } y) \rangle$  eq-LConsD lappend-lnull1 lfirst-def lfirst-lfusecat
    llast-singleton lnull-ltlI xss y)
from  $\langle ?lhs \rangle$  us have [simp]:  $xs = ( \text{lfuse } ys' ( \text{lfusecat } vs) )$ 
  using lfusecat-lappend[of us (LCons y vs) ] lfusecat-LCons[of y vs] y xss
  101  $\langle \text{lfinite } us \rangle$   $\langle \text{lfusecat } xss = LCons \text{ } x \text{ } xs \rangle$ 
  by (metis 103 1041 106 eq-LConsD lappend-code(1) lappend-lnull1 lbutlast-simps(2) lfirst-def
    lfuse-LCons-a lfuse-conv-lnull lfuse-lbutlast lnull-ltlI)
from  $\langle \text{lset } us \subseteq \{xs. \text{lnull } (\text{ltl } xs)\} \rangle$  ys show ?rhs unfolding xss y
  using 101 by auto
qed

```

lemma *lfusecat-eq-One-conv*:
 $\text{lfusecat } xss = LCons \text{ } x \text{ } LNil \longleftrightarrow$

$(\exists xss' xss''. xss = \text{lappend } (\text{llist-of } xss') (LCons (LCons x LNil) xss'') \wedge$
 $LNil = (\text{ltl } (\text{lfusecat } xss'')) \wedge \text{set } xss' \subseteq \{xs. \text{lnull } xs\})$
by (*metis lappend-eq-LNil-iff lfusecat-eq-LCons-conv*)

lemma *llength-lfusecat-eq-one-conv*:

$\text{llength } (\text{lfusecat } xss) = 1 \iff$
 $(\exists x xss' xss''. xss = \text{lappend } (\text{llist-of } xss') (LCons (LCons x LNil) xss'') \wedge$
 $LNil = (\text{ltl } (\text{lfusecat } xss'')) \wedge \text{set } xss' \subseteq \{xs. \text{lnull } xs\})$
by (*metis is-LEmpty-llength lfusecat-eq-One-conv*)

lemma *lfusecat-lfinite-b*:

assumes *lfinite*(*lfusecat xss*)
 $\forall xs \in \text{lset } xss. \neg \text{lnull } (\text{ltl } xs)$
 $\forall xs \in \text{lset } xss. \text{lfinite } xs$
 $\text{llastlfirst } xss$
shows *lfinite* *xss*
using *assms*
proof (*induct zs* \equiv (*lfusecat xss*) *arbitrary: xss*)
case *lfinite-LNil*
then show ?*case*
by (*metis lfusecat-not-lnull-var llist.disc(1) lnull-imp-lfinite lset-eq-empty ltl-simps(1)*
not-lnull-lset-conv-a)
next
case (*lfinite-LConsI xs x*)
then show ?*case*
proof –
have 1: $(\exists xs' xss' xss''. xss = \text{lappend } (\text{llist-of } xss') (LCons (LCons x xs') xss'') \wedge$
 $xs = \text{lappend } xs' (\text{ltl } (\text{lfusecat } xss'')) \wedge \text{set } xss' \subseteq \{xs. \text{lnull } xs\})$
using *lfinite-LConsI.hyps(3) lfusecat-eq-LCons-conv* **by** *fastforce*
obtain $xs' xss' xss''$ **where** 2: $xss = \text{lappend } (\text{llist-of } xss') (LCons (LCons x xs') xss'') \wedge$
 $xs = \text{lappend } xs' (\text{ltl } (\text{lfusecat } xss'')) \wedge \text{set } xss' \subseteq \{xs. \text{lnull } xs\}$
using 1 **by** *blast*
have 3: $(\text{llist-of } xss') = LNil$
by (*metis 2 in-lset-lappend-iff lconcat-eq-LNil lfinite-LConsI.prem(1) llist.collapse(1)*
llist.set-sel(1) lnull-lconcat lset-eq-forall-lnull lset-llist-of ltl-simps(1))
have 4: $xss = (LCons (LCons x xs') xss'')$
by (*simp add: 2 3*)
have 5: $\text{llastlfirst } xss$
by (*simp add: lfinite-LConsI.prem(3)*)
have 6: $\neg \text{lnull } xs'$
using 4 *lfinite-LConsI.prem(1)* **by** *force*
have 7: $\text{lnull } xss'' \implies ?thesis$
by (*simp add: 4*)
have 8: $\neg \text{lnull } xss'' \implies ?thesis$
proof –
assume *a*: $\neg \text{lnull } xss''$
have 9: $\text{llast } (LCons x xs') = \text{lfirst } (\text{lfirst } (xss''))$
using *a 6 4 5 llastlfirst-LCons* **by** *blast*
have 10: $\forall xs \in \text{lset } xss''. \neg \text{lnull } (\text{ltl } xs)$

```

    by (simp add: 4 lfinite-LConsI.premis)
  have 11:  $\forall xs \in lset\ xss''.\ lfinite\ xs$ 
    by (simp add: 4 lfinite-LConsI.premis)
  have 12:  $llastlfirst\ xss''$ 
    using 4 5 a by auto
  have 13:  $lfinite(lfusecat\ xss'')$ 
    using 2 lfinite-LConsI.hyps(1) by auto
  have 14:  $xs = lappend\ (lbutlast\ xs')\ (lfusecat\ xss'')$ 
    by (metis 10 2 6 9 a lappend-lbutlast-llast-id lappend-snocL1-conv-LCons2 lfirst-def
        lfirst-lfusecat-lfirst lfusecat-not-lnull-var lhd-LCons-ltl llast-LCons llist.disc(1)
        llist.set-sel(1) lset-eq-empty ltl-simps(1) not-lnull-lset-conv-a)
  have 15:  $xs = lfuse\ xs'\ (lfusecat\ xss'')$ 
    by (simp add: 2 6 lappend-lnull2 lfuse-def)
  have 16:  $xs = lfusecat\ (LCons\ xs'\ xss'')$ 
    by (simp add: 15)
  have 17:  $lnull\ (ltl\ xs') \implies ?thesis$ 
    by (metis 10 11 12 14 4 lappend-code(1) lbutlast.ctr(1) lfinite.lfinite-LConsI
        lfinite-LConsI.hyps(2) llist.collapse(1))
  have 18:  $\neg lnull\ (ltl\ xs') \implies (\forall xs \in lset\ (LCons\ xs'\ xss'').\ \neg lnull\ (ltl\ xs))$ 
    by (simp add: 10)
  have 19:  $\forall xs \in lset\ (LCons\ xs'\ xss'').\ lfinite\ xs$ 
    by (metis 11 2 insert-iff lappend-inf lfinite-LConsI.hyps(1) llist.simps(19))
  have 20:  $llastlfirst\ (LCons\ xs'\ xss'')$ 
    by (metis 12 6 9 a llast-LCons llastlfirst-LCons)
  have 21:  $lfinite\ (LCons\ xs'\ xss'')$ 
    by (metis 16 17 18 19 20 4 lfinite-LConsI.hyps(2) lfinite-code(2) )
  show ?thesis
    using 21 4 by auto
qed
show ?thesis
using 7 8 by blast
qed
qed

```

lemma *lfusecat-lfinite:*

assumes $\forall xs \in lset\ xss.\ \neg lnull\ (ltl\ xs)$

$\forall xs \in lset\ xss.\ lfinite\ xs$

$llastlfirst\ xss$

shows $lfinite\ (lfusecat\ xss) \longleftrightarrow lfinite\ xss$

using *assms lfusecat-lfinite-a lfusecat-lfinite-b* **by** *blast*

lemma *ridx-lfusecat-ltake:*

assumes $ridx\ R\ (lfusecat\ xss)$

$n \leq llength\ xss$

shows $ridx\ R\ (lfusecat\ (ltake\ n\ xss))$

using *assms*

by (metis *lfusecat-split ltake-all ltake-lfuse nle-le ridx-ltake-a*)

lemma *ridx-lfusecat-ldrop:*

assumes $\text{ridx } R \text{ (lfusecat } xss)$
 $(\text{enat } n) < \text{llength } xss$
 $\text{llastlfirst } xss$
 $\forall xs \in \text{lset } xss. \neg \text{lnull } xs$
 $\forall xs \in \text{lset } xss. \text{lfinite } xs$
shows $\text{ridx } R \text{ (lfusecat (ldrop } n \text{ } xss))$
proof –
have 1: $(\text{lfusecat (ldrop } n \text{ } xss)) =$
 $\text{ldrop (if } n = 0 \text{ then } 0 \text{ else } (\sum i = 0 .. (n-1) . \text{epred}(\text{llength (lnth } xss \text{ } i))))$
 $(\text{lfusecat } xss)$
using $\text{assms gr-implies-not-zero lfusecat-ldrop llength-eq-0 by blast}$
have 2: $(\text{if } n = 0 \text{ then } 0 \text{ else } (\sum i = 0 .. (n-1) . \text{epred}(\text{llength (lnth } xss \text{ } i)))) < \text{llength (lfusecat } xss)$
by $(\text{metis (no-types, lifting) 1 add.commute add.right-neutral assms(2) assms(4) in-lset-conv-lnth}$
 $\text{lfusecat-not-lnull llist.set-sel(1) lnth-0-conv-lhd lnth-ldrop lnull-ldrop not-less-iff-gr-or-eq}$
 $\text{order.order-iff-strict the-enat.simps zero-enat-def})$
have 3: $\exists k. (\text{enat } k) = (\text{if } n = 0 \text{ then } 0 \text{ else } (\sum i = 0 .. (n-1) . \text{epred}(\text{llength (lnth } xss \text{ } i))))$
proof $(\text{cases } n)$
case 0
then show $?thesis$ **by** $(\text{simp add: zero-enat-def})$
next
case $(\text{Suc } nat)$
then show $?thesis$ **by** $(\text{metis (mono-tags, lifting) 2 enat-iless enat-less-imp-le leD})$
qed
have 4: $\text{ridx } R \text{ (ldrop (if } n = 0 \text{ then } 0 \text{ else } (\sum i = 0 .. (n-1) . \text{epred}(\text{llength (lnth } xss \text{ } i))))$
 $(\text{lfusecat } xss))$
using $\text{ridx-ldrop[of } R \text{ (lfusecat } xss)]$
using 2 $\text{assms(1) order.strict-implies-order by blast}$
show $?thesis$ **by** (simp add: 1 4)
qed

lemma ridx-lfusecat-a:

assumes $\text{llastlfirst } xss$
 $\forall xs \in \text{lset } xss. \neg \text{lnull } xs$
 $\forall xs \in \text{lset } xss. \text{lfinite } xs$
 $\text{ridx } R \text{ (lfusecat } xss)$
 $i < \text{llength } xss$
shows $\text{ridx } R \text{ (lnth } xss \text{ } i)$
proof –
have 1: $\text{lsub } i \text{ } xss = (\text{LCons (lnth } xss \text{ } i) \text{ LNil})$
by $(\text{simp add: assms(5) lsub-same})$
have 2: $\text{lsub } i \text{ } xss = \text{ltake (eSuc (enat (i - i))) (ldrop (enat } i) \text{ } xss)$
by $(\text{simp add: lsub-def})$
have 3: $\text{ltake (eSuc (enat (i - i))) (ldrop (enat } i) \text{ } xss) = \text{ltake } 1 \text{ (ldrop (enat } i) \text{ } xss)$
by $(\text{simp add: eSuc-enat one-enat-def})$
have 4: $\text{ridx } R \text{ (lfusecat (ltake } 1 \text{ (ldrop (enat } i) \text{ } xss)))$
by $(\text{metis assms ltake-all nle-le ridx-lfusecat-ldrop ridx-lfusecat-ltake})$
have 5: $(\text{lfusecat (ltake } 1 \text{ (ldrop (enat } i) \text{ } xss))) = (\text{lnth } xss \text{ } i)$
by $(\text{metis 1 2 3 lfuse-LNil-2 lfusecat-LCons lfusecat-LNil})$
show $?thesis$
using 4 5 **by** auto

qed

lemma *ridx-lfuse-lfinite:*

assumes *lfinite l1*

llast l1 = lfirst l2

shows $\text{ridx } R \text{ (lfuse } l1 \text{ } l2) \longleftrightarrow \text{ridx } R \text{ } l1 \wedge \text{ridx } R \text{ } l2$

using *assms*

proof (*cases* $\neg \text{lnull } l1 \wedge \neg \text{lnull } l2$)

case *True*

then show *?thesis*

proof (*cases* $\neg \text{lnull (ltl } l2)$)

case *True*

then show *?thesis*

proof –

have 1: $\text{lfuse } l1 \text{ } l2 = \text{lappend } l1 \text{ (ltl } l2)$

unfolding *lfuse-def* **using** $\langle \neg \text{lnull } l1 \wedge \neg \text{lnull } l2 \rangle$ **by** *simp*

have 2: $\text{ridx } R \text{ (lappend } l1 \text{ (ltl } l2)) = (\text{ridx } R \text{ } l1 \wedge (R \text{ (llast } l1) \text{ (lhd (ltl } l2)))} \wedge \text{ridx } R \text{ (ltl } l2))$

using *assms ridx-lappend-lfinite*[*of l1 R ltl l2*] *True* $\langle \neg \text{lnull } l1 \wedge \neg \text{lnull } l2 \rangle$

by *auto*

have 3: $(R \text{ (llast } l1) \text{ (lhd (ltl } l2))} \wedge \text{ridx } R \text{ (ltl } l2)) = \text{ridx } R \text{ } l2$

using *True* $\langle \neg \text{lnull } l1 \wedge \neg \text{lnull } l2 \rangle$ *ridx-LCons-1*[*of R lfirst l2 ltl l2*]

by (*simp add: assms lfirst-def*)

show *?thesis*

by (*simp add: 1 2 3*)

qed

next

case *False*

then show *?thesis*

proof –

have 4: $\text{lfuse } l1 \text{ } l2 = \text{lappend } l1 \text{ (ltl } l2)$

unfolding *lfuse-def* **using** $\langle \neg \text{lnull } l1 \wedge \neg \text{lnull } l2 \rangle$ **by** *simp*

have 5: $\text{ridx } R \text{ (lappend } l1 \text{ (ltl } l2)) = (\text{ridx } R \text{ } l1 \wedge \text{ridx } R \text{ } l2)$

using *assms ridx-lappend-lfinite*[*of l1 R ltl l2*] *False* $\langle \neg \text{lnull } l1 \wedge \neg \text{lnull } l2 \rangle$

ridx-expand-1 **by** (*metis lhd-LCons-ltl ridx-LCons-1*)

show *?thesis* **by** (*simp add: 4 5*)

qed

qed

next

case *False*

then show *?thesis*

by (*metis lfuse-LNil-1 lfuse-LNil-2 llist.collapse(1) ridx-expand-1*)

qed

lemma *ridx-lfusecat-lfuse:*

assumes *llastlfirst xss*

$\forall xs \in \text{lset } xss. \neg \text{lnull } xs$

$\forall xs \in \text{lset } xss. \text{lfinite } xs$

$\forall i. i < \text{length } xss \longrightarrow \text{ridx } R (\text{lnth } xss \ i)$
 $(\text{Suc } n) < \text{length } xss$
shows $\text{ridx } R (\text{lfuse } (\text{lnth } xss \ n) (\text{lnth } xss (\text{Suc } n)))$
using *assms*
by (*metis Suc-ile-eq lfuse-inf llastlfirst-def order-less-imp-le ridx-lfuse-lfinite*)

lemma *ridx-lfusecat-ltake-a:*
assumes *llastlfirst xss*
 $\forall xs \in \text{lset } xss. \neg \text{lnull } xs$
 $\forall xs \in \text{lset } xss. \text{lfinite } xs$
 $\forall i. i < \text{length } xss \longrightarrow \text{ridx } R (\text{lnth } xss \ i)$
 $n \leq \text{length } xss$
shows $\text{ridx } R (\text{lfusecat } (\text{ltake } (\text{enat } n) \ xss))$
using *assms*
proof (*induct n arbitrary: xss*)
case 0
then show ?case
by (*metis LNil-eq-ltake-iff gr-implies-not-zero lfusecat-LNil llength-eq-0 llist.disc(1) ridx-def zero-enat-def*)
next
case (*Suc n*)
then show ?case
proof –
have 0: $n=0 \implies ?thesis$
proof –
assume a0: $n=0$
have 000: $\neg \text{lnull } xss$
using *Suc.prem(5) a0 one-enat-def* **by** *force*
have 001: $(\text{ltake } (\text{enat } (\text{Suc } n)) \ xss) = (\text{LCons } (\text{lnth } xss \ 0) \ \text{LNil})$
using a0 000
by (*metis One-nat-def lhd-LCons-ltl lnth-0-conv-lhd ltake.ctr(1) ltake-eSuc-LCons one-eSuc one-enat-def*)
have 002: $\text{lfusecat } (\text{ltake } (\text{enat } (\text{Suc } n)) \ xss) = (\text{lnth } xss \ 0)$
using a0 000 001 *lfusecat-LCons[of (lnth xss 0) LNil]* **by** *simp*
show ?thesis **using** 002 *Suc.prem(4) Suc.prem(5) Suc-ile-eq a0* **by** *auto*
qed
have 01: $n>0 \implies ?thesis$
proof –
assume a: $n>0$
have 1: $(\text{lfusecat } (\text{ltake } (\text{enat } (\text{Suc } n)) \ xss)) =$
 $\text{ltake } (\text{eSuc } (\sum i::\text{nat} = 0::\text{nat}.. \text{Suc } n - (1::\text{nat}). \text{epred } (\text{length } (\text{lnth } xss \ i)))) (\text{lfusecat } xss)$
using *lfusecat-ltake[of xss (Suc n)]*
using *Suc.prem(1) Suc.prem(2) Suc.prem(3) Suc.prem(5) Suc-ile-eq* **by** *force*
have 2: $\text{ridx } R (\text{lfusecat } (\text{ltake } (\text{enat } n) \ xss))$
using *Suc Suc-ile-eq order.strict-implies-order* **by** *blast*
have 20: $\text{lfinite } (\text{ltake } (\text{enat } n) \ xss)$
by *simp*
have 21: $\neg \text{lnull } (\text{ltake } (\text{enat } n) \ xss)$
using *Suc.prem(5) a enat-0-iff(2)* **by** *fastforce*

```

have 22:  $\forall xs \in lset (ltake (enat n) xss). \neg lnull xs$ 
  by (meson Suc.prem(2) lset-ltake subsetD)
have 23:  $\forall xs \in lset (ltake (enat n) xss). lfinite xs$ 
  by (meson Suc.prem(3) lset-ltake subsetD)
have 24: llastfirst (ltake (enat n) xss)
  using Suc.prem(1) Suc.prem(5) Suc-ile-eq less-imp-le llastfirst-ltake by blast
have 25: llast (lfusecat (ltake (enat n) xss)) = llast (llast (ltake (enat n) xss))
  using 20 21 22 23 24 llastfirst-lfusecat-llast by blast
have 26: lfinite (llast (ltake (enat n) xss))
  by (metis Suc.prem(3) Suc.prem(5) Suc-ile-eq Suc-pred' a enat-ord-code(4) in-lset-conv-lnth
    lfinite-ltake llast-lappend-LCons llast-singleton ltake-Suc-conv-snoc-lnth order-less-imp-le)
have 27:  $n < llength xss$ 
  using Suc.prem(5) Suc-ile-eq order-less-imp-le by blast
have 271: (llast (ltake (enat n) xss)) = (lnth xss (n-1))
  by (metis 27 Suc-diff-1 Suc-ile-eq a enat-ord-code(4) lfinite-ltake llast-lappend-LCons
    llast-singleton ltake-Suc-conv-snoc-lnth order.strict-implies-order)
have 28: llast (llast (ltake (enat n) xss)) = llast (lnth xss (n-1))
  using 271 by auto
have 29: llast (lnth xss (n-1)) = lfirst (lnth xss n)
  by (metis 27 Suc.prem(1) Suc-pred' a llastfirst-def)
have 30: ridx R (lfuse (lfusecat (ltake (enat n) xss)) (lnth xss (n)))
  by (metis 2 25 27 28 29 Suc.prem(4) lfuse-inf ridx-lfuse-lfinite)
have 31: (lfuse (lfusecat (ltake (enat n) xss)) (lnth xss (n))) =
  (lfusecat (ltake (enat (Suc n)) xss))
  by (simp add: 27 lfusecat-lappend ltake-Suc-conv-snoc-lnth)
show ?thesis
using 30 31 by auto
qed
show ?thesis
using 0 01 by fastforce
qed
qed

```

lemma lfusecat-ltake-llength-less-than-llength-lfusecat:

assumes llastfirst xss

$\forall xs \in lset xss. 1 < llength xs$

$\forall xs \in lset xss. lfinite xs$

$(enat n) \leq llength xss$

shows min

(if $n = 0$ then 0 else $eSuc(\sum i = 0 .. (n-1) . epred(llength (lnth xss i)))$)

$(llength (lfusecat xss)) =$

(if $n = 0$ then 0 else $eSuc(\sum i = 0 .. (n-1) . epred(llength (lnth xss i)))$)

proof –

have 0: $\forall xs \in lset xss. \neg lnull xs$

using assms(2) **by** fastforce

have 1: $lfinite xss \implies \neg lnull xss \implies llength(lfusecat xss) =$

$eSuc(\sum i = 0 .. (the-enat(epred(llength xss))) . epred(llength (lnth xss i)))$

using lfusecat-llength-lfinite[of xss] assms 0 **by** fastforce

have 2: $\neg lfinite xss \implies \neg lfinite (lfusecat xss)$

using assms lfusecat-lfinite[of xss] 0

by (metis less-numeral-extra(4) lhd-LCons-ltl llength-LCons llength-LNil llist.collapse(1) one-eSuc)
have 3: $\neg \text{lfinite } xss \implies \text{llength } (\text{lfusecat } xss) = \infty$
 using not-lfinite-llength 2 by blast
have 50: $\neg \text{lnull } xss \implies \text{lfusecat } (\text{ltake } (\text{enat } n) \ xss) =$
 $\text{ltake } (\text{if } n = 0 \text{ then } 0 \text{ else } \text{eSuc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i))))$
 $(\text{lfusecat } xss)$
 using assms lfusecat-ltake[of xss n] 0 by fastforce
have 51: $\text{lfinite } (\text{lfusecat } (\text{ltake } (\text{enat } n) \ xss))$
 by (meson assms(3) enat-ord-code(4) lfinite-ltake lfusecat-lfinite-a lset-ltake subsetD)
have 52: $\text{lfinite } (\text{ltake } (\text{if } n = 0 \text{ then } 0 \text{ else } \text{eSuc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i))))$
 $(\text{lfusecat } xss))$
 using 50 51 llist.collapse(1) by fastforce
have 53: $\exists m. \text{llength } (\text{ltake } (\text{if } n = 0 \text{ then } 0 \text{ else } \text{eSuc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i))))$
 $(\text{lfusecat } xss)) = (\text{enat } m)$
 using 52 lfinite-llength-enat by blast
have 54: $\text{llength } (\text{ltake } (\text{if } n = 0 \text{ then } 0 \text{ else } \text{eSuc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i))))$
 $(\text{lfusecat } xss)) \leq \text{llength } (\text{lfusecat } xss)$
 by auto
have 55: $\text{llength } (\text{ltake } (\text{if } n = 0 \text{ then } 0 \text{ else } \text{eSuc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i))))$
 $(\text{lfusecat } xss)) =$
 \min
 $(\text{if } n = 0 \text{ then } 0 \text{ else } \text{eSuc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i))))$
 $(\text{llength } (\text{lfusecat } xss))$
 using llength-ltake[of (if n = 0 then 0 else eSuc(\sum i = 0 .. (n-1) . epred(llength (lnth xss i))))
 $(\text{lfusecat } xss)]$ by auto
have 56: \min
 $(\text{if } n = 0 \text{ then } 0 \text{ else } \text{eSuc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i))))$
 $(\text{llength } (\text{lfusecat } xss)) < \infty$
 by (metis 53 55 enat-ord-code(4))
have 57: $(\text{llength } (\text{lfusecat } xss)) = \text{llength } (\text{lfusecat } (\text{lappend } (\text{ltake } n \ xss) \ (\text{ldrop } n \ xss)))$
 by (simp add: lappend-ltake-ldrop)
have 58: $\text{llength } (\text{lfusecat } (\text{lappend } (\text{ltake } n \ xss) \ (\text{ldrop } n \ xss))) =$
 $\text{llength } (\text{lfusecat } (\text{ltake } n \ xss)) (\text{lfusecat } (\text{ldrop } n \ xss))$
 by (metis 57 lfusecat-split)
have 59: $\text{lnull } (\text{lfusecat } (\text{ltake } (\text{enat } n) \ xss)) \longleftrightarrow n = 0$
 proof (cases xss)
 case LNil
 then show ?thesis using assms enat-0-iff(2) by force
 next
 case (LCons x21 x22)
 then show ?thesis using 1 2 50 by force
 qed
have 60: $\text{llength } (\text{lfusecat } (\text{ltake } n \ xss)) (\text{lfusecat } (\text{ldrop } n \ xss)) =$
 $\text{llength } (\text{lfusecat } (\text{ltake } (\text{enat } n) \ xss)) +$
 $(\text{if } n = 0 \text{ then } \text{llength } (\text{lfusecat } (\text{ldrop } (\text{enat } n) \ xss))$
 $\text{else } \text{epred } (\text{llength } (\text{lfusecat } (\text{ldrop } (\text{enat } n) \ xss))))$
 using lfuse-llength[of (lfusecat (ltake n xss)) (lfusecat (ldrop n xss))]
 using 59 by presburger
have 62: $n = 0 \implies \text{llength } (\text{lfusecat } (\text{ldrop } (\text{enat } n) \ xss)) = \text{llength } (\text{lfusecat } xss)$
 using 57 58 59 llist.collapse(1) by force

```

have 620:  $n > 0 \implies n < \text{length } xss \implies \neg \text{lnull } (\text{ldrop } (\text{enat } n) \ xss)$ 
  using Suc-ile-eq assms by simp
have 621:  $n > 0 \implies n < \text{length } xss \implies (\exists xs \in \text{lset } (\text{ldrop } (\text{enat } n) \ xss). \neg \text{lnull } xs)$ 
  by (meson 0 620 in-lset-ldropD lfusecat-not-lnull lfusecat-not-lnull-var)
have 622:  $n > 0 \implies n < \text{length } xss \implies (\neg \text{lnull } (\text{lfusecat } (\text{ldrop } (\text{enat } n) \ xss)))$ 
  using 620 621 lfusecat-not-lnull[of (ldrop (enat n) xss) ] by blast
have 623:  $n > 0 \implies \neg \text{lnull } xss$ 
  using assms 59 by force
have 63:  $n > 0 \implies n < \text{length } xss \implies (\text{length } (\text{lfusecat } (\text{ldrop } (\text{enat } n) \ xss))) =$ 
   $(\text{length } (\text{ldrop } ((\sum i = 0 \dots (n-1) . \text{epred}(\text{length } (\text{lnth } xss \ i)))) (\text{lfusecat } xss) ))$ 
  using lfusecat-ldrop[of xss n] assms
  using 623 length-LNil not-one-less-zero 0 by presburger
have 630:  $n > 0 \implies n < \text{length } xss \implies \text{epred}(\text{length } (\text{lfusecat } (\text{ldrop } (\text{enat } n) \ xss))) =$ 
   $\text{epred}(\text{length } (\text{ldrop } ((\sum i = 0 \dots (n-1) . \text{epred}(\text{length } (\text{lnth } xss \ i)))) (\text{lfusecat } xss) ))$ 
  using 63 by presburger
have 631:  $n > 0 \implies n < \text{length } xss \implies$ 
   $(\text{length } (\text{ldrop } ((\sum i = 0 \dots (n-1) . \text{epred}(\text{length } (\text{lnth } xss \ i)))) (\text{lfusecat } xss) )) > 0$ 
  by (metis 622 63 gr-zeroI length-eq-0)
have 64:  $\text{length } (\text{lfusecat } (\text{ltake } (\text{enat } n) \ xss)) \leq$ 
   $\text{length } (\text{lfuse } (\text{lfusecat } (\text{ltake } n \ xss)) (\text{lfusecat } (\text{ldrop } n \ xss)))$ 
  using lfuse-length-le-a by blast
have 65:  $0 < n \implies n < \text{length } xss \implies 0 < (\text{length } (\text{lfusecat } (\text{ldrop } (\text{enat } n) \ xss)))$ 
  using 63 631 by presburger
have 66:  $n > 0 \implies \text{length } xss > 0$ 
  using 623 gr-zeroI length-eq-0 by blast
have 67:  $n > 0 \implies n - 1 \leq (\text{epred } (\text{length } xss))$ 
  using assms 66
  by (metis epred-enat epred-le-epredI)
have 68:  $\min$ 
   $(\text{if } n = 0 \text{ then } 0 \text{ else } \text{eSuc}(\sum i = 0 \dots (n-1) . \text{epred}(\text{length } (\text{lnth } xss \ i))))$ 
   $(\text{length } (\text{lfusecat } xss)) =$ 
   $(\text{if } n = 0 \text{ then } 0 \text{ else } \text{eSuc}(\sum i = 0 \dots (n-1) . \text{epred}(\text{length } (\text{lnth } xss \ i))))$ 
proof (cases lfinite xss)
case True
then show ?thesis
  proof –
    have 681:  $n > 0 \implies n < \text{length } xss \implies \text{eSuc}(\sum i = 0 \dots (n-1) . \text{epred}(\text{length } (\text{lnth } xss \ i))) \leq$ 
     $\text{eSuc}(\sum i = 0 \dots \text{the-enat } (\text{epred } (\text{length } xss)). \text{epred } (\text{length } (\text{lnth } xss \ i)))$ 
    by (metis 1 631 True gr-implies-not-zero ileI1 linorder-not-less length-lnull lnull-ldrop)
    have 682:  $n > 0 \implies n = \text{length } xss \implies \text{eSuc}(\sum i = 0 \dots (n-1) . \text{epred}(\text{length } (\text{lnth } xss \ i))) \leq$ 
     $\text{eSuc}(\sum i = 0 \dots \text{the-enat } (\text{epred } (\text{length } xss)). \text{epred } (\text{length } (\text{lnth } xss \ i)))$ 
    by (metis dual-order.refl epred-enat the-enat.simps)
    show ?thesis
    using 1 681 682 True 623 assms(4) min.absorb1 order.order-iff-strict by auto
  qed
next
case False
then show ?thesis using 3 min-enat-simps(4) by presburger
qed
show ?thesis

```

using 68 by blast
qed

lemma *lfusecat-ltake-llength-less-than-next:*

assumes *llastlfirst xss*

$\forall xs \in \text{lset } xss. 1 < \text{llength } xs$

$\forall xs \in \text{lset } xss. \text{lfinite } xs$

$(\text{Suc } n) < \text{llength } xss$

shows $(\text{if } n = 0 \text{ then } 0 \text{ else } e\text{Suc}(\sum i = 0 .. (n-1) . e\text{pred}(\text{llength } (\text{lnth } xss \ i))))$
 $< (\text{if } (\text{Suc } n) = 0 \text{ then } 0 \text{ else } e\text{Suc}(\sum i = 0 .. (n) . e\text{pred}(\text{llength } (\text{lnth } xss \ i))))$

proof –

have 0: $n < \infty$

by *simp*

have 1: $\bigwedge i. i < \text{llength } xss \implies 1 < \text{llength } (\text{lnth } xss \ i)$

by (*meson assms(2) in-lset-conv-lnth*)

have 01: $\bigwedge i. i < \text{llength } xss \implies \neg \text{lnull } (\text{lnth } xss \ i)$

by (*metis 1 llength-LNil llist.collapse(1) not-one-less-zero*)

have 02: $\forall xs \in \text{lset } xss. \neg \text{lnull } xs$

using *assms(2)* **by** *fastforce*

have 2: $\bigwedge i . i < \text{llength } xss \implies 0 < e\text{pred}(\text{llength } (\text{lnth } xss \ i))$

by (*metis 1 co.enat.exhaust-sel gr-zeroI less-numeral-extra(4) not-one-less-zero one-eSuc*)

have 3: $n = 0 \implies ?thesis$

by *auto*

have 4: $0 < n \implies (\sum i = 0 .. (n) . e\text{pred}(\text{llength } (\text{lnth } xss \ i))) =$
 $(\sum i = 0 .. (n-1) . e\text{pred}(\text{llength } (\text{lnth } xss \ i))) +$
 $e\text{pred}(\text{llength } (\text{lnth } xss \ n))$

by (*metis (no-types, lifting) Suc-diff-1 sum.atLeast0-atMost-Suc*)

have 5: $\bigwedge i. i < \text{llength } xss \implies e\text{pred}(\text{llength } (\text{lnth } xss \ i)) < \infty$

using *assms(3)*

by (*metis enat-ord-simps(4) epred-0 epred-Infty epred-inject in-lset-conv-lnth infinity-ne-i0*
 $\text{llength-eq-infty-conv-lfinite}$)

have 50: $\text{llength } (\text{lfusecat } (\text{ltake } n \ xss)) \leq \text{llength } (\text{lfusecat } xss)$

by (*simp add: lprefix-lfusecatI lprefix-llength-le*)

have 6: $0 < n \implies e\text{Suc}(\sum i = 0 .. (n-1) . e\text{pred}(\text{llength } (\text{lnth } xss \ i))) < \infty$

proof –

assume *a: n > 0*

have 60: $(\text{lfusecat } (\text{ltake } n \ xss)) =$

$(\text{ltake } (e\text{Suc}(\sum i = 0 .. (n-1) . e\text{pred}(\text{llength } (\text{lnth } xss \ i)))) (\text{lfusecat } xss))$

using *lfusecat-ltake[of xss n]* **using** *assms 02 a* **by** *simp*

$(\text{metis Suc-ile-eq gr-implies-not-zero llength-eq-0 order.strict-implies-order})$

have 61: $\text{llength } (\text{lfusecat } (\text{ltake } n \ xss)) =$

$\text{llength } (\text{ltake } (e\text{Suc}(\sum i = 0 .. (n-1) . e\text{pred}(\text{llength } (\text{lnth } xss \ i)))) (\text{lfusecat } xss))$

using 60 **by** *fastforce*

have 62: $\text{llength } (\text{ltake } (e\text{Suc}(\sum i = 0 .. (n-1) . e\text{pred}(\text{llength } (\text{lnth } xss \ i)))) (\text{lfusecat } xss)) =$
 $\min (e\text{Suc}(\sum i = 0 .. (n-1) . e\text{pred}(\text{llength } (\text{lnth } xss \ i))))$
 $(\text{llength } (\text{lfusecat } xss))$

```

    using llength-ltake by blast
  have 63: min (eSuc(∑ i = 0 .. (n-1) . epred(llength (lnth xss i))))
    (llength (lfusecat xss)) =
    (eSuc(∑ i = 0 .. (n-1) . epred(llength (lnth xss i))))
    using lfusecat-ltake-llength-less-than-llength-lfusecat[of xss n] a assms
    using Suc-ile-eq order-less-imp-le by simp blast
  show ?thesis
  by (metis 0 60 62 63 assms(3) enat-ord-simps(4) lfinite-ltake lfusecat-lfinite-a
    llength-eq-infty-conv-lfinite lset-ltake subsetD)
qed
have 7: 0 < n ⟹ eSuc(∑ i = 0 .. (n-1) . epred(llength (lnth xss i))) <
  eSuc(∑ i = 0 .. (n-1) . epred(llength (lnth xss i))) +
  epred(llength (lnth xss n))
  by (metis 2 6 Suc-ile-eq add.right-neutral assms(4) enat-add-mono less-infinityE
    order-less-imp-le)
have 8: 0 < n ⟹ ?thesis
  using 4 7 eSuc-plus by auto
show ?thesis
using 3 8 by blast
qed

```

lemma *ridx-lfusecat-b*:

assumes *llastlfirst xss*

$\forall xs \in lset\ xss. 1 < llength\ xs$

$\forall xs \in lset\ xss. lfinite\ xs$

$\forall i. i < llength\ xss \longrightarrow ridx\ R\ (lnth\ xss\ i)$

shows $ridx\ R\ (lfusecat\ xss)$

proof –

have 0: $\forall xs \in lset\ xss. \neg lnull\ xs$

using *assms(2) gr-implies-not-zero llength-LNil llist.collapse(1)* **by** *blast*

have 1: $\bigwedge n. n \leq llength\ xss \implies ridx\ R\ (lfusecat\ (ltake\ (enat\ n)\ xss))$

using *assms 0 ridx-lfusecat-ltake-a* **by** *blast*

have 2: $\bigwedge n. n \leq llength\ xss \implies$

$ridx\ R\ (ltake\ (if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . epred(llength (lnth\ xss\ i))))$
 $(lfusecat\ xss))$

using *lfusecat-ltake[of xss] 1 assms using 0 llist.collapse(1)* **by** *fastforce*

have 30: $\bigwedge n. (enat\ n) \leq llength\ xss \implies$

$(if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . epred(llength (lnth\ xss\ i)))) \leq$
 $llength\ (lfusecat\ xss)$

proof –

fix *n::nat*

assume *a: n ≤ llength xss*

show $(if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . epred(llength (lnth\ xss\ i)))) \leq$
 $llength\ (lfusecat\ xss)$

using *lfusecat-ltake-llength-less-than-llength-lfusecat[of xss n] 0 assms*

a min.order-iff[of (if n = 0 then 0 else eSuc(∑ i = 0 .. (n-1) . epred(llength (lnth xss i))))
 $llength\ (lfusecat\ xss)]$

by *presburger*

qed

have 3: $\bigwedge n. (enat\ n) \leq llength\ xss \implies$

(if $n = 0$ then 0 else $eSuc(\sum i = 0 .. (n-1) . ePred(lLength (lNth xss i)))$) \leq
 $lLength (lfuse (lfusecat (lTake n xss)) (lfusecat (ldrop n xss)))$
 by (metis 30 lfusecat-split)
have 6: $\bigwedge n. (Suc\ n) < lLength\ xss \implies$
 (if $n = 0$ then 0 else $eSuc(\sum i = 0 .. (n-1) . ePred(lLength (lNth xss i)))$) $<$
 (if $(Suc\ n) = 0$ then 0 else $eSuc(\sum i = 0 .. (n) . ePred(lLength (lNth xss i)))$)
 using *assms(1) assms(2) assms(3) assms(4) lfusecat-lTake-lLength-less-than-next* by blast
have 61: $\bigwedge i. i < lLength\ xss \implies ePred(lLength (lNth\ xss\ i)) < \infty$
 by (metis *assms(3) enat-ord-simps(4) ePred-0 ePred-Infty ePred-inject i0-ne-infinity*
in-lset-conv-lNth lLength-eq-infty-conv-lfinite)
have 62: $\bigwedge n. n \leq lLength\ xss \implies$
 (if $n = 0$ then 0 else $eSuc(\sum i = 0 .. (n-1) . ePred(lLength (lNth xss i)))$) $< \infty$
 by (metis (*no-types, lifting*) 30 6 *add-diff-cancel-left' assms(3) eSuc-enat enat-ord-code(4)*
enat-ord-simps(4) ileI1 leD lfusecat-lfinite-a lLength-eq-infty-conv-lfinite plus-1-eq-Suc)
have 7: $\bigwedge k\ n. k \leq (if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . ePred(lLength (lNth\ xss\ i))))$
 $\wedge\ n \leq lLength\ xss$
 \implies
 $ridx\ R\ (lTake\ (enat\ k)\ (lfusecat\ xss))$
proof –
fix $k\ n$
show $k \leq (if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . ePred(lLength (lNth\ xss\ i))))$
 $\wedge\ n \leq lLength\ xss$
 $\implies ridx\ R\ (lTake\ (enat\ k)\ (lfusecat\ xss))$
 using *ridx-lTake[of R (if n = 0 then 0 else eSuc(∑ i = 0 .. (n-1) . ePred(lLength (lNth xss i))))*
(lfusecat xss) k]
 using 2 30 by *presburger*
qed
have 5: $\bigwedge k. k < lLength\ xss \implies$
 $ridx\ R\ (lTake\ (enat\ k)\ (lfusecat\ xss))$
proof –
fix k
show $k < lLength\ xss \implies$
 $ridx\ R\ (lTake\ (enat\ k)\ (lfusecat\ xss))$
proof (*cases lfinite xss*)
case *True*
then show *?thesis*
 by (metis 1 *lfinite-lLength-enat linorder-le-cases lTake-all ridx-lTake-a*)
next
case *False*
then show *?thesis*
proof –
have 51: $\neg lfinite(lfusecat\ xss)$
 using *assms 0*
 by (metis *False ileSS-Suc-eq lfusecat-lfinite-b lhd-LCons-ltl*
lLength-LCons not-lnull-lLength one-enat-def)
have 52: $\bigwedge k. (\exists n. enat\ k < (if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . ePred(lLength (lNth\ xss\ i))))$
 $\wedge\ n \leq lLength\ xss \wedge$
 $ridx\ R\ (lTake\ (if\ n = 0\ then\ 0\ else$
 $eSuc(\sum i = 0 .. (n-1) . ePred(lLength (lNth\ xss\ i)))) (lfusecat\ xss)))$

```

proof –
  fix  $k$ 
  show  $(\exists n. \text{enat } k < (\text{if } n = 0 \text{ then } 0 \text{ else } e\text{Suc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i))))$ 
     $\wedge n \leq \text{llength } xss \wedge$ 
     $\text{ridx } R (\text{ltake } (\text{if } n = 0 \text{ then } 0 \text{ else } e\text{Suc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i)))) (\text{lfusecat } xss)))$ 
    using 2 6
  proof (induct  $k$ )
    case 0
    then show ?case
      proof –
        have 521:  $(\text{enat } 0) < e\text{Suc}(\sum i = 0 .. (1-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i)))$ 
          using i0-iless-eSuc zero-enat-def by presburger
        have 522:  $(\text{enat } 1) \leq \text{llength } xss$ 
          using False
          using lnull-imp-lfinite not-lnull-llength one-enat-def by auto
        have 523:  $\text{ridx } R (\text{ltake } (\text{if } (\text{enat } 1) = 0 \text{ then } 0 \text{ else } e\text{Suc}(\sum i = 0 .. (1-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i)))) (\text{lfusecat } xss))$ 
          using 0.premis(1) 522 one-enat-def by fastforce
        show ?thesis using 521 522 523 one-enat-def by force
      qed
    next
    case (Suc  $k$ )
    then show ?case
      by (metis (no-types, lifting) False antisym-conv2 diff-Suc-1 eSuc-enat enat-ord-code(4)
        ileI1 llength-eq-infnty-conv-lfinite)
    qed
  qed
have 53:  $\bigwedge k. (\exists m. (\text{enat } k) < m \wedge \text{ridx } R (\text{ltake } (\text{enat } m) (\text{lfusecat } xss)))$ 
proof –
  fix  $k$ 
show  $(\exists m. (\text{enat } k) < m \wedge \text{ridx } R (\text{ltake } (\text{enat } m) (\text{lfusecat } xss)))$ 
proof –
  have 54:  $(\exists n. \text{enat } k < (\text{if } n = 0 \text{ then } 0 \text{ else } e\text{Suc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i))))$ 
     $\wedge n \leq \text{llength } xss \wedge$ 
     $\text{ridx } R (\text{ltake } (\text{if } n = 0 \text{ then } 0 \text{ else } e\text{Suc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i)))) (\text{lfusecat } xss)))$ 
    using 52 by blast
  obtain  $n$  where 55:  $\text{enat } k < (\text{if } n = 0 \text{ then } 0 \text{ else } e\text{Suc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i)))) \wedge$ 
     $n \leq \text{llength } xss \wedge$ 
     $\text{ridx } R (\text{ltake } (\text{if } n = 0 \text{ then } 0 \text{ else } e\text{Suc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss \ i)))) (\text{lfusecat } xss))$ 
    using 54 by blast
  show ?thesis
    using 55 62 by force
  qed
qed
show ?thesis

```


by (metis 51 53 enat-ord-code(4) llength-eq-infty-conv-lfinite order.strict-implies-order
 ridx-ltake)
 qed
 qed
 qed
 show ?thesis
 by (metis 1 5 dual-order.refl enat-ord-code(4) lfinite-conv-llength-enat
 llength-eq-infty-conv-lfinite ltake-all ridx-ltake-all)
 qed

lemma *ridx-lfusecat*:
assumes *llastlfirst xss*
 $\forall xs \in \text{lset } xss. 1 < \text{llength } xs$
 $\forall xs \in \text{lset } xss. \text{lfinite } xs$
shows $\text{ridx } R (\text{lfusecat } xss) \longleftrightarrow (\forall i. i < \text{llength } xss \longrightarrow \text{ridx } R (\text{lnth } xss i))$
using *ridx-lfusecat-a ridx-lfusecat-b assms*
using *gr-implies-not-zero llength-eq-0* **by** *blast*

lemma *lfusecat-split-lsub*:
assumes *llastlfirst xss*
 $\forall xs \in \text{lset } xss. 1 < \text{llength } xs$
 $\forall xs \in \text{lset } xss. \text{lfinite } xs$
shows $(\forall i. i < \text{llength } xss \longrightarrow \text{ridx } (\lambda a b. f (\text{lsub } a b \sigma)) (\text{lnth } xss i)) \longleftrightarrow$
 $\text{ridx } (\lambda a b. f (\text{lsub } a b \sigma)) (\text{lfusecat } xss)$
using *assms ridx-lfusecat* **by** *blast*

lemma *lidx-lfuse-lfinite*:
assumes *lfinite xs*
 $\text{llast } xs = \text{lfirst } ys$
shows $\text{lidx } (\text{lfuse } xs \text{ } ys) \longleftrightarrow \text{lidx } xs \wedge \text{lidx } ys$
using *assms*
using *ridx-lfuse-lfinite ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-ltake*:
assumes *lidx (lfusecat xss)*
 $n \leq \text{llength } xss$
shows $\text{lidx } (\text{lfusecat } (\text{ltake } n \text{ } xss))$
using *assms ridx-lfusecat-ltake ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-ldrop*:
assumes *lidx (lfusecat xss)*
 $(\text{enat } n) < \text{llength } xss$
 $\text{llastlfirst } xss$
 $\forall xs \in \text{lset } xss. \neg \text{lnull } xs$
 $\forall xs \in \text{lset } xss. \text{lfinite } xs$
shows $\text{lidx } (\text{lfusecat } (\text{ldrop } n \text{ } xss))$
using *assms ridx-lfusecat-ldrop ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-a*:

assumes *llastlfirst xss*
 $\forall xs \in lset\ xss. \neg lnull\ xs$
 $\forall xs \in lset\ xss. lfinite\ xs$
 $lidx\ (lfusecat\ xss)$
 $i < llength\ xss$
shows $lidx\ (lnth\ xss\ i)$
using *assms ridx-lfusecat-a ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-lfuse*:
assumes *llastlfirst xss*
 $\forall xs \in lset\ xss. \neg lnull\ xs$
 $\forall xs \in lset\ xss. lfinite\ xs$
 $\forall i. i < llength\ xss \longrightarrow lidx\ (lnth\ xss\ i)$
 $(Suc\ n) < llength\ xss$
shows $lidx\ (lfuse\ (lnth\ xss\ n)\ (lnth\ xss\ (Suc\ n)))$
using *assms ridx-lfusecat-lfuse ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-ltake-a*:
assumes *llastlfirst xss*
 $\forall xs \in lset\ xss. \neg lnull\ xs$
 $\forall xs \in lset\ xss. lfinite\ xs$
 $\forall i. i < llength\ xss \longrightarrow lidx\ (lnth\ xss\ i)$
 $n \leq llength\ xss$
shows $lidx\ (lfusecat\ (ltake\ (enat\ n)\ xss))$
using *assms ridx-lfusecat-ltake-a ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-b*:
assumes *llastlfirst xss*
 $\forall xs \in lset\ xss. 1 < llength\ xs$
 $\forall xs \in lset\ xss. lfinite\ xs$
 $\forall i. i < llength\ xss \longrightarrow lidx\ (lnth\ xss\ i)$
shows $lidx\ (lfusecat\ xss)$
using *assms ridx-lfusecat-b ridx-lidx* **by** *blast*

lemma *lidx-lfusecat*:
assumes *llastlfirst xss*
 $\forall xs \in lset\ xss. 1 < llength\ xs$
 $\forall xs \in lset\ xss. lfinite\ xs$
shows $lidx\ (lfusecat\ xss) \longleftrightarrow (\forall i. i < llength\ xss \longrightarrow lidx\ (lnth\ xss\ i))$
using *assms ridx-lfusecat ridx-lidx* **by** *blast*

lemma *lnth-sum-expand*:
assumes $\neg lnull\ xss$
 $llastlfirst\ xss$
 $\forall xs \in lset\ xss. 1 < llength\ xs$

$\forall xs \in lset\ xss. lfinite\ xs$
 $(enat\ i) < llength\ xss$
 $lfinite\ xss$
shows $(\sum i = 0 .. (the-enat(epred(llength\ xss))) . epred(llength\ (lnth\ xss\ i))) =$
 $epred(llength\ (lnth\ xss\ i)) +$
 $(\sum j \in \{k. k \neq i \wedge k \leq (the-enat(epred(llength\ xss)))\} . epred(llength\ (lnth\ xss\ j)))$
proof –
have 1: $\{k. k \leq (the-enat(epred(llength\ xss)))\} = insert\ i\ \{k. k \neq i \wedge k \leq (the-enat(epred(llength\ xss)))\}$
using *assms by auto*
 $(metis\ eSuc-epred\ enat-ord-simps(1)\ epred-enat\ gr-implies-not-zero\ illess-Suc-eq$
 $lfinite-llength-enat\ the-enat.simps)$
have 2: $\{0.. (the-enat(epred(llength\ xss)))\} = \{k. k \leq (the-enat(epred(llength\ xss)))\}$
by *auto*
have 3: $(\sum i = 0 .. (the-enat(epred(llength\ xss))) . epred(llength\ (lnth\ xss\ i))) =$
 $(\sum i \in \{k. k \leq (the-enat(epred(llength\ xss)))\} . epred(llength\ (lnth\ xss\ i)))$
using 2 **by** *presburger*
have 4: $(\sum i \in \{k. k \leq (the-enat(epred(llength\ xss)))\} . epred(llength\ (lnth\ xss\ i))) =$
 $(\sum j \in (insert\ i\ \{k. k \neq i \wedge k \leq (the-enat(epred(llength\ xss)))\}) . epred(llength\ (lnth\ xss\ j)))$
using 1 **by** *presburger*
have 5: $(\sum j \in (insert\ i\ \{k. k \neq i \wedge k \leq (the-enat(epred(llength\ xss)))\}) . epred(llength\ (lnth\ xss\ j))) =$
 $epred(llength\ (lnth\ xss\ i)) +$
 $(\sum j \in \{k. k \neq i \wedge k \leq (the-enat(epred(llength\ xss)))\} . epred(llength\ (lnth\ xss\ j)))$
by *auto*
show *?thesis*
using 3 4 5 **by** *presburger*
qed

lemma *lfusecat-llength-a:*

assumes *llastlfirst xss*
 $\forall xs \in lset\ xss. 1 < llength\ xs$
 $\forall xs \in lset\ xss. lfinite\ xs$
 $i < llength\ xss$
 $(enat\ j) < llength\ (lnth\ xss\ i)$
shows $(enat\ j) < llength\ (lfusecat\ xss)$
proof (*cases lfinite xss*)
case *True*
then show *?thesis*
proof –
have 1: $lnull\ xss \implies ?thesis$
using *assms(4) gr-implies-not-zero llength-eq-0 by blast*
have 2: $\neg lnull\ xss \implies$
 $llength\ (lfusecat\ xss) =$
 $eSuc(\sum i = 0 .. (the-enat(epred(llength\ xss))) . epred(llength\ (lnth\ xss\ i)))$
using *True assms lfusecat-llength-lfinite by fastforce*
have 3: $\neg lnull\ xss \implies$
 $llength\ (lnth\ xss\ i) \leq$
 $eSuc(\sum i = 0 .. (the-enat(epred(llength\ xss))) . epred(llength\ (lnth\ xss\ i)))$
using *lnth-sum-expand[of xss i] assms*
by (*metis (no-types, lifting) True co.enat.collapse eSuc-plus gr-implies-not-zero le-iff-add*)

```

have 4:  $\neg \text{lnull } xss \implies ?thesis$ 
  using 2 3
  by (metis assms(5) order-less-le-trans)
show ?thesis
  using 1 4 by blast
qed
next
case False
then show ?thesis
  proof -
    have 5:  $\neg \text{lfinite } (\text{lfusecat } xss)$ 
      by (metis False assms(1) assms(2) assms(3) gr-implies-not-zero iless-Suc-eq lfusecat-lfinite-b
        lhd-LCons-ltl llength-LCons llength-eq-0 not-lnull-llength one-enat-def)
    show ?thesis
      using 5 enat-iless lfinite-conv-llength-enat not-less-iff-gr-or-eq by blast
  qed
qed

```

```

lemma lmap-lfusecat:
   $\text{lmap } f (\text{lfusecat } xss) = (\text{lfusecat } (\text{lmap } (\text{lmap } f) ) xss)$ 
proof (induct xss)
case adm
then show ?case by simp
next
case LNil
then show ?case
  by simp
next
case (LCons xs xss)
then show ?case
  by (simp add: lfuse-def )
  (metis lmap-eq-LNil lmap-lappend-distrib lnull-def ltl-lmap)
qed

```

```

lemma lfusecat-is-lfirst-conv:
  assumes  $\forall i. i < \text{llength } xss \longrightarrow \text{is-lfirst}(\text{lnth } xss \ i)$ 
     $\text{is-lfirst } xss$ 
  shows  $\text{is-lfirst}(\text{lfusecat } xss)$ 
  using assms
proof (induction xss)
case adm
  then show ?case
    by (rule ccpo.admissibleI) (auto, metis lnth-0 zero-enat-def)
next
case LNil
then show ?case by simp
next
case (LCons xs xss)

```

then show *?case*
by (*simp add: zero-enat-def*)
qed

1.1.8 kfilter

lemma *kfilter-code* [*simp, code*]:
shows *kfilter-LNil*: $kfilter\ P\ n\ LNil = LNil$
and *kfilter-LCons*: $kfilter\ P\ n\ (LCons\ x\ xs) =$
 $(if\ P\ x\ then\ LCons\ n\ (kfilter\ P\ (Suc\ n)\ xs)\ else\ kfilter\ P\ (Suc\ n)\ xs)$
by (*auto simp add: kfilter-def lzip.ctr(2)*)

lemma *lmap-fst-imp-a*:
assumes *lnull* (*lfilter* *P* *xs*)
shows *lnull* (*lmap* *fst* (*lfilter* (*P* \circ *fst*) (*lzip* *xs* (*iterates* *Suc* *n*))))
using *assms lset-lzipD1* **by** *fastforce*

lemma *lmap-fst-imp-b*:
assumes *lnull* (*lmap* *fst* (*lfilter* (*P* \circ *fst*) (*lzip* *xs* (*iterates* *Suc* *n*))))
shows *lnull* (*lfilter* *P* *xs*)
proof –
have 0: *lnull* (*lmap* *fst* (*lfilter* ($\lambda\ (x,k).\ P\ x$) (*lzip* *xs* (*iterates* *Suc* *n*))))
using *assms* **by** *auto*
have 1: *lnull* (*lfilter* ($\lambda\ (x,k).\ P\ x$) (*lzip* *xs* (*iterates* *Suc* *n*))))
using 0 **by** *auto*
have 2: $(\forall\ (x,k) \in lset(lzip\ xs\ (iterates\ Suc\ n)).\ \neg\ P\ x)$
using 1 **by** (*simp add: prod.case-eq-if*)
have 3: $(\forall\ (x,k) \in \{ (lnth\ xs\ i,\ n+i) \mid i.\ enat\ i < llength\ xs \}.\ \neg\ P\ x)$
using 2 **by** (*simp add: lset-lzip*)
have 4: $(\forall\ x \in \{ lnth\ xs\ i \mid i.\ enat\ i < llength\ xs \}.\ \neg\ P\ x)$
using 3 **by** *blast*
from 4 **show** *?thesis* **by** (*simp add: lset-conv-lnth*)
qed

lemma *lmap-snd-lnull*:
 $lnull\ (lmap\ snd\ (lfilter\ (P\ \circ\ fst)\ (lzip\ xs\ (iterates\ Suc\ n)))) = lnull(lfilter\ P\ xs)$
by (*metis llist.collapse(1) llist.disc(1) lmap-eq-LNil lmap-fst-imp-a lmap-fst-imp-b*)

lemma *kfilter-lnull-conv*:
 $lnull\ (kfilter\ P\ n\ xs) \longleftrightarrow (\forall\ x \in lset\ xs.\ \neg\ P\ x)$
unfolding *kfilter-def* **using** *lmap-snd-lnull*[*of* *P* *xs*] *lnull-lfilter*[*of* *P* *xs*] **by** *blast*

lemma *kfilter-not-lnull-conv*:
 $\neg lnull\ (kfilter\ P\ n\ xs) \longleftrightarrow (\exists\ x \in lset\ xs.\ P\ x)$
by (*simp add: kfilter-lnull-conv*)

lemma *lmap-fst-lfilter*:
 $lfilter\ P\ (lmap\ fst\ (lzip\ xs\ (iterates\ Suc\ n))) =$
 $lmap\ fst\ (lfilter\ (P\ \circ\ fst)\ (lzip\ xs\ (iterates\ Suc\ n)))$
using *lfilter-lmap* **by** *blast*

lemma *lmap-fst-lzip*:

$(\text{lmap fst (lzip xs (iterates Suc n))}) = \text{xs}$

by (*coinduction arbitrary: xs*) (*auto, simp add: lmap-fst-lzip-conv-ltake*)

lemma *lfilter-kfilter-1*:

$(\text{lmap fst (lfilter (P \circ \text{fst}) (lzip xs (iterates Suc n)))) =$
 (lfilter P xs)

by (*metis (mono-tags, lifting) lmap-fst-lfilter lmap-fst-lzip*)

lemma *lfilter-kfilter-snd-llength*:

$\text{llength (lmap fst (lfilter (P \circ \text{fst}) (lzip xs (iterates Suc n))))} =$
 $\text{llength (lmap snd (lfilter (P \circ \text{fst}) (lzip xs (iterates Suc n))))}$

by *simp*

lemma *kfilter-llength*:

$\text{llength (kfilter P n xs)} = \text{llength (lfilter P xs)}$

proof –

have 1: $\text{llength (kfilter P n xs)} = \text{llength (lmap snd (lfilter (P \circ \text{fst}) (lzip xs (iterates Suc n))))}$
by (*simp add: kfilter-def*)

have 2: $\text{llength (lmap snd (lfilter (P \circ \text{fst}) (lzip xs (iterates Suc n))))} =$
 $\text{llength (lmap fst (lfilter (P \circ \text{fst}) (lzip xs (iterates Suc n))))}$

using *lfilter-kfilter-snd-llength* **by** *auto*

have 3: $\text{llength (lmap fst (lfilter (P \circ \text{fst}) (lzip xs (iterates Suc n))))} =$
 $\text{llength (lfilter P xs)}$

by (*metis lfilter-kfilter-1*)

from 1 2 3 **show** *?thesis* **by** *auto*

qed

lemma *ldropWhile-LEAST*:

assumes $\exists n < \text{llength xs. (Not } \circ P) (\text{lnth xs } n)$

shows $\text{ldropWhile P xs} = \text{ldropn (LEAST n. } n < \text{llength xs} \wedge (\text{Not } \circ P) (\text{lnth xs } n)) \text{ xs}$

proof –

from *assms* **obtain** *m* **where**

$m < \text{llength xs} \wedge (\text{Not } \circ P) (\text{lnth xs } m)$

$\bigwedge n. n < \text{llength xs} \longrightarrow (\text{Not } \circ P) (\text{lnth xs } n) \implies m \leq n$

and $*$: $(\text{LEAST n. } n < \text{llength xs} \wedge (\text{Not } \circ P) (\text{lnth xs } n)) = m$

by *atomize-elim*

$(\text{metis (no-types, lifting) dual-order.strict-trans1 enat-ord-code(2) less-imp-le not-le-imp-less}$
 $\text{not-less-Least wellorder-Least-lemma(1)})$

thus *?thesis* **unfolding** $*$

proof (*induct m arbitrary: xs*)

case 0

then show *?case*

by (*cases xs*) *simp-all*

next

case (*Suc m*)

then show *?case*

proof –

have 1: $\text{ldropWhile P (ltl xs)} = \text{ldropn m (ltl xs)}$ **using** *Suc*

```

    by (cases xs)
      (simp,
        metis Suc-le-mono ldrop-eSuc-ltl ldropn-Suc-conv-ldropn ldropn-eq-LNil llist.simps(3)
        lnth-Suc-LCons ltl-simps(2) not-le-imp-less)
  have 2: P (lnth xs 0)
    using Suc.premis(2) by auto
show ?thesis
  by (metis 1 2 ldropWhile-LCons ldrop-eSuc-ltl lhd-LCons-ltl llist.collapse(1)
    lnth-0-conv-lhd ltl-simps(1))
qed
qed
qed

```

lemma *ldropWhile-LEAST-not*:

```

assumes  $\exists n < \text{length } xs. P (\text{lnth } xs \ n)$ 
shows  $\text{ldropWhile } (Not \circ P) \ xs = \text{ldropn } (LEAST \ n. n < \text{length } xs \wedge P (\text{lnth } xs \ n)) \ xs$ 
using assms ldropWhile-LEAST[of xs Not  $\circ$  P] by simp

```

lemma *ltakeWhile-LEAST*:

```

assumes  $\exists n < \text{length } xs. (Not \circ P) (\text{lnth } xs \ n)$ 
shows  $(\text{ltakeWhile } P \ xs) = (\text{ltake } (\text{lleast } (Not \circ P) \ xs) \ xs)$ 
proof–
  from assms obtain m where
     $m < \text{length } xs \wedge (Not \circ P) (\text{lnth } xs \ m)$ 
     $\bigwedge n. n < \text{length } xs \longrightarrow (Not \circ P) (\text{lnth } xs \ n) \implies m \leq n$ 
  and *:  $(\text{lleast } (Not \circ P) \ xs) = m$  unfolding lleast-def
  by atomize-elim
  (metis (no-types, lifting) Least-le dual-order.trans enat-ord-code(2) not-less not-less-iff-gr-or-eq
    wellorder-Least-lemma(1))
  thus ?thesis unfolding *
  proof (induct m arbitrary: xs)
  case 0
  then show ?case
  proof –
  have  $\text{ltakeWhile } P \ (LCons (\text{lnth } xs \ 0) (\text{ldropn } (Suc \ 0) \ xs)) = LNil$ 
  using 0 by fastforce
  then show ?thesis
  by (metis (no-types) lappend.disc-iff(2) lappend-ltakeWhile-ldropWhile ldropn-0
    ldropn-Suc-conv-ldropn llist.collapse(1) lnull-ldropn ltake.ctr(1) not-less zero-enat-def)
  qed
  next
  case (Suc m)
  then show ?case
  proof –
  have 1:  $\text{ltakeWhile } P \ (ltl \ xs) = \text{ltake } m \ (ltl \ xs)$ 
    using Suc by (cases xs)
    (simp,
      metis Extended-Nat.eSuc-mono Suc-le-mono eSuc-enat llength-LCons lnth-Suc-LCons ltl-simps(2))
  have 2: P (lnth xs 0)
    using Suc by auto

```

```

show ?thesis
  by (metis 1 2 eSuc-enat lhd-LCons-ltl lnth-0-conv-lhd ltake.ctr(1) ltakeWhile.code
    ltake-eSuc-LCons)
qed
qed
qed

```

```

lemma llength-LEAST-not:
  assumes  $\exists n < \text{llength } xs. (\text{Not} \circ P) (\text{lnth } xs \ n)$ 
  shows  $(\text{lleast } (\text{Not} \circ P) \ xs) < \text{llength } xs$ 
  using assms unfolding lleast-def
  by (metis (no-types, lifting) LeastI)

```

```

lemma llength-LEAST:
  assumes  $\exists n < \text{llength } xs. P (\text{lnth } xs \ n)$ 
  shows  $(\text{lleast } P \ xs) < \text{llength } xs$ 
  using assms llength-LEAST-not[of xs Not  $\circ$  P] unfolding lleast-def by simp

```

```

lemma llength-ltakeWhile-LEAST:
  assumes  $\exists n < \text{llength } xs. (\text{Not} \circ P) (\text{lnth } xs \ n)$ 
  shows  $(\text{llength } (\text{ltakeWhile } P \ xs)) = (\text{lleast } (\text{Not} \circ P) \ xs)$ 
  using assms ltakeWhile-LEAST[of xs P] unfolding lleast-def
  by (metis (no-types, lifting) dual-order.strict-trans1 enat-ord-simps(2) le-cases llength-ltake
    min-def not-less-Least)

```

```

lemma llength-ltakeWhile-LEAST-not:
  assumes  $\exists n < \text{llength } xs. P (\text{lnth } xs \ n)$ 
  shows  $(\text{llength } (\text{ltakeWhile } (\text{Not} \circ P) \ xs)) = (\text{lleast } P \ xs)$ 
  using assms llength-ltakeWhile-LEAST[of xs Not  $\circ$  P] unfolding lleast-def by simp

```

```

lemma lzip-ldropWhile-fst:
  assumes  $\text{llength } (\text{lzip } xs \ ys) = \text{llength } xs$ 
  shows  $\text{lzip } (\text{ldropWhile } P \ xs) (\text{lmap snd } (\text{ldropWhile } (P \circ \text{fst}) (\text{lzip } xs \ ys))) =$ 
     $\text{ldropWhile } (P \circ \text{fst}) (\text{lzip } xs \ ys)$ 
  using assms
  proof -
  have f1:  $\text{ldropWhile } P \ xs = \text{lmap fst } (\text{ldropWhile } (P \circ \text{fst}) (\text{lzip } xs \ ys))$ 
  by (metis (no-types) llength-lzip assms ldropWhile-lmap lmap-fst-lzip-conv-ltake ltake-all order-refl)
  have ltake (min (llength (ldrop (llength (ltakeWhile (P  $\circ$  fst) (lzip xs ys))) xs))
    (llength (ldrop (llength (ltakeWhile (P  $\circ$  fst) (lzip xs ys))) ys)))
    (ldrop (llength (ltakeWhile (P  $\circ$  fst) (lzip xs ys))) (lzip xs ys)) =
    ldrop (llength (ltakeWhile (P  $\circ$  fst) (lzip xs ys))) (lzip xs ys)
  by (simp add: ltake-all)
  then show ?thesis
  using f1 by (simp add: ldropWhile-eq-ldrop lmap-fst-lzip-conv-ltake lmap-snd-lzip-conv-ltake ltake-lzip)
  qed

```

```

lemma lzip-ldropWhile-fst-iterates:
   $\text{ldropWhile } (\text{Not} \circ P \circ \text{fst}) (\text{lzip } xs \ (\text{iterates } \text{Suc } n)) =$ 
   $\text{lzip } (\text{ldropWhile } (\text{Not} \circ P) \ xs) (\text{lmap snd } (\text{ldropWhile } (\text{Not} \circ P \circ \text{fst}) (\text{lzip } xs \ (\text{iterates } \text{Suc } n))))$ 

```


by (metis llength-lmap lmap-fst-lzip lzip-ldropWhile-fst)

lemma *ldropWhile-iterates-split*:

assumes $\exists n < \text{llength } xs. P (\text{lnth } xs \ n)$

shows $((\text{ldropWhile } (\text{Not} \circ P \circ \text{fst}) (\text{lzip } xs (\text{iterates } \text{Suc } n)))) =$
 $(\text{lzip } (\text{ldropWhile } (\text{Not} \circ P) \ xs)$
 $(\text{iterates } \text{Suc } (n + (\text{lleast } P \ xs))))$

proof –

have 1: $\exists na. \text{enat } na < \text{llength } (\text{lzip } xs (\text{iterates } \text{Suc } n)) \wedge$
 $(\text{Not} \circ (\text{Not} \circ P) \circ \text{fst}) (\text{lnth } (\text{lzip } xs (\text{iterates } \text{Suc } n)) \ na)$

using *lmap-fst-lzip*[of *xs n*]

by (metis *assms comp-apply llength-lmap lnth-lmap*)

have 2: $(\text{ldropWhile } ((\text{Not} \circ P) \circ \text{fst}) (\text{lzip } xs (\text{iterates } \text{Suc } n))) =$
 $(\text{ldropn } (\text{LEAST } na. na < \text{llength}(\text{lzip } xs (\text{iterates } \text{Suc } n)) \wedge$
 $((\text{Not} \circ (\text{Not} \circ P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs (\text{iterates } \text{Suc } n)) \ na)) (\text{lzip } xs (\text{iterates } \text{Suc } n)))$

using 1 *ldropWhile-LEAST*[of $(\text{lzip } xs (\text{iterates } \text{Suc } n)) (\text{Not} \circ P) \circ \text{fst}$] **by** *auto*

have 3: $(\text{ldropn } (\text{LEAST } na. na < \text{llength}(\text{lzip } xs (\text{iterates } \text{Suc } n)) \wedge$
 $((\text{Not} \circ (\text{Not} \circ P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs (\text{iterates } \text{Suc } n)) \ na)) (\text{lzip } xs (\text{iterates } \text{Suc } n))) =$
 $(\text{ldropn } (\text{LEAST } na. na < \text{llength}(\text{lzip } xs (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs (\text{iterates } \text{Suc } n)) \ na)) (\text{lzip } xs (\text{iterates } \text{Suc } n)))$

by *auto*

have 4: $(\text{ldropn } (\text{LEAST } na. na < \text{llength}(\text{lzip } xs (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs (\text{iterates } \text{Suc } n)) \ na)) (\text{lzip } xs (\text{iterates } \text{Suc } n))) =$
 $(\text{lzip } (\text{ldropn } (\text{LEAST } na. na < \text{llength}(\text{lzip } xs (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs (\text{iterates } \text{Suc } n)) \ na)) \ xs)$
 $(\text{ldropn } (\text{LEAST } na. na < \text{llength}(\text{lzip } xs (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs (\text{iterates } \text{Suc } n)) \ na)) (\text{iterates } \text{Suc } n)))$

using *ldropn-lzip* **by** *blast*

have 5: $(\text{ldropn } (\text{LEAST } na. na < \text{llength}(\text{lzip } xs (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs (\text{iterates } \text{Suc } n)) \ na)) (\text{iterates } \text{Suc } n)) =$
 $(\text{iterates } \text{Suc } (n + (\text{LEAST } na. na < \text{llength}(\text{lzip } xs (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs (\text{iterates } \text{Suc } n)) \ na))))$

by (metis (no-types, lifting) *funpow-Suc-conv ldropn-iterates*)

have 6: $(\text{LEAST } na. na < \text{llength}(\text{lzip } xs (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs (\text{iterates } \text{Suc } n)) \ na)) =$
 $(\text{LEAST } na. na < \text{llength}(\text{lzip } xs (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{xs}) \ na, \text{lnth } (\text{iterates } \text{Suc } n) \ na))$

by (metis (no-types, opaque-lifting) *comp-def fst-conv lmap-fst-lzip lnth-lmap*)

have 7: $\text{llength}(\text{lzip } xs (\text{iterates } \text{Suc } n)) = \text{llength } xs$

by *simp*

have 8: $(\text{LEAST } na. na < \text{llength}(\text{lzip } xs (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{xs}) \ na, \text{lnth } (\text{iterates } \text{Suc } n) \ na)) =$
 $(\text{LEAST } na. na < \text{llength } xs \wedge P (\text{lnth } xs \ na))$

by *auto*

have 9: $(\text{lzip } (\text{ldropn } (\text{LEAST } na. na < \text{llength}(\text{lzip } xs (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs (\text{iterates } \text{Suc } n)) \ na)) \ xs)$
 $(\text{ldropn } (\text{LEAST } na. na < \text{llength}(\text{lzip } xs (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs (\text{iterates } \text{Suc } n)) \ na)) (\text{iterates } \text{Suc } n))) =$
 $(\text{lzip } (\text{ldropn } (\text{LEAST } na. na < \text{llength } xs \wedge P (\text{lnth } xs \ na)) \ xs)$
 $(\text{iterates } \text{Suc } (n + (\text{LEAST } na. na < \text{llength } xs \wedge P (\text{lnth } xs \ na))))$

```

using 5 6 by auto
show ?thesis unfolding lleast-def
  using assms 2 3 4 9 ldropWhile-LEAST-not[of xs P] by presburger
qed

lemma kfilter-ldropWhile :
  assumes  $\neg \text{lnull}(\text{kfilter } P \ n \ xs)$ 
  shows  $\text{kfilter } P \ n \ xs =$ 
    ( $\text{LCons } (n+ \ (\text{lleast } P \ xs))$ 
      ( $\text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst})$ 
        ( $\text{lzip } (\text{ldrop } (\text{Suc } (\text{lleast } P \ xs)) \ ( \ xs))$ 
          ( $\text{iterates } \text{Suc } (\text{Suc } (n+(\text{lleast } P \ xs))))))$ )))

proof -
  let ?Least = ( $\text{lleast } P \ xs$ )
  have 1:  $\text{lhd}(\text{lfilter } (P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n))) =$ 
     $\text{lhd}(\text{ldropWhile } (\text{Not } \circ P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n)))$ 
  by simp
  have 2:  $\text{ltl}(\text{lfilter } (P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n))) =$ 
    ( $\text{lfilter } (P \circ \text{fst}) \ (\text{ltl}(\text{ldropWhile } (\text{Not } \circ P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n))))$ )
  by (simp add: ltl-lfilter)
  have 3:  $\text{lfilter } (P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n)) =$ 
    ( $\text{LCons } (\text{lhd}(\text{lfilter } (P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n))))$ 
      ( $\text{ltl}(\text{lfilter } (P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n))))$ )
  by (metis assms kfilter-llength llength-eq-0 llength-lmap llist.disc(1) llist.exhaust-sel
    lmap-fst-lfilter lmap-fst-lzip)
  have 4:  $\text{kfilter } P \ n \ xs =$ 
     $\text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n)))$ 
  by (simp add: kfilter-def)
  have 5:  $\text{ltl}(\text{ldropWhile } (\text{Not } \circ P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n))) =$ 
    ( $\text{ltl}((\text{lzip } (\text{ldropWhile } (\text{Not } \circ P) \ xs)$ 
      ( $\text{lmap } \text{snd } (\text{ldropWhile } (\text{Not } \circ P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n))))))$ )
  by (metis lzip-ldropWhile-fst-iterates)
  have 6:  $\text{ltl}((\text{lzip } (\text{ldropWhile } (\text{Not } \circ P) \ xs)$ 
    ( $\text{lmap } \text{snd } (\text{ldropWhile } (\text{Not } \circ P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n)))))) =$ 
    ( $\text{lzip } (\text{ltl}(\text{ldropWhile } (\text{Not } \circ P) \ xs)$ 
      ( $\text{ltl}(\text{lmap } \text{snd } (\text{ldropWhile } (\text{Not } \circ P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n))))))$ )
  using lzip-ldropWhile-fst-iterates[of P xs n]
    ltl-lzip[of ( $\text{ldropWhile } (\text{Not } \circ P) \ xs$ )
      ( $\text{lmap } \text{snd } (\text{ldropWhile } (\text{Not } \circ P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n))))]$ 
  by force
  have 7:  $\text{lmap } \text{snd } ($ 
    ( $\text{LCons } (\text{lhd}(\text{ldropWhile } (\text{Not } \circ P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n))))$ 
      ( $\text{lfilter } (P \circ \text{fst}) \ (\text{ltl}(\text{ldropWhile } (\text{Not } \circ P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n)))))) =$ 
    ( $\text{LCons } (\text{snd}(\text{lhd}(\text{ldropWhile } (\text{Not } \circ P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n))))$ 
      ( $\text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) \ (\text{ltl}(\text{ldropWhile } (\text{Not } \circ P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n))))))$ )
  by simp
  have 8:  $\exists n. \text{enat } n < \text{llength } xs \wedge P \ (\text{lnth } xs \ n)$ 
  by (metis assms in-lset-conv-lnth kfilter-llength llength-eq-0 lnull-lfilter)
  have 9: ( $\text{snd}(\text{lhd}(\text{ldropWhile } (\text{Not } \circ P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } n)))) =$ 
    ( $\text{snd}(\text{lhd}(\text{lzip } (\text{ldropWhile } (\text{Not } \circ P) \ xs)$ 

```

```

      (iterates Suc (n+?Least)) )))
  using 8 ldropsWhile-iterates-split[of xs P n] by simp
have 10: (snd (lhd (lzip (ldropsWhile (Not ∘ P) xs)
      (iterates Suc (n+?Least)) ))) =
      lhd (iterates Suc (n+?Least))
  by (metis (no-types, lifting) 3 comp-assoc iterates.disc-iff lfilter-eq-LCons lhd-lzip
      llist.disc(2) lzip.disc(1) lzip-ldropsWhile-fst-iterates snd-conv)
have 11: lhd (iterates Suc (n+?Least)) = (n+?Least)
  by simp
have 12: ltl (ldropsWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))) =
      ltl (lzip (ldropsWhile (Not ∘ P) xs)
      (iterates Suc (n+?Least)))
  using 8 ldropsWhile-iterates-split[of xs P n] by simp
have 13: ltl (lzip (ldropsWhile (Not ∘ P) xs) (iterates Suc (n+?Least))) =
      (lzip (ltl (ldropsWhile (Not ∘ P) xs)) (ltl (iterates Suc (n+?Least))))
  by (metis (no-types, lifting) 3 comp-assoc iterates.disc-iff lfilter-eq-LCons llist.discI(2)
      ltl-lzip lzip.disc-iff(2) lzip-ldropsWhile-fst-iterates)
have 14: (ltl (iterates Suc (n+?Least))) = (iterates Suc (Suc (n+?Least)))
  by simp
have 15: ltl (ldropsWhile (Not ∘ P) xs) = ltl (ldrop (llength (ltakeWhile (Not ∘ P) xs)) xs)
  by (simp add: ldropsWhile-eq-ldrop)
have 16: ltl (ldrop (llength (ltakeWhile (Not ∘ P) xs)) xs) =
      ldrops (eSuc (llength (ltakeWhile (Not ∘ P) xs))) (xs)
  by (simp add: ldrops-eSuc-conv-ltl)
have 17: (llength (ltakeWhile (Not ∘ P) xs)) = (llength (ltake ?Least xs))
  using 8 ltakeWhile-LEAST[of xs Not ∘ P] unfolding lleast-def by auto
have 18: (llength (ltake ?Least xs)) = enat ?Least
  using 17 8 llength-ltakeWhile-LEAST-not unfolding lleast-def by fastforce
have 19: ldrops (eSuc (llength (ltakeWhile (Not ∘ P) xs))) (xs) = ldrops (Suc ?Least) (xs)
  using 17 18 by (simp add: eSuc-enat)
have 20: ltl (ldropsWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))) =
      (lzip (ldrops (Suc ?Least) (xs)) (iterates Suc (Suc (n+?Least))))
  using 12 13 15 16 19 by auto
show ?thesis
using 2 3 4 10 20 9 by auto
qed

```

lemma *kfilter-eq-LCons*:

```

kfilter P n xs = LCons x xs' ⟹
  x = n+(lleast P xs) ∧
  xs' = (lmap snd (lfilter (P ∘ fst)
      (lzip (ldrops (Suc (lleast P xs)) (xs))
      (iterates Suc (Suc (n+(lleast P xs)))))))

```

using *kfilter-ldropsWhile*[of P n xs] **by** auto

lemma *kfilter-eq-LCons-1*:

```

kfilter P n xs = LCons x xs' ⟹
  x = (n+(lleast P xs)) ∧
  xs' = kfilter P (Suc (n+(lleast P xs))) (ldrops (Suc (lleast P xs)) xs)

```

using *kfilter-eq-LCons*[of P n xs x xs']

```

      kfilter-def[of P (Suc (n + (lleast P xs)))
        (ldrop (Suc (lleast P xs)) xs)]
by auto

lemma kfilter-eq-conv:
  kfilter P n xs = LNil ∨
    kfilter P n xs =
      LCons (n+(lleast P xs)) (kfilter P (Suc (n+(lleast P xs))) (ldrop (Suc (lleast P xs)) xs))
proof (cases kfilter P n xs)
case LNil
then show ?thesis by simp
next
case (LCons x21 x22)
then show ?thesis
  proof -
    let ?Least = (lleast P xs)
    have 1: x21 = n+?Least
      using kfilter-eq-LCons-1[of P n xs x21 x22] by (meson LCons)
    have 2: x22 = (kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs))
      using kfilter-eq-LCons-1[of P n xs x21 x22] by (meson LCons)
    show ?thesis
      by (metis 1 2 LCons)
  qed
qed

lemma kfilter-lnth-zero:
  assumes ¬lnull(kfilter P n xs)
  shows lnth (kfilter P n xs) 0 = n+ (lleast P xs)
using assms
by (metis kfilter-eq-LCons-1 lhd-LCons-ltl lnth-0-conv-lhd)

lemma length-LEAST-a:
  assumes ¬lnull(kfilter P n xs)
  shows lleast P xs < llength xs
using assms exist-lset-lnth[of xs P] kfilter-not-lnull-conv[of P n xs]
length-LEAST[of xs P] by blast

lemma kfilter-upperbound:
  assumes i < llength(kfilter P n xs)
  shows (lnth (kfilter P n xs) i) < n + llength xs
proof (cases lfinite (kfilter P n xs))
case True
then show ?thesis using assms
  proof (induct zs≡kfilter P n xs arbitrary: xs n i rule: lfinite-induct)
    case (LNil xs)
    then show ?case
      using gr-implies-not-zero llength-lnull by blast
    next
    case (LCons xs)
    then show ?case

```

```

proof -
  let ?Least = (lleast P xs)
  have 1:  $\exists x \, xs'. \, kfilter \, P \, n \, xs = LCons \, x \, xs'$ 
    using  $LCons.hyps(2) \, kfilter\text{-}ldropWhile$  by blast
  obtain  $x \, xs'$  where 2:  $kfilter \, P \, n \, xs = LCons \, x \, xs'$ 
    using 1 by auto
  have 3:  $x = n + ?Least$ 
    using  $kfilter\text{-}ldropWhile[of \, P \, n \, xs]$  by (simp add: 2)
  have 4:  $i=0 \implies (lnth \, (kfilter \, P \, n \, xs) \, i) = x$ 
    by (simp add: 2)
  have 5:  $enat \, n + enat \, ?Least < enat \, n + llength \, xs$ 
    using  $length\text{-}LEAST\text{-}a[of \, P \, n \, xs] \, LCons.hyps(2) \, enat\text{-}add\text{-}mono$  by blast
  have 6:  $i=0 \implies enat \, (lnth \, (kfilter \, P \, n \, xs) \, i) < enat \, n + llength \, xs$ 
    using 3 4 5 by auto
  have 7:  $xs' = kfilter \, P \, (Suc \, (n + ?Least)) \, (ldrop \, (Suc \, ?Least) \, xs)$ 
    using  $kfilter\text{-}ldropWhile[of \, P \, n \, xs] \, 2 \, kfilter\text{-}eq\text{-}LCons\text{-}1$  by blast
  have 8:  $i>0 \implies enat \, (lnth \, (kfilter \, P \, n \, xs) \, i) = enat \, (lnth \, xs' \, (i-1))$ 
    by (simp add: 2  $lnth\text{-}LCons'$ )
  have 9:  $i>0 \wedge lnull \, xs' \implies enat \, (lnth \, (kfilter \, P \, n \, xs) \, i) < enat \, n + llength \, xs$ 
    by (metis 2  $LCons.prem(1) \, One\text{-}nat\text{-}def \, enat\text{-}ord\text{-}sims(2) \, llength\text{-}LCons \, llength\text{-}LNil \, llist.collapse(1) \, not\text{-}less\text{-}eq \, one\text{-}eSuc \, one\text{-}enat\text{-}def$ )
  have 10:  $i>0 \wedge \neg lnull \, xs' \implies (i-1) < llength \, xs'$ 
    using 2  $LCons.prem(1) \, Suc\text{-}ile\text{-}eq$  by fastforce
  have 11:  $i>0 \wedge \neg lnull \, xs' \implies$ 
     $enat \, (lnth \, xs' \, (i-1)) < enat \, (Suc \, (n + ?Least)) + llength \, (ldrop \, (Suc \, ?Least) \, xs)$ 
    by (metis (no-types, lifting) 10 2 7  $LCons.hyps(3) \, ltl\text{-}sims(2)$ )
  have 111:  $lappend \, (ltake \, (Suc \, (?Least)) \, xs) \, (ldrop \, (Suc \, (?Least)) \, xs) = xs$ 
    using  $lappend\text{-}ltake\text{-}ldrop$  by blast
  have 112:  $enat \, (Suc \, (n + ?Least)) = n + (Suc \, ?Least)$ 
    by auto
  have 113:  $Suc \, ?Least \leq (llength \, xs)$ 
    using 5  $Suc\text{-}ile\text{-}eq \, enat\text{-}add\text{-}mono$  by blast
  have 114:  $llength \, (ltake \, (Suc \, (?Least)) \, xs) = Suc \, ?Least$ 
    using  $llength\text{-}ltake[of \, (Suc \, (?Least)) \, xs] \, 113$  by linarith
  have 12:  $enat \, (Suc \, (n + ?Least)) + llength \, (ldrop \, (Suc \, ?Least) \, xs) \leq enat \, n + llength \, xs$ 
    using  $llength\text{-}lappend[of \, (ltake \, (Suc \, (?Least)) \, xs) \, (ldrop \, (Suc \, (?Least)) \, xs)]$ 
    by (metis (no-types, lifting) 111 112 114  $eq\text{-}refl \, group\text{-}cancel.add1 \, plus\text{-}enat\text{-}sims(1)$ )
  have 14:  $i>0 \wedge \neg lnull \, xs' \implies enat \, (lnth \, (kfilter \, P \, n \, xs) \, i) < enat \, n + llength \, xs$ 
    using 11 12 8 by auto
  have 15:  $i>0 \implies enat \, (lnth \, (kfilter \, P \, n \, xs) \, i) < enat \, n + llength \, xs$ 
    using 14 9  $dual\text{-}order.strict\text{-}implies\text{-}order$  by blast
  show ?thesis
    using 15 6 by blast
qed
qed
next
case False
then show ?thesis
by (metis  $enat\text{-}ord\text{-}sims(4) \, iadd\text{-}le\text{-}enat\text{-}iff \, kfilter\text{-}llength \, leD \, leI \, llength\text{-}eq\text{-}enat\text{-}lfiniteD \, llength\text{-}eq\text{-}infty\text{-}conv\text{-}lfinite \, llength\text{-}lfilter\text{-}ile$ )

```

qed

lemma *kfilter-lowerbound*:

assumes $i < \text{length}(\text{kfilter } P \ n \ xs)$

shows $n \leq (\text{lnth } (\text{kfilter } P \ n \ xs) \ i)$

using *assms*

proof (*induct i arbitrary: xs n*)

case 0

then show ?case

using *kfilter-ldropWhile zero-enat-def* **by** *fastforce*

next

case (*Suc i*)

then show ?case

proof –

let ?Least = *least P xs*

have 7: $\exists \ x \ xs'. (\text{kfilter } P \ n \ xs) = \text{LCons } x \ xs'$

using *Suc.premis gr-implies-not-zero kfilter-ldropWhile length-lnull* **by** *blast*

obtain $x \ xs'$ **where** 8: $(\text{kfilter } P \ n \ xs) = \text{LCons } x \ xs'$

using 7 **by** *auto*

have 9: $\text{lnth } (\text{kfilter } P \ n \ xs) \ (\text{Suc } i) = \text{lnth } xs' \ i$

by (*simp add: 8*)

have 10: $xs' = \text{kfilter } P \ (\text{Suc } (n + ?\text{Least})) \ (\text{ldrop } (\text{Suc } ?\text{Least}) \ xs)$

using *kfilter-ldropWhile[of P n xs] 8 kfilter-eq-LCons-1* **by** *blast*

have 11: $\text{enat } i < \text{length} (\text{kfilter } P \ (\text{Suc } (n + ?\text{Least})) \ (\text{ldrop } (\text{Suc } ?\text{Least}) \ xs))$

by (*metis 10 8 Extended-Nat.eSuc-mono Suc.premis(1) eSuc-enat length-LCons*)

have 12: $\text{lnth } xs' \implies n \leq \text{lnth } (\text{kfilter } P \ n \ xs) \ (\text{Suc } i)$

using 10 11 *gr-implies-not-zero length-eq-0* **by** *blast*

have 13: $\neg \text{lnth } xs' \implies (\text{Suc } (n + ?\text{Least})) \leq \text{lnth } xs' \ i$

using 10 11 *Suc.hyps* **by** *blast*

have 14: $\neg \text{lnth } xs' \implies n \leq \text{lnth } (\text{kfilter } P \ n \ xs) \ (\text{Suc } i)$

using 13 9 **by** *linarith*

show ?thesis **using** 12 14 **by** *blast*

qed

qed

lemma *kfilter-mono*:

assumes $(\text{Suc } i) < \text{length}(\text{kfilter } P \ n \ xs)$

shows $(\text{lnth } (\text{kfilter } P \ n \ xs) \ i) < (\text{lnth } (\text{kfilter } P \ n \ xs) \ (\text{Suc } i))$

using *assms*

proof (*induct i arbitrary: xs n*)

case 0

then show ?case

proof –

let ?Least = *least P xs*

have 1: $\exists \ x \ xs'. \text{kfilter } P \ n \ xs = \text{LCons } x \ xs'$

using 0.premis *gr-implies-not-zero kfilter-ldropWhile length-lnull* **by** *blast*

obtain $x \ xs'$ **where** 2: $\text{kfilter } P \ n \ xs = \text{LCons } x \ xs'$

using 1 **by** *auto*

have 3: $x = n + ?\text{Least}$

using *kfilter-ldropWhile[of P n xs]* **by** (*simp add: 2*)

```

have 4: lnth (kfilter P n xs) 0 = n + ?Least
  by (simp add: 2 3)
have 5: lnth (kfilter P n xs) (Suc 0) = lnth xs' 0
  by (simp add: 2)
have 6: xs' = kfilter P (Suc (n + ?Least)) (ldrop (Suc ?Least) xs)
  using kfilter-ldropWhile[of P n xs] 2 kfilter-eq-LCons-1 by blast
have 7: n + ?Least < lnth xs' 0
  by (metis 0.prem1 2 6 Extended-Nat.eSuc-mono One-nat-def Suc-le-lessD kfilter-lowerbound
    llength-LCons one-eSuc one-enat-def zero-enat-def)
show ?thesis by (simp add: 4 5 7)
qed
next
case (Suc i)
then show ?case
proof -
  let ?Least = lleast P xs
  have 8:  $\exists x xs'. kfilter P n xs = LCons x xs'$ 
    using Suc.prem1 gr-implies-not-zero kfilter-ldropWhile llength-lnull by blast
  obtain x xs' where 9:  $kfilter P n xs = LCons x xs'$ 
    using 8 by auto
  have 10: xs' = kfilter P (Suc (n + ?Least)) (ldrop (Suc ?Least) xs)
    using kfilter-ldropWhile[of P n xs] 9 kfilter-eq-LCons-1 by blast
  have 11: lnth (kfilter P n xs) (Suc (Suc i)) = lnth xs' (Suc i)
    by (simp add: 9)
  have 12: lnth (kfilter P n xs) (Suc i) < lnth xs' (Suc i)
    by (metis (no-types, lifting) 10 9 Extended-Nat.eSuc-mono Suc(1) Suc(2) eSuc-enat
      llength-LCons lnth-Suc-LCons)
  show ?thesis by (simp add: 11 12)
qed
qed

lemma lfilter-kfilter:
  assumes i < llength(kfilter P n xs)
  shows (lnth xs ((lnth (kfilter P n xs) i) - n)) = (lnth (lfilter P xs) i)
  using assms
  proof (induct i arbitrary: xs n)
  case 0
  then show ?case
  proof -
    let ?Least = lleast P xs
    have 1: lnth (kfilter P n xs) 0 = n + ?Least
      using 0.prem1 kfilter-ldropWhile zero-enat-def by fastforce
    have 2: (lnth xs ((lnth (kfilter P n xs) 0) - n)) =
      (lnth xs ?Least)
      by (simp add: 1)
    have 3: (lnth (lfilter P xs) 0) = lhd (ldropWhile (Not o P) xs)
      by (metis (full-types) 0.prem1 kfilter-llength lhd-conv-lnth lhd-lfilter llength-eq-0 not-iless0)
    have 4:  $\exists n < llength xs. P (lnth xs n)$ 
      by (metis 0.prem1 dual-order.irrefl in-lset-conv-lnth kfilter-llength llength-eq-0 lnull-lfilter
        zero-enat-def)

```

```

have 5: (ldropWhile (Not ∘ P) xs) = ldropn ?Least xs
  using ldropWhile-LEAST-not[of xs P ] 4 unfolding lleast-def by blast
have 6: lhd (ldropn ?Least xs) = (lnth xs ?Least)
  by (simp add: 4 lhd-ldropn llength-LEAST)
show ?thesis using 2 3 5 6 by auto
qed
next
case (Suc i)
then show ?case
proof -
let ?Least = lleast P xs
have 7: ∃ x xs'. (kfilter P n xs) = LCons x xs'
  using Suc.premis gr-implies-not-zero kfilter-ldropWhile llength-lnull by blast
obtain x xs' where 8: (kfilter P n xs) = LCons x xs'
  using 7 by auto
have 9: lnth (kfilter P n xs) (Suc i) = lnth xs' i
  by (simp add: 8)
have 10: xs' = kfilter P (Suc (n + ?Least)) (ldrop (Suc ?Least) xs)
  using kfilter-ldropWhile[of P n xs] by (simp add: 8 kfilter-eq-LCons-1)
have 11: enat i < llength (kfilter P (Suc (n + ?Least)) (ldrop (Suc ?Least) xs))
  by (metis 10 8 Extended-Nat.eSuc-mono Suc.premis(1) eSuc-enat llength-LCons)
have 12: lnull xs' ⇒
  lnth xs (lnth (kfilter P n xs) (Suc i) - n) = lnth (lfilter P xs) (Suc i)
  by (metis 10 11 llength-LNil llist.collapse(1) not-less-zero)
have 13: ¬lnull xs' ⇒
  lnth (ldrop (Suc ?Least) xs) (lnth xs' i - (Suc (n + ?Least))) =
  lnth (lfilter P (ldrop (Suc ?Least) xs)) i
  by (metis (no-types, lifting) 10 11 Suc.hyps)
have 14: (lnth (lfilter P xs) (Suc i)) = (lnth (ltl(lfilter P xs)) i)
  by (metis 8 kfilter-not-llnull-conv llist.disc(2) lnth-ltl lnull-lfilter)
have 15: (ltl(lfilter P xs)) = lfilter P (ltl (ldropWhile (Not ∘ P) xs))
  using ltl-lfilter by blast
have 16: (ltl (ldropWhile (Not ∘ P) xs)) =
  (ltl (ldropn (LEAST n. n < llength xs ∧ (Not ∘ (Not ∘ P)) (lnth xs n) ) xs))
  using ldropWhile-LEAST[of xs Not ∘ P]
  by (metis (no-types, lifting) 8 comp-apply in-lset-conv-lnth kfilter-llnull-conv llist.disc(2))
have 17: (ltl (ldropn (LEAST n. n < llength xs ∧ (Not ∘ (Not ∘ P)) (lnth xs n) ) xs)) =
  (ltl (ldropn ?Least xs))
  unfolding lleast-def by auto
have 18: (ltl (ldropn ?Least xs)) =
  ldropn (Suc ?Least) xs
  by (simp add: ldropn-ltl ltl-ldropn)
have 19: ldropn (Suc ?Least) xs =
  ldrop (Suc ?Least) xs
  by (simp add: ldrop-enat)
have 20: lnth (lfilter P xs) (Suc i) = lnth (lfilter P (ldrop (Suc ?Least) xs)) i
  using 14 15 16 18 19 unfolding lleast-def by auto
have 21: lnth xs (lnth (kfilter P n xs) (Suc i) - n) = lnth xs (lnth xs' i - n)
  by (simp add: 10 9)
have 22: n ≤ lnth xs' i

```



```

using 9 Suc.premis(1) kfilter-lowerbound by fastforce
have 23:  $(\text{Suc } (n + ?\text{Least})) \leq \text{lnth } (k\text{filter } P \ (\text{Suc } (n + ?\text{Least})) \ (\text{ldrop } (\text{Suc } ?\text{Least}) \ xs)) \ i$ 
using kfilter-lowerbound[of i P Suc (n+?Least) (ldrop (Suc ?Least) xs) ] 11 by force
have 24:  $(\text{Suc } ?\text{Least}) > \text{llength } (\text{ltake } ?\text{Least } xs)$ 
by (simp add: min.strict-coboundedI1)
have 25:  $(\text{ldrop } (\text{Suc } ?\text{Least}) \ (\text{lappend } (\text{ltake } ?\text{Least } xs) \ (\text{ldrop } ?\text{Least } xs))) =$ 
 $\text{ldrop } ((\text{Suc } ?\text{Least}) - \text{llength}(\text{ltake } ?\text{Least } xs) \ ) \ (\text{ldrop } ?\text{Least } xs)$ 
by (simp add: ldrop-lappend)
have 26:  $\text{lappend } (\text{ltake } (\text{Suc } ?\text{Least}) \ xs) \ (\text{ldrop } (\text{Suc } ?\text{Least}) \ xs) = xs$ 
by (simp add: lappend-ltake-ldrop)
have 27:  $(\text{Suc } ?\text{Least}) \leq (\text{lnth } xs' \ i - n)$ 
using 10 23 by auto
have 271:  $\text{lnth } xs' \ i - n - (\text{Suc } ?\text{Least}) = \text{lnth } xs' \ i - (\text{Suc } (n + ?\text{Least}))$ 
by auto
have 28:  $\text{lnth } (\text{lappend } (\text{ltake } (\text{Suc } ?\text{Least}) \ xs) \ (\text{ldrop } (\text{Suc } ?\text{Least}) \ xs)) \ (\text{lnth } xs' \ i - n) =$ 
 $\text{lnth } (\text{ldrop } (\text{Suc } ?\text{Least}) \ xs) \ (\text{lnth } xs' \ i - (\text{Suc } (n + ?\text{Least})))$ 
using lnth-lappend[of  $(\text{ltake } (\text{Suc } ?\text{Least}) \ xs) \ (\text{ldrop } (\text{Suc } ?\text{Least}) \ xs) \ (\text{lnth } xs' \ i - n)$ ]
27 271 11 19
by (metis (no-types, lifting) gr-implies-not-zero kfilter-LNil ldropn-eq-LNil
le-cases llength-lnull llength-ltake llist.disc(1) lnth-lappend2 min-def)
have 29:  $\text{lnth } xs \ (\text{lnth } xs' \ i - n) =$ 
 $\text{lnth } (\text{ldrop } (\text{Suc } ?\text{Least}) \ xs) \ (\text{lnth } xs' \ i - (\text{Suc } (n + ?\text{Least})))$ 
using 28 by (metis (no-types, lifting) 26)
have 30:  $\neg \text{lnull } xs' \implies$ 
 $\text{lnth } xs \ (\text{lnth } (k\text{filter } P \ n \ xs) \ (\text{Suc } i) - n) = \text{lnth } (l\text{filter } P \ xs) \ (\text{Suc } i)$ 
by (metis 13 20 21 29)
show ?thesis
using 12 30 by blast
qed
qed

```

lemma *in-kfilter-lset*:

shows $x \in \text{lset } (k\text{filter } P \ n \ xs) \longleftrightarrow x \in \{ n + i \mid i. i < \text{llength } xs \wedge P \ (\text{lnth } xs \ i) \}$
(is *?lhs = ?rhs***)**

proof

assume *?lhs*

thus *?rhs*

proof (*induct zs \equiv kfilter P n xs arbitrary: n xs rule: llist-set-induct*)

case (*find*)

then show *?case*

proof –

let *?Least = lleast P xs*

have 1: $\text{lhd } (k\text{filter } P \ n \ xs) = n + ?\text{Least}$

using *kfilter-ldropWhile*[of *P n xs*] **find** **by** *auto*

have 2: $P \ (\text{lnth } xs \ ?\text{Least})$ **unfolding** *lleast-def*

by (*metis (mono-tags, lifting) LeastI exist-lset-lnth find.hyps kfilter-lnull-conv*)

have 3: $?\text{Least} < \text{llength } xs$

by (*metis find.hyps length-LEAST-a*)

have 4: $n + ?\text{Least} \in \{ n + i \mid i. \text{enat } i < \text{llength } xs \wedge P \ (\text{lnth } xs \ i) \}$

using 2 3 **by** *blast*

```

  show ?thesis using 1 4 by auto
qed
next
case (step y)
then show ?case
proof -
  let ?Least = lleast P xs
  have 5: ltl (kfilter P n xs) = kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs)
    using kfilter-ldropWhile[of P n xs]
    by (metis kfilter-def ltl-simps(2) step.hyps(1))
  have 6: lnull (kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs))  $\implies$ 
     $y \in \{n + i \mid \text{enat } i < \text{llength } xs \wedge P (\text{lnth } xs \ i)\}$ 
    by (metis 5 gr-implies-not-zero in-lset-conv-lnth llength-eq-0 step.hyps(2))
  have 7:  $\neg$  lnull (kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs))  $\implies$ 
     $y \in \{(Suc (n+?Least)) + i \mid \text{enat } i < \text{llength } (\text{ldrop } (Suc ?Least) \ xs) \wedge$ 
       $P (\text{lnth } (\text{ldrop } (Suc ?Least) \ xs) \ i)\}$ 
    using step.hyps using 5 by blast
  have 8: (Suc (n+?Least)) = n + Suc ?Least
    by auto
  have 9:  $\neg$  lnull(kfilter P (Suc (n + ?Least)) (ldrop (enat (Suc ?Least)) xs))  $\implies$ 
     $\exists i. y = n + Suc (?Least) + i \wedge \text{enat } i < \text{llength } (\text{ldrop } (\text{enat } (Suc ?Least)) \ xs) \wedge$ 
       $P (\text{lnth } (\text{ldrop } (\text{enat } (Suc ?Least)) \ xs) \ i) \implies$ 
       $\exists i. y = n + i \wedge \text{enat } i < \text{llength } xs \wedge P (\text{lnth } xs \ i)$ 
  proof -
    assume a0:  $\neg$  lnull(kfilter P (Suc (n + ?Least)) (ldrop (enat (Suc ?Least)) xs))
    assume a1:  $\exists i. y = n + Suc (?Least) + i \wedge \text{enat } i < \text{llength } (\text{ldrop } (\text{enat } (Suc ?Least)) \ xs) \wedge$ 
       $P (\text{lnth } (\text{ldrop } (\text{enat } (Suc ?Least)) \ xs) \ i)$ 
    show  $\exists i. y = n + i \wedge \text{enat } i < \text{llength } xs \wedge P (\text{lnth } xs \ i)$ 
    proof -
      obtain i where 10:  $y = n + Suc (?Least) + i \wedge \text{enat } i < \text{llength } (\text{ldrop } (\text{enat } (Suc ?Least)) \ xs) \wedge$ 
         $P (\text{lnth } (\text{ldrop } (\text{enat } (Suc ?Least)) \ xs) \ i)$ 
      using a1 by auto
      have 11:  $Suc (?Least) + i < \text{llength } xs$ 
        by (metis 10 add commute ldrop-enat ldrop-ldrop leD le-less-linear lnull-ldropn
          plus-enat-simps(1))
      have 12:  $P (\text{lnth } xs \ (Suc (?Least) + i))$ 
        by (metis 10 11 add commute ldrop-enat lnth-ldropn)
      show ?thesis
        using 10 11 12 ab-semigroup-add-class.add-ac(1) by blast
    qed
  qed
  have 13:  $\neg$  lnull (kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs))  $\implies$ 
     $y \in \{n + i \mid \text{enat } i < \text{llength } xs \wedge P (\text{lnth } xs \ i)\}$ 
    using 7 9 by auto
  show ?thesis
    using 13 6 by blast
qed
qed
next
assume ?rhs

```

```

then obtain  $i$  where 15:  $x = n+i \wedge \text{enat } i < \text{llength } xs \wedge P (\text{lnth } xs \ i)$ 
  by blast
thus ?lhs
proof (induct  $i$  arbitrary:  $xs \ n$ )
case 0
then show ?case
  proof -
    have 16:  $\text{lhd } (kfilter \ P \ n \ xs) = n+(\text{lleast } P \ xs)$ 
      using  $kfilter\text{-ldropWhile}[of \ P \ n \ xs]$  using 0.premis
    by (metis  $\text{in-lset-conv-lnth } kfilter\text{-lnull-conv } \text{lhd-LCons}$ )
    show ?thesis
      by (metis 0.premis  $\text{add.right-neutral } kfilter\text{-LCons } \text{ldropn-0 } \text{ldropn-Suc-conv-ldropn}$ 
         $\text{llist.set-intros}(1))$ 
  qed
next
case (Suc  $i$ )
then show ?case
  proof -
    have 18:  $\text{lnull } xs \implies x \in \text{lset } (kfilter \ P \ n \ xs)$ 
      using  $\text{Suc}(2)$  by auto
    have 19:  $\neg \text{lnull } xs \implies P (\text{lnth } xs \ (\text{Suc } i)) = P (\text{lnth } (\text{ltl } xs) \ (i))$ 
      by (simp add:  $\text{lnth-ltl}$ )
    have 20:  $\neg \text{lnull } xs \implies x = (\text{Suc } n) + i \wedge \text{enat } i < \text{llength } (\text{ltl } xs) \wedge P (\text{lnth } (\text{ltl } xs) \ (i))$ 
      by (metis 19  $\text{Extended-Nat.eSuc-mono } \text{Suc.premis}(1) \text{ add-Suc-shift } \text{eSuc-enat } \text{lhd-LCons-ltl}$ 
         $\text{llength-LCons}$ )
    have 21:  $\neg \text{lnull } xs \implies \text{lnull } (kfilter \ P \ n \ (\text{ltl } xs)) \implies x \in \text{lset } (kfilter \ P \ n \ xs)$ 
      by (metis 20  $\text{in-lset-conv-lnth } kfilter\text{-lnull-conv}$ )
    have 22:  $\neg \text{lnull } xs \implies \neg \text{lnull } (kfilter \ P \ n \ (\text{ltl } xs)) \implies x \in \text{lset } (kfilter \ P \ (\text{Suc } n) \ (\text{ltl } xs))$ 
      by (metis 20  $\text{Suc.hyps}$ )
    have 23:  $\neg \text{lnull } xs \implies x \in \text{lset } (kfilter \ P \ n \ xs)$ 
      using  $kfilter\text{-LCons}[of \ P \ n \ \text{lhd } xs \ \text{ltl } xs]$ 
         $\text{in-lset-ltlD } \text{lhd-LCons-ltl}[of \ (kfilter \ P \ n \ (\text{ltl } xs))]$ 
      using 21 22 by fastforce
    show ?thesis
      using 18 23 by blast
  qed
qed
qed

```

lemma $kfilter\text{-lset}$:

shows $\text{lset } (kfilter \ P \ n \ xs) = \{ n+i \mid i. i < \text{llength } xs \wedge P (\text{lnth } xs \ i) \}$
using in-kfilter-lset **by** blast

lemma $lfilter\text{-lnth-exist}$:

assumes

$i < \text{llength } (lfilter \ P \ xs)$

shows $(\exists k < \text{llength } xs. \text{lnth } (lfilter \ P \ xs) \ i = \text{lnth } xs \ k)$

using $\text{assms } \text{lset-lfilter}[of \ P \ xs]$

by (metis (no-types, opaque-lifting) $\text{add.left-neutral } \text{diff-zero } kfilter\text{-llength } kfilter\text{-upperbound}$
 $lfilter\text{-kfilter } \text{zero-enat-def}$)

```

lemma ldistinct-kfilter:
  ldistinct(kfilter P n xs)
proof (coinduction arbitrary: n xs)
case (ldistinct n1 xs1)
then show ?case
  proof –
    have 1: lhd (kfilter P n1 xs1)  $\notin$  lset (ltl (kfilter P n1 xs1))
    proof –
      have f1: kfilter P n1 xs1 =
        LCons (n1 + lleast P xs1)
          (lmap snd
            (lfilter (P  $\circ$  fst)
              (lzip
                (ldrop (enat (Suc (lleast P xs1))) xs1)
                (iterates Suc (Suc (n1 + lleast P xs1)))))))
      by (meson kfilter-ldropWhile ldistinct)
    then have lmap snd
      (lfilter (P  $\circ$  fst)
        (lzip (ldrop (enat (Suc (lleast P xs1))) xs1)
          (iterates Suc (Suc (n1 + lleast P xs1)))))) =
      kfilter P (Suc (n1 + lleast P xs1)) (ldrop (enat (Suc (lleast P xs1))) xs1)
    using kfilter-eq-LCons-1 by blast
    then show ?thesis
    using f1 by (metis (no-types) in-lset-conv-lnth kfilter-lowerbound leD lessI lhd-LCons ltl-simps(2))
  qed
have 2: ( $\exists$  n xs. ltl (kfilter P n1 xs1) = kfilter P n xs)  $\vee$  ldistinct (ltl (kfilter P n1 xs1))
  by (metis kfilter-eq-LCons-1 ldistinct llist.discI(1) llist.exhaust-sel)
show ?thesis
  using 1 2 by auto
qed
qed

```

```

lemma kfilter-llength-ltake:
  llength(kfilter P n (ltake k xs))  $\leq$  llength(kfilter P n xs)
by (simp add: kfilter-llength lprefix-lfilterI lprefix-llength-le)

```

```

lemma kfilter-ldropn-lset:
  assumes k < llength xs
  shows lset(kfilter P n (ldropn k xs)) =
    { n + i | i. i < llength xs - k  $\wedge$  P (lnth xs (k + i)) }
using assms
kfilter-lset[of P n (ldropn k xs) ]
by auto
(metis (no-types, lifting) add commute enat-min lnth-ldropn order.strict-implies-order
  plus-enat-simps(1),
metis (no-types, lifting) add commute dual-order.strict-implies-order enat-min lnth-ldropn
  plus-enat-simps(1))

```

lemma *kfilter-ldropn-lset-a*:
assumes $k < \text{length } xs$
shows $\text{lset}(\text{kfilter } P \ n \ (\text{ldropn } k \ xs)) =$
 $\{ n+(i-k) \mid i. k \leq i \wedge i < \text{length } xs \wedge P \ (\text{lnth } xs \ i) \}$
proof –
have 1: $\bigwedge x. x \in \{ n+i \mid i. i < \text{length } xs - k \wedge P \ (\text{lnth } xs \ (k+i)) \} \longleftrightarrow$
 $x \in \{ n+(i-k) \mid i. k \leq i \wedge i < \text{length } xs \wedge P \ (\text{lnth } xs \ i) \}$
proof *auto*
show $\bigwedge i. \text{enat } i < \text{length } xs - \text{enat } k \implies$
 $P \ (\text{lnth } xs \ (k + i)) \implies$
 $\exists ia. i = ia - k \wedge k \leq ia \wedge \text{enat } ia < \text{length } xs \wedge P \ (\text{lnth } xs \ ia)$
using *assms*
by (*metis add.commute add-diff-cancel-left' enat-min le-add1 less-imp-le plus-enat-simps(1)*)
show $\bigwedge i. k \leq i \implies$
 $\text{enat } i < \text{length } xs \implies$
 $P \ (\text{lnth } xs \ i) \implies$
 $\exists ia. n + i - k = n + ia \wedge \text{enat } ia < \text{length } xs - \text{enat } k \wedge P \ (\text{lnth } xs \ (k + ia))$
using *assms ldropn-Suc-conv-ldropn[of - xs] ldropn-eq-LConsD[of - ldropn k xs]*
 $\text{ldropn-ldropn}[of - - xs]$
by (*metis Nat.add-diff-assoc le-add-diff-inverse le-add-diff-inverse2 length-ldropn*)
qed
show ?thesis **using** *assms 1 kfilter-ldropn-lset[of k xs P n]* **by** *auto*
qed

lemma *kfilter-ldropn-lset-b*:
assumes $k < \text{length } xs$
shows $\text{lset}(\text{kfilter } P \ n \ (\text{ldropn } k \ xs)) =$
 $\{ n+i \mid i. i < \text{length } xs - k \wedge P \ (\text{lnth } xs \ (i+k)) \}$
proof –
have 1: $\bigwedge x. x \in \{ n+i \mid i. i < \text{length } xs - k \wedge P \ (\text{lnth } xs \ (i+k)) \} \longleftrightarrow$
 $x \in \{ n+i \mid i. i < \text{length } xs - k \wedge P \ (\text{lnth } xs \ (i+k)) \}$
by (*auto simp add: add.commute*)
show ?thesis **using** *assms 1 kfilter-ldropn-lset[of k xs P n]* **by** *auto*
qed

lemma *kfilter-length-n-zero*:
shows $\text{length}(\text{kfilter } P \ n \ xs) = \text{length}(\text{kfilter } P \ 0 \ xs)$
by (*simp add: kfilter-length*)

lemma *kfilter-lnth-n-zero-a*:
assumes $k < \text{length} \ (\text{kfilter } P \ n \ xs)$
shows $n \leq (\text{lnth} \ (\text{kfilter } P \ n \ xs) \ k)$
using *assms* **by** (*simp add: kfilter-lnull-conv kfilter-lowerbound*)

lemma *kfilter-lnth-n-zero*:
assumes $k < \text{length} \ (\text{kfilter } P \ n \ xs)$
shows $(\text{lnth} \ (\text{kfilter } P \ n \ xs) \ k) - n = (\text{lnth} \ (\text{kfilter } P \ 0 \ xs) \ k)$
using *assms*
proof (*induct k arbitrary: xs n*)
case 0

```

then show ?case by (cases (kfilter P n xs))
  (simp,
   metis add.left-neutral add-left-cancel kfilter-ldropWhile kfilter-lnull-conv kfilter-lowerbound
   le-add-diff-inverse llength-eq-0 lnth-0 not-gr-zero zero-enat-def)
next
case (Suc k)
then show ?case
proof -
have 1: lnull(kfilter P n xs)  $\implies$  lnth (kfilter P n xs) (Suc k) - n = lnth (kfilter P 0 xs) (Suc k)
  using Suc.premis gr-implies-not-zero llength-lnull by blast
have 2:  $\neg$ lnull(kfilter P n xs)  $\implies$  lnth (kfilter P n xs) (Suc k) - n = lnth (kfilter P 0 xs) (Suc k)
proof -
assume a0:  $\neg$ lnull(kfilter P n xs)
show lnth (kfilter P n xs) (Suc k) - n = lnth (kfilter P 0 xs) (Suc k)
proof -
let ?Least = lleast P xs
have 3:  $\exists x\ xs'. (kfilter P n xs) = LCons x\ xs'$ 
  using a0 kfilter-ldropWhile by blast
obtain x xs' where 4: (kfilter P n xs) = LCons x xs'
  using 3 by auto
have 5: x = n + ?Least
  using kfilter-ldropWhile[of P n xs]
  by (simp add: 4)
have 6:  $\neg$ lnull(kfilter P n xs)  $\longleftrightarrow$   $\neg$ lnull(kfilter P 0 xs)
  by (simp add: kfilter-not-lnull-conv)
have 7:  $\exists y\ ys'. (kfilter P 0 xs) = LCons y\ ys'$ 
  using 6 a0 kfilter-ldropWhile by blast
obtain y ys' where 8: (kfilter P 0 xs) = LCons y ys'
  using 7 by auto
have 9: y = ?Least
  using kfilter-ldropWhile[of P 0 xs] by (simp add: 8)
have 10: x - n = y
  using 5 9 diff-add-inverse by blast
have 11: xs' = kfilter P (Suc (n + ?Least)) (ldrop (Suc ?Least) xs)
  using kfilter-ldropWhile[of P n xs]
  by (metis (no-types, lifting) 4 a0 kfilter-def ltl-simps(2))
have 12: ys' = kfilter P (Suc (0 + ?Least)) (ldrop (Suc ?Least) xs)
  using kfilter-ldropWhile[of P 0 xs]
  by (metis (no-types, lifting) 6 8 a0 kfilter-def ltl-simps(2))
have 13: lnth (kfilter P n xs) (Suc k) - n = lnth xs' k - n
  by (simp add: 4)
have 14: lnth (kfilter P 0 xs) (Suc k) = lnth ys' k
  by (simp add: 8)
have 15:  $\neg(\exists x \in \text{lset } (\text{ldrop } (\text{Suc } ?\text{Least})\ xs). P\ x) \implies$ 
  lnth (kfilter P n xs) (Suc k) - n = lnth (kfilter P 0 xs) (Suc k)
  using 8 Suc by auto
  (metis (full-types) gr-implies-not-zero kfilter-eq-conv kfilter-not-lnull-conv
   ldropn-Suc-LCons ldropn-Suc-conv-ldropn ldropn-eq-LConsD llength-LNil llist.disc(2))
have 16: enat k < llength (kfilter P (Suc (n + ?Least)) (ldrop (Suc ?Least) xs))
  by (metis (no-types, lifting) 12 8 Extended-Nat.eSuc-mono Suc.premis eSuc-enat

```

$kfilter_llength\ llength_LCons)$
have 17: $(\exists x \in lset\ (ldrop\ (Suc\ ?Least)\ xs). P\ x) \implies$
 $lnth\ xs'\ k - (Suc\ (n+?Least)) = lnth\ (kfilter\ P\ 0\ (ldrop\ (Suc\ ?Least)\ xs))\ k$
using *Suc.hyps* **using** 11 16 **by** *blast*
have 18: $enat\ k < llength\ (kfilter\ P\ (Suc\ (?Least))\ (ldrop\ (Suc\ ?Least)\ xs))$
by (*metis* 16 *kfilter-llength*)
have 19: $(\exists x \in lset\ (ldrop\ (Suc\ ?Least)\ xs). P\ x) \implies$
 $lnth\ ys'\ k - (Suc\ (0+?Least)) =$
 $lnth\ (kfilter\ P\ 0\ (ldrop\ (Suc\ ?Least)\ xs))\ k$
using *Suc.hyps* **by** (*simp* *add*: 12 18)
have 20: $(\exists x \in lset\ (ldrop\ (Suc\ ?Least)\ xs). P\ x) \implies$
 $lnth\ (kfilter\ P\ n\ xs)\ (Suc\ k) - n = lnth\ xs'\ k - n$
using 13 **by** *blast*
have 21: $(\exists x \in lset\ (ldrop\ (Suc\ ?Least)\ xs). P\ x) \implies$
 $lnth\ xs'\ k - n = lnth\ (kfilter\ P\ 0\ (ldrop\ (Suc\ ?Least)\ xs))\ k + (Suc\ (?Least))$
using 11 16 17 *kfilter-lowerbound* **by** *fastforce*
have 22: $(\exists x \in lset\ (ldrop\ (Suc\ ?Least)\ xs). P\ x) \implies$
 $lnth\ (kfilter\ P\ 0\ (ldrop\ (Suc\ ?Least)\ xs))\ k + (Suc\ (?Least)) = lnth\ ys'\ k$
using 12 18 19 *kfilter-lowerbound* **by** *fastforce*
have 23: $(\exists x \in lset\ (ldrop\ (Suc\ ?Least)\ xs). P\ x) \implies$
 $lnth\ (kfilter\ P\ n\ xs)\ (Suc\ k) - n = lnth\ (kfilter\ P\ 0\ xs)\ (Suc\ k)$
using 14 20 21 22 **by** *linarith*
show *?thesis*
using 15 23 **by** *blast*
qed
qed
show *?thesis*
using 1 2 **by** *blast*
qed
qed

lemma *kfilter-n-zero*:

shows $(kfilter\ P\ n\ xs) = lmap\ (\lambda i. i+n)\ (kfilter\ P\ 0\ xs)$

proof –

have 1: $llength(kfilter\ P\ n\ xs) = llength\ (lmap\ (\lambda i. i+n)\ (kfilter\ P\ 0\ xs))$

by (*simp* *add*: *kfilter-llength*)

have 2: $\bigwedge k. k < llength(kfilter\ P\ n\ xs) \longrightarrow$

$lnth\ (kfilter\ P\ n\ xs)\ k = lnth\ (lmap\ (\lambda i. i+n)\ (kfilter\ P\ 0\ xs))\ k$

using *kfilter-lnth-n-zero* *kfilter-lnth-n-zero-a*

by (*metis* 1 *le-add-diff-inverse2* *llength-lmap* *lnth-lmap*)

show *?thesis*

by (*simp* *add*: 1 2 *llist-eq-lnth-eq*)

qed

lemma *kfilter-n-zero-a*:

shows $(kfilter\ P\ 0\ xs) = lmap\ (\lambda i. i-n)\ (kfilter\ P\ n\ xs)$

proof –

have 1: $llength(kfilter\ P\ 0\ xs) = llength\ (lmap\ (\lambda i. i-n)\ (kfilter\ P\ n\ xs))$

by (*simp* *add*: *kfilter-llength*)

have 2: $\bigwedge k. k < llength(kfilter\ P\ 0\ xs) \longrightarrow$

$l\text{nth } (k\text{filter } P \ 0 \ xs) \ k = l\text{nth } (l\text{map } (\lambda i. \ i - n) \ (k\text{filter } P \ n \ xs)) \ k$
by (*simp add: kfilter-llength kfilter-lnth-n-zero*)
show *?thesis*
using 1 2 *l\text{list-eq-lnth-eq}* **by** *blast*
qed

lemma *kfilter-holds*:
assumes $y \in l\text{set}(k\text{filter } P \ n \ xs)$
shows $P \ (l\text{nth } xs \ (y - n))$
using *assms in-kfilter-lset[of y P n xs] kfilter-lnull-conv[of P n xs]*
using *lset-lnull* **by** *fastforce*

lemma *kfilter-holds-not*:
assumes $y \in (\{i + n \mid i. \ i < l\text{length } xs\} - (l\text{set } (k\text{filter } P \ n \ xs)))$
shows $\neg P \ (l\text{nth } xs \ (y - n))$
using *assms kfilter-lset[of P n xs] kfilter-lnull-conv[of P n xs]*
by *auto*

lemma *kfilter-holds-a*:
assumes $i < l\text{length } xs$
 $(i + n) \in l\text{set}(k\text{filter } P \ n \ xs)$
shows $P \ (l\text{nth } xs \ i)$
using *assms kfilter-holds[of i + n P n xs]* **by** *simp*

lemma *kfilter-holds-not-a*:
assumes $i < l\text{length } xs$
 $P \ (l\text{nth } xs \ i)$
shows $(i + n) \in l\text{set}(k\text{filter } P \ n \ xs)$
using *assms*
by (*simp add: in-kfilter-lset kfilter-lnull-conv*)

lemma *kfilter-holds-b*:
assumes $i < l\text{length } xs$
shows $(i + n) \in l\text{set}(k\text{filter } P \ n \ xs) = P \ (l\text{nth } xs \ i)$
using *assms*
by (*meson kfilter-holds-a kfilter-holds-not-a*)

lemma *kfilter-holds-c*:
assumes $n \leq i$
 $i - n < l\text{length } xs$
shows $i \in l\text{set}(k\text{filter } P \ n \ xs) = P \ (l\text{nth } xs \ (i - n))$
using *assms*
by (*metis diff-add idiff-enat-enat kfilter-holds kfilter-holds-not-a*)

lemma *kfilter-holds-not-b*:
assumes $n \leq i$
 $i - n < l\text{length } xs$
shows $i \notin l\text{set}(k\text{filter } P \ n \ xs) = (\neg P \ (l\text{nth } xs \ (i - n)))$
using *assms* **by** (*simp add: kfilter-holds-c*)

lemma *kfilter-disjoint-lset-coset*:

shows $(\{i+n \mid i. i < \text{length } xs\} - (\text{lset } (\text{kfilter } P \ n \ xs))) \cap \text{lset } (\text{kfilter } P \ n \ xs) = \{\}$
by *blast*

lemma *lidx-kfilter-expand*:

assumes $(\text{Suc } na) < \text{length}(\text{kfilter } P \ n \ xs)$
shows $\text{lnth } (\text{kfilter } P \ n \ xs) \ na < \text{lnth } (\text{kfilter } P \ n \ xs) (\text{Suc } na)$
using *assms kfilter-mono* **by** *force*

lemma *lidx-kfilter*:

shows *lidx* $(\text{kfilter } P \ n \ xs)$
unfolding *lidx-def*
using *lidx-kfilter-expand* **by** *blast*

lemma *lidx-kfilter-gr-eq*:

assumes
 $k \leq j$
 $j < \text{length}(\text{kfilter } P \ n \ xs)$
shows $\text{lnth } (\text{kfilter } P \ n \ xs) \ k \leq \text{lnth } (\text{kfilter } P \ n \ xs) \ j$
using *assms*
using *lidx-kfilter lidx-less-eq* **by** *blast*

lemma *lidx-kfilter-gr*:

shows $\forall j. k < j \wedge j < \text{length}(\text{kfilter } P \ n \ xs) \longrightarrow$
 $\text{lnth } (\text{kfilter } P \ n \ xs) \ k < \text{lnth } (\text{kfilter } P \ n \ xs) \ j$
using *less-imp-Suc-add lidx-kfilter lidx-less* **by** *blast*

lemma *kfilter-not-before*:

assumes $0 < \text{length}(\text{kfilter } P \ 0 \ xs)$
 $i < \text{lnth } (\text{kfilter } P \ 0 \ xs) \ 0$
shows $\neg P (\text{lnth } xs \ i)$
proof –
have *0*: $(\text{lnth } (\text{kfilter } P \ 0 \ xs) \ 0) < \text{length } xs$
by *(metis assms(1) gen-llength-def kfilter-upperbound llength-code zero-enat-def)*
have *1*: $\neg \text{lnull } (\text{kfilter } P \ 0 \ xs)$
using *assms(1)* **by** *auto*
have *2*: $i \in \text{lset } (\text{kfilter } P \ 0 \ xs) \implies$
 $\neg \text{lnull } (\text{kfilter } P \ 0 \ xs) \implies$
 $i < \text{lnth } (\text{kfilter } P \ 0 \ xs) \ 0 \implies$
False
proof *(induct zs ≡ (kfilter P 0 xs) arbitrary: xs rule: lset-induct)*
case *(find xs)*
then show *?case* **by** *(metis less-not-refl3 lhd-LCons lnth-0-conv-lhd)*
next
case *(step x' xs)*
then show *?case*
proof –
have $\forall ns \ n. \exists na. ((n::\text{nat}) \notin \text{lset } ns \vee \text{lnth } ns \ na = n) \wedge (n \notin \text{lset } ns \vee \text{enat } na < \text{length } ns)$
by *(meson in-lset-conv-lnth)*
then show *?thesis* **using** *step*

```

  by (metis (no-types) leD less-nat-zero-code lidx-kfilter-gr-eq llist.set-intros(2) not-le-imp-less)
qed
qed
have 3:  $i \notin \text{lset } (\text{kfilter } P \ 0 \ xs) \wedge i < \text{length } xs \longrightarrow \neg P (\text{lnth } xs \ i)$ 
  by (simp add: 1 in-kfilter-lset)
have 4:  $i < \text{length } xs$ 
  using 0 assms(2) dual-order.strict-trans enat-ord-simps(2) by blast
show ?thesis
using 1 2 3 4 assms(2) by blast
qed

lemma kfilter-n-not-before:
  assumes  $0 < \text{length } (\text{kfilter } P \ n \ (\text{ldropn } n \ xs))$ 
     $n < \text{length } xs$ 
     $n \leq i$ 
     $i < \text{lnth } (\text{kfilter } P \ n \ (\text{ldropn } n \ xs)) \ 0$ 
  shows  $\neg P (\text{lnth } xs \ i)$ 
proof -
  have 00:  $\neg \text{lnull } (\text{kfilter } P \ n \ (\text{ldropn } n \ xs))$ 
    by (metis assms(1) ldrop-enat less-numeral-extra(3) llength-LNil llist.collapse(1))
  have 0:  $\text{lnth } (\text{kfilter } P \ n \ (\text{ldropn } n \ xs)) \ 0 < \text{length } xs$ 
    using assms kfilter-upperbound[of 0 P n (ldropn n xs)]
    by (metis lappend-ltake-enat-ldropn ldrop-enat llength-lappend llength-ltake min.strict-order-iff
      zero-enat-def)
  have 1:  $i \in \text{lset } (\text{kfilter } P \ n \ (\text{ldropn } n \ xs)) \implies$ 
     $\neg \text{lnull } (\text{kfilter } P \ n \ (\text{ldropn } n \ xs)) \implies$ 
     $i < \text{lnth } (\text{kfilter } P \ n \ (\text{ldropn } n \ xs)) \ 0 \implies$ 
     $n < \text{length } xs \implies$ 
     $n \leq i \implies$ 
    False
  proof (induct zs $\equiv$ (kfilter P n (ldropn n xs)) arbitrary: n xs rule: lset-induct)
  case (find xs)
  then show ?case
  by (metis lnth-0 nat-less-le)
  next
  case (step x' xs)
  then show ?case
  by (metis (no-types, lifting) in-lset-conv-lnth kfilter-eq-LCons-1 kfilter-lowerbound leD
    less-SucI lnth-0)
  qed
  have 2:  $i \notin \text{lset } (\text{kfilter } P \ n \ (\text{ldropn } n \ xs)) \wedge n \leq i \wedge i < \text{length } xs \longrightarrow \neg P (\text{lnth } xs \ i)$ 
    using assms kfilter-holds-not-a[of i] 00
    by (simp add: kfilter-ldropn-lset-a ldrop-enat)
  have 3:  $n \leq i \wedge i < \text{length } xs$ 
    using assms 00 kfilter-ldropWhile[of P n ldropn n xs]
    by (metis 0 enat-ord-simps(2) ldrop-enat less-trans)
  show ?thesis
  using 1 2 3 assms(1) assms(2) assms(4) kfilter-not-lnull-conv by auto
qed

```

lemma *kfilter-not-after:*

assumes $0 < \text{length}(kfilter\ P\ 0\ xs)$

$\text{lnth}(kfilter\ P\ 0\ xs)\ (k-1) < i$

$\text{length}(kfilter\ P\ 0\ xs) = (\text{enat}\ k)$

$i < \text{length}\ xs$

shows $\neg P\ (\text{lnth}\ xs\ i)$

proof –

have $0: \neg \text{lnull}(kfilter\ P\ 0\ xs)$

using *assms(1)* **by** *auto*

have $01: 0 < k$

using $0\ \text{assms}(3)\ \text{gr0I}\ \text{zero-enat-def}$ **by** *fastforce*

have $02: i \notin \text{lset}(kfilter\ P\ 0\ xs)$

by (*metis* $01\ \text{One-nat-def}\ \text{Suc-pred}\ \text{assms}(2)\ \text{assms}(3)\ \text{diff-less}\ \text{enat-ord-simps}(2)$

$\text{in-lset-conv-lnth}\ \text{leD}\ \text{less-Suc-eq-le}\ \text{lid}\ x\text{-kfilter-gr-eq}\ \text{zero-less-one}$)

from $0\ 01\ 02$ **show** *?thesis*

by (*metis* $\text{add.right-neutral}\ \text{assms}(4)\ \text{kfilter-holds-not-a}$)

qed

lemma *kfilter-n-not-after:*

assumes $0 < \text{length}(kfilter\ P\ n\ (\text{ldropn}\ n\ xs))$

$n < \text{length}\ xs$

$\text{lnth}(kfilter\ P\ n\ (\text{ldropn}\ n\ xs))\ (k-1) < i$

$\text{length}(kfilter\ P\ n\ (\text{ldropn}\ n\ xs)) = (\text{enat}\ k)$

$i < \text{length}\ xs$

shows $\neg P\ (\text{lnth}\ xs\ i)$

proof –

have $0: \neg \text{lnull}(kfilter\ P\ n\ (\text{ldropn}\ n\ xs))$

using *assms(1)* **by** *auto*

have $1: 0 < k$

using *assms(1)* *assms(4)* **by** (*simp* $\text{add: zero-enat-def}$)

have $2: i \notin \text{lset}(kfilter\ P\ n\ (\text{ldropn}\ n\ xs))$

using *assms*

by (*metis* $1\ \text{Suc-diff-1}\ \text{enat-ord-simps}(2)\ \text{in-lset-conv-lnth}\ \text{leD}\ \text{less-Suc-eq-le}$

$\text{lid}\ x\text{-kfilter-gr-eq}\ \text{order-refl}$)

from $0\ 1\ 2$ **show** *?thesis*

using *assms* $\text{kfilter-ldropn-lset-a}[\text{of}\ n\ xs\ P\ n]\ \text{kfilter-lowerbound}[\text{of}\ -\ P\ n\ (\text{ldropn}\ n\ xs)]$

by *auto* (*metis* $\text{One-nat-def}\ \text{diff-less}\ \text{le-trans}\ \text{less-imp-le}\ \text{zero-less-one}$)

qed

lemma *kfilter-not-between:*

assumes $\text{lnth}(kfilter\ P\ 0\ xs)\ (k) < i$

$i < \text{lnth}(kfilter\ P\ 0\ xs)\ (\text{Suc}\ k)$

$(\text{Suc}\ k) < \text{length}(kfilter\ P\ 0\ xs)$

shows $\neg P\ (\text{lnth}\ xs\ i)$

proof –

have $0: \exists\ x \in \text{lset}\ xs.\ P\ x$

using *assms(3)* $\text{gr-implies-not-zero}\ \text{kfilter-not-lnull-conv}$ **by** *fastforce*

have $1: \neg \text{lnull}(kfilter\ P\ 0\ xs)$

by (*simp* $\text{add: } 0\ \text{kfilter-not-lnull-conv}$)

```

have 2: lnth (kfilter P 0 xs) (Suc k) ≤ llength xs
  using kfilter-upperbound[of Suc k P 0 xs]
  using 1 assms(3) zero-enat-def by auto
have 3: i ∉ lset(kfilter P 0 xs)
  using assms
  by (metis Suc-ile-eq dual-order.strict-implies-order in-lset-conv-lnth leD lidx-kfilter-gr-eq
    not-less-eq-eq)
from 1 2 3 show ?thesis
by (metis add.right-neutral assms(2) enat-ord-simps(2) kfilter-holds-not-a less-le-trans)
qed

```

```

lemma lfilter-kfilter-ltake-lidx-a:
  assumes k < llength(lfilter P xs)
  shows lidx (ltake k (kfilter P n xs))
  unfolding lidx-def
  using assms
  by (metis Suc-ile-eq kfilter-mono less-imp-le llength-ltake lnth-ltake min.strict-boundedE)

```

```

lemma lfilter-kfilter-ldropn-lidx-a:
  assumes k < llength(lfilter P xs)
  shows lidx (ldropn k (kfilter P n xs))
  using assms unfolding lidx-def
  proof auto
    fix na
    assume a0: enat k < llength (lfilter P xs)
    assume a1: enat (Suc na) < llength (kfilter P n xs) – enat k
    show lnth (ldropn k (kfilter P n xs)) na < lnth (ldropn k (kfilter P n xs)) (Suc na)
    proof –
      have 1: enat (k + (Suc na)) < llength (kfilter P n xs)
      proof –
        have Suc na + k = k + Suc na
        by presburger
        then show ?thesis
        by (metis (no-types) a1 ldropn-eq-LNil ldropn-ldropn leD leI llength-ldropn)
      qed
      have 2 : enat (k + na) < llength (kfilter P n xs)
      by (metis 1 Suc-ile-eq add-Suc-right less-imp-le)
      have 3: lnth (kfilter P n xs) (k + (na)) < lnth (kfilter P n xs) (k + (Suc na))
      using 1 kfilter-mono by auto
      show ?thesis by (metis 1 2 3 add commute lnth-ldropn)
    qed
  qed

```

```

lemma lfilter-kfilter-ldropn-lidx-b:
  assumes k < llength(lfilter P xs)
  shows lidx (kfilter P (lnth (kfilter P n xs) k) (ldropn (lnth (kfilter P n xs) k) xs))
  using assms using lidx-kfilter by blast

```

```

lemma ltake-lset:
  assumes k < llength xs

```

shows $\text{lset } (\text{ltake } k \text{ } xs) = \{(\text{lnth } xs \ i) \mid i. i < k\}$
using *assms*
by (*auto simp add: in-lset-conv-lnth lnth-ltake*)
(blast, meson less-trans lnth-ltake)

lemma *ldropn-lset*:
assumes $k < \text{llength } xs$
shows $\text{lset } (\text{ldropn } k \text{ } xs) = \{(\text{lnth } xs \ i) \mid i. k \leq i \wedge i < \text{llength } xs\}$
using *assms*
proof (*auto simp add: in-lset-conv-lnth*)
fix $n :: \text{nat}$
assume $a1: \text{enat } n < \text{llength } xs - \text{enat } k$
assume $\text{enat } k < \text{llength } xs$
then have $\text{enat } k \leq \text{llength } xs$
by (*meson dual-order.strict-implies-order*)
then have $\text{enat } n + \text{enat } k < \text{llength } xs$
using $a1$ **by** (*meson enat-min*)
then show $\exists na. \text{lnth } (\text{ldropn } k \text{ } xs) \ n = \text{lnth } xs \ na \wedge k \leq na \wedge \text{enat } na < \text{llength } xs$
using $a1$ **by** (*metis ldropn-ldropn le-add2 lhd-ldropn llength-ldropn plus-enat-simps(1)*)

next
fix $i :: \text{nat}$
assume $a1: \text{enat } i < \text{llength } xs$
assume $a2: k \leq i$
have $1: \text{enat } (i-k) < \text{llength } xs - \text{enat } k$
by (*metis a1 a2 diff-add ldropn-Suc-conv-ldropn ldropn-ldropn llength-ldropn llist.disc(2)*
lnull-ldropn not-le-imp-less)
have $2: \text{lnth } (\text{ldropn } k \text{ } xs) \ (i-k) = \text{lnth } xs \ i$
by (*simp add: a1 a2*)
show $\exists n. \text{enat } n < \text{llength } xs - \text{enat } k \wedge \text{lnth } (\text{ldropn } k \text{ } xs) \ n = \text{lnth } xs \ i$
using $1 \ 2$ **by** *blast*

qed

lemma *lfilter-kfilter-ltake-lset-eq*:
assumes
 $k < \text{llength}(\text{lfilter } P \text{ } xs)$
shows $\text{lset } (\text{ltake } k \text{ } (\text{kfilter } P \ 0 \text{ } xs)) =$
 $\text{lset } (\text{kfilter } P \ 0 \text{ } (\text{ltake } ((\text{lnth } (\text{kfilter } P \ 0 \text{ } xs) \ k)) \text{ } xs))$
proof –
have $1: (\text{lnth } (\text{kfilter } P \ 0 \text{ } xs) \ k) < \text{llength } xs$
using *kfilter-llength kfilter-upperbound*
by (*metis assms gen-llength-def kfilter-llength kfilter-upperbound llength-code*)
have $2: \exists x \in \text{lset } xs. P \ x$
using *assms gr-implies-not-zero llength-eq-0 lnull-lfilter* **by** *blast*
have $3: \{i. i < \text{llength}(\text{ltake } ((\text{lnth } (\text{kfilter } P \ 0 \text{ } xs) \ k)) \text{ } xs) \wedge$
 $P \ (\text{lnth } (\text{ltake } ((\text{lnth } (\text{kfilter } P \ 0 \text{ } xs) \ k)) \text{ } xs) \ i)\} =$
 $\{i. i < (\text{lnth } (\text{kfilter } P \ 0 \text{ } xs) \ k) \wedge P \ (\text{lnth } xs \ i)\}$
by (*auto simp add: lnth-ltake 1 order-less-subst2*)
have $5: \{i. i < (\text{lnth } (\text{kfilter } P \ 0 \text{ } xs) \ k) \wedge P \ (\text{lnth } xs \ i)\} =$
 $\{i. i < (\text{lnth } (\text{kfilter } P \ 0 \text{ } xs) \ k) \wedge i \in \text{lset}(\text{kfilter } P \ 0 \text{ } xs)\}$
by (*auto simp add: 1 kfilter-holds-c order-less-subst2*)

have 6: $\{i. i < (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \wedge i \in \text{lset}(\text{kfilter } P \ 0 \ xs)\} =$
 $\{i. i < (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \wedge i \in \{(\text{lnth } (\text{kfilter } P \ 0 \ xs) \ j) \mid j. j < \text{llength}(\text{kfilter } P \ 0 \ xs)\}\}$
by (*simp add: lset-conv-lnth*)
have 7: $\{i. i < (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \wedge i \in \{(\text{lnth } (\text{kfilter } P \ 0 \ xs) \ j) \mid j. j < \text{llength}(\text{kfilter } P \ 0 \ xs)\}\} =$
 $\{(\text{lnth } (\text{kfilter } P \ 0 \ xs) \ j) \mid j. j < k\}$
by (*auto simp add: assms lidx-kfilter-gr kfilter-llength*)
(metis kfilter-llength leD lidx-kfilter-gr-eq not-le-imp-less,
metis assms enat-ord-simps(2) less-trans)
have 8: $k < \text{llength}(\text{kfilter } P \ 0 \ xs)$
by (*simp add: assms kfilter-llength*)
have 9: $\{(\text{lnth } (\text{kfilter } P \ 0 \ xs) \ j) \mid j. j < k\} = \text{lset}(\text{ltake } k \ (\text{kfilter } P \ 0 \ xs))$
using *ltake-lset[of k (kfilter P 0 xs)]* 8 **by** *auto*
have 10: $\text{lset } (\text{kfilter } P \ 0 \ (\text{ltake } ((\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k)) \ xs)) =$
 $\{i. i < (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \wedge$
 $P \ (\text{lnth } (\text{ltake } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs) \ i) \}$
by (*auto simp add: in-kfilter-lset*)
(meson 1 enat-ord-simps(2) less-trans)
have 11: $(\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) = \text{llength}(\text{ltake } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)$
by (*simp add: 1 dual-order.strict-implies-order min-def*)
have 12: $\{i. i < (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \wedge$
 $P \ (\text{lnth } (\text{ltake } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs) \ i) \} =$
 $\{i. i < \text{llength}(\text{ltake } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs) \wedge$
 $P \ (\text{lnth } (\text{ltake } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs) \ i) \}$
using 11 **by** *auto*
show *?thesis*
using 10 12 3 5 6 7 9 **by** *auto*
qed

lemma *lfilter-kfilter-ldropn-lset-eq*:
assumes $k < \text{llength}(\text{lfilter } P \ xs)$
shows $\text{lset}(\text{kfilter } P \ (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)) =$
 $\text{lset}(\text{ldropn } k \ (\text{kfilter } P \ 0 \ xs))$
proof –
have 1: $(\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) < \text{llength } xs$
using *kfilter-llength kfilter-upperbound*
by (*metis assms gen-llength-def kfilter-llength kfilter-upperbound llength-code*)
have 2: $\exists x \in \text{lset } xs. P \ x$
using *assms gr-implies-not-zero llength-eq-0 null-lfilter* **by** *blast*
have 10: $\text{lset}(\text{kfilter } P \ (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)) =$
 $\{(\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) + i \mid i. i < \text{llength } xs - (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \wedge$
 $P \ (\text{lnth } xs \ (i + (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k))) \}$
using 1 *kfilter-ldropn-lset-b[of (lnth (kfilter P 0 xs) k) xs P (lnth (kfilter P 0 xs) k)]*
by *linarith*
have 5: $\{(\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) + i \mid i. i < \text{llength } xs - (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \wedge$
 $P \ (\text{lnth } xs \ (i + (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k))) \} =$
 $\{(\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) + i \mid i. i < \text{llength } xs - (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \wedge$
 $(i + (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k)) \in \text{lset}(\text{kfilter } P \ 0 \ xs) \}$
using *kfilter-holds-b[of - xs 0 P]* **using** 1 2
by *auto*
(metis ldropn-eq-LNil ldropn-ldropn leD llength-ldropn not-le-imp-less,

metis gen-llength-def in-lset-conv-lnth kfilter-upperbound llength-code)

have 51: $\{(lnth (kfilter P 0 xs) k) + i \mid i. i < llength xs - (lnth (kfilter P 0 xs) k) \wedge$
 $(i + (lnth (kfilter P 0 xs) k)) \in lset(kfilter P 0 xs) \} =$
 $\{(lnth (kfilter P 0 xs) k) + i \mid i.$
 $(lnth (kfilter P 0 xs) k) \leq i + (lnth (kfilter P 0 xs) k) \wedge$
 $i + (lnth (kfilter P 0 xs) k) < llength xs \wedge$
 $(i + (lnth (kfilter P 0 xs) k)) \in lset(kfilter P 0 xs) \}$

by auto
 (metis 1 enat-min less-imp-le plus-enat-simps(1),
 metis ldropn-eq-LNil ldropn-ldropn leD llength-ldropn not-le-imp-less)

have 52: $\{(lnth (kfilter P 0 xs) k) + i \mid i.$
 $(lnth (kfilter P 0 xs) k) \leq i + (lnth (kfilter P 0 xs) k) \wedge$
 $i + (lnth (kfilter P 0 xs) k) < llength xs \wedge$
 $(i + (lnth (kfilter P 0 xs) k)) \in lset(kfilter P 0 xs) \} =$
 $\{j. (lnth (kfilter P 0 xs) k) \leq j \wedge j < llength xs \wedge j \in lset(kfilter P 0 xs)\}$

by (metis (no-types, lifting) add commute le-iff-add)

have 53: $\{j. (lnth (kfilter P 0 xs) k) \leq j \wedge j < llength xs \wedge j \in lset(kfilter P 0 xs)\} =$
 $\{j. (lnth (kfilter P 0 xs) k) \leq j \wedge j < llength xs \wedge$
 $j \in \{(lnth (kfilter P 0 xs) jj) \mid jj. jj < llength(kfilter P 0 xs)\}\}$

by (simp add: lset-conv-lnth)

have 54: $\{j. (lnth (kfilter P 0 xs) k) \leq j \wedge j < llength xs \wedge$
 $j \in \{(lnth (kfilter P 0 xs) jj) \mid jj. jj < llength(kfilter P 0 xs)\}\} =$
 $\{(lnth (kfilter P 0 xs) j) \mid j. k \leq j \wedge j < llength (kfilter P 0 xs)\}$

by auto
 (metis assms kfilter-llength lidk-kfilter-gr not-less,
 meson lidk-kfilter-gr-eq,
 metis gen-llength-def kfilter-upperbound llength-code)

have 8: $k < llength(kfilter P 0 xs)$

by (simp add: assms kfilter-llength)

have 9: $\{(lnth (kfilter P 0 xs) j) \mid j. k \leq j \wedge j < llength (kfilter P 0 xs)\} =$
 $lset(ldropn k (kfilter P 0 xs))$

using ldropn-lset[of k (kfilter P 0 xs)] **using 8 by blast**

show ?thesis

using 10 5 51 52 53 54 9 by auto

qed

lemma kfilter-kfilter-ltake:

assumes $k < llength(lfilter P xs)$

shows $(ltake k (kfilter P 0 xs)) =$

$(kfilter P 0 (ltake ((lnth (kfilter P 0 xs) k)) xs))$

using assms

by (simp add: lfilter-kfilter-ltake-lidx-a lfilter-kfilter-ltake-lset-eq lidk-kfilter lidk-lset-eq)

lemma kfilter-kfilter-ldropn:

assumes $k < llength(lfilter P xs)$

shows $(ldropn k (kfilter P 0 xs)) =$

$(kfilter P (lnth (kfilter P 0 xs) k) (ldropn (lnth (kfilter P 0 xs) k) xs))$

using assms

by (simp add: lfilter-kfilter-ldropn-lidx-a lfilter-kfilter-ldropn-lidx-b

lfilter-kfilter-ldropn-lset-eq lidk-lset-eq)

lemma *kfilter-lmap-lfilter*:
shows $\text{lmap } (\lambda n. (\text{lnth } xs \ n)) \ (kfilter \ P \ 0 \ xs) = \text{lfilter } P \ xs$
using *lfilter-kfilter*[of - $P \ 0 \ xs$]
by (*metis* (*no-types*, *lifting*) *diff-zero* *kfilter-llength* *llength-lmap*
llist-eq-lnth-eq *lnth-lmap*)

lemma *lfilter-kfilter-ltake*:
assumes $k < \text{llength}(\text{lfilter } P \ xs)$
shows $\text{ltake } k \ (\text{lfilter } P \ xs) =$
 $(\text{lfilter } P \ (\text{ltake } (\text{lnth } (kfilter \ P \ 0 \ xs) \ k) \ xs))$

proof –
have 2: $\text{lmap } (\lambda n. (\text{lnth } xs \ n)) \ (kfilter \ P \ 0 \ xs) = \text{lfilter } P \ xs$
using *kfilter-lmap-lfilter* **by** *blast*
have 3: $\text{ltake } k \ (\text{lfilter } P \ xs) =$
 $\text{ltake } k \ (\text{lmap } (\lambda n. (\text{lnth } xs \ n)) \ (kfilter \ P \ 0 \ xs))$
by (*simp* *add*: 2)
have 4: $\text{ltake } k \ (\text{lmap } (\lambda n. (\text{lnth } xs \ n)) \ (kfilter \ P \ 0 \ xs)) =$
 $\text{lmap } (\lambda s. \text{lnth } xs \ s) \ (\text{ltake } k \ (kfilter \ P \ 0 \ xs))$
using *ltake-lmap* **by** *blast*
have 6: $(\text{lfilter } P \ (\text{ltake } (\text{lnth } (kfilter \ P \ 0 \ xs) \ k) \ xs)) =$
 $\text{lmap } (\lambda s. (\text{lnth } (\text{ltake } (\text{lnth } (kfilter \ P \ 0 \ xs) \ k) \ xs) \ s))$
 $(kfilter \ P \ 0 \ (\text{ltake } (\text{lnth } (kfilter \ P \ 0 \ xs) \ k) \ xs))$
by (*simp* *add*: *kfilter-lmap-lfilter*)
have 7: $\text{lmap } (\lambda s. \text{lnth } xs \ s) \ (\text{ltake } k \ (kfilter \ P \ 0 \ xs)) =$
 $\text{lmap } (\lambda s. \text{lnth } xs \ s) \ (kfilter \ P \ 0 \ (\text{ltake } (\text{lnth } (kfilter \ P \ 0 \ xs) \ k) \ xs))$
by (*simp* *add*: *assms* *kfilter-kfilter-ltake*)
have 8: $\text{lmap } (\lambda s. \text{lnth } xs \ s) \ (kfilter \ P \ 0 \ (\text{ltake } (\text{lnth } (kfilter \ P \ 0 \ xs) \ k) \ xs)) =$
 $\text{lmap } (\lambda s. (\text{lnth } (\text{ltake } (\text{lnth } (kfilter \ P \ 0 \ xs) \ k) \ xs) \ s))$
 $(kfilter \ P \ 0 \ (\text{ltake } (\text{lnth } (kfilter \ P \ 0 \ xs) \ k) \ xs))$
using *kfilter-kfilter-ltake*[of $k \ P \ xs$]
by (*metis* *gen-llength-def* *kfilter-upperbound* *lappend-ltake-enat-ldropn* *ldistinct-Ex1*
ldistinct-kfilter *llength-code* *llist.map-cong0* *lnth-lappend*)
show *?thesis*
using 2 4 6 7 8 **by** *auto*
qed

lemma *kfilter-lmap-shift-ldropn*:
shows $\text{lmap } (\lambda s. \text{lnth } xs \ (s + (\text{lnth } (kfilter \ P \ 0 \ xs) \ k))))$
 $(kfilter \ P \ 0 \ (\text{ldropn } (\text{lnth } (kfilter \ P \ 0 \ xs) \ k) \ xs)) =$
 $\text{lmap } (\lambda s. \text{lnth } xs \ s)$
 $(kfilter \ P \ (\text{lnth } (kfilter \ P \ 0 \ xs) \ k) \ (\text{ldropn } (\text{lnth } (kfilter \ P \ 0 \ xs) \ k) \ xs))$

proof –
have 1: $\text{llength } (\text{lmap } (\lambda s. \text{lnth } xs \ (s + (\text{lnth } (kfilter \ P \ 0 \ xs) \ k))))$
 $(kfilter \ P \ 0 \ (\text{ldropn } (\text{lnth } (kfilter \ P \ 0 \ xs) \ k) \ xs))) =$
 $\text{llength } (\text{lmap } (\lambda s. \text{lnth } xs \ s)$
 $(kfilter \ P \ (\text{lnth } (kfilter \ P \ 0 \ xs) \ k) \ (\text{ldropn } (\text{lnth } (kfilter \ P \ 0 \ xs) \ k) \ xs))))$
by (*simp* *add*: *kfilter-llength*)
have 2: $\bigwedge i. i < \text{llength } (\text{lmap } (\lambda s. \text{lnth } xs \ (s + (\text{lnth } (kfilter \ P \ 0 \ xs) \ k))))$
 $(kfilter \ P \ 0 \ (\text{ldropn } (\text{lnth } (kfilter \ P \ 0 \ xs) \ k) \ xs))) \implies$

$l\text{nth } (l\text{map } (\lambda s. l\text{nth } xs \ (s + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k))))$
 $\quad (k\text{filter } P \ 0 \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \ i =$
 $l\text{nth } xs \ ((l\text{nth } (k\text{filter } P \ 0 \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \ i) +$
 $\quad (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k))$

by *simp*

have 3: $\bigwedge i. i < l\text{length } (l\text{map } (\lambda s. l\text{nth } xs \ s))$
 $\quad (k\text{filter } P \ (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \implies$
 $l\text{nth } (l\text{map } (\lambda s. l\text{nth } xs \ s))$
 $\quad (k\text{filter } P \ (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k)$
 $\quad \quad (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \ i =$
 $l\text{nth } xs \ (l\text{nth } (k\text{filter } P \ (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k)$
 $\quad \quad (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \ i)$

by *simp*

have 4: $\bigwedge i. i < l\text{length } (l\text{map } (\lambda s. l\text{nth } xs \ (s + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k))))$
 $\quad (k\text{filter } P \ 0 \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \implies$
 $l\text{nth } xs \ ((l\text{nth } (k\text{filter } P \ 0 \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \ i) +$
 $\quad (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k)) =$
 $l\text{nth } xs \ (l\text{nth } (k\text{filter } P \ (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k)$
 $\quad \quad (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \ i)$
using *kfilter-lnth-n-zero*[of - $P \ (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs)$]
kfilter-lowerbound[of - $P \ (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs)$]
using 1 *diff-add* **by** *fastforce*

show *?thesis*

using *l\text{list-eq-lnth-eq}*[of $l\text{map } (\lambda s. l\text{nth } xs \ (s + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k)))$
 $\quad (k\text{filter } P \ 0 \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))]$
using 1 2 3 4 **by** *presburger*

qed

lemma *lfilter-kfilter-l\text{dropn}*:

assumes $k < l\text{length } (l\text{filter } P \ xs)$

shows $(l\text{dropn } k \ (l\text{filter } P \ xs)) =$
 $\quad (l\text{filter } P \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))$

proof –

have 1: $\exists x \in l\text{set } xs. P \ x$

using *assms gr-implies-not-zero l\text{length-eq-0 l\text{null-lfilter}}* **by** *blast*

have 2: $(l\text{filter } P \ xs) = l\text{map } (\lambda s. l\text{nth } xs \ s) \ (k\text{filter } P \ 0 \ xs)$

by (*simp add: kfilter-lmap-lfilter*)

have 3: $l\text{drop } k \ (l\text{filter } P \ xs) =$
 $l\text{drop } k \ (l\text{map } (\lambda s. l\text{nth } xs \ s) \ (k\text{filter } P \ 0 \ xs))$

by (*simp add: 2*)

have 4: $(l\text{drop } k \ (l\text{map } (\lambda s. l\text{nth } xs \ s) \ (k\text{filter } P \ 0 \ xs))) =$
 $\quad (l\text{map } (\lambda s. l\text{nth } xs \ s) \ (l\text{drop } k \ (k\text{filter } P \ 0 \ xs)))$

using *ldrop-lmap* **by** *blast*

have 6: $(l\text{filter } P \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs)) =$
 $l\text{map } (\lambda s. l\text{nth } (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs) \ s)$
 $\quad (k\text{filter } P \ 0 \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))$

by (*simp add: kfilter-lmap-lfilter*)

have 61: $\bigwedge z. z \in l\text{set } ((k\text{filter } P \ 0 \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs)))$
 $\implies (\lambda s. l\text{nth } (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs) \ s) \ z =$
 $(\lambda s. l\text{nth } xs \ (s + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k))) \ z$

```

using assms in-lset-conv-lnth[of - ((kfilter P 0 (ldropn (lnth (kfilter P 0 xs) k ) xs)))]]
by simp
  (metis add commute add.left-neutral kfilter-upperbound ldropn-eq-LNil ldropn-ldropn leD
    lnth-ldropn not-le-imp-less zero-enat-def)
have 7: lmap ( $\lambda s. \text{lnth } (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs) \ s$ )
  (kfilter P 0 (ldropn (lnth (kfilter P 0 xs) k) xs)) =
  lmap ( $\lambda s. \text{lnth } xs \ (s + (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k))$ )
  (kfilter P 0 (ldropn (lnth (kfilter P 0 xs) k) xs))
using 61 by auto
have 8: lmap ( $\lambda s. \text{lnth } xs \ (s + (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k))$ )
  (kfilter P 0 (ldropn (lnth (kfilter P 0 xs) k) xs)) =
  lmap ( $\lambda s. \text{lnth } xs \ s$ )
  (kfilter P (lnth (kfilter P 0 xs) k) (ldropn (lnth (kfilter P 0 xs) k) xs))
by (simp add: kfilter-lmap-shift-ldropn)
have 9: lmap ( $\lambda s. \text{lnth } xs \ s$ )
  (kfilter P (lnth (kfilter P 0 xs) k) (ldropn (lnth (kfilter P 0 xs) k) xs)) =
  lmap ( $\lambda s. \text{lnth } xs \ s$ ) (ldropn k (kfilter P 0 xs))
by (simp add: assms kfilter-kfilter-ldropn)
show ?thesis
by (metis 4 7 8 9 kfilter-lmap-lfilter ldrop-enat)
qed

```

```

lemma lfilter-lnth-aa:
assumes  $n < \text{llength } (\text{lfilter } P \ xs)$ 
shows  $P \ (\text{lnth } (\text{lfilter } P \ xs) \ n)$ 
using assms
by (meson in-lset-conv-lnth lfilter-id-conv lfilter-idem)

```

```

lemma exist-one-conv:
 $(\exists! i. i < \text{llength } xs \wedge P \ (\text{lnth } xs \ i)) \longleftrightarrow$ 
 $(\exists k < \text{llength } xs. P \ (\text{lnth } xs \ k) \wedge$ 
 $(\forall j < \text{llength } xs. j \neq k \longrightarrow \neg P \ (\text{lnth } xs \ j)))$ 
by blast

```

```

lemma lfilter-llength-one-conv-a:
assumes  $\text{llength}(\text{lfilter } P \ xs) = 1$ 
shows  $\exists k < \text{llength } xs. P \ (\text{lnth } xs \ k) \wedge$ 
 $(\forall j < \text{llength } xs. j \neq k \longrightarrow \neg P \ (\text{lnth } xs \ j))$ 
proof –
have 1:  $P \ (\text{lnth } xs \ (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ 0))$ 
by (metis assms(1) kfilter-llength kfilter-lmap-lfilter less-numeral-extra(1) lfilter-lnth-aa
  lnth-lmap zero-enat-def)
have 2:  $(\text{lnth } (\text{kfilter } P \ 0 \ xs) \ 0) < \text{llength } xs$ 
by (metis assms(1) gen-llength-def kfilter-llength kfilter-upperbound less-numeral-extra(1)
  llength-code zero-enat-def)
have 3:  $(\forall j < \text{llength } xs. j \neq (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ 0) \longrightarrow \neg P \ (\text{lnth } xs \ j))$ 
using assms kfilter-not-after[of P xs] kfilter-not-before[of P xs]
by (metis One-nat-def diff-Suc-1 kfilter-llength linorder-neqE-nat one-enat-def zero-less-one)
show ?thesis
using 1 2 3 by blast

```

qed

lemma *lfilter-llength-one-conv-c:*

($\exists k < \text{llength } xs. P (\text{lnth } xs \ k) \wedge$
 $(\forall j < \text{llength } xs. j \neq k \longrightarrow \neg P (\text{lnth } xs \ j))) \longleftrightarrow$
 $(\exists k < \text{llength } xs. P (\text{lnth } xs \ k) \wedge$
 $(\forall j < \text{llength } xs. j < k \vee k < j \longrightarrow \neg P (\text{lnth } xs \ j)))$

using *antisym-conv3* **by** *auto*

lemma *lfilter-llength-one-conv-d:*

($\exists k < \text{llength } xs. P (\text{lnth } xs \ k) \wedge$
 $(\forall j < \text{llength } xs. j < k \vee k < j \longrightarrow \neg P (\text{lnth } xs \ j))) \longleftrightarrow$
 $(\exists k < \text{llength } xs. P (\text{lnth } xs \ k) \wedge$
 $(\forall j. j < k \longrightarrow \neg P (\text{lnth } xs \ j)) \wedge$
 $(\forall j < \text{llength } xs. k < j \longrightarrow \neg P (\text{lnth } xs \ j)))$

by (*meson enat-ord-simps(2) less-trans*)

lemma *exist-one-lfilter-llength-one:*

assumes ($\exists! i. i < \text{llength } xs \wedge P (\text{lnth } xs \ i)$)

shows $\text{llength}(\text{lfilter } P \ xs) \leq 1$

using *assms*

proof *auto*

fix *i :: nat*

assume *a1*: $\forall y \ y'. \text{enat } y < \text{llength } xs \wedge P (\text{lnth } xs \ y) \wedge \text{enat } y' < \text{llength } xs \wedge P (\text{lnth } xs \ y') \longrightarrow$
 $y = y'$

show $\text{llength}(\text{lfilter } P \ xs) \leq 1$

proof –

have *f1*: $\forall e \ ea. (e::\text{enat}) \leq ea \vee ea < e$

by (*meson not-le-imp-less*)

have *f3*: $\text{llength}(\text{kfilter } P \ 0 \ xs) = \text{gen-llength } 0 \ (\text{lfilter } P \ xs)$

by (*metis kfilter-llength-llength-code*)

have *f4*: $\text{enat } 0 + \text{llength } xs = \text{llength } xs$

by (*metis gen-llength-def llength-code*)

have $\text{llength } xs = \text{enat } 0 + \text{llength } xs$

by (*metis (full-types) gen-llength-def llength-code*)

then have *f5*: $\neg 1 < \text{llength}(\text{lfilter } P \ xs)$

proof –

have *g1*: $\text{Suc } 0 = 0 + \text{Suc } 0$

by *auto*

have *g2*: $\bigwedge i. \text{enat } i < \text{llength}(\text{kfilter } P \ 0 \ xs) \implies$

$\text{lnth } xs (\text{lnth}(\text{kfilter } P \ 0 \ xs) \ i - 0) = \text{lnth}(\text{lfilter } P \ xs) \ i$

using *lfilter-kfilter[of - P 0 xs]* **by** *auto*

have *g3*: $\bigwedge k. \text{enat } k < \text{llength}(\text{kfilter } P \ 0 \ xs) \implies$

$\text{lnth}(\text{kfilter } P \ 0 \ xs) \ k - 0 = \text{lnth}(\text{kfilter } P \ 0 \ xs) \ k$

using *kfilter-mono[of - P 0 xs]* **by** *auto*

have *g4*: $\bigwedge n. \text{enat } n < \text{llength}(\text{lfilter } P \ xs) \implies P (\text{lnth}(\text{lfilter } P \ xs) \ n)$

using *lfilter-lnth-aa[of - P xs]* **by** *auto*

have *g5*: $\text{enat } (0 + \text{Suc } 0) = 1$

using *one-enat-def* **by** *auto*

have $\neg 1 < \text{gen-llength } 0 \ (\text{lfilter } P \ xs) \vee \text{enat } 0 < \text{gen-llength } 0 \ (\text{lfilter } P \ xs)$

```

    using zero-enat-def by fastforce
  then show ?thesis
    by (metis g1 g2 g3 g4 g5 a1 f3 f4 kfilter-upperbound ldistinct-conv-lnth ldistinct-kfilter
        llength-code n-not-Suc-n)
  qed
show ?thesis
using f5 not-le by blast
qed
qed

```

```

lemma lfilter-llength-one-conv-b:
assumes  $\exists k < \text{llength } xs. P (\text{lnth } xs \ k) \wedge$ 
         $(\forall j < \text{llength } xs. j \neq k \longrightarrow \neg P (\text{lnth } xs \ j))$ 
shows  $\text{llength}(\text{lfilter } P \ xs) = 1$ 
proof -
  have 1:  $\text{llength}(\text{lfilter } P \ xs) \leq 1$ 
    by (metis assms exist-one-lfilter-llength-one)
  have 2:  $0 < \text{llength}(\text{lfilter } P \ xs)$ 
    by (metis assms gr-zeroI in-lset-conv-lnth llength-eq-0 lnull-lfilter)
  show ?thesis
    by (metis 1 2 One-nat-def Suc-ile-eq dual-order.antisym one-enat-def zero-enat-def)
qed

```

```

lemma lfilter-llength-one-conv:
shows  $(\exists k < \text{llength } xs. P (\text{lnth } xs \ k) \wedge$ 
         $(\forall j < \text{llength } xs. j \neq k \longrightarrow \neg P (\text{lnth } xs \ j))) \longleftrightarrow$ 
         $\text{llength}(\text{lfilter } P \ xs) = 1$ 
using lfilter-llength-one-conv-a[of P xs] lfilter-llength-one-conv-b[of xs P] by blast

```

```

lemma lfilter-llength-one-conv-1:
shows  $(\exists k < \text{llength } xs. P (\text{lnth } xs \ k) \wedge$ 
         $(\forall j < \text{llength } xs. j < k \vee k < j \longrightarrow \neg P (\text{lnth } xs \ j))) \longleftrightarrow$ 
         $\text{llength}(\text{lfilter } P \ xs) = 1$ 
using lfilter-llength-one-conv[of xs P] lfilter-llength-one-conv-c[of xs P]
by blast

```

```

lemma lfilter-llength-one-conv-2:
shows  $(\exists k < \text{llength } xs. P (\text{lnth } xs \ k) \wedge$ 
         $(\forall j. j < k \longrightarrow \neg P (\text{lnth } xs \ j)) \wedge$ 
         $(\forall j < \text{llength } xs. k < j \longrightarrow \neg P (\text{lnth } xs \ j))) \longleftrightarrow$ 
         $\text{llength}(\text{lfilter } P \ xs) = 1$ 
using lfilter-llength-one-conv-1[of xs P] lfilter-llength-one-conv-d[of xs P]
by blast

```

```

lemma lfilter-lappend-ltake:
assumes  $k < \text{llength } xs$ 
shows  $\text{lfilter } P \ (\text{ltake } k \ xs) = \text{ltake } (\text{llength}(\text{lfilter } P \ (\text{ltake } k \ xs))) \ (\text{lfilter } P \ xs)$ 
proof -
  have 1:  $\text{lfilter } P \ xs =$ 
         $\text{lappend } (\text{lfilter } P \ (\text{ltake } (\text{the-enat } k) \ xs)) \ (\text{lfilter } P \ (\text{ldropn } (\text{the-enat } k) \ xs))$ 

```

```

  by (metis enat-ord-code(4) lappend-ltake-enat-ldropn lfilter-lappend-lfinite lfinite-ltake)
have 2: ltake (llength(lfilter P (ltake k xs)))
  (lappend (lfilter P (ltake (the-enat k) xs)) (lfilter P (ldropn (the-enat k) xs))) =
  lfilter P (ltake k xs)
  by (metis assms enat-the-enat lfilter-idem lfinite-ltake llength-eq-infty-conv-lfinite
    llength-lfilter-ile llength-ltake ltake-all ltake-lappend1 min.strict-order-iff)
show ?thesis using 1 2 by simp
qed

```

lemma *kfilter-lappend-lfinite*:

lfinite xs \implies

kfilter P n (lappend xs ys) =
lappend (kfilter P n xs) (kfilter P (n+ (the-enat(llength xs))) ys)

unfolding *kfilter-def*

proof (*induct arbitrary: n rule: lfinite.induct*)

case *lfinite-LNil*

then show ?case **by** *simp*

next

case (*lfinite-LConsI xs x*)

then show ?case

proof –

have 1: (*lzip (lappend (LCons x xs) ys) (iterates Suc n) =*
(LCons (x , n) (lzip (lappend xs ys) (iterates Suc (Suc n)))))

by (*simp add: lzip.ctr(2)*)

have 3: (*lmap snd (lfilter (P \circ fst) (lzip (lappend (LCons x xs) ys) (iterates Suc n))) =*
(if P x then (LCons n (lmap snd (lfilter (P \circ fst) (lzip (lappend xs ys) (iterates Suc (Suc n))))))
else lmap snd (lfilter (P \circ fst) (lzip (lappend xs ys) (iterates Suc (Suc n)))))

using 1 **by** *auto*

have 4: (*(if P x then (LCons n (lmap snd (lfilter (P \circ fst) (lzip (lappend xs ys) (iterates Suc (Suc n))))))*
else lmap snd (lfilter (P \circ fst) (lzip (lappend xs ys) (iterates Suc (Suc n)))) =
(if P x then
(LCons n (lappend (lmap snd (lfilter (P \circ fst) (lzip xs (iterates Suc (Suc n))))
(lmap snd (lfilter (P \circ fst) (lzip ys (iterates Suc ((Suc n) + the-enat (llength
xs)))))))))

else (lappend (lmap snd (lfilter (P \circ fst) (lzip xs (iterates Suc (Suc n))))

(lmap snd (lfilter (P \circ fst) (lzip ys (iterates Suc ((Suc n) + the-enat (llength xs)))))))

using *lfinite-LConsI.hyps(2)* **by** *auto*

have 5: (*lmap snd (lfilter (P \circ fst) (lzip (LCons x xs) (iterates Suc n))) =*
(if P x then (LCons n (lmap snd (lfilter (P \circ fst) (lzip (xs) (iterates Suc (Suc n))))))
else (lmap snd (lfilter (P \circ fst) (lzip (xs) (iterates Suc (Suc n)))))

by (*simp add: lzip.ctr(2)*)

have 6: (*lmap snd (lfilter (P \circ fst) (lzip ys (iterates Suc (n + the-enat (llength (LCons x xs)))))) =*
(lmap snd (lfilter (P \circ fst) (lzip ys (iterates Suc ((Suc n) + the-enat (llength (xs)))))))

using *eSuc-enat lfinite-LConsI.hyps(1) llength-eq-infty-conv-lfinite* **by** *force*

have 7: (*lappend (lmap snd (lfilter (P \circ fst) (lzip (LCons x xs) (iterates Suc n)))*
(lmap snd (lfilter (P \circ fst) (lzip ys (iterates Suc (n + the-enat (llength (LCons x xs)))))) =
lappend (if P x then (LCons n (lmap snd (lfilter (P \circ fst) (lzip xs (iterates Suc (Suc n))))))
else (lmap snd (lfilter (P \circ fst) (lzip xs (iterates Suc (Suc n))))
(lmap snd (lfilter (P \circ fst) (lzip ys (iterates Suc ((Suc n) + (the-enat (llength xs)))))))

```

    using 5 6 by auto
    show ?thesis using 3 4 7 by auto
qed
qed

lemma kfilter-lappend-ltake:
  assumes  $k < \text{length } xs$ 
  shows  $kfilter\ P\ n\ (\text{ltake } k\ xs) = \text{ltake } (\text{length}(kfilter\ P\ n\ (\text{ltake } k\ xs)))\ (kfilter\ P\ n\ xs)$ 
proof -
  have 1:  $kfilter\ P\ n\ xs = \text{lappend } (kfilter\ P\ n\ (\text{ltake } (\text{the-enat } k)\ xs))$ 
     $(kfilter\ P\ (n + (\text{the-enat } k))\ (\text{ldropn } (\text{the-enat } k)\ xs))$ 
    using kfilter-lappend-lfinite[of  $(\text{ltake } (\text{the-enat } k)\ xs)\ P\ n\ (\text{ldropn } (\text{the-enat } k)\ xs)$ ]
    by (metis assms enat-iless enat-ord-code(4) lappend-ltake-ldrop ldrop-enat lfinite-ltake
      length-ltake min.strict-order-iff neq-iff the-enat.simps)
  have 2:  $\text{ltake } (\text{length}(kfilter\ P\ n\ (\text{ltake } k\ xs)))$ 
     $(\text{lappend } (kfilter\ P\ n\ (\text{ltake } (\text{the-enat } k)\ xs))$ 
     $(kfilter\ P\ (n + (\text{the-enat } k))\ (\text{ldropn } (\text{the-enat } k)\ xs))) =$ 
     $kfilter\ P\ n\ (\text{ltake } k\ xs)$ 
    by (metis assms enat-the-enat lfinite-ltake length-eq-infty-conv-lfinite length-ltake
      ltake-all ltake-lappend1 min.strict-order-iff order-refl)
  show ?thesis using 1 2 by simp
qed

```

```

lemma lfilter-ldropn-length:
  assumes  $k < \text{length } xs$ 
  shows  $\text{length } (lfilter\ P\ (\text{ldropn } k\ xs)) \leq \text{length } (lfilter\ P\ (xs))$ 
using assms
proof (induct k arbitrary: xs)
  case 0
  then show ?case by simp
next
  case (Suc k)
  then show ?case
  proof (cases xs)
    case LNil
    then show ?thesis by simp
  next
    case (LCons x21 x22)
    then show ?thesis using Suc Suc-ile-eq by auto
    (meson Suc-ile-eq dual-order.trans ile-eSuc)
  qed
qed

```

```

lemma lfilter-nlength-imp:
  shows  $\text{length } (lfilter\ (\lambda x. P\ x \wedge Q\ x)\ xs) \leq \text{length } (lfilter\ P\ xs)$ 
proof -
  have 0:  $lfilter\ (\lambda x. P\ x \wedge Q\ x)\ xs = lfilter\ (\lambda x. Q\ x \wedge P\ x)\ xs$ 
    by meson
  have 1:  $lfilter\ (\lambda x. Q\ x \wedge P\ x)\ xs = lfilter\ Q\ (lfilter\ P\ xs)$ 
    using lfilter-lfilter[of  $Q\ P\ xs$ ] by auto

```

have 2: $\text{llength } (\text{lfilter } Q \ (\text{lfilter } P \ xs)) \leq \text{llength } (\text{lfilter } P \ xs)$
using *llength-filter-ile* **by** *blast*
show ?thesis **by** (*simp add: 1 2 0*)
qed

lemma *lnths-kfilter-lfilter*:

$\text{lnths } xs \ (\text{lset}(\text{kfilter } P \ 0 \ xs)) = \text{lfilter } P \ xs$
using *lfilter-conv-lnths[of P xs] kfilter-lset[of P 0 xs]*
by *simp*

1.1.9 lrev

lemma *lnull-lrev[simp]*:

$\text{lnull } (\text{lrev } xs) = (\text{lnull } xs)$
unfolding *lrev-def*
by (*metis (full-types) llist-of-list-of lnull-llist-of rev.simps(1) rev-rev-ident*)

lemma *lrev-LNil[simp]*:

$\text{lrev } LNil = LNil$
unfolding *lrev-def*
by *simp*

lemma *lrev-LCons[simp]*:

$\text{lrev } (LCons \ x \ xs) = (\text{if } \text{lfinite } xs \text{ then } (\text{lappend } (\text{lrev } xs) \ (LCons \ x \ LNil)) \text{ else } (LCons \ x \ xs))$
unfolding *lrev-def*
by *simp*
(metis lappend-llist-of-llist-of llist-of.simps(1) llist-of.simps(2))

lemma *ltl-lrev[simp]*:

$\neg \text{lnull } xs \implies$
 $\text{ltl } (\text{lrev } xs) =$
(if lfinite xs then
(if ($\exists a. xs = (LCons \ a \ LNil)$) then $LNil$ else $\text{lappend } (\text{ltl } (\text{lrev } (\text{ltl } xs))) \ (LCons \ (\text{lhs } xs) \ LNil)$)
else ltl xs)
using *ltl-lappend[of (lrev (ltl xs)) (LCons (lhs xs) LNil)]*
by (*metis lfinite-ltl lhs-LCons-ltl llist.collapse(1) llist.disc(1) lnull-lrev lrev-LCons ltl.simps(2)*)

lemma *lmap-lrev*:

$\text{lmap } f \ (\text{lrev } xs) = \text{lrev } (\text{lmap } f \ xs)$
unfolding *lrev-def*
by (*simp add: rev-map*)

lemma *lfinite-lrev [simp]*:

$\text{lfinite } (\text{lrev } xs) = \text{lfinite } xs$
by (*simp add: lrev-def*)

lemma *lset-lrev*:

$\text{lset } (\text{lrev } xs) = \text{lset } xs$
unfolding *lrev-def*

by *simp*

lemma *llength-lrev* [*simp*]:

$llength (lrev\ xs) = llength\ xs$

unfolding *lrev-def*

by (*simp add: length-list-of*)

lemma *lrev-llist-of* [*simp*]:

$lrev (llist-of\ xs) = llist-of (rev\ xs)$

unfolding *lrev-def*

by *simp*

lemma *list-of-lrev* [*simp*]:

$lfinite\ xs \implies list-of (lrev\ xs) = rev (list-of\ xs)$

unfolding *lrev-def*

by *simp*

lemma *lnth-lrev*:

assumes *lfinite xs*

$(enat\ i) < llength\ xs$

shows $(lnth (lrev\ xs)\ i) = (lnth\ xs ((the-enat (llength\ xs)) - (Suc\ i)))$

using *assms*

unfolding *lrev-def*

by *simp*

$(metis\ enat-ord-simps(2)\ length-list-of\ length-list-of-conv-the-enat\ nth-list-of\ rev-nth)$

lemma *lappend-lrev-lfinite*:

assumes *lfinite xs*

lfinite ys

shows $lrev (lappend\ xs\ ys) = lappend (lrev\ ys) (lrev\ xs)$

using *assms*

unfolding *lrev-def*

by (*simp add: lappend-llist-of-llist-of list-of-lappend*)

lemma *lappend-lrev-infa*:

assumes *lfinite xs*

$\neg lfinite\ ys$

shows $lrev (lappend\ xs\ ys) = lappend\ xs\ ys$

using *assms*

unfolding *lrev-def* **by** *auto*

lemma *lappend-lrev--infb*:

assumes $\neg lfinite\ xs$

shows $lrev (lappend\ xs\ ys) = lappend (lrev\ xs) (lrev\ ys)$

using *assms*

unfolding *lrev-def* **by** (*simp add: lappend-inf*)

lemma *lrev-lrev*:

$lrev (lrev\ xs) = xs$

by (*simp add: lrev-def*)

lemma *lrev-ltake*:
assumes *lfinite xs*
 $\text{enat } k < \text{llength } xs$
shows $\text{lrev } (\text{ltake } k \text{ } xs) = \text{ldrop } (\text{the-enat}(\text{llength } xs) - k) (\text{lrev } xs)$
using *assms*
unfolding *lrev-def*
by *simp*
(metis ldrop-llist-of length-list-of-conv-the-enat rev-take take-list-of)

lemma *lrev-ldrop*:
assumes *lfinite xs*
 $\text{enat } k < \text{llength } xs$
shows $\text{lrev } (\text{ldrop } k \text{ } xs) = \text{ltake } (\text{the-enat}(\text{llength } xs) - k) (\text{lrev } xs)$
using *assms*
unfolding *lrev-def*
by *simp*
(metis drop-list-of ldrop-enat length-list-of-conv-the-enat rev-drop)

lemma *lrev-lsubc*:
assumes *lfinite xs*
 $\text{enat } i \leq j$
 $\text{enat } j < \text{llength } xs$
shows $\text{lrev } (\text{lsubc } i \text{ } j \text{ } xs) = \text{lsubc } ((\text{the-enat}(\text{llength } xs)) - (j+1)) ((\text{the-enat}(\text{llength } xs)) - (i+1)) (\text{lrev } xs)$
proof –
have 1: $\text{lrev } (\text{lsubc } i \text{ } j \text{ } xs) = \text{lrev } (\text{ltake } (\text{eSuc}(j-i)) (\text{ldrop } i \text{ } xs))$
unfolding *lsubc-def min-def* **using** *assms apply simp*
by *(metis assms(2) co.enat.exhaust-sel iless-Suc-eq not-less-zero order-le-less-trans)*
have 20: $i < \infty$
by *simp*
have 21: $(\text{llength } (\text{ldrop } (\text{enat } i) \text{ } xs)) = \text{llength } xs - i$
by *(simp add: ldrop-enat)*
have 22: $(\text{the-enat } (\text{llength } xs - i)) - (j - i + 1) = (\text{llength } xs) - \text{eSuc } j$
using *assms apply simp*
using *eSuc-enat llength-eq-infty-conv-lfinite* **by** *fastforce*
have 23: $(\text{enat } (\text{the-enat } (\text{llength } (\text{ldrop } (\text{enat } i) \text{ } xs)) - (j - i + 1))) = (\text{llength } xs) - \text{eSuc } j$
using 21 22 **by** *presburger*
have 2: $\text{lrev } (\text{ltake } (\text{eSuc}(j-i)) (\text{ldrop } i \text{ } xs)) = \text{ldrop } ((\text{llength } xs) - \text{eSuc } j) (\text{lrev } (\text{ldrop } (\text{enat } i) \text{ } xs))$
using *lrev-ltake[of (ldrop i xs) (j-i)+1]* **using** *assms 23 apply auto*
by *(metis 21 22 add commute diff-is-0-eq' eSuc-enat enat-ord-simps(1) enat-the-enat ldrop-enat ldropn-0 lfinite-ldrop llength-eq-infty-conv-lfinite ltake-all not-le-imp-less plus-1-eq-Suc)*
have 3: $(\text{lrev } (\text{ldrop } (\text{enat } i) \text{ } xs)) = (\text{ltake } ((\text{llength } xs) - i) (\text{lrev } xs))$
using *lrev-ldrop[of xs i]* **using** *assms order-le-less-trans*
by *(metis enat-the-enat idiff-enat-enat llength-eq-infty-conv-lfinite)*
have 30: $\text{eSuc } j \leq \text{llength } xs$
using *assms(3) ileI1* **by** *blast*
have 31: $\text{llength } (\text{lrev } (\text{ldrop } (\text{enat } i) \text{ } xs)) = \text{llength } xs - i$

by (*simp add: 21*)
have 32: $\text{length} (\text{ldrop} ((\text{length } xs) - \text{eSuc } j) (\text{lev} (\text{ldrop} (\text{enat } i) xs))) =$
 $(\text{length } xs - i) - ((\text{length } xs) - \text{eSuc } j)$
by (*metis 23 31 ldrops-enat length-ldropn*)
have 33: $(\text{length } xs - i) - ((\text{length } xs) - \text{eSuc } j) = (\text{eSuc } j) - i$
by (*metis 1 2 32 add.commute assms(2) assms(3) eSuc-enat enat-ord-simps(1) idiff-enat-enat length-lev*
length-lsubc ordered-cancel-comm-monoid-diff-class.diff-add-assoc2 plus-1-eq-Suc)

have 4: $\text{ldrop} ((\text{length } xs) - \text{eSuc } j) (\text{lev} (\text{ldrop} (\text{enat } i) xs)) =$
 $\text{ldrop} ((\text{length } xs) - \text{eSuc } j) (\text{ltake} ((\text{length } xs) - i) (\text{lev } xs))$
using 3 **by** *presburger*
have 40: $(\min (\text{enat} (\text{the-enat} (\text{length } xs) - (j+1))) (\text{epred} (\text{length} (\text{lev } xs)))) =$
 $((\text{length } xs) - \text{eSuc } j)$
by (*metis add.commute assms(1) diff-diff-left eSuc-plus-1 enat-less-imp-le enat-ord-simps(2) epred-enat*
idiff-enat-enat less-imp-diff-less lfinite-length-enat length-lev min.absorb1 one-enat-def plus-enat-simps(1)
the-enat.simps)
have 41: $\min (\text{enat} (\text{the-enat} (\text{length } xs) - (j + 1))) (\text{epred} (\text{length} (\text{lev } xs))) =$
 $(\text{enat} (\text{the-enat} (\text{length } xs) - (j + 1)))$
by (*metis 40 add.commute assms(1) eSuc-enat enat-the-enat idiff-enat-enat length-eq-infty-conv-lfinite*
plus-1-eq-Suc)
have 42: $\text{eSuc} (\text{enat} (\text{the-enat} (\text{length } xs) - (i + 1)) - \text{enat} (\text{the-enat} (\text{length } xs) - (j + 1))) =$
 $\text{eSuc} (\text{length } xs - (i+1) - (\text{length } xs - (j+1)))$
using *assms(1) length-eq-infty-conv-lfinite* **by** *fastforce*
have 43: $\text{eSuc} (\text{length } xs - (i+1) - (\text{length } xs - (j+1))) =$
 $\text{eSuc} (j - i)$
using 30 *assms(1) eSuc-enat length-eq-infty-conv-lfinite* **by** *fastforce*
have 44: $\text{eSuc} (j - i) + (\text{enat} (\text{the-enat} (\text{length } xs) - (j + 1))) =$
 $\text{length } xs - i$
using 30 *assms(1) assms(2) eSuc-enat length-eq-infty-conv-lfinite* **by** *fastforce*
have 45: $(\text{eSuc} (\text{enat} (\text{the-enat} (\text{length } xs) - (i + 1)) - \text{enat} (\text{the-enat} (\text{length } xs) - (j + 1))) +$
 $(\text{enat} (\text{the-enat} (\text{length } xs) - (j + 1))) =$
 $\text{length } xs - i$
using 42 43 44 **by** *presburger*
have 5: $\text{ldrop} ((\text{length } xs) - \text{eSuc } j) (\text{ltake} ((\text{length } xs) - i) (\text{lev } xs)) =$
 $\text{lsubc} ((\text{the-enat} (\text{length } xs)) - (j+1)) ((\text{the-enat} (\text{length } xs)) - (i+1)) (\text{lev } xs)$
unfolding *lsubc-def ltake-ldrop*
using 40 41 42 43 44 **by** *presburger*
show *?thesis*
using 1 2 4 5 **by** *presburger*
qed

lemma *llast-lev*:
assumes $\neg \text{null } xs$
shows $\text{llast} (\text{lev } xs) = (\text{if } \text{lfinite } xs \text{ then } \text{lfirst } xs \text{ else } \text{llast } xs)$
using *assms*
unfolding *lev-def*
by (*simp add: last-rev lfirst-def*)

lemma *lfirst-lev*:
assumes $\neg \text{null } xs$

shows $lfirst\ (lrev\ xs) = (if\ lfinite\ xs\ then\ llast\ xs\ else\ lfirst\ xs)$
using *assms*
unfolding *lrev-def*
by *simp*
(metis hd-rev lfirst-def lhd-llist-of llast-llist-of llist-of-list-of)

lemma *llist-all2-lrevI*:
 $llist-all2\ P\ xs\ ys \implies llist-all2\ P\ (lrev\ xs)\ (lrev\ ys)$
unfolding *lrev-def*
by *simp*
(metis llist-all2-lfiniteD llist-all2-llist-of llist-of-list-of)

lemma *llist-all2-lrevD-lfinite*:
assumes $llist-all2\ P\ (lrev\ xs)\ (lrev\ ys)$
 $lfinite\ xs$
 $lfinite\ ys$
shows $llist-all2\ P\ xs\ ys$
using *assms*
unfolding *lrev-def*
by *simp*
(metis llist-all2-llist-of llist-of-list-of)

lemma *llist-all2-lrevD-not-lfinite*:
assumes $llist-all2\ P\ (lrev\ xs)\ (lrev\ ys)$
 $\neg lfinite\ xs$
 $\neg lfinite\ ys$
shows $llist-all2\ P\ xs\ ys$
using *assms* **by** *(simp add: lrev-def)*

lemma *llist-all2-lrevD*:
assumes $llist-all2\ P\ (lrev\ xs)\ (lrev\ ys)$
shows $llist-all2\ P\ xs\ ys$
by *(metis assms lfinite-lrev llist-all2-lfiniteD llist-all2-lrevD-lfinite llist-all2-lrevD-not-lfinite)*

lemma *llist-all2-lrev*:
 $llist-all2\ P\ (lrev\ xs)\ (lrev\ ys) \longleftrightarrow llist-all2\ P\ xs\ ys$
using *llist-all2-lrevD llist-all2-lrevI* **by** *blast*

1.1.10 Transfer rules

context includes *lifting-syntax*
begin

lemma *lbutlast-transfer* [*transfer-rule*]:
 $(llist-all2\ A \implies llist-all2\ A)\ lbutlast\ lbutlast$
by *(auto simp add: rel-fun-def lbutlast-conv-ltake llist-all2-llengthD llist-all2-ltakeI)*

lemma *lleast-transfer* [*transfer-rule*]:
 $((A \implies (=)) \implies llist-all2\ A \implies (=))\ lleast\ lleast$
unfolding *lleast-def[abs-def]*

```

by (auto simp add: rel-fun-def)
  (metis (full-types, opaque-lifting) llist-all2-conv-all-lnth)

lemma lfuse-transfer [transfer-rule]:
  (llist-all2 A ==> llist-all2 A ==> llist-all2 A) lfuse lfuse
by (auto simp add: rel-fun-def intro: llist-all2-lfuseI)

lemma ridx-transfer [transfer-rule]:
  ((R ==> R ==> (=)) ==> llist-all2 R ==> (=)) ridx ridx
by (simp add: llist-all2-rsp rel-fun-def ridx-def llist-all2-conv-all-lnth)
  (meson Suc-ile-eq order-less-imp-le)

lemma lsub-transfer [transfer-rule]:
  ((=) ==> (=) ==> llist-all2 A ==> llist-all2 A) lsub lsub
by (auto simp add: lsub-def rel-fun-def intro: llist-all2-ltakeI llist-all2-ldropI)

lemma lsubc-transfer [transfer-rule]:
  ((=) ==> (=) ==> llist-all2 A ==> llist-all2 A) lsubc lsubc
by (auto simp add: lsubc-def rel-fun-def min-def llist-all2-llengthD
  intro: llist-all2-ltakeI llist-all2-ldropI)

lemma lfusecat-transfer [transfer-rule]:
  (llist-all2 (llist-all2 A) ==> llist-all2 A) lfusecat lfusecat
by (auto intro: llist-all2-lfusecatI)

lemma lrev-parametric [transfer-rule]:
  shows (llist-all2 A ==> llist-all2 A) lrev lrev
by (rule rel-funI)(rule llist-all2-lrevI)

end

end

```

1.2 Non-empty coinductive lists

Coinductive lists are formalised by Andreas Lochbihler in [5]. We define coinductive non-empty lists *'a nellist* as a subtype of coinductive lists using the quotient type construction. The usual operators, like take, drop, length, nth, filter etc. are defined for *'a nellist*. The formalisation is based on terminated coinductive list defined by Andreas Lochbihler.

theory *NELList* **imports**

LList-Extras

begin

Coinductive non-empty lists *'a nellist* are the codatatype defined by the constructors *NNil* of type *'a ⇒ 'a nellist* and *NCons* of type *'a ⇒ 'a nellist ⇒ 'a nellist*.

1.2.1 Type definition

consts *nlast0* :: *'a*

```

codatatype (nset: 'a) nellist =
  NNil (nlast : 'a)
| NCons (nhd : 'a) (ntl : 'a nellist)

```

```

for
  map: nmap
  rel: nellist-all2

```

```

where
  nhd (NNil -) = undefined
| ntl (NNil b) = NNil b
| nlast (NCons - nell) = nlast0 nell

```

overloading

```

  nlast0 == nlast0::'a nellist  $\Rightarrow$  'a

```

begin

partial-function (tailrec) nlast0

```

where nlast0 nell = (case nell of (NNil x)  $\Rightarrow$  x | (NCons y nell')  $\Rightarrow$  nlast0 nell')

```

end

lemma nlast0-nlast [simp]: nlast0 = nlast

proof –

```

  have 1:  $\bigwedge x. nlast0\ x = nlast\ x$ 
  by (simp add: nlast0.simps nlast-def)
  show ?thesis using 1 by (rule ext)

```

qed

lemmas nlast-NNil [code, nitpick-simp] = nellist.sel(1)

lemma nlast-NCons [simp, code, nitpick-simp]: nlast (NCons x nell) = nlast nell

by simp

declare nellist.sel(2) [simp del]

definition nfirst :: 'a nellist \Rightarrow 'a

```

where nfirst nell = (case nell of (NNil b)  $\Rightarrow$  b | (NCons b nell')  $\Rightarrow$  b)

```

primcorec unfold-nellist :: ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'b nellist

where

```

  p a  $\Longrightarrow$  unfold-nellist p g1 g21 g22 a = NNil (g1 a) |
  -  $\Longrightarrow$  unfold-nellist p g1 g21 g22 a =
    NCons (g21 a) (unfold-nellist p g1 g21 g22 (g22 a))

```

declare

```

  unfold-nellist.ctr(1) [simp]
  nellist.corec(1) [simp]

```

1.2.2 Code generator setup

More lemmas about generated constants

lemma *ntl-unfold-nellist*:

$ntl (unfold-nellist IS-NNIL NNIL NHD NTL a) =$
(if IS-NNIL a then NNil (NNIL a) else unfold-nellist IS-NNIL NNIL NHD NTL (NTL a))
by(*simp*)

lemma *is-NNil-ntl* [*simp*]:

$is-NNil nell \implies is-NNil (ntl nell)$
by(*cases nell*) *simp-all*

lemma *nlast-ntl* [*simp*]: $nlast (ntl nell) = nlast nell$

by(*cases nell*) *simp-all*

lemma *unfold-nellist-eq-NNil* [*simp*]:

$unfold-nellist IS-NNIL NNIL NHD NTL a = NNil b \longleftrightarrow IS-NNIL a \wedge b = NNIL a$
by(*auto simp add: unfold-nellist.code*)

lemma *NNil-eq-unfold-nellist* [*simp*]:

$NNil b = unfold-nellist IS-NNIL NNIL NHD NTL a \longleftrightarrow IS-NNIL a \wedge b = NNIL a$
by(*auto simp add: unfold-nellist.code*)

lemma *nmap-is-NNil*:

$is-NNil nell \implies nmap f nell = NNil (f (nlast nell))$
by(*clarsimp simp add: is-NNil-def*)

declare *nellist.map-sel*(2)[*simp*]

lemma *ntl-nmap* [*simp*]:

$ntl (nmap f nell) = nmap f (ntl nell)$
by(*cases nell*) *simp-all*

lemma *nmap-eq-NNil-conv*:

$nmap f nell = NNil y \longleftrightarrow (\exists y'. nell = NNil y' \wedge f y' = y)$
by(*cases nell*) *simp-all*

lemma *NNil-eq-nmap-conv*:

$NNil y = nmap f nell \longleftrightarrow (\exists y'. nell = NNil y' \wedge f y' = y)$
by(*cases nell*) *auto*

declare *nellist.set-sel*(1)[*simp*]

lemma *nset-ntl*: $nset (ntl nell) \subseteq nset nell$

by(*cases nell*) *auto*

lemma *in-nset-ntlD*: $x \in nset (ntl nell) \implies x \in nset nell$

using *nset-ntl[of nell]* **by** *auto*

theorem *nellist-set-induct*[*consumes 1, case-names findnil find step*]:

assumes $x \in \text{nset } nell$
and $\bigwedge nell. \text{is-NNil } nell \implies P (\text{nlast } nell) \text{ } nell$
and $\bigwedge nell. \neg \text{is-NNil } nell \implies P (\text{nhd } nell) \text{ } nell$
and $\bigwedge nell y. [\neg \text{is-NNil } nell; y \in \text{nset } (\text{ntl } nell); P y (\text{ntl } nell)] \implies P y \text{ } nell$
shows $P x \text{ } nell$

using *assms*

proof (*induct*)

case (*NNil z*)

then show ?*case* **by** *force*

next

case (*NCons1 z1 z2*)

then show ?*case* **by** (*fastforce simp del: nellist.disc(2) iff: nellist.disc(2)*)

next

case (*NCons2 z1 z2 xa*)

then show ?*case* **by** *auto*

qed

lemma *nset-induct'* [*consumes 1, case-names findnil find step*]:

assumes *major*: $x \in \text{nset } nell$

and *0*: $\bigwedge nell. \text{is-NNil } nell \implies P (\text{nlast } nell)$

and *1*: $\bigwedge nell. P (\text{NCons } x \text{ } nell)$

and *2*: $\bigwedge x' nell. [x \in \text{nset } nell; P \text{ } nell] \implies P (\text{NCons } x' \text{ } nell)$

shows $P \text{ } nell$

using *major*

proof (*induct y \equiv x nell rule: nellist-set-induct*)

case (*findnil nell*)

then show ?*case* **using** *0 1 2* **by** (*metis nellist.collapse(1) nellist.map-disc-iff nellist.map-sel(1)*)

next

case (*find nell*)

then show ?*case* **by** (*metis 1 nellist.collapse(2)*)

next

case (*step nell*)

then show ?*case* **by** (*metis 2 nellist.collapse(2)*)

qed

lemma *nset-induct* [*consumes 1, case-names findnil find step, induct set: nset*]:

assumes *major*: $x \in \text{nset } nell$

and *findnil*: $\bigwedge nell. \text{is-NNil } nell \implies P (\text{nlast } nell)$

and *find*: $\bigwedge nell. P (\text{NCons } x \text{ } nell)$

and *step*: $\bigwedge x' nell. [x \in \text{nset } nell; x \neq x'; P \text{ } nell] \implies P (\text{NCons } x' \text{ } nell)$

shows $P \text{ } nell$

using *major*

proof (*induct rule: nset-induct'*)

case (*findnil nell*)

then show ?*case* **by** (*simp add: assms(2)*)

next

case (*find nell*)

then show ?*case* **by** (*simp add: assms(3)*)

next

case (*step x' nell*)

then show $?case$ **by** (*metis* *assms*(4) *find*)
qed

1.2.3 Connection with $'a$ *llist*

primcorec *llist-of-nellist* :: $'a$ *nellist* \Rightarrow $'a$ *llist*
where *llist-of-nellist* *nell* = (*case nell of* *NNil* $b \Rightarrow$ *LCons* b *LNil* |

$$NCons\ x\ nell' \Rightarrow LCons\ x\ (llist-of-nellist\ nell'))$$

context

fixes

$b :: 'a$

begin

primcorec *nellist-of-llist-a* :: $'a$ *llist* \Rightarrow $'a$ *nellist* **where**
nellist-of-llist-a *ll* = (*case ll of* *LNil* \Rightarrow *NNil* b |

$$LCons\ x\ ll' \Rightarrow NCons\ x\ (nellist-of-llist-a\ ll'))$$

end

abbreviation *nellist-of-llist* == (λ *ll*. *nellist-of-llist-a* (*llast* *ll*) (*lbutlast* *ll*))

simps-of-case *nellist-of-llist-a-simps* [*simp*, *code*, *nitpick-simp*]: *nellist-of-llist-a.code*

lemmas *nellist-of-llist-a-LNil* = *nellist-of-llist-a-simps*(1)
and *nellist-of-llist-a-LCons* = *nellist-of-llist-a-simps*(2)

lemma *nlast-nellist-of-llist-a-lnull* [*simp*]:
 $lnull\ ll \Longrightarrow nlast\ (nellist-of-llist-a\ b\ ll) = b$

unfolding *lnull-def* **by** *simp*

declare *nellist-of-llist-a.sel*(1)[*simp del*]

lemma *lhd-LNil*:

lhd *LNil* = *undefined*

by(*simp add: lhd-def*)

lemma *nhd-NNil*:

nhd (*NNil* b) = *undefined*

by(*simp add: nhd-def*)

lemma *nhd-nellist-of-llist-a* [*simp*]:

nhd (*nellist-of-llist-a* $b\ ll$) = *lhd* *ll*

by (*cases ll*)

(*simp-all add: lhd-LNil nhd-NNil*)

lemma *ntl-nellist-of-llist-a* [*simp*]:

ntl (*nellist-of-llist-a* $b\ ll$) = *nellist-of-llist-a* b (*ltl* *ll*)

by(*cases ll*) *simp-all*

lemma *llist-of-nellist-eq-LNil*:

*l*list-of-nellist *nell* = *LCons* (*nlast nell*) *LNil* \longleftrightarrow *is-NNil nell*
by (*simp add: nellist.case-eq-if llist-of-nellist.code*)

simps-of-case *l*list-of-nellist-simps [*simp, code, nitpick-simp*]: *l*list-of-nellist.code

lemmas *l*list-of-nellist-NNil = *l*list-of-nellist-simps(1)
and *l*list-of-nellist-NCons = *l*list-of-nellist-simps(2)

declare *l*list-of-nellist.sel [*simp del*]

lemma *lhd-l*list-of-nellist [*simp*]:
 \neg *is-NNil nell* \implies *lhd* (*l*list-of-nellist *nell*) = *nhd nell*
by(*cases nell*) *simp-all*

lemma *lhd-l*list-of-nellist1 [*simp*]:
is-NNil nell \implies *lhd* (*l*list-of-nellist *nell*) = *nlast nell*
by (*cases nell*) *simp-all*

lemma *lhd-l*list-of-nellist2 [*simp*]:
(*case nell of* (*NNil b*) \Rightarrow *lhd LNil* | (*NCons b nell'*) \Rightarrow *lhd* (*l*list-of-nellist *nell*)) = *nhd nell*
by (*cases nell*) (*simp-all add: lhd-LNil nhd-NNil*)

lemma *l*tl-llist-of-nellist [*simp*]:
 \neg *is-NNil nell* \implies *l*tl (*l*list-of-nellist *nell*) = *l*list-of-nellist (*ntl nell*)
by(*cases nell*) *simp-all*

lemma *l*tl-llist-of-nellist1 [*simp*]:
is-NNil nell \implies *l*tl (*l*list-of-nellist *nell*) = *LNil*
by(*cases nell*) *simp-all*

lemma *l*tl-llist-of-nellist2 [*simp*]:
(*case nell of* (*NNil b*) \Rightarrow (*LCons b LNil*) |
(*NCons b nell'*) \Rightarrow *l*tl (*l*list-of-nellist *nell*)) = *l*list-of-nellist (*ntl nell*)
by (*simp add: llist-of-nellist.code nellist.case-eq-if*)

lemma *nellist-of-l*list-a-cong [*cong*]:
assumes *ll* = *ll'* *lfinite ll'* \implies *b* = *b'*
shows *nellist-of-l*list-a *b ll* = *nellist-of-l*list-a *b' ll'*
proof(*unfold* $\langle ll = ll' \rangle$)
from *assms* **have** *lfinite ll'* \longrightarrow *b* = *b'* **by** *simp*
thus *nellist-of-l*list-a *b ll'* = *nellist-of-l*list-a *b' ll'*
by(*coinduction arbitrary: ll'*) *auto*
qed

primcorec *snocn* :: '*a* nellist \Rightarrow '*a* \Rightarrow '*a* nellist
where *snocn nell a* =
(*case nell of* (*NNil x*) \Rightarrow *NCons x* (*NNil a*) |
(*NCons x nell'*) \Rightarrow *NCons x* (*snocn nell' a*))

simps-of-case *snocn-code* [*code*, *simp*, *nitpick-simp*]: *snocn.code*

lemma *snocn-simps* [*simp*]:

shows *nhd-snocn*: $nhd(snocn\ nell\ a) = nfirst\ nell$

and *ntl-snocn*: $ntl(snocn\ nell\ a) = (if\ is-NNil\ nell\ then\ (NNil\ a)\ else\ snocn\ (ntl\ nell)\ a)$

by (*case-tac* [!] *nell*)

(*auto simp add: nfirst-def*)

lemma *is-NNil-snocn*:

is-NNil(snocn nell a) \longleftrightarrow False

by (*auto simp add: snocn-def*)

lemma *nmap-snocn-distrib*:

$nmap\ f\ (snocn\ nell\ a) = snocn\ (nmap\ f\ nell)\ (f\ a)$

proof (*coinduction arbitrary: nell rule: nellist.coinduct-strong*)

case (*Eq-nellist nella*)

then show ?*case*

by (*auto simp add: nellist.case-eq-if nellist.map-sel(1)*)

qed

definition *nfinite* :: '*a* *nellist* \Rightarrow *bool*

where *nfinite nell* \equiv *lfinit* (*llist-of-nellist nell*)

lemma *nfinite-induct* [*consumes 1*, *case-names* *NNil* *NCons*]:

assumes *nfinite nell*

and $\bigwedge y. P\ (NNil\ y)$

and $\bigwedge x\ nell. \llbracket nfinite\ nell; P\ nell \rrbracket \Longrightarrow P\ (NCons\ x\ nell)$

shows *P nell*

using *assms*

unfolding *nfinite-def*

proof (*induct ll \equiv llist-of-nellist nell arbitrary: nell rule: lfinit-induct*)

case *LNil*

then show ?*case* **by** *simp*

next

case *LCons*

then show ?*case* **by** (*metis lfinit.cases lnull-def ltl-llist-of-nellist ltl-simps(2)*)

nellist.collapse(1) nellist.exhaust-sel)

qed

lemma

shows *nfinite-NNil*: *nfinite (NNil x)*

and *nfinite-NConsI*: *nfinite nell \Longrightarrow nfinite (NCons x nell)*

unfolding *nfinite-def*

by *auto*

declare *nfinite-NNil* [*iff*]

lemma *is-NNil-imp-nfinite* [*simp*]:

is-NNil nell \Longrightarrow nfinite nell

using *lfinit.simps llist-of-nellist-eq-LNil* **by** (*auto simp add: nfinite-def*)

lemma *nfinite-NCons* [*simp*]:
 $nfinite\ (NCons\ x\ nell) = nfinite\ nell$
by (*simp add: nfinite-def*)

lemma *nfinite-ntl* [*simp*]:
 $nfinite\ (ntl\ nell) = nfinite\ nell$
by (*cases nell simp-all*)

lemma *nfinite-code* [*code*]:
 $nfinite\ (NNil\ x) = True$
 $nfinite\ (NCons\ x\ nell) = nfinite\ nell$
by *simp-all*

lemma *nfinite-imp-finite-nset*:
assumes *nfinite nell*
shows $finite\ (nset\ nell)$
using *assms*
by (*induct nell rule:nfinite-induct simp-all*)

lemma *nfinite-snocn* [*simp*]:
 $nfinite(snocn\ nell\ a) \longleftrightarrow nfinite\ nell$
(*is ?lhs \longleftrightarrow ?rhs*)
proof
assume *?lhs thus ?rhs*
proof (*induct zs \equiv snocn nell a arbitrary: nell rule: nfinite-induct*)
case (*NNil y*)
then show *?case*
by (*metis is-NNil-snocn nelllist.disc(1)*)
next
case (*NCons x nell*)
then show *?case*
by (*cases nell simp-all*)
qed
next
assume *?rhs thus ?lhs*
by (*induct rule: nfinite-induct auto*)
qed

lemma *snocn-inf*:
 $\neg nfinite\ nell \implies snocn\ nell\ a = nell$
proof (*coinduction arbitrary: nell*)
case (*Eq-nelllist nella*)
then show *?case*
proof –
have 1: $is-NNil\ (snocn\ nella\ a) = is-NNil\ nella$
using *Eq-nelllist by auto*
have 2: $(is-NNil\ (snocn\ nella\ a) \longrightarrow is-NNil\ nella \longrightarrow nlast\ (snocn\ nella\ a) = nlast\ nella)$

```

  by auto
  have 3: ( $\neg$  is-NNil (snocn nella a)  $\longrightarrow$   $\neg$  is-NNil nella  $\longrightarrow$ 
    nhd (snocn nella a) = nhd nella  $\wedge$ 
    ( $\exists$  nell. ntl (snocn nella a) = snocn nell a  $\wedge$  ntl nella = nell  $\wedge$   $\neg$  nfinite nell))
  by (simp add: Eq-nellist nellist.case-eq-if)
  from 1 2 3 show ?thesis by blast
qed
qed

```

```

lemma nfinite-nmap [simp]:
  nfinite (nmap f nell) = nfinite nell (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  assume ?lhs thus ?rhs
  proof (induct zs $\equiv$ nmap f nell arbitrary: nell rule: nfinite-induct)
  case (NNil y)
  then show ?case by (metis nellist.disc(1) nellist.map-disc-iff is-NNil-imp-nfinite)
  next
  case (NCons x nell)
  then show ?case by (metis nellist.sel(5) nfinite-ntl ntl-nmap)
  qed
next
  assume ?rhs thus ?lhs
  by (induct rule: nfinite-induct) simp-all
qed

```

```

lemma nset-snocn-nfinite [simp]:
  nfinite nell  $\implies$  nset(snocn nell a) = nset nell  $\cup$  {a}
by (induct rule: nfinite-induct) auto

```

```

lemma nset-snocn1:
  nset (snocn nell a)  $\subseteq$  nset nell  $\cup$  {a}
proof (cases nfinite nell)
case True
then show ?thesis by simp
next
case False
then show ?thesis by (auto simp add: snocn-inf)
qed

```

```

lemma nset-snocn-conv:
  nset (snocn nell a) = (if nfinite nell then nset nell  $\cup$  {a} else nset nell)
by (simp add: snocn-inf)

```

```

lemma in-nset-snocn-iff:
   $x \in$  nset (snocn nell a)  $\longleftrightarrow$   $x \in$  nset nell  $\vee$  nfinite nell  $\wedge$   $x = a$ 
by (metis Un-iff empty-iff insert-iff nset-snocn-conv)

```

```

lemma llist-of-nellist-inverse-1:
  assumes  $\neg$  nfinite nell
  shows nellist-of-llist-a b (llist-of-nellist nell) = snocn nell b

```

```

using assms
proof (coinduction arbitrary: nell)
case (Eq-nellist nella)
then show ?case
  proof –
    have 1: is-NNil (nellist-of-llist-a b (llist-of-nellist nella)) = is-NNil (snocn nella b)
      by auto
    have 2: (is-NNil (nellist-of-llist-a b (llist-of-nellist nella))  $\longrightarrow$ 
      is-NNil (snocn nella b)  $\longrightarrow$ 
      nlast (nellist-of-llist-a b (llist-of-nellist nella)) = nlast (snocn nella b))
      by simp
    have 3: ( $\neg$  is-NNil (nellist-of-llist-a b (llist-of-nellist nella))  $\longrightarrow$ 
       $\neg$  is-NNil (snocn nella b)  $\longrightarrow$ 
      nhd (nellist-of-llist-a b (llist-of-nellist nella)) = nhd (snocn nella b)  $\wedge$ 
      ( $\exists$  nell.
        ntl (nellist-of-llist-a b (llist-of-nellist nella)) =
        nellist-of-llist-a b (llist-of-nellist nell)  $\wedge$ 
        ntl (snocn nella b) = snocn nell b  $\wedge$   $\neg$  nfinite nell))
      by (metis Eq-nellist lhd-llist-of-nellist ltl-llist-of-nellist nfinite-ntl
        nhd-nellist-of-llist-a ntl-nellist-of-llist-a snocn-inf)
    from 1 2 3 show ?thesis by blast
  qed
qed

```

```

lemma llist-of-nellist-inverse-2:
assumes nfinite nell
shows nellist-of-llist-a b (llist-of-nellist nell) = snocn nell b
using assms
by (induct rule: nfinite-induct) simp-all

```

```

lemma llist-of-nellist-inverse [simp]:
shows nellist-of-llist-a b (llist-of-nellist nell) = snocn nell b
using llist-of-nellist-inverse-1 llist-of-nellist-inverse-2 by fastforce

```

```

lemma llist-of-nellist-inverse-3:
assumes  $\neg$  nfinite nell
shows nellist-of-llist-a (nlast nell) (lbutlast (llist-of-nellist nell)) = nell
using assms
proof (coinduction arbitrary: nell)
case (Eq-nellist nella)
then show ?case
  proof –
    have 1: is-NNil (nellist-of-llist-a (nlast nella) (lbutlast (llist-of-nellist nella))) =
      is-NNil nella
      by (metis Eq-nellist is-NNil-imp-nfinite lbutlast.disc(2) llist-of-nellist.disc-iff
        ltl-llist-of-nellist nellist-of-llist-a.disc(2))
    have 2: (is-NNil (nellist-of-llist-a (nlast nella) (lbutlast (llist-of-nellist nella)))  $\longrightarrow$ 
      is-NNil nella  $\longrightarrow$ 
      nlast (nellist-of-llist-a (nlast nella)
        (lbutlast (llist-of-nellist nella))) = nlast nella)

```

```

using Eq-nellist is-NNil-imp-nfinite by blast
have 3: ( $\neg$  is-NNil (nellist-of-llist-a (nlast nella) (lbutlast (llist-of-nellist nella))))  $\longrightarrow$ 
   $\neg$  is-NNil nella  $\longrightarrow$ 
    nhd (nellist-of-llist-a (nlast nella) (lbutlast (llist-of-nellist nella))) =
      nhd nella  $\wedge$ 
      ( $\exists$  nell.
        ntl (nellist-of-llist-a (nlast nella) (lbutlast (llist-of-nellist nella)))) =
          nellist-of-llist-a (nlast nell) (lbutlast (llist-of-nellist nell))  $\wedge$ 
          ntl nella = nell  $\wedge$   $\neg$  nfinite nell)
      )
by (auto simp add: llist.case-eq-if Eq-nellist)
from 1 2 3 show ?thesis by blast
qed
qed

```

```

lemma llist-of-nellist-inverse-4:
assumes nfinite nell
shows nellist-of-llist-a (nlast nell) (lbutlast (llist-of-nellist nell)) = nell
using assms
by (induct rule: nfinite-induct) (simp-all add: llist-of-nellist.code)

```

```

lemma llist-of-nellist-inverse-a [simp]:
shows nellist-of-llist-a (nlast nell) (lbutlast (llist-of-nellist nell)) = nell
using llist-of-nellist-inverse-3 llist-of-nellist-inverse-4 by fastforce

```

```

lemma nlast-llast:
assumes nfinite nell
shows nlast nell = llast(llist-of-nellist nell)
using assms
by (induct rule: nfinite-induct)
  (simp-all add: llist-of-nellist.code)

```

```

lemma llist-of-nellist-inverse-b [simp]:
shows nellist-of-llist (llist-of-nellist nell) = nell
by (metis lappend-inf lbutlast-snoc llist-of-nellist-inverse llist-of-nellist-inverse-a
  nfinite-def snocn-inf nlast-llast)

```

```

lemma nellist-of-llist-a-eq [simp]:
  nellist-of-llist-a b' ll = NNil b  $\longleftrightarrow$  b = b'  $\wedge$  ll = LNil
by(cases ll) auto

```

```

lemma NNil-eq-nellist-of-llist-a [simp]:
  NNil b = nellist-of-llist-a b' ll  $\longleftrightarrow$  b = b'  $\wedge$  ll = LNil
by(cases ll) auto

```

```

lemma nellist-of-llist-a-inject [simp]:
  nellist-of-llist-a b llx = nellist-of-llist-a c lly  $\longleftrightarrow$  llx = lly  $\wedge$  (lfinite lly  $\longrightarrow$  b = c)
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof(intro iffI conjI impI)
  assume ?rhs
  thus ?lhs by(auto intro: nellist-of-llist-a-cong)

```

```

next
  assume ?lhs
  thus llx = lly
    by (coinduction arbitrary: llx lly) (auto simp add: lnull-def neq-LNil-conv)
  assume lfinite lly
  thus b = c using ‹?lhs›
    unfolding ‹llx = lly› by (induct) simp-all
qed

```

```

lemma nellist-of-llist-a-inverse-1:
  assumes ¬ lfinite ll
  shows llist-of-nellist (nellist-of-llist-a b ll) = lappend ll (LCons b LNil)
  using assms
  by (coinduction arbitrary: ll) auto

```

```

lemma nellist-of-llist-a-inverse-2:
  assumes lfinite ll
  shows llist-of-nellist (nellist-of-llist-a b ll) = lappend ll (LCons b LNil)
  using assms
  proof (induct ll rule: lfinite-induct)
    case (LNil xs)
    then show ?case by (simp add: lnull-def)
  next
    case (LCons xs)
    then show ?case
      by (metis nellist-of-llist-a.disc(2) lappend-code(2) lhd-LCons-ltl lhd-llist-of-nellist
        llist-of-nellist.code llist-of-nellist.simps(2) llist-of-nellist.simps(3)
        ltl-llist-of-nellist nhd-nellist-of-llist-a ntl-nellist-of-llist-a)
  qed

```

```

lemma nellist-of-llist-a-inverse [simp]:
  shows llist-of-nellist (nellist-of-llist-a b ll) = lappend ll (LCons b LNil)
  using nellist-of-llist-a-inverse-1 nellist-of-llist-a-inverse-2 by metis

```

```

lemma nellist-of-llist-inverse [simp]:
  assumes ¬ lnull ll
  shows llist-of-nellist (nellist-of-llist ll) = ll
  using assms by simp

```

```

lemma nmap-nellist-of-llist-a:
  nmap f (nellist-of-llist-a b ll) = nellist-of-llist-a (f b) (lmap f ll)
  by (coinduction arbitrary: ll) (auto simp add: nmap-is-NNil)

```

```

lemma nmap-nellist-of-llist:
  assumes ¬ lnull ll
  shows nmap f (nellist-of-llist ll) = nellist-of-llist (lmap f ll)
  using assms
  by (metis lappend-inf lbutlast-snoc lfinite-lmap llast-lmap lmap-lbutlast nellist-of-llist-a-cong
    nmap-nellist-of-llist-a)

```

lemma *lmap-llist-of-nellist*:
 $\text{lmap } f \text{ (llist-of-nellist nell)} = \text{llist-of-nellist (nmap } f \text{ nell)}$
by (*metis llist.map-disc-iff llist-of-nellist.disc-iff llist-of-nellist-inverse-b nellist-of-llist-inverse nmap-nellist-of-llist*)

definition *cr-nellist* :: 'a llist \Rightarrow 'a nellist \Rightarrow bool
where *cr-nellist* = (λ ll nell. *llist-of-nellist* nell = ll)

lemma *llist-of-nellist-not-lnull*:
 $\neg (\text{lnull (llist-of-nellist nell)})$
by *simp*

lemma *not-lnull-eq-lappend-lbutlast-llast*:
 $\neg(\text{lnull ll}) \longleftrightarrow \text{ll} = \text{lappend (lbutlast ll) (LCons (llast ll) LNil)}$
using *llist.collapse(1)* **by** *fastforce*

lemma *Domainp-help*:
 $\neg \text{lnull ll} \Longrightarrow \exists \text{nell. llist-of-nellist nell} = \text{ll}$
using *nellist-of-llist-inverse* **by** *blast*

lemma *not-lnull-conv-llist-of-nellist*:
 $\neg \text{lnull ll} \longleftrightarrow (\exists \text{nell. llist-of-nellist nell} = \text{ll})$
using *Domainp-help llist-of-nellist-not-lnull* **by** *blast*

lemma *Domainp-cr-nellist* [*transfer-domain-rule*]:
 $\text{Domainp } \text{cr-nellist} = (\lambda \text{ll. } \neg(\text{lnull ll}))$
unfolding *cr-nellist-def Domainp-iff[abs-def]*
using *Domainp-help* **by** *fastforce*

lemma *bi-unique-cr-nellist-help*:
 $\text{llist-of-nellist nelly} = \text{llist-of-nellist nellz} \Longrightarrow \text{nelly} = \text{nellz}$
by (*coinduction arbitrary: nelly nellz*)
(metis llist-of-nellist-inverse-b)

lemma *quotient-help-2*:
 $(\neg \text{lnull (llist-of-nellist nell)} \wedge \text{nellist-of-llist (llist-of-nellist nell)} = \text{nell})$
by (*simp add: bi-unique-cr-nellist-help*)

lemma *quotient-help-nellist-1*:
 $\text{cr-nellist ll nell} \longrightarrow \text{nellist-of-llist ll} = \text{nell}$
by (*metis cr-nellist-def llist-of-nellist-inverse-b*)

lemma *quotient-help-nellist-2*:
 $(\text{cr-nellist (llist-of-nellist nell) nell})$
by (*simp add: cr-nellist-def*)

lemma *quotient-help-3-nellist*:
 $(\neg \text{lnull llx} \wedge \text{llx} = \text{lly}) =$

$(cr_nellist\ llx\ (nellist-of-llist\ llx) \wedge$
 $cr_nellist\ lly\ (nellist-of-llist\ lly) \wedge$
 $nellist-of-llist\ llx =$
 $nellist-of-llist\ lly))$
by (*metis Domainp.DomainI Domainp-cr-nellist cr-nellist-def nellist-of-llist-inverse*)

lemma *Quotient-nellist*:

$Quotient\ (\lambda\ llx\ lly.\ \neg\ lnull\ llx \wedge llx = lly)$
 $nellist-of-llist\ llist-of-nellist\ cr_nellist$

unfolding *Quotient-alt-def*

using *quotient-help-3-nellist quotient-help-nellist-1 quotient-help-nellist-2* **by** *blast*

setup-lifting *Quotient-nellist*

context includes *lifting-syntax*

begin

lemma *bi-unique-cr-nellist* [*transfer-rule*]:

$bi_unique\ cr_nellist$

unfolding *cr-nellist-def bi-unique-def*

by (*auto simp add: bi-unique-cr-nellist-help*)

lemma *right-total-cr-nellist* [*transfer-rule*]:

$right_total\ cr_nellist$

unfolding *cr-nellist-def right-total-def*

by *simp*

lemma *NNil-transfer* [*transfer-rule*]:

$(A ==> pcr_nellist\ A)\ (\lambda b.\ LCons\ b\ LNil)\ NNil$

by (*auto simp add: pcr-nellist-def OO-def cr-nellist-def rel-fun-def*)

lemma *NCons-transfer* [*transfer-rule*]:

$(A ==> pcr_nellist\ A ==> pcr_nellist\ A)\ LCons\ NCons$

by (*auto simp add: pcr-nellist-def OO-def cr-nellist-def rel-fun-def*)

lemma *nmap-transfer* [*transfer-rule*]:

$((=) ==> pcr_nellist\ (=) ==> pcr_nellist\ (=))\ lmap\ nmap$

by (*auto simp add: cr-nellist-def nellist.pcr-cr-eq rel-fun-def lmap-llist-of-nellist*)

lemma *is-NNil-transfer* [*transfer-rule*]:

$(pcr_nellist\ (=) ==> (=))\ is_lfirst\ is_NNil$

by (*simp add: cr-nellist-def nellist.pcr-cr-eq rel-fun-def*)

$(metis\ lbutlast-simps(2)\ llast-singleton\ llist.disc(2)\ llist-of-nellist-eq-LNil$
 $llist-of-nellist-inverse-a\ nellist-of-llist-inverse)$

lemma *is-NNil-nellist-of-llist-conv-is-lfirst*:

assumes $\neg\ lnull\ lx$

shows $is_NNil(nellist-of-llist\ lx) \longleftrightarrow is_lfirst\ lx$

using *assms*

by (*cases lx*)

(*simp*, *metis lbutlast.ctr*(1) *lbutlast.disc-iff*(2) *lbutlast-eq-LNil-conv llist.disc*(1)
nellist-of-llist-a.disc-iff(2))

lemma *nfirst-transfer-a* [*transfer-rule*]:

(*pcr-nellist* (=) \implies (=)) *lhd nfirst*

by (*simp add: cr-nellist-def nellist.pcr-cr-eq nfirst-def rel-fun-def llist-of-nellist.simps*(2))

lemma *nhd-transfer-a1* [*transfer-rule*]:

(*pcr-nellist* (=) \implies (=)) ($\lambda ll.$ *if is-lfirst ll then lhd LNil else lhd ll*) *nhd*

by (*simp add: cr-nellist-def nellist.pcr-cr-eq rel-fun-def OO-def*)

(*metis lbutlast-simps*(2) *lhd-llist-of-nellist llist-of-nellist-eq-LNil nhd-nellist-of-llist-a*
quotient-help-2)

lemma *ntl-transfer* [*transfer-rule*]:

(*pcr-nellist* $A \implies$ *pcr-nellist* A) ($\lambda ll.$ *if is-lfirst ll then ll else ltl ll*) *ntl*

proof (*auto simp add: pcr-nellist-def cr-nellist-def OO-def rel-fun-def intro!*: *llist-all2-ltlI*
lfinite-LConsI dest: llist-all2-lnullD)

show $\bigwedge b y.$ *llist-all2* A (*LCons* b *LNil*) (*llist-of-nellist* y) \implies
llist-all2 A (*LCons* b *LNil*) (*llist-of-nellist* (*ntl* y))

using *llist-all2-ltlI*

by (*metis eq-LConsD is-NNil-ntl llist-all2-LNil1 llist-of-nellist.code llist-of-nellist.simps*(3)
llist-of-nellist-eq-LNil ltl-llist-of-nellist nlast-ntl)

next

show $\bigwedge x y.$ $\forall b. x \neq LCons\ b\ LNil \implies$ *llist-all2* A x (*llist-of-nellist* y) \implies
llist-all2 A (*ltl* x) (*llist-of-nellist* (*ntl* y))

by (*metis lhd-LCons-ltl llist-all2-LNil2 llist-all2-lnullD llist-all2-ltlI*
llist-of-nellist.disc-iff ltl-llist-of-nellist ltl-llist-of-nellist1)

qed

lemma *nfinite-transfer* [*transfer-rule*]:

(*pcr-nellist* (=) \implies (=)) *lfinite nfinite*

by (*auto simp add: nellist.pcr-cr-eq cr-nellist-def nfinite-def rel-fun-def*)

lemma *llist-of-nellist-transfer* [*transfer-rule*]:

(*pcr-nellist* (=) \implies (=)) *id llist-of-nellist*

by (*simp add: pcr-nellist-def cr-nellist-def OO-def rel-fun-def llist.rel-eq*)

lemma *nellist-of-llist-a-transfer* [*transfer-rule*]:

((=) \implies (=) \implies *pcr-nellist* (=)) ($\lambda b ll.$ *lappend ll* (*LCons* b *LNil*)) *nellist-of-llist-a*

by (*auto simp add: pcr-nellist-def cr-nellist-def OO-def rel-fun-def*)

lemma *nlast-nellist-of-llist-a-lfinite* [*simp*]:

lfinite ll \implies *nlast* (*nellist-of-llist-a* b ll) = b

by(*induct rule: lfinite.induct*) *simp-all*

lemma *snocn-transfer* [*transfer-rule*]:

(*pcr-nellist* (A) \implies (A) \implies *pcr-nellist* (A)) ($\lambda ll a.$ *lappend ll* (*LCons* a *LNil*)) *snocn*

unfolding *rel-fun-def*

by (*auto simp add: pcr-nellist-def nellist.pcr-cr-eq cr-nellist-def OO-def*)
 (*metis LNil-transfer llist.rel-intros(2) llist-all2-lappendI llist-of-nellist-inverse*
nellist-of-llist-a-inverse)

lemma *nellist-all2-help-a:*

llist-all2 P (llist-of-nellist nella) (llist-of-nellist nellb) \implies nellist-all2 P nella nellb

by (*coinduction arbitrary: nella nellb*)

(*metis lhd-llist-of-nellist llist.disc(1) llist-all2-LCons-LCons llist-all2-LNil1*
llist-all2-LNil2 llist-all2-lhdD llist-all2-ltlI llist-of-nellist.disc-iff
llist-of-nellist-eq-LNil ltl-llist-of-nellist ltl-simps(2))

lemma *nellist-all2-help-b:*

nellist-all2 P nella nellb \implies llist-all2 P (llist-of-nellist nella) (llist-of-nellist nellb)

proof (*coinduction arbitrary: nella nellb*)

case *LNil*

then show *?case*

by *simp*

next

case *LCons*

then show *?case*

by *auto*

(*metis llist-of-nellist.simps(2) nellist.case-eq-if nellist.rel-sel,*
metis llist-all2-LNil1 ltl-llist-of-nellist ltl-llist-of-nellist1 nellist.rel-sel)

qed

lemma *nellist-all2-transfer [transfer-rule]:*

(*(=) \implies pcr-nellist (=) \implies pcr-nellist (=) \implies (=)*) *llist-all2 nellist-all2*

unfolding *nellist.pcr-cr-eq cr-nellist-def*

using *nellist-all2-help-a nellist-all2-help-b* **by** *blast*

lift-definition *nappend :: 'a nellist \Rightarrow 'a nellist \Rightarrow 'a nellist*

is (λ *llx lly. lappend llx lly*)

by *auto*

lemma *llist-all2-llist-of-nellist-1:*

assumes \neg *lnull y*

llist-all2 ($\lambda x z. \text{llist-of-nellist } z = x$) llist1 y

llist-all2 ($\lambda x z. \text{llist-of-nellist } z = x$) llist2 y

shows *llist1 = llist2*

proof –

have 1: *llist-all2 ($\lambda x z. \text{llist-of-nellist } z = x$) llist1 y =*

llist-all2 (=) llist1 (lmap llist-of-nellist y)

using *llist-all2-lmap2[of (=) llist1 llist-of-nellist y]*

using *llist-all2-mono* **by** *fastforce*

have 2: *llist-all2 ($\lambda x z. \text{llist-of-nellist } z = x$) llist2 y =*

llist-all2 (=) llist2 (lmap llist-of-nellist y)

using *llist-all2-lmap2[of (=) llist2 llist-of-nellist y]*

using *llist-all2-mono* **by** *fastforce*

show *?thesis* **using** *assms*

by (metis (full-types) 1 2 llist.rel-eq)
qed

lemma *llist-all2-llist-of-nellist-2:*

assumes $\neg \text{lnull } y$

llist-all2 $(\lambda z x. \text{llist-of-nellist } z = x) \ y \ \text{llist1}$

llist-all2 $(\lambda z x. \text{llist-of-nellist } z = x) \ y \ \text{llist2}$

shows $\text{llist1} = \text{llist2}$

proof –

have 1: *llist-all2* $(\lambda z x. \text{llist-of-nellist } z = x) \ y \ \text{llist1} =$

llist-all2 $(=) \ (\text{lmap } \text{llist-of-nellist } y) \ \text{llist1}$

using *llist-all2-lmap1*[of $(=)$]

using *llist-all2-mono* **by** *fastforce*

have 2: *llist-all2* $(\lambda z x. \text{llist-of-nellist } z = x) \ y \ \text{llist2} =$

llist-all2 $(=) \ (\text{lmap } \text{llist-of-nellist } y) \ \text{llist2}$

using *llist-all2-lmap1*[of $(=)$]

using *llist-all2-mono* **by** *fastforce*

show *?thesis* **using** *assms*

by (metis (full-types) 1 2 llist.rel-eq)

qed

lift-definition *nconcat* :: $'a \ \text{nellist} \ \text{nellist} \Rightarrow 'a \ \text{nellist}$

is $(\lambda \text{ lxs}. \text{lconcat } \text{lxs})$

apply (*simp add: pcr-nellist-def cr-nellist-def OO-def rel-fun-def llist.rel-eq*)

using *llist.rel-cases mem-Collect-eq llist-all2-llist-of-nellist-1* **by** *fastforce*

lift-definition *nfilter* :: $('a \Rightarrow \text{bool}) \Rightarrow 'a \ \text{nellist} \Rightarrow 'a \ \text{nellist}$

is $\lambda P \ ll. \ (\text{if } \text{lnull}(\text{lfilter } P \ ll) \text{ then } ll \text{ else } \text{lfilter } P \ ll)$

by *auto*

lift-definition *lappendn* :: $'a \ \text{llist} \Rightarrow 'a \ \text{nellist} \Rightarrow 'a \ \text{nellist}$

is *lappend*

by *auto*

lift-definition *nzip* :: $'a \ \text{nellist} \Rightarrow 'b \ \text{nellist} \Rightarrow ('a \times 'b) \ \text{nellist}$

is $(\lambda \text{ llx } \text{lly}. \text{lzip } \text{llx } \text{lly})$

by *auto*

lift-definition *niterates* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'a \ \text{nellist}$

is $(\lambda f \ a. \ \text{iterates } f \ a)$

by *auto*

lift-definition *ndistinct* :: $'a \ \text{nellist} \Rightarrow \text{bool}$

is *ldistinct*

by *auto*

lift-definition *nnth* :: $'a \ \text{nellist} \Rightarrow \text{nat} \Rightarrow 'a$

is $\lambda \text{ ll } n. \ \text{lnth } ll \ (\text{the-enat } (\text{min } (\text{enat } n) \ ((\text{epred } (\text{llength } ll))))))$

by blast

lift-definition *nlength* :: 'a nellist \Rightarrow enat

is $\lambda ll. \text{epred}(\text{llength } ll)$

by auto

lift-definition *ndropn* :: nat \Rightarrow 'a nellist \Rightarrow 'a nellist

is $\lambda n ll. \text{ldropn } (\text{the-enat } (\text{min } (\text{enat } n) ((\text{epred}(\text{llength } ll)))) ll$

by auto

(metis co.enat.exhaust-sel enat-the-enat iless-Suc-eq infinity-ileE leD llength-eq-0
min.cobounded1 min.cobounded2)

lift-definition *ntake* :: enat \Rightarrow 'a nellist \Rightarrow 'a nellist

is $\lambda n ll. \text{ltake } (\text{eSuc } n) ll$

by auto

lift-definition *ntaken* :: nat \Rightarrow 'a nellist \Rightarrow 'a nellist

is $\lambda n ll. \text{ltake } (\text{Suc } n) ll$

using enat-0-iff(2) by auto

1.2.4 The nlast element *nlast*

lemma *nlast-not-nfinite*:

assumes $\neg nfinite \text{ nell}$

shows *nlast nell* = undefined

unfolding *nlast0-nlast*[symmetric]

using *assms*

by (rule contrapos-np)

(induct nell rule: *nlast0.raw-induct*[rotated 1, OF refl, consumes 1],
auto split: *nellist.split-asm*)

lemma *nlast-nellist-of-llist-a*:

nlast (*nellist-of-llist-a* *y ll*) = (if *lfinit* *ll* then *y* else undefined)

by (simp add: *nfinite-def nlast-not-nfinite*)

lemma *nlast-transfer* [transfer-rule]:

(*pcr-nellist* (=) \implies (=)) ($\lambda ll. \text{if } lfinit \text{ } ll \text{ then } llast \text{ } ll \text{ else undefined}$) *nlast*

by (auto simp add: *cr-nellist-def pcr-nellist-def nlast-nellist-of-llist-a OO-def*
dest: llist-all2-lfinitD)

(simp add: *llist.rel-eq nfinite-def nlast-llast nlast-not-nfinite rel-funI*)

lemma *nlast-nmap* [simp]:

nfinite nell \implies *nlast* (*nmap* *f nell*) = *f* (*nlast nell*)

by (induct rule: *nfinite-induct*)

(auto simp add: *nellist.map-sel*(1))

lemma *nset-nlast*:

nfinite nell \implies *nlast nell* \in *nset nell*

by (induct rule: *nfinite-induct*)

(simp-all add: *nellist.set-sel*(3))

1.2.5 nset

lemma *lset-llist-of-nellist-1:*

assumes *nfinite nell*

shows $\text{lset } (\text{lbutlast } (\text{llist-of-nellist } \text{nell})) \cup \{\text{nlast } \text{nell}\} = \text{nset } \text{nell}$ (**is** $?lhs = ?rhs$)

proof(*intro set-eqI iffI*)

fix *x*

assume $x \in ?lhs$

thus $x \in ?rhs$

proof –

have 1: $\text{nlast } \text{nell} = x \implies x \in \text{nset } \text{nell}$

using *assms nset-nlast* **by** *blast*

have 2: $\text{nlast } \text{nell} = x \implies x \in \text{nset } \text{nell}$

unfolding *nlast0-nlast[symmetric]* **by** (*simp add: 1*)

have 3: $x \in \text{lset } (\text{lbutlast } (\text{llist-of-nellist } \text{nell})) \implies x \in \text{nset } \text{nell}$

proof (*induct lbutlast (llist-of-nellist nell) arbitrary: nell rule: llist-set-induct*)

case *find*

then show *?case*

by (*metis lbutlast.disc(1) llist.disc(1) llist-of-nellist-inverse-b ltl-llist-of-nellist1 nellist.set-sel(2) nhd-nellist-of-llist-a*)

next

case (*step y*)

then show *?case*

by (*metis lbutlast.disc(1) lbutlast-ltl llist.disc(1) ltl-llist-of-nellist ltl-llist-of-nellist1 nellist.disc(1) nellist.exhaust-sel nellist.set-intros(3)*)

qed

show *?thesis*

using 2 3 $\langle x \in \text{lset } (\text{lbutlast } (\text{llist-of-nellist } \text{nell})) \cup \{\text{nlast } \text{nell}\} \rangle$ **by** *blast*

qed

next

fix *x*

assume $x \in ?rhs$

thus $x \in ?lhs$

proof(*induct rule: nellist-set-induct*)

case (*findnil nell*)

then show *?case*

by (*cases nell*) *auto*

next

case (*find nell*)

thus *?case*

by (*metis UnI1 lbutlast.disc-iff(2) llist.set-sel(1) ltl-llist-of-nellist nhd-nellist-of-llist-a quotient-help-2*)

next

case *step*

thus *?case*

by (*metis Un-iff in-lset-ltlD lbutlast-ltl ltl-llist-of-nellist nlast-ntl*)

qed

qed

lemma *lset-llist-of-nellist-2:*

assumes $\neg \text{nfinite } \text{nell}$

```

shows  $lset\ (lbutlast\ (l\text{list-of-nellist}\ nell)) = nset\ nell$  (is  $?lhs = ?rhs$ )
proof(intro set-eqI iffI)
  fix  $x$ 
  assume  $x \in ?lhs$ 
  thus  $x \in ?rhs$ 
  proof –
    have  $\exists x \in lset\ (lbutlast\ (l\text{list-of-nellist}\ nell)) \implies x \in nset\ nell$ 
    proof (induct lbutlast (l\text{list-of-nellist} nell) arbitrary: nell rule: llist-set-induct)
      case find
      then show  $?case$ 
      by (metis lbutlast.disc(1) llist.disc(1) llist-of-nellist-inverse-a ltl-llist-of-nellist1
        nellist.set-sel(2) nhd-nellist-of-llist-a)
      next
      case (step y)
      then show  $?case$ 
      by (metis lbutlast.disc(1) lbutlast-ltl llist.disc(1) ltl-llist-of-nellist
        ltl-llist-of-nellist1 nellist.collapse(2) nellist.set-intros(3))
      qed
    show  $?thesis$ 
    using  $\exists x \in lset\ (lbutlast\ (l\text{list-of-nellist}\ nell))$  by blast
  qed
next
  fix  $x$ 
  assume  $x \in ?rhs$ 
  thus  $x \in ?lhs$ 
  using assms
  proof(induct rule: nellist-set-induct)
    case (findnil nell)
    then show  $?case$ 
    by (cases nell) auto
    next
    case (find nell)
    thus  $?case$ 
    by (metis lbutlast-not-lfinite lhd-llist-of-nellist llist.set-sel(1) llist-of-nellist-not-lnull
      nfinite-def)
    next
    case step
    thus  $?case$ 
    by (metis in-lset-ltlD lbutlast-ltl ltl-llist-of-nellist nfinite-ntl)
  qed
qed

lemma lset-llist-of-nellist [simp]:
  (if nfinite nell then  $lset\ (lbutlast(l\text{list-of-nellist}\ nell)) \cup \{nlast\ nell\}$ 
    else  $lset\ (lbutlast\ (l\text{list-of-nellist}\ nell)) = nset\ nell$ )
using lset-llist-of-nellist-1 lset-llist-of-nellist-2 by auto

lemma lset-llist-of-nellist-a [simp]:
   $lset(l\text{list-of-nellist}\ nell) = nset\ nell$ 
proof (cases nfinite nell)

```

```

case True
then show ?thesis
by (metis lappend-lbutlast-llast-id-lfinite lbutlast-lfinite llist.simps(19)
      llist-of-nellist-not-lnull lset-LNil lset-lappend-lfinite lset-llist-of-nellist-1 nfinite-def
      nlast-llast)
next
case False
then show ?thesis by (metis lbutlast-not-lfinite lset-llist-of-nellist-2 nfinite-def)
qed

```

```

lemma nset-nellist-of-llist-a [simp]:
  shows nset (nellist-of-llist-a b ll) = (if lfinite ll then lset ll ∪ {b} else lset ll)
proof (cases lfinite ll)
case True
then show ?thesis
by (metis llist.simps(19) lset-LNil lset-lappend-lfinite lset-llist-of-nellist-a
      nellist-of-llist-a-inverse)
next
case False
then show ?thesis
by (metis lappend-inf lset-llist-of-nellist-a nellist-of-llist-a-inverse)
qed

```

```

lemma nset-transfer [transfer-rule]:
  (pcr-nellist (=) ==> (=)) lset nset
by(auto simp add: cr-nellist-def nellist.pcr-cr-eq)

```

end

1.2.6 *nmap*

```

lemma nmap-eq-NCons-conv:
  nmap f nellx = NCons y nelly  $\longleftrightarrow$ 
  ( $\exists z\ nellz. nellx = NCons z nellz \wedge f z = y \wedge nmap f nellz = nelly$ )
by(cases nellx) simp-all

```

```

lemma NCons-eq-nmap-conv:
  NCons y nelly = nmap f nellx  $\longleftrightarrow$ 
  ( $\exists z\ nellz. nellx = NCons z nellz \wedge f z = y \wedge nmap f nellz = nelly$ )
by(cases nellx) auto

```

1.2.7 Appending two nonempty lazy lists *nappend*

```

lemma nappend-NNil [simp, code, nitpick-simp]:
  nappend (NNil b) nell = (NCons b nell)
by transfer auto

```

```

lemma nappend-NCons [simp, code, nitpick-simp]:
  nappend (NCons a nellx) nelly = NCons a (nappend nellx nelly)
by transfer auto

```


lemma *nhd-nappend* [*simp*]:
 $nhd(nappend\ nellx\ nelly) = (if\ is\ NNil\ nellx\ then\ nlast\ nellx\ else\ nhd\ nellx)$
by (*cases nellx*) *auto*

lemma *ntl-nappend* [*simp*]:
 $ntl(nappend\ nellx\ nelly) = (if\ is\ NNil\ nellx\ then\ nelly\ else\ nappend\ (ntl\ nellx)\ nelly)$
by (*cases nellx*) *auto*

lemma *is-NNil-nappend*:
 $is-NNil(nappend\ nellx\ nelly) \longleftrightarrow False$
by (*cases nellx*) *auto*

lemma *nappend-assoc*:
 $nappend\ (nappend\ nellx\ nelly)\ nellz = nappend\ nellx\ (nappend\ nelly\ nellz)$
by *transfer* (*auto simp add: split-beta lappend-assoc*)

lemma *nmap-nappend-distrib*:
 $nmap\ f\ (nappend\ nellx\ nelly) = nappend\ (nmap\ f\ nellx)\ (nmap\ f\ nelly)$
by *transfer* (*auto simp add: split-beta lmap-lappend-distrib*)

lemma *nlast-nappend*:
 $nlast\ (nappend\ nellx\ nelly) = (if\ nfinite\ nellx\ then\ nlast\ nelly\ else\ nlast\ nellx)$
by *transfer* (*auto simp add: llast-lappend*)

lemma *nfinite-nappend*:
 $nfinite\ (nappend\ nellx\ nelly) \longleftrightarrow nfinite\ nellx \wedge nfinite\ nelly$
by *transfer auto*

lemma *nappend-inf*:
 $\neg nfinite\ nellx \implies nappend\ nellx\ nelly = nellx$
by *transfer* (*auto simp add: lappend-inf*)

lemma *nappend-snocn-inf*:
assumes $\neg nfinite\ nell$
shows $nappend\ nell\ (NNil\ a) = snocn\ nell\ a$
using *assms*
by (*simp add: nappend-inf snocn-inf*)

lemma *nappend-snocn-finite*:
assumes $nfinite\ nell$
shows $nappend\ nell\ (NNil\ a) = snocn\ nell\ a$
using *assms*
by (*induct rule: nfinite-induct*) *simp-all*

lemma *nappend-snocn*:
 $nappend\ nell\ (NNil\ a) = snocn\ nell\ a$
by (*meson nappend-snocn-finite nappend-snocn-inf*)

lemma *split-nellist-first*:

assumes $x \in \text{nset } nell$
shows $nell = (NNil\ x) \vee (\exists\ ys.\ nell = nappend\ (NNil\ x)\ ys) \vee$
 $(\exists\ ys.\ nell = nappend\ ys\ (NNil\ x) \wedge nfinite\ ys \wedge x \notin \text{nset } ys) \vee$
 $(\exists\ ys\ zs.\ nell = nappend\ ys\ (NCons\ x\ zs) \wedge nfinite\ ys \wedge x \notin \text{nset } ys)$
using *assms*
by *transfer*
 $(auto,$
 $\text{metis } eq\text{-}LConsD\ lhd\text{-}lappend\ llist.disc(1)\ llist.expand\ split\text{-}l\text{-}list\text{-}first)$

lemma *split-nellist*:

assumes $x \in \text{nset } nell$
shows $nell = (NNil\ x) \vee (\exists\ ys.\ nell = nappend\ (NNil\ x)\ ys) \vee$
 $(\exists\ ys\ zs.\ nell = nappend\ ys\ (NCons\ x\ zs) \wedge nfinite\ ys) \vee$
 $(\exists\ ys.\ nell = nappend\ ys\ (NNil\ x) \wedge nfinite\ ys)$

using *assms*

by (*meson split-nellist-first*)

lemma *split-nellist-a*:

assumes $nell = (NNil\ x) \vee (\exists\ ys.\ nell = nappend\ (NNil\ x)\ ys) \vee$
 $(\exists\ ys\ zs.\ nell = nappend\ ys\ (NCons\ x\ zs) \wedge nfinite\ ys) \vee$
 $(\exists\ ys.\ nell = nappend\ ys\ (NNil\ x) \wedge nfinite\ ys)$

shows $x \in \text{nset } nell$

proof –

have 1: $nell = (NNil\ x) \implies x \in \text{nset } nell$

by *simp*

have 2: $(\exists\ ys.\ nell = nappend\ (NNil\ x)\ ys) \implies x \in \text{nset } nell$

by *auto*

have 3: $(\exists\ ys\ zs.\ nell = nappend\ ys\ (NCons\ x\ zs) \wedge nfinite\ ys) \implies x \in \text{nset } nell$

by *transfer auto*

have 4: $(\exists\ ys.\ nell = nappend\ ys\ (NNil\ x) \wedge nfinite\ ys) \implies x \in \text{nset } nell$

by *transfer auto*

show *?thesis* **using** 1 2 3 4 *assms* **by** *blast*

qed

1.2.8 Appending a nonempty lazy list to a lazy list *lappendn*

lemma *lappendn-LNil* [*simp*, *code*, *nitpick-simp*]:

lappendn LNil nell = nell

by *transfer auto*

lemma *lappendn-LCons* [*simp*, *code*, *nitpick-simp*]:

lappendn (LCons x ll) nell = NCons x (lappendn ll nell)

by *transfer auto*

lemma *nlast-lappendn-lfinite* [*simp*]:

lfinite ll \implies nlast (lappendn ll nell) = nlast nell

by *transfer*

$(auto\ simp\ add:\ llast\text{-}lappend)$

lemma *nset-lappendn-lfinite* [*simp*]:

$lfinite\ ll \implies nset\ (lappendn\ ll\ nell) = lset\ ll \cup nset\ nell$
by *transfer auto*

lemma *nlength-nappend* [*simp*]:

$nlength\ (nappend\ nellx\ nelly) = nlength\ nellx + nlength\ nelly + 1$

by *transfer*

$(auto, metis\ co.enat.exhaust-sel\ epred-iadd1\ iadd-Suc-right\ llength-eq-0\ plus-1-eSuc(2))$

lemma *nfinite-nlength-enat*:

assumes *nfinite nell*

shows $\exists\ n.\ nlength\ nell = enat\ n$

using *assms*

by *transfer (metis epred-conv-minus idiff-enat-enat lfinite-llength-enat one-enat-def)*

lemma *nlength-eq-enat-nfiniteD*:

$nlength\ nell = enat\ n \implies nfinite\ nell$

by *transfer (metis epred-Infty llength-eq-enat-lfiniteD not-lfinite-llength)*

lemma *nfinite-conv-nlength-enat*:

$nfinite\ nell \longleftrightarrow (\exists\ n.\ nlength\ nell = enat\ n)$

using *nfinite-nlength-enat nlength-eq-enat-nfiniteD* **by** *blast*

1.2.9 The length of a nonempty lazy list *nlength*

lemma [*simp*, *nitpick-simp*]:

shows *nlength-NNil*: $nlength\ (NNil\ b) = 0$

and *nlength-NCons*: $nlength\ (NCons\ x\ nell) = eSuc\ (nlength\ nell)$

by *(transfer, simp) (transfer, auto)*

lemma *llength-llist-of-nellist* [*simp*]:

$epred(llength\ (llist-of-nellist\ nell)) = nlength\ nell$

by *transfer auto*

lemma *nlength-nmap* [*simp*]:

$nlength\ (nmap\ f\ nell) = nlength\ nell$

by *transfer simp*

definition *gen-nlength* :: $nat \Rightarrow 'a\ nellist \Rightarrow enat$

where *gen-nlength* *n nell* = $enat\ n + nlength\ nell$

lemma *gen-nlength-code* [*code*]:

$gen-nlength\ n\ (NNil\ b) = enat\ n$

$gen-nlength\ n\ (NCons\ x\ nell) = gen-nlength\ (n + 1)\ nell$

by *(simp-all add: gen-nlength-def iadd-Suc eSuc-enat[symmetric] iadd-Suc-right)*

lemma *nlength-code* [*code*]:

$nlength = gen-nlength\ 0$

by *(simp add: gen-nlength-def fun-eq-iff zero-enat-def)*

1.2.10 The n th element of a nonempty lazy list $nnth$

lemma *nnth-NNil* [*nitpick-simp*]:

$$nnth\ (NNil\ b)\ n = b$$

by *transfer simp*

lemma *nnth-NCons*:

$$nnth\ (NCons\ x\ nell)\ n = (\text{case } n \text{ of } 0 \Rightarrow x \mid \text{Suc } n' \Rightarrow nnth\ nell\ n')$$

by (*transfer fixing: n*)

(*auto simp add: lnth-LCons Nitpick.case-nat-unfold zero-enat-def min-enat1-conv-enat, metis enat-0-iff(1) less-not-refl3 llength-eq-0 min-def min-enat1-conv-enat, metis enat-min-eq-0-iff min-enat1-conv-enat not-gr-zero the-enat.simps the-enat-0, metis One-nat-def epred-enat epred-min min-enat1-conv-enat the-enat.simps*)

lemma *nnth-code* [*simp, nitpick-simp, code*]:

shows *nnth-0*: $nnth\ (NCons\ x\ nell)\ 0 = x$

and *nnth-Suc-NCons*: $nnth\ (NCons\ x\ nell)\ (\text{Suc } n) = nnth\ nell\ n$

by(*simp-all add: nnth-NCons*)

lemma *lnth-llist-of-nellist* [*simp*]:

$$lnth\ (l\text{list-of-nellist}\ nell)\ (the-enat\ (min\ (enat\ n)\ ((epred\ (llength\ (l\text{list-of-nellist}\ nell)))))) \\ = nnth\ nell\ n$$

by *transfer auto*

lemma *nnth-nmap* [*simp*]:

$$enat\ n \leq nlength\ nell \implies nnth\ (nmap\ f\ nell)\ n = f\ (nnth\ nell\ n)$$

by *transfer*

(*metis co-enat.exhaust-sel iless-Suc-eq llength-eq-0 llength-lmap lnth-lmap min.orderE the-enat.simps*)

lemma *nhd-conv-nnth*:

$$\neg is-NNil\ nell \implies nhd\ nell = nnth\ nell\ 0$$

by (*metis nellist.collapse(2) nnth-0*)

lemmas *nnth-0-conv-nhd* = *nhd-conv-nnth*[*symmetric*]

lemma *nnth-ntl*:

$$nnth\ (ntl\ nell)\ n = nnth\ nell\ (\text{Suc } n)$$

by (*metis nellist.exhaust-sel nellist.sel(4) nnth-NNil nnth-Suc-NCons*)

lemma *in-nset-conv-nnth*:

$$x \in nset\ nell \longleftrightarrow (\exists\ n. enat\ n \leq nlength\ nell \wedge nnth\ nell\ n = x)$$

by *transfer*

(*metis eSuc-epred iless-Suc-eq in-lset-conv-lnth llength-eq-0 min-absorb1 the-enat.simps*)

lemma *nnth-beyond*:

$$nlength\ nell < enat\ n \implies nnth\ nell\ n = nlast\ nell$$

by *transfer*

(*metis co-enat.exhaust-sel epred-llength less-enatE lfinite-ltl llast-conv-lnth llength-eq-0 llength-eq-enat-lfiniteD min.absorb4 the-enat.simps*)

lemma *exists-Pred-nnth-nset*:

$(\exists x \in \text{nset } nell. P x) = (\exists n. n \leq \text{nlength } nell \wedge P (\text{nnth } nell n))$
by (*metis in-nset-conv-nnth*)

lemma *nset-conv-nnth*:

$\text{nset } nell = \{\text{nnth } nell n \mid n. \text{enat } n \leq \text{nlength } nell\}$
by (*auto simp add: in-nset-conv-nnth*)

lemma *nnth-nappend1*:

$\text{enat } n \leq \text{nlength } nellx \implies \text{nnth } (\text{nappend } nellx \text{ nelly}) n = \text{nnth } nellx n$
proof (*induct n arbitrary: nellx*)
case 0
then show ?case
by (*metis is-NNil-def is-NNil-nappend nellist.sel(1) nhd-nappend nnth-0-conv-nhd nnth-NNil*)
next
case (*Suc n*)
then show ?case
proof (*cases nellx*)
case (*NNil x1*)
then show ?thesis
using *Suc.premis enat-0-iff(1)* **by** *auto*
next
case (*NCons x21 x22*)
then show ?thesis
using *Suc.hyps Suc.premis Suc-ile-eq* **by** *auto*
qed
qed

lemma *nnth-nappend2*:

$\llbracket \text{nlength } nellx = \text{enat } k; k < n \rrbracket \implies \text{nnth } (\text{nappend } nellx \text{ nelly}) n = \text{nnth } nelly (n - \text{Suc } k)$
proof (*induct n arbitrary: nellx k*)
case 0
then show ?case **by** *blast*
next
case (*Suc n*)
then show ?case
by (*cases nellx*)
(auto simp add: eSuc-def zero-enat-def split: enat.split-asm)
qed

lemma *nnth-nappend*:

$\text{nnth } (\text{nappend } nellx \text{ nelly}) n =$
(if enat n ≤ nlength nellx then nnth nellx n else nnth nelly (n − Suc(the-enat(nlength nellx))))
by (*cases nlength nellx*)
(auto simp add: nnth-nappend1 nnth-nappend2)

lemma *nnth-nlast*:

$n\text{finite } nell \implies n\text{last } nell = \text{nnth } nell (\text{the-enat } (\text{nlength } nell))$
by *transfer*
(simp,

*metis co.enat.exhaust-sel enat-the-enat ile-eSuc infinity-ileE lfinite-llength-enat
llast-conv-lnth llength-eq-0 min.idem)*

1.2.11 *ntake*

lemma *ntake-NNil* [*simp*, *code*, *nitpick-simp*]:

ntake n (NNil b) = (NNil b)

by *transfer auto*

lemma *ntake-0* [*simp*]:

ntake 0 nell = (NNil (nfirst nell))

by *transfer (auto simp add: ltake.ctr(2))*

lemma *ntake-Suc-NCons* [*simp*]:

ntake (eSuc n) (NCons x nell) = (NCons x (ntake n nell))

by *transfer auto*

lemma *ntake-Suc*:

ntake (eSuc n) nell =

(case nell of (NNil b) \Rightarrow (NNil b) | (NCons x nell') \Rightarrow (NCons x (ntake n nell')))

by *(cases nell) simp-all*

lemma *is-NNil-ntake* [*simp*]:

is-NNil(ntake n nell) \longleftrightarrow is-NNil nell \vee n=0

proof *(cases nell)*

case *(NNil x1)*

then show *?thesis by simp*

next

case *(NCons x nell1)*

then show *?thesis*

proof *(cases n)*

case *(enat nat)*

then show *?thesis*

by *(metis NCons enat-coexhaust nell1.disc(1) nell1.disc(2) ntake-0 ntake-Suc-NCons)*

next

case *infinity*

then show *?thesis*

by *(metis NCons eSuc-infinity i0-ne-infinity nell1.disc(2) ntake-Suc-NCons)*

qed

qed

lemma *ntake-eq-NNil-iff* [*simp*]:

ntake n nell = (NNil x) \longleftrightarrow nell = (NNil x) \vee (n = 0 \wedge nfirst nell = x)

proof *(cases nell)*

case *(NNil x1)*

then show *?thesis*

using *ntake-0 by fastforce*

next

case *(NCons x nell1)*

then show *?thesis*

proof *(cases n)*

```

case (enat nat)
then show ?thesis
by (metis NCons is-NNil-ntake nellist.disc(1) nellist.disc(2) nellist.inject(1) ntake-0)
next
case infinity
then show ?thesis
by (metis NCons eSuc-infinity infinity-ne-i0 nellist.distinct(1) ntake-Suc-NCons)
qed
qed

```

```

lemma NNil-eq-ntake-iff [simp]:
  (NNil x) = ntake n nell  $\longleftrightarrow$  nell = (NNil x)  $\vee$  (n = 0  $\wedge$  nfirst nell = x)
by (metis ntake-eq-NNil-iff)

```

```

lemma ntake-NCons [code, nitpick-simp]:
  ntake n (NCons x nell) = (case n of 0  $\Rightarrow$  (NNil x) | (eSuc n')  $\Rightarrow$  (NCons x (ntake n' nell)) )
by (simp add: co.enat.case-eq-if)
  (metis eSuc-epred nellist.simps(6) nfirst-def ntake-Suc-NCons)

```

```

lemma nhd-ntake [simp]:
  n  $\neq$  0  $\implies$  nhd(ntake n nell) = nhd nell
unfolding nhd-def
by (simp add: nellist.case-eq-if )
  (metis (no-types, lifting) co.enat.case-eq-if nellist.collapse(2) nellist.sel(3) ntake-NCons)

```

```

lemma ntl-ntake:
  n  $\neq$  0  $\implies$  ntl(ntake n nell) = ntake (epred n) (ntl nell)
by (cases nell) (simp, metis eSuc-epred nellist.sel(5) ntake-Suc-NCons)

```

```

lemma ntl-ntake-0:
  ntl(ntake 0 nell) = (NNil (nfirst nell))
by simp

```

```

lemma ntake-ntl:
  ntake n (ntl nell) = ntl(ntake (Suc n) nell)
by (simp add: enat-0-iff(1) ntl-ntake)

```

```

lemma nlength-ntake [simp]:
  nlength (ntake n nell) = min n (nlength nell)
by transfer simp

```

```

lemma ntake-nmap [simp]:
  ntake n (nmap f nell) = nmap f (ntake n nell)
by transfer simp

```

```

lemma ntake-ntake [simp]:
  ntake n (ntake m nell) = ntake (min n m) nell
by transfer simp

```

```

lemma nset-ntake:

```

$nset (ntake\ n\ nell) \subseteq nset\ nell$
by *transfer (simp add: lset-ltake)*

lemma *ntake-all*:

$nlength\ nell \leq m \implies ntake\ m\ nell = nell$
by *transfer (auto, metis eSuc-epred eSuc-ile-mono llength-eq-0 ltake-all)*

lemma *nfinite-ntake [simp]*:

$nfinite (ntake\ n\ nell) \longleftrightarrow nfinite\ nell \vee n < \infty$
by *transfer (metis Extended-Nat.eSuc-mono eSuc-infinity lfinite-ltake)*

lemma *ntake-nappend1*:

$n \leq nlength\ nellx \implies ntake\ n\ (nappend\ nellx\ nelly) = ntake\ n\ nellx$
by *transfer (auto, metis eSuc-epred eSuc-ile-mono llength-eq-0 ltake-lappend1)*

lemma *ntake-nappend2*:

assumes $nlength\ nellx < n$
shows $ntake\ n\ (nappend\ nellx\ nelly) = nappend\ nellx\ (ntake\ (n - nlength\ nellx - 1)\ nelly)$
proof *(cases nellx)*
case *(NNil x1)*
then show *?thesis using assms*
by *(metis eSuc-le-iff eSuc-minus-1 idiff-0-right ileI1 nappend-NNil nlength-NNil ntake-Suc-NCons)*
next
case *(NCons x21 x22)*
then show *?thesis*
proof *(cases nfinite nellx)*
case *True*
then show *?thesis*
using *assms*
proof *(transfer)*
fix *llxa :: 'a llist*
fix *na*
fix *llya :: 'a llist*
assume *a0: $\neg lnull\ llxa \wedge llxa = llxa$*
assume *a1: $lfinite\ llxa$*
assume *a2: $epred\ (llength\ llxa) < na$*
assume *a3: $\neg lnull\ llya \wedge llya = llya$*
show $\neg lnull\ (ltake\ (eSuc\ na)\ (lappend\ llxa\ llya)) \wedge$
 $ltake\ (eSuc\ na)\ (lappend\ llxa\ llya) =$
 $lappend\ llxa\ (ltake\ (eSuc\ (na - epred\ (llength\ llxa) - 1))\ llya)$
proof *—*
have *1: $\neg lnull\ (ltake\ (eSuc\ na)\ (lappend\ llxa\ llya))$*
using *a0 by force*
have *2: $ltake\ (eSuc\ na)\ (lappend\ llxa\ llya) =$*
 $lappend\ (ltake\ (eSuc\ na)\ llxa)\ (ltake\ ((eSuc\ na) - llength\ llxa)\ llya)$
by *(meson ltake-lappend)*
have *21: $llength\ llxa \leq (eSuc\ na)$*
by *(metis a0 a2 co.enat.exhaust-sel eSuc-ile-mono leD le-cases llength-eq-0)*
have *3: $lappend\ (ltake\ (eSuc\ na)\ llxa)\ (ltake\ ((eSuc\ na) - llength\ llxa)\ llya) =$*
 $lappend\ llxa\ (ltake\ (eSuc\ na - llength\ llxa)\ llya)$


```

    using ltake-lappend2[of lla (eSuc na) lly] 21 2 by auto
  have 4: (eSuc na - llength lla) = (eSuc (na - epred (llength lla) - 1))
    by (metis a0 a1 a2 canonically-ordered-monoid-add-class.lessE co.enat.exhaust-sel eSuc-infinity
      eSuc-minus-1 eSuc-minus-eSuc enat-add-sub-same llength-eq-0 llength-eq-infty-conv-lfinite)
  have 5: (ltake (eSuc na - llength lla) lly) =
    (ltake (eSuc (na - epred (llength lla) - 1)) lly)
    using 4 by auto
  show ?thesis using 1 2 3 5 by fastforce
qed
qed
next
case False
then show ?thesis using assms
by (simp add: nappend-inf ntake-all)
qed
qed

```

lemma *ntake-eq-ntake-antimono*:

```

  [| ntake n nellx = ntake n nelly; m ≤ n |] ⇒ ntake m nellx = ntake m nelly
by (metis min.orderE ntake-ntake)

```

lemma *ntake-nnth*:

```

assumes enat m ≤ n
shows (nnth (ntake n nell) m) = (nnth nell m)
using assms
proof (induct m arbitrary: nell n)
case 0
then show ?case
proof (cases n rule: enat-coexhaust)
case 0
then show ?thesis
using 0.premis
by (metis nellist.case(2) nellist.collapse(1) nellist.exhaust-sel nfirst-def
  nnth-0-conv-nhd nnth-NNil ntake-eq-NNil-iff)
next
case (eSuc n')
then show ?thesis
by (simp add: nellist.case-eq-if nnth-0-conv-nhd ntake-Suc)
qed
next
case (Suc m)
then show ?case
proof (cases n rule: enat-coexhaust)
case 0
then show ?thesis
using Suc.premis by (simp add: enat-0-iff(1))
next
case (eSuc n')
then show ?thesis
proof (cases nell)

```

```

case (NNil x1)
then show ?thesis by simp
next
case (NCons x21 x22)
then show ?thesis
using Suc.hyps Suc.premis Suc-ile-eq eSuc by force
qed
qed
qed

```

1.2.12 *ntaken*

lemma *ntaken-NNil* [*simp*, *code*, *nitpick-simp*]:

ntaken *n* (NNil *b*) = (NNil *b*)

by *transfer*

(*metis* eSuc-enat llist.disc(2) ltake-LNil ltake-eSuc-LCons)

lemma *ntaken-0* [*simp*]:

ntaken 0 *nell* = (NNil (*nfirst* *nell*))

proof (*cases* *nell*)

case (NNil *x1*)

then show ?thesis **by** (*metis* ntake-0 ntake-NNil ntaken-NNil)

next

case (NCons *x21* *x22*)

then show ?thesis

by *transfer* (*simp*, (*metis* One-nat-def ltake-0 ltake-eSuc-LCons one-eSuc one-enat-def zero-neq-one))

qed

lemma *ntaken-Suc-NCons* [*simp*]:

ntaken (Suc *n*) (NCons *x* *nell*) = (NCons *x* (*ntaken* *n* *nell*))

by *transfer* (*auto* *simp* *add*: zero-enat-def, *metis* eSuc-enat ltake-eSuc-LCons)

lemma *ntaken-Suc*:

ntaken (Suc *n*) *nell* =

(*case* *nell* of (NNil *b*) \Rightarrow (NNil *b*) | (NCons *x* *nell'*) \Rightarrow (NCons *x* (*ntaken* *n* *nell'*)))

by (*cases* *nell*) *simp-all*

lemma *is-NNil-ntaken* [*simp*]:

is-NNil(*ntaken* *n* *nell*) \longleftrightarrow *is-NNil* *nell* \vee *n*=0

proof (*cases* *nell*)

case (NNil *x1*)

then show ?thesis

by *simp*

next

case (NCons *x* *nell1*)

then show ?thesis

proof (*cases* *n*)

case 0

then show ?thesis

by *simp*

```

next
case (Suc nat)
then show ?thesis
by (simp add: NCons)
qed
qed

```

lemma *ntaken-eq-NNil-iff* [simp]:

$ntaken\ n\ nell = (NNil\ x) \longleftrightarrow nell = (NNil\ x) \vee (n = 0 \wedge nfirst\ nell = x)$

```

proof (cases nell)
case (NNil x1)
then show ?thesis
by (metis ntake-0 ntake-NNil ntaken-NNil)
next
case (NCons x nell1)
then show ?thesis
proof (cases n)
case 0
then show ?thesis
by (simp add: NCons)
next
case (Suc nat)
then show ?thesis
using NCons by force
qed
qed

```

lemma *NNil-eq-ntaken-iff* [simp]:

$(NNil\ x) = ntaken\ n\ nell \longleftrightarrow nell = (NNil\ x) \vee (n = 0 \wedge nfirst\ nell = x)$

by (metis *ntaken-eq-NNil-iff*)

lemma *ntaken-NCons* [code, nitpick-simp]:

$ntaken\ n\ (NCons\ x\ nell) = (case\ n\ of\ 0 \Rightarrow (NNil\ x) \mid (Suc\ n') \Rightarrow (NCons\ x\ (ntaken\ n'\ nell)))$

by (cases n) (auto simp add: nfirst-def)

lemma *nhd-ntaken* [simp]:

$n \neq 0 \implies nhd(ntaken\ n\ nell) = nhd\ nell$

by (cases nell)
(simp-all add: Nitpick.case-nat-unfold ntaken-NCons)

lemma *ntl-ntaken*:

$n \neq 0 \implies ntl(ntaken\ n\ nell) = ntaken\ (n-1)\ (ntl\ nell)$

by simp-all
(metis Suc-pred nellist.exhaust-sel nellist.sel(4) nellist.sel(5) ntaken-NNil ntaken-Suc-NCons)

lemma *ntl-ntaken-0*:

$ntl(ntaken\ 0\ nell) = (NNil\ (nfirst\ nell))$

by simp

lemma *ntaken-ntl*:

$ntaken\ n\ (ntl\ nell) = ntl(ntaken\ (Suc\ n)\ nell)$
by (*simp add: enat-0-iff(1) ntl-ntaken*)

lemma *ntaken-nlength* [*simp*]:
 $nlength\ (ntaken\ n\ nell) = min\ n\ (nlength\ nell)$
by *transfer simp*

lemma *ntaken-nmap* [*simp*]:
 $ntaken\ n\ (nmap\ f\ nell) = nmap\ f\ (ntaken\ n\ nell)$
using *enat-0-iff(2)* **by** *transfer auto*

lemma *ntaken-ntaken* [*simp*]:
 $ntaken\ n\ (ntaken\ m\ nell) = ntaken\ (min\ n\ m)\ nell$
using *enat-0-iff(2)* **by** *transfer auto*

lemma *nset-ntaken*:
 $nset\ (ntaken\ n\ nell) \subseteq nset\ nell$
by *transfer (simp add: lset-ltake)*

lemma *ntaken-all*:
 $nlength\ nell \leq m \implies ntaken\ m\ nell = nell$
by *transfer*
(auto simp add: zero-enat-def, metis eSuc-enat eSuc-epred eSuc-ile-mono llength-eq-0 ltake-all)

lemma *ntaken-nnth*:
shows $(nnth\ (ntaken\ m\ nell)\ k) = (nnth\ nell\ (min\ k\ m))$
apply *transfer*
by (*auto simp add: min-def lnth-ltake ltake-all*)
(metis co.enat.sel(2) enat-eSuc-iff enat-ord-simps(1) epred-le-epredI order-trans,
metis Suc-ile-eq co.enat.exhaust-sel iless-Suc-eq llength-eq-0,
metis Suc-ile-eq co.enat.exhaust-sel iless-Suc-eq llength-eq-0,
meson dual-order.strict-iff-order enat-ord-simps(2) linorder-not-less order-less-le-trans,
metis Suc-ile-eq co.enat.exhaust-sel iless-Suc-eq llength-eq-0)

lemma *nfinite-ntaken* [*simp*]:
 $nfinite\ (ntaken\ n\ nell)$
by *transfer simp*

lemma *ntaken-nlast*:
 $nlast\ (ntaken\ n\ nell) = nnth\ nell\ n$
using *nnth-nlast[of ntaken n nell] ntaken-nlength[of n nell]*
by (*metis min.absorb3 min.idem min.orderI nfinite-ntaken nnth-beyond not-less-iff-gr-or-eq*
ntaken-all ntaken-nnth the-enat.simps)

lemma *ntaken-nfirst*:
 $nfirst\ (ntaken\ n\ nell) = nfirst\ nell$
by *transfer (simp add: enat-0-iff(1))*

lemma *ntaken-nappend1*:

$n \leq \text{nlength } \text{nellx} \implies \text{ntaken } n (\text{nappend } \text{nellx } \text{nelly}) = \text{ntaken } n \text{nellx}$
by *transfer*
(simp add: zero-enat-def, metis eSuc-enat eSuc-epred eSuc-ile-mono llength-eq-0 ltake-lappend1)

lemma *ntaken-nappend2:*

$\text{nlength } \text{nellx} < (\text{enat } n) \implies$
 $\text{ntaken } n (\text{nappend } \text{nellx } \text{nelly}) = \text{nappend } \text{nellx} (\text{ntaken } (n - (\text{the-enat}(\text{nlength } \text{nellx})) - 1) \text{nelly})$
proof *(induct n arbitrary: nellx nelly)*
case 0
then show *?case* **using** *zero-enat-def* **by** *auto*
next
case *(Suc n)*
then show *?case*
proof *(cases nellx)*
case *(NNil x1)*
then show *?thesis*
by *simp*
next
case *(NCons x21 x22)*
then show *?thesis* **using** *Suc* **by** *simp*
(metis Extended-Nat.eSuc-mono eSuc-enat enat-ord-code(4) order-less-imp-not-less the-enat-eSuc)
qed
qed

lemma *ntaken-eq-ntaken-antimono:*

$\llbracket \text{ntaken } n \text{nellx} = \text{ntaken } n \text{nelly}; m \leq n \rrbracket \implies \text{ntaken } m \text{nellx} = \text{ntaken } m \text{nelly}$
by *(metis min.orderE ntaken-ntaken)*

lemma *ntake-eq-ntaken:*

assumes $(\text{enat } k) = m$
shows $\text{ntake } m \text{nell} = \text{ntaken } k \text{nell}$
using *assms apply transfer*
using *eSuc-enat* **by** *auto*

1.2.13 Concatenating a nonempty lazy list of nonempty lazy lists *nconcat*

lemma *nconcat-NNil [simp]:*

$\text{nconcat } (\text{NNil } \text{nell}) = \text{nell}$
by *transfer auto*

lemma *nconcat-NCons [simp]:*

$\text{nconcat } (\text{NCons } \text{nell } \text{nells}) = \text{nappend } \text{nell} (\text{nconcat } \text{nells})$
by *transfer auto*

lemma *nconcat-def2:*

$\text{nconcat} = \text{nellist-of-llist} \circ \text{lconcat} \circ (\text{lmap } \text{llist-of-nellist} \circ \text{llist-of-nellist})$
by *(simp add: map-fun-def nconcat-def)*

lemma *not-null-lconcat:*

$\neg \text{lnull}((\text{lconcat} \circ (\text{lmap } \text{llist-of-nellist} \circ \text{llist-of-nellist})) \text{nells})$

by (simp add: llist-of-nellist.code)

lemma nconcat-def3:

((llist-of-nellist \circ nconcat) nells) =
 ((lconcat \circ (lmap llist-of-nellist \circ llist-of-nellist)) nells)

proof –

let ?n2l = (lmap llist-of-nellist \circ llist-of-nellist)

have 1: llist-of-nellist \circ nconcat =
 llist-of-nellist \circ nellist-of-llist \circ lconcat \circ ?n2l

by (simp add: nconcat-def2 rewriteL-comp-comp)

have 2: \neg lnull((lconcat \circ ?n2l) nells)

using not-null-lconcat by blast

have 3: (llist-of-nellist \circ nellist-of-llist \circ lconcat \circ ?n2l) nells =
 (lconcat \circ ?n2l) nells

using 2 nellist-of-llist-inverse by simp

show ?thesis

by (metis 1 3)

qed

lemma nmap-lmap-inverse:

nmap nellist-of-llist (nellist-of-llist (lmap llist-of-nellist (llist-of-nellist nells))) = nells

proof –

let ?n2l = (lmap llist-of-nellist \circ llist-of-nellist)

have 1: nmap nellist-of-llist (nellist-of-llist (?n2l nells)) =
 nmap nellist-of-llist (nmap llist-of-nellist nells)

by (simp add: lmap-llist-of-nellist)

have 2: nmap nellist-of-llist (nmap llist-of-nellist nells) =
 nmap (nellist-of-llist \circ llist-of-nellist) nells

by (simp add: nellist.map-comp)

have 3: (nellist-of-llist \circ llist-of-nellist) = id

by auto

show ?thesis

by (metis 1 2 3 comp-apply nellist.map-id)

qed

lemma lmap-nmap-inverse:

assumes \neg lnull nells

\forall nell \in lset nells. \neg lnull nell

shows (lmap llist-of-nellist (llist-of-nellist (nmap nellist-of-llist (nellist-of-llist nells)))) = nells

proof –

let ?l2n = nmap nellist-of-llist \circ nellist-of-llist

have 0: (nmap nellist-of-llist (nellist-of-llist nells)) =
 nellist-of-llist (lmap nellist-of-llist nells)

using nmap-nellist-of-llist assms by simp

have 00: (llist-of-nellist (nellist-of-llist (lmap nellist-of-llist nells))) =
 (lmap nellist-of-llist nells)

using assms by simp

have 1: (lmap llist-of-nellist (llist-of-nellist (?l2n nells))) =
 (lmap llist-of-nellist ((lmap nellist-of-llist nells)))

using 0 00 by auto

```

have 2: (lmap llist-of-nellist ( ( lmap nellist-of-llist nells))) =
  (lmap (llist-of-nellist ∘ nellist-of-llist) nells)
  using llist.map-comp by blast
have 3:  $\forall \text{nell} \in \text{lset } \text{nells}. (\text{llist-of-nellist} \circ \text{nellist-of-llist}) \text{ nell} = \text{nell}$ 
  using assms by simp
have 4: (lmap (llist-of-nellist ∘ nellist-of-llist) nells) = nells
  using 3 by auto
show ?thesis
using 1 2 4 0 00 by presburger
qed

```

lemma nconcat-expand:

```

  nconcat nells =
    (if is-NNil nells then nfirst nells else nappend (nfirst nells) (nconcat (ntl nells)))
by (metis nconcat-NCons nconcat-NNil nellist.case-eq-if nellist.collapse(1) nellist.collapse(2)
  nfirst-def)

```

lemma lmap-llist-of-nellist-nmap:

```

(lmap (lmap f) (lmap llist-of-nellist (llist-of-nellist nells))) =
  (lmap llist-of-nellist (lmap (nmap f) (llist-of-nellist nells)))
proof –
have 5: (lmap (lmap f) (lmap llist-of-nellist (llist-of-nellist nells))) =
  (lmap ((lmap f) ∘ llist-of-nellist) (llist-of-nellist nells))
  using llist.map-comp by blast
have 6: (lmap ((lmap f) ∘ llist-of-nellist) (llist-of-nellist nells)) =
  (lmap (llist-of-nellist ∘ (nmap f)) (llist-of-nellist nells))
  by (simp add: lmap-llist-of-nellist)
have 7: (lmap (llist-of-nellist ∘ (nmap f)) (llist-of-nellist nells)) =
  (lmap llist-of-nellist (lmap (nmap f) (llist-of-nellist nells)))
  using llist.map-comp[of llist-of-nellist (nmap f) (llist-of-nellist nells)]
  by presburger
show ?thesis
using 5 6 7 by presburger
qed

```

lemma nmap-nconcat :

```

  nmap f (nconcat nells) = nconcat (nmap (nmap f) nells)
proof –
let ?n2l = (lmap llist-of-nellist ∘ llist-of-nellist)
have 1: nmap f (nconcat nells) =
  nmap f ((nellist-of-llist ∘ lconcat ∘ ?n2l) nells)
  by (simp add: nconcat-def2)
have 2: nmap f ((nellist-of-llist ∘ lconcat ∘ ?n2l) nells) =
  nellist-of-llist (lmap f ((lconcat ∘ ?n2l) nells))
  using nmap-nellist-of-llist
  using not-null-lconcat by fastforce
have 3: (lmap f ((lconcat ∘ ?n2l) nells)) =
  lconcat (lmap (lmap f) (?n2l nells))
  by (simp add: lmap-lconcat)

```

```

have 4: (nconcat (nmap ( nmap f ) nells)) =
  (nellist-of-llist ∘ lconcat ∘ ?n2l) (nmap ( nmap f ) nells)
by (simp add: nconcat-def2)
have 8: (lmap (lmap f) (?n2l nells)) =
  (lmap llist-of-nellist (lmap (nmap f) (llist-of-nellist nells)))
using lmap-llist-of-nellist-nmap by auto
show ?thesis
using 1 2 3 4 8
by (metis (no-types, opaque-lifting) comp-eq-dest-lhs llist.map-disc-iff llist-of-nellist-inverse-b
  llist-of-nellist-not-lnull lmap-lconcat lmap-llist-of-nellist-nmap nconcat-def3
  nellist-of-llist-inverse nmap-nellist-of-llist)
qed

```

lemma nconcat-eq-NNil:

```

  nconcat nells = (NNil x) ⟷ nells = (NNil (NNil x))
by (metis is-NNil-def is-NNil-nappend nconcat-NNil nconcat-expand)

```

lemma nhd-nconcat [simp]:

```

  ⟦ ¬ is-NNil nells; ¬ is-NNil (nhd nells) ⟧ ⟹ nhd (nconcat nells) = nhd (nhd nells)
by (metis nconcat-NCons nellist.collapse(2) nhd-nappend)

```

lemma ntl-nconcat [simp]:

```

  ⟦ ¬ is-NNil nells; ¬ is-NNil (nhd nells) ⟧ ⟹
    ntl (nconcat nells) = nappend (ntl (nhd nells)) (nconcat (ntl nells))
by (metis nconcat-NCons nellist.collapse(2) ntl-nappend)

```

lemma nconcat-nappend [simp]:

```

  assumes nfinite nells
  shows nconcat (nappend nells nells1) = nappend (nconcat nells) (nconcat nells1)
using assms
by (induct rule:nfinite-induct) (simp-all add: nappend-assoc)

```

lemma nconcat-eq-NCons-conv:

```

  nconcat nells = NCons x nell ⟷
    nells = (NNil (NCons x nell)) ∨
    (∃ nells'. nells = (NCons (NNil x) nells') ∧ nell = nconcat nells') ∨
    (∃ nell' nells''. nells = (NCons (NCons x nell') nells'') ∧ nell = nappend nell' (nconcat nells''))

```

proof (cases nells)

case (NNil nell1)

then show ?thesis **by** simp

next

case (NCons nell nell2)

then show ?thesis

proof (cases is-NNil nell)

case True

then show ?thesis

using NCons **by** simp

(metis nappend-NCons nappend-NNil nellist.collapse(1) nellist.sel(3) nellist.sel(5))

next


```

case False
then show ?thesis
  using NCons by simp
  (metis nappend-NCons nellist.collapse(2) nellist.distinct(1) nellist.sel(3) nellist.sel(5))
qed
qed

```

lemma *nlenght-nconcat*:

```

shows nlenght (nconcat nells) =
  (case nells of (NNil nell)  $\Rightarrow$  nlenght nell |
    (NCons nell nells1  $\Rightarrow$  eSuc(nlenght nell) + nlenght (nconcat nells1))
proof (cases nells)
case (NNil nell1)
then show ?thesis by simp
next
case (NCons nell nell2)
then show ?thesis by (simp add: eSuc-plus plus-1-eSuc(2))
qed

```

lemma *nlenght-nconcat-nfinite-conv-sum*:

```

assumes nfinite nells
shows nlenght (nconcat nells) =
  nlenght nells + ( $\sum i = 0..(the-enat (nlenght nells)). nlenght (nnth nells i)$ )
using assms
proof(induct rule: nfinite-induct)
case (NNil y)
then show ?case by (simp add: zero-enat-def) (simp add: enat-0-iff(2) nnth-NNil)
next
case (NCons nell nells)
then show ?case
  proof –
    have 1: nlenght (nconcat (NCons nell nells)) = 1+ nlenght nell + nlenght (nconcat nells)
      by simp
    have 2: nlenght (nconcat nells) =
      nlenght nells + ( $\sum i = 0..(the-enat (nlenght nells)). nlenght (nnth nells i)$ )
      using NCons.hyps(2) by blast
    have 3: nlenght (NCons x nells) = 1+ nlenght nells
      by (simp add: plus-1-eSuc(1))
    have 4: ( $\sum i = 0..the-enat (nlenght (NCons nell nells)). nlenght (nnth (NCons nell nells) i)$ ) =
      nlenght (nnth (NCons nell nells) 0) +
      ( $\sum i = 1..the-enat (nlenght (NCons nell nells)). nlenght (nnth (NCons nell nells) i)$ )
      by (simp add: sum.atLeast-Suc-atMost)
    have 5: ( $\sum i = 1..the-enat (nlenght (NCons nell nells)). nlenght (nnth (NCons nell nells) i)$ ) =
      ( $\sum i = 1..(Suc (the-enat (nlenght (nells))))). nlenght (nnth (NCons nell nells) i)$ )
      using NCons.hyps(1) eSuc-enat nfinite-nlenght-enat by fastforce
    have 6: ( $\sum i = 1..(Suc (the-enat (nlenght (nells))))). nlenght (nnth (NCons nell nells) i)$ ) =
      ( $\sum i = 0..(the-enat (nlenght (nells))))). nlenght (nnth (NCons nell nells) (Suc i))$ )
      using sum.shift-bounds-cl-nat-ivl[of  $\lambda i. nlenght (nnth (NCons nell nells) i)$  0 1
        ((the-enat (nlenght (nells)))))]

```

```

    by simp
  have 7:  $(\sum i = 0..(the-enat (nlength (nells)))) . nlength (nnth (NCons nell nells) (Suc i))) =$ 
     $(\sum i = 0..(the-enat (nlength (nells)))) . nlength (nnth (nells) (i)))$ 
    by auto
  show ?thesis
  using 2 3 4 5 6 by force
qed
qed

lemma nlength-nconcat-nfinite-conv-sum-alt:
  assumes nfinite nells
  shows  $nlength\ nells + (\sum i = 0..(the-enat (nlength\ nells)) . nlength (nnth\ nells\ i)) =$ 
     $epred(\sum i = 0..(the-enat (nlength\ nells)) . eSuc (nlength (nnth\ nells\ i)))$ 
  using assms
  proof(induct rule: nfinite-induct)
  case (NNil y)
  then show ?case by simp
  next
  case (NCons nell nells)
  then show ?case
    proof -
      have 1:  $(\sum i = 0..the-enat (nlength (NCons nell nells)) . nlength (nnth (NCons nell nells) i)) =$ 
         $nlength (nnth (NCons nell nells) 0) +$ 
         $(\sum i = 1..the-enat (nlength (NCons nell nells)) . nlength (nnth (NCons nell nells) i))$ 
        by (simp add: sum.atLeast-Suc-atMost)
      have 2:  $(\sum i = 1..the-enat (nlength (NCons nell nells)) . nlength (nnth (NCons nell nells) i)) =$ 
         $(\sum i = 1..(Suc (the-enat (nlength (nells)))) . nlength (nnth (NCons nell nells) i))$ 
        using NCons.hyps(1) eSuc-enat nfinite-nlength-enat by fastforce
      have 3:  $(\sum i = 1..(Suc (the-enat (nlength (nells)))) . nlength (nnth (NCons nell nells) i)) =$ 
         $(\sum i = 0..(the-enat (nlength (nells)))) . nlength (nnth (NCons nell nells) (Suc i)))$ 
        using sum.shift-bounds-cl-nat-ivl[of  $\lambda i . nlength (nnth (NCons nell nells) i)$  0 1
          ( $(the-enat (nlength (nells)))$ )]
        by simp
      have 4:  $(\sum i = 0..(the-enat (nlength (nells)))) . nlength (nnth (NCons nell nells) (Suc i))) =$ 
         $(\sum i = 0..(the-enat (nlength (nells)))) . nlength (nnth (nells) (i)))$ 
        by auto
      have 5:  $(\sum i = 0..(the-enat (nlength (NCons nell nells))) . eSuc (nlength (nnth (NCons nell nells) i)))$ 
      =
         $eSuc (nlength (nnth (NCons nell nells) 0)) +$ 
         $(\sum i = 1..(the-enat (nlength (NCons nell nells))) . eSuc (nlength (nnth (NCons nell nells) i)))$ 
        by (simp add: sum.atLeast-Suc-atMost)
      have 6:  $(\sum i = 1..(the-enat (nlength (NCons nell nells))) . eSuc (nlength (nnth (NCons nell nells) i)))$ 
      =
         $(\sum i = 1..(Suc (the-enat (nlength (nells)))) . eSuc (nlength (nnth (NCons nell nells) i)))$ 
        using NCons.hyps(1) eSuc-enat nfinite-nlength-enat by fastforce
      have 7:  $(\sum i = 1..(Suc (the-enat (nlength (nells)))) . eSuc (nlength (nnth (NCons nell nells) i))) =$ 
         $(\sum i = 0..(the-enat (nlength (nells))) . eSuc (nlength (nnth (NCons nell nells) (Suc i))))$ 
        using sum.shift-bounds-cl-nat-ivl[of  $\lambda i . eSuc(nlength (nnth (NCons nell nells) i))$  0 1
          ( $(the-enat (nlength (nells)))$ )]
        by simp
    end
  end

```

```

have 8: ( $\sum i = 0.. ( (the-enat (nlength ( nells))))$ ).  $eSuc (nlength (nnth (NCons nell nells) (Suc i))) =$ 
  ( $\sum i = 0.. ( (the-enat (nlength ( nells))))$ ).  $eSuc (nlength (nnth ( nells) ( i)))$ )
by auto
have 9: ( $\sum i = 0.. (the-enat (nlength (NCons nell nells)))$ ).  $eSuc (nlength (nnth (NCons nell nells) i))$ )
=
  ( $eSuc(nlength nell) +$ 
    ( $\sum i = 0.. ( (the-enat (nlength ( nells))))$ ).  $eSuc (nlength (nnth ( nells) ( i)))$ )
  by (metis 5 6 7 8 nnth-0)
have 10:  $epred(\sum i = 0.. (the-enat (nlength (NCons nell nells)))$ .  $eSuc (nlength (nnth (NCons nell nells)$ 
i))) =
   $epred( eSuc(nlength nell) +$ 
    ( $\sum i = 0.. ( (the-enat (nlength ( nells))))$ ).  $eSuc (nlength (nnth ( nells) ( i)))$ )
  using 9 by presburger
have 11:  $epred( eSuc(nlength nell) +$ 
    ( $\sum i = 0.. ( (the-enat (nlength ( nells))))$ ).  $eSuc (nlength (nnth ( nells) ( i)))$ ) =
     $eSuc(nlength nell) +$ 
     $epred(\sum i = 0.. ( (the-enat (nlength ( nells))))$ .  $eSuc (nlength (nnth ( nells) ( i)))$ )
  by (metis add.commute eSuc-ne-0 epred-iadd1 le-add1 le-add-same-cancel1 sum.last-plus
    zero-eq-add-iff-both-eq-0)
show ?thesis
using 1 11 2 3 9 NCons.hyps(2) eSuc-plus by fastforce
qed
qed

```

```

lemma nset-nconcat-nfinite:
assumes  $\forall xs \in nset\ nells. nfinite\ xs$ 
shows  $nset (nconcat\ nells) = (\bigcup xs \in nset\ nells. nset\ xs)$ 
proof –
let ?n2l = (lmap llist-of-nellist  $\circ$  llist-of-nellist)
have 1:  $nset (nconcat\ nells) =$ 
   $nset (((nlist-of-llist \circ lconcat \circ ?n2l)\ nells))$ 
  by (simp add: nconcat-def2)
have 2:  $nset (((nlist-of-llist \circ lconcat \circ ?n2l)\ nells)) =$ 
   $lset (llist-of-nellist (((nlist-of-llist \circ lconcat \circ ?n2l)\ nells)))$ 
  by (metis lset-llist-of-nellist-a)
have 3: (llist-of-nellist (((nlist-of-llist  $\circ$  lconcat  $\circ$  ?n2l) nells))) =
  ( $((lconcat \circ ?n2l)\ nells)$ )
  using nlist-of-llist-inverse not-null-lconcat by fastforce
have 4:  $lset (llist-of-nellist (((nlist-of-llist \circ lconcat \circ ?n2l)\ nells))) =$ 
   $lset ( (((lconcat \circ ?n2l)\ nells)))$ 
  using 3 by presburger
have 5:  $\forall nell \in lset ( ?n2l\ nells) . lfinite\ nell$ 
  using assms nfinite-def by fastforce
have 6:  $lset ( (((lconcat \circ ?n2l)\ nells))) = (\bigcup nell \in lset ( ?n2l\ nells). lset\ nell)$ 
  by (simp add: 5 lset-lconcat-lfinite)
have 7:  $(\bigcup nell \in lset ( ?n2l\ nells). lset\ nell) = \bigcup (nset\ ' nset\ nells)$ 
  by simp
show ?thesis
by (metis 6 7 lset-llist-of-nellist-a nconcat-def3 o-apply)

```

qed

lemma *nconcat-ntake*:

shows *nconcat* (*ntake* (*enat* *n*) *nells*) =

ntake (*n* + ($\sum i=0..n. \text{nlength } (\text{nnth } \text{nells } i)$)) (*nconcat* *nells*)

proof(*induct* *n* *arbitrary*: *nells*)

case 0 **thus** ?*case*

proof (*cases* *nells*)

case (*NNil* *nell*)

then show ?*thesis* **by** (*simp* *add*: *zero-enat-def[symmetric]* *nnth-NNil* *ntake-all*)

next

case (*NCons* *nell* *nell2*)

then show ?*thesis* **by** (*simp* *add*: *zero-enat-def[symmetric]*)

(*metis* *dual-order.refl* *nlast-NNil* *nnth-0* *ntake-all* *ntake-nappend1* *ntaken-0* *ntaken-nlast*)

qed

next

case (*Suc* *n*)

show ?*case*

proof (*cases* *nells*)

case (*NNil* *nell*)

then show ?*thesis*

by (*metis* (*no-types*, *lifting*) *dual-order.trans* *enat-le-plus-same(1)* *enat-le-plus-same(2)* *nconcat-NNil* *nfinite-NNil* *nlast-NNil* *nlength-NNil* *nnth-nlast* *ntake-NNil* *ntake-all* *sum.atLeast0-atMost-Suc-shift* *the-enat-0*)

next

case (*NCons* *nell* *nells1*)

then show ?*thesis*

proof –

have 1: *nconcat* (*ntake* (*enat* (*Suc* *n*)) *nells*) =

nappend *nell* (*nconcat* (*ntake* (*enat* *n*) *nells1*))

by (*metis* *NCons* *eSuc-enat* *nconcat-NCons* *ntake-Suc-NCons*)

have 2: (*nconcat* (*ntake* (*enat* *n*) *nells1*)) =

ntake (*enat* *n* + ($\sum i = 0..n. \text{nlength } (\text{nnth } \text{nells1 } i)$)) (*nconcat* *nells1*)

using *NCons* *Suc.hyps* *Suc.premis* *Suc-ile-eq* **by** *auto*

have 3: *nlength* (*nappend* *nell* (*nconcat* *nells1*)) =

nlength *nell* + 1 + *nlength* (*nconcat* *nells1*)

by *simp*

have 4: *nappend* *nell* (*ntake* (*enat* *n* + ($\sum i = 0..n. \text{nlength } (\text{nnth } \text{nells1 } i)$)) (*nconcat* *nells1*)) =

ntake (*nlength* *nell* + 1 + (*enat* *n* + ($\sum i = 0..n. \text{nlength } (\text{nnth } \text{nells1 } i)$))) (*nappend* *nell*

(*nconcat* *nells1*))

proof (*cases* *nlength* *nell* = ∞)

case *True*

then show ?*thesis* **by** (*metis* *enat-le-plus-same(2)* *ntake-all* *ntake-nappend1* *plus-enat-simps(2)*)

next

case *False*

then show ?*thesis* **by** (*simp* *add*: *ab-semigroup-add-class.add-ac(1)* *enat-0-iff(1)* *ntake-nappend2*)

qed

have 5: ($\sum i = 0..Suc\ n. \text{nlength } (\text{nnth } \text{nells } i)$) =

nlength (*nnth* *nells* 0) + ($\sum i = 1..Suc\ n. \text{nlength } (\text{nnth } \text{nells } i)$)

by (simp add: sum.atLeast-Suc-atMost)
 have 6: $\text{nlength } (\text{nnth nells } 0) = \text{nlength nell}$
 by (simp add: NCons)
 have 7: $(\sum i = 1.. \text{Suc } n. \text{nlength } (\text{nnth nells } i)) =$
 $(\sum i = 0..n. \text{nlength } (\text{nnth nells } (\text{Suc } i)))$
 using sum.shift-bounds-cl-nat-ivl[of $\lambda i. \text{nlength } (\text{nnth nells } i) \ 0 \ 1 \ n$]
 by simp
 have 8: $(\sum i = 0..n. \text{nlength } (\text{nnth nells } (\text{Suc } i))) =$
 $(\sum i = 0..n. \text{nlength } (\text{nnth nells1 } (i)))$
 using NCons by auto
 have 9: $\text{nlength nell} + 1 + (\text{enat } n + (\sum i = 0..n. \text{nlength } (\text{nnth nells1 } i))) =$
 $(\text{enat } (\text{Suc } n) + (\sum i = 0.. \text{Suc } n. \text{nlength } (\text{nnth nells } i)))$
 by (metis (no-types, lifting) 5 6 7 8 ab-semigroup-add-class.add-ac(1)
 add.left-commute eSuc-enat plus-1-eSuc(2))
 show ?thesis
 by (metis 1 2 4 9 NCons nconcat-NCons)
 qed
 qed
 qed

lemma *nnth-nconcat-conv*:

assumes $\text{enat } n \leq \text{nlength } (\text{nconcat nells})$
 shows $\exists m \ n'. \text{nnth } (\text{nconcat nells}) \ n = \text{nnth } (\text{nnth nells } m) \ n' \wedge \text{enat } n' \leq \text{nlength } (\text{nnth nells } m) \wedge$
 $\text{enat } m \leq \text{nlength nells} \wedge$
 $\text{enat } n = (\sum i < m. \text{eSuc}(\text{nlength } (\text{nnth nells } i))) + \text{enat } n'$

proof –

let $?n2l = (\text{lmap llist-of-nellist} \circ \text{llist-of-nellist})$
 have 1: $\bigwedge \text{nell}. \text{nlength nell} = \text{epred } (\text{llength } (\text{llist-of-nellist nell}))$
 unfolding nlength-def by auto
 have 2: $\bigwedge \text{nell } j. j \leq \text{nlength nell} \longrightarrow \text{nnth nell } j = \text{lnth } (\text{llist-of-nellist nell}) \ j$
 unfolding nnth-def by auto
 have 3: $\text{nlength } (\text{nconcat nells}) =$
 $\text{nlength } (((\text{nellist-of-llist} \circ \text{lconcat} \circ ?n2l) \text{ nells}))$
 by (simp add: nconcat-def2)
 have 4: $\text{nlength } (((\text{nellist-of-llist} \circ \text{lconcat} \circ ?n2l) \text{ nells})) =$
 $\text{epred } (\text{llength } (\text{llist-of-nellist } (((\text{nellist-of-llist} \circ \text{lconcat} \circ ?n2l) \text{ nells}))))$
 using nlength.rep-eq by blast
 have 5: $(\text{llist-of-nellist } (((\text{nellist-of-llist} \circ \text{lconcat} \circ ?n2l) \text{ nells}))) =$
 $((((\text{lconcat} \circ ?n2l) \text{ nells})))$
 using nellist-of-llist-inverse not-null-lconcat by fastforce
 have 6: $\text{epred } (\text{llength } (\text{llist-of-nellist } (((\text{nellist-of-llist} \circ \text{lconcat} \circ ?n2l) \text{ nells})))) =$
 $\text{epred } (\text{llength } (((\text{lconcat} \circ ?n2l) \text{ nells})))$
 using 5 by presburger
 have 7: $\text{enat } n < (\text{llength } (((\text{lconcat} \circ ?n2l) \text{ nells})))$
 by (metis (no-types, lifting) 3 4 5 assms co.enat.collapse iless-Suc-eq llength-eq-0
 llist-of-nellist-not-lnull)
 have 8: $(((\text{lconcat} \circ ?n2l) \text{ nells})) =$
 $\text{lconcat } (?n2l \text{ nells})$
 by simp
 have 9: $\exists m \ n'.$

$l\text{nth} (\text{lconcat} (?n2l \text{ nells})) n = l\text{nth} (l\text{nth} (?n2l \text{ nells}) m) n' \wedge$
 $\text{enat } n' < \text{llength} (l\text{nth} (?n2l \text{ nells}) m) \wedge \text{enat } m < \text{llength} (?n2l \text{ nells}) \wedge$
 $\text{enat } n = (\sum i < m. \text{llength} (l\text{nth} (?n2l \text{ nells}) i)) + \text{enat } n'$
using 7 *lnth-lconcat-conv*[of $n (?n2l \text{ nells})$]
by *fastforce*
obtain $m n'$ **where** 10: $l\text{nth} (\text{lconcat} (?n2l \text{ nells})) n = l\text{nth} (l\text{nth} (?n2l \text{ nells}) m) n' \wedge$
 $\text{enat } n' < \text{llength} (l\text{nth} (?n2l \text{ nells}) m) \wedge \text{enat } m < \text{llength} (?n2l \text{ nells}) \wedge$
 $\text{enat } n = (\sum i < m. \text{llength} (l\text{nth} (?n2l \text{ nells}) i)) + \text{enat } n'$
using 9 **by** *blast*
have 11: $l\text{nth} (\text{lconcat} (?n2l \text{ nells})) n = \text{nnth} (\text{nconcat } \text{nells}) n$
by (*metis* 2 5 8 *assms nconcat-def2*)
have 12: $l\text{nth} (l\text{nth} (?n2l \text{ nells}) m) n' = \text{nnth} (\text{nnth } \text{nells } m) n'$
by (*metis* 10 2 *comp-apply co.enat.collapse iless-Suc-eq llength-eq-0 llength-lmap*
llist-of-nellist-not-lnull lnth-lmap nlength.rep-eq)
have 13: $\text{llength} (l\text{nth} (?n2l \text{ nells}) m) = \text{eSuc} (\text{nlength} (\text{nnth } \text{nells } m))$
by (*metis* 10 2 *comp-apply co.enat.collapse iless-Suc-eq llength-eq-0 llength-lmap*
llist-of-nellist-not-lnull lnth-lmap nlength.rep-eq)
have 14: $\text{llength} (?n2l \text{ nells}) = \text{eSuc} (\text{nlength } \text{nells})$
by (*metis comp-apply co.enat.exhaust-sel llength-eq-0 llength-llist-of-nellist llength-lmap*
llist-of-nellist-not-lnull)
have 15: $\bigwedge i. i < m \implies \text{llength} (l\text{nth} (?n2l \text{ nells}) i) = \text{eSuc} (\text{nlength} (\text{nnth } \text{nells } i))$
proof –
fix i
assume $a: i < m$
show $\text{llength} (l\text{nth} (?n2l \text{ nells}) i) = \text{eSuc} (\text{nlength} (\text{nnth } \text{nells } i))$
proof –
have 151: $l\text{nth} (?n2l \text{ nells}) i = \text{llist-of-nellist} (\text{nnth } \text{nells } i)$
using a 10 14 2
by (*metis comp-apply enat-ord-simps(2) iless-Suc-eq llength-lmap lnth-lmap order-less-trans*)
have 152: $\text{llength} (\text{llist-of-nellist} (\text{nnth } \text{nells } i)) = \text{eSuc} (\text{nlength} (\text{nnth } \text{nells } i))$
using *epred-inject* **by** *force*
show *?thesis* **using** 151 152 **by** *presburger*
qed
qed
have 16: $(\sum i < m. \text{llength} (l\text{nth} (?n2l \text{ nells}) i)) = (\sum i < m. \text{eSuc} (\text{nlength} (\text{nnth } \text{nells } i)))$
by (*meson* 15 *lessThan-iff sum.cong*)
show *?thesis*
using 10 11 12 13 14 16 **by** (*metis iless-Suc-eq*)
qed

lemma *nnth-nconcat-ntake*:

assumes $\text{enat } w \leq \text{nlength} (\text{nconcat} (\text{ntake} (\text{enat } n) \text{ nells}))$
shows $\text{nnth} (\text{nconcat} (\text{ntake} (\text{enat } n) \text{ nells})) w = \text{nnth} (\text{nconcat } \text{nells}) w$
using *assms* **by** (*simp add: nconcat-ntake ntake-nnth*)

lemma *nfinite-nconcat [simp]*:

$n\text{finite} (\text{nconcat } \text{nells}) \longleftrightarrow n\text{finite } \text{nells} \wedge (\forall \text{nell} \in \text{nset } \text{nells}. n\text{finite } \text{nell})$
(is ?lhs \longleftrightarrow ?rhs)

proof

assume *?lhs*

```

thus ?rhs (is ?concl nells)
proof(induct nconcat nells arbitrary: nells rule: nfinite-induct)
case (NNil y)
then show ?case
by (metis is-NNil-imp-nfinite nconcat-eq-NNil nelist.discI(1) nelist.simps(20) singleton-iff)
next
case (NCons x nell)
then show ?case
  proof (cases nells)
  case (NNil nell1)
  then show ?thesis using NCons.hyps by auto
  next
  case (NCons nell1 nells1)
  then show ?thesis using NCons.hyps by simp
    (metis nconcat-NCons nelist.sel(5) nelist.set-intros(3) nfinite-NCons nfinite-nappend ntl-nappend)
  qed
qed
next
assume ?rhs
then obtain nfinite ( nells)
  and  $\forall nell \in \text{nset } nells. \text{nfinite } nell \dots$ 
thus ?lhs
proof(induct nells rule: nfinite-induct)
case (NNil nell)
then show ?case
by simp
next
case (NCons nell nells)
then show ?case
by (simp add: nfinite-nappend)
qed
qed

lemma nfilter-nconcat-nfinite-help:
assumes  $(\forall nell \in \text{nset } nells. (\exists x \in \text{nset } nell. P\ x))$ 
shows  $(\exists nell \in \text{nset } (nconcat\ nells). P\ nell)$ 
proof (cases nells)
case (NNil nell)
then show ?thesis using assms by simp
next
case (NCons nell nells1)
then show ?thesis using assms
  by simp (metis dual-order.trans enat-le-plus-same(1) in-nset-conv-nnth nlength-nappend nnth-nappend1)
qed

lemma nfilter-nconcat-nfinite:
assumes  $\forall nell \in \text{nset } nells. \text{nfinite } nell$ 
   $\forall nell \in \text{nset } nells. (\exists x \in \text{nset } nell. P\ x)$ 
shows  $\text{nfilter } P\ (nconcat\ nells) = nconcat\ (nmap\ (\text{nfilter } P)\ nells)$ 
proof –

```

```

let ?n2l = (lmap llist-of-nellist  $\circ$  llist-of-nellist)
have 0:  $\exists$  nell  $\in$  nset (nconcat nells).  $P$  nell
  using assms nfilter-nconcat-nfinite-help by blast
have 1: nset nells = lset(llist-of-nellist nells)
  by simp
have 2:  $\bigwedge$  nell. nell  $\in$  lset(llist-of-nellist nells)  $\longrightarrow$  lfinite (llist-of-nellist nell)
  using 1 assms nfinite-def by blast
have 3:  $\bigwedge$  nell  $Q$ . ( $\exists x \in$  nset nell.  $Q$   $x$ )  $\longrightarrow$ 
  nfilter  $Q$  nell = nellist-of-llist (lfilter  $Q$  (llist-of-nellist nell))
  unfolding nfilter-def by simp
have 4: nfilter  $P$  (nconcat nells) =
  nfilter  $P$  (((nellist-of-llist  $\circ$  lconcat  $\circ$  ?n2l) nells))
  by (simp add: nconcat-def2)
have 5: nfilter  $P$  (((nellist-of-llist  $\circ$  lconcat  $\circ$  ?n2l) nells)) =
  nellist-of-llist (lfilter  $P$  (llist-of-nellist (((nellist-of-llist  $\circ$  lconcat  $\circ$  ?n2l) nells))))
  by (metis 3 0 nconcat-def2)
have 6: (llist-of-nellist (((nellist-of-llist  $\circ$  lconcat  $\circ$  ?n2l) nells))) =
  ( ((( lconcat  $\circ$  ?n2l) nells)))
  using nellist-of-llist-inverse not-null-lconcat by fastforce
have 7: (lfilter  $P$  (llist-of-nellist (((nellist-of-llist  $\circ$  lconcat  $\circ$  ?n2l) nells)))) =
  (lfilter  $P$  ( ((( lconcat  $\circ$  ?n2l) nells))))
  using 6 by presburger
have 8: ((( lconcat  $\circ$  ?n2l) nells)) = lconcat (?n2l nells)
  by simp
have 9:  $\forall$  nell  $\in$  lset (?n2l nells). lfinite nell
  by (simp add: 2)
have 10: (lfilter  $P$  ( ((( lconcat  $\circ$  ?n2l) nells)))) = lconcat (lmap (lfilter  $P$ ) ( ?n2l nells))
  by (simp add: 9 lfilter-lconcat-lfinite)
have 11: nconcat (nmap (nfilter  $P$ ) nells) =
  (((nellist-of-llist  $\circ$  lconcat  $\circ$  ?n2l) (nmap (nfilter  $P$ ) nells)))
  by (simp add: nconcat-def2)
have 12:  $\bigwedge$  nell  $f$ . nmap  $f$  nell = nellist-of-llist (lmap  $f$  (llist-of-nellist nell))
  by (metis llist-of-nellist-inverse-b llist-of-nellist-not-lnull nmap-nellist-of-llist)
have 13: (nmap (nfilter  $P$ ) nells) = nellist-of-llist (lmap (nfilter  $P$ ) (llist-of-nellist nells))
  using 12 by blast
have 14: nellist-of-llist (lmap (nfilter  $P$ ) (llist-of-nellist nells)) =
  nellist-of-llist (lmap ( $\lambda$ ys. nellist-of-llist (lfilter  $P$  (llist-of-nellist ys))) (llist-of-nellist nells))
  by (metis (mono-tags, lifting) 1 3 assms(2) llist.map-cong)
have 15: lmap (lfilter  $P$ ) (lmap llist-of-nellist (llist-of-nellist nells)) =
  lmap ((lfilter  $P$ )  $\circ$  llist-of-nellist) (llist-of-nellist nells)
  using llist.map-comp by blast
have 16: ( ( ?n2l (nmap (nfilter  $P$ ) nells))) =
  ( ( ?n2l (nellist-of-llist (lmap (nfilter  $P$ ) (llist-of-nellist nells))))))
  using 13 by presburger
have 17: ( ( ?n2l (nellist-of-llist (lmap (nfilter  $P$ ) (llist-of-nellist nells)))))) =
  ( ( ?n2l (nellist-of-llist
    (lmap ( $\lambda$ ys. nellist-of-llist (lfilter  $P$  (llist-of-nellist ys))) (llist-of-nellist nells))))))
  using 14 by presburger
have 18: ( ( ?n2l (nellist-of-llist
    (lmap ( $\lambda$ ys. nellist-of-llist (lfilter  $P$  (llist-of-nellist ys))) (llist-of-nellist nells)))))) =

```



```

    ( (lmap llist-of-nellist
      ( ( (lmap (λys. nellist-of-llist (lfilter P (llist-of-nellist ys))) (llist-of-nellist nells))))))
  by simp
have 19: ( (lmap llist-of-nellist
  ( ( (lmap (λys. nellist-of-llist (lfilter P (llist-of-nellist ys))) (llist-of-nellist nells)))))) =
  ( (lmap (llist-of-nellisto(λys. nellist-of-llist (lfilter P (llist-of-nellist ys))) (llist-of-nellist nells)))
    using llist.map-comp by blast
have 20: ∧ nell. nell ∈ lset(llist-of-nellist nells) → ¬lnull (lfilter P (llist-of-nellist nell))
  by (simp add: assms(2))
have 21: ( (lmap (llist-of-nellisto(λys. nellist-of-llist (lfilter P (llist-of-nellist ys))) (llist-of-nellist nells)))
  =
    ( (lmap ((λys. (lfilter P (llist-of-nellist ys))) (llist-of-nellist nells)))
      by (metis (no-types, lifting) 20 comp-def llist.map-cong0 nellist-of-llist-inverse)
have 22: ( (lmap llist-of-nellist (llist-of-nellist (nmap (nfilter P) nells)))) =
  (lmap (lfilter P) ( ((( (lmap llist-of-nellist o llist-of-nellist)) nells))))
  using 13 14 15 19 21 by force
have 23: nellist-of-llist (lconcat (lmap (lfilter P) ( ?n2l nells))) =
  nconcat (nmap (nfilter P) nells)
  by (simp add: 22 nconcat-def2)
show ?thesis
by (metis 10 23 5 6 nconcat-def2)
qed

```

lemma *nconcat-nmap-singleton* [simp]:

nconcat (nmap (λx. NNil (f x)) nell) = nmap f nell

proof –

let ?n2l = (lmap llist-of-nellist o llist-of-nellist)

have 1: *nconcat (nmap (λx. NNil (f x)) nell) =*
((nlist-of-llist o lconcat o ?n2l) (nmap (λx. NNil (f x)) nell))

by (simp add: nconcat-def2)

have 2: *(nmap (λx. NNil (f x)) nell) =*
nlist-of-llist (lmap (λx. NNil (f x)) (llist-of-nellist nell))

by (simp add: lmap-llist-of-nellist)

have 3: *nlist-of-llist (lmap (λx. NNil (f x)) (llist-of-nellist nell)) =*
nlist-of-llist (lmap (λx. nellist-of-llist (LCons (f x) LNil)) (llist-of-nellist nell))

by force

have 4: *nmap f nell = nlist-of-llist (lmap f (llist-of-nellist nell))*

by (simp add: lmap-llist-of-nellist)

have 5: *lconcat (?n2l (nlist-of-llist*
(lmap (λx. nellist-of-llist (LCons (f x) LNil)) (llist-of-nellist nell)))) =
lconcat (lmap llist-of-nellist (lmap (λz. NNil (f z)) (llist-of-nellist nell)))

by simp

have 6: *(lmap llist-of-nellist (lmap (λz. NNil (f z)) (llist-of-nellist nell))) =*
(lmap (llist-of-nellisto(λz. NNil (f z))) (llist-of-nellist nell))

using *llist.map-comp* **by** *metis*

have 7: *(lmap (llist-of-nellisto(λz. NNil (f z))) (llist-of-nellist nell)) =*
(lmap (λz. LCons (f z) LNil) (llist-of-nellist nell))

by *auto*

have 8: *lconcat (?n2l (nlist-of-llist*
(lmap (λx. nellist-of-llist (LCons (f x) LNil)) (llist-of-nellist nell))))

```

      = (lmap f (llist-of-nellist nell))
    using 6 by force
  have 9: (((nellist-of-llist ∘ lconcat ∘ ?n2l) (nmap (λx. NNil (f x) ) nell))) =
    nellist-of-llist (lmap f (llist-of-nellist nell))
    using 2 8 by auto
  show ?thesis
  using 1 4 9 by presburger
qed

```

lemma *nset-nconcat-subset*:

$$nset (nconcat nells) \subseteq (\bigcup_{nell \in nset \ nells} nset \ nell)$$

proof –

```

let ?n2l = (lmap llist-of-nellist ∘ llist-of-nellist)
have 1: nset (nconcat nells) =
  nset (((nellist-of-llist ∘ lconcat ∘ ?n2l) nells))
  by (simp add: nconcat-def2)
have 2: nset (((nellist-of-llist ∘ lconcat ∘ ?n2l) nells)) =
  lset (llist-of-nellist (((nellist-of-llist ∘ lconcat ∘ ?n2l) nells)))
  by (metis lset-llist-of-nellist-a)
have 3: (llist-of-nellist (((nellist-of-llist ∘ lconcat ∘ ?n2l) nells))) =
  ( ((( lconcat ∘ ?n2l) nells)))
  using nellist-of-llist-inverse not-null-lconcat by fastforce
have 4: lset (llist-of-nellist (((nellist-of-llist ∘ lconcat ∘ ?n2l) nells))) =
  lset ( ((( lconcat ∘ ?n2l) nells)))
  using 3 by presburger
have 5: lset ( ((( lconcat ∘ ?n2l) nells))) ⊆
  (⋃_{nell ∈ lset (?n2l nells). lset nell}
  using lset-lconcat-subset by fastforce
have 6: (⋃_{nell ∈ lset (?n2l nells). lset nell} =
  ⋃ (nset ‘ nset nells)
  by simp
show ?thesis
by (metis 1 2 3 5 6)
qed

```

lemma *ndistinct-nconcat*:

assumes *ndistinct nells*

$$\bigwedge_{nell. nell \in nset \ nells} \implies ndistinct \ nell$$

$$\bigwedge_{nell \ nell1. \llbracket nell \in nset \ nells; nell1 \in nset \ nells; nell \neq nell1 \rrbracket \implies nset \ nell \cap nset \ nell1 = \{\}}$$

shows *ndistinct (nconcat nells)*

proof –

```

let ?n2l = (lmap llist-of-nellist ∘ llist-of-nellist)
have 1: ldistinct (llist-of-nellist nells)
  using assms(1) ndistinct.rep-eq by auto
have 2: ⋀_{nell. nell ∈ lset (llist-of-nellist nells)} ⟹ ldistinct (llist-of-nellist nell)
  using assms(2) ndistinct.rep-eq by auto
have 3: ⋀_{nell nell1. ⌈ nell ∈ lset (llist-of-nellist nells); nell1 ∈ lset (llist-of-nellist nells); nell ≠ nell1 ⌋}
  ⟹ lset (llist-of-nellist nell) ∩ lset (llist-of-nellist nell1) = {}
  using assms(3) by simp
have 4: ndistinct (nconcat nells) =

```

```

      ndistinct (((nellist-of-llist  $\circ$  lconcat  $\circ$  ?n2l) nells))
    by (simp add: nconcat-def2)
  have 5: ndistinct (((nellist-of-llist  $\circ$  lconcat  $\circ$  ?n2l) nells)) =
    ldistinct (llist-of-nellist (((nellist-of-llist  $\circ$  lconcat  $\circ$  ?n2l) nells)))
    using ndistinct.rep-eq by blast
  have 6: (llist-of-nellist (((nellist-of-llist  $\circ$  lconcat  $\circ$  ?n2l) nells))) =
    ( ((( lconcat  $\circ$  ?n2l) nells) ))
    using nellist-of-llist-inverse not-null-lconcat by fastforce
  have 7: ldistinct (llist-of-nellist (((nellist-of-llist  $\circ$  lconcat  $\circ$  ?n2l) nells))) =
    ldistinct ( ((( lconcat  $\circ$  ?n2l) nells) ))
    using 6 by presburger
  have 8: ldistinct (?n2l nells)
    by (metis 1 bi-unique-cr-nellist-help comp-eq-dest-lhs inj-on-def ldistinct-lmap)
  have 9:  $\bigwedge$  nell. nell  $\in$  lset (?n2l nells)  $\implies$  ldistinct nell
    using 2 by auto
  have 10:  $\bigwedge$  nell nell1.  $\llbracket$  nell  $\in$  lset (?n2l nells);
    nell1  $\in$  lset (?n2l nells); nell  $\neq$  nell1  $\rrbracket \implies$ 
    lset nell  $\cap$  lset nell1 =  $\{\}$ 
    by (metis (no-types, lifting) assms(3) comp-def imageE lset-llist-of-nellist-a lset-lmap)
  have 11: ldistinct (lconcat (?n2l nells))
    using 10 8 9 ldistinct-lconcat by blast
  show ?thesis
    using 11 4 5 6 by fastforce
qed

```

1.2.14 nellist-all2

lemmas nellist-all2-NNil = nellist.rel-inject(1)

lemmas nellist-all2-NCons = nellist.rel-inject(2)

lemma nellist-all2-NNil1:

$nellist\text{-}all2\ Q\ (NNil\ b)\ nell \longleftrightarrow (\exists b'.\ nell = NNil\ b' \wedge Q\ b\ b')$

using nellist.rel-cases **by** fastforce

lemma nellist-all2-NNil2:

$nellist\text{-}all2\ Q\ nell\ (NNil\ b') \longleftrightarrow (\exists b.\ nell = NNil\ b \wedge Q\ b\ b')$

using nellist.rel-sel

by (metis is-NNil-def nellist-all2-NNil)

lemma nellist-all2-NCons1:

$nellist\text{-}all2\ P\ (NCons\ x\ nell)\ nell' \longleftrightarrow$
 $(\exists x'\ nell''.\ nell' = NCons\ x'\ nell'' \wedge P\ x\ x' \wedge nellist\text{-}all2\ P\ nell\ nell'')$

using nellist.rel-sel

by (metis nellist.collapse(2) nellist.disc(2) nellist.sel(3) nellist.sel(5))

lemma nellist-all2-NCons2:

$nellist\text{-}all2\ P\ nell'\ (NCons\ x\ nell) \longleftrightarrow$
 $(\exists x'\ nell''.\ nell' = NCons\ x'\ nell'' \wedge P\ x'\ x \wedge nellist\text{-}all2\ P\ nell''\ nell)$

by (metis nellist.collapse(2) nellist.disc(2) nellist.rel-sel nellist.sel(3) nellist.sel(5))

lemma *nellist-all2-coinduct* [consumes 1, case-names *ilist-all2*,
case-conclusion *nellist-all2 is-NNil NNil NCons*,
coinduct pred: *nellist-all2*]:
assumes $X\ n\ell x\ n\ell y$
and $\bigwedge n\ell i x\ n\ell i y.$
 $X\ n\ell i x\ n\ell i y \implies$
 $(is-NNil\ n\ell i x = is-NNil\ n\ell i y) \wedge$
 $(is-NNil\ n\ell i x \longrightarrow is-NNil\ n\ell i y \longrightarrow P\ (nlast\ n\ell i x)\ (nlast\ n\ell i y)) \wedge$
 $(\neg is-NNil\ n\ell i x \longrightarrow \neg is-NNil\ n\ell i y \longrightarrow P\ (nhd\ n\ell i x)\ (nhd\ n\ell i y) \wedge$
 $(X\ (ntl\ n\ell i x)\ (ntl\ n\ell i y) \vee nellist-all2\ P\ (ntl\ n\ell i x)\ (ntl\ n\ell i y)))$
shows *nellist-all2 P nℓx nℓy*
using *assms*
nellist.rel-coinduct[of $(\lambda\ n\ell x\ n\ell y. X\ n\ell x\ n\ell y \vee nellist-all2\ P\ n\ell x\ n\ell y)\ n\ell x\ n\ell y\ P]$
by (*metis nellist.rel-sel*)

lemma *nellist-all2-cases*[consumes 1, case-names *NNil NCons*, cases *pred*]:
assumes *nellist-all2 P nℓx nℓy*
obtains $(NNil)\ b\ b'$ **where** $n\ell x = NNil\ b\ n\ell y = NNil\ b'\ P\ b\ b'$
 $| (NCons)\ x\ n\ell x'\ y\ n\ell y'$
where $n\ell x = NCons\ x\ n\ell x'\ \mathbf{and}\ n\ell y = NCons\ y\ n\ell y'$
and $P\ x\ y\ \mathbf{and}\ nellist-all2\ P\ n\ell x'\ n\ell y'$
using *assms*
using *nellist.rel-cases* **by** *blast*

lemma *nellist-all2-nmap*:
 $nellist-all2\ P\ (nmap\ f\ n\ell x)\ n\ell y \longleftrightarrow nellist-all2\ (\lambda x\ y. P\ (f\ x)\ y)\ n\ell x\ n\ell y$
using *nellist.rel-map*(1) **by** *blast*

lemma *nellist-all2-nmap2*:
 $nellist-all2\ P\ n\ell x\ (nmap\ f\ n\ell y) \longleftrightarrow nellist-all2\ (\lambda x\ y. P\ x\ (f\ y))\ n\ell x\ n\ell y$
using *nellist.rel-map*(2) **by** *blast*

lemma *nellist-all2-mono*:
 $\llbracket nellist-all2\ P\ n\ell x\ n\ell y; \bigwedge x\ y. P\ x\ y \implies P'\ x\ y \rrbracket$
 $\implies nellist-all2\ P'\ n\ell x\ n\ell y$
using *nellist.rel-mono-strong* **by** *blast*

lemma *nellist-all2-nlengthD*:
 $nellist-all2\ P\ n\ell x\ n\ell y \implies nlength\ n\ell x = nlength\ n\ell y$
by(*transfer*)(*auto dest: llist-all2-llengthD*)

lemma *nellist-all2-nfiniteD*:
 $nellist-all2\ P\ n\ell x\ n\ell y \implies nfinite\ n\ell x = nfinite\ n\ell y$
by *transfer*
(*auto dest: llist-all2-lfiniteD*)

lemma *nellist-all2-nfinite1-nlastD*:
 $\llbracket nellist-all2\ P\ n\ell x\ n\ell y; nfinite\ n\ell x \rrbracket \implies P\ (nlast\ n\ell x)\ (nlast\ n\ell y)$
by (*frule nellist-all2-nfiniteD*)
(*transfer*,

auto simp add: llist-all2-conv-all-lnth,
metis Suc-ile-eq eSuc-enat lfinite.simps lfinite-llength-enat llast-conv-lnth llength-LCons
llist.discI(1) order-refl)

lemma *nellist-all2-nfinite2-nlastD:*

$\llbracket \text{nellist-all2 } P \text{ nellx nelly; nfinite nelly} \rrbracket \implies P (\text{nlast nellx}) (\text{nlast nelly})$
by(*metis nellist-all2-nfinite1-nlastD nellist-all2-nfiniteD*)

lemma *nellist-all2D-llist-all2-llist-of-nellist:*

$\text{nellist-all2 } P \text{ nellx nelly} \implies \text{llist-all2 } P (\text{llist-of-nellist nellx}) (\text{llist-of-nellist nelly})$
by *transfer*
(simp add: nellist-all2-help-b)

lemma *nellist-all2-is-NNilD:*

$\text{nellist-all2 } P \text{ nellx nelly} \implies \text{is-NNil nellx} \longleftrightarrow \text{is-NNil nelly}$
by (*cases nellx*) (*auto simp add: nellist-all2-NNil1 nellist-all2-NCons1*)

lemma *nellist-all2-nhdD:*

$\llbracket \text{nellist-all2 } P \text{ nellx nelly; } \neg \text{is-NNil nellx} \vee \neg \text{is-NNil nelly} \rrbracket \implies P (\text{nhd nellx}) (\text{nhd nelly})$
by (*cases nellx*) (*auto simp add: nellist-all2-NNil1 nellist-all2-NCons1*)

lemma *nellist-all2-ntlI:*

$\llbracket \text{nellist-all2 } P \text{ nellx nelly; } \neg \text{is-NNil nellx} \vee \neg \text{is-NNil nelly} \rrbracket \implies$
 $\text{nellist-all2 } P (\text{ntl nellx}) (\text{ntl nelly})$
by (*cases nellx*) (*auto simp add: nellist-all2-NNil1 nellist-all2-NCons1*)

lemma *nellist-all2-refl:*

$\text{nellist-all2 } P \text{ nell nell} \longleftrightarrow$
 $(\forall x \in \text{nset nell. } P x x) \wedge (\text{nfinite nell} \longrightarrow P (\text{nlast nell}) (\text{nlast nell}))$
by *transfer*
(auto, metis in-lset-lappend-iff lappend-lbutlast-llast-id-lfinite lfinite-lappend
llist.set-intros(1))

lemma *nellist-all2-reflI:*

$\llbracket \bigwedge x. x \in \text{nset nell} \implies P x x; \text{nfinite nell} \implies P (\text{nlast nell}) (\text{nlast nell}) \rrbracket$
 $\implies \text{nellist-all2 } P \text{ nell nell}$
by(*simp add: nellist-all2-refl*)

lemma *nellist-all2-conv-all-nnth-help1:*

$\neg \text{lnull nellx} \implies \neg \text{lnull nelly} \implies \text{lfinite nelly} \implies \text{llist-all2 } P \text{ nellx nelly} \implies$
 $P (\text{llast nellx}) (\text{llast nelly})$

proof –

assume *a1: lfinite nelly*

assume *a2: llist-all2 P nellx nelly*

assume *a3: $\neg \text{lnull nellx}$*

assume *a4: $\neg \text{lnull nelly}$*

have *f5: lfinite (lappend (lbutlast nellx) (LCons (llast nellx) LNil))*

using *a3 a2 a1 by (simp add: llist-all2-lfiniteD)*

have *f6: llength (ltl nellx) = epred (llength nelly)*

using *a2 by (metis (no-types) epred-llength llist-all2-llengthD)*

have $f7$: $\text{lappend } (\text{lbutlast } \text{nelly}) (\text{LCons } (\text{llast } \text{nelly}) \text{ LNil}) = \text{nelly}$
using $a4$ **by** ($\text{meson } \text{lappend-lbutlast-lldrop}$)
have $\text{llength } (\text{lbutlast } \text{nellx}) = \text{llength } (\text{ltl } \text{nellx})$
using epred-llength **by** auto
then show $?thesis$
using $f7 f6 f5 a3 a2 a1$
by ($\text{metis } (\text{no-types}) \text{lappend-eq-lappend-conv } \text{lappend-lbutlast-lldrop-lldrop}$
 $\text{lbutlast-conv-ltake } \text{lfinite-lappend } \text{lhs-LCons } \text{llist.disc}(2) \text{ llist-all2-lappend1D}(2)$
 llist-all2-lhdD2)
qed

lemma $\text{nellist-all2-conv-all-nnth}$:

$\text{nellist-all2 } P \text{ nellx } \text{nelly} \longleftrightarrow$
 $\text{nlength } \text{nellx} = \text{nlength } \text{nelly} \wedge$
 $(\forall n. \text{enat } n \leq \text{nlength } \text{nellx} \longrightarrow P (\text{nnth } \text{nellx } n) (\text{nnth } \text{nelly } n))$

by transfer

$(\text{auto simp add: llist-all2-llengthD},$
 $\text{metis eSuc-epred iless-Suc-eq llength-eq-0 llist-all2-lnthD2},$
 $\text{metis eSuc-epred iless-Suc-eq llength-eq-0 llist-all2-conv-all-lnth})$

lemma nellist-all2-True [simp]:

$\text{nellist-all2 } (\lambda - . \text{True}) \text{ nellx } \text{nelly} \longleftrightarrow \text{nlength } \text{nellx} = \text{nlength } \text{nelly}$

by ($\text{simp add: nellist-all2-conv-all-nnth}$)

lemma $\text{nellist-all2-nnthD}$:

$\llbracket \text{nellist-all2 } P \text{ nellx } \text{nelly}; \text{enat } n \leq \text{nlength } \text{nellx} \rrbracket \Longrightarrow P (\text{nnth } \text{nellx } n) (\text{nnth } \text{nelly } n)$

by ($\text{simp add: nellist-all2-conv-all-nnth}$)

lemma $\text{nellist-all2-nnthD2}$:

$\llbracket \text{nellist-all2 } P \text{ nellx } \text{nelly}; \text{enat } n \leq \text{nlength } \text{nelly} \rrbracket \Longrightarrow P (\text{nnth } \text{nellx } n) (\text{nnth } \text{nelly } n)$

by ($\text{simp add: nellist-all2-conv-all-nnth}$)

lemmas $\text{nellist-all2-eq} = \text{nellist.rel-eq}$

lemma $\text{nmap-eq-nmap-conv-nellist-all2}$:

$\text{nmap } f \text{ nellx} = \text{nmap } f' \text{ nelly} \longleftrightarrow$
 $\text{nellist-all2 } (\lambda x y. f x = f' y) \text{ nellx } \text{nelly}$

by transfer

$(\text{clarsimp simp add: lmap-eq-lmap-conv-llist-all2})$

lemma $\text{nellist-all2-trans}$:

$\llbracket \text{nellist-all2 } P \text{ nellx } \text{nelly}; \text{nellist-all2 } P \text{ nelly } \text{nellz}; \text{transp } P \rrbracket$
 $\Longrightarrow \text{nellist-all2 } P \text{ nellx } \text{nellz}$

by $\text{transfer } (\text{auto elim: llist-all2-trans dest: llist-all2-lfiniteD transpD})$

lemma $\text{nellist-all2-nappendI}$:

$\llbracket \text{nellist-all2 } P \text{ nellx } \text{nelly};$
 $\llbracket \text{nfinite } \text{nellx}; \text{nfinite } \text{nelly}; P (\text{nlast } \text{nellx}) (\text{nlast } \text{nelly}) \rrbracket$
 $\Longrightarrow \text{nellist-all2 } P \text{ nellx}' \text{ nelly}' \rrbracket$

$\implies \text{nellist-all2 } P \text{ (nappend nellx nellx')} \text{ (nappend nelly nelly')}$
by *transfer*
(auto simp add: lnull-def lappend-eq-lappend-conv nellist-all2-conv-all-nnth-help1
intro: llist-all2-lappendI)

lemma *nested-nellist-all2-nested-llist-all2:*
nellist-all2 (nellist-all2 A) nells nells1 =
llist-all2 (llist-all2 A) (lmap llist-of-nellist (llist-of-nellist nells))
(lmap llist-of-nellist (llist-of-nellist nells1))

proof –
have 1: *nellist-all2 (nellist-all2 A) nells nells1 =*
llist-all2 (nellist-all2 A) (llist-of-nellist nells) (llist-of-nellist nells1)
using *nellist-all2-help-a nellist-all2-help-b* **by** *blast*
have 2: *(λ xx yy. nellist-all2 A xx yy) =*
(λ xx yy. llist-all2 A (llist-of-nellist xx) (llist-of-nellist yy))
by *(meson nellist-all2D-llist-all2-llist-of-nellist nellist-all2-help-a)*
have 3: *llist-all2 (nellist-all2 A) (llist-of-nellist nells) (llist-of-nellist nells1) =*
llist-all2 (λ xx yy. llist-all2 A (llist-of-nellist xx) (llist-of-nellist yy))
(llist-of-nellist nells) (llist-of-nellist nells1)
using 2 **by** *auto*
have 6: *llist-all2 (λ xx yy. llist-all2 A (llist-of-nellist xx) (llist-of-nellist yy))*
(llist-of-nellist nells) (llist-of-nellist nells1) =
llist-all2 (llist-all2 A) (lmap llist-of-nellist (llist-of-nellist nells))
(lmap llist-of-nellist (llist-of-nellist nells1))
using *llist-all2-lmap1[of (llist-all2 A) llist-of-nellist (llist-of-nellist nells)*
(lmap llist-of-nellist (llist-of-nellist nells1))]
llist-all2-lmap2[of (llist-all2 A) - llist-of-nellist (llist-of-nellist nells1)]
by *(simp add: llist-all2-conv-all-lnth)*
show *?thesis*
using 1 3 6 **by** *blast*
qed

lemma *nellist-all2-nconcatI:*
assumes *nellist-all2 (nellist-all2 A) nells nells1*
shows *nellist-all2 A (nconcat nells) (nconcat nells1)*
proof –
have 3: *nellist-all2 A (nconcat nells) (nconcat nells1) =*
llist-all2 A (llist-of-nellist (nconcat nells)) (llist-of-nellist (nconcat nells1))
using *nellist-all2-help-a nellist-all2-help-b* **by** *blast*
have 4: *(llist-of-nellist (nconcat nells)) =*
((lconcat ∘ (lmap llist-of-nellist ∘ llist-of-nellist)) nells)
using *nconcat-def3* **by** *auto*
have 5: *(llist-of-nellist (nconcat nells1)) =*
((lconcat ∘ (lmap llist-of-nellist ∘ llist-of-nellist)) nells1)
using *nconcat-def3* **by** *auto*
have 7: *llist-all2 (llist-all2 A) (lmap llist-of-nellist (llist-of-nellist nells))*
(lmap llist-of-nellist (llist-of-nellist nells1)) \implies
llist-all2 A ((lconcat ∘ (lmap llist-of-nellist ∘ llist-of-nellist)) nells)
((lconcat ∘ (lmap llist-of-nellist ∘ llist-of-nellist)) nells1)
using *llist-all2-lconcatI[of A (lmap llist-of-nellist (llist-of-nellist nells))*

```

      (lmap llist-of-nellist (llist-of-nellist nells1)) ]
    by simp
  have 8:
    llist-all2 A ((lconcat ∘ (lmap llist-of-nellist ∘ llist-of-nellist)) nells)
      ((lconcat ∘ (lmap llist-of-nellist ∘ llist-of-nellist)) nells1)
    = nellist-all2 A (nconcat nells) (nconcat nells1)
  using 3 4 5 by presburger
  have 9: nellist-all2 (nellist-all2 A) nells nells1 =
    llist-all2 (llist-all2 A) (lmap llist-of-nellist (llist-of-nellist nells))
      (lmap llist-of-nellist (llist-of-nellist nells1))
  using nested-nellist-all2-nested-llist-all2 by blast
  show ?thesis using assms 7 8 9
  by blast
qed

```

lemma *nlength-nconcat-eqI*:

```

  fixes nells :: 'a nellist nellist and nells1 :: 'b nellist nellist
  assumes nellist-all2 (λxs ys. nlength xs = nlength ys) nells nells1
  shows nlength (nconcat nells) = nlength (nconcat nells1)
proof –
  have nellist-all2 (nellist-all2 (λa b. True)) nells nells1
  using assms nellist.rel-mono-strong nellist-all2-True by blast
  then show ?thesis using nellist-all2-nconcatI nellist-all2-nlengthD by blast
qed

```

lemma *llist-all2-nellist-of-llistI*:

```

  nellist-all2 A nellx nelly ⇒
    llist-all2 A (lbutlast (llist-of-nellist nellx)) (lbutlast (llist-of-nellist nelly))
proof (coinduction arbitrary: nellx nelly)
  case LNil
  then show ?case
  by (metis lbutlast.disc-iff(1) llist.disc(1) llist-of-nellist-inverse-a ltl-llist-of-nellist1
    nellist-all2-is-NNilD nellist-of-llist-a.disc(1))
  next
  case (LCons nell1 nell2)
  then show ?case
  proof –
    assume a0: nellist-all2 A nell1 nell2
    assume a1: ¬ lnull (lbutlast (llist-of-nellist nell1))
    assume a2: ¬ lnull (lbutlast (llist-of-nellist nell2))
    have 1: A (lhd (lbutlast (llist-of-nellist nell1))) (lhd (lbutlast (llist-of-nellist nell2)))
    using a0 a1 a2
    by (metis lbutlast.disc(1) llist.disc(1) llist-of-nellist-inverse-a ltl-llist-of-nellist1
      nellist-all2-nhdD nhd-nellist-of-llist-a)
    have 2: ((∃ nellx nelly.
      ltl (lbutlast (llist-of-nellist nell1)) = lbutlast (llist-of-nellist nellx) ∧
      ltl (lbutlast (llist-of-nellist nell2)) = lbutlast (llist-of-nellist nelly) ∧
      nellist-all2 A nellx nelly) ∨

```



```

      llist-all2 A (ltl (lbutlast (llist-of-nellist nell1))) (ltl (lbutlast (llist-of-nellist nell2))))
  by (metis a0 a1 lbutlast.ctr(1) lbutlast-ltl llist.discI(1) ltl-llist-of-nellist ltl-llist-of-nellist1
      nellist.rel-sel)
  show ?thesis using 1 2 by blast
qed
qed

```

```

lemma nellist-all2-nellist-of-llist-a [simp]:
  nellist-all2 A (nellist-of-llist-a b llx) (nellist-of-llist-a c lly)  $\longleftrightarrow$ 
  llist-all2 A llx lly  $\wedge$  (lfinite llx  $\longrightarrow$  A b c)
proof (cases lfinite llx)
  case True
  then show ?thesis
  using llist-all2-nellist-of-llistI
  by (auto simp add: llist-all2-lappendI nellist-all2-help-a, fastforce,
      metis lbutlast-lfinite lbutlast-snoc llist-all2-lfiniteD nellist-all2-nfinite1-nlastD
      nellist-of-llist-a-inverse nfinite-def nlast-nellist-of-llist-a-lfinite)
  next
  case False
  then show ?thesis
  by (metis lbutlast-snoc llist-all2-lappendI llist-all2-nellist-of-llistI nellist-all2-help-a
      nellist-of-llist-a-inverse)
qed

```

1.2.15 From a nonempty lazy list to a lazy list *llist-of-nellist*

```

lemma llist-of-nellist-nmap [simp]:
  llist-of-nellist (nmap f nell) = lmap f (llist-of-nellist nell)
by (simp add: lmap-llist-of-nellist)

```

```

lemma llist-of-nellist-nappend:
  llist-of-nellist (nappend nellx nelly) = lappend (llist-of-nellist nellx) (llist-of-nellist nelly)
by (transfer) auto

```

```

lemma llist-of-nellist-lappendn [simp]:
  llist-of-nellist (lappendn ll nell) = lappend ll (llist-of-nellist nell)
by transfer auto

```

```

lemma llist-of-nellist-nconcat [simp]:
  llist-of-nellist (nconcat nell) = lconcat ((lmap llist-of-nellist  $\circ$  llist-of-nellist) nell)
using nconcat-def3 by fastforce

```

```

lemma llist-of-nellist-nfilter [simp]:
assumes  $\exists x \in \text{nset } \text{nell}. P x$ 
shows llist-of-nellist (nfilter P nell) = lfilter P (llist-of-nellist nell)
using assms
by transfer auto

```

1.2.16 *ndropn*

lemma *ndropn-0* [*simp*, *code*, *nitpick-simp*]:

ndropn 0 nell = nell

using *zero-enat-def* **by** *transfer auto*

lemma *ndropn-NNil* [*simp*, *code*]:

ndropn n (NNil b) = (NNil b)

by *transfer auto*

lemma *ndropn-Suc-NCons* [*simp*, *code*]:

ndropn (Suc n) (NCons x nell) = ndropn n nell

proof (*cases nfinite nell*)

case *True*

then show *?thesis*

by *transfer*

(*auto simp add: min-def not-lnull-conv Suc-ile-eq llength-eq-infty-conv-lfinite the-enat-eSuc ,
metis Extended-Nat.eSuc-mono eSuc-enat iless-Suc-eq leD,
metis antisym eSuc-enat enat-the-enat ile-eSuc llength-eq-infty-conv-lfinite n-not-Suc-n
the-enat.simps*)

next

case *False*

then show *?thesis*

by *transfer*

(*simp,
metis co.enat.sel(2) eSuc-infinity infinity-ileE ldropn-Suc-LCons llength-eq-infty-conv-lfinite
min.cobounded1 min-def the-enat.simps*)

qed

lemma *ndropn-Suc* [*nitpick-simp*]:

ndropn (Suc n) nell = (case nell of NNil b \Rightarrow NNil b | NCons x nell' \Rightarrow ndropn n nell')

by(*cases nell*) *simp-all*

lemma *ltl-power-NNil-help*:

($(\lambda ll. \text{if } \exists b. ll = LCons b LNil \text{ then } ll \text{ else } ltl ll) \sim^n$) (*LCons b LNil*) = *LCons b LNil*

by (*induction n*) *simp-all*

lemma *ntl-power-NNil*:

(*ntl* \sim^n *n*) (*NNil b*) = (*NNil b*)

by *transfer (auto simp add: lnull-def ltl-power-NNil-help)*

lemma *ntl-power-NCons*:

*ntl ((ntl \sim^n *n*) (NCons x nell)) = (ntl \sim^n *n*) nell*

by (*induction n*) (*transfer, auto*)

lemma *ntl-power-Suc* [*simp*]:

(*ntl* \sim^n (*Suc n*)) *nell* = (*case nell of NNil b \Rightarrow NNil b | NCons x nell' \Rightarrow (ntl \sim^n *n*) nell'*)

by (*cases nell*)

(*simp-all add: ntl-power-NNil ntl-power-NCons*)

lemma *llist-of-nellist-ndropn* [*simp*]:

```

llist-of-nellist (ndropn n nell) =
  ldropn (the-enat (min (enat n) ((epred(llength((llist-of-nellist nell)))))))
    (llist-of-nellist nell)
by transfer auto

lemma ndropn-Suc-conv-ndropn:
  enat n < nlength nell  $\implies$  NCons (nnth nell n) (ndropn (Suc n) nell) = ndropn n nell
proof (induct n arbitrary: nell)
case 0
then show ?case
proof (cases nell)
case (NNil x1)
then show ?thesis using 0.prems by auto
next
case (NCons x nell1)
then show ?thesis by simp
qed
next
case (Suc n)
then show ?case
proof (cases nell)
case (NNil x1)
then show ?thesis using Suc.prems by auto
next
case (NCons x nell1)
then show ?thesis using Suc
by (metis One-nat-def add.commute add-left-mono gen-nlength-code(2) gen-nlength-def leD
  ndropn-Suc-NCons nlength-code nnth-Suc-NCons not-le-imp-less of-nat-Suc of-nat-eq-enat
  one-enat-def plus-1-eq-Suc)
qed
qed

lemma ndropn-nlength [simp]:
  nlength (ndropn n nell) = nlength nell - enat n
proof (induct n arbitrary: nell)
case 0
then show ?case by simp
next
case (Suc n)
then show ?case
proof (cases nell)
case (NNil x1)
then show ?thesis by simp
next
case (NCons x nell1)
then show ?thesis using Suc
by (metis eSuc-enat eSuc-minus-eSuc ndropn-Suc-NCons nlength-NCons)
qed
qed

```

```

lemma ndropn-nnth [simp]:
  nnth (ndropn n nell) m = nnth nell (n+m)
proof (induct n arbitrary: m nell)
  case 0
  then show ?case by simp
  next
  case (Suc n)
  then show ?case
  proof (cases nell)
  case (NNil x1)
  then show ?thesis by (simp add: nnth-NNil)
  next
  case (NCons x nell1)
  then show ?thesis by (simp add: Suc.hyps)
qed
qed

lemma ndropn-nnth-a:
assumes nlenght nell ≤ enat(n+m)
shows nnth (ndropn n nell) m = (nlast nell)
proof –
  have 1: nfinite nell
  using assms enat-ile nfinite-conv-nlenght-enat by auto
  have 2: nfinite nell ⇒ nlenght nell ≤ enat(n+m) ⇒ nnth (ndropn n nell) m = (nlast nell)
  proof (induct arbitrary: n m rule: nfinite-induct)
  case (NNil y)
  then show ?case by (simp add: nnth-NNil)
  next
  case (NCons x nell)
  then show ?case
  proof (cases n)
  case 0
  then show ?thesis
  by (metis NCons(2) NCons(3) Suc-ile-eq Suc-pred add-cancel-right-left enat-le-plus-same(1)
    gen-nlenght-def iless-Suc-eq leD le-add1 ndropn-0 nellist.sel(2) nlast0-nlast nlenght-NCons
    nlenght-code nnth-Suc-NCons not-le-imp-less order.not-eq-order-implies-strict)
  next
  case (Suc nat)
  then show ?thesis
  by (metis NCons(2) NCons(3) add-Suc eSuc-enat epred-eSuc epred-le-epredI ndropn-Suc-NCons
    nlast-NCons nlenght-NCons)
qed
qed
show ?thesis using 1 2 assms by auto
qed

lemma ndropn-ntl :
  ndropn n nell = (ntl ~ n) nell
proof (induction n arbitrary: nell)
case 0

```

```

then show ?case by simp
next
case (Suc n)
then show ?case
  proof (cases nell)
    case (NNil x1)
    then show ?thesis
      by (simp add: ntl-power-NNil)
  next
    case (NCons x nell)
    then show ?thesis
      by (metis Suc.IH funpow-Suc-right ndropn-Suc-NCons nellist.sel(5) o-apply)
  qed
qed

```

```

lemma ndropn-is-NNil:
  is-NNil nell  $\implies$  ndropn n nell = nell
proof (induct n arbitrary: nell)
  case 0
  then show ?case by auto
  next
    case (Suc n)
    then show ?case by (simp add: ndropn-Suc nellist.case-eq-if)
  qed

```

```

lemma is-NNil-ndropn:
  is-NNil(ndropn n nell)  $\longleftrightarrow$  nlength nell  $\leq$  (enat n)
proof (induct n arbitrary: nell)
  case 0
  then show ?case
    proof (cases nell)
      case (NNil x1)
      then show ?thesis by simp
    next
      case (NCons x nell)
      then show ?thesis using zero-enat-def by auto
    qed
  next
    case (Suc n)
    then show ?case
      proof (cases nell)
        case (NNil x1)
        then show ?thesis by simp
      next
        case (NCons x nell)
        then show ?thesis
          by (metis One-nat-def Suc.hyps add commute add-left-mono co.enat.sel(2) eSuc-enat epred-le-epredI
            gen-nlength-code(2) gen-nlength-def ndropn-Suc-NCons nlength-NCons nlength-code of-nat-Suc
            of-nat-eq-enat one-enat-def plus-1-eq-Suc)
      qed
    qed

```

qed

lemma *ndropn-eq-NNil*:

$ndropn\ n\ nell = (NNil\ b) \longleftrightarrow nnth\ nell\ (the-enat(nlength\ nell)) = b \wedge nlength\ nell \leq (enat\ n)$

proof –

have 1: $nfinite\ nell \implies ndropn\ n\ nell = NNil\ b \implies nnth\ nell\ (the-enat\ (nlength\ nell)) = b$

by (*metis add-cancel-left-right enat-le-plus-same(2) gen-nlength-def is-NNil-ndropn ndropn-NNil ndropn-nnth ndropn-nnth-a nfinite-nlength-enat nlast-NNil nlength-NNil nlength-code plus-enat-simps(1) the-enat.simps zero-enat-def*)

have 2: $nfinite\ nell \implies ndropn\ n\ nell = NNil\ b \implies nlength\ nell \leq enat\ n$

by (*metis is-NNil-ndropn nellist.disc(1)*)

have 3: $nfinite\ nell \implies nlength\ nell \leq enat\ n \implies b = nnth\ nell\ (the-enat\ (nlength\ nell)) \implies ndropn\ n\ nell = NNil\ (nnth\ nell\ (the-enat\ (nlength\ nell)))$

by (*metis add.right-neutral is-NNil-ndropn ndropn-nnth-a nellist.collapse(1) nfinite-NNil nlength-NNil nnth-nlast the-enat-0*)

have 4: $\neg nfinite\ nell \implies$

$ndropn\ n\ nell = (NNil\ b) \longleftrightarrow$

$nnth\ nell\ (the-enat(nlength\ nell)) = b \wedge nlength\ nell \leq (enat\ n)$

by (*metis enat-ile is-NNil-ndropn nellist.disc(1) nfinite-conv-nlength-enat*)

show *?thesis*

using 1 2 3 4 **by** *fastforce*

qed

lemma *ntl-ndropn*:

$ntl(ndropn\ n\ nell) = ndropn\ n\ (ntl\ nell)$

by (*simp add: funpow-swap1 ndropn-ntl*)

lemma *nfinite-ndropn-a*:

assumes *nfinite nell*

shows *nfinite(ndropn n nell)*

using *assms*

proof (*induct n arbitrary: nell*)

case 0

then show *?case by auto*

next

case (*Suc n*)

then show *?case by (simp add: ndropn-ntl)*

qed

lemma *nfinite-ndropn-b*:

assumes *nfinite(ndropn n nell)*

shows *nfinite nell*

using *assms*

proof (*induct ys \equiv ndropn n nell arbitrary: n nell rule: nfinite-induct*)

case (*NNil y*)

then show *?case by (metis enat-ile ndropn-eq-NNil nfinite-conv-nlength-enat)*

next

case (*NCons x nell*)

then show *?case by (metis nellist.sel(5) nfinite-ntl ntl-ndropn)*

qed

lemma *nfinite-ndropn[simp]*:
 $nfinite(ndropn\ n\ nell) = nfinite\ nell$
using *nfinite-ndropn-a nfinite-ndropn-b* **by** *blast*

lemma *ndropn-ndropn*:
 $ndropn\ m\ (ndropn\ n\ nell) = ndropn\ (n+m)\ nell$
proof (*induct n arbitrary: nell*)
case 0
then show ?*case* **by** *simp*
next
case (*Suc n*)
then show ?*case*
by (*metis add-Suc ndropn-NNil ndropn-Suc nellist.case-eq-if*)
qed

lemma *ndropn-nlast*:
 $nfinite\ nell \implies ndropn\ (the-enat(nlength\ nell))\ nell = (NNil\ (nlast\ nell))$
by (*metis add.left-neutral enat.simps(3) enat-the-enat ndropn-eq-NNil ndropn-nnth ndropn-nnth-a nfinite-conv-nlength-enat order-refl*)

lemma *ndropn-nfirst*:
 $nfirst\ (ndropn\ n\ nell) = (nnth\ nell\ n)$
by *transfer*
(*metis lhd-ldropn llist-of-nellist-ndropn lnull-ldropn not-le-imp-less not-lnull-conv-llist-of-nellist*)

lemma *ndropn-all*:
 $nlength\ nell \leq enat\ n \implies ndropn\ n\ nell = (NNil\ (nlast\ nell))$
by (*metis enat-ile ndropn-eq-NNil ndropn-nlast nlength-eq-enat-nfiniteD*)

lemma *ndropn-nappend1*:
 $nfinite\ nellx \implies n \leq nlength\ nelly \implies$
 $ndropn\ (Suc(the-enat\ (nlength\ nellx) + n))\ (nappend\ nellx\ nelly) = ndropn\ n\ nelly$
proof (*induct arbitrary: nelly n rule: nfinite-induct*)
case (*NNil y*)
then show ?*case* **by** *simp*
next
case (*NCons x nell*)
then show ?*case*
by (*metis ab-semigroup-add-class.add-ac(1) eSuc-enat nappend-NCons ndropn-Suc-NCons nfinite-nlength-enat nlength-NCons plus-1-eq-Suc the-enat.simps*)
qed

lemma *ndropn-nappend2*:
 $enat\ n \leq (nlength\ nellx) \implies ndropn\ n\ (nappend\ nellx\ nelly) = nappend\ (ndropn\ n\ nellx)\ nelly$
proof (*induct n arbitrary: nellx nelly*)
case 0
then show ?*case* **by** *simp*
next

```

case (Suc n)
then show ?case
  proof (cases nellx)
    case (NNil x1)
    then show ?thesis
    using Suc.premis enat-0-iff(1) by auto
  next
    case (NCons x21 x22)
    then show ?thesis
by (metis Suc.hyps Suc.premis eSuc-enat eSuc-ile-mono nappend-NCons ndropn-Suc-NCons nlength-NCons)
qed
qed

```

```

lemma ndropn-nappend3:
  nlength nellx < enat n  $\implies$ 
  ndropn n (nappend nellx nelly) = ndropn (n - (the-enat (eSuc(nlength nellx)))) nelly
proof (induct n arbitrary: nellx nelly)
case 0
then show ?case using zero-enat-def by auto
next
case (Suc n)
then show ?case
  proof (cases nellx)
    case (NNil x1)
    then show ?thesis
    by (metis add-diff-cancel-left' nappend-NNil ndropn-Suc-NCons nlength-NNil one-eSuc
      one-enat-def plus-1-eq-Suc the-enat.simps)
  next
    case (NCons x21 x22)
    then show ?thesis
    by (metis Extended-Nat.eSuc-mono Suc.hyps Suc.premis add-diff-cancel-left' diff-Suc-eq-diff-pred
      eSuc-enat enat-ord-code(4) nappend-NCons ndropn-Suc-NCons nlength-NCons
      order-less-imp-not-less plus-1-eq-Suc the-enat-eSuc)
  qed
qed

```

```

lemma nset-ndropn:
  nset (ndropn n nell)  $\subseteq$  nset nell
by transfer (simp add: lset-ldropn-subset)

```

```

lemma ndropn-nmap:
  ndropn n (nmap f nell) = nmap f (ndropn n nell)
by transfer
  (auto,
    metis eSuc-epred enat-the-enat iless-Suc-eq infinity-ileE leD llength-eq-0 min.cobounded1
    min.cobounded2)

```

```

lemma nappend-ntaken-ndropn:
  assumes (Suc k)  $\leq$  nlength nell

```


shows $nappend (ntaken\ k\ nell) (ndropn\ (Suc\ k)\ nell) = nell$
using *assms*
by *transfer (simp add: min-absorb1)*

lemma *nfirst-eq-nnth-zero*:
 $nfirst\ nell = nnth\ nell\ 0$
by *(metis ndropn-0 ndropn-nfirst)*

1.2.17 *nzip*

lemma *nzip-nhd*:
 $\neg is-NNil\ nellx \wedge \neg is-NNil\ nelly \implies nhd\ (nzip\ nellx\ nelly) = (nhd\ nellx, nhd\ nelly)$
by *transfer*
 $(auto\ simp\ add: lzip-eq-LCons-conv,$
 $metis\ lzip-eq-LNil-conv)$

lemma *nzip-ntl*:
 $\neg is-NNil\ nellx \wedge \neg is-NNil\ nelly \implies ntl(nzip\ nellx\ nelly) = nzip\ (ntl\ nellx)\ (ntl\ nelly)$
by *transfer*
 $(auto,$
 $metis\ lhd-LCons-ltl\ ltl-lzip\ ltl-simps(2)\ lzip-eq-LNil-conv,$
 $metis\ (full-types)\ lhd-LCons-ltl\ lnull-def,$
 $metis\ lhd-LCons-ltl\ llist.collapse(1))$

lemma *nzip-simps* [*simp, code, nitpick-simp*]:
 $nzip\ (NNil\ b)\ nelly = (NNil\ (b, (nnth\ nelly\ 0)))$
 $nzip\ nellx\ (NNil\ b) = (NNil\ ((nnth\ nellx\ 0), b))$
 $nzip\ (NCons\ x\ nellx)\ (NCons\ y\ nelly) = NCons\ (x, y)\ (nzip\ nellx\ nelly)$
apply *transfer*
apply *(auto simp add: not-lnull-conv)*
apply *(metis lnth-0 min-enat-simps(3) the-enat-0 zero-enat-def)*
apply *transfer*
apply *(auto simp add: not-lnull-conv)*
apply *(metis lnth-0 min-enat-simps(3) the-enat-0 zero-enat-def)*
apply *transfer*
by *(auto simp add: not-lnull-conv)*

lemma *is-NNil-nzip* [*simp*]:
 $is-NNil\ (nzip\ nellx\ nelly) \longleftrightarrow (is-NNil\ nellx) \vee (is-NNil\ nelly)$
by *transfer*
 $(auto\ simp\ add: lzip-eq-LCons-conv\ not-lnull-conv,$
 $metis\ lzip-eq-LNil-conv)$

lemma *nzip-eq-NNil-conv*:
 $nzip\ nellx\ nelly = (NNil\ (x, y)) \longleftrightarrow$
 $((is-NNil\ nellx) \vee (is-NNil\ nelly)) \wedge (nnth\ nellx\ 0) = x \wedge (nnth\ nelly\ 0) = y$
by *auto*
 $(metis\ is-NNil-nzip\ nellist.disc(1),$
 $metis\ Pair-inject\ is-NNil-nzip\ nellist.collapse(1)\ nellist.disc(1)\ nlast-NNil\ nzip-simps(1))$

nzip-simps(2),
metis Pair-inject is-NNil-def is-NNil-nzip nellist.inject(1) *nzip-simps*(1) *nzip-simps*(2),
metis nellist.collapse(1) *nnth-NNil nzip-simps*(1),
metis nellist.collapse(1) *nnth-NNil nzip-simps*(2))

lemma *nzip-eq-NCons-conv*:

nzip nellx nelly = (*NCons z zs*) \longleftrightarrow
 $(\exists x \text{ nellx}' y \text{ nelly}'. \text{nellx} = (\text{NCons } x \text{ nellx}') \wedge \text{nelly} = (\text{NCons } y \text{ nelly}') \wedge$
 $z = (x, y) \wedge zs = (\text{nzip nellx}' \text{nelly}'))$

by (*cases nellx nelly rule: nellist.exhaust[case-product nellist.exhaust]*)
auto

lemma *nzip-nappend*:

nlength nellx = *nlength nellu*
 $\implies \text{nzip } (\text{nappend } \text{nellx } \text{nelly}) (\text{nappend } \text{nellu } \text{nellv}) =$
 $\text{nappend } (\text{nzip } \text{nellx } \text{nellu}) (\text{nzip } \text{nelly } \text{nellv})$

by *transfer*
(simp,
meson co.enat.expand llength-eq-0 lzip-lappend)

lemma *nlength-nzip [simp]*:

nlength (nzip nellx nelly) = (*min (nlength nellx) (nlength nelly)*)

by *transfer simp*

lemma *ntake-nzip*:

ntake n (nzip nellx nelly) = *nzip (ntake n nellx) (ntake n nelly)*

by *transfer*
(simp add: ltake-lzip)

lemma *ntaken-nzip*:

ntaken n (nzip nellx nelly) = *nzip (ntaken n nellx) (ntaken n nelly)*

by *transfer*
(simp add: enat-0-iff(1) ltake-lzip)

lemma *ndropn-nzip [simp]*:

$n \leq \text{nlength } \text{nellx} \wedge n \leq \text{nlength } \text{nelly} \implies$
 $\text{ndropn } n (\text{nzip } \text{nellx } \text{nelly}) = \text{nzip } (\text{ndropn } n \text{ nellx}) (\text{ndropn } n \text{ nelly})$

by *transfer*
(auto simp add: min-def,
metis co.enat.exhaust-sel iless-Suc-eq leD llength-eq-0,
metis co.enat.exhaust-sel iless-Suc-eq leD llength-eq-0)

lemma *nzip-niterates*:

nzip (niterates f x) (niterates g y) = *niterates ($\lambda(x,y). (f x, g y)$) (x, y)*

by *transfer*
(simp add: lzip-iterates)

lemma *nnth-nzip*:

assumes $n \leq \text{nlength } \text{nellx}$
 $n \leq \text{nlength } \text{nelly}$

shows $nnth\ (nzip\ nellx\ nelly)\ n = (nnth\ nellx\ n,\ nnth\ nelly\ n)$
using *assms*
by *transfer*
 $(auto\ simp\ add:\ min\text{-}def,\$
 $metis\ co.enat.exhaust\text{-}sel\ iless\text{-}Suc\text{-}eq\ llength\text{-}eq\ 0\ lnth\text{-}lzip)$

lemma *nset-nzip*:

$nset\ (nzip\ nellx\ nelly) =$
 $\{ (nnth\ nellx\ n,\ nnth\ nelly\ n) \mid n.\ n \leq \min\ (nlength\ nellx)\ (nlength\ nelly) \}$

by *transfer*
 $(auto\ simp\ add:\ in\text{-}lset\text{-}conv\text{-}lnth,\$
 $metis\ Pair\text{-}inject\ co.enat.exhaust\text{-}sel\ iless\text{-}Suc\text{-}eq\ llength\text{-}eq\ 0\ lnth\text{-}lzip\ min.orderE$
 $the\text{-}enat.simps,$
 $metis\ eSuc\text{-}epred\ iless\text{-}Suc\text{-}eq\ llength\text{-}eq\ 0\ lnth\text{-}lzip)$

lemma *nset-nzipD1*:

$(x,\ y) \in nset\ (nzip\ nellx\ nelly) \implies x \in nset\ nellx$
by *transfer*
 $(meson\ lset\text{-}lzipD1)$

lemma *nset-nzipD2*:

$(x,\ y) \in nset\ (nzip\ nellx\ nelly) \implies y \in nset\ nelly$
by *transfer*
 $(meson\ lset\text{-}lzipD2)$

lemma *nset-nzip-same* [*simp*]:

$nset\ (nzip\ nellx\ nellx) = (\lambda\ x.\ (x,\ x))\ `nset\ nellx$
by *transfer*
 $simp$

lemma *nfinite-nzip* [*simp*]:

$nfinite\ (nzip\ nellx\ nelly) \longleftrightarrow nfinite\ nellx \vee nfinite\ nelly$
by *transfer*
 $simp$

lemma *nzip-eq-nappend-conv*:

assumes *eq*: $nzip\ nellx\ nelly = nappend\ nellu\ nellv$
shows $\exists\ nellx'\ nellx'' nelly'\ nelly''.$
 $nellx = nappend\ nellx'\ nellx'' \wedge nelly = nappend\ nelly'\ nelly'' \wedge$
 $nlenght\ nellx' = nlenght\ nelly' \wedge$
 $nellu = nzip\ nellx'\ nelly' \wedge nellv = nzip\ nellx'' nelly''$

using *assms*
apply *transfer*
using *lzip-eq-lappend-conv*
by $(auto\ simp\ add:\ lzip\text{-}eq\text{-}lappend\text{-}conv)$
 $fastforce$

lemma *nzip-nmap* [*simp*]:

$nzip\ (nmap\ f\ nellx)\ (nmap\ g\ nelly) = nmap\ (\lambda(x,\ y).\ (f\ x,\ g\ y))\ (nzip\ nellx\ nelly)$
by *transfer auto*

lemma *nzip-nmap1*:

$nzip (nmap f nellx) nelly = nmap (\lambda(x, y). (f x, y)) (nzip nellx nelly)$

by *transfer*

(*simp add: lzip-lmap1*)

lemma *nzip-nmap2*:

$nzip nellx (nmap f nelly) = nmap (\lambda(x, y). (x, f y)) (nzip nellx nelly)$

by *transfer*

(*simp add: lzip-lmap2*)

lemma *nmap-fst-nzip-conv-ntake*:

$nmap fst (nzip nellx nelly) = ntake (min (nlength nellx) (nlength nelly)) nellx$

by *transfer*

(*auto*,

metis co.enat.exhaust-sel llength-eq-0 lmap-fst-lzip-conv-ltake min-eSuc-eSuc)

lemma *nmap-snd-nzip-conv-ntake*:

$nmap snd (nzip nellx nelly) = ntake (min (nlength nellx) (nlength nelly)) nelly$

by *transfer*

(*auto*,

metis co.enat.exhaust-sel llength-eq-0 lmap-snd-lzip-conv-ltake min-eSuc-eSuc)

lemma *nzip-conv-nzip-ntake-min-nlength*:

$nzip nellx nelly =$

$nzip (ntake (min (nlength nellx) (nlength nelly)) nellx)$

$(ntake (min (nlength nellx) (nlength nelly)) nelly)$

by *transfer*

(*auto*,

metis co.enat.exhaust-sel epred-min i0-lb llength-lzip ltake-all ltake-lzip order-refl)

lemma *nellist-all2-conv-nzip*:

$nellist-all2 P nellx nelly \longleftrightarrow$

$nlength nellx = nlength nelly \wedge (\forall (x, y) \in nset(nzip nellx nelly). P x y)$

using *nset-nzip[of nellx nelly]*

by (*auto simp add: nellist-all2-conv-all-nnth*)

blast

lemma *nellist-all2-all-nnthI*:

assumes $nlength nellx = nlength nelly$

$\bigwedge n. enat n \leq nlength nellx \implies P (nnth nellx n) (nnth nelly n)$

shows $nellist-all2 P nellx nelly$

using *assms* **by** (*simp add: nellist-all2-conv-all-nnth*)

lemma *nellist-all2-nsetD1*:

assumes $nellist-all2 P nellx nelly$

$x \in nset nellx$

shows $\exists y \in nset nelly. P x y$

using *assms*

by (*metis in-nset-conv-nnth nellist-all2-conv-all-nnth*)

```

lemma nellist-all2-nsetD2:
assumes nellist-all2 P nellx nelly
           $y \in \text{nset } nelly$ 
shows  $\exists x \in \text{nset } nellx. P x y$ 
using assms
by (metis in-nset-conv-nnth nellist-all2-conv-all-nnth)

lemma nellist-all2-nzipI:
assumes nellist-all2 P nellx nelly
          nellist-all2 P' nellx' nelly'
shows nellist-all2 (rel-prod P P') (nzip nellx nellx') (nzip nelly nelly')
using assms
proof (coinduction arbitrary: nellx nellx' nelly nelly')
case (ilist-all2 nx nx' ny ny')
then show ?case
proof –
  have 1: is-NNil (nzip nx nx') = is-NNil (nzip ny ny')
    by (metis ilist-all2(1) ilist-all2(2) is-NNil-nzip nellist-all2-is-NNilD)
  have 2: (is-NNil (nzip nx nx')  $\implies$ 
            is-NNil (nzip ny ny')  $\implies$ 
            rel-prod P P' (nlast (nzip nx nx')) (nlast (nzip ny ny')))
    by (metis (no-types, lifting) enat-min-eq-0-iff ilist-all2(1) ilist-all2(2)
            is-NNil-nzip min-def-raw nellist.sel(1) nellist-all2-conv-all-nnth
            nzip-eq-NNil-conv order-refl rel-prod-inject zero-enat-def)
  have 3: ( $\neg$  is-NNil (nzip nx nx')  $\implies$ 
             $\neg$  is-NNil (nzip ny ny')  $\implies$ 
            (( $\exists$  nellx nellx' nelly nelly'.
              ntl (nzip nx nx') = nzip nellx nellx'  $\wedge$ 
              ntl (nzip ny ny') = nzip nelly nelly'  $\wedge$ 
              nellist-all2 P nellx nelly  $\wedge$  nellist-all2 P' nellx' nelly')  $\vee$ 
              nellist-all2 (rel-prod P P') (ntl (nzip nx nx')) (ntl (nzip ny ny')))))
    by (auto simp add: nzip-eq-NCons-conv dest: nellist-all2-nhdD intro: nellist-all2-ntlI)
    (meson ilist-all2(1) ilist-all2(2) nellist-all2-ntlI nzip-ntl)
  have 4:  $\neg$  is-NNil (nzip nx nx')  $\longrightarrow$ 
             $\neg$  is-NNil (nzip ny ny')  $\longrightarrow$ 
            rel-prod P P' (nhd (nzip nx nx')) (nhd (nzip ny ny'))
    by (simp add: ilist-all2(1) ilist-all2(2) nellist-all2-nhdD nzip-nhd)
  have 5: ( $\neg$  is-NNil (nzip nx nx')  $\longrightarrow$ 
             $\neg$  is-NNil (nzip ny ny')  $\longrightarrow$ 
            rel-prod P P' (nhd (nzip nx nx')) (nhd (nzip ny ny'))  $\wedge$ 
            (( $\exists$  nellx nellx' nelly nelly'.
              ntl (nzip nx nx') = nzip nellx nellx'  $\wedge$ 
              ntl (nzip ny ny') = nzip nelly nelly'  $\wedge$ 
              nellist-all2 P nellx nelly  $\wedge$  nellist-all2 P' nellx' nelly')  $\vee$ 
              nellist-all2 (rel-prod P P') (ntl (nzip nx nx')) (ntl (nzip ny ny')))))
    using 3 4 by blast
show ?thesis
    using 1 2 3 4 by presburger
qed

```

qed

lemma *ndistinct-nzipI1*:

$ndistinct\ nellx \implies ndistinct\ (nzip\ nellx\ nelly)$

by *transfer*

(*simp add: ldistinct-lzipI1*)

lemma *ndistinct-nzipI2*:

$ndistinct\ nelly \implies ndistinct\ (nzip\ nellx\ nelly)$

by *transfer*

(*simp add: ldistinct-lzipI2*)

1.2.18 *niterates*

lemma *niterates-not-is-NNil* [*nitpick-simp*, *simp*]:

$\neg is-NNil\ (niterates\ f\ x)$

by *transfer*

(*metis lfinite-LConsI lfinite-code(1) lfinite-iterates*)

lemma *nhd-niterates* [*code*, *simp*, *nitpick-simp*]:

$nhd(niterates\ f\ x) = x$

by *transfer*

(*metis lfinite-LConsI lfinite-LNil lfinite-iterates lhd-iterates*)

lemma *ntl-niterates* [*code*, *simp*, *nitpick-simp*]:

$ntl(niterates\ f\ x) = niterates\ f\ (f\ x)$

by *transfer*

(*simp*,
metis lfinite-LConsI lfinite-LNil lfinite-iterates)

lemma *nfinite-niterates* [*iff*]:

$\neg nfinite\ (niterates\ f\ x)$

by *transfer simp*

lemma *niterates-nmap*:

$niterates\ f\ x = NCons\ x\ (nmap\ f\ (niterates\ f\ x))$

by *transfer*

(*meson iterates.disc-iff iterates-lmap*)

lemma [*simp*]:

fixes $f :: 'a \Rightarrow 'a$

shows *is-NNil-funpow-nmap*: $is-NNil\ ((nmap\ f\ \frown n)\ nellx) \longleftrightarrow is-NNil\ nellx$

and *nhd-funpow-nmap*: $\neg is-NNil\ nellx \implies nhd\ ((nmap\ f\ \frown n)\ nellx) = (f\ \frown n)\ (nhd\ nellx)$

and *ntl-funpow-nmap*: $\neg is-NNil\ nellx \implies ntl\ ((nmap\ f\ \frown n)\ nellx) = (nmap\ f\ \frown n)\ (ntl\ nellx)$

by (*induct n*) *simp-all*

lemma *niterates-equality*:

assumes $h: \bigwedge x. h\ x = NCons\ x\ (nmap\ f\ (h\ x))$

shows $h = niterates\ f$

proof –

```

{ fix x
  have  $\neg is\_NNil (h\ x) \ nhd\ (h\ x) = x \ ntl\ (h\ x) = nmap\ f\ (h\ x)$ 
  by (subst h, simp)+ }
note [simp] = this
{ fix x
  define n :: nat where n = 0
  have (nmap f  $\sim$  n) (h x) = (nmap f  $\sim$  n) (niterates f x)
  proof (coinduction arbitrary: n)
  case (Eq-nellist nn)
  then show ?case
  proof -
  have 1:  $is\_NNil ((nmap\ f\ \sim\ nn)\ (h\ x)) = is\_NNil ((nmap\ f\ \sim\ nn)\ (niterates\ f\ x))$ 
  by auto
  have 2:  $(is\_NNil ((nmap\ f\ \sim\ nn)\ (h\ x)) \longrightarrow$ 
     $is\_NNil ((nmap\ f\ \sim\ nn)\ (niterates\ f\ x)) \longrightarrow$ 
     $nlast\ ((nmap\ f\ \sim\ nn)\ (h\ x)) = nlast\ ((nmap\ f\ \sim\ nn)\ (niterates\ f\ x))$ 
  by simp
  have 3:  $(\neg is\_NNil ((nmap\ f\ \sim\ nn)\ (h\ x)) \longrightarrow$ 
     $\neg is\_NNil ((nmap\ f\ \sim\ nn)\ (niterates\ f\ x)) \longrightarrow$ 
     $nhd\ ((nmap\ f\ \sim\ nn)\ (h\ x)) = nhd\ ((nmap\ f\ \sim\ nn)\ (niterates\ f\ x))$ 
  by simp
  have 4:  $(\neg is\_NNil ((nmap\ f\ \sim\ nn)\ (h\ x)) \longrightarrow$ 
     $\neg is\_NNil ((nmap\ f\ \sim\ nn)\ (niterates\ f\ x)) \longrightarrow$ 
     $(\ ntl\ ((nmap\ f\ \sim\ nn)\ (h\ x)) = (nmap\ f\ \sim\ (Suc\ nn))\ (h\ x) \wedge$ 
     $ntl\ ((nmap\ f\ \sim\ nn)\ (niterates\ f\ x)) = (nmap\ f\ \sim\ (Suc\ nn))\ (niterates\ f\ x))$ 
  by (metis  $\langle \bigwedge x. \neg is\_NNil\ (h\ x) \rangle \langle \bigwedge x. \ ntl\ (h\ x) = nmap\ f\ (h\ x) \rangle$  funpow-simps-right(2)
    nellist.sel(5) niterates-nmap niterates-not-is-NNil ntl-funpow-nmap o-apply)
  show ?thesis using 1 2 3 4 by blast
  qed
  qed
}
thus ?thesis by auto
qed

```

lemma nlength-niterates [simp]:

$nlength\ (niterates\ f\ x) = \infty$

by transfer auto

lemma ndropn-niterates:

$ndropn\ n\ (niterates\ f\ x) = niterates\ f\ ((f\ \sim\ n)\ x)$

by transfer

(simp add: ldrown-iterates)

lemma nnth-niterates [simp]:

$nnth\ (niterates\ f\ x)\ n = (f\ \sim\ n)\ x$

by transfer auto

lemma nset-niterates:

$nset\ (niterates\ f\ x) = \{ (f\ \sim\ n)\ x \mid n. \ True \}$

by transfer

(metis lset-iterates)

```
lemma nnth-niterates-Suc:
  nnth (niterates Suc 0) i = i
proof (induct i)
case 0
then show ?case
by force
next
case (Suc i)
then show ?case by simp
qed
```

1.2.19 Filtering non-empty lazy lists *nfilter*

```
lemma nfilter-NNil [simp]:
shows nfilter P (NNil b) = NNil b
by transfer auto
```

```
lemma nfilter-True [simp]:
shows nfilter ( $\lambda x. \text{True}$ ) nell = nell
by transfer auto
```

```
lemma nfilter-False-finite:
assumes nfinite nell
shows nfilter ( $\lambda x. \text{False}$ ) nell = nell
using assms by transfer auto
```

```
lemma nfilter-NCons [simp]:
assumes ( $\exists x \in \text{nset nell}. P x$ )
shows nfilter P (NCons x nell) = (if P x then NCons x (nfilter P nell) else nfilter P nell)
using assms by transfer auto
```

```
lemma nfilter-NCons-a [simp]:
assumes  $\neg(\exists x \in \text{nset nell}. P x)$ 
      P x
shows nfilter P (NCons x nell) = (NNil x)
using assms by transfer auto
```

```
lemma nfilter-expand:
assumes  $\exists x \in \text{nset nell}. P x$ 
shows nfilter P nell =
  (if is-NNil nell then nell
   else
    (if ( $\exists x \in \text{nset}(\text{ntl nell}). P x$ ) then
      (if P (nhd nell) then (NCons (nhd nell) (nfilter P (ntl nell)))
       else (nfilter P (ntl nell) ) )
      else (NNil (nhd nell) ) )
   )
using assms by (cases nell) auto
```


lemma *nset-nfilter*:

assumes $\exists x \in \text{nset nell}. P x$

shows $\text{nset} (\text{nfilter } P \text{ nell}) = \text{nset nell} \cap \{xa. P xa\}$

using *assms*

by *transfer auto*

lemma *exist-conj*:

assumes $\exists x \in \text{nset} (\text{nfilter } Q \text{ nell}). P x$

$\exists x \in \text{nset nell}. Q x$

shows $\exists x \in \text{nset nell}. P x \wedge Q x$

using *assms*

by *transfer (auto split: if-split-asm)*

lemma *nfilter-nfilter [simp]*:

assumes $\exists x \in \text{nset} (\text{nfilter } Q \text{ nell}). P x$

$\exists x \in \text{nset nell}. Q x$

shows $\text{nfilter } P (\text{nfilter } Q \text{ nell}) = \text{nfilter } (\lambda x. P x \wedge Q x) \text{ nell}$

using *assms*

proof (*transfer fixing: P Q*)

fix *xsa* :: 'a llist

assume $\neg \text{lnull } xsa \wedge xsa = xsa$

assume *a1*: $\text{Bex} (\text{lset} (\text{if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa})) P$

assume $\text{Bex} (\text{lset } xsa) Q$

then have $\neg \text{lnull} (\text{lfilter } Q \text{ xsa})$

by *simp*

then have $\text{lfilter } P (\text{if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa}) = \text{lfilter } P (\text{lfilter } Q \text{ xsa}) \wedge$
 $\neg \text{lnull} (\text{lfilter } P (\text{lfilter } Q \text{ xsa})) \wedge$
 $\neg \text{lnull} (\text{if lnull (lfilter } P (\text{if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa}))$
 $\text{then if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa}$
 $\text{else lfilter } P (\text{if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa}))$

using *a1* **by** (*auto split: if-split-asm*)

then show $\neg \text{lnull} (\text{if lnull (lfilter } P (\text{if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa}))$
 $\text{then if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa}$
 $\text{else lfilter } P (\text{if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa})) \wedge$
 $(\text{if lnull (lfilter } P (\text{if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa}))$
 $\text{then if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa}$
 $\text{else lfilter } P (\text{if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa})) =$
 $(\text{if lnull (lfilter } (\lambda a. P a \wedge Q a) \text{ xsa) then xsa else lfilter } (\lambda a. P a \wedge Q a) \text{ xsa})$

using *lfilter-lfilter* **by** *auto*

qed

lemma *length-nfilter-le [simp]*:

$\text{nlength} (\text{nfilter } P \text{ nell}) \leq \text{nlength nell}$

by *transfer (simp add: epred-le-epredI llength-lfilter-ile)*

lemma *nfilter-nnth*:

assumes $(\exists x \in \text{nset nell}. P x)$

$i \leq \text{nlength} (\text{nfilter } P \text{ nell})$

shows $(\exists k \leq \text{nlength nell}. \text{nnth}(\text{nfilter } P \text{ nell}) i = \text{nnth nell } k)$

using *assms nset-nfilter[of nell P]*

using *in-nset-conv-nnth*
by (*metis Int-iff*)

lemma *nfilter-nappend1*:
assumes $\forall x \in \text{nset } nell. \neg P x$
 $\text{nfinite } nell$
 $\exists y \in \text{nset } nell1. P y$
shows $\text{nfilter } P (\text{nappend } nell \text{ } nell1) = \text{nfilter } P \text{ } nell1$
using *assms* **by** *transfer auto*

lemma *nfilter-nappend2*:
assumes $\forall x \in \text{nset } nell. \neg P x$
 $\exists y \in \text{nset } nell1. P y$
shows $\text{nfilter } P (\text{nappend } nell1 \text{ } nell) = \text{nfilter } P \text{ } nell1$
using *assms*
by *transfer*
(*auto split: if-split-asm,*
meson in-lset-lappend-iff,
metis lappend-LNil2 lappend-inf lfilter-empty-conv lfilter-lappend-lfinite)

lemma *nfilter-nappend [simp]*:
assumes $(\exists x \in \text{nset } (\text{nappend } nell \text{ } nell1). P x)$
 $(\exists x \in \text{nset } nell. P x)$
 $(\exists x \in \text{nset } nell1. P x)$
shows $\text{nfilter } P (\text{nappend } nell \text{ } nell1) =$
 $(\text{if } \text{nfinite } nell \text{ then } \text{nappend } (\text{nfilter } P \text{ } nell) \text{ } (\text{nfilter } P \text{ } nell1)$
 $\text{else } (\text{nfilter } P \text{ } nell))$
proof (*cases nfinite nell*)
case *True*
then show *?thesis* **using** *assms* **by** *transfer auto*
next
case *False*
then show *?thesis* **using** *assms* **by** *transfer (auto simp add: lappend-inf)*
qed

lemma *nfilter-nmap*:
shows $\text{nfilter } P (\text{nmap } f \text{ } nell) = \text{nmap } f (\text{nfilter } (P \circ f) \text{ } nell)$
by *transfer (auto simp add: lfilter-lmap)*

lemma *nlenght-nfilter-nmap[simp]*:
shows $\text{nlenght } (\text{nfilter } P (\text{nmap } f \text{ } nell)) = \text{nlenght}(\text{nfilter } (P \circ f) \text{ } nell)$
by (*simp add: nfilter-nmap*)

lemma *nfilter-is-subset [simp]*:
assumes $\exists x \in \text{nset } nell. P x$
shows $\text{nset } (\text{nfilter } P \text{ } nell) \leq \text{nset } nell$
using *assms* **by** (*simp add: nset-nfilter*)

lemma *nfilter-cong[fundef-cong]*:
assumes $nell = nell1$

$(\bigwedge x. x \in \text{nset } \text{nell1} \implies P x = Q x)$
shows $\text{nfilter } P \text{ nell} = \text{nfilter } Q \text{ nell1}$
using *assms by transfer auto*

lemma *nset-nappend:*

$\text{nset } (\text{nappend } \text{nell } \text{nell1}) = (\text{if } \text{nfinite } \text{nell} \text{ then } \text{nset } \text{nell} \cup \text{nset } \text{nell1} \text{ else } \text{nset } \text{nell})$
by transfer (*simp add: lappend-inf*)

lemma *split-nellist-nappend:*

assumes $\exists i \leq \text{nlength } \text{nell}. \text{nnth } \text{nell } i = x \wedge P (\text{nnth } \text{nell } i) \wedge$
 $(\forall j. j \neq i \wedge j \leq \text{nlength } \text{nell} \longrightarrow \neg P(\text{nnth } \text{nell } j))$

shows $P x \wedge$

$(\text{nell} = (\text{NNil } x) \vee$
 $(\exists \text{vs}. \text{nell} = (\text{NCons } x \text{ vs}) \wedge (\forall v \in \text{nset } \text{vs}. \neg P v)) \vee$
 $(\exists \text{us}. \text{nell} = \text{nappend } \text{us } (\text{NNil } x) \wedge \text{nfinite } \text{us} \wedge (\forall u \in \text{nset } \text{us}. \neg P u)) \vee$
 $(\exists \text{us vs}. \text{nell} = \text{nappend } \text{us } (\text{NCons } x \text{ vs}) \wedge \text{nfinite } \text{us} \wedge$
 $(\forall u \in \text{nset } \text{us}. \neg P u) \wedge (\forall v \in \text{nset } \text{vs}. \neg P v))$
 $)$

proof –

obtain i **where** $1: i \leq \text{nlength } \text{nell} \wedge \text{nnth } \text{nell } i = x \wedge P (\text{nnth } \text{nell } i) \wedge$
 $(\forall j. j \neq i \wedge j \leq \text{nlength } \text{nell} \longrightarrow \neg P(\text{nnth } \text{nell } j))$

using *assms by auto*

have $01: (\forall j. (j < i \vee i < j) \wedge j \leq \text{nlength } \text{nell} \longrightarrow \neg P(\text{nnth } \text{nell } j))$

using 1

by *auto*

have $2: i=0 \wedge \text{nlength } \text{nell} = 0 \implies P x \wedge \text{nell} = (\text{NNil } x)$

by (*metis 1 ndropn-0 ndropn-eq-NNil the-enat-0 zero-enat-def*)

have $3: i=0 \wedge \text{nlength } \text{nell} > 0 \implies P x \wedge \text{nell} = (\text{NCons } x (\text{ntl } \text{nell})) \wedge (\forall v \in \text{nset } (\text{ntl } \text{nell}). \neg P v)$

using 1 *in-nset-conv-nnth[of - ntl nell]*

by (*metis Suc-ile-eq iless-Suc-eq nat.simps(3) nellist.disc(2) nellist.exhaust-sel nlength-NCons nlength-NNil nnth-0-conv-nhd nnth-ntl not-less-iff-gr-or-eq*)

have $4: i=0 \wedge \text{nlength } \text{nell} > 0 \implies P x \wedge (\exists \text{vs}. \text{nell} = (\text{NCons } x \text{ vs}) \wedge (\forall v \in \text{nset } \text{vs}. \neg P v))$

using 3 **by** *auto*

have $5: i > 0 \wedge i = \text{nlength } \text{nell} \wedge \text{nfinite } \text{nell} \implies P x \wedge \text{nell} = \text{nappend } (\text{ntaken } (i-1) \text{ nell}) (\text{NNil } x)$

using 1

by *transfer*

$(\text{auto},$
 $\text{metis } \text{eSuc-epred lappend-lbutlast-lldid-id-lfinite lbutlast-conv-ltake llast-conv-lnth}$
 $\text{llength-eq-0 the-enat.simps})$

have $6: i > 0 \wedge i = \text{nlength } \text{nell} \wedge \text{nfinite } \text{nell} \implies (\forall u \in \text{nset } (\text{ntaken } (i-1) \text{ nell}). \neg P u)$

using 1

by *transfer*

$(\text{auto},$
 $\text{metis } \text{in-lset-conv-lnth lbutlast-conv-ltake less-imp-le llength-lbutlast lnth-ltake}$
 $\text{min.strict-order-iff})$

have $7: i > 0 \wedge i = \text{nlength } \text{nell} \wedge \text{nfinite } \text{nell} \implies P x \wedge (\exists \text{us}. \text{nell} = \text{nappend } \text{us } (\text{NNil } x) \wedge \text{nfinite } \text{us} \wedge (\forall u \in \text{nset } \text{us}. \neg P u))$

using $5\ 6$ **by** *auto*

have $8: i > 0 \wedge i < \text{nlength } \text{nell} \implies$

$P\ x \wedge nell = nappend\ (ntaken\ (i-1)\ nell)\ (NCons\ x\ (ndropn\ (i+1)\ nell))$
using 1
by transfer
(auto,
metis co.enat.collapse eSuc-enat ileI1 illess-Suc-eq lappend-ltake-enat-ldropn
ldropn-Suc-conv-ldropn llength-eq-0 min.absorb1 the-enat.simps)
have 9: $i > 0 \wedge i < nlength\ nell \implies nfinite\ (ntaken\ (i-1)\ nell)$
using enat-ord-code(4) nfinite-ntaken **by** blast
have 10: $i > 0 \wedge i < nlength\ nell \implies (\forall\ u \in nset\ (ntaken\ (i-1)\ nell). \neg P\ u)$
using 01 in-nset-conv-nnth[of - (ntaken (i-1) nell)] **by** simp
(metis Suc-pred le-imp-less-Suc min.orderE ntaken-nnth)
have 11: $i > 0 \wedge i < nlength\ nell \implies (\forall\ v \in nset\ (ndropn\ (i+1)\ nell). \neg P\ v)$
using 1 01 in-nset-conv-nnth[of - (ndropn (i+1) nell)]
by simp
(metis Suc-ile-eq illess-Suc-eq is-NNil-ndropn le-add1 le-imp-less-Suc ndropn-Suc-conv-ndropn
ndropn-ndropn ndropn-nlength nlength-NCons not-less)
have 12: $i > 0 \wedge i < nlength\ nell \implies (\exists\ us\ vs.\ nell = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
 $(\forall\ u \in nset\ us. \neg P\ u) \wedge (\forall\ v \in nset\ vs. \neg P\ v))$
using 8 9 10 11 **by** blast
show ?thesis
by (metis 1 12 2 4 7 dual-order.order-iff-strict neq0-conv nlength-eq-enat-nfiniteD
zero-enat-def)
qed

lemma nellist-split-2-first:

assumes $0 < nlength\ nell$
shows $nell = (NCons\ (nnth\ nell\ 0)\ (ntl\ nell))$
using assms **by** (metis ndropn-0 ndropn-Suc-conv-ndropn nellist.sel(5) zero-enat-def)

lemma nellist-split-2-last:

assumes $0 < i$
 $i = nlength\ nell$
 $nfinite\ nell$
shows $nell = nappend\ (ntaken\ (i-1)\ nell)\ (NNil\ (nnth\ nell\ i))$
using assms
by transfer
(simp,
metis Suc-ile-eq co.enat.exhaust-sel eSuc-enat llength-eq-0 ltake-Suc-conv-snoc-lnth ltake-all
order-refl the-enat.simps)

lemma nellist-split-3:

assumes $0 < i$
 $i < nlength\ nell$
shows $nell = nappend\ (ntaken\ (i-1)\ nell)\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (i+1)\ nell))$
using assms **by transfer**
(auto,
metis eSuc-enat eSuc-epred ileI1 illess-Suc-eq lappend-ltake-enat-ldropn ldropn-Suc-conv-ldropn
less-imp-le llength-eq-0 min-def the-enat.simps)

lemma NNil-eq-nfilterD:

assumes $\exists x \in \text{nset } \text{nell}. P x$
 $(\text{NNil } x) = \text{nfilter } P \text{ nell}$
shows $(\exists us \text{ vs. } (\text{nell} = (\text{NNil } x) \vee$
 $(\text{nell} = (\text{NCons } x \text{ vs}) \wedge (\forall v \in \text{nset } \text{vs. } \neg P v)) \vee$
 $(\text{nell} = \text{nappend } us (\text{NNil } x) \wedge \text{nfinite } us \wedge (\forall u \in \text{nset } \text{us. } \neg P u)) \vee$
 $(\text{nell} = \text{nappend } us (\text{NCons } x \text{ vs}) \wedge \text{nfinite } us \wedge$
 $(\forall u \in \text{nset } \text{us. } \neg P u) \wedge (\forall v \in \text{nset } \text{vs. } \neg P v))$
 $) \wedge P x)$
proof –
have 1: $(\exists i \leq \text{nlength } \text{nell}. P (\text{nnth } \text{nell } i))$
by (metis assms(1) in-nset-conv-nnth)
obtain i **where** 2: $i \leq \text{nlength } \text{nell} \wedge P (\text{nnth } \text{nell } i)$
using 1 **by** auto
have 3: $i = 0 \wedge \text{nlength } \text{nell} = 0 \implies P x \wedge \text{nell} = (\text{NNil } x)$
by (metis 2 assms(2) ndropn-0 ndropn-eq-NNil nfilter-NNil the-enat-0 zero-enat-def)
have 4: $i = 0 \wedge \text{nlength } \text{nell} > 0 \implies P x \wedge \text{nell} = (\text{NCons } x (\text{ntl } \text{nell})) \wedge (\forall v \in \text{nset } (\text{ntl } \text{nell}). \neg P v)$
using 2 assms nellist-split-2-first[of nell] nfilter-expand[of nell P]
by (metis nellist.distinct(1) nellist.sel(3) nlast-NNil)
have 5: $i = 0 \wedge \text{nlength } \text{nell} > 0 \implies P x \wedge (\exists \text{vs. } \text{nell} = (\text{NCons } x \text{ vs}) \wedge (\forall v \in \text{nset } \text{vs. } \neg P v))$
using 4 **by** auto
have 60: $0 < i \wedge i = \text{nlength } \text{nell} \wedge \text{nfinite } \text{nell} \implies x = (\text{nnth } \text{nell } i)$
using 2 assms nellist-split-2-last[of i nell]
proof simp
assume a1: $\text{nell} = \text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ nell}) (\text{NNil } (\text{nnth } \text{nell } i))$
assume a2: $P (\text{nnth } \text{nell } i)$
assume a3: $\text{NNil } x = \text{nfilter } P \text{ nell}$
assume a4: $\exists x \in \text{nset } \text{nell}. P x$
have f5: $\forall a \text{ s } p \text{ asa. } ((\exists a. (a::'a) \in \text{nset } \text{as} \wedge p a) \vee \neg \text{nfinite } \text{as} \vee (\forall a. a \notin \text{nset } \text{asa} \vee \neg p a))$
 $\vee \text{nfilter } p (\text{nappend } \text{as } \text{asa}) = \text{nfilter } p \text{ asa}$
by (metis (full-types) nfilter-nappend1)
have f7: $\exists a. a \in \text{nset } (\text{NNil } (\text{nnth } \text{nell } i)) \wedge P a$
using a2 **by** auto
have f8: $\text{nfilter } P (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ nell}) (\text{NNil } (\text{nnth } \text{nell } i))) \neq$
 $\text{nappend } (\text{nfilter } P (\text{ntaken } (i - \text{Suc } 0) \text{ nell})) (\text{nfilter } P (\text{NNil } (\text{nnth } \text{nell } i)))$
using a3 a1 **by** (metis (full-types) is-NNil-nappend nellist.disc(1))
have f9: $\text{nfinite } (\text{ntaken } (i - \text{Suc } 0) \text{ nell})$
by simp
obtain $aaa :: 'a$ **where**
f9: $aaa \in \text{nset } \text{nell} \wedge P aaa$
using a4 **by** blast
then have $aaa \in \text{nset } (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ nell}) (\text{NNil } (\text{nnth } \text{nell } i)))$
using a1 **by** presburger
then have f10: $\forall a. a \notin \text{nset } (\text{ntaken } (i - \text{Suc } 0) \text{ nell}) \vee \neg P a$
using f9 f8 f7 **by** (meson nfilter-nappend nfinite-ntaken)
then show ?thesis
using f9 f7 f5 a3 a1
proof –
have $\text{NNil } (\text{nnth } \text{nell } i) = \text{NNil } x$
using nfilter-NNil[of P (nnth nell i)] **by** (metis (no-types) f10 a1 a3 f5 f7 nfinite-ntaken)
then show ?thesis

```

by blast
qed
qed
have 6:  $0 < i \wedge i = \text{nlength } nell \wedge \text{nfinite } nell \implies P x \wedge nell = \text{nappend } (\text{ntaken } (i-1) \text{ nell}) (NNil x)$ 
  using 2 60 assms nellist-split-2-last[of  $i$   $nell$ ]
  by blast
have 7:  $i > 0 \wedge i = \text{nlength } nell \wedge \text{nfinite } nell \implies (\forall u \in \text{nset } (\text{ntaken } (i-1) \text{ nell}). \neg P u)$ 
  using assms 6 nfilter-nappend[of  $(\text{ntaken } (i-1) \text{ nell}) - P$ ]
  by (metis is-NNil-nappend nellist.disc(1) nfinite-ntaken split-nellist-a)
have 8:  $i > 0 \wedge i = \text{nlength } nell \wedge \text{nfinite } nell \implies P x \wedge (\exists us. nell = \text{nappend } us (NNil x) \wedge \text{nfinite } us \wedge (\forall u \in \text{nset } us. \neg P u))$ 
  using 6 7 by auto
have 9:  $i > 0 \wedge i < \text{nlength } nell \implies P x \wedge nell = \text{nappend } (\text{ntaken } (i-1) \text{ nell}) (NCons x (\text{ndropn } (i+1) \text{ nell}))$ 
  using 2 assms nellist-split-3[of  $i$   $nell$ ]
  nfilter-nappend1[of  $(\text{ntaken } (i-1) \text{ nell}) P (NCons x (\text{ndropn } (i+1) \text{ nell}))]$ 
  nfilter-nappend[of  $(\text{ntaken } (i-1) \text{ nell}) - P$ ]
  proof simp
    assume a1:  $\text{enat } i \leq \text{nlength } nell \wedge P (\text{nnth } nell \ i)$ 
    assume a2:  $NNil x = \text{nfilter } P \text{ nell}$ 
    assume a3:  $\exists x \in \text{nset } nell. P x$ 
    assume a4:  $nell = \text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ nell}) (NCons (\text{nnth } nell \ i) (\text{ndropn } (\text{Suc } i) \text{ nell}))$ 
    have f5:  $P (\text{nnth } (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ nell}) (NCons (\text{nnth } nell \ i) (\text{ndropn } (\text{Suc } i) \text{ nell}))) i$ 
    using a1 a4 by auto
    have f6:  $NNil x = \text{nfilter } P (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ nell}) (NCons (\text{nnth } nell \ i) (\text{ndropn } (\text{Suc } i) \text{ nell})))$ 
    using a2 a4 by auto
    have f7:  $\forall as \ p \ a. ((\exists a. (a::'a) \in \text{nset } as \wedge p \ a) \vee \neg \text{nfinite } as \vee (\forall a. a \notin \text{nset } as \vee \neg p \ a)) \vee \text{nfilter } p (\text{nappend } as \ a) = \text{nfilter } p \ a$ 
    by (metis (full-types) nfilter-nappend1)
    have  $\text{nfilter } P (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ nell}) (NCons (\text{nnth } nell \ i) (\text{ndropn } (\text{Suc } i) \text{ nell}))) = \text{nappend } (\text{nfilter } P (\text{ntaken } (i - \text{Suc } 0) \text{ nell})) (\text{nfilter } P (NCons (\text{nnth } nell \ i) (\text{ndropn } (\text{Suc } i) \text{ nell})))$ 
     $\longrightarrow$ 
     $\text{nappend } (\text{nfilter } P (\text{ntaken } (i - \text{Suc } 0) \text{ nell})) (\text{nfilter } P (NCons (\text{nnth } nell \ i) (\text{ndropn } (\text{Suc } i) \text{ nell}))) =$ 
     $NNil x$ 
    using f6 by presburger
    then have  $\text{nfilter } P (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ nell}) (NCons (\text{nnth } nell \ i) (\text{ndropn } (\text{Suc } i) \text{ nell}))) \neq \text{nappend } (\text{nfilter } P (\text{ntaken } (i - \text{Suc } 0) \text{ nell})) (\text{nfilter } P (NCons (\text{nnth } nell \ i) (\text{ndropn } (\text{Suc } i) \text{ nell})))$ 
    by (metis (no-types) is-NNil-nappend nellist.disc(1))
    then have f9:  $(\forall a. a \notin \text{nset } (NCons (\text{nnth } nell \ i) (\text{ndropn } (\text{Suc } i) \text{ nell})) \vee \neg P a) \vee \text{nfilter } P (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ nell}) (NCons (\text{nnth } nell \ i) (\text{ndropn } (\text{Suc } i) \text{ nell}))) =$ 
     $\text{nfilter } P (NCons (\text{nnth } nell \ i) (\text{ndropn } (\text{Suc } i) \text{ nell})) \vee (\forall a. a \notin \text{nset } (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ nell}) (NCons (\text{nnth } nell \ i) (\text{ndropn } (\text{Suc } i) \text{ nell}))) \vee \neg P a)$ 

```

```

using nfilter-nappend[of (ntaken (i - Suc 0) nell) (NCons (nnth nell i) (ndropn (Suc i) nell)) P]
  nfinite-ntaken[of (i - Suc 0) nell]
by (metis f7)
obtain aaa :: 'a where f10: aaa ∈ nset nell ∧ P aaa
using a3 by blast
then have f11: aaa ∈ nset (nappend (ntaken (i - Suc 0) nell) (NCons (nnth nell i) (ndropn (Suc i)
nell)))
using a4 by presburger
have f12: nnth (nappend (ntaken (i - Suc 0) nell) (NCons (nnth nell i) (ndropn (Suc i) nell))) i ∈
  nset (NCons (nnth nell i) (ndropn (Suc i) nell))
using a4 by fastforce
then have NNil x = nfilter P (NCons (nnth nell i) (ndropn (Suc i) nell))
using f11 f10 f9 f6 f5 by (metis (no-types))
then have NNil x = nfilter P (NCons (nnth (nappend (ntaken (i - Suc 0) nell)
  (NCons (nnth nell i) (ndropn (Suc i) nell)))
    (ndropn (Suc i) nell))
using a4 by presburger
then have f13: ∀ a. a ∉ nset (ndropn (Suc i) nell) ∨ ¬ P a
using f5 by (metis (full-types) nellist.distinct(1) nfilter-NCons)
have nfilter P (NCons (nnth (nappend (ntaken (i - Suc 0) nell)
  (NCons (nnth nell i) (ndropn (Suc i) nell)))
    (ndropn (Suc i) nell)) = NNil x
using ⟨NNil x = nfilter P (NCons (nnth nell i) (ndropn (Suc i) nell))⟩ a4 by presburger
then have x = nnth (nappend (ntaken (i - Suc 0) nell) (NCons (nnth nell i) (ndropn (Suc i) nell))) i
using f13 f5 by simp
then have f14: x = nnth nell i
using a4 by presburger
then have nell = nappend (ntaken (i - Suc 0) nell) (NCons x (ndropn (Suc i) nell))
using a4 by blast
then show P x ∧ nell = nappend (ntaken (i - Suc 0) nell) (NCons x (ndropn (Suc i) nell))
using f14 a1 by blast
qed
have 10: i > 0 ∧ i < nlength nell ⇒ (∀ u ∈ nset (ntaken (i-1) nell). ¬ P u)
using 9 assms nfilter-nappend[of (ntaken (i-1) nell) - P]
by (metis is-NNil-nappend nellist.disc(1) nellist.set-intros(2) nfinite-ntaken)
have 11: i > 0 ∧ i < nlength nell ⇒ (∀ v ∈ nset (ndropn (i+1) nell). ¬ P v)
using 9 10 assms nfilter-nappend[of (ntaken (i-1) nell) - P]
  nfilter-nappend1[of (ntaken (i-1) nell) P (NCons x (ndropn (i+1) nell))]
by (metis nellist.distinct(1) nellist.set-intros(2) nfilter-NCons nfinite-ntaken)
have 12: i > 0 ∧ i < nlength nell ⇒
  P x ∧ (∃ us vs . nell = nappend us (NCons x vs) ∧ nfinite us ∧
    (∀ u ∈ nset us. ¬ P u) ∧ (∀ v ∈ nset vs. ¬ P v))
using 9 10 11 using assms(1) by fastforce
show ?thesis
by (metis 12 2 3 5 8 dual-order.order-iff-strict neq0-conv nlength-eq-enat-nfiniteD
  zero-enat-def)
qed

lemma nfilter-eq-NNilD:
assumes ∃ x ∈ nset nell. P x

```

$nfilter\ P\ nell = (NNil\ x)$
shows $(\exists\ us\ vs.\ .\ (nell = (NNil\ x) \vee$
 $(nell = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs.\ \neg P\ v)) \vee$
 $(nell = nappend\ us\ (NNil\ x) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us.\ \neg P\ u)) \vee$
 $(nell = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
 $(\forall\ u \in nset\ us.\ \neg P\ u) \wedge (\forall\ v \in nset\ vs.\ \neg P\ v))$
 $) \wedge P\ x)$
using *assms NNil-eq-nfilterD[of nell P x]* **by** *simp*

lemma *nfilter-eq-NNil-iff*:

assumes $\exists\ x \in nset\ nell.\ P\ x$
shows $(nfilter\ P\ nell = (NNil\ x)) \longleftrightarrow$
 $(\exists\ us\ vs.\ .\ (nell = (NNil\ x) \vee$
 $(nell = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs.\ \neg P\ v)) \vee$
 $(nell = nappend\ us\ (NNil\ x) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us.\ \neg P\ u)) \vee$
 $(nell = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
 $(\forall\ u \in nset\ us.\ \neg P\ u) \wedge (\forall\ v \in nset\ vs.\ \neg P\ v))$
 $) \wedge P\ x)$

proof –

have 1: $(nfilter\ P\ nell = (NNil\ x)) \implies$
 $(\exists\ us\ vs.\ .\ (nell = (NNil\ x) \vee$
 $(nell = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs.\ \neg P\ v)) \vee$
 $(nell = nappend\ us\ (NNil\ x) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us.\ \neg P\ u)) \vee$
 $(nell = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
 $(\forall\ u \in nset\ us.\ \neg P\ u) \wedge (\forall\ v \in nset\ vs.\ \neg P\ v))$
 $) \wedge P\ x)$

using *assms nfilter-eq-NNilD[of nell P x]* **by** *simp*

have 2: $(\exists\ us\ vs.\ .\ (nell = (NNil\ x) \vee$
 $(nell = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs.\ \neg P\ v)) \vee$
 $(nell = nappend\ us\ (NNil\ x) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us.\ \neg P\ u)) \vee$
 $(nell = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
 $(\forall\ u \in nset\ us.\ \neg P\ u) \wedge (\forall\ v \in nset\ vs.\ \neg P\ v))$
 $) \wedge P\ x) \implies (nfilter\ P\ nell = (NNil\ x))$

using *nfilter-NCons-a nfilter-NNil[of P x]*

by (*auto simp add: nfilter-nappend1*)

show *?thesis*

using 1 2 **by** *blast*

qed

lemma *NCons-eq-nfilterD-help*:

assumes $\neg\ lnull\ ys$

$\neg\ lnull\ xs$

$LCons\ x\ xs = lfilter\ P\ ys$

$xa \in lset\ ys$

$P\ xa$

shows $\exists\ us.\ \neg\ lnull\ us \wedge$

$(\exists\ vs.\ (\exists\ x \in lset\ vs.\ P\ x) \wedge$

$((\exists\ x \in lset\ vs.\ P\ x) \longrightarrow$

$\neg\ lnull\ vs \wedge (ys = LCons\ x\ vs \vee ys = lappend\ us\ (LCons\ x\ vs) \wedge lfinite\ us \wedge$
 $(\forall\ u \in lset\ us.\ \neg P\ u)) \wedge P\ x \wedge xs = lfilter\ P\ vs))$

unfolding *lnull-def* **using** *assms lfilter-eq-LConsD*[of *P ys x xs*]
by (*metis diverge-lfilter-LNil lappend-code*(1) *lfilter-LNil llist.disc*(1))

lemma *NCons-eq-nfilterD*:

assumes $\exists x \in \text{nset } nell. P x$

$(NCons x nell1) = nfilter P nell$

shows $(\exists us vs.$

$(nell = (NCons x vs) \vee$

$nell = nappend us (NCons x vs) \wedge nfinite us \wedge (\forall u \in \text{nset } us. \neg P u)) \wedge$

$(\exists x \in \text{nset } vs. P x) \wedge P x \wedge nell1 = nfilter P vs)$

using *assms* **by** *transfer* (*auto split: if-split-asm simp add: NCons-eq-nfilterD-help*)

lemma *nfilter-eq-NConsD*:

assumes $\exists x \in \text{nset } nell. P x$

$nfilter P nell = (NCons x nell1)$

shows $(\exists us vs.$

$(nell = (NCons x vs) \vee$

$nell = nappend us (NCons x vs) \wedge nfinite us \wedge (\forall u \in \text{nset } us. \neg P u)) \wedge$

$(\exists x \in \text{nset } vs. P x) \wedge P x \wedge nell1 = nfilter P vs)$

using *assms NCons-eq-nfilterD*[of *nell P x nell1*] **by** *simp*

lemma *nfilter-eq-NCons-iff*:

assumes $\exists x \in \text{nset } nell. P x$

shows $nfilter P nell = (NCons x nell1) \longleftrightarrow$

$(\exists us vs.$

$(nell = (NCons x vs) \vee$

$nell = nappend us (NCons x vs) \wedge nfinite us \wedge (\forall u \in \text{nset } us. \neg P u)) \wedge$

$(\exists x \in \text{nset } vs. P x) \wedge P x \wedge nell1 = nfilter P vs)$

proof –

have 1: $nfilter P nell = (NCons x nell1) \implies$

$(\exists us vs.$

$(nell = (NCons x vs) \vee$

$nell = nappend us (NCons x vs) \wedge nfinite us \wedge (\forall u \in \text{nset } us. \neg P u)) \wedge$

$(\exists x \in \text{nset } vs. P x) \wedge P x \wedge nell1 = nfilter P vs)$

using *assms nfilter-eq-NConsD*[of *nell P x nell1*] **by** *simp*

have 2: $(\exists us vs.$

$(nell = (NCons x vs) \vee$

$nell = nappend us (NCons x vs) \wedge nfinite us \wedge (\forall u \in \text{nset } us. \neg P u)) \wedge$

$(\exists x \in \text{nset } vs. P x) \wedge P x \wedge nell1 = nfilter P vs) \implies nfilter P nell = (NCons x nell1)$

using *nfilter-nappend1*[of - *P*] *nfilter-NCons*[of - *P x*]

by (*auto, meson nfilter-NCons, metis*)

show *?thesis* **using** 1 2 **by** *blast*

qed

lemma *nfilter-id-conv*:

assumes $(\exists x \in \text{nset } nell. P x)$

shows $(nfilter P nell = nell) = (\forall x \in \text{nset } nell. P x)$ (**is** *?lhs* = *?rhs*)

using *assms* **by** *transfer* (*auto simp add: lfilter-id-conv*)

lemma *nfilter-idem*:

assumes $(\exists x \in \text{nset } nell. P x)$
shows $\text{nfilter } P (\text{nfilter } P nell) = \text{nfilter } P nell$
using *assms* **by** *transfer auto*

lemma *nfilter-ndropn-nlength*:

assumes $k \leq \text{nlength } nell$
 $\exists x \in \text{nset } ((\text{ndropn } k nell)). P x$
shows $\text{nlength}(\text{nfilter } P ((\text{ndropn } k nell))) \leq \text{nlength } (\text{nfilter } P (nell))$
using *assms*
by *transfer*
(auto simp add: min-def dest: in-lset-ldropnD,
metis co.enat.exhaust-sel epred-le-epredI iless-Suc-eq lfilter-ldropn-llength llength-eq-0)

1.2.20 Setup for Lifting/Transfer

Relator and predicate properties

abbreviation *nellist-all* == *pred-nellist*

Transfer rules for the Transfer package

context *includes* *lifting-syntax*

begin

lemma *set1-pre-nellist-transfer* [*transfer-rule*]:

$(\text{rel-pre-nellist } A \ C ==> \text{rel-set } A) \text{ set1-pre-nellist set1-pre-nellist}$
by *(auto simp add: rel-pre-nellist-def vimage2p-def rel-fun-def set1-pre-nellist-def rel-set-def*
collect-def sum-set-defs prod-set-defs elim: rel-sum.cases split: sum.split-asm)

lemma *set2-pre-nellist-transfer* [*transfer-rule*]:

$(\text{rel-pre-nellist } A \ C ==> \text{rel-set } C) \text{ set2-pre-nellist set2-pre-nellist}$
by *(auto simp add: rel-pre-nellist-def vimage2p-def rel-fun-def set2-pre-nellist-def*
rel-set-def collect-def sum-set-defs prod-set-defs elim: rel-sum.cases split: sum.split-asm)

lemma *NCons-transfer2* [*transfer-rule*]:

$(A ==> \text{nellist-all2 } A ==> \text{nellist-all2 } A) \text{ NCons NCons}$

unfolding *rel-fun-def* **by** *simp*

declare *NCons-transfer* [*transfer-rule*]

lemma *case-nellist-transfer* [*transfer-rule*]:

$((A ==> C) ==> (A ==> \text{nellist-all2 } A ==> C) ==> \text{nellist-all2 } A ==> C)$
case-nellist case-nellist

unfolding *rel-fun-def*

by *(simp add: nellist-all2-NNil1 nellist-all2-NNil2 split: nellist.split)*

lemma *unfold-nellist-transfer* [*transfer-rule*]:

$((A ==> (=)) ==> (A ==> C) ==> (A ==> C) ==> (A ==> A) ==> A ==> \text{nellist-all2 } C)$

unfold-nellist unfold-nellist

proof *(rule rel-funI)+*

fix *IS-NNIL1* :: '*a* \Rightarrow bool **and** IS-NNIL2

```

  NLAST1 NLAST2 NHD1 NHD2 NTL1 NTL2 x y
assume rel: (A ==> (=)) IS-NNIL1 IS-NNIL2 (A ==> C) NLAST1 NLAST2
  (A ==> C) NHD1 NHD2 (A ==> A) NTL1 NTL2
and A x y
show nellist-all2 C (unfold-nellist IS-NNIL1 NLAST1 NHD1 NTL1 x)
  (unfold-nellist IS-NNIL2 NLAST2 NHD2 NTL2 y)
using ⟨A x y⟩ using rel by (coinduction arbitrary: x y) (auto 4 4 elim: rel-funE)
qed

lemma corec-nellist-transfer [transfer-rule]:
  ((A ==> (=)) ==> (A ==> C) ==> (A ==> C) ==> (A ==> (=)) ==> (A ==>
nellist-all2 C)
  ==> (A ==> A) ==> A ==> nellist-all2 C) corec-nellist corec-nellist
by (simp add: nellist.corec-transfer)

lemma ntl-transfer2 [transfer-rule]:
  (nellist-all2 A ==> nellist-all2 A) ntl ntl
unfolding ntl-def[abs-def] by transfer-prover
declare ntl-transfer [transfer-rule]

lemma nset-transfer2 [transfer-rule]:
  (nellist-all2 A ==> rel-set A) nset nset
by (simp add: nellist.set-transfer)

lemma nmap-transfer4 [transfer-rule]:
  ((A ==> B) ==> nellist-all2 A ==> nellist-all2 B) nmap nmap
by (simp add: nellist.map-transfer)
declare nmap-transfer [transfer-rule]

lemma is-NNil-transfer2 [transfer-rule]:
  (nellist-all2 A ==> (=)) is-NNil is-NNil
by(auto dest: nellist-all2-is-NNilD)
declare is-NNil-transfer [transfer-rule]

lemma snocn-transfer2 [transfer-rule]:
  (nellist-all2 A ==> A ==> nellist-all2 A) snocn snocn
unfolding rel-fun-def
by (metis nappend-snocn nellist-all2-NNil nellist-all2-nappendI)
declare snocn-transfer[transfer-rule]

lemma nappend-transfer [transfer-rule]:
  (nellist-all2 A ==> ( nellist-all2 A) ==> nellist-all2 A) nappend nappend
by(auto intro: nellist-all2-nappendI elim: rel-funE)
declare nappend.transfer [transfer-rule]

lemma lappendn-transfer [transfer-rule]:
  (llist-all2 A ==> nellist-all2 A ==> nellist-all2 A) lappendn lappendn
unfolding rel-fun-def
by transfer(auto intro: llist-all2-lappendI)
declare lappendn.transfer [transfer-rule]

```

```

lemma nellist-of-llist-a-transfer2 [transfer-rule]:
  (A ==> llist-all2 A ==> nellist-all2 A) nellist-of-llist-a nellist-of-llist-a
by (simp add: rel-funI)
declare nellist-of-llist-a-transfer [transfer-rule]

lemma nlength-transfer [transfer-rule]:
  (nellist-all2 A ==> (=)) nlength nlength
by(auto dest: nellist-all2-nlengthD)
declare nlength.transfer [transfer-rule]

lemma ndropn-transfer [transfer-rule]:
  ((=) ==> nellist-all2 A ==> nellist-all2 A) ndropn ndropn
unfolding rel-fun-def
by transfer (auto intro: llist-all2-ldropnI simp add: llist-all2-ldropnI llist-all2-llengthD )
declare ndropn.transfer [transfer-rule]

lemma ntake-transfer [transfer-rule]:
  ((=) ==> nellist-all2 A ==> nellist-all2 A) ntake ntake
unfolding rel-fun-def
by transfer (auto simp add: llist-all2-ltakeI)
declare ntake.transfer [transfer-rule]

lemma ntaken-transfer [transfer-rule]:
  ((=) ==> nellist-all2 A ==> nellist-all2 A) ntaken ntaken
unfolding rel-fun-def
by transfer (auto simp add: llist-all2-ltakeI)
declare ntaken.transfer [transfer-rule]

lemma nzip-transfer [transfer-rule]:
  (nellist-all2 A ==> nellist-all2 B ==> nellist-all2 (rel-prod A B)) nzip nzip
by (auto intro: nellist-all2-nzipI)

lemma niterates-transfer [transfer-rule]:
  ((A ==> A) ==> A ==> nellist-all2 A) niterates niterates
unfolding rel-fun-def
apply transfer
using iterates-transfer unfolding rel-fun-def
by blast

lemma nfilter-transfer [transfer-rule]:
  ( (A ==> (=)) ==> nellist-all2 A ==> nellist-all2 A) nfilter nfilter
unfolding rel-fun-def
by transfer
  (auto intro: llist-all2-lfilterI dest: llist-all2-lfiniteD llist-all2-lsetD1,
    meson llist-all2-lsetD2)
declare nfilter.transfer [transfer-rule]

lemma nconcat-transfer [transfer-rule]:

```

```

  ( nellist-all2 (nellist-all2 A) ==> nellist-all2 A ) nconcat nconcat
unfolding rel-fun-def
using nellist-all2-nconcatI
by auto
declare nconcat.transfer [transfer-rule]

lemma nellist-all2-rsp:
  assumes R1:  $\forall x y. R1\ x\ y \longrightarrow (\forall a\ b. R1\ a\ b \longrightarrow S\ x\ a = T\ y\ b)$ 
  and R2:  $\forall x y. R2\ x\ y \longrightarrow (\forall a\ b. R2\ a\ b \longrightarrow S'\ x\ a = T'\ y\ b)$ 
  and xsys: nellist-all2 R1 xs ys
  and xs'ys': nellist-all2 R1 xs' ys'
  shows nellist-all2 S xs xs' = nellist-all2 T ys ys'
proof
  assume nellist-all2 S xs xs'
  with xsys xs'ys' show nellist-all2 T ys ys'
  proof(coinduction arbitrary: ys ys' xs xs')
    case (ilist-all2 ys ys' xs xs')
    thus ?case
    by cases (auto 4 4 simp add: nellist-all2-NCons1 nellist-all2-NCons2 nellist-all2-NNil1
      nellist-all2-NNil2 dest: R1[rule-format] R2[rule-format])
  qed
next
  assume nellist-all2 T ys ys'
  with xsys xs'ys' show nellist-all2 S xs xs'
  proof(coinduction arbitrary: xs xs' ys ys')
    case (ilist-all2 xs xs' ys ys')
    thus ?case
    by cases (auto 4 4 simp add: nellist-all2-NCons1 nellist-all2-NCons2 nellist-all2-NNil1
      nellist-all2-NNil2 dest: R1[rule-format] R2[rule-format])
  qed
qed

lemma nellist-all2-transfer2 [transfer-rule]:
  ((R1 ==> R1 ==> (=) ) ==>
    nellist-all2 R1 ==> nellist-all2 R1 ==> (=)) nellist-all2 nellist-all2
by (simp add: nellist-all2-rsp rel-fun-def)
declare nellist-all2-transfer [transfer-rule]

end

Delete lifting rules for 'a nellist because the parametricity rules take precedence over most of the
transfer rules. They can be restored by including the bundle nellist.lifting.

lifting-update nellist.lifting
lifting-forget nellist.lifting

end

```

1.3 Extra operations on non-empty conductive lists

The operations `ndropns`, `nkfilter`, `nleast`, `nidx`, `nfuse`, `nbutlast`, `nsubn`, `nridx`, `nlastnfirst`, `nfusecat` and `nrev` are defined for `nellists` together with a library of lemmas.

```
theory NELList-Extras
  imports NELList
begin
```

1.3.1 `ndropns`

```
primcorec ndropns :: 'a nellist  $\Rightarrow$  'a nellist nellist
```

```
  where ndropns nell =
    (case nell of (NNil b)  $\Rightarrow$  (NNil (NNil b)) |
      (NCons x nell')  $\Rightarrow$  (NCons (NCons x nell') (ndropns nell')))
```

```
simps-of-case ndropns-code [code, simp, nitpick-simp]: ndropns.code
```

```
lemma ndropns-simps [simp]:
```

```
  shows nhd-ndropns:  $\neg$  is-NNil nell  $\Longrightarrow$  nhd (ndropns nell) = nell
  and ndropns-NCons: ntl (ndropns nell) = (case nell of (NNil b)  $\Rightarrow$  (NNil (NNil b)) |
    (NCons x nell')  $\Rightarrow$  ndropns nell')
```

```
by (auto simp add: nellist.case-eq-if)
   (metis ndropns-code(1) nellist.collapse(1) nellist.sel(4))
```

```
lemma ndropns-nnth:
```

```
  assumes i  $\leq$  nlength nell
  shows nnth (ndropns nell) i = ndropn i nell
```

```
using assms
```

```
proof (induction i arbitrary: nell)
```

```
case 0
```

```
then show ?case
```

```
  proof (cases nell)
```

```
    case (NNil x1)
```

```
    then show ?thesis by (simp add: nnth-NNil)
```

```
  next
```

```
    case (NCons x21 x22)
```

```
    then show ?thesis by simp
```

```
  qed
```

```
next
```

```
case (Suc i)
```

```
then show ?case
```

```
  proof (cases nell)
```

```
    case (NNil x1)
```

```
    then show ?thesis by (simp add: nnth-NNil)
```

```
  next
```

```
    case (NCons x21 x22)
```

```
    then show ?thesis using Suc by (simp add: Suc-ile-eq)
```

```
  qed
```

```
qed
```

lemma *ndropns-nlength*:

$nlength\ (ndropns\ nell) = (nlength\ nell)$

by (*coinduction arbitrary: nell rule: enat-coinduct*)
(case-tac nell, auto)

lemma *in-nset-ndropns*:

$nell \in nset(ndropns\ nellx) \longleftrightarrow (\exists\ i.\ i \leq nlength\ nellx \wedge nell = ndropn\ i\ nellx)$

by (*metis in-nset-conv-nnth ndropns-nlength ndropns-nnth*)

lemma *nset-ndropns*:

$nset\ (ndropns\ nell) = \{ ndropn\ i\ nell \mid i.\ i \leq nlength\ nell \}$

using *in-nset-conv-nnth[of - ndropns nell] nset-conv-nnth[of ndropns nell]*

using *in-nset-ndropns[of - nell]* **by** *auto*

lemma *nmap-first-ndropns*:

$nmap\ (\lambda\ nell.\ nnth\ nell\ 0)\ (ndropns\ nell) = nell$

by (*coinduction arbitrary: nell*)

(case-tac nell, auto simp add: nnth-NNil)

lemma *ndropn-ndropns*:

assumes $i \leq nlength(ndropns\ nell)$

shows $ndropn\ i\ (ndropns\ nell) = ndropns\ (ndropn\ i\ nell)$

using *assms*

proof (*coinduction arbitrary: nell i*)

case (*Eq-nellist ia nellx*)

then show *?case*

proof –

have *1: enat nellx ≤ nlength (ndropns ia) ⇒*

is-NNil (ndropn nellx (ndropns ia)) = is-NNil (ndropns (ndropn nellx ia))

by (*simp add: is-NNil-ndropn ndropns-nlength*)

have *2: enat nellx ≤ nlength (ndropns ia) ⇒*

(is-NNil (ndropn nellx (ndropns ia)) →

is-NNil (ndropns (ndropn nellx ia)) →

nlast (ndropn nellx (ndropns ia)) = nlast (ndropns (ndropn nellx ia)))

by (*metis dual-order.antisym ndropn-eq-NNil ndropns.disc(2) ndropns.simps(3) ndropns-nlength ndropns-nnth nellist.case-eq-if nellist.collapse(1) the-enat.simps*)

have *3: enat nellx ≤ nlength (ndropns ia) ⇒*

(¬ is-NNil (ndropn nellx (ndropns ia)) →

¬ is-NNil (ndropns (ndropn nellx ia)) →

nhd (ndropn nellx (ndropns ia)) = nhd (ndropns (ndropn nellx ia)))

by (*metis Nat.add-0-right ndropn-nnth ndropns.disc(1) ndropns-nlength ndropns-nnth nhd-conv-nnth nhd-ndropns*)

have *4: enat nellx ≤ nlength (ndropns ia) ⇒*

(¬ is-NNil (ndropn nellx (ndropns ia)) →

¬ is-NNil (ndropns (ndropn nellx ia)) →

(∃ nell i.

ntl (ndropn nellx (ndropns ia)) = ndropn i (ndropns nell) ∧

ntl (ndropns (ndropn nellx ia)) = ndropns (ndropn i nell) ∧

enat i ≤ nlength (ndropns nell)))

by (*metis Suc-ile-eq is-NNil-ndropn ndropn-Suc-conv-ndropn ndropns-code(2) ndropns-nlength*)

```

      nellist.sel(5) order.order-iff-strict)
show ?thesis
using 1 2 3 4 Eq-nellist by blast
qed
qed

lemma ndropns-nfilter-nnth:
  assumes  $i \leq \text{nlength } (\text{nfilter } P \text{ (ndropns nell)})$ 
     $\exists \text{ nelly} \in \text{nset}(\text{ndropns nell}). P \text{ nelly}$ 
  shows  $P (\text{nnth } (\text{nfilter } P \text{ (ndropns nell)}) i)$ 
  using assms using nset-nfilter[of ndropns nell P]
  by (metis (full-types) Int-iff in-nset-conv-nnth mem-Collect-eq )

```

```

lemma nnth-zero-ndropn:
   $\text{nnth } (\text{ndropn } n \text{ nell}) 0 = \text{nnth } \text{nell } n$ 
by simp

```

```

lemma in-nset-ndropns-nhd:
   $x \in \text{nset } \text{nell} \longleftrightarrow (\exists \text{ ys}. x = (\text{nnth } \text{ys } 0) \wedge \text{ys} \in \text{nset}(\text{ndropns nell}))$ 
by auto
  (metis in-nset-conv-nnth ndropns-nlength nmap-first-ndropns nnth-nmap,
   metis Nat.add-0-right in-nset-conv-nnth in-nset-ndropns ndropn-nnth)

```

```

lemma nset-ndropns-nhd:
   $\text{nset } \text{nell} = \{(\text{nnth } \text{nelly } 0) \mid \text{nelly}. \text{nelly} \in \text{nset}(\text{ndropns nell}) \}$ 
by auto
  (meson in-nset-ndropns-nhd,
   metis in-nset-conv-nnth in-nset-ndropns nnth-zero-ndropn)

```

```

lemma nellist-all2-ndropnsI:
   $\text{nellist-all2 } A \text{ nellx nelly} \implies \text{nellist-all2 } (\text{nellist-all2 } A) (\text{ndropns nellx}) (\text{ndropns nelly})$ 
by (coinduction arbitrary: nellx nelly)
  (auto simp add: nellist.case-eq-if dest: nellist-all2-nhdD nellist-all2-is-NNilD
   intro: nellist-all2-ntII)

```

1.3.2 Definitions

```

context
includes nellist.lifting
begin

```

```

lift-definition nkfilter :: ('a  $\Rightarrow$  bool)  $\Rightarrow$  nat  $\Rightarrow$  'a nellist  $\Rightarrow$  nat nellist
is  $\lambda P n \text{ xs}. \text{ (if } \text{lnull}(\text{kfilter } P \text{ } n \text{ xs}) \text{ then } (\text{iterates } \text{Suc } n) \text{ else } \text{kfilter } P \text{ } n \text{ xs})$ 
by simp

```

```

lift-definition nleast :: ('a  $\Rightarrow$  bool)  $\Rightarrow$  'a nellist  $\Rightarrow$  nat
is  $\lambda P \text{ xs}. \text{ lleast } P \text{ xs}$ 
by auto

```

```

lift-definition nridx :: ('a  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$  'a nellist  $\Rightarrow$  bool

```


is $\lambda R \ xs. \text{ridx } R \ xs$
 by *auto*

lift-definition *nidx* :: *nat nelist* \Rightarrow *bool*
 is $\lambda \ xs. \text{lidx } xs$
 by *auto*

lift-definition *nbutlast* :: '*a nelist* \Rightarrow '*a nelist*
 is $\lambda \ xs. (\text{if } \text{lnull } (\text{lbutlast } xs) \text{ then } (LCons (\text{llast } xs) \text{ LNil}) \text{ else } \text{lbutlast } xs)$
 by *auto*

lift-definition *nfuse* :: '*a nelist* \Rightarrow '*a nelist* \Rightarrow '*a nelist*
 is $\lambda \ xs \ ys. \text{lfuse } xs \ ys$
 using *lfuse-conv-lnull* by *blast*

lift-definition *nsubn* :: '*a nelist* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow '*a nelist*
 is $(\lambda \ xs \ n1 \ n2. \text{lsubc } n1 \ n2 \ xs)$
unfolding *lsubc-def*
 by *auto*
 (*metis co.enat.exhaust-sel dual-order.refl enat-ile iless-Suc-eq leD llength-eq-0*)

lift-definition *nlastnfirst* :: '*a nelist nelist* \Rightarrow *bool*
 is $\lambda \ xss. \text{llastlfirst } xss$
apply (*simp add: pcr-nelist-def cr-nelist-def OO-def rel-fun-def llist.rel-eq*)
unfolding *llastlfirst-def*
 using *llist-all2-llist-of-nelist-1* by *blast*

lemma *nfusecat-help*:
assumes $\exists y. \text{llist-all2 } (\lambda x \ z. \text{llist-of-nelist } z = x) \text{ llist1 } y \wedge y \neq \text{LNil} \wedge$
 $\text{llist-all2 } (\lambda x \ z. \text{llist-of-nelist } z = x) \text{ llist2 } y$
shows $\text{lfusecat } \text{llist1} \neq \text{LNil} \wedge \text{lfusecat } \text{llist1} = \text{lfusecat } \text{llist2}$
proof –
have 1: $\text{lfusecat } \text{llist1} = \text{lfusecat } \text{llist2}$
 using *assms llist-all2-llist-of-nelist-1 lnull-def* by *blast*
have 2: $\text{lfusecat } \text{llist1} \neq \text{LNil}$
 using *assms apply auto*
 using *lfusecat-not-lnull-var llist-all2-lnullD llist-all2-lsetD1* by *fastforce*
show *?thesis* using 1 2 by *auto*
 qed

lift-definition *nfusecat* :: '*a nelist nelist* \Rightarrow '*a nelist*
 is $\lambda \ xss. \text{lfusecat } xss$
 using *nfusecat-help*
 by (*simp add: pcr-nelist-def OO-def cr-nelist-def nelist.pcr-cr-eq rel-fun-def lnull-def*
llist-all2-cases llist-all2-rsp llist.rel-eq not-lnull-conv-llist-of-nelist) *blast*

lift-definition *nrev* :: '*a nelist* \Rightarrow '*a nelist*
 is $\lambda \ xs. \text{if } \text{lfinite } xs \text{ then } \text{lrev } xs \text{ else } xs$
 by *auto*

1.3.3 nbutlast

lemma *nbutlast-NNil*[simp]:
 nbutlast (NNil *x*) = (NNil *x*)

apply *transfer*
by *auto*

lemma *nbutlast-snoc* [simp]:
 nbutlast (nappend *nell* (NNil *x*)) = *nell*
apply *transfer*
by *auto*

lemma *nbutlast-def1*:
 nbutlast nell =
 (case *nell* of (NNil *x*) \Rightarrow (NNil *x*) |
 (NCons *x nell1*) \Rightarrow
 (case *nell1* of (NNil *y*) \Rightarrow (NNil *x*) |
 (NCons *z nell2*) \Rightarrow (NCons *x* (*nbutlast nell1*))))
proof (cases *nell*)
case (NNil *x1*)
then show ?thesis **by** *simp*
next
case (NCons *x21 x22*)
then show ?thesis
 proof –
 have 1: *is-NNil x22* \Longrightarrow *nbutlast* (NCons *x21 x22*) = (NNil *x21*)
 by (*metis* nappend-NNil *nbutlast-snoc nellist.collapse*(1))
 have 2: \neg *is-NNil x22* \Longrightarrow *nbutlast* (NCons *x21 x22*) = (NCons *x21* (*nbutlast x22*))
 apply *transfer*
 by *auto*
 (*metis* lhd-LCons-ltl *lhist.collapse*(1),
 metis lbutlast-simps(3) lhd-LCons-ltl)
 show ?thesis
 by (*simp* add: 1 2 NCons *nellist.case-eq-if*)
 qed
qed

lemma *ntl-nbutlast*:
 ntl (*nbutlast nell*) =
 (if *is-NNil nell* then *ntl nell* else
 (if *is-NNil* (*ntl nell*) then NNil (*nhd nell*) else *nbutlast* (*ntl nell*)))
by (*auto simp* add: *nbutlast-def1 nellist.case-eq-if*)

lemma *nbutlast-not-nfinite*:
 assumes \neg *nfinite nell*
 shows *nbutlast nell* = *nell*
 using *assms*
 apply *transfer*
 by *simp*
 (*metis* lbutlast.disc-iff(2) lbutlast-not-lfinite)

lemma *nbutlast-nfinite*:

nfinite (*nbutlast nell*) \longleftrightarrow *nfinite nell*

apply *transfer*

by (*metis* (*full-types*) *lbutlast-lfinite lbutlast-not-lfinite lfinite-LNil lfinite-code*(2))

lemma *nlenght-nbutlast* [*simp*]:

nlenght (*nbutlast nell*) = *epred* (*nlenght nell*)

apply *transfer*

by (*simp*, *simp add: epred-lenght*)

lemma *nbutlast-nappend*:

nbutlast (*nappend nellx nelly*) =

(*if is-NNil nelly then nellx else nappend nellx* (*nbutlast nelly*))

apply *transfer*

by *simp*

(*metis lbutlast-lappend lbutlast-snoc lhd-LCons-ltl lnull-def*)

lemma *nappend-nbutlast-nlast-id-nfinite*:

assumes *nfinite nellx*

\neg *is-NNil nellx*

shows (*nappend* (*nbutlast nellx*) (*NNil* (*nlast nellx*))) = *nellx*

using *assms*

apply *transfer*

by *simp*

(*metis lhd-LCons-ltl llist.collapse*(1))

lemma *nappend-nbutlast-nlast-id-not-nfinite*:

assumes \neg *nfinite nellx*

\neg *is-NNil nellx*

shows (*nappend* (*nbutlast nellx*) (*NNil* (*nlast nellx*))) = *nellx*

using *assms*

apply *transfer*

by *simp*

(*metis lappend-inf lbutlast.disc-iff*(2) *lbutlast-snoc*)

lemma *nappend-nbutlast-nlast-id* [*simp*]:

shows \neg *is-NNil nell* \implies (*nappend* (*nbutlast nell*) (*NNil* (*nlast nell*))) = *nell*

using *nappend-nbutlast-nlast-id-nfinite nappend-nbutlast-nlast-id-not-nfinite* **by** *blast*

lemma *nbutlast-eq-NNil-conv*:

nbutlast nell = (*NNil* (*nfirst nell*)) \longleftrightarrow

nell = (*NNil* (*nfirst nell*)) \vee ($\exists x.$ *nell* = (*NCons* *x* (*NNil* (*nlast nell*))))

apply *transfer*

by (*auto simp add: llist.expand lbutlast-not-lfinite*)

(*metis lhd-LCons-ltl llast-LCons*,

metis eq-LConsD lbutlast-eq-LNil-conv lbutlast-ltl lhd-LCons-ltl llast-LCons2 llast-singleton,

simp add: eq-LConsD,

simp add: eq-LConsD lbutlast-eq-LCons-conv,

metis lfinite-LNil lfinite-ltl llist.collapse(1))

lemma *nbutlast-eq-NCons-conv*:

$nbutlast\ nell = (NCons\ x\ ys) \longleftrightarrow$
 $nell = (NCons\ x\ (nappend\ ys\ (NNil\ (nlast\ nell))))$

apply *transfer*

by (*auto simp add: eq-LConsD lbutlast-eq-LCons-conv llast-linfinite*)

lemma *nbutlast-conv-ntake*:

$nbutlast\ nell = ntake\ (epred\ (nlength\ nell))\ nell$

apply *transfer*

by *simp*

(*metis co.enat.exhaust-sel ileI1 ileSS-eSuc0 lappend-lbutlast-llast-id lappend-lnull1*
lbutlast.disc-iff(2) lbutlast-conv-ltake llength-eq-0 llength-lbutlast ltake-all)

lemma *nmap-nbutlast*:

$nmap\ f\ (nbutlast\ nell) = nbutlast\ (nmap\ f\ nell)$

apply *transfer*

by *simp*

(*metis lfinite-ltl llast-lmap lmap-lbutlast lnull-imp-lfinite*)

lemma *snocs-eq-iff-nbutlast*:

$nappend\ nell\ (NNil\ x) = nell1 \longleftrightarrow$
 $((\ nfinite\ nell1 \wedge \neg\ is-NNil\ nell1 \wedge nbutlast\ nell1 = nell \wedge nlast\ nell1 = x)$
 $\vee (\neg\ nfinite\ nell1 \wedge nbutlast\ nell = nell1))$

by (*metis is-NNil-nappend nappend-inf nappend-nbutlast-nlast-id nbutlast-nfinite nbutlast-snoc*
nlast-NNil nlast-nappend)

lemma *in-nset-nbutlastD*:

$x \in nset(nbutlast\ nell) \implies x \in nset\ nell$

by (*metis in-nset-snocn-iff nappend-nbutlast-nlast-id-nfinite nappend-snocn nbutlast-NNil*
nbutlast-not-nfinite nellist.collapse(1))

lemma *in-nset-nbutlast-nappendI*:

$x \in nset\ (nbutlast\ nell) \vee (nfinite\ nell \wedge \neg\ is-NNil\ nell1 \wedge x \in nset(nbutlast\ nell1)) \implies$
 $x \in nset\ (nbutlast\ (nappend\ nell\ nell1))$

unfolding *nbutlast-nappend*

by (*metis (full-types) Un-iff in-nset-nbutlastD nset-nappend*)

lemma *nnth-nbutlast*:

assumes $n \leq nlength(nbutlast\ nell)$

shows $nnth\ (nbutlast\ nell)\ n = nnth\ nell\ n$

by (*metis assms nappend-nbutlast-nlast-id-nfinite nbutlast-eq-NNil-conv nbutlast-not-nfinite*
ndropn-is-NNil ndropn-nfirst nellist.collapse(1) nnth-nappend1 nnth-nlast)

1.3.4 nsubn

lemma *nsubn-def1*:

$nsubn\ nell\ i\ j = ntaken\ (j-i)\ (ndropn\ i\ nell)$

apply *transfer*

unfolding *lsubc-def*

by *auto*

(*metis co.enat.exhaust-sel dual-order.refl enat-ile illess-Suc-eq leD llength-eq-0* ,
metis eSuc-enat enat-the-enat infinity-ileE ldrop-enat min.cobounded1)

lemma *nsubn-same:*

shows $nsubn\ nell\ k\ k = (NNil\ (nnth\ nell\ k))$

unfolding *nsubn-def1*

by (*simp add: ndropn-nfirst*)

lemma *nsubn-nlength:*

$nlength(nsubn\ nell\ i\ j) = \min\ (j-i)\ (nlength\ nell - i)$

by (*simp add: nsubn-def1*)

lemma *nsubn-nlength-gr-one:*

assumes $k < n$

$n \leq nlength\ nell$

shows $0 < nlength\ (nsubn\ nell\ k\ n)$

using *assms*

unfolding *nsubn-nlength*

by (*metis enat-minus-mono1 enat-ord-simps(2) idiff-enat-enat min-def zero-enat-def zero-less-diff*)

lemma *nsubn-nfinite:*

shows *nfinite* (*nsubn nell k n*)

by (*simp add: nsubn-def1*)

lemma *nsubn-nnth:*

shows $nnth\ (nsubn\ nell\ i\ j)\ k = nnth\ nell\ (i + \min\ k\ (j - i))$

unfolding *nsubn-def1*

using *ntaken-nnth[of j-i (ndropn i nell) k] ndropn-nnth[of i nell (min k (j - i))]*

by *simp*

lemma *ntaken-ndropn:*

$ntaken\ n\ (ndropn\ k\ nell) = nsubn\ nell\ k\ (n+k)$

by (*simp add: nsubn-def1*)

lemma *ntaken-ndropn-nfirst:*

$nfirst\ (ntaken\ n\ (ndropn\ k\ nell)) = nnth\ nell\ k$

by (*metis min-0L ndropn-nfirst ntaken-0 ntaken-ntaken nlast-NNil*)

lemma *ntaken-ndropn-nfirst-a:*

$nfirst\ (ntaken\ n\ (ndropn\ k\ nell)) = nfirst(ndropn\ k\ nell)$

by (*simp add: ndropn-nfirst ntaken-ndropn-nfirst*)

lemma *ntaken-ndropn-nlast:*

$nlast(ntaken\ n\ (ndropn\ k\ nell)) = nnth\ nell\ (n+k)$

by (*simp add: add commute ntaken-nlast*)

lemma *nsubn-nfirst*:

$nfirst (nsubn\ nell\ i\ j) = nnth\ nell\ i$

by (*simp add: nsubn-def1 ntaken-ndropn-nfirst*)

lemma *nsubn-nlast*:

$nlast (nsubn\ nell\ i\ j) = nnth\ nell\ (j - i + i)$

by (*simp add: nsubn-def1 ntaken-ndropn-nlast*)

lemma *nsubn-ndropn*:

assumes $i < j$

shows $nsubn\ nell\ (i+k)\ (j+k) = nsubn\ (ndropn\ k\ nell)\ i\ j$

using *assms*

by (*simp add: add commute ndropn-ndropn nsubn-def1*)

lemma *pref-ntaken-3*:

$(ntaken\ i\ (ntaken\ (i+k)\ nell)) = (ntaken\ i\ nell)$

by (*metis le-add1 min.orderE ntaken-ntaken*)

lemma *ntaken-ndropn-swap-nlength*:

assumes $ia + i \leq nlength\ nell$

shows $nlength\ (ntaken\ ia\ (ndropn\ i\ nell)) = nlength\ (ndropn\ i\ (ntaken\ (ia+i)\ nell))$

using *assms ndropn-nlength[of i (ntaken (ia+i) nell)] ntaken-nlength[of ia (ndropn i nell)]*

by *auto*

(*metis add-diff-cancel-right' enat-minus-mono1 idiff-enat-enat min.orderE*)

lemma *ntaken-ndropn-swap-nnth*:

assumes $m \leq ia$

$ia + i \leq nlength\ nell$

shows $nnth\ (ntaken\ ia\ (ndropn\ i\ nell))\ m = nnth\ (ndropn\ i\ (ntaken\ (ia+i)\ nell))\ m$

using *assms*

by (*simp add: ntaken-nnth*)

lemma *nellist-eq-nnth-eq*:

$(nellx = nelly) \longleftrightarrow nlength\ nellx = nlength\ nelly \wedge (\forall\ i \leq nlength\ nellx. nnth\ nellx\ i = nnth\ nelly\ i)$

by *transfer*

(*metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0 llist-eq-lnth-eq min-absorb1 the-enat.simps*)

lemma *ntaken-ndropn-swap*:

assumes $ia + i \leq nlength\ nell$

shows $(ntaken\ ia\ (ndropn\ i\ nell)) = (ndropn\ i\ (ntaken\ (ia+i)\ nell))$

using *assms nellist-eq-nnth-eq[of (ntaken ia (ndropn i nell)) (ndropn i (ntaken (ia+i) nell))]*

using *ntaken-ndropn-swap-nlength*

using *ntaken-ndropn-swap-nnth by fastforce*

lemma *ntaken-nsubn*:

assumes $n \leq nlength\ nell$

$m + k \leq n$

shows $ntaken\ m\ (nsubn\ nell\ k\ n) = nsubn\ nell\ k\ (m+k)$
using *assms*
unfolding *nsubn-def1*
by *simp*

lemma *ndropn-nsubn*:

assumes $n \leq nlength\ nell$

$m + k \leq n$

shows $ndropn\ m\ (nsubn\ nell\ k\ n) = nsubn\ nell\ (m+k)\ n$

proof –

have 1: $ntaken\ (n-k)\ (ndropn\ k\ nell) = ndropn\ k\ (ntaken\ n\ nell)$

by (*metis add-leD2 assms(1) assms(2) diff-add ntaken-ndropn-swap plus-enat-simps(1)*)

have 2: $ndropn\ m\ (ndropn\ k\ (ntaken\ n\ nell)) =$
 $ndropn\ (m+k)\ (ntaken\ n\ nell)$

by (*simp add: add.commute ndropn-ndropn*)

show ?thesis **unfolding** *nsubn-def1* **using** 1 2

by (*simp add: assms(1) assms(2) ntaken-ndropn-swap*)

qed

lemma *ntl-nsubn*:

assumes $n \leq nlength\ nell$

$k \leq n$

shows $ntl(nsubn\ nell\ k\ n) = (if\ n=k\ then\ (NNil\ (nnth\ nell\ k))\ else\ nsubn\ (ntl\ nell)\ k\ (n-1))$

using *assms* **unfolding** *nsubn-def1*

using *ntl-ntaken[of n-k ndropn k nell]* *ntl-ndropn[of k nell]*

by (*metis diff-diff-cancel diff-right-commute diff-zero nellist.sel(4) nsubn-def1 nsubn-same*)

lemma *nsubn-nsubn*:

assumes $n1 \leq n2$

$n0 \leq n4$

$n2 \leq n4 - n0$

$n4 \leq n3$

$n3 \leq nlength\ nell$

shows $(nsubn\ (nsubn\ nell\ n0\ n3)\ n1\ n2) = (nsubn\ (nsubn\ nell\ n0\ n4)\ n1\ n2)$

proof –

have 1: $nlength(nsubn\ nell\ n0\ n3) = n3 - n0$

using *assms* **by** (*metis enat-minus-mono1 idiff-enat-enat min.orderE nsubn-nlength*)

have 2: $nlength\ (nsubn\ (nsubn\ nell\ n0\ n3)\ n1\ n2) = n2 - n1$

using *assms*

by (*metis Nat.le-diff-conv2 enat-minus-mono1 enat-ord-simps(1) idiff-enat-enat le-trans min.orderE nsubn-nlength*)

have 3: $nlength(nsubn\ nell\ n0\ n4) = n4 - n0$

using *assms nsubn-nlength[of nell n0 n4]*

unfolding *min-def*

by (*metis enat-minus-mono1 idiff-enat-enat min.coboundedI1 min.left-commute min-absorb1 min-enat-simps(1)*)

have 4: $nlength\ (nsubn\ (nsubn\ nell\ n0\ n4)\ n1\ n2) = n2 - n1$

using *assms*

by (*metis 3 enat-minus-mono1 enat-ord-simps(1) idiff-enat-enat min.orderE nsubn-nlength*)

have 5: $\bigwedge i. i \leq (n3 - n0) \longrightarrow (nnth\ (nsubn\ nell\ n0\ n3)\ i) = (nnth\ nell\ (n0+i))$

using *assms* **by** (*simp add: nsubn-def1 ntaken-nnth*)
have 6: $\bigwedge i. i \leq (n2 - n1) \longrightarrow$
 $(nnth (nsubn (nsubn nell\ n0\ n3)\ n1\ n2)\ i) = (nnth (nsubn nell\ n0\ n3)\ (n1 + i))$
using *assms* **by** (*simp add: Nat.le-diff-conv2 nsubn-nnth*)
have 7: $\bigwedge i. i \leq (n2 - n1) \longrightarrow$
 $(nnth (nsubn (nsubn nell\ n0\ n3)\ n1\ n2)\ i) = (nnth nell\ (n0 + (n1 + i)))$
using 5 6 *assms* **by** *auto*
have 8: $n0 \leq n4 \wedge n4 \leq nlength\ nell$
using *assms* **by** (*simp add: order-subst2*)
have 9: $\bigwedge i. i \leq (n4 - n0) \longrightarrow (nnth (nsubn nell\ n0\ n4)\ i) = (nnth nell\ (n0 + i))$
using 8 **by** (*simp add: nsubn-def1 ntaken-nnth*)
have 10: $\bigwedge i. i \leq (n2 - n1) \longrightarrow$
 $(nnth (nsubn (nsubn nell\ n0\ n4)\ n1\ n2)\ i) = (nnth (nsubn nell\ n0\ n4)\ (n1 + i))$
by (*simp add: nsubn-def1 ntaken-nnth*)
have 11: $\bigwedge i. i \leq (n2 - n1) \longrightarrow$
 $(nnth (nsubn (nsubn nell\ n0\ n4)\ n1\ n2)\ i) = (nnth nell\ (n0 + (n1 + i)))$
by (*metis 10 9 Nat.le-diff-conv2 add.commute assms(1) assms(3) le-trans*)
have 12: $\bigwedge i. i \leq (n2 - n1) \longrightarrow$
 $(nnth (nsubn (nsubn nell\ n0\ n3)\ n1\ n2)\ i) = (nnth (nsubn (nsubn nell\ n0\ n4)\ n1\ n2)\ i)$
by (*simp add: 11 7*)
from 12 2 4 **show** *?thesis*
using *nellist-eq-nnth-eq*[*of (nsubn (nsubn nell\ n0\ n3)\ n1\ n2)\ (nsubn (nsubn nell\ n0\ n4)\ n1\ n2)*]
enat-ord-simps(1) **by** *presburger*
qed

lemma *nsubn-nsubn-1*:

assumes $n1 \leq n2$

$n0 \leq n3$

$n2 \leq n3 - n0$

$n3 \leq nlength\ nell$

shows $(nsubn (nsubn nell\ n0\ n3)\ n1\ n2) = (nsubn nell\ (n0 + n1)\ (n0 + n2))$

proof –

have 0: $nlength(nsubn (nsubn nell\ n0\ n3)\ n1\ n2) = n2 - n1$

using *assms*

by (*metis enat-minus-mono1 idiff-enat-enat min.orderE nsubn-nlength of-nat-eq-enat of-nat-mono*)

have 1: $nlength(nsubn nell\ (n1 + n0)\ (n2 + n0)) = (n2 + n0) - (n1 + n0)$

using *assms nsubn-nlength*[*of nell\ (n1 + n0)\ (n2 + n0)*]

unfolding *min-def*

by (*metis Nat.le-diff-conv2 dual-order.trans enat-minus-mono1 enat-ord-simps(1) idiff-enat-enat*)

have 10: $(n2 + n0) - (n1 + n0) = n2 - n1$

using *diff-cancel2* **by** *blast*

have 2: $\bigwedge i. i \leq (n2 - n1) \longrightarrow$

$(nnth (nsubn (nsubn nell\ n0\ n3)\ n1\ n2)\ i) = (nnth nell\ (n0 + (n1 + i)))$

using *assms* **by** (*simp add: nsubn-nnth*)

have 3: $\bigwedge i. i \leq (n2 - n1) \longrightarrow$

$(nnth (nsubn nell\ (n0 + n1)\ (n0 + n2))\ i) = (nnth nell\ (n0 + (n1 + i)))$

using *assms* **by** (*metis 10 add.assoc add.commute min.orderE nsubn-nnth*)

have 4: $\bigwedge i. i \leq (n2 - n1) \longrightarrow$

$(nnth (nsubn (nsubn nell\ n0\ n3)\ n1\ n2)\ i) = (nnth (nsubn nell\ (n0 + n1)\ (n0 + n2))\ i)$

by (*simp add: 2 3*)

show *?thesis* **using** *nellist-eq-nnth-eq*[*of* (*nsubn* (*nsubn nell n0 n3*) *n1 n2*)
(*nsubn nell (n0+n1) (n0+n2)*)]
0 10 4 1
by (*metis add commute enat-ord-simps*(1))
qed

lemma *nsubn-eq*:

assumes $n \leq m$
 $m \leq \text{nlength } \text{nellx}$
 $m \leq \text{nlength } \text{nelly}$
 $\forall j. n \leq j \wedge j \leq m \longrightarrow \text{nnth } \text{nellx } j = \text{nnth } \text{nelly } j$
shows $\text{nsubn } \text{nellx } n m = \text{nsubn } \text{nelly } n m$
proof –
have 1: $\text{nlength } (\text{nsubn } \text{nellx } n m) = \text{nlength } (\text{nsubn } \text{nelly } n m)$
using *assms* **by** (*metis enat-minus-mono1 min-def nsubn-nlength*)
have 2: $\bigwedge j. j \leq \text{nlength } (\text{nsubn } \text{nellx } n m) \longrightarrow \text{nnth } (\text{nsubn } \text{nellx } n m) j = \text{nnth } (\text{nsubn } \text{nelly } n m) j$
using *assms* **by** (*simp add: Nat.le-diff-conv2 nsubn-nlength nsubn-nnth*)
show *?thesis*
by (*simp add: 1 2 nellist-eq-nnth-eq*)
qed

1.3.5 nfuse

lemma *nfuse-def1*:

$\text{nfuse } \text{nellx } \text{nelly} = (\text{if is-NNil } \text{nelly} \text{ then } \text{nellx} \text{ else } \text{nappend } \text{nellx } (\text{ntl } \text{nelly}))$
apply *transfer*
unfolding *lfuse-def* **by** *simp force*

lemma *nfuse-NCons-a*:

$\text{nfuse } (\text{NCons } x \text{ nellx}) \text{ nelly} = (\text{NCons } x (\text{nfuse } \text{nellx } \text{nelly}))$
by (*simp add: nfuse-def1*)

lemma *nfuse-NCons-b*:

$\text{nfuse } \text{nellx } (\text{NCons } y \text{ nelly}) = \text{nappend } \text{nellx } \text{nelly}$
by (*simp add: nfuse-def1*)

lemma *nfuse-simps* [*simp*]:

shows *nhd-nfuse*: $\text{nhd}(\text{nfuse } \text{nellx } \text{nelly}) =$
 $(\text{if is-NNil } \text{nellx} \text{ then}$
 $(\text{if is-NNil } \text{nelly} \text{ then } \text{nhd } \text{nellx} \text{ else } \text{nlast } \text{nellx})$
 $\text{else } \text{nhd } \text{nellx})$
and *ntl-nfuse*: $\text{ntl}(\text{nfuse } \text{nellx } \text{nelly}) =$
 $(\text{if is-NNil } \text{nellx} \text{ then}$
 $(\text{if is-NNil } \text{nelly} \text{ then } \text{ntl } \text{nellx} \text{ else } \text{ntl } \text{nelly})$
 $\text{else } (\text{if is-NNil } \text{nellx} \text{ then } \text{ntl } \text{nellx}$
 $\text{else } \text{nappend } (\text{ntl } \text{nellx}) (\text{ntl } \text{nelly})))$
by (*simp-all add: nfuse-def1*)

lemma *nfuse-nbutlast*:

assumes $nlast\ nellx = nfirst\ nelly$
 $\neg is_NNil\ nellx$
 $\neg is_NNil\ nelly$
shows $nfuse\ nellx\ nelly = nappend\ (nbutlast\ nellx)\ nelly$
using *assms*
by (*metis nappend-NCons nappend-NNil nappend-assoc nappend-nbutlast-nlast-id nappend-snocn*
 $nellist.collapse(2)\ nellist.sel(3)\ nfuse-def1\ nhd-snocn$)

lemma *nfuse-nlength*:
shows $nlength\ (nfuse\ nellx\ nelly) = (nlength\ nellx) + (nlength\ nelly)$
unfolding *nfuse-def1*
by(*cases nelly*) (*simp, simp add: eSuc-plus-1*)

lemma *nfuse-nnth*:
assumes $i \leq nlength\ (nfuse\ nellx\ nelly)$
 $nlast\ nellx = nfirst\ nelly$
shows $(i \leq nlength\ nellx \longrightarrow nnth\ (nfuse\ nellx\ nelly)\ i = nnth\ nellx\ i)$
 \wedge
 $(nlength\ nellx < i \wedge i \leq nlength\ (nfuse\ nellx\ nelly) \longrightarrow$
 $nnth\ (nfuse\ nellx\ nelly)\ i = nnth\ nelly\ (i - (the-enat\ (nlength\ nellx))))$
unfolding *nfuse-def1*
by (*cases nelly*)
(auto simp add: nnth-nappend,
 $metis\ Suc-diff-Suc\ enat-iless\ enat-ord-simps(2)\ ndropn-Suc-NCons\ ndropn-nfirst\ the-enat.simps)$

lemma *nfuse-nnth-a*:
assumes $j \leq nlength\ nelly$
 $nlast\ nellx = nfirst\ nelly$
 $nfinite\ nellx$
shows $nnth\ (nfuse\ nellx\ nelly)\ ((the-enat(nlength\ nellx)) + j) = (nnth\ nelly\ j)$
using *assms* **unfolding** *nfuse-def1*
by (*cases is-NNil nelly*)
(simp-all,
 $metis\ assms(2)\ assms(3)\ is-NNil-def\ is-NNil-imp-nfinite\ ndropn-nlast\ ndropn-nfirst$
 $ndropn-nnth\ nnth-NNil,$
 $metis\ enat-le-plus-same(2)\ gen-nlength-def\ nappend-NNil\ ndropn-nlast\ ndropn-nappend2$
 $ndropn-nnth\ nellist.case-eq-if\ nellist.collapse(2)\ nfinite-nlength-enat\ nfirst-def$
 $nlength-code\ the-enat.simps)$

lemma *nfuse-nappend*:
assumes $nlast\ nellx = nfirst\ nelly$
shows $nfuse\ nellx\ nelly =$
 $(if\ nfinite\ nellx\ then$
 $(if\ is_NNil\ nellx\ then\ nelly\ else\ nappend\ (ntaken\ (the-enat(epred(nlength\ nellx))))\ nellx)\ nelly)$
 $else\ nellx)$
proof (*cases nelly*)
case (*NNil x1*)
then show *?thesis*
proof (*cases nfinite nellx*)
case *True*

```

then show ?thesis
proof (cases nellx)
case (NNil x1)
then show ?thesis using assms
by (simp add: nfuse-def1)
(metis nappend-snocn nellist.collapse(1) nellist.collapse(2) nhd-nappend nhd-snocn)
next
case (NCons x21 x22)
then show ?thesis unfolding nfuse-def1 using assms NNil NCons True
by (auto simp add: nfirst-def)
(metis True add-diff-cancel-left' assms eSuc-enat ndropn-nfirst ndropn-nlast
nellist-split-2-last nfinite-nlength-enat nlast-NCons nlength-NCons plus-1-eq-Suc
the-enat.simps zero-less-Suc)
qed
next
case False
then show ?thesis
by (simp add: NNil nfuse-def1)
qed
next
case (NCons x21 x22)
then show ?thesis
proof (cases nfinite nellx)
case True
then show ?thesis
unfolding nfuse-def1
proof auto
show is-NNil nelly  $\implies$  is-NNil nellx  $\implies$  nellx = nelly
by (simp add: NCons)
show nfinite nellx  $\implies$  is-NNil nelly  $\implies$   $\neg$  is-NNil nellx  $\implies$ 
nellx = nappend (ntaken (the-enat (epred (nlength nellx)))) nellx nelly
using assms NCons True by simp
show  $\neg$  is-NNil nelly  $\implies$  is-NNil nellx  $\implies$  nappend nellx (ntl nelly) = nelly
using assms NCons True
by (metis nappend-NNil nellist.collapse(1) nellist.sel(5) nnth-0 ntaken-0 ntaken-nlast)
show nfinite nellx  $\implies$   $\neg$  is-NNil nelly  $\implies$   $\neg$  is-NNil nellx  $\implies$ 
nappend nellx (ntl nelly) = nappend (ntaken (the-enat (epred (nlength nellx)))) nellx nelly
using assms NCons True
by (cases nellx)
( auto,
metis True add-diff-cancel-left' eSuc-enat nappend-NCons nappend-NNil nappend-assoc
ndropn-eq-NNil ndropn-nlast nellist.sel(5) nellist-split-2-last nfinite-nlength-enat
nlast-NNil nlast-ntl nlength-NCons nnth-0 ntaken-nlast ntl-ntaken-0 plus-1-eq-Suc
the-enat.simps zero-less-Suc)
qed
next
case False
then show ?thesis
by (simp add: nappend-inf nfuse-def1)
qed

```

qed

lemma *nfuse-leftneutral* :
 nfuse (*NNil* (*nfir*st *nell*)) *nell* = *nell*
by (*simp add: nfuse-nappend*)

lemma *nfuse-rightneutral* :
 nfuse *nell* (*NNil* (*nlast* *nell*)) = *nell*
unfolding *nfuse-def*
by *simp*

lemma *nfir*st-*nfuse* :
 assumes *nlast nellx = nfir*st *nelly*
 shows *nfir*st (*nfuse* *nellx nelly*) = *nfir*st *nellx*
using *assms unfolding nfuse-def1* **by** *transfer auto*

lemma *nlast-nfuse* :
 assumes *nlast nellx = nfir*st *nelly*
 nfinite nellx
 shows *nlast* (*nfuse* *nellx nelly*) = *nlast nelly*
using *assms unfolding nfuse-def1* **by** *transfer (auto, metis lhd-LCons-rtl llast-LCons llast-lappend)*

lemma *nfuseassoc* :
 shows (*nfuse* *nellx (nfuse* *nelly nellz*)) = (*nfuse* (*nfuse* *nellx nelly*) *nellz*)
unfolding *nfuse-def1*
by *transfer (auto simp add: lappend-assoc, simp add: lappend-rtl)*

lemma *ntaken-nfuse* :
 assumes *nlast nellx = nfir*st *nelly*
 nfinite nellx
 shows (*ntaken* (*the-enat* (*nlength* *nellx*)) (*nfuse* *nellx nelly*)) = *nellx*
proof (*cases is-NNil nelly*)
case *True*
then show *?thesis* **using** *assms* **by** (*metis ndropn-eq-NNil ndropn-nlast nfuse-def1 ntaken-all*)
next
case *False*
then show *?thesis* **using** *assms*
by (*metis add.left-neutral enat-le-plus-same(2) nfinite-nlength-enat nfuse-def1 ntaken-all*
 ntaken-nappend1 the-enat.simps)
qed

lemma *ndropn-nfuse* :
 assumes *nlast nellx = nfir*st *nelly*
 nfinite nellx
 shows (*ndropn* (*the-enat* (*nlength* *nellx*)) (*nfuse* *nellx nelly*)) = *nelly*
proof (*cases is-NNil nelly*)
case *True*
then show *?thesis* **using** *assms* **by** (*metis ndropn-nlast nfuse-def1 nfuse-leftneutral*)
next
case *False*

then show *?thesis using assms*
by (*metis enat-le-plus-same(2) gen-nlength-def ndropn-nappend2 ndropn-nlast nfinite-nlength-enat*
nfuse-def1 nfuse-leftneutral nlength-code the-enat.simps)
qed

lemma *nfuse-ntaken-ndropn-nlength :*
assumes $n \leq nlength\ nelly$
shows $nlength\ (nfuse\ (ntaken\ n\ nelly)\ (ndropn\ n\ nelly)) = nlength\ nelly$
using *assms*
by (*metis dual-order.order-iff-strict enat-add-sub-same infinity-ileE less-eqE min.absorb1*
ndropn-nlength nfuse-nlength ntaken-nlength)

lemma *nfuse-ntaken-ndropn-nnth :*
assumes $n \leq nlength\ nelly$
 $i \leq nlength\ nelly$
shows $nnth\ (nfuse\ (ntaken\ n\ nelly)\ (ndropn\ n\ nelly))\ i = nnth\ nelly\ i$
using *assms*
nfuse-nnth[of i (ntaken n nelly) (ndropn n nelly)]
nfuse-ntaken-ndropn-nlength[of n nelly]
by (*metis dual-order.strict-iff-order enat-ord-simps(1) le-add-diff-inverse min.orderE ndropn-nfirst*
ndropn-nnth not-less ntaken-nlast ntaken-nlength ntaken-nnth the-enat.simps)

lemma *nfuse-ntaken-ndropn:*
assumes $n \leq nlength\ nelly$
shows $nfuse\ (ntaken\ n\ nelly)\ (ndropn\ n\ nelly) = nelly$
using *assms*
by (*simp add: nfuse-ntaken-ndropn-nlength nfuse-ntaken-ndropn-nnth nelly-eq-nnth-eq*)

lemma *nfuse-nnth-var:*
assumes $enat\ i \leq nlength\ (nfuse\ nellyx\ nelly)$
 $nlast\ nellyx = nfirst\ nelly$
shows $(enat\ i \leq nlength\ nellyx \longrightarrow nnth\ (nfuse\ nellyx\ nelly)\ i = nnth\ nellyx\ i) \wedge$
 $(nlength\ nellyx \leq enat\ i \wedge enat\ i \leq nlength\ (nfuse\ nellyx\ nelly) \longrightarrow$
 $nnth\ (nfuse\ nellyx\ nelly)\ i = nnth\ nelly\ (i - the-enat\ (nlength\ nellyx)))$
using *nfuse-nnth assms*
by (*metis cancel-comm-monoid-add-class.diff-cancel dual-order.strict-iff-order*
ndropn-nfuse nlength-eq-enat-nfiniteD nnth-zero-ndropn the-enat.simps)

lemma *nset-nfuse:*
 $nset\ (nfuse\ nellyx\ nelly) =$
 $(if\ nfinite\ nellyx\ then$
 $(if\ is-NNil\ nelly\ then\ nset\ nellyx\ else\ nset\ nellyx \cup nset\ (ntl\ nelly))$
 $else\ nset\ nellyx)$
by (*simp add: nfuse-def1 nset-nappend*)

lemma *nsubn-nfuse:*
assumes $(enat\ k) \leq n$
 $(enat\ n) \leq m$
 $(enat\ m) \leq nlength\ nelly$
shows $nfuse\ (nsubn\ nelly\ k\ n)\ (nsubn\ nelly\ n\ m) = (nsubn\ nelly\ k\ m)$

```

using assms
proof -
  have 1:  $nlast(nsubn\ nell\ k\ n) = (nnth\ nell\ n)$ 
    by (metis assms(1) enat-ord-simps(1) le-add-diff-inverse2 nsubn-def1 ntaken-ndropn-nlast)
  have 2:  $nfirst(nsubn\ nell\ n\ m) = (nnth\ nell\ n)$ 
    by (simp add: ndropn-nfirst nsubn-def1 ntaken-ndropn-nfirst-a)
  have 3:  $nlength\ (nsubn\ nell\ k\ n) = n - k$ 
    using assms nsubn-nlength[of nell k n]
    by (metis dual-order.trans enat-minus-mono1 idiff-enat-enat min-absorb1)
  have 4:  $nlength\ (nsubn\ nell\ n\ m) = m - n$ 
    by (metis assms(3) enat-minus-mono1 idiff-enat-enat min-def nsubn-nlength)
  have 5:  $nlength\ (nsubn\ nell\ k\ m) = m - k$ 
    by (metis assms(3) enat-minus-mono1 idiff-enat-enat min-absorb1 nsubn-nlength)
  have 6:  $nlength(nfuse\ (nsubn\ nell\ k\ n)\ (nsubn\ nell\ n\ m)) = nlength(nsubn\ nell\ k\ m)$ 
    unfolding nsubn-def
    by (metis 3 4 5 Nat.add-diff-assoc2 assms(1) assms(2) enat-ord-simps(1) nfuse-nlength
        nsubn-def ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-enat-simps(1))
  have 7:  $(\forall\ i.\ i \leq nlength(nsubn\ nell\ k\ m) \longrightarrow$ 
     $(nnth\ (nfuse\ (nsubn\ nell\ k\ n)\ (nsubn\ nell\ n\ m))\ i) = (nnth\ nell\ (k+i)))$ 
  proof
    fix i
    show  $i \leq nlength(nsubn\ nell\ k\ m) \longrightarrow$ 
       $(nnth\ (nfuse\ (nsubn\ nell\ k\ n)\ (nsubn\ nell\ n\ m))\ i) = (nnth\ nell\ (k+i))$ 
  proof -
    have 41:  $nlength\ (nsubn\ nell\ k\ m) = (m - k)$ 
      using 5 by blast
    have 42:  $i \leq nlength\ (nsubn\ nell\ k\ m) \longrightarrow nnth\ (nfuse\ (nsubn\ nell\ k\ n)\ (nsubn\ nell\ n\ m))\ i =$ 
       $(if\ i \leq n - k\ then\ (nnth\ (nsubn\ nell\ k\ n)\ i)$ 
         $else\ (nnth\ (nsubn\ nell\ n\ m)\ (i - (n - k))))$ 
      by (simp add: 1 2 3 6 nfuse-nnth)
    have 43:  $i \leq (m - k) \longrightarrow nnth\ (nfuse\ (nsubn\ nell\ k\ n)\ (nsubn\ nell\ n\ m))\ i =$ 
       $(if\ i \leq (n - k)\ then\ (nnth\ nell\ (k+i))\ else\ (nnth\ nell\ (n + (i - (n - k)))))$ 
      using assms 42 5 unfolding nsubn-def1
      by (auto simp add: ntaken-nnth)
      (metis enat-ord-simps(1) min.bounded-iff,
        metis enat-ord-simps(1) min.bounded-iff)
    have 44:  $i \leq (m - k) \longrightarrow nnth\ (nfuse\ (nsubn\ nell\ k\ n)\ (nsubn\ nell\ n\ m))\ i =$ 
       $(nnth\ nell\ (k+i))$ 
      by (metis 43 Nat.diff-diff-right Nat.le-diff-conv2 add commute assms(1) enat-ord-simps(1)
        le-add-diff-inverse nat-le-linear)
    show ?thesis
      by (simp add: 41 44)
  qed
qed
have 8:  $(\forall\ i.\ i \leq nlength(nsubn\ nell\ k\ m) \longrightarrow (nnth\ (nsubn\ nell\ k\ m)\ i) = (nnth\ nell\ (k+i)))$ 
  using assms by (simp add: nsubn-def1 ntaken-nnth)
show ?thesis
  by (simp add: 6 7 8 nellist-eq-nnth-eq)
qed

```

lemma *nmap-nfuse*:

$nmap\ f\ (nfuse\ nellx\ nelly) = nfuse\ (nmap\ f\ nellx)\ (nmap\ f\ nelly)$
by (*simp add: nfuse-def1 nmap-nappend-distrib*)

lemma *nzip-nfuse*:

assumes $nlength\ \sigma 1 = nlength\ l1$
 $nlength\ \sigma 2 = nlength\ l2$
 $nfirst\ \sigma 2 = nlast\ \sigma 1$
 $nfinite\ \sigma 1$
 $nfirst\ l2 = nlast\ l1$
shows $(nzip\ (nfuse\ \sigma 1\ \sigma 2)\ (nfuse\ l1\ l2)) = nfuse\ (nzip\ \sigma 1\ l1)\ (nzip\ \sigma 2\ l2)$
using *assms*
apply *transfer*
by (*simp add: epred-inject lfuse-def lzip-lappend*)

1.3.6 nridx and nidx

lemma *nridx-nidx*:

$nridx\ (<) \ nell = nidx\ nell$
apply *transfer*
by (*simp add: ridx-lidx*)

lemma *nridx-expand*:

$nridx\ R\ nell \longleftrightarrow (\forall i. (Suc\ i) \leq nlength\ nell \longrightarrow R\ (nnth\ nell\ i)\ (nnth\ nell\ (Suc\ i)))$
by *transfer*
(auto simp add: min-def Suc-ile-eq,
metis co.enat.exhaust-sel eSuc-enat ileI1 iless-Suc-eq llength-eq-0 ridx-def,
metis Extended-Nat.eSuc-mono co.enat.exhaust-sel eSuc-enat llength-eq-0 order-le-less ridx-def)

lemma *nidx-expand*:

$nidx\ nell \longleftrightarrow (\forall i. (Suc\ i) \leq nlength\ nell \longrightarrow nnth\ nell\ i < nnth\ nell\ (Suc\ i))$
using *nridx-nidx nridx-expand* **by** *blast*

lemma *nridx-NCons*:

$nridx\ R\ (NCons\ x\ nell) \longleftrightarrow$
 $(\forall n. (Suc\ n) \leq eSuc\ (nlength\ nell) \longrightarrow R\ (nnth\ (NCons\ x\ nell)\ n)\ (nnth\ (NCons\ x\ nell)\ (Suc\ n)))$
using *nridx-expand[of R (NCons x nell)]*
by *auto*

lemma *nidx-NCons*:

$nidx\ (NCons\ x\ nell) \longleftrightarrow$
 $(\forall n. (Suc\ n) \leq eSuc\ (nlength\ nell) \longrightarrow (nnth\ (NCons\ x\ nell)\ n) < (nnth\ (NCons\ x\ nell)\ (Suc\ n)))$
using *nridx-NCons[of (<) x nell]* *nridx-nidx* **by** *blast*

lemma *nridx-LCons-conv*:

$(\forall n. (Suc\ n) \leq eSuc\ (nlength\ nell) \longrightarrow R\ (nnth\ (NCons\ x\ nell)\ n)\ (nnth\ (NCons\ x\ nell)\ (Suc\ n))) \longleftrightarrow$
 $R\ x\ (nfirst\ nell) \wedge$
 $(\forall n. 0 \leq n \wedge (Suc\ n) \leq (nlength\ nell) \longrightarrow R\ (nnth\ nell\ n)\ (nnth\ nell\ (Suc\ n)))$
proof –

have 1: $(\forall n. (Suc\ n) \leq eSuc\ (nlength\ nell) \longrightarrow R\ (nnth\ (NCons\ x\ nell)\ n)\ (nnth\ (NCons\ x\ nell)\ (Suc\ n))) \longleftrightarrow$
 $(\forall n. 0 \leq n \wedge (Suc\ n) \leq eSuc\ (nlength\ nell) \longrightarrow R\ (nnth\ (NCons\ x\ nell)\ n)\ (nnth\ (NCons\ x\ nell)\ (Suc\ n)))$
by *blast*
have 2: $(\forall n. 0 \leq n \wedge (Suc\ n) \leq eSuc\ (nlength\ nell) \longrightarrow R\ (nnth\ (NCons\ x\ nell)\ n)\ (nnth\ (NCons\ x\ nell)\ (Suc\ n))) \longleftrightarrow$
 $R\ x\ (nfirst\ nell) \wedge$
 $(\forall n. 1 \leq n \wedge (Suc\ n) \leq eSuc\ (nlength\ nell) \longrightarrow R\ (nnth\ (NCons\ x\ nell)\ n)\ (nnth\ (NCons\ x\ nell)\ (Suc\ n)))$
by *(metis 1 One-nat-def add-diff-cancel-left' eSuc-ile-mono enat-le-plus-same(1) gen-nlength-def le-SucE le-add1*
ndropn-Suc-NCons ndropn-nfirst ndropn-nfuse nfuse-leftneutral nlast-NNil nlength-NNil nlength-code
nlength-eq-enat-nfiniteD nnth-0 one-eSuc one-enat-def plus-1-eq-Suc the-enat-0 zero-enat-def)
have 3: $R\ x\ (nfirst\ nell) \wedge$
 $(\forall n. 1 \leq n \wedge (Suc\ n) \leq eSuc\ (nlength\ nell) \longrightarrow R\ (nnth\ (NCons\ x\ nell)\ n)\ (nnth\ (NCons\ x\ nell)\ (Suc\ n))) \longleftrightarrow$
 $R\ x\ (nfirst\ nell) \wedge$
 $(\forall n. 1 \leq n \wedge (n) \leq (nlength\ nell) \longrightarrow R\ (nnth\ (NCons\ x\ nell)\ n)\ (nnth\ (NCons\ x\ nell)\ (Suc\ n)))$
by *(metis eSuc-enat eSuc-ile-mono)*
have 4: $R\ x\ (nfirst\ nell) \wedge$
 $(\forall n. 1 \leq n \wedge (n) \leq (nlength\ nell) \longrightarrow R\ (nnth\ (NCons\ x\ nell)\ n)\ (nnth\ (NCons\ x\ nell)\ (Suc\ n)))$
 \longleftrightarrow
 $R\ x\ (nfirst\ nell) \wedge$
 $(\forall n. 0 \leq (n-1) \wedge (Suc\ (n-1)) \leq (nlength\ nell) \longrightarrow R\ (nnth\ nell\ (n-1))\ (nnth\ nell\ (Suc\ (n-1))))$
by *(metis add.commute add.right-neutral diff-add le-add1 nnth-Suc-NCons plus-1-eq-Suc)*
have 5: $R\ x\ (nfirst\ nell) \wedge$
 $(\forall n. 0 \leq (n-1) \wedge (Suc\ (n-1)) \leq (nlength\ nell) \longrightarrow R\ (nnth\ nell\ (n-1))\ (nnth\ nell\ (Suc\ (n-1))))$
 \longleftrightarrow
 $R\ x\ (nfirst\ nell) \wedge$
 $(\forall n. 0 \leq n \wedge (Suc\ n) \leq (nlength\ nell) \longrightarrow R\ (nnth\ nell\ n)\ (nnth\ nell\ (Suc\ n)))$
by *(metis diff-Suc-1)*
show *?thesis*
using *1 2 3 4 5 by presburger*
qed

lemma *nidx-LCons-conv:*

$(\forall n. (Suc\ n) \leq eSuc\ (nlength\ nell) \longrightarrow (nnth\ (NCons\ x\ nell)\ n) < (nnth\ (NCons\ x\ nell)\ (Suc\ n))) \longleftrightarrow$
 $x < (nfirst\ nell) \wedge$
 $(\forall n. 0 \leq n \wedge (Suc\ n) \leq (nlength\ nell) \longrightarrow (nnth\ nell\ n) < (nnth\ nell\ (Suc\ n)))$
by *(metis nridx-LCons-conv)*

lemma *nridx-LCons-1 [simp]:*

$nridx\ R\ (NCons\ x\ nell) \longleftrightarrow (R\ x\ (nfirst\ nell) \wedge nridx\ R\ nell)$
by *(metis nridx-LCons-conv nridx-NCons nridx-expand zero-le)*

lemma *nidx-LCons-1 [simp]:*

$nidx\ (NCons\ x\ nell) \longleftrightarrow (x < (nfirst\ nell) \wedge nidx\ nell)$
by *(metis nridx-LCons-1 nridx-nidx)*

lemma *nridx-less*:
assumes *nridx R nell*
 $Suc(n+k) \leq nlength\ nell$
transp R
shows $R\ (nnth\ nell\ n)\ (nnth\ nell\ (Suc(n+k)))$
using *assms*
proof (*induct k*)
case 0
then show ?*case*
by (*simp add: nridx-expand*)
next
case (*Suc k*)
then show ?*case*
by (*metis add-Suc-right dual-order.trans eSuc-enat ile-eSuc nridx-expand transpE*)
qed

lemma *nidx-less*:
assumes *nidx nell*
 $Suc(n+k) \leq nlength\ nell$
shows $nnth\ nell\ n < nnth\ nell\ (Suc(n+k))$
using *assms*
by (*simp add: nridx-less nridx-nidx*)

lemma *nridx-less-eq*:
assumes *nridx R nell*
 $k \leq j$
 $j \leq nlength\ nell$
transp R
reflp R
shows $R\ (nnth\ nell\ k)\ (nnth\ nell\ j)$
proof (*cases k=j*)
case *True*
then show ?*thesis* **using** *assms* **by** (*meson reflpE*)
next
case *False*
then show ?*thesis* **using** *assms*
by (*metis (full-types) Suc-diff-Suc add.left-commute le-add-diff-inverse nridx-less order-neq-le-trans plus-1-eq-Suc*)
qed

lemma *nidx-less-eq*:
assumes *nidx nell*
 $k \leq j$
 $j \leq nlength\ nell$
shows $(nnth\ nell\ k) \leq (nnth\ nell\ j)$
using *assms*
by (*metis Orderings.order-eq-iff antisym-conv2 less-iff-Suc-add nidx-less order.strict-implies-order*)

lemma *nridx-less-last*:
assumes *nridx R nell*

$Suc\ i < k$
 $nlength\ nell = (enat\ k)$
 $transp\ R$
shows $R\ (nnth\ nell\ i)\ (nnth\ nell\ (k-1))$
using *assms less-imp-Suc-add nridx-less* **by** *fastforce*

lemma *nidx-less-last*:
assumes $nidx\ nell$
 $Suc\ i < k$
 $nlength\ nell = (enat\ k)$
shows $nnth\ nell\ i < nnth\ nell\ (k-1)$
using *assms less-imp-Suc-add nidx-less* **by** *fastforce*

lemma *nidx-less-last-1*:
assumes $nidx\ nell$
 $i < nlength\ nell$
 $nlength\ nell = (enat\ k)$
shows $nnth\ nell\ i < nnth\ nell\ (k)$
using *assms*
by *(metis enat-ord-simps(2) less-imp-Suc-add linorder-le-cases nridx-less nridx-nidx transp-on-less)*

lemma *nridx-gr-first*:
assumes $nridx\ R\ nell$
 $0 < i$
 $i \leq nlength\ nell$
 $transp\ R$
shows $R\ (nnth\ nell\ 0)\ (nnth\ nell\ i)$
using *assms nridx-less[of R nell 0 i-1]* **by** *simp*

lemma *nidx-gr-first*:
assumes $nidx\ nell$
 $0 < i$
 $i \leq nlength\ nell$
shows $(nnth\ nell\ 0) < nnth\ nell\ i$
using *assms nidx-less[of nell 0 i-1]*
by *simp*

lemma *nridx-ntake-a*:
assumes $nridx\ R\ nell$
 $n \leq nlength\ nell$
shows $nridx\ R\ (ntake\ n\ nell)$
using *assms*
by *transfer*
(metis co.enat.exhaust-sel eSuc-ile-mono llength-eq-0 ridx-ltake-a)

lemma *nidx-ntake-a*:
assumes $nidx\ nell$
 $n \leq nlength\ nell$

shows $nidx$ ($ntake\ n\ nell$)
using $assms$
using $nridx-ntake-a\ nridx-nidx$ **by** $blast$

lemma $nridx-nappend-nfinite$:
assumes $nfinite\ nell1$
shows $nridx\ R\ (nappend\ nell1\ nell2) \longleftrightarrow$
 $nridx\ R\ nell1 \wedge (R\ (nlast\ nell1)\ (nfirst\ nell2)) \wedge nridx\ R\ nell2$
using $assms$
by $transfer$
 $(simp\ add:\ ridx-lappend-lfinite)$

lemma $nidx-nappend-nfinite$:
assumes $nfinite\ nell1$
shows $nidx\ (nappend\ nell1\ nell2) \longleftrightarrow$
 $nidx\ nell1 \wedge ((nlast\ nell1) < (nfirst\ nell2)) \wedge nidx\ nell2$
using $assms$
by $(metis\ nridx-nappend-nfinite\ nridx-nidx)$

lemma $nidx-nfuse$:
assumes $nfinite\ nell1$
 $nidx\ nell1$
 $nnth\ nell1\ 0 = 0$
 $nidx\ nell2$
 $nnth\ nell2\ 0 = nlast\ nell1$
shows $nidx\ (nfuse\ nell1\ nell2)$
using $assms$
proof $(cases\ is-NNil\ nell2)$
case $True$
then show $?thesis$ **unfolding** $nfuse-def1$
by $(simp\ add:\ assms(2))$
next
case $False$
then show $?thesis$
proof –
have $1: nlast\ nell1 = nfirst\ nell2$
by $(metis\ assms(5)\ ndropn-0\ ndropn-nfirst)$
have $2: nidx\ (ntl\ nell2)$
by $(metis\ False\ assms(4)\ eSuc-enat\ ileI1\ ileSSuc-eq\ nellist.collapse(2)\ nidx-expand\ nlength-NCons\ nnth-ntl)$
have $3: nlast\ nell1 < nfirst(ntl\ nell2)$
by $(metis\ False\ assms(4)\ assms(5)\ eSuc-enat\ ileI1\ ileSSuc-eq\ ndropn-0\ ndropn-nfirst\ nellist.collapse(2)\ nhd-conv-nnth\ nidx-expand\ nlength-NCons\ nnth-ntl\ zero-enat-def\ zero-le)$
have $4: nidx\ (nappend\ nell1\ (ntl\ nell2))$
by $(simp\ add:\ 2\ 3\ assms(1)\ assms(2)\ nidx-nappend-nfinite)$
show $?thesis$ **using** $False$ **unfolding** $nfuse-def1$
using 4 **by** $auto$
qed
qed

lemma *nridx-ndropn*:
assumes *nridx R nell*
 $n \leq \text{nlength } nell$
shows *nridx R (ndropn n nell)*
using *assms*
by *transfer*
*(metis co.enat.exhaust-sel iless-Suc-eq ldrop-enat llength-eq-0 min.orderE nless-le
ridx-ldrop the-enat.simps)*

lemma *nidx-ndropn*:
assumes *nidx nell*
 $n \leq \text{nlength } nell$
shows *nidx (ndropn n nell)*
using *assms*
using *nridx-ndropn nridx-nidx* **by** *blast*

lemma *nridx-ntake-all*:
assumes $\bigwedge n. n \leq \text{nlength } nell \implies \text{nridx } R (\text{ntake } (enat \ n) \ nell)$
shows *nridx R nell*
using *assms*
by *(auto simp add: nridx-expand)*
(metis Suc-ile-eq linorder-le-cases ntake-nnth ntake-nnth order-less-imp-le)

lemma *nidx-ntake-all*:
assumes $\bigwedge n. n \leq \text{nlength } nell \implies \text{nidx } (\text{ntake } (enat \ n) \ nell)$
shows *nidx nell*
using *assms*
using *nridx-nidx nridx-ntake-all* **by** *blast*

lemma *nridx-ntake*:
assumes *nridx R (ntake n nell)*
 $n \leq \text{nlength } nell$
 $k \leq n$
shows *nridx R (ntake (enat k) nell)*
using *assms*
using *nridx-ntake-a* **by** *fastforce*

lemma *nidx-ntake*:
assumes *nidx (ntake n nell)*
 $n \leq \text{nlength } nell$
 $k \leq n$
shows *nidx (ntake (enat k) nell)*
using *assms*
using *nridx-nidx nridx-ntake* **by** *blast*

lemma *nidx-imp-ndistinct*:
assumes *nidx nell*
shows *ndistinct nell*
using *assms*

apply *transfer*
using *lidx-imp-ldistinct* **by** *auto*

lemma *ndistinct-Ex1*:
assumes *ndistinct nell*
 $x \in \text{nset } nell$
shows $\exists !i. i \leq \text{nlength } nell \wedge (\text{nnth } nell \ i) = x$
using *assms*
by *transfer*
(auto,
metis co.enat.exhaust-sel iless-Suc-eq ldistinct-Ex1 llength-eq-0 min.orderE the-enat.simps,
metis co.enat.exhaust-sel iless-Suc-eq ldistinct-conv-lnth llength-eq-0)

lemma *nidx-nset-eq*:
assumes *nidx nellx*
 $nidx \text{ nelly}$
 $\text{nset } nellx = \text{nset } nelly$
shows $nellx = nelly$
using *assms*
by *transfer*
(simp add: bi-unique-cr-nellist-help lidx-lset-eq nidx.rep-eq)

lemma *nridx-nfuse-nfirst-nlast*:
assumes *nridx R nell1*
 $(\text{nnth } nell1 \ 0) = (0::nat)$
 $nridx \ R \ nell2$
 $(\text{nnth } nell2 \ 0) = 0$
 $nfinite \ nell1$
 $nfinite \ nell$
 $nlast \ nell1 = cp$
shows $nlast \ nell1 = nfirst(\text{nmap } (\lambda x. x+cp) \ nell2)$
using *assms*
by *(metis add.commute add.right-neutral ndropn-0 ndropn-nfirst nnth-nmap zero-enat-def zero-le)*

lemma *nidx-nfuse-nfirst-nlast*:
assumes *nidx nell1*
 $(\text{nnth } nell1 \ 0) = (0::nat)$
 $nidx \ nell2$
 $(\text{nnth } nell2 \ 0) = 0$
 $nfinite \ nell1$
 $nfinite \ nell$
 $nlast \ nell1 = cp$
shows $nlast \ nell1 = nfirst(\text{nmap } (\lambda x. x+cp) \ nell2)$
using *assms*
by *(metis add.commute add.right-neutral ndropn-0 ndropn-nfirst nnth-nmap zero-enat-def zero-le)*

lemma *nridx-nfuse-nnth-cp*:
assumes *nridx R nell1*
 $(\text{nnth } nell1 \ 0) = 0$

$nridx\ R\ nell2$
 $(nnth\ nell2\ 0) = 0$
 $nfinite\ nell1$
 $nfinite\ nell2$
 $nfinite\ nell$
 $nlast\ nell1 = cp$
 $nlast\ nell2 = the-enat((nlength\ nell)) - cp$
 $i \leq (nlength\ nell2)$
 $cp \leq nlength\ nell$
shows $nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x+cp)\ nell2))\ (the-enat((nlength\ nell1)) + i) = cp + (nnth\ nell2\ i)$
proof –
have 1: $nlast\ nell1 = nfirst\ (nmap\ (\lambda x. x+cp)\ nell2)$
by $(metis\ add-0\ assms(4)\ assms(8)\ ndropn-0\ ndropn-nfirst\ nnth-nmap\ zero-enat-def\ zero-le)$
have 2: $nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x+cp)\ nell2))\ (the-enat((nlength\ nell1)) + i) =$
 $nnth\ (nmap\ (\lambda x. x+cp)\ nell2)\ i$
by $(simp\ add: 1\ assms\ nfuse-nnth-a)$
have 3: $nnth\ (nmap\ (\lambda x. x+cp)\ nell2)\ i = cp + (nnth\ nell2\ i)$
by $(simp\ add: assms)$
show $?thesis$
by $(simp\ add: 2\ 3)$
qed

lemma $nidx-nfuse-nnth-cp$:

assumes $nidx\ nell1$
 $(nnth\ nell1\ 0) = 0$
 $nidx\ nell2$
 $(nnth\ nell2\ 0) = 0$
 $nfinite\ nell1$
 $nfinite\ nell2$
 $nfinite\ nell$
 $nlast\ nell1 = cp$
 $nlast\ nell2 = the-enat((nlength\ nell)) - cp$
 $i \leq (nlength\ nell2)$
 $cp \leq nlength\ nell$
shows $nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x+cp)\ nell2))\ (the-enat((nlength\ nell1)) + i) = cp + (nnth\ nell2\ i)$
using $assms\ nridx-nfuse-nnth-cp\ nridx-nidx$ **by** $blast$

lemma $nridx-nfuse-nnth-cp-a$:

assumes $nridx\ R\ nell1$
 $(nnth\ nell1\ 0) = (0::nat)$
 $nridx\ R\ nell2$
 $(nnth\ nell2\ 0) = 0$
 $nfinite\ nell1$
 $nfinite\ nell2$
 $nfinite\ nell$
 $nlast\ nell1 = cp$
 $nlast\ nell2 = the-enat((nlength\ nell)) - cp$
 $i \leq ((nlength\ nell1)) + (nlength\ nell2)$
 $the-enat((nlength\ nell1)) \leq i$
 $cp \leq nlength\ nell$

shows $\text{nnth } (\text{nfuse } \text{nell1 } (\text{nmap } (\lambda x. x+cp) \text{ nell2})) (i) = cp + (\text{nnth } \text{nell2 } (i - \text{the-enat}((\text{nlength } \text{nell1}))))$

proof –

have 1: $i = (\text{the-enat } ((\text{nlength } \text{nell1})) + (i - \text{the-enat } ((\text{nlength } \text{nell1}))))$

by (*simp add: assms*)

have 2: $\text{enat } ((i::\text{nat}) - \text{the-enat } ((\text{nlength } \text{nell1}))) \leq \text{nlength } \text{nell2}$

using *assms*

by (*metis enat-add-sub-same enat-minus-mono1 enat-ord-simps(1)*
idiff-enat-enat infinity-ileE nfinite-conv-nlength-enat the-enat.simps)

show ?thesis **using** 1 2 *assms nridx-nfuse-nnth-cp*[of *R nell1 nell2 nell cp (i - the-enat ((nlength nell1)))*]

by *presburger*

qed

lemma *nidx-nfuse-nnth-cp-a*:

assumes *nidx nell1*

$(\text{nnth } \text{nell1 } 0) = (0::\text{nat})$

nidx nell2

$(\text{nnth } \text{nell2 } 0) = 0$

nfinite nell1

nfinite nell2

nfinite nell

$\text{nlast } \text{nell1} = cp$

$\text{nlast } \text{nell2} = \text{the-enat}((\text{nlength } \text{nell})) - cp$

$i \leq ((\text{nlength } \text{nell1})) + (\text{nlength } \text{nell2})$

$\text{the-enat}((\text{nlength } \text{nell1})) \leq i$

$cp \leq \text{nlength } \text{nell}$

shows $\text{nnth } (\text{nfuse } \text{nell1 } (\text{nmap } (\lambda x. x+cp) \text{ nell2})) (i) = cp + (\text{nnth } \text{nell2 } (i - \text{the-enat}((\text{nlength } \text{nell}))))$

using *assms nridx-nidx nridx-nfuse-nnth-cp-a* **by** *blast*

lemma *nridx-nfuse-nnth-cp-nlast*:

assumes *nridx R nell1*

$(\text{nnth } \text{nell1 } 0) = 0$

nridx R nell2

$(\text{nnth } \text{nell2 } 0) = 0$

nfinite nell1

nfinite nell2

nfinite nell

$\text{nlast } \text{nell1} = cp$

$\text{nlast } \text{nell2} = \text{the-enat}((\text{nlength } \text{nell})) - cp$

$i \leq (\text{nlength } \text{nell2})$

$cp \leq \text{nlength } \text{nell}$

shows $\text{nlast } (\text{nfuse } \text{nell1 } (\text{nmap } (\lambda x. x+cp) \text{ nell2})) = (\text{the-enat } ((\text{nlength } \text{nell})))$

proof –

have 1: $\text{nlast } (\text{nfuse } \text{nell1 } (\text{nmap } (\lambda x. x+cp) \text{ nell2})) = \text{nlast } (\text{nmap } (\lambda x. x+cp) \text{ nell2})$

using *assms*

by (*metis add-0 ndropn-0 ndropn-nfirst nlast-nfuse nnth-nmap zero-enat-def zero-le*)

have 2: $\text{nlast } (\text{nmap } (\lambda x. x+cp) \text{ nell2}) = cp + (\text{nlast } \text{nell2})$

by (*simp add: assms(6)*)

have 3: $cp + (nlast\ nell2) = (the-enat\ ((nlength\ nell)))$
using *assms*
by (*metis* *add.commute* *diff-add* *enat-ord-simps*(1) *nfinite-conv-nlength-enat* *the-enat.simps*)
show *?thesis*
by (*simp* *add: 1 3* *add.commute* *assms*(6))
qed

lemma *nidx-nfuse-nnth-cp-nlast*:

assumes *nidx nell1*
 $(nnth\ nell1\ 0) = 0$
nidx nell2
 $(nnth\ nell2\ 0) = 0$
nfinite nell1
nfinite nell2
nfinite nell
 $nlast\ nell1 = cp$
 $nlast\ nell2 = the-enat\ ((nlength\ nell)) - cp$
 $i \leq (nlength\ nell2)$
 $cp \leq nlength\ nell$
shows $nlast\ (nfuse\ nell1\ (nmap\ (\lambda x. x+cp)\ nell2)) = (the-enat\ ((nlength\ nell)))$
using *assms nridx-nidx nridx-nfuse-nnth-cp-nlast* **by** *blast*

lemma *nridx-nfuse-nnth-cp-infinite*:

assumes *nridx R nell1*
 $(nnth\ nell1\ 0) = (0::nat)$
nridx R nell2
 $(nnth\ nell2\ 0) = 0$
nfinite nell1
 $\neg nfinite\ nell2$
 $\neg nfinite\ nell$
 $nlast\ nell1 = cp$
shows $nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x+cp)\ nell2))\ (the-enat\ ((nlength\ nell1)) + i) = cp + (nnth\ nell2\ i)$
proof –
have 1: $nlast\ nell1 = nfirst(nmap\ (\lambda x. x+cp)\ nell2)$
by (*metis* *assms*(1) *assms*(2) *assms*(3) *assms*(4) *assms*(5) *assms*(8) *nridx-nfuse-nfirst-nlast*)
have 2: $nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x+cp)\ nell2))\ (the-enat\ ((nlength\ nell1)) + 0) = nlast\ nell1$
using *assms* **by** (*metis* 1 *add.right-neutral* *ntaken-nfuse* *ntaken-nlast*)
have 3: $nfirst(nmap\ (\lambda x. x+cp)\ nell2) = cp + (nnth\ nell2\ 0)$
using 1 *assms* **by** *auto*
have 8: $i \leq (nlength\ nell2)$
by (*metis* *assms*(6) *enat-ile* *nfinite-conv-nlength-enat* *wlog-linorder-le*)
have 10: $(nlength\ nell1) \leq enat\ (the-enat\ ((nlength\ nell1)) + (i::nat))$
using *assms*(5) *nfinite-nlength-enat* **by** *fastforce*
have 11: $enat\ (the-enat\ ((nlength\ nell1)) + i) \leq nlength\ (nfuse\ nell1\ (nmap\ (\lambda x::nat. x + cp)\ nell2))$
by (*metis* 10 *assms*(6) *enat-less-enat-plusI2* *enat-ord-code*(4) *enat-the-enat* *leD* *nfuse-nlength* *nlength-eq-enat-nfiniteD* *nlength-nmap* *order-less-imp-le*)
have 12: $(the-enat\ ((nlength\ nell1)) + i - the-enat\ ((nlength\ nell1))) = i$
by *auto*
have 4: $nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x+cp)\ nell2))\ (the-enat\ ((nlength\ nell1)) + i) = (nnth\ (nmap\ (\lambda x. x+cp)\ nell2)\ i)$


```

  by (simp add: 1 8 assms nfuse-nnth-a)
have 5: (nnth (nmap (λx. x+cp) nell2) i) = cp + (nnth nell2 i)
  by (simp add: 8)
show ?thesis
using 4 5 by presburger
qed

```

lemma *nidx-nfuse-nnth-cp-infinite*:

```

assumes nidx nell1
  (nnth nell1 0) = 0
  nidx nell2
  (nnth nell2 0) = 0
  nfinite nell1
  ¬nfinite nell2
  ¬nfinite nell
  nlast nell1 = cp
shows nnth (nfuse nell1 (nmap (λx. x+cp) nell2)) (the-enat((nlength nell1)) + i) = cp + (nnth nell2 i)
using assms nridx-nidx nridx-nfuse-nnth-cp-infinite by blast

```

lemma *nidx-nfuse-nidx*:

```

assumes nidx nell1
  nnth nell1 0 = 0
  nidx nell2
  nnth nell2 0 = 0
  nfinite nell1
  nlast nell1 = cp
  nfinite nell2
  nfinite nell
  nlast nell2 = the-enat((nlength nell)) - cp
  cp ≤ nlength nell
shows nidx (nfuse nell1 (nmap (λx. x+ cp) nell2)) ∧ (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) 0)
= 0

```

proof –

```

have 1: nlast nell1 = nfirst(nmap (λx. x+ cp) nell2)
  using assms(1) assms(2) assms(3) assms(4) assms(5) assms(6) nidx-nfuse-nfirst-nlast by blast
have 2: nfirst (nfuse nell1 (nmap (λx. x+ cp) nell2)) = nfirst nell1
  by (simp add: 1 nfirst-nfuse)
have 4:  $\bigwedge j. j \leq \text{nlength } nell1 \implies \text{nnth } (nfuse \text{ nell1 } (nmap (\lambda x. x+ cp) \text{ nell2})) j = \text{nnth } nell1 j$ 
  using assms by (simp add: 1 add-increasing2 nfuse-nlength nfuse-nnth)
have 40:  $\exists k1. \text{nlength } nell1 = (\text{enat } k1)$ 
  by (simp add: assms(5) nfinite-nlength-enat)
obtain k1 where 41:  $\text{nlength } nell1 = (\text{enat } k1)$ 
  using 40 by blast
have 5:  $\bigwedge j. (\text{nlength } nell1) \leq j \wedge j \leq (\text{nlength } nell1) + \text{nlength } nell2 \implies$ 
   $\text{nnth } (nfuse \text{ nell1 } (nmap (\lambda x. x+ cp) \text{ nell2})) j =$ 
   $cp + (\text{nnth } nell2 (j - \text{the-enat}((\text{nlength } nell1))))$ 
  using assms nidx-nfuse-nnth-cp-a[of nell1 nell2 nell cp]
  by (metis 41 enat-ord-simps(1) the-enat.simps)
have 45:  $\bigwedge j. k1 \leq j \wedge j \leq (\text{enat } (k1)) + \text{nlength } nell2 \implies$ 
   $\text{nnth } (nfuse \text{ nell1 } (nmap (\lambda x. x+ cp) \text{ nell2})) j =$ 

```

```

      cp + (nnth nell2 (j- (k1)))
    by (simp add: 41 5 order-less-imp-le)
  have 50: nlength(nfuse nell1 (nmap (λx. x+ cp) nell2)) = (nlength nell1) + nlength nell2
    by (simp add: nfuse-nlength)
  have 51: ∧j. enat (Suc j) ≤ nlength nell1 ⇒
      (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j) <
      (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) (Suc j))
    using 4 Suc-ile-eq assms(1) nidx-expand by auto
  have 52: ∧j. k1 ≤ j ∧ (Suc j) ≤ enat (k1) + nlength nell2 ⇒
      cp + (nnth nell2 (j- (k1))) <
      cp + (nnth nell2 ((Suc j)- (k1)))
    using assms(3) unfolding nidx-def
    by (metis Nat.add-diff-assoc add-strict-left-mono assms(3) enat.simps(3) enat-add-sub-same
        enat-minus-mono1 idiff-enat-enat nidx-expand plus-1-eq-Suc)
  have 6: ∧j. enat (Suc j) ≤ nlength(nfuse nell1 (nmap (λx. x+ cp) nell2)) ⇒
      (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j) <
      (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) (Suc j))
    proof -
      fix j
      assume a: enat (Suc j) ≤ nlength(nfuse nell1 (nmap (λx. x+ cp) nell2))
      show (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j) < (nnth (nfuse nell1 (nmap (λx. x+ cp)
nell2)) (Suc j))
    proof -
      have 61: enat (Suc j) ≤ nlength nell1 ⇒
          (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j) < (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2))
(Suc j))
        using 51 by blast
      have 62: k1 ≤ j ∧ (Suc j) ≤ nlength(nfuse nell1 (nmap (λx. x+ cp) nell2)) ⇒
          (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j) < (nnth (nfuse nell1 (nmap (λx. x+ cp) nell2))
(Suc j))
        using 41 5 50 52 Suc-ile-eq by force
      show ?thesis
        using 41 61 62 a by fastforce
      qed
    qed
  show ?thesis
  unfolding nidx-expand
    using 4 6 assms zero-enat-def by force
  qed

```

lemma *nidx-nfuse-nidx-infinite*:

```

  assumes nidx nell1
    nnth nell1 0 = 0
    nidx nell2
    nnth nell2 0 = 0
    nfinite nell1
    nlast nell1 = cp
    ¬nfinite nell2
    ¬nfinite nell

```

shows $nidx (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2)) \wedge (nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ 0)$

= 0

proof –

have 1: $nlast\ nell1 = nfirst(nmap\ (\lambda x. x + cp)\ nell2)$
by (metis add-0 assms(4) assms(6) assms(7) enat-le-plus-same(1) enat-le-plus-same(2) enat-the-enat
infinity-ileE ndropn-0 ndropn-nfirst nfinite-conv-nlength-enat nnth-nmap)
have 2: $nfirst\ (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2)) = nfirst\ nell1$
by (simp add: 1 nfirst-nfuse)
have 4: $\bigwedge j. j \leq nlength\ nell1 \implies nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ j = nnth\ nell1\ j$
by (simp add: 1 add-increasing2 nfuse-nlength nfuse-nnth)
have 40: $\exists k1. nlength\ nell1 = (enat\ k1)$
by (simp add: assms(5) nfinite-nlength-enat)
obtain k1 **where** 41: $nlength\ nell1 = (enat\ k1)$
using 40 **by** blast
have 5: $\bigwedge j. (nlength\ nell1) \leq j \wedge j \leq (nlength\ nell1) + nlength\ nell2 \implies$
 $nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ j =$
 $cp + (nnth\ nell2\ (j - the-enat((nlength\ nell1))))$
using assms nidx-nfuse-nnth-cp-infinite[of nell1 nell2 nell cp]
by (metis 41 enat-ord-simps(1) le-add-diff-inverse the-enat.simps)
have 45: $\bigwedge j. k1 \leq j \wedge j \leq (enat\ (k1)) + nlength\ nell2 \implies$
 $nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ j =$
 $cp + (nnth\ nell2\ (j - (k1)))$
using 41 5 enat-ord-simps(1) the-enat.simps **by** presburger
have 50: $nlength(nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2)) = (nlength\ nell1) + nlength\ nell2$
by (simp add: nfuse-nlength)
have 51: $\bigwedge j. enat\ (Suc\ j) \leq nlength\ nell1 \implies$
 $(nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ j) <$
 $(nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ (Suc\ j))$
using 4 Suc-ile-eq assms(1) nidx-expand **by** auto
have 52: $\bigwedge j. k1 \leq j \wedge (Suc\ j) \leq enat\ (k1) + nlength\ nell2 \implies$
 $cp + (nnth\ nell2\ (j - (k1))) <$
 $cp + (nnth\ nell2\ ((Suc\ j) - (k1)))$
by (metis Nat.add-diff-assoc add-strict-left-mono assms(3) enat.simps(3) enat-add-sub-same
enat-minus-mono1 idiff-enat-enat nidx-expand plus-1-eq-Suc)
have 53: $\bigwedge j. k1 \leq j \wedge (Suc\ j) \leq enat\ (k1) + nlength\ nell2 \implies$
 $nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ j <$
 $nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ (Suc\ j)$
by (metis 45 52 Suc-ile-eq nle-le not-less-eq-eq order-less-imp-le)
have 6: $\bigwedge j. enat\ (Suc\ j) \leq nlength(nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2)) \implies$
 $(nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ j) <$
 $(nnth\ (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ (Suc\ j))$
by (metis 41 50 51 53 enat-ord-simps(1) le-SucE linorder-le-cases)
show ?thesis **unfolding** nidx-expand
using 4 6 assms zero-enat-def **by** fastforce
qed

lemma nsubn-nfuse-nidx:

assumes nidx nl
nfinite nl
nfinite nell

$nl_{last} \, nl = (nlength \, nell)$
 $(Suc \, i) \leq (nlength \, nl)$
shows $nfuse \, (nsubn \, nell \, (nnth \, nl \, i) \, (nnth \, nl \, (Suc \, i))) \, (nsubn \, nell \, (nnth \, nl \, (Suc \, i)) \, (nl_{last} \, nl)) =$
 $(nsubn \, nell \, (nnth \, nl \, i) \, (nl_{last} \, nl))$
proof –
have 1: $(nnth \, nl \, i) \leq (nnth \, nl \, (Suc \, i))$
by $(simp \, add: \, assms(1) \, assms(5) \, nidx-less-eq)$
have 2: $nl_{last} \, nl = (nnth \, nl \, (the-enat(\, (nlength \, nl))))$
by $(simp \, add: \, assms(2) \, nnth-nlast)$
have 3: $1 \leq nlength \, nl$
by $(metis \, assms(5) \, dual-order.trans \, enat-0-iff(1) \, illess-Suc-eq \, le-zero-eq \, linorder-le-cases$
 $nat.simps(3) \, one-eSuc \, order-neq-le-trans)$
have 4: $(nnth \, nl \, (Suc \, i)) \leq (nl_{last} \, nl)$
by $(metis \, 2 \, assms(1) \, assms(2) \, assms(5) \, dual-order.eq-iff \, enat-ord-simps(1)$
 $nfinite-conv-nlength-enat \, nidx-less-eq \, the-enat.simps)$
have 5: $enat \, (nnth \, nl \, (Suc \, i)) \leq nlength \, nell$
by $(metis \, 4 \, assms(4) \, enat-ord-simps(1))$
have 6: $nl_{last} \, (nsubn \, nell \, (nnth \, nl \, i) \, (nnth \, nl \, (Suc \, i))) = (nnth \, nell \, (nnth \, nl \, (Suc \, i)))$
by $(simp \, add: \, 1 \, nsubn-def1 \, ntaken-nlast)$
have 7: $nfirst \, (nsubn \, nell \, (nnth \, nl \, (Suc \, i)) \, (nl_{last} \, nl)) = (nnth \, nell \, (nnth \, nl \, (Suc \, i)))$
by $(simp \, add: \, nsubn-def1 \, ntaken-ndropn-nfirst)$
show *?thesis*
by $(simp \, add: \, 1 \, 5 \, assms(4) \, nsubn-nfuse)$
qed

lemma *nidx-nfuse-split*:

assumes $nl_{last} \, nell1 = nfirst \, nell2$
shows $nridx \, R \, (nfuse \, nell1 \, nell2) \longleftrightarrow$
 $(if \, nfinite \, nell1 \, then \, nridx \, R \, nell1 \wedge nridx \, R \, nell2 \, else \, nridx \, R \, nell1)$

proof $(cases \, nfinite \, nell1)$

case *True*

then show *?thesis*

by $(metis \, assms \, nappend-nbutlast-nlast-id \, nbutlast-nfinite \, nellist.collapse(2) \, nellist.disc(1)$
 $nfuse-def1 \, nfuse-leftneutral \, nhd-nfuse \, nlast-NNil \, nridx-LCons-1 \, nridx-nappend-nfinite)$

next

case *False*

then show *?thesis* **by** $(simp \, add: \, assms \, nfuse-nappend)$

qed

lemma *nidx-all-le-nlast*:

assumes $nidx \, nell$

$nfinite \, nell$

$j \leq nlength \, nell$

shows $nnth \, nell \, j \leq nl_{last} \, nell$

using *assms*

by $(metis \, nfinite-conv-nlength-enat \, nidx-less-last-1 \, nnth-nlast \, order.order-iff-strict \, the-enat.simps)$

lemma *nidx-shiftm* :

assumes $nidx \, nell$

$nnth\ nell\ 0 = k$
shows $nidx\ (nmap\ (\lambda x.\ x - k)\ nell) \wedge nnth\ (nmap\ (\lambda x.\ x - k)\ nell)\ 0 = 0 \wedge k \leq (nnth\ nell\ 0)$
using *assms zero-enat-def*
by (*auto simp add: nidx-expand*)
(metis Suc-ile-eq add-diff-inverse-nat add-gr-0 assms(1) diff-less-mono nidx-gr-first
nnth-nmap order-less-imp-le zero-less-Suc zero-less-diff)

lemma *nidx-nsubn*:

assumes $k \leq n$
 $n \leq nlength\ nell$
 $nidx\ nell$
 $nnth\ nell\ 0 = 0$
shows $nidx\ (nsubn\ nell\ k\ n) \wedge nnth\ (nsubn\ nell\ k\ n)\ 0 = (nnth\ nell\ k)$
using *assms unfolding nidx-expand nsubn-def1*
by (*auto simp add: ntaken-nnth*)
(simp add: Nat.le-diff-conv2 add commute order-subst2)

lemma *nidx-ntaken-niterates-Suc*:

$nidx\ (ntaken\ n\ (niterates\ Suc\ 0))$
proof –
have 1: $nidx\ (ntaken\ n\ (niterates\ Suc\ 0)) =$
 $(\forall i::nat.$
 $enat\ (Suc\ i) \leq nlength\ (ntaken\ n\ (niterates\ Suc\ (0::nat))) \longrightarrow$
 $nnth\ (ntaken\ n\ (niterates\ Suc\ (0::nat)))\ i <$
 $nnth\ (ntaken\ n\ (niterates\ Suc\ (0::nat)))\ (Suc\ i))$
unfolding *nidx-expand* **by** *auto*
have 2: $(\forall i::nat.$
 $enat\ (Suc\ i) \leq nlength\ (ntaken\ n\ (niterates\ Suc\ (0::nat))) \longrightarrow$
 $nnth\ (ntaken\ n\ (niterates\ Suc\ (0::nat)))\ i <$
 $nnth\ (ntaken\ n\ (niterates\ Suc\ (0::nat)))\ (Suc\ i))$

proof
fix i
show $enat\ (Suc\ i) \leq nlength\ (ntaken\ n\ (niterates\ Suc\ (0::nat))) \longrightarrow$
 $nnth\ (ntaken\ n\ (niterates\ Suc\ (0::nat)))\ i <$
 $nnth\ (ntaken\ n\ (niterates\ Suc\ (0::nat)))\ (Suc\ i)$
proof (*induct i arbitrary: n*)
case 0
then show ?*case*
by (*simp add: ntaken-nnth*)
next
case ($Suc\ i$)
then show ?*case* **by** (*simp add: ntaken-nnth*)
qed
qed
show ?*thesis*
using 1 2 **by** *fastforce*
qed

lemma *forall-nappend-single*:

assumes $nfinite\ l$
shows $(\forall\ i < nlength\ (nappend\ l\ (NNil\ x)).\ P\ i\ (Suc\ i)) \longleftrightarrow$
 $(P\ (the-enat(nlength\ l))\ (Suc(the-enat(nlength\ l))) \wedge (\forall\ i < nlength\ l.\ P\ i\ (Suc\ i)))$
proof (cases $nlength\ l = 0$)
case *True*
then show *?thesis* **using** *assms*
by (*metis enat-0-iff(1) gr-implies-not-zero illess-eSuc0 nappend-NNil*
ndropn-0 ndropn-nlast nlength-NCons nlength-NNil the-enat-0)
next
case *False*
then show *?thesis* **using** *assms*
by (*simp add: enat-the-enat nfinite-conv-nlength-enat*)
(metis dual-order.order-iff-strict illess-Suc-eq plus-1-eSuc(2) the-enat.simps)
qed

lemma *forall-ncons-split:*

assumes $nfinite\ l$
shows $(\forall\ i.\ enat\ i < nlength\ (NCons\ 0\ l) \longrightarrow$
 $P\ (nnth\ (NCons\ 0\ l)\ i)\ (nnth\ (NCons\ 0\ l)\ (Suc\ i))) \longleftrightarrow$
 $P\ 0\ (nnth\ l\ 0) \wedge$
 $(\forall\ i.\ enat\ i < nlength\ l \longrightarrow$
 $P\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))$
using *assms*
by (*auto simp add: zero-enat-def*)
(metis nnth-0 zero-enat-def zero-le,
metis eSuc-enat ileI1 nnth-Suc-NCons,
metis Suc-ile-eq Suc-pred less-le-not-le nle-le nnth-0 nnth-Suc-NCons zero-le)

lemma *forall-ncons-split-alt:*

assumes $\neg nfinite\ l$
shows $(\forall\ i.\ enat\ i < nlength\ (NCons\ 0\ l) \longrightarrow$
 $P\ (nnth\ (NCons\ 0\ l)\ i)\ (nnth\ (NCons\ 0\ l)\ (Suc\ i))) \longleftrightarrow$
 $P\ 0\ (nnth\ l\ 0) \wedge$
 $(\forall\ i.\ enat\ i < nlength\ l \longrightarrow$
 $P\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))$
using *assms*
by *auto*
(metis nnth-0 zero-enat-def zero-le,
metis eSuc-enat ileI1 nnth-Suc-NCons,
metis Suc-ile-eq Suc-pred le-zero-eq less-le-not-le nle-le nnth-0 nnth-Suc-NCons)

lemma *nidx-Ncons-shift:*

assumes $nfinite\ l$
 $nidx\ l$
shows $nidx\ (NCons\ 0\ (nmap\ (\lambda x.\ x + Suc\ ia)\ l))$
proof –
have 1: $nidx\ (NCons\ 0\ (nmap\ (\lambda x.\ x + Suc\ ia)\ l)) \longleftrightarrow$
 $0 < nfirst\ (nmap\ (\lambda x.\ x + Suc\ ia)\ l) \wedge$

```

      nidx (nmap (λx. x+ Suc ia) l)
    by auto
  have 2: 0 < nfirst (nmap (λx. x+ Suc ia) l)
    by (metis Zero-not-Suc add-Suc-right bot-nat-0.not-eq-extremum enat-defs(1) ndropn-0
        ndropn-nfirst nnth-nmap zero-le)
  have 3: nidx (nmap (λx. x+ Suc ia) l)
    using nidx-expand Suc-ile-eq assms(2) by force
  show ?thesis
  using 2 3 by auto
qed

```

lemma *nidx-Ncons-shift-alt:*

```

  assumes ¬nfinite l
          nidx l
  shows  nidx (NCons 0 (nmap (λx. x+ Suc ia) l))
proof -
  have 1: nidx (NCons 0 (nmap (λx. x+ Suc ia) l)) ⟷
    0 < nfirst (nmap (λx. x+ Suc ia) l) ∧
    nidx (nmap (λx. x+ Suc ia) l)
    by auto
  have 2: 0 < nfirst (nmap (λx. x+ Suc ia) l)
    by (metis Zero-not-Suc add-Suc-right bot-nat-0.not-eq-extremum enat-defs(1) ndropn-0
        ndropn-nfirst nnth-nmap zero-le)
  have 3: nidx (nmap (λx. x+ Suc ia) l)
    using nidx-expand Suc-ile-eq assms(2) by force
  show ?thesis
  using 2 3 by auto
qed

```

lemma *infinite-nidx-imp-infinite-interval:*

```

  assumes ¬ nfinite l
          nidx l
          (nnth l 0) = 0
          ∀ i. (nnth l i) ≤ nlength s
  shows ¬ nfinite s
proof
  assume nfinite s
  thus False
    using assms
  proof (induct s rule: nfinite-induct)
  case (NNil y)
  then show ?case
  by (metis dual-order.antisym enat-ord-simps(1) linorder-linear nfinite-ntaken nidx-gr-first nlength-NNil
      not-gr-zero ntaken-all zero-le zero-less-Suc)
  next
  case (NCons x nell)
  then show ?case
  proof -
    have 1: ⋀ j. (nnth l j) < (nnth l (Suc j))
      by (metis assms(1) assms(2) linorder-le-cases nfinite-ntaken nidx-expand ntaken-all)

```

```

have 2:  $\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } (NCons \ x \ \text{nell}) \implies \text{False}$ 
by (metis 1 NCons.hyps(2) assms(1) assms(2) assms(3) dual-order.strict-iff-order
    enat-ord-simps(1) iless-Suc-eq linorder-not-le nlength-NCons)
show ?thesis
using 2 NCons.prem(4) by auto
qed
qed
qed

```

1.3.7 nlastnfirst

```

lemma nlastnfirst-def2:
nlastnfirst = llastfirst  $\circ$  (lmap llist-of-nellist  $\circ$  llist-of-nellist)
by (simp add: map-fun-def nlastnfirst-def)

```

```

lemma nlastnfirst-NNil[simp]:
  nlastnfirst (NNil x)
apply transfer
by simp

```

```

lemma nlastnfirst-LCons[simp]:
  shows nlastnfirst (NCons nell nells)  $\longleftrightarrow$ 
    nlast nell = nfirst (nfirst nells)  $\wedge$  nlastnfirst nells
proof -
let ?n2l = (lmap llist-of-nellist  $\circ$  llist-of-nellist)
have 1: nlastnfirst (NCons nell nells) =
  (llastfirst  $\circ$  ?n2l) (NCons nell nells)
by (simp add: nlastnfirst-def2)
have 2: (llastfirst  $\circ$  ?n2l) (NCons nell nells) =
  (llast (llist-of-nellist nell) = lfirst (lfirst (?n2l nells))  $\wedge$  llastfirst (?n2l nells))
by simp
have 3: llastfirst (?n2l nells) = nlastnfirst nells
by (simp add: nlastnfirst.rep-eq)
have 4: llast (llist-of-nellist nell) = nlast nell
by (metis llast-linfinite nfinite-def nlast-llast nlast-not-nfinite)
have 5: lfirst (lfirst (?n2l nells)) = nfirst (nfirst nells)
by (simp add: lfirst-def llist-of-nellist.simps(2) nfirst-def)
show ?thesis
using 1 2 3 4 5 by presburger
qed

```

```

lemma nlastnfirst-def1:
shows nlastnfirst nells =
  ( $\forall i. (\text{Suc } i) \leq \text{nlength } \text{nells} \longrightarrow \text{nlast}(\text{nnth } \text{nells } i) = \text{nfirst}(\text{nnth } \text{nells } (\text{Suc } i))$ )
proof -
let ?n2l = (lmap llist-of-nellist  $\circ$  llist-of-nellist)
have 1: nlastnfirst nells = (llastfirst  $\circ$  ?n2l) nells
by (simp add: nlastnfirst-def2)
have 2: (llastfirst  $\circ$  ?n2l) nells = llastfirst (?n2l nells)

```


by *simp*
have 3: $\text{llastlfirst } (?n2l \text{ nells}) \longleftrightarrow$
 $(\forall i. (\text{Suc } i) < \text{llength } (?n2l \text{ nells}) \longrightarrow$
 $\text{llast } (\text{lnth } (?n2l \text{ nells}) i) = \text{lfirst } (\text{lnth } (?n2l \text{ nells}) (\text{Suc } i)))$
 using *llastlfirst-def* by *blast*
have 4: $\text{epred } (\text{llength } (?n2l \text{ nells})) = \text{nlength nells}$
 by *simp*
have 5: $\bigwedge xs \ j. j \leq \text{nlength } xs \longrightarrow \text{nnth } xs \ j = \text{lnth } (\text{llist-of-nellist } xs) \ j$
 unfolding *nnth-def* by *auto*
have 6: $\bigwedge lx \ j. j < \text{llength } lx \wedge \neg \text{lnull } lx \longrightarrow \text{lnth } lx \ j = \text{nnth } (\text{nellist-of-llist } lx) \ j$
 by (*metis* 5 *co.enat.exhaust-sel iless-Suc-eq llength-eq-0 nellist-of-llist-inverse nlength.abs-eq*)
have 7: $\bigwedge xs. \text{nlast } xs = \text{llast } (\text{llist-of-nellist } xs)$
 by (*metis* *llast-linfinite nfinite-def nlast-llast nlast-not-nfinite*)
have 8: $\bigwedge xs. \text{nfirst } xs = \text{lfirst } (\text{llist-of-nellist } xs)$
 by (*simp* add: *lfirst-def llist-of-nellist.simps*(2) *nfirst-def*)
have 9: $\bigwedge lx. \neg \text{lnull } lx \Longrightarrow \text{llast } lx = \text{nlast } (\text{nellist-of-llist } lx)$
 by (*simp* add: 7)
have 10: $\bigwedge lx. \neg \text{lnull } lx \Longrightarrow \text{lfirst } lx = \text{nfirst } (\text{nellist-of-llist } lx)$
 by (*simp* add: 8)
have 11: $\bigwedge j. j < \text{llength } (?n2l \text{ nells}) \longrightarrow$
 $\text{llast } (\text{lnth } (?n2l \text{ nells}) j) =$
 $\text{nlast } (\text{nellist-of-llist } (\text{lnth } (?n2l \text{ nells}) j))$
 by (*simp* add: 9)
have 12: $\bigwedge j. (\text{enat } j) < \text{llength } (?n2l \text{ nells}) \longrightarrow$
 $\text{nlast } (\text{nellist-of-llist } (\text{lnth } (?n2l \text{ nells}) j)) =$
 $\text{nlast } (\text{nellist-of-llist } (\text{llist-of-nellist } (\text{lnth } (\text{llist-of-nellist nells}) j)))$
 by *simp*
have 13: $\bigwedge j. (\text{enat } j) < \text{llength } (?n2l \text{ nells}) \longrightarrow$
 $\text{nlast } (\text{nellist-of-llist } (\text{llist-of-nellist } (\text{lnth } (\text{llist-of-nellist nells}) j))) =$
 $\text{nlast } (\text{lnth } (\text{llist-of-nellist nells}) j)$
 by *auto*
have 14: $\bigwedge j. (\text{enat } j) < \text{llength } (?n2l \text{ nells}) \longrightarrow$
 $\text{nlast } (\text{lnth } (\text{llist-of-nellist nells}) j) = \text{nlast}(\text{nnth nells } j)$
 by (*simp* add: 6)
have 15: $\bigwedge j. j < \text{llength } (?n2l \text{ nells}) \longrightarrow$
 $\text{llast } (\text{lnth } (?n2l \text{ nells}) j) =$
 $\text{nlast}(\text{nnth nells } j)$
 using 11 12 13 14 by *presburger*
have 16: $\bigwedge j. (\text{Suc } j) < \text{llength } (?n2l \text{ nells}) \longrightarrow$
 $\text{lfirst } (\text{lnth } (?n2l \text{ nells}) (\text{Suc } j)) =$
 $\text{nfirst } (\text{nellist-of-llist } (\text{lnth } (?n2l \text{ nells}) (\text{Suc } j)))$
 by (*simp* add: 10)
have 17: $\bigwedge j. (\text{Suc } j) < \text{llength } (?n2l \text{ nells}) \longrightarrow$
 $\text{nfirst } (\text{nellist-of-llist } (\text{lnth } (?n2l \text{ nells}) (\text{Suc } j))) =$
 $\text{nfirst } (\text{nellist-of-llist } (\text{llist-of-nellist } (\text{lnth } (\text{llist-of-nellist nells}) (\text{Suc } j))))$
 by *simp*
have 18: $\bigwedge j. (\text{Suc } j) < \text{llength } (?n2l \text{ nells}) \longrightarrow$
 $\text{nfirst } (\text{nellist-of-llist } (\text{llist-of-nellist } (\text{lnth } (\text{llist-of-nellist nells}) (\text{Suc } j)))) =$
 $\text{nfirst } (\text{lnth } (\text{llist-of-nellist nells}) (\text{Suc } j))$
 by *auto*

have 19: $\bigwedge j. (Suc\ j) < llength\ (?n2l\ nells) \longrightarrow$
 $nfirst\ (lnth\ (l\!list\text{-}of\text{-}nellist\ nells)\ (Suc\ j)) = nfirst\ (nnth\ nells\ (Suc\ j))$
by (*simp add: 6*)
have 20: $\bigwedge j. (Suc\ j) < llength\ (?n2l\ nells) \longrightarrow$
 $lfirst\ (lnth\ (?n2l\ nells)\ (Suc\ j)) = nfirst\ (nnth\ nells\ (Suc\ j))$
using 16 17 18 19 **by** *presburger*
have 21: $\bigwedge j. (Suc\ j) < llength\ (?n2l\ nells) \longleftrightarrow$
 $(Suc\ j) \leq nlength\ nells$
by (*metis 4 co.enat.exhaust-sel dual-order.strict-iff-order enat-0-iff(1) epred-0 iless-Suc-eq nat.simps(3)*)
have 22: $(\forall i. (Suc\ i) < llength\ (?n2l\ nells) \longrightarrow$
 $llast\ (lnth\ (?n2l\ nells)\ i) = lfirst\ (lnth\ (?n2l\ nells)\ (Suc\ i)))$
 \longleftrightarrow
 $(\forall i. (Suc\ i) \leq nlength\ nells \longrightarrow nlast\ (nnth\ nells\ i) = nfirst\ (nnth\ nells\ (Suc\ i)))$
by (*metis 15 20 21 Suc-ile-eq order-less-imp-le*)
have 23: $llastlfirst\ (?n2l\ nells) \longleftrightarrow$
 $(\forall i. (Suc\ i) \leq nlength\ nells \longrightarrow nlast\ (nnth\ nells\ i) = nfirst\ (nnth\ nells\ (Suc\ i)))$
using 22 3 **by** *presburger*
show *?thesis* **using** 1 23 **by** *auto*
qed

lemma *nlastnfirst-nridx*:

$nlastnfirst\ nells = nridx\ (\lambda\ a\ b. nlast\ a = nfirst\ b)\ nells$
by (*simp add: nlastnfirst-def1 nridx-expand*)

lemma *nlastnfirst-nappend-nfinite*:

assumes *nfinite nells*
shows $nlastnfirst\ (nappend\ nells\ nellys) \longleftrightarrow$
 $nlastnfirst\ nells \wedge nlastnfirst\ nellys \wedge nlast\ (nlast\ nells) = nfirst\ (nfirst\ nellys)$
using *assms*
by (*metis (mono-tags, lifting) nlastnfirst-nridx nridx-nappend-nfinite*)

1.3.8 nfusecat

lemma *nfusecat-def2*:

$nfusecat = nellist\text{-}of\text{-}l\!list \circ lfusecat \circ (lmap\ l\!list\text{-}of\text{-}nellist \circ l\!list\text{-}of\text{-}nellist)$
by (*simp add: map-fun-def nfusecat-def*)

lemma *not-null-lfusecat*:

$\neg lnull((lfusecat \circ (lmap\ l\!list\text{-}of\text{-}nellist \circ l\!list\text{-}of\text{-}nellist))\ nells)$
by (*simp add: llist-of-nellist.code*)

lemma *nfusecat-def3*:

$((l\!list\text{-}of\text{-}nellist \circ nfusecat)\ nells) =$
 $((lfusecat \circ (lmap\ l\!list\text{-}of\text{-}nellist \circ l\!list\text{-}of\text{-}nellist))\ nells)$

proof –

let *?n2l* = $(lmap\ l\!list\text{-}of\text{-}nellist \circ l\!list\text{-}of\text{-}nellist)$

have 1: $l\!list\text{-}of\text{-}nellist \circ nfusecat =$
 $l\!list\text{-}of\text{-}nellist \circ nellist\text{-}of\text{-}l\!list \circ lfusecat \circ ?n2l$

```

  by (simp add: nfusecat-def2 rewriteL-comp-comp)
have 2:  $\neg \text{Inull}((\text{lfusecat} \circ ?n2l) \text{ nells})$ 
using not-null-lfusecat by auto
have 3:  $(\text{llist-of-nellist} \circ \text{nellist-of-llist} \circ \text{lfusecat} \circ ?n2l) \text{ nells} =$ 
 $(\text{lfusecat} \circ ?n2l) \text{ nells}$ 
using 2 nellist-of-llist-inverse by simp
show ?thesis
by (metis 1 3)
qed

```

lemma *nfusecat-NNil*[simp]:

nfusecat (NNil nell) = nell

apply *transfer*

by *simp*

lemma *nfusecat-NCons*[simp]:

nfusecat (NCons nell nells) = nfuse nell (nfusecat nells)

apply *transfer*

by (simp add: lfuse-def llist.case-eq-if)

lemma *nfusecat-nfirst*:

assumes $(\exists x1. \text{nells} = \text{NNil } x1)$

shows *nfusecat nells = nfirst nells*

proof –

obtain *nell* **where** 1: *nells = NNil nell*

using *assms* **by** *auto*

have 2: *nfusecat nells = nell*

by (simp add: 1)

have 3: *nfirst nells = nell*

by (metis 1 ndropn-0 ndropn-nfirst nth-NNil)

show ?thesis

by (simp add: 2 3)

qed

lemma *nfusecat-NCons-nfirst*:

assumes $(\forall \text{nell}. \text{nells} \neq \text{NNil } \text{nell})$

shows *nfusecat nells = nfuse (nfirst nells) (nfusecat (ntl nells))*

by (metis *assms nellist-split-2-first nlast-NNil nfusecat-NCons not-le-imp-less ntaken-0 ntaken-all ntaken-nlast zero-enat-def*)

lemma *nfusecat-expand*:

nfusecat nells =

(if is-NNil nells then nfirst nells else nfuse (nfirst nells) (nfusecat (ntl nells)))

unfolding *is-NNil-def*

using *nfusecat-NCons-nfirst nfusecat-nfirst* **by** *simp blast*

lemma *nfusecat-expand-case*:

nfusecat nells = (case nells of (NNil nell) \Rightarrow nell |

$(NCons\ nell'\ nells1) \Rightarrow nfuse\ nell'\ (nfusecat\ nells1)$

by (*metis is-NNil-def nelist.case-eq-if nelist.collapse(2) nlast-NNil nfusecat-NCons nfusecat-NNil*)

lemma *nmap-nfusecat:*

$nmap\ f\ (nfusecat\ nells) = (nfusecat\ (nmap\ (\ nmap\ f\)\ nells))$

proof –

let $?n2l = (lmap\ llist-of-nelist \circ llist-of-nelist)$

have 1: $nmap\ f\ (nfusecat\ nells) =$
 $nmap\ f\ ((nelist-of-llist \circ lfusecat \circ ?n2l)\ nells)$

by (*simp add: nfusecat-def2*)

have 2: $nmap\ f\ ((nelist-of-llist \circ lfusecat \circ ?n2l)\ nells) =$
 $nelist-of-llist\ (lmap\ f\ ((lfusecat \circ ?n2l)\ nells))$

using *nmap-nelist-of-llist not-null-lfusecat* **by** *auto*

have 3: $(lmap\ f\ ((lfusecat \circ ?n2l)\ nells)) =$
 $lfusecat\ (lmap\ (lmap\ f)\ ((lmap\ llist-of-nelist \circ llist-of-nelist)\ nells))$

by (*simp add: lmap-lfusecat*)

have 4: $(nfusecat\ (nmap\ (\ nmap\ f\)\ nells)) =$
 $(nelist-of-llist \circ lfusecat \circ ?n2l)\ (nmap\ (\ nmap\ f\)\ nells)$

by (*simp add: nfusecat-def2*)

have 8: $(lmap\ (lmap\ f)\ (lmap\ llist-of-nelist\ (llist-of-nelist\ nells))) =$
 $(lmap\ llist-of-nelist\ (lmap\ (nmap\ f)\ (llist-of-nelist\ nells)))$

using *lmap-llist-of-nelist-nmap* **by** *blast*

show *?thesis*

using 1 2 3 4 8 **by** *fastforce*

qed

lemma *nfirsr-nfuse-var:*

$nfirsr\ (nfuse\ nellx\ nelly) = nfirsr\ nellx$

by (*metis nfuse-def1 nlast-NNil ntaken-0 ntaken-nappend1 zero-enat-def zero-le*)

lemma *nfirsr-nfusecat-nfirsr:*

shows $nfirsr(nfusecat\ nells) = nfirsr(nfirsr\ nells)$

proof (*cases nells*)

case (*NNil nell*)

then show *?thesis*

by (*simp add: nfusecat-expand*)

next

case (*NCons nell nells1*)

then show *?thesis*

proof –

have 1: $nfusecat\ (NCons\ nell\ nells1) = nfuse\ nell\ (nfusecat\ nells1)$

by (*simp*)

have 2: $nfirsr\ (nfuse\ nell\ (nfusecat\ nells1)) = nfirsr\ nell$

using *nfirsr-nfuse-var* **by** *auto*

have 3: $nfirsr(nfirsr\ (NCons\ nell\ nells1)) = nfirsr\ nell$

by (*metis 1 nfirsr-nfuse-var nfusecat-expand*)

show *?thesis*

using 1 2 3 *NCons* **by** *fastforce*

qed
qed

lemma *nfusecat-nlength*:

shows $nfusecat (NCons\ nell\ nells) = nfusecat\ nell$
by (*simp add: nfusecat-def*)

lemma *nfusecat-nlength-eq-zero-conv*:

$nlength\ (nfusecat\ nells) = 0 \longleftrightarrow (\forall\ nell \in nset\ nells.\ nlength\ nell = 0)$
proof –
let $?n2l = (lmap\ llist-of-nellist \circ llist-of-nellist)$
have 1: $nlength\ (nfusecat\ nells) =$
 $nlength\ ((nellist-of-llist \circ lfusecat \circ ?n2l)\ nells)$
by (*simp add: nfusecat-def2*)
have 2: $nlength\ ((nellist-of-llist \circ lfusecat \circ ?n2l)\ nells) =$
 $epred\ (llength\ (lfusecat\ (?n2l\ nells)))$
by *simp*
 $(metis\ lbutlast-snoc\ llength-lbutlast\ nellist-of-llist-a-inverse\ nlength.rep-eq)$
have 3: $nlength\ (nfusecat\ nells) = 0 \longleftrightarrow$
 $epred\ (llength\ (lfusecat\ (?n2l\ nells))) = 0$
using 1 2 **by** *presburger*
have 4: $llength\ (?n2l\ nells) > 0$
by *simp*
have 5: $(llength\ (lfusecat\ (?n2l\ nells))) > 0$
by (*simp add: lfusecat-not-lnull-var*)
have 6: $epred\ (llength\ (lfusecat\ (?n2l\ nells))) = 0 \longleftrightarrow$
 $(llength\ (lfusecat\ (lmap\ llist-of-nellist\ (llist-of-nellist\ nells)))) = 1$
by (*metis 5 comp-apply epred-1 epred-inject not-iless0 zero-one-enat-neq(1)*)
have 7: $(llength\ (lfusecat\ (?n2l\ nells))) \leq 1 \longleftrightarrow$
 $(\forall\ nell \in lset\ (?n2l\ nells).\ llength\ nell \leq 1)$
using *lfusecat-all-empty-or-LNil lset-lltl-llength-var* **by** *blast*
have 8: $(\forall\ nell \in lset\ (?n2l\ nells).\ llength\ nell > 0)$
by *simp*
have 9: $(llength\ (lfusecat\ (?n2l\ nells))) = 1 \longleftrightarrow$
 $(\forall\ nell \in lset\ (?n2l\ nells).\ llength\ nell = 1)$
by (*metis 5 7 8 ileI1 one-eSuc order-antisym-conv*)
have 10: $(\forall\ nell \in lset\ (?n2l\ nells).\ llength\ nell = 1) \longleftrightarrow$
 $(\forall\ nell \in nset\ nells.\ nlength\ nell = 0)$
by *simp*
 $(metis\ co.enat.exhaust-sel\ epred-1\ llength-eq-0\ llist-of-nellist-not-lnull\ nlength.rep-eq\ zero-neq-one)$
show *?thesis* **using** 3 6 9 10 **by** *simp*
qed

lemma *is-NNil-nfusecat-a*:

assumes $\forall i.\ i \leq nlength\ nells \longrightarrow is-NNil\ (nnth\ nells\ i)$
shows $is-NNil\ (nfusecat\ nells)$
using *assms nfusecat-nlength-eq-zero-conv[of nells]*

by (*metis in-nset-conv-nnth is-NNil-def nle-le nlength-NNil ntaken-0 ntaken-all zero-enat-def*)

lemma *is-NNil-nfusecat-b*:

assumes *is-NNil(nfusecat nells)*

shows $\forall i. i \leq nlength\ nells \longrightarrow is-NNil\ (nnth\ nells\ i)$

using *assms nfusecat-nlength-eq-zero-conv[of nells]*

by (*metis in-nset-conv-nnth nelist.collapse(1) nelist.collapse(2) nlength-NCons nlength-NNil zero-ne-eSuc*)

lemma *ntl-lfusecat* :

shows *ntl(nfusecat nells) =*

(if is-NNil nells then ntl (nfirst nells) else

(if is-NNil (nfirst nells) then

if is-NNil (nfusecat (ntl nells))

then ntl (nfirst nells)

else ntl (nfusecat (ntl nells))

else

if is-NNil (nfusecat (ntl nells))

then ntl (nfirst nells)

else nappend (ntl (nfirst nells)) (ntl (nfusecat (ntl nells)))))

proof (*cases nells*)

case (*NNil nell*)

then show *?thesis* **by** (*metis nelist.disc(1) nfusecat-NNil nfusecat-expand*)

next

case (*NCons nell nells1*)

then show *?thesis*

using *nfusecat-NCons[of nell nells1] ntl-nfuse[of nell (nfusecat nells1)]* **by** *simp*
(metis ndropn-0 ndropn-nfirst nnth-0)

qed

lemma *nfusecat-nappend*:

assumes *nfinite llx*

shows *nfusecat (nappend llx lly) = nfuse (nfusecat llx) (nfusecat lly)*

proof –

let *?n2l = (lmap llist-of-nelist \circ llist-of-nelist)*

have 1: *nfusecat (nappend llx lly) =*

((nlist-of-llist \circ lfusecat \circ ?n2l) (nappend llx lly))

by (*simp add: nfusecat-def2*)

have 2: *((nlist-of-llist \circ lfusecat \circ ?n2l) (nappend llx lly)) =*

nlist-of-llist (lfusecat (?n2l (nappend llx lly)))

by *simp*

have 3: *(lmap llist-of-nelist (nappend llx lly)) =*

lappend (lmap llist-of-nelist llx) (lmap llist-of-nelist lly)

by (*simp add: llist-of-nelist-nappend*)

have 4: *(lmap llist-of-nelist (lappend (lmap llist-of-nelist llx) (lmap llist-of-nelist lly))) =*

lappend (?n2l llx) (?n2l lly)

using *lmap-lappend-distrib* **by** *auto*

have 5: *(lfusecat (lappend (?n2l llx) (?n2l lly))) =*

```

    lfuse (lfusecat (?n2l llx)) (lfusecat (?n2l lly))
  using assms lfinite-lmap lfusecat-lappend nfinite-def by (metis comp-def)
have 6: nellist-of-llist (lfuse (lfusecat (?n2l llx)) (lfusecat (?n2l lly))) =
    nfuse (nfusecat llx) (nfusecat lly)
  by (simp add: llist-of-nellist.code nfuse.abs-eq nfusecat-def2)
show ?thesis
using 1 2 3 4 5 6 by simp
qed

lemma nfusecat-split:
  assumes (Suc n) ≤ nlength nells
  shows nfusecat nells = nfuse (nfusecat (ntaken n nells)) (nfusecat (ndropn (Suc n) nells))
  using assms
  by (metis nappend-ntaken-ndropn nfinite-ntaken nfusecat-nappend)

lemma nfusecat-split-1:
  assumes (Suc n) ≤ nlength nells
  shows nfusecat nells = nfuse (nfusecat (ntake n nells)) (nfusecat (ndropn (Suc n) nells))
  using assms
  by (simp add: nfusecat-split ntake-eq-ntaken)

lemma nfuse-nfinite:
  shows nfinite (nfuse nellx nelly) ⟷ nfinite nellx ∧ nfinite nelly
  apply transfer
  using lfuse-lfinite by blast

lemma nfusecat-nfinite:
  assumes ∀ nell ∈ nset nells. nlength nell > 0
    ∀ nell ∈ nset nells. nfinite nell
    nlastnfirst nells
  shows nfinite (nfusecat nells) ⟷ nfinite nells
  proof -
    let ?n2l = (lmap llist-of-nellist ∘ llist-of-nellist)
    have 1: nfinite (nfusecat nells) ⟷
      nfinite (((nellist-of-llist ∘ lfusecat ∘ ?n2l) nells))
      by (simp add: nfusecat-def2)
    have 2: nfinite (((nellist-of-llist ∘ lfusecat ∘ ?n2l) nells)) ⟷
      lfinite (llist-of-nellist (((nellist-of-llist ∘ lfusecat ∘ ?n2l) nells)))
      using nfinite-def by blast
    have 3: lfinite (llist-of-nellist (((nellist-of-llist ∘ lfusecat ∘ ?n2l) nells))) ⟷
      lfinite (lfusecat (?n2l nells))
      by (simp add: lbutlast-lfinite)
    have 4: ∀ nell ∈ lset (llist-of-nellist nells). nlength nell > 0
      using assms(1) by force
    have 5: ∀ nell ∈ lset (llist-of-nellist nells). epred(llength (llist-of-nellist nell)) > 0
      using 4 by fastforce
    have 6: ∀ nell ∈ lset (llist-of-nellist nells). ¬ lnull (ltl (llist-of-nellist nell))
      by (metis 5 epred-llength less-numeral-extra(3) llength-LNil llist.collapse(1))
    have 7: ∀ nell ∈ lset (?n2l nells). ¬ lnull (ltl nell)

```

```

  by (simp add: 6)
have 8:  $\forall nell \in lset (?n2l\ nells). \text{lfinitel nell}$ 
  using assms(2) nfinite-def by fastforce
have 9: llastlfirst (?n2l nells)
  using assms(3) nlastnfirst.rep-eq by auto
have 10: lfinitel (lfusecat (?n2l nells))  $\longleftrightarrow$  lfinitel (?n2l nells)
  using 7 8 9 lfusecat-lfinitel by blast
have 11: lfinitel (?n2l nells)  $\longleftrightarrow$  nfinite nells
  by (simp add: nfinite-def)
show ?thesis using 1 2 3 10 11 by presburger
qed

```

lemma nlastfirst-nfusecat-nlast:

assumes nfinite nells

nlastnfirst nells

$\forall nell \in nset\ nells. \text{nfinitel nell}$

shows nlast(nfusecat nells) = nlast(nlast nells)

proof –

let ?n2l = (lmap llist-of-nellist \circ llist-of-nellist)

have 1: nlast(nfusecat nells) =
 nlast (((nellist-of-llist \circ lfusecat \circ ?n2l) nells))

by (simp add: nfusecat-def2)

have 2: nlast (((nellist-of-llist \circ lfusecat \circ ?n2l) nells)) =
 llast (llist-of-nellist (((nellist-of-llist \circ lfusecat \circ ?n2l) nells)))

by (metis (no-types, lifting) llast-linfinite nfinite-def nlast-llast nlast-not-nfinite)

have 3: llast (llist-of-nellist (((nellist-of-llist \circ lfusecat \circ ?n2l) nells))) =
 llast (lfusecat (?n2l nells))

by (metis (no-types, lifting) 2 comp-def lbutlast.disc-iff(1) nellist-of-llist-a-inverse
 nlast-nellist-of-llist-a-lnull not-lnull-eq-lappend-lbutlast-llast)

have 4: lfinitel (?n2l nells)

using assms(1) lfinitel-lmap nfinite-def **by** auto

have 5: $\neg \text{lnull} (?n2l\ nells)$

by simp

have 6: $\forall nell \in lset (?n2l\ nells). \neg \text{lnull nell}$

by simp

have 7: llastlfirst (?n2l nells)

using assms(2) nlastnfirst.rep-eq **by** auto

have 8: $\forall nell \in lset (?n2l\ nells). \text{lfinitel nell}$

using assms(3) nfinite-def **by** auto

have 9: llast (lfusecat (?n2l nells)) = llast (llast (?n2l nells))

using 4 5 6 7 8 llastfirst-lfusecat-llast **by** blast

have 10: llast (llast (?n2l nells)) = nlast (nlast nells)

by (metis assms(1) assms(3) comp-apply llast-lmap llist-of-nellist-not-lnull nfinite-def
 nlast-llast nset-nlast)

show ?thesis **using** 1 2 3 9 10 **by** presburger

qed

lemma nfusecat-nlength-nfinite:

assumes nfinite nells

$\forall nell \in nset\ nells. \ nfinite\ nell$
 $nlastnfirst\ nells$
shows $nlength(nfusecat\ nells) =$
 $(\sum i = 0 \dots (the-enat((nlength\ nells))) \cdot (nlength\ (nnth\ nells\ i)))$
proof –
let $?n2l = (lmap\ llist-of-nellist \circ llist-of-nellist)$
have 1: $nlength(nfusecat\ nells) = nlength\ (((nellist-of-llist \circ lfusecat \circ ?n2l)\ nells))$
by (*simp add: nfusecat-def2*)
have 2: $nlength\ (((nellist-of-llist \circ lfusecat \circ ?n2l)\ nells)) =$
 $epred\ (llength\ (llist-of-nellist\ (((nellist-of-llist \circ lfusecat \circ ?n2l)\ nells))))$
using $nlength.rep-eq$ **by** *blast*
have 3: $epred\ (llength\ (llist-of-nellist\ (((nellist-of-llist \circ lfusecat \circ ?n2l)\ nells)))) =$
 $epred\ (llength\ (lfusecat\ (?n2l\ nells)))$
using *one-eSuc plus-1-eSuc(2)* **by** *simp*
have 4: $lfinite\ (?n2l\ nells)$
using *assms(1) lfinite-lmap nfinite-def* **by** *auto*
have 5: $\neg lnull\ (?n2l\ nells)$
by *simp*
have 6: $\forall nell \in lset\ (?n2l\ nells). \neg lnull\ nell$
by *simp*
have 7: $\forall nell \in lset\ (?n2l\ nells). \ lfinite\ nell$
using *assms(2) nfinite-def* **by** *auto*
have 8: $llastlfirst\ (?n2l\ nells)$
using *assms(3) nlastnfirst.rep-eq* **by** *auto*
have 9: $epred\ (llength\ (lfusecat\ (?n2l\ nells))) =$
 $epred\ (eSuc(\sum i = 0 \dots (the-enat(epred(llength\ (?n2l\ nells)))) \cdot$
 $epred(llength\ (lnth\ (?n2l\ nells\ i))))$
by (*metis 4 5 6 7 8 lfusecat-llength-lfinite*)
have 10: $epred\ (eSuc(\sum i = 0 \dots (the-enat(epred(llength\ (?n2l\ nells)))) \cdot$
 $epred(llength\ (lnth\ (?n2l\ nells\ i)))) =$
 $((\sum i = 0 \dots (the-enat(epred(llength\ (?n2l\ nells)))) \cdot$
 $epred(llength\ (lnth\ (?n2l\ nells\ i))))$
using *epred-eSuc* **by** *blast*
have 11: $((epred(llength\ (?n2l\ nells)))) = (nlength\ nells)$
by *simp*
have 12: $\bigwedge j. j \leq ((epred(llength\ (?n2l\ nells)))) \longrightarrow$
 $epred(llength\ (lnth\ (?n2l\ nells\ j))) = nlength\ (nnth\ nells\ j)$
by (*metis 5 co.enat.exhaust-sel comp-apply iless-Suc-eq llength-eq-0 llength-llist-of-nellist*
 $llength-lmap\ lnth-llist-of-nellist\ lnth-lmap\ min-def\ the-enat.simps$)
have 13: $((\sum i = 0 \dots (the-enat(epred(llength\ (?n2l\ nells)))) \cdot$
 $epred(llength\ (lnth\ (?n2l\ nells\ i)))) =$
 $((\sum i = 0 \dots (the-enat(nlength\ nells)) \cdot nlength\ (nnth\ nells\ i)))$
using 12 *assms(1) nfinite-nlength-enat* **by** *force*
show *?thesis* **using** 1 2 3 9 10 13 **by** *metis*
qed

lemma *nlastnfirst-ntaken*:
assumes $n \leq nlength\ nells$
 $nlastnfirst\ nells$
shows $nlastnfirst\ (ntaken\ n\ nells)$

```

using assms
by (metis Suc-ile-eq antisym-conv2 nappend-ntaken-ndropn nfinite-ntaken nlastnfirst-nappend-nfinite
     ntaken-all)

```

```

lemma nlastnfirst-ntake:
  assumes  $n \leq \text{nlength } \text{nells}$ 
           nlastnfirst nells
  shows nlastnfirst (ntake n nells)
proof (cases n)
case (enat nat)
then show ?thesis by (metis assms nlastnfirst-ntaken ntake-eq-ntaken)
next
case infinity
then show ?thesis
by (simp add: assms ntake-all)
qed

```

```

lemma nfusecat-ntake:
  assumes  $(\text{enat } n) \leq \text{nlength } \text{nells}$ 
            $\forall \text{ nell} \in \text{nset } \text{nells}. \text{nfinite } \text{nell}$ 
           nlastnfirst nells
  shows nfusecat (ntake n nells) =
           ntake ( (( $\sum i = 0 .. n . (\text{nlength } (\text{nnth } \text{nells } i)))$ )
             (nfusecat nells))
proof –
  let ?n2l = (lmap llist-of-nellist  $\circ$  llist-of-nellist)
  have 1: nfusecat (ntake n nells) =
           ((nlist-of-llist  $\circ$  lfusecat  $\circ$  ?n2l) (ntake n nells))
    by (simp add: nfusecat-def2)
  have 2: ((nlist-of-llist  $\circ$  lfusecat  $\circ$  ?n2l) (ntake n nells)) =
           nlist-of-llist (lfusecat (?n2l (ntake n nells)))
    by simp
  have 3: llist-of-nellist (ntake n nells) = ltake (eSuc n) (llist-of-nellist nells)
    by (metis co.enat.distinct(1) llist-of-nellist-inverse-b llist-of-nellist-not-lnull
        ltake.disc(2) nlist-of-llist-inverse ntake.abs-eq)
  have 4: (?n2l (ntake n nells)) = ltake (eSuc n) (?n2l nells)
    using 3 by auto
  have 5:  $\neg \text{lnull } (?n2l \text{ nells})$ 
    by simp
  have 6:  $(\text{Suc } n) \leq \text{llength } (?n2l \text{ nells})$ 
by (metis 5 Suc-ile-eq assms(1) co.enat.collapse comp-apply iless-Suc-eq llength-eq-0 llength-lmap
     nlength.rep-eq)
  have 7:  $\forall \text{ nell} \in \text{lset } (?n2l \text{ nells}). \neg \text{lnull } \text{nell}$ 
    by simp
  have 8:  $\forall \text{ nell} \in \text{lset } (?n2l \text{ nells}). \text{lfinite } \text{nell}$ 
    using assms(2) nfinite-def by fastforce

```

have 9: $llastlfirst (?n2l\ nells)$
using $assms(3)\ nlastnfirst.rep-eq$ **by** $auto$
have 10: $(lfusecat\ (ltake\ (eSuc\ n)\ (?n2l\ nells))) =$
 $ltake\ (if\ (Suc\ n) = 0\ then\ 0\ else\ eSuc(\sum\ i = 0 .. n .$
 $epred(llength\ (lnth\ (?n2l\ nells)\ i))))$
 $(lfusecat\ (?n2l\ nells))$
using $lfusecat-ltake[of\ (?n2l\ nells)\ Suc\ n\]$
 $5\ 6\ 7\ 8\ 9\ diff-Suc-1\ eSuc-enat$ **by** $presburger$
have 11: $(if\ (Suc\ n) = 0\ then\ 0\ else\ eSuc(\sum\ i = 0 .. n .$
 $epred(llength\ (lnth\ (?n2l\ nells)\ i)))) =$
 $(eSuc(\sum\ i = 0 .. n .\ epred(llength\ (lnth\ (?n2l\ nells)\ i))))$
using $nat.simps(3)$ **by** $presburger$
have 111: $\bigwedge j. j \leq n \longrightarrow (min\ (enat\ j)\ (epred\ (llength\ (llist-of-nellist\ nells)))) = j$
by $(metis\ assms(1)\ enat-ord-simps(1)\ llength-llist-of-nellist\ min.bounded-iff\ min-def)$
have 12: $\bigwedge j. j \leq n \longrightarrow$
 $epred(llength\ (lnth\ (?n2l\ nells)\ j)) = nlength\ (nnth\ nells\ j)$
using $5\ 111\ assms\ lnth-llist-of-nellist[of\ nells]$
by $(metis\ (full-types)\ co.enat.exhaust-sel\ iless-Suc-eq\ llength-eq-0\ llength-lmap\ lnth-lmap$
 $min.order-iff\ nlength.rep-eq\ o-apply\ the-enat.simps)$
have 13: $(lfusecat\ (?n2l\ nells)) = llist-of-nellist\ (nfusecat\ nells)$
by $(simp\ add:\ lfusecat-not-lnull-var\ nfusecat-def2)$
have 14: $ltake\ (if\ (Suc\ n) = 0\ then\ 0\ else\ eSuc(\sum\ i = 0 .. n .$
 $epred(llength\ (lnth\ (?n2l\ nells)\ i))))$
 $(lfusecat\ (?n2l\ nells)) =$
 $ltake\ (eSuc(\sum\ i = 0 .. n .\ nlength\ (nnth\ nells\ i)))\ (llist-of-nellist\ (nfusecat\ nells))$
using $12\ 13$ **by** $auto$
have 15: $nellist-of-llist\ (ltake\ (eSuc(\sum\ i = 0 .. n .$
 $nlength\ (nnth\ nells\ i)))\ (llist-of-nellist\ (nfusecat\ nells))) =$
 $ntake\ ((\sum\ i = 0 .. n .\ nlength\ (nnth\ nells\ i)))\ (nfusecat\ nells)$
by $(metis\ llist-of-nellist-inverse-b\ llist-of-nellist-not-lnull\ ntake.abs-eq)$
show $?thesis$
by $(metis\ 10\ 14\ 15\ 2\ 4\ nfusecat-def2)$
qed

lemma $nfusecat-ndropn$:

assumes $n < nlength\ nells$

$\forall\ nell \in nset\ nells. nfinite\ nell$

$nlastnfirst\ nells$

shows $nfusecat\ (ndropn\ n\ nells) =$

$ndropn\ (the-enat(if\ n = 0\ then\ 0\ else\ (\sum\ i = 0 .. (n-1) . nlength(nnth\ nells\ i))))$
 $(nfusecat\ nells)$

proof –

let $?n2l = (lmap\ llist-of-nellist\ o\ llist-of-nellist)$

have 1: $nfusecat\ (ndropn\ n\ nells) =$

$((nlist-of-llist\ o\ lfusecat\ o\ ?n2l)\ (ndropn\ n\ nells))$

by $(simp\ add:\ nfusecat-def2)$

have 2: $((nlist-of-llist\ o\ lfusecat\ o\ ?n2l)\ (ndropn\ n\ nells)) =$

$nlist-of-llist\ (lfusecat\ (lmap\ llist-of-nellist\ (ldropn\ n\ (llist-of-nellist\ nells))))$

using $assms$ **by** $simp$

have 4: $nlist-of-llist\ (lfusecat\ (lmap\ llist-of-nellist\ (ldropn\ n\ (llist-of-nellist\ nells)))) =$

$nellist\text{-}of\text{-}l\text{-}list$ ($lfusecat$ ($lmap$ $l\text{-}list\text{-}of\text{-}nellist$ ($ldrop$ n ($l\text{-}list\text{-}of\text{-}nellist$ $nells$))))
by (*simp* *add: ldrops-enat*)
have 5: ($lmap$ $l\text{-}list\text{-}of\text{-}nellist$ ($ldrop$ n ($l\text{-}list\text{-}of\text{-}nellist$ $nells$))) = $ldrop$ n ($?n2l$ $nells$)
by (*simp*)
have 6: $\neg lnull$ ($?n2l$ $nells$)
by *simp*
have 7: $n < llength$ ($?n2l$ $nells$)
by (*metis* 5 *assms*(1) *ldrops-enat* $l\text{-}list.map\text{-}disc\text{-}iff$ $l\text{-}list\text{-}of\text{-}nellist\text{-}ndropn$ $l\text{-}list\text{-}of\text{-}nellist\text{-}not\text{-}lnull$ $lnull\text{-}ldrop$ $min\text{-}def$ $nlength.rep\text{-}eq$ $not\text{-}le\text{-}imp\text{-}less$ $order\text{-}less\text{-}imp\text{-}le$ *the-enat.simps*)
have 8: $\forall nell \in lset$ ($?n2l$ $nells$). $\neg lnull$ $nell$
by *simp*
have 9: $\forall nell \in lset$ ($?n2l$ $nells$). $lfinite$ $nell$
using *assms*(2) *nfinite-def* **by** *fastforce*
have 10: $llastlfirst$ ($?n2l$ $nells$)
using *assms*(3) *nlastnfirst.rep-eq* **by** *auto*
have 11: $lfusecat$ ($ldrop$ ($enat$ n) ($?n2l$ $nells$)) =
 $ldrop$ (*if* $n = 0$ *then* 0 *else* ($\sum i = 0 .. (n-1)$.
 $epred(llength$ ($lnth$ ($?n2l$ $nells$) i)))
 $(lfusecat$ ($?n2l$ $nells$))
using 10 6 7 8 9 *lfusecat-ldrop* **by** *blast*
have 111: $\bigwedge j. 0 < j \wedge j < n-1 \longrightarrow$
 $(min$ ($enat$ j) ($epred$ ($llength$ ($l\text{-}list\text{-}of\text{-}nellist$ $nells$)))) = j
by (*metis* 7 *comp-apply* *epred-enat* *epred-min* *less-imp-of-nat-less* *llength-lmap* *min.absorb3* *min-less-iff-conj* *of-nat-eq-enat*)
have 112: $n-1 < llength$ ($?n2l$ $nells$)
by (*metis* 7 *One-nat-def* *Suc-ile-eq* *Suc-pred* *epred-0* *epred-enat* *not-gr-zero* *order-less-imp-le* *zero-enat-def*)

have 12: $\bigwedge j. 0 < n \wedge j < n-1 \longrightarrow$
 $epred(llength$ ($lnth$ ($?n2l$ $nells$) j)) = $nlength(nnth$ $nells$ j)
using 6 7 111 *lnth-llist-of-nellist[of nells]*
by *simp*
(*metis* *canonically-ordered-monoid-add-class.lessE* *diff-le-self* *dual-order.strict-trans1* *enat-less-enat-plusI* *enat-min-eq-0-iff* *lnth-lmap* *nlength.rep-eq* *not-gr-zero* *the-enat.simps* *zero-enat-def*)
have 13: ($lfusecat$ ($?n2l$ $nells$)) = $l\text{-}list\text{-}of\text{-}nellist$ ($nfusecat$ $nells$)
by (*simp* *add: lfusecat-not-lnull-var nfusecat-def2*)
have 131: $\bigwedge j. j < n \implies (min$ ($enat$ j) ($epred$ ($llength$ ($l\text{-}list\text{-}of\text{-}nellist$ $nells$)))) = j
by (*metis* (*full-types*) *assms*(1) *min.assoc* *min.orderE* *min-enat-simps*(1) *nlength.rep-eq* *order-less-imp-le*)
have 132: (*if* $n = 0$ *then* 0 *else* $\sum i = 0..n-1. epred$ ($llength$ ($lnth$ ($?n2l$ $nells$) i))) =
(*if* $n = 0$ *then* 0 *else* ($\sum i = 0 .. (n-1)$. $nlength(nnth$ $nells$ i)))
using *sum.cong[of {0..n-1} {0..n-1} $\lambda i. epred$ ($llength$ ($lnth$ ($?n2l$ $nells$) i))*
 $\lambda i. nlength(nnth$ $nells$ i)] *assms* 7 131 12 *lnth-llist-of-nellist[of nells]*
by (*metis* 112 *atLeastAtMost-iff* *comp-apply* *diff-less* *less-numeral-extra*(1) *llength-lmap* *lnth-lmap* *nlength.rep-eq* *not-gr-zero* *order-neq-le-trans* *the-enat.simps*)
have 14: $ldrop$ (*if* $n = 0$ *then* 0 *else* ($\sum i = 0 .. (n-1)$. $epred(llength$ ($lnth$ ($?n2l$ $nells$) i)))
 $(lfusecat$ ($?n2l$ $nells$)) =
 $ldrop$ (*if* $n = 0$ *then* 0 *else* ($\sum i = 0 .. (n-1)$. $nlength(nnth$ $nells$ i))) ($l\text{-}list\text{-}of\text{-}nellist$ ($nfusecat$ $nells$))
using 13 132 **by** *presburger*

```

have 15:  $0 < n \implies (\bigwedge j. j \leq n-1 \implies \text{epred}(\text{llength}(\text{lnth}(\text{?n2l nells } j))) < \infty)$ 
  using 7 9
  by (metis (no-types, opaque-lifting) 112 enat-ord-simps(1) enat-ord-simps(4) epred-llength
    in-lset-conv-lnth lfinite-ltl llength-eq-infty-conv-lfinite order-le-less-trans)
have 16:  $0 < n \implies ((\sum i = 0 .. (n-1) . \text{epred}(\text{llength}(\text{lnth}(\text{?n2l nells } i)))) < \infty$ 
  using 15
  proof (induct n)
  case 0
  then show ?case by simp
  next
  case (Suc n)
  then show ?case
    proof (cases n=0)
    case True
    then show ?thesis using Suc by simp
    next
    case False
    then show ?thesis using Suc
    by simp
    (metis One-nat-def Suc-pred' dual-order.refl plus-enat-simps(1) sum.atLeast0-atMost-Suc)
  qed
qed
have 17:  $\exists m. (\text{enat } m) = (\text{if } n = 0 \text{ then } 0 \text{ else } (\sum i = 0 .. (n-1) . \text{epred}(\text{llength}(\text{lnth}(\text{?n2l nells } i))))$ 
  by (metis 16 less-infinityE not-gr-zero zero-enat-def)
obtain m where 18:  $(\text{enat } m) = (\text{if } n = 0 \text{ then } 0 \text{ else } (\sum i = 0 .. (n-1) . \text{epred}(\text{llength}(\text{lnth}(\text{?n2l nells } i))))$ 
  using 17 by blast
have 19:  $\text{ldrop}(\text{if } n = 0 \text{ then } 0 \text{ else } (\sum i = 0 .. (n-1) . \text{epred}(\text{llength}(\text{lnth}(\text{?n2l nells } i))))$ 
   $(\text{lfusecat } (\text{?n2l nells})) =$ 
   $\text{ldropn } m (\text{lfusecat } (\text{?n2l nells}))$ 
  using 18  $\text{ldrop-enat}[of m (\text{lfusecat } (\text{?n2l nells}))]$ 
  by presburger
have 20:  $\text{enat } m \leq \text{epred}(\text{llength}(\text{lfusecat } (\text{?n2l nells})))$ 
  by (metis (no-types, lifting) 11 19 6 7 8 co.enat.exhaust-sel iless-Suc-eq in-lset-ldropD
    lfusecat-not-lnull-var linorder-not-le llength-eq-0 lnull-ldrop lnull-ldropn)
have 21:  $\text{nellist-of-llist}(\text{ldropn } m (\text{lfusecat } (\text{?n2l nells}))) = \text{ndropn } m (\text{nfusecat nells})$ 
  using 20
  by (metis 13 llist-of-nellist-inverse-b llist-of-nellist-not-lnull min-def ndropn.abs-eq
    the-enat.simps)
have 22:  $m = (\text{the-enat}(\text{if } n = 0 \text{ then } 0 \text{ else } (\sum i = 0 .. (n-1) . \text{nlength}(\text{nnth nells } i))))$ 
  by (metis (mono-tags, lifting) 132 18 the-enat.simps)
show ?thesis using 1 2 4 5 11 19 21 22 by (metis)
qed

```

lemma nridx-nfusecat-ntake:

```

assumes nridx R (nfusecat nells)
   $(\text{enat } n) \leq \text{nlength nells}$ 
   $\text{nlastnfirst nells}$ 

```

$\forall \text{ nell} \in \text{nset nells. nfinite nell}$
shows $\text{nridx } R \text{ (nfusecat (ntake n nells))}$
using *assms* **by** (*metis nfusecat-ntake nle-le nridx-ntake-a ntake-all*)

lemma *nridx-nfusecat-ndrop*:
assumes $\text{nridx } R \text{ (nfusecat nells)}$
 $(\text{enat } n) < \text{nlength nells}$
 nlastnfirst nells
 $\forall \text{ nell} \in \text{nset nells. nfinite nell}$
shows $\text{nridx } R \text{ (nfusecat (ndropn n nells))}$
proof –
let $?n2l = (\text{lmap llist-of-nellist} \circ \text{llist-of-nellist})$
have 1: $\text{nridx } R \text{ (nfusecat (ndropn n nells))} \longleftrightarrow$
 $\text{nridx } R \text{ ((nellist-of-llist} \circ \text{lfusecat} \circ ?n2l) \text{ (ndropn n nells))}$
by (*simp add: nfusecat-def2*)
have 2: $\text{nridx } R \text{ ((nellist-of-llist} \circ \text{lfusecat} \circ ?n2l) \text{ (ndropn n nells))} \longleftrightarrow$
 $\text{ridx } R \text{ (llist-of-nellist ((nellist-of-llist} \circ \text{lfusecat} \circ ?n2l) \text{ (ndropn n nells)))}$
using *nridx.rep-eq* **by** *blast*
have 3: $\text{ridx } R \text{ (llist-of-nellist ((nellist-of-llist} \circ \text{lfusecat} \circ ?n2l) \text{ (ndropn n nells)))} \longleftrightarrow$
 $\text{ridx } R \text{ ((lfusecat} \circ ?n2l) \text{ (ndropn n nells))}$
using *not-null-lfusecat*
by (*metis (no-types, lifting) comp-apply nridx.abs-eq nridx.rep-eq*)
have 4: $\text{llastlfirst} (?n2l \text{ nells})$
using *assms nlastnfirst.rep-eq* **by** *auto*
have 5: $\forall \text{ nell} \in \text{lset} (?n2l \text{ nells}). \text{lfinite nell}$
using *assms nfinite-def* **by** *fastforce*
have 6: $((\text{lfusecat} \circ ?n2l) \text{ (ndropn n nells))) =$
 $\text{lfusecat} (\text{lmap llist-of-nellist} (\text{ldropn } n \text{ (llist-of-nellist nells)}))$
by (*simp add: assms(2)*)
have 7: $(\text{lmap llist-of-nellist} (\text{ldropn } n \text{ (llist-of-nellist nells)})) =$
 $\text{ldropn } n \text{ (lmap llist-of-nellist (llist-of-nellist nells))}$
by *auto*
have 8: $\text{ridx } R \text{ (lfusecat (?n2l nells))}$
by (*metis (no-types, lifting) assms(1) comp-apply nfusecat-def2 nridx.abs-eq ridx-expand-1*)
have 9: $\text{enat } n < \text{llength} (?n2l \text{ nells})$
by (*metis assms(2) co.enat.exhaust-sel comp-apply iless-Suc-eq llength-eq-0 llength-lmap*
 $\text{llist-of-nellist-not-lnull nlength.rep-eq order-less-imp-le}$)
have 10: $\text{ridx } R \text{ (lfusecat (ldropn } n \text{ (?n2l nells))}$
using *ridx-lfusecat-ldrop[of R (?n2l nells) n]*
by (*metis (mono-tags, lifting) 4 5 8 9 comp-apply in-lset-conv-lnth ldrop-enat llength-lmap*
 $\text{llist-of-nellist-not-lnull lnth-lmap}$)
show *?thesis*
using 1 10 2 3 6 7 **by** (*metis comp-apply*)
qed

lemma *nridx-nfusecat*:
assumes nlastnfirst nells
 $\forall \text{ nell} \in \text{nset nells. } 0 < \text{nlength nell}$
 $\forall \text{ nell} \in \text{nset nells. nfinite nell}$

shows $\text{nridx } R \text{ (nfusecat nells)} \longleftrightarrow (\forall i. i \leq \text{nlength nells} \longrightarrow \text{nridx } R \text{ (nnth nells } i))$
proof –
let $?n2l = (\text{lmap llist-of-nellist} \circ \text{llist-of-nellist})$
have 1: $\text{nridx } R \text{ (nfusecat nells)} \longleftrightarrow$
 $\text{nridx } R \text{ ((nellist-of-llist} \circ \text{lfusecat} \circ ?n2l) \text{ nells)}$
by (*simp add: nfusecat-def*)
have 2: $\text{nridx } R \text{ ((nellist-of-llist} \circ \text{lfusecat} \circ ?n2l) \text{ nells)} \longleftrightarrow$
 $\text{ridx } R \text{ (llist-of-nellist ((nellist-of-llist} \circ \text{lfusecat} \circ ?n2l) \text{ nells))}$
using *nridx.rep-eq* **by** *blast*
have 3: $\text{ridx } R \text{ (llist-of-nellist ((nellist-of-llist} \circ \text{lfusecat} \circ ?n2l) \text{ nells))} \longleftrightarrow$
 $\text{ridx } R \text{ ((lfusecat} \circ ?n2l) \text{ nells)}$
using *not-null-lfusecat nridx.abs-eq* **by** *fastforce*
have 4: $\text{llastlfirst} (?n2l \text{ nells})$
using *assms(1) nlastnfirst.rep-eq* **by** *auto*
have 5: $\forall \text{ nell} \in \text{lset} (?n2l \text{ nells}). 1 < \text{llength nell}$
by (*metis assms(2) epred-1 comp-apply imageE less-numeral-extra(3) llist.set-map*
 $\text{llist-of-nellist-not-lnull lset-llist-of-nellist-a nlength.rep-eq not-lnull-llength}$
 $\text{order-neq-le-trans}$)
have 6: $\forall \text{ nell} \in \text{lset} (?n2l \text{ nells}). \text{lfinite nell}$
using *assms(3) nfinite-def* **by** *fastforce*
have 7: $\text{ridx } R \text{ ((lfusecat} \circ ?n2l) \text{ nells)} \longleftrightarrow$
 $(\forall i. i < \text{llength} (?n2l \text{ nells}) \longrightarrow \text{ridx } R \text{ (lnth} (?n2l \text{ nells) } i))$
using 4 5 6 *ridx-lfusecat* **by** *fastforce*
have 8: $\text{epred}(\text{llength} (?n2l \text{ nells})) = \text{nlength nells}$
by *simp*
have 9: $\bigwedge j. j < \text{llength} (?n2l \text{ nells}) \longrightarrow$
 $(\text{lnth} (?n2l \text{ nells) } j) = \text{llist-of-nellist}(\text{nnth nells } j)$
by *simp*
 $(\text{metis eSuc-epred eSuc-ile-mono gr-implies-not-zero ileI1 lnth-llist-of-nellist}$
 $\text{min-def the-enat.simps})$
have 10: $(\forall i. i < \text{llength} (?n2l \text{ nells}) \longrightarrow \text{ridx } R \text{ (lnth} (?n2l \text{ nells) } i)) \longleftrightarrow$
 $(\forall i. i \leq \text{nlength} (\text{nells}) \longrightarrow \text{nridx } R \text{ (nnth nells } i))$
by (*metis (mono-tags, lifting) 8 9 co.enat.exhaust-sel comp-apply iless-Suc-eq*
 $\text{llength-eq-0 llist.map-disc-iff llist-of-nellist-not-lnull nridx.rep-eq}$)
show *?thesis* **using** 1 2 3 10 7 **by** *blast*
qed

lemma *nfusecat-split-nsubn*:

assumes *nlastnfirst nells*

$\forall \text{ nell} \in \text{nset nells}. 0 < \text{nlength nell}$

$\forall \text{ nell} \in \text{nset nells}. \text{nfinite nell}$

shows $(\forall i. i \leq \text{nlength nells} \longrightarrow \text{nridx} (\lambda a b. f (\text{nsubn } \sigma a b)) (\text{nnth nells } i)) \longleftrightarrow$
 $\text{nridx} (\lambda a b. f (\text{nsubn } \sigma a b)) (\text{nfusecat nells})$

using *assms nridx-nfusecat* **by** *blast*

lemma *nidx-nfusecat*:

assumes *nlastnfirst nells*

$\forall \text{ nell} \in \text{nset nells}. 0 < \text{nlength nell}$

$\forall \text{ nell} \in \text{nset nells}. \text{nfinite nell}$

shows $\text{nidx} (\text{nfusecat nells}) \longleftrightarrow (\forall i. i \leq \text{nlength nells} \longrightarrow \text{nidx} (\text{nnth nells } i))$

using *assms nridx-nfusecat nridx-nidx* **by** *blast*

lemma *nnth-sum-expand*:

assumes *nlastnfirst nells*

$\forall \text{ nell} \in \text{nset nells. } 0 < \text{nlength nell}$

$\forall \text{ nell} \in \text{nset nells. nfinite nell}$

$(\text{enat } i) \leq \text{nlength nells}$

nfinite nells

shows $(\sum i = 0 .. (\text{the-enat}((\text{nlength nells}))) . (\text{nlength } (\text{nnth nells } i))) =$
 $(\text{nlength } (\text{nnth nells } i)) +$
 $(\sum j \in \{k. k \neq i \wedge k \leq (\text{the-enat}((\text{nlength nells})))\} . (\text{nlength } (\text{nnth nells } j)))$

proof –

have 1: $\{k. k \leq (\text{the-enat}((\text{nlength nells})))\} = \text{insert } i \{k. k \neq i \wedge k \leq (\text{the-enat}((\text{nlength nells})))\}$

using *assms* **by** *auto*

(metis enat-ord-simps(1) nfinite-nlength-enat the-enat.simps)

have 2: $\{0.. (\text{the-enat}((\text{nlength nells})))\} = \{k. k \leq (\text{the-enat}((\text{nlength nells})))\}$

by *auto*

have 3: $(\sum i = 0 .. (\text{the-enat}((\text{nlength nells}))) . (\text{nlength } (\text{nnth nells } i))) =$
 $(\sum i \in \{k. k \leq (\text{the-enat}((\text{nlength nells})))\} . (\text{nlength } (\text{nnth nells } i)))$

using 2 **by** *presburger*

have 4: $(\sum i \in \{k. k \leq (\text{the-enat}((\text{nlength nells})))\} . (\text{nlength } (\text{nnth nells } i))) =$
 $(\sum j \in (\text{insert } i \{k. k \neq i \wedge k \leq (\text{the-enat}((\text{nlength nells})))\}) . (\text{nlength } (\text{nnth nells } j)))$

using 1 **by** *presburger*

have 5: $(\sum j \in (\text{insert } i \{k. k \neq i \wedge k \leq (\text{the-enat}((\text{nlength nells})))\}) . (\text{nlength } (\text{nnth nells } j))) =$
 $(\text{nlength } (\text{nnth nells } i)) +$

$(\sum j \in \{k. k \neq i \wedge k \leq (\text{the-enat}((\text{nlength nells})))\} . (\text{nlength } (\text{nnth nells } j)))$

by *auto*

show *?thesis*

using 3 4 5 **by** *presburger*

qed

lemma *nfusecat-nlength-a*:

assumes *nlastnfirst nells*

$\forall \text{ nell} \in \text{nset nells. } 0 < \text{nlength nell}$

$\forall \text{ nell} \in \text{nset nells. nfinite nell}$

$i \leq \text{nlength nells}$

$(\text{enat } j) \leq \text{nlength } (\text{nnth nells } i)$

shows $(\text{enat } j) \leq \text{nlength } (\text{nfusecat nells})$

proof (*cases nfinite nells*)

case *True*

then show *?thesis*

proof –

have 2: $\text{nlength } (\text{nfusecat nells}) =$
 $(\sum i = 0 .. (\text{the-enat}((\text{nlength nells}))) . (\text{nlength } (\text{nnth nells } i)))$

using *True assms nfusecat-nlength-nfinite* **by** *fastforce*

have 3: $\text{nlength } (\text{nnth nells } i) \leq$
 $(\sum i = 0 .. (\text{the-enat}((\text{nlength nells}))) . (\text{nlength } (\text{nnth nells } i)))$

using *nnth-sum-expand[of nells i] assms*

by (*metis (no-types, lifting) True le-iff-add*)

show *?thesis* **using** *assms* 2 3 **by** *force*


```

qed
next
case False
then show ?thesis
  proof -
    have 5:  $\neg$  nfinite (nfusecat nells)
      using False assms nfusecat-nfinite by blast
    show ?thesis
      using 5
      by (simp add: nfinite-conv-nlength-enat)
  qed
qed

```

1.3.9 nkfilter

```

lemma nkfilter-def1:
  assumes  $(\exists x \in \text{nset } nell. P x)$ 
  shows  $\text{nkfilter } P n nell = \text{nmap snd } (\text{nfilter } (P \circ \text{fst}) (\text{nzip } nell (\text{niterates } \text{Suc } n)))$ 
using assms
apply transfer
by (simp add: kfilter-def)
  (metis imageE llist.set-map lmap-fst-lzip)

```

```

lemma nmap-fst-nzip:
   $(\text{nmap fst } (\text{nzip } nell (\text{niterates } \text{Suc } n))) = nell$ 
apply transfer
by (simp add: lmap-fst-lzip)

```

```

lemma nfilter-nkfilter-1:
  assumes  $(\exists x \in \text{nset } nell. P x)$ 
  shows  $(\text{nmap fst } (\text{nfilter } (P \circ \text{fst}) (\text{nzip } nell (\text{niterates } \text{Suc } n)))) =$ 
     $(\text{nfilter } P nell)$ 
using assms
by (metis nfilter-nmap nmap-fst-nzip)

```

```

lemma nkfilter-NNil [simp]:
  shows  $P b \implies \text{nkfilter } P n (\text{NNil } b) = (\text{NNil } n)$ 
by transfer auto

```

```

lemma nkfilter-NCons [simp]:
  assumes  $(\exists x \in \text{nset } nell. P x)$ 
  shows  $\text{nkfilter } P n (\text{NCons } x nell) =$ 
     $(\text{if } P x \text{ then } \text{NCons } n (\text{nkfilter } P (\text{Suc } n) nell) \text{ else } \text{nkfilter } P (\text{Suc } n) nell)$ 
using assms
by transfer
  (auto simp add: kfilter-lnull-conv)

```

```

lemma nkfilter-NCons-a [simp]:
  assumes  $\neg(\exists x \in \text{nset } nell. P x)$ 

```

$P\ x$
shows $nkfilter\ P\ n\ (NCons\ x\ nell) = (NNil\ n)$
using *assms*
by *transfer*
 $(auto\ simp\ add: kfilter-lnull-conv)$

lemma *nkfilter-nlength*:
assumes $\exists\ x \in nset\ nell. P\ x$
shows $nlength(nkfilter\ P\ n\ nell) = nlength(nfilter\ P\ nell)$
using *assms*
by *transfer*
 $(auto\ simp\ add: kfilter-lnull-conv\ kfilter-llength)$

lemma *nkfilter-upperbound*:
assumes $\exists\ x \in nset\ nell. P\ x$
 $i \leq nlength\ (nkfilter\ P\ n\ nell)$
shows $nnth\ (nkfilter\ P\ n\ nell)\ i \leq n + nlength\ nell$
using *assms*
by *transfer*
 $(auto,$
 $\quad simp\ add: kfilter-lnull-conv,$
 $\quadmetis\ co.enat.exhaust-sel\ iadd-Suc-right\ iless-Suc-eq\ kfilter-upperbound\ llength-eq-0)$

lemma *nkfilter-lowerbound*:
assumes $\exists\ x \in nset\ nell. P\ x$
 $i \leq nlength\ (nkfilter\ P\ n\ nell)$
shows $n \leq nnth\ (nkfilter\ P\ n\ nell)\ i$
using *assms*
by *transfer*
 $(auto,$
 $\quadmetis\ co.enat.collapse\ iless-Suc-eq\ kfilter-lnth-n-zero-a\ llength-eq-0)$

lemma *nkfilter-mono*:
assumes $\exists\ x \in nset\ nell. P\ x$
 $(Suc\ i) \leq nlength\ (nkfilter\ P\ n\ nell)$
shows $nnth\ (nkfilter\ P\ n\ nell)\ i < nnth\ (nkfilter\ P\ n\ nell)\ (Suc\ i)$
using *assms*
by *transfer*
 $(auto,$
 $\quadmetis\ diff-Suc-1\ eSuc-epred\ epred-enat\ epred-le-epredI\ lidk-filter-gr\ iless-Suc-eq\ leD\ lessI$
 $\quad llength-eq-0\ min.cobounded1\ min-def\ the-enat.simps)$

lemma *nkfilter-nfilter*:
assumes $\exists\ x \in nset\ nell. P\ x$
 $i \leq nlength\ (nkfilter\ P\ n\ nell)$
shows $(nnth\ nell\ ((nnth\ (nkfilter\ P\ n\ nell)\ i) - n)) = (nnth\ (nfilter\ P\ nell)\ i)$
using *assms*
apply *transfer*
proof $(auto\ simp\ add: kfilter-lnull-conv\ split:if-split-asm)$
fix *nella* :: 'a llist **and** *Pa* :: 'a \Rightarrow bool **and** *ia* :: nat **and** *na* :: nat **and** *x* :: 'a

```

assume a1:  $x \in \text{lset nella}$ 
assume a2:  $\text{Pa } x$ 
assume a3:  $\text{enat } ia \leq \text{epred } (\text{llength } (\text{kfilter } \text{Pa } na \text{ nella}))$ 
assume a4:  $\neg \text{lnull nella}$ 
have f5:  $\forall e \text{ ea. } \neg (e::\text{enat}) \leq \text{ea} \vee \text{min } e \text{ ea} = e$ 
by (simp add: min-def-raw)
have f6:  $\forall n \text{ p } na \text{ as. } \neg \text{enat } n < \text{llength } (\text{kfilter } p \text{ na } (as::'a \text{ llist})) \vee$ 
 $\text{enat } (\text{lnth } (\text{kfilter } p \text{ na } as) \text{ n}) < \text{enat } na + \text{llength } as$ 
by (meson kfilter-upperbound)
have f7:  $\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{kfilter } \text{Pa } na \text{ nella}))) = \text{enat } ia$ 
using f5 a3 by blast
then have f8:  $(\text{enat } ia < \text{llength } (\text{kfilter } \text{Pa } 0 \text{ nella})) =$ 
 $(\text{enat } (\text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{kfilter } \text{Pa } na \text{ nella})))))) <$ 
 $\text{llength } (\text{kfilter } \text{Pa } na \text{ nella}))$ 
by (simp add: kfilter-llength)
have f9:  $\forall n \text{ p } na \text{ as. } \neg \text{enat } n < \text{llength } (\text{kfilter } p \text{ na } (as::'a \text{ llist})) \vee$ 
 $\text{lnth } (\text{kfilter } p \text{ na } as) \text{ n} - na = \text{lnth } (\text{kfilter } p \text{ 0 } as) \text{ n}$ 
using kfilter-lnth-n-zero by blast
have eSuc  $(\text{epred } (\text{llength } nella)) = \text{enat } 0 + \text{llength } nella$ 
using a4 by (metis co.enat.exhaust-sel gen-llength-def llength-code llength-eq-0)
then have enat  $(\text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{kfilter } \text{Pa } na \text{ nella})))))) <$ 
 $\text{llength } (\text{kfilter } \text{Pa } na \text{ nella}) \longrightarrow$ 
 $\text{enat } (\text{lnth } (\text{kfilter } \text{Pa } na \text{ nella}))$ 
 $(\text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{kfilter } \text{Pa } na \text{ nella})))))) - na)$ 
 $< \text{eSuc } (\text{epred } (\text{llength } nella))$ 
using f9 f8 f7 f6 the-enat.simps by presburger
then have f10: enat  $(\text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{kfilter } \text{Pa } na \text{ nella})))))) <$ 
 $\text{llength } (\text{kfilter } \text{Pa } na \text{ nella}) \longrightarrow$ 
 $\text{enat } (\text{lnth } (\text{kfilter } \text{Pa } na \text{ nella}))$ 
 $(\text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{kfilter } \text{Pa } na \text{ nella})))))) - na)$ 
 $\leq \text{epred } (\text{llength } nella)$ 
using iless-Suc-eq by blast
have f11:  $\forall n \text{ p } na \text{ as. } \neg \text{enat } n < \text{llength } (\text{kfilter } p \text{ na } as) \vee \text{lnth } as (\text{lnth } (\text{kfilter } p \text{ na } as) \text{ n} - na) =$ 
 $(\text{lnth } (\text{lfilter } p \text{ as}) \text{ n}::'a)$ 
using lfilter-kfilter by blast
have f12:  $\text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{lfilter } \text{Pa } nella)))) =$ 
 $\text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{kfilter } \text{Pa } na \text{ nella}))))$ 
by (simp add: kfilter-llength)
have  $\neg \text{lnull } (\text{kfilter } \text{Pa } na \text{ nella})$ 
using a1 a2 kfilter-not-lnull-conv by auto
then have enat  $(\text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{kfilter } \text{Pa } na \text{ nella})))))) <$ 
 $\text{llength } (\text{kfilter } \text{Pa } na \text{ nella})$ 
using f7 a3 by (metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0 the-enat.simps)
then show  $\neg \text{lnull } nella \Longrightarrow$ 
 $\text{enat } ia \leq \text{epred } (\text{llength } (\text{kfilter } \text{Pa } na \text{ nella})) \Longrightarrow$ 
 $x \in \text{lset nella} \Longrightarrow$ 
 $\text{Pa } x \Longrightarrow$ 
 $\text{lnth } nella (\text{the-enat } (\text{min } (\text{enat } (\text{lnth } (\text{kfilter } \text{Pa } na \text{ nella}) \text{ ia} - na)) (\text{epred } (\text{llength } nella)))) =$ 
 $\text{lnth } (\text{lfilter } \text{Pa } nella) (\text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{lfilter } \text{Pa } nella))))))$ 
using f12 f11 f10 f5 by simp

```

qed

lemma *in-nkfilter-nset*:

assumes $\exists x \in \text{nset } nell. P x$

shows $x \in \text{nset}(\text{nkfilter } P n nell) \longleftrightarrow x \in \{ n+i \mid i. i \leq \text{nlength } nell \wedge P (\text{nnth } nell i) \}$

using *assms*

by *transfer*

(*auto simp add: kfilter-lnull-conv min-def,*
metis co.enat.exhaust-sel gen-llength-def iless-Suc-eq kfilter-holds kfilter-llength-n-zero
kfilter-lnth-n-zero kfilter-lowerbound kfilter-upperbound ldistinct-Ex1 ldistinct-kfilter
le-add-diff-inverse llength-code llength-eq-0,
metis add commute co.enat.exhaust-sel iless-Suc-eq kfilter-holds-not-a llength-eq-0)

lemma *nkfilter-nset*:

assumes $\exists x \in \text{nset } nell. P x$

shows $\text{nset}(\text{nkfilter } P n nell) = \{ n+i \mid i. i \leq \text{nlength } nell \wedge P (\text{nnth } nell i) \}$

using *assms in-nkfilter-nset[of nell P - n] by blast*

lemma *nlength-nkfilter-le [simp]*:

assumes $\exists x \in \text{nset } nell. P x$

shows $\text{nlength } (\text{nkfilter } P n nell) \leq \text{nlength } nell$

using *assms*

by *transfer*

(*auto simp add: kfilter-lnull-conv epred-le-epredI kfilter-llength llength-lfilter-ile*)

lemma *nkfilter-nleast*:

assumes $\exists x \in \text{nset } xs. P x$

shows $\text{nnth } (\text{nkfilter } P n xs) 0 = n + (\text{nleast } P xs)$

using *assms*

by *transfer*

(*auto simp add: kfilter-not-lnull-conv kfilter-lnull-conv,*
metis enat-min-eq-0-iff kfilter-lnth-zero kfilter-lnull-conv the-enat-0 zero-enat-def)

lemma *ndistinct-nkfilter*:

assumes $\exists x \in \text{nset } nell. P x$

shows $\text{ndistinct}(\text{nkfilter } P n nell)$

using *assms*

by *transfer*

(*auto simp add: kfilter-lnull-conv ldistinct-kfilter*)

lemma *nkfilter-ndropn-nset*:

assumes $\exists x \in \text{nset } (\text{ndropn } k nell). P x$

$k \leq \text{nlength } nell$

shows $\text{nset}(\text{nkfilter } P n (\text{ndropn } k nell)) = \{ n+i \mid i. i \leq \text{nlength } nell - k \wedge P (\text{nnth } nell (k+i)) \}$

using *assms nkfilter-nset[of (ndropn k nell) P n]*

ndropn-nnth[of - nell k] by auto

lemma *nkfilter-ndropn-nset-b*:

assumes $\exists x \in \text{nset } (\text{ndropn } k nell). P x$

$k \leq \text{nlength } nell$

shows $\text{nset}(\text{nkfilter } P \ n \ (\text{ndropn } k \ \text{nell})) = \{ n + i \mid i. i \leq \text{nlength } \text{nell} - k \wedge P \ (\text{nnth } \text{nell } (i+k)) \}$
proof –
have 1: $\{ n+i \mid i. i \leq \text{nlength } \text{nell} - k \wedge P \ (\text{nnth } \text{nell } (k+i)) \} =$
 $\{ n + i \mid i. i \leq \text{nlength } \text{nell} - k \wedge P \ (\text{nnth } \text{nell } (i+k)) \}$
using *assms* **by** (*auto simp add: add commute*)
show ?thesis **using** *assms* **by** (*simp add: 1 nkfilter-ndropn-nset*)
qed

lemma *nfiter-nkfilter-ntaken-nlength-0*:
assumes $P \ (\text{nnth } \text{nell } (\ (\text{nnth } (\text{nkfilter } P \ n \ \text{nell}) \ k) - n \))$
 $k \leq \text{nlength } (\text{nfiter } P \ \text{nell})$
shows $(\exists x \in \text{nset } \text{nell}. P \ x)$
using *assms*
by (*metis enat-ile in-nset-conv-nnth linorder-le-cases ntaken-all ntaken-nlast*)

lemma *nkfilter-nlength-n-zero*:
assumes $(\exists x \in \text{nset } \text{nell}. P \ x)$
shows $\text{nlength}(\text{nkfilter } P \ n \ \text{nell}) = \text{nlength}(\text{nkfilter } P \ 0 \ \text{nell})$
using *assms* **by** (*simp add: nkfilter-nlength*)

lemma *nkfilter-nnth-n-zero*:
assumes $(\exists x \in \text{nset } \text{nell}. P \ x)$
 $k \leq \text{nlength}(\text{nkfilter } P \ n \ \text{nell})$
shows $(\text{nnth } (\text{nkfilter } P \ n \ \text{nell}) \ k) - n = (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)$
using *assms*
by *transfer*
(auto split: if-split-asm,
metis kfilter-lnull-conv,
metis kfilter-lnull-conv,
metis co.enat.exhaust-sel iless-Suc-eq kfilter-llength-n-zero kfilter-lnth-n-zero llength-eq-0
min.orderE the-enat.simps)

lemma *nkfilter-n-zero*:
assumes $(\exists x \in \text{nset } \text{nell}. P \ x)$
shows $(\text{nkfilter } P \ n \ \text{nell}) = \text{nmap } (\lambda i. i+n) \ (\text{nkfilter } P \ 0 \ \text{nell})$
proof –
have 1: $\text{nlength}(\text{nkfilter } P \ n \ \text{nell}) = \text{nlength } (\text{nmap } (\lambda i. i+n) \ (\text{nkfilter } P \ 0 \ \text{nell}))$
using *assms nkfilter-nlength-n-zero* **by** *fastforce*
have 2: $\bigwedge k. k \leq \text{nlength } (\text{nkfilter } P \ n \ \text{nell}) \longrightarrow$
 $\text{nnth } (\text{nkfilter } P \ n \ \text{nell}) \ k = \text{nnth } (\text{nmap } (\lambda i. i+n) \ (\text{nkfilter } P \ 0 \ \text{nell})) \ k$
using 1 *assms nkfilter-nnth-n-zero[of nell P - n]*
by (*metis diff-add nkfilter-lowerbound nlength-nmap nnth-nmap*)
show ?thesis **by** (*simp add: 1 2 nellist-eq-nnth-eq*)
qed

lemma *nkfilter-n-zero-a*:
assumes $(\exists x \in \text{nset } \text{nell}. P \ x)$
shows $(\text{nkfilter } P \ 0 \ \text{nell}) = \text{nmap } (\lambda i. i-n) \ (\text{nkfilter } P \ n \ \text{nell})$
proof –
have 1: $\text{nlength}(\text{nkfilter } P \ 0 \ \text{nell}) = \text{nlength } (\text{nmap } (\lambda i. i-n) \ (\text{nkfilter } P \ n \ \text{nell}))$

by (metis assms nkfilter-nlength-n-zero nlength-nmap)
 have 2: $\bigwedge k. k \leq \text{nlength}(\text{nkfilter } P \ 0 \ \text{nell}) \longrightarrow$
 $\text{nnth}(\text{nkfilter } P \ 0 \ \text{nell}) \ k = \text{nnth}(\text{nmap } (\lambda i. i-n) (\text{nkfilter } P \ n \ \text{nell})) \ k$
 by (simp add: 1 assms nkfilter-nnth-n-zero)
 show ?thesis by (simp add: 1 2 nellist-eq-nnth-eq)
 qed

lemma *nkfilter-holds*:

assumes $(\exists x \in \text{nset } \text{nell}. P \ x)$
 $y \in \text{nset}(\text{nkfilter } P \ n \ \text{nell})$
 shows $P(\text{nnth } \text{nell} \ (y-n))$
 using assms in-nkfilter-nset[of nell P] by force

lemma *nkfilter-holds-not*:

assumes $(\exists x \in \text{nset } \text{nell}. P \ x)$
 $y \in \{i+n \mid i. i \leq \text{nlength } \text{nell}\} - (\text{nset}(\text{nkfilter } P \ n \ \text{nell}))$
 shows $\neg P(\text{nnth } \text{nell} \ (y-n))$
 using assms nkfilter-nset[of nell P n] by auto

lemma *nkfilter-holds-a*:

assumes $(\exists x \in \text{nset } \text{nell}. P \ x)$
 $i \leq \text{nlength } \text{nell}$
 $(i+n) \in \text{nset}(\text{nkfilter } P \ n \ \text{nell})$
 shows $P(\text{nnth } \text{nell} \ i)$
 using assms nkfilter-holds[of nell P i+n n] by simp

lemma *nkfilter-holds-not-a*:

assumes $(\exists x \in \text{nset } \text{nell}. P \ x)$
 $i \leq \text{nlength } \text{nell}$
 $P(\text{nnth } \text{nell} \ i)$
 shows $(i+n) \in \text{nset}(\text{nkfilter } P \ n \ \text{nell})$
 using assms by (simp add: in-nkfilter-nset)

lemma *nkfilter-holds-b*:

assumes $(\exists x \in \text{nset } \text{nell}. P \ x)$
 $i \leq \text{nlength } \text{nell}$
 shows $(i+n) \in \text{nset}(\text{nkfilter } P \ n \ \text{nell}) = P(\text{nnth } \text{nell} \ i)$
 using assms by (meson nkfilter-holds-a nkfilter-holds-not-a)

lemma *nkfilter-holds-c*:

assumes $(\exists x \in \text{nset } \text{nell}. P \ x)$
 $n \leq i$
 $i-n \leq \text{nlength } \text{nell}$
 shows $i \in \text{nset}(\text{nkfilter } P \ n \ \text{nell}) = P(\text{nnth } \text{nell} \ (i-n))$
 using assms
 by (metis diff-add idiff-enat-enat nkfilter-holds-b)

lemma *nkfilter-holds-not-b*:

assumes $(\exists x \in \text{nset } \text{nell}. P \ x)$
 $n \leq i$

$i - n \leq \text{nlength } nell$
shows $i \notin \text{nset } (\text{nkfilter } P \ n \ nell) = (\neg P \ (\text{nnth } nell \ (i - n)))$
using *assms*
by (*simp add: nkfilter-holds-c*)

lemma *nkfilter-disjoint-nset-coset*:
assumes $(\exists x \in \text{nset } nell. P \ x)$
shows $(\{i + n \mid i. i \leq \text{nlength } nell\} - \text{nset}(\text{nkfilter } P \ n \ nell)) \cap \text{nset}(\text{nkfilter } P \ n \ nell) = \{\}$
using *assms* **by** (*simp add: inf.commute*)

lemma *nidx-nkfilter-expand*:
assumes $(\exists x \in \text{nset } nell. P \ x)$
 $(\text{Suc } i) \leq \text{nlength}(\text{nkfilter } P \ n \ nell)$
shows $\text{nnth } (\text{nkfilter } P \ n \ nell) \ i < \text{nnth } (\text{nkfilter } P \ n \ nell) \ (\text{Suc } i)$
using *assms* **by** (*simp add: nkfilter-mono*)

lemma *nidx-nkfilter*:
assumes $(\exists x \in \text{nset } nell. P \ x)$
shows $\text{nidx } (\text{nkfilter } P \ n \ nell)$
using *assms* *nidx-nkfilter-expand*[*of nell P - n*] *nidx-expand*[*of (nkfilter P n nell)*]
by *blast*

lemma *nidx-nkfilter-gr-eq*:
assumes $(\exists x \in \text{nset } nell. P \ x)$
 $k \leq j$
 $j \leq \text{nlength } (\text{nkfilter } P \ n \ nell)$
shows $\text{nnth } (\text{nkfilter } P \ n \ nell) \ k \leq \text{nnth } (\text{nkfilter } P \ n \ nell) \ j$
using *assms* *nidx-nkfilter*[*of nell P n*] *nidx-less-eq*[*of nkfilter P n nell k j*]
by *blast*

lemma *nidx-nkfilter-gr*:
assumes $(\exists x \in \text{nset } nell. P \ x)$
shows $(\forall j. k < j \wedge j \leq \text{nlength } (\text{nkfilter } P \ n \ nell) \longrightarrow \text{nnth } (\text{nkfilter } P \ n \ nell) \ k < \text{nnth}(\text{nkfilter } P \ n \ nell) \ j)$
using *assms* *nidx-nkfilter*[*of nell P n*] *nidx-less*[*of nkfilter P n nell*]
using *less-imp-Suc-add* **by** *blast*

lemma *nidx-nkfilter-less-eq*:
assumes $(\exists x \in \text{nset } nell. P \ x)$
 $k \leq \text{nlength } (\text{nkfilter } P \ n \ nell)$
shows $\forall j. j \leq k \longrightarrow \text{nnth } (\text{nkfilter } P \ n \ nell) \ j \leq \text{nnth } (\text{nkfilter } P \ n \ nell) \ k$
using *assms* **by** (*simp add: nidx-nkfilter-gr-eq*)

lemma *nkfilter-not-before*:
assumes $(\exists x \in \text{nset } nell. P \ x)$
 $i < (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ 0)$
shows $\neg P \ (\text{nnth } nell \ i)$
using *assms*
by *transfer*

```

(auto simp add: min-def split: if-split-asm,
meson kfilter-not-before llength-eq-0 not-gr-zero,
metis dual-order.strict-trans kfilter-not-before less-enatE llength-eq-0 not-gr-zero
not-le-imp-less the-enat.simps,
meson le-less-linear less-enatE less-nat-zero-code,
meson le-less-linear less-enatE not-less0)

```

lemma *nkfilter-n-not-before*:

```

assumes ( $\exists x \in \text{nset } (\text{ndropn } n \text{ nell}). P x$ )
           $n \leq \text{nlength nell}$ 
           $n \leq i$ 
           $i < (\text{nnth } (\text{nkfilter } P n (\text{ndropn } n \text{ nell})) ) 0$ 
shows  $\neg P (\text{nnth nell } i)$ 

```

using *assms*

apply *transfer*

unfolding *min-def*

using *zero-enat-def*

proof (*auto split: if-split-asm*)

show $\bigwedge n \text{ nell } P i x.$

$\text{enat } 0 = 0 \implies$

$\neg \text{lnull nell} \implies$

$\text{enat } n \leq \text{epred } (\text{llength nell}) \implies$

$n \leq i \implies$

$\neg \text{lnull } (\text{kfilter } P n (\text{ldropn } n \text{ nell})) \implies$

$i < \text{lnth } (\text{kfilter } P n (\text{ldropn } n \text{ nell})) 0 \implies$

$x \in \text{lset } (\text{ldropn } n \text{ nell}) \implies P x \implies \text{enat } i \leq \text{epred } (\text{llength nell}) \implies P (\text{lnth nell } i) \implies \text{False}$

by (*metis co.enat.exhaust-sel iless-Suc-eq kfilter-n-not-before llength-eq-0 not-gr-zero*)

next

fix *na :: nat and nella :: 'b llist and Pa :: 'b \Rightarrow bool and ia :: nat and x :: 'b*

assume *a1: ia < lnth (kfilter Pa na (ldropn na nella)) 0*

assume *a2: $\neg \text{lnull } (\text{kfilter } Pa na (\text{ldropn } na \text{ nella}))$*

assume *a3: enat na \leq epred (llength nella)*

assume *a4: $\neg \text{lnull nella}$*

assume *a5: $\neg \text{enat ia} \leq \text{epred } (\text{llength nella})$*

assume *a6: Pa (lnth nella (the-enat (epred (llength nella))))*

have *f7: $\forall p n \text{ bs. lnull } (\text{kfilter } p n (\text{bs}::'b \text{ llist})) \vee \text{lnth } (\text{kfilter } p n \text{ bs}) 0 = n + \text{lleast } p \text{ bs}$*

by (*meson kfilter-lnth-zero*)

then have *f8: lnth (kfilter Pa na (ldropn na nella)) 0 = na + lleast Pa (ldropn na nella)*

using *a2 by blast*

have *f9: 0 < llength (kfilter Pa na (ldropn na nella))*

using *a2 by simp*

have *f10: na \leq the-enat (epred (llength nella))*

by (*metis a3 a5 enat-ord-code(4) enat-ord-simps(1) enat-the-enat less-imp-le*)

have *f11: the-enat (epred (llength nella)) < lnth (kfilter Pa na (ldropn na nella)) 0*

by (*metis a1 a5 dual-order.strict-trans enat-ord-simps(2) enat-ord-simps(3) enat-the-enat not-le-imp-less*)

then show *False using kfilter-n-not-before[of Pa na nella (the-enat (epred (llength nella)))]*

using *f10 f11*

by (*metis a3 a4 a6 co.enat.exhaust-sel f9 iless-Suc-eq llength-eq-0*)

qed

lemma *nkfilter-not-after*:
assumes $(\exists x \in \text{nset nell}. P x)$
 $\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k < i$
 $\text{nlength}(\text{nkfilter } P \ 0 \ \text{nell}) = (\text{enat } k)$
 $i \leq \text{nlength nell}$
shows $\neg P (\text{nnth nell } i)$
using *assms*
by *transfer*
(auto simp add: min-def split: if-split-asm,
metis co.enat.exhaust-sel diff-Suc-1 eSuc-enat iless-Suc-eq kfilter-not-after llength-eq-0
not-gr-zero)

lemma *nkfilter-n-not-after*:
assumes $(\exists x \in \text{nset} (\text{ndropn } n \ \text{nell}). P x)$
 $n \leq \text{nlength nell}$
 $\text{nnth} (\text{nkfilter } P \ n \ (\text{ndropn } n \ \text{nell})) \ k < i$
 $\text{nlength}(\text{nkfilter } P \ n \ (\text{ndropn } n \ \text{nell})) = (\text{enat } k)$
 $i \leq \text{nlength nell}$
shows $\neg P (\text{nnth nell } i)$
using *assms*
by *transfer*
(auto split: if-split-asm,
metis diff-Suc-1 eSuc-enat eSuc-epred iless-Suc-eq kfilter-n-not-after llength-eq-0 not-gr-zero)

lemma *nkfilter-not-between*:
assumes $(\exists x \in \text{nset nell}. P x)$
 $\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k < i$
 $i < \text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) (\text{Suc } k)$
 $(\text{Suc } k) \leq \text{nlength} (\text{nkfilter } P \ 0 \ \text{nell})$
shows $\neg P (\text{nnth nell } i)$
using *assms*
by *transfer*
(auto simp add: min-def split: if-split-asm,
metis co.enat.exhaust-sel iless-Suc-eq kfilter-not-between llength-eq-0,
metis co.enat.exhaust-sel gen-llength-def iless-Suc-eq kfilter-upperbound le-less-trans
llength-code llength-eq-0 min.cobounded2 min.strict-order-iff min-enat-simps(1),
meson Suc-ile-eq less-imp-le,
meson Suc-ile-eq dual-order.strict-implies-order)

lemma *ntaken-nset*:
assumes $k \leq \text{nlength nell}$
shows $\text{nset} (\text{ntaken } k \ \text{nell}) = \{ (\text{nnth nell } i) \mid i. i \leq k \}$
using *assms*
by *(auto simp add: in-nset-conv-nnth)*
(metis min.orderE ntaken-nnth,
metis min.orderE ntaken-nnth)

lemma *ndropn-nset*:
assumes $k \leq \text{nlength } \text{nell}$
shows $\text{nset } (\text{ndropn } k \text{ nell}) = \{ (\text{nnth } \text{nell } i) \mid i. k \leq i \wedge i \leq \text{nlength } \text{nell} \}$
using *assms*
by (*auto simp add: in-nset-conv-nnth*)
(metis add.commute enat-min-eq le-add1 plus-enat-simps(1),
metis enat-minus-mono1 idiff-enat-enat le-add-diff-inverse)

lemma *nfilter-nkfilter-ntaken-nidx-a*:
assumes $\exists x \in \text{nset } \text{nell}. P x$
 $k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
shows $\text{nidx } (\text{ntaken } k (\text{nkfilter } P \text{ n nell}))$
using *assms*
by (*simp add: nidx-expand nidx-nkfilter-gr ntaken-nnth*)

lemma *nfilter-nkfilter-ndropn-nidx-a*:
assumes $\exists x \in \text{nset } \text{nell}. P x$
 $k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
shows $\text{nidx } (\text{ndropn } k (\text{nkfilter } P \text{ n nell}))$
using *assms*
by *transfer*
(auto simp add: kfilter-lnull-conv,
metis (no-types, lifting) gr-implies-not-zero kfilter-llength leI lfilter-kfilter-ldropn-lidx-a
lidx-def llength-eq-0 lnull-ldropn)

lemma *nfilter-nkfilter-ntaken-nidx-b*:
assumes $P (\text{nnth } \text{nell } ((\text{nnth } (\text{nkfilter } P \text{ } 0 \text{ nell}) k)))$
 $k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
shows $\text{nidx } (\text{nkfilter } P \text{ } 0 (\text{ntaken } (\text{nnth } (\text{nkfilter } P \text{ } 0 \text{ nell}) k) \text{ nell}))$
using *assms nidx-nkfilter[of (ntaken (nnth (nkfilter P 0 nell) k) nell) P 0]*
by (*metis nfinite-ntaken nset-nlast ntaken-nlast*)

lemma *nfilter-nkfilter-ntaken-nidx-b-1*:
assumes $\exists x \in \text{nset } \text{nell}. P x$
 $k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
shows $\text{nidx } (\text{nkfilter } P \text{ } 0 (\text{ntaken } (\text{nnth } (\text{nkfilter } P \text{ } 0 \text{ nell}) k) \text{ nell}))$
using *assms nfilter-nkfilter-ntaken-nidx-b[of P nell k] nkfilter-holds[of nell P]*
by (*metis in-nset-conv-nnth nkfilter-nlength nkfilter-nnth-n-zero*)

lemma *nfilter-nkfilter-ntaken-nidx-b-2*:
assumes $P (\text{nnth } \text{nell } ((\text{nnth } (\text{nkfilter } P \text{ } n \text{ nell}) k) - n))$
 $k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
shows $\text{nidx } (\text{nkfilter } P \text{ } n (\text{ntaken } (\text{nnth } (\text{nkfilter } P \text{ } n \text{ nell}) k) \text{ nell}))$
using *assms nidx-nkfilter[of (ntaken (nnth (nkfilter P n nell) k) nell) P n]*
by (*metis (mono-tags, lifting) diff-le-self linorder-le-cases mem-Collect-eq*
nfilter-nkfilter-ntaken-nlength-0 ntaken-all ntaken-nset)

lemma *nfilter-nkfilter-ntaken-nidx-b-3*:

assumes $\exists x \in \text{nset nell}. P x$
 $k \leq \text{nlength} (\text{nfilter } P \text{ nell})$
shows $\text{nidx} (\text{nkfilter } P n (\text{ntaken} (\text{nnth} (\text{nkfilter } P n \text{ nell}) k) \text{ nell}))$
using *assms nfilter-nkfilter-ntaken-nidx-b-2*[of $P \text{ nell } n k$] *nkfilter-holds*
by (*metis in-nset-conv-nnth nkfilter-nlength*)

lemma *nfilter-nkfilter-ndropn-nidx-b:*

assumes $P (\text{nnth nell} ((\text{nnth} (\text{nkfilter } P 0 \text{ nell}) k)))$
 $k \leq \text{nlength} (\text{nfilter } P \text{ nell})$
shows $\text{nidx} (\text{nkfilter } P 0 (\text{ndropn} (\text{nnth} (\text{nkfilter } P 0 \text{ nell}) k) \text{ nell}))$
using *assms nidx-nkfilter*[of $(\text{ndropn} (\text{nnth} (\text{nkfilter } P 0 \text{ nell}) k) \text{ nell}) P 0$]
by (*metis diff-add diff-zero in-nset-conv-nnth ndropn-nnth zero-enat-def zero-le*)

lemma *nfilter-nkfilter-ndropn-nidx-b-1:*

assumes $\exists x \in \text{nset nell}. P x$
 $k \leq \text{nlength} (\text{nfilter } P \text{ nell})$
shows $\text{nidx} (\text{nkfilter } P 0 (\text{ndropn} (\text{nnth} (\text{nkfilter } P 0 \text{ nell}) k) \text{ nell}))$
using *assms nfilter-nkfilter-ndropn-nidx-b*[of $P \text{ nell } k$] *nkfilter-holds*[of $\text{nell } P$]
by (*metis in-nset-conv-nnth nkfilter-nlength nkfilter-nnth-n-zero*)

lemma *nfilter-nkfilter-ndropn-nidx-b-2:*

assumes $P (\text{nnth nell} ((\text{nnth} (\text{nkfilter } P n \text{ nell}) k) - n))$
 $k \leq \text{nlength} (\text{nfilter } P \text{ nell})$
shows $\text{nidx} (\text{nkfilter } P ((\text{nnth} (\text{nkfilter } P n \text{ nell}) k) - n)$
 $(\text{ndropn} ((\text{nnth} (\text{nkfilter } P n \text{ nell}) k) - n) \text{ nell}))$

proof –

have 1: $\text{enat} (\text{nnth} (\text{nkfilter } P n \text{ nell}) k) - n \leq \text{nlength nell}$
using *nkfilter-upperbound*[of $\text{nell } P$] *nkfilter-nnth-n-zero*[of $\text{nell } P k n$] *assms*
by (*metis gen-nlength-def idiff-enat-enat nfilter-nkfilter-ntaken-nlength-0*
nkfilter-nlength nlength-code)
have 2: $\text{nset} (\text{ndropn} ((\text{nnth} (\text{nkfilter } P n \text{ nell}) k) - n) \text{ nell}) =$
 $\{ (\text{nnth nell } i) \mid i. ((\text{nnth} (\text{nkfilter } P n \text{ nell}) k) - n) \leq i \wedge i \leq \text{nlength nell} \}$
using *ndropn-nset*[of $(\text{nnth} (\text{nkfilter } P n \text{ nell}) k) - n \text{ nell}$] 1 **by** *simp*
have 3: $\exists x \in \text{nset} (\text{ndropn} ((\text{nnth} (\text{nkfilter } P n \text{ nell}) k) - n) \text{ nell}). P x$
using 1 2 *assms*(1) **by** *auto*
show ?thesis **using** 3
 nidx-nkfilter [of $(\text{ndropn} ((\text{nnth} (\text{nkfilter } P n \text{ nell}) k) - n) \text{ nell}) P (\text{nnth} (\text{nkfilter } P n \text{ nell}) k) - n]$
by *blast*
qed

lemma *nfilter-nkfilter-ndropn-nidx-b-3:*

assumes $\exists x \in \text{nset nell}. P x$
 $k \leq \text{nlength} (\text{nfilter } P \text{ nell})$
shows $\text{nidx} (\text{nkfilter } P ((\text{nnth} (\text{nkfilter } P n \text{ nell}) k) - n)$
 $(\text{ndropn} ((\text{nnth} (\text{nkfilter } P n \text{ nell}) k) - n) \text{ nell}))$
using *assms nfilter-nkfilter-ndropn-nidx-b-2*
by (*metis in-nset-conv-nnth nkfilter-holds nkfilter-nlength*)

lemma *ntaken-nkfilter-ntaken-nset-eq:*

assumes $P (\text{nnth nell} ((\text{nnth} (\text{nkfilter } P 0 \text{ nell}) k)))$

$k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
shows $\text{nset}(\text{ntaken } k \text{ (nkfilter } P \text{ 0 nell)}) =$
 $\text{nset}(\text{nkfilter } P \text{ 0 (ntaken ((nnth (nkfilter } P \text{ 0 nell) } k)) \text{ nell}))$
proof –
have 1: $(\text{nnth (nkfilter } P \text{ 0 nell) } k) \leq \text{nlength nell}$
using *assms nkfilter-upperbound[of nell P k 0]*
by (*metis diff-zero gen-nlength-def nfilter-nkfilter-ntaken-nlength-0 nkfilter-nlength nlength-code*)
have 2: $\exists x \in \text{nset nell. } P x$
using *assms by (metis 1 exists-Pred-nnth-nset)*
have 3: $\{i. i \leq \text{nlength}(\text{ntaken (nnth (nkfilter } P \text{ 0 nell) } k) \text{ nell}) \wedge$
 $P (\text{nnth (ntaken (nnth (nkfilter } P \text{ 0 nell) } k) \text{ nell) } i) \}$
 $= \{i. i \leq (\text{nnth (nkfilter } P \text{ 0 nell) } k) \wedge P (\text{nnth nell } i)\}$
by (*auto simp add: ntaken-nnth 1 order-subst2*)
have 4: $\exists x \in \text{nset}(\text{ntaken (nnth (nkfilter } P \text{ 0 nell) } k) \text{ nell}). P x$
using 1 *assms(1) ntaken-nset by fastforce*
have 5: $\{i. i \leq (\text{nnth (nkfilter } P \text{ 0 nell) } k) \wedge P (\text{nnth nell } i)\} =$
 $\{i. i \leq (\text{nnth (nkfilter } P \text{ 0 nell) } k) \wedge i \in \text{nset}(\text{nkfilter } P \text{ 0 nell})\}$
using 4 1 2 *nkfilter-holds-b*
by (*metis (mono-tags, opaque-lifting) add-cancel-left-right enat-ord-simps(1) order.trans*)
have 6: $\{i. i \leq (\text{nnth (nkfilter } P \text{ 0 nell) } k) \wedge i \in \text{nset}(\text{nkfilter } P \text{ 0 nell})\} =$
 $\{i. i \leq (\text{nnth (nkfilter } P \text{ 0 nell) } k) \wedge$
 $i \in \{(\text{nnth (nkfilter } P \text{ 0 nell) } j) \mid j. j \leq \text{nlength (nkfilter } P \text{ 0 nell)}\}\}$
by (*simp add: nset-conv-nnth*)
have 7: $\{i. i \leq (\text{nnth (nkfilter } P \text{ 0 nell) } k) \wedge$
 $i \in \{(\text{nnth (nkfilter } P \text{ 0 nell) } j) \mid j. j \leq \text{nlength (nkfilter } P \text{ 0 nell)}\}\} =$
 $\{(\text{nnth (nkfilter } P \text{ 0 nell) } j) \mid j. j \leq k\}$
by (*auto simp add: assms 2 nidx-nkfilter-less-eq nkfilter-nlength*)
(metis 2 dual-order.antisym le-cases nidx-nkfilter-gr-eq nkfilter-nlength,
metis assms(2) dual-order.trans enat-ord-simps(1))
have 8: $k \leq \text{nlength (nkfilter } P \text{ 0 nell)}$
by (*simp add: 2 assms(2) nkfilter-nlength*)
have 9: $\{(\text{nnth (nkfilter } P \text{ 0 nell) } j) \mid j. j \leq k\} = \text{nset}(\text{ntaken } k \text{ (nkfilter } P \text{ 0 nell)})$
using *ntaken-nset[of k (nkfilter } P \text{ 0 nell)] 8 by auto*
have 91: $\text{min (enat (nnth (nkfilter } P \text{ 0 nell) } k)) (\text{nlength nell}) = (\text{nnth (nkfilter } P \text{ 0 nell) } k)$
by (*simp add: 1 min-def*)
have 10: $\text{nset}(\text{nkfilter } P \text{ 0 (ntaken ((nnth (nkfilter } P \text{ 0 nell) } k)) \text{ nell})) =$
 $\{i. i \leq (\text{nnth (nkfilter } P \text{ 0 nell) } k) \wedge$
 $P (\text{nnth (ntaken (nnth (nkfilter } P \text{ 0 nell) } k) \text{ nell) } i) \}$
using *nkfilter-nset[of (ntaken (nnth (nkfilter } P \text{ 0 nell) } k) \text{ nell) } P \text{ 0]*
1 4 91 *ntaken-nlength[of (nnth (nkfilter } P \text{ 0 nell) } k) \text{ nell]*
by *auto*
have 11: $\text{nlength}(\text{ntaken ((nnth (nkfilter } P \text{ 0 nell) } k)) \text{ nell}) = (\text{nnth (nkfilter } P \text{ 0 nell) } k)$
by (*simp add: 1*)
have 12: $\{i. i \leq (\text{nnth (nkfilter } P \text{ 0 nell) } k) \wedge$
 $P (\text{nnth (ntaken (nnth (nkfilter } P \text{ 0 nell) } k) \text{ nell) } i) \} =$
 $\{i. i \leq \text{nlength}((\text{ntaken ((nnth (nkfilter } P \text{ 0 nell) } k)) \text{ nell})) \wedge$
 $P (\text{nnth (ntaken (nnth (nkfilter } P \text{ 0 nell) } k) \text{ nell) } i) \}$
using 11 *by auto*
show *?thesis*
using 10 12 3 5 6 7 9 *by auto*

qed

lemma *ntaken-nkfilter-ntaken-nset-eq-1:*

assumes $\exists x \in \text{nset } \text{nell}. P x$

$k \leq \text{nlength } (\text{nfilter } P \text{ nell})$

shows $\text{nset}(\text{ntaken } k (\text{nkfilter } P 0 \text{ nell})) =$

$\text{nset}(\text{nkfilter } P 0 (\text{ntaken } ((\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k)) \text{ nell}))$

using *assms ntaken-nkfilter-ntaken-nset-eq[of P nell k]*

nkfilter-holds[of nell P (nnth (nkfilter P 0 nell) k) 0]

by (*metis in-nset-conv-nnth nkfilter-nlength nkfilter-nnth-n-zero*)

lemma *ndropn-nkfilter-ndropn-nset-eq:*

assumes $P (\text{nnth } \text{nell} ((\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k)))$

$k \leq \text{nlength } (\text{nfilter } P \text{ nell})$

shows $\text{nset}(\text{ndropn } k (\text{nkfilter } P 0 \text{ nell})) =$

$\text{nset}(\text{nkfilter } P (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) (\text{ndropn } ((\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k)) \text{ nell}))$

proof –

have 1: $(\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \leq \text{nlength } \text{nell}$

using *nkfilter-upperbound[of nell P k 0] assms*

by (*metis diff-zero gen-nlength-def nfilter-nkfilter-ntaken-nlength-0 nkfilter-nlength nlength-code*)

have 2: $\exists x \in \text{nset } \text{nell}. P x$

using *assms by (metis 1 exists-Pred-nnth-nset)*

have 4: $\exists x \in \text{nset}(\text{ndropn } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \text{ nell}). P x$

using 1 *assms(1) ndropn-nset by fastforce*

have 10: $\text{nset}(\text{nkfilter } P (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k)$

$(\text{ndropn } ((\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k)) \text{ nell})) =$

$\{(\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) + i \mid i. i \leq \text{nlength } \text{nell} - (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \wedge$
 $P (\text{nnth } \text{nell} (i + (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k))) \}$

using 1

nkfilter-ndropn-nset-b[of (nnth (nkfilter P 0 nell) k) nell P (nnth (nkfilter P 0 nell) k)]

4 **by** *linarith*

have 5: $\{(\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) + i \mid i. i \leq \text{nlength } \text{nell} - (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \wedge$
 $P (\text{nnth } \text{nell} (i + (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k))) \} =$

$\{(\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) + i \mid i. i \leq \text{nlength } \text{nell} - (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \wedge$
 $(i + (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k)) \in \text{nset}(\text{nkfilter } P 0 \text{ nell})\}$

proof (*auto simp add: add.commute*)

fix $i :: \text{nat}$

assume *a1: enat i ≤ nlength nell – enat (nnth (nkfilter P 0 nell) k)*

assume *a2: P (nnth nell (i + nnth (nkfilter P 0 nell) k))*

have *f0: enat (i + (nnth (nkfilter P 0 nell) k)) ≤ nlength nell*

using 1 *a1 enat-min-eq by auto*

show $i + \text{nnth } (\text{nkfilter } P 0 \text{ nell}) k \in \text{nset } (\text{nkfilter } P 0 \text{ nell})$

by (*metis Nat.add-0-right a2 f0 in-nset-conv-nnth nkfilter-holds-b*)

next

fix i

assume *b0: enat i ≤ nlength nell – enat (nnth (nkfilter P 0 nell) k)*

assume *b1: i + nnth (nkfilter P 0 nell) k ∈ nset (nkfilter P 0 nell)*

show $P (\text{nnth } \text{nell} (i + \text{nnth } (\text{nkfilter } P 0 \text{ nell}) k))$

using *nkfilter-holds[of nell P] 2 by (metis b1 minus-nat.diff-0)*

qed

have 51: $\{(nnth (nkfilter P 0 nell) k) + i \mid i. i \leq nlength\ nell - (nnth (nkfilter P 0 nell) k) \wedge (i + (nnth (nkfilter P 0 nell) k)) \in nset(nkfilter P 0 nell)\} =$
 $\{(nnth (nkfilter P 0 nell) k) + i \mid i. (nnth (nkfilter P 0 nell) k) \leq i + (nnth (nkfilter P 0 nell) k) \wedge i + (nnth (nkfilter P 0 nell) k) \leq nlength\ nell \wedge (i + (nnth (nkfilter P 0 nell) k)) \in nset(nkfilter P 0 nell)\}$
by *auto*
 $(metis\ 2\ add.left-neutral\ in-nset-conv-nnth\ nkfilter-upperbound\ zero-enat-def,$
 $metis\ add-diff-cancel-right'\ enat-minus-mono1\ idiff-enat-enat)$
have 52: $\{(nnth (nkfilter P 0 nell) k) + i \mid i. (nnth (nkfilter P 0 nell) k) \leq i + (nnth (nkfilter P 0 nell) k) \wedge i + (nnth (nkfilter P 0 nell) k) \leq nlength\ nell \wedge (i + (nnth (nkfilter P 0 nell) k)) \in nset(nkfilter P 0 nell)\} =$
 $\{j. (nnth (nkfilter P 0 nell) k) \leq j \wedge j \leq nlength\ nell \wedge j \in nset(nkfilter P 0 nell)\}$
by $(metis\ (no-types,\ lifting)\ add.commute\ diff-add)$
have 53: $\{j. (nnth (nkfilter P 0 nell) k) \leq j \wedge j \leq nlength\ nell \wedge j \in nset(nkfilter P 0 nell)\} =$
 $\{j. (nnth (nkfilter P 0 nell) k) \leq j \wedge j \leq nlength\ nell \wedge j \in \{(nnth (nkfilter P 0 nell) jj) \mid jj. jj \leq nlength\ (nkfilter P 0 nell)\}\}$
by $(simp\ add:\ nset-conv-nnth)$
have 54: $\{j. (nnth (nkfilter P 0 nell) k) \leq j \wedge j \leq nlength\ nell \wedge j \in \{(nnth (nkfilter P 0 nell) jj) \mid jj. jj \leq nlength\ (nkfilter P 0 nell)\}\} =$
 $\{(nnth (nkfilter P 0 nell) j) \mid j. k \leq j \wedge j \leq nlength(nkfilter P 0 nell)\}$
using *assms 2*
by *(auto,*
 $metis\ dual-order.antisym\ le-cases\ nidx-less-eq\ nidx-nkfilter\ nkfilter-nlength,$
 $meson\ nidx-nkfilter-gr-eq,$
 $metis\ gen-nlength-def\ nkfilter-upperbound\ nlength-code)$
have 8: $k \leq nlength\ (nkfilter P 0 nell)$
by $(simp\ add:\ 2\ assms(2)\ nkfilter-nlength)$
have 9: $\{(nnth (nkfilter P 0 nell) j) \mid j. k \leq j \wedge j \leq nlength(nkfilter P 0 nell)\} =$
 $nset(ndropn k (nkfilter P 0 nell))$
by $(simp\ add:\ 8\ ndropn-nset)$
show *?thesis*
using *10 5 51 52 53 54 9 by auto*
qed

lemma *ndropn-nkfilter-ndropn-nset-eq-1:*

assumes $\exists x \in nset\ nell. P\ x$
 $k \leq nlength\ (nfilter\ P\ nell)$
shows $nset(ndropn\ k\ (nkfilter\ P\ 0\ nell)) =$
 $nset(nkfilter\ P\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ (ndropn\ ((nnth\ (nkfilter\ P\ 0\ nell)\ k))\ nell))$
using *assms ndropn-nkfilter-ndropn-nset-eq[of P nell k]*
by $(metis\ diff-zero\ in-nset-conv-nnth\ nkfilter-holds\ nkfilter-nlength)$

lemma *nkfilter-nkfilter-ntaken:*

assumes $\exists x \in nset\ nell. P\ x$
 $k \leq nlength\ (nfilter\ P\ nell)$
shows $ntaken\ k\ (nkfilter\ P\ 0\ nell) =$
 $(nkfilter\ P\ 0\ (ntaken\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell))$
using *assms*

by (*simp add: nfilter-nkfilter-ntaken-nidx-a nfilter-nkfilter-ntaken-nidx-b-1 nidx-nset-eq*
ntaken-nkfilter-ntaken-nset-eq-1)

lemma *nkfilter-nkfilter-ndropn:*

assumes $\exists x \in \text{nset } \text{nell}. P x$

$k \leq \text{nlength } (\text{nfilter } P \text{ nell})$

shows $\text{ndropn } k (\text{nkfilter } P 0 \text{ nell}) =$

$(\text{nkfilter } P (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) (\text{ndropn } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \text{ nell}))$

proof –

have 1: $P (\text{nnth } \text{nell } ((\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k)))$

using *nkfilter-holds[of nell P] assms*

by (*metis diff-zero in-nset-conv-nnth nkfilter-nlength*)

show *?thesis*

by (*metis 1 assms(1) assms(2) diff-zero ndropn-nkfilter-ndropn-nset-eq*

nfilter-nkfilter-ndropn-nidx-a nfilter-nkfilter-ndropn-nidx-b-3 nidx-nset-eq)

qed

lemma *nkfilter-nmap-nfilter:*

assumes $\exists x \in \text{nset } \text{nell}. P x$

shows $\text{nmap } (\lambda n. \text{nnth } \text{nell } n) (\text{nkfilter } P 0 \text{ nell}) = \text{nfilter } P \text{ nell}$

using *assms nellist-eq-nnth-eq[of nmap ($\lambda n. \text{nnth } \text{nell } n$) (nkfilter P 0 nell) nfilter P nell]*

nkfilter-nfilter[of nell P - 0] nkfilter-nnth-n-zero[of nell P - 0]

by (*simp add: nkfilter-nlength*)

lemma *nfilter-nkfilter-ntaken:*

assumes $P (\text{nnth } \text{nell } ((\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k)))$

$k \leq \text{nlength } (\text{nkfilter } P 0 \text{ nell})$

shows $\text{ntaken } k (\text{nfilter } P \text{ nell}) =$

$\text{nfilter } P (\text{ntaken } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \text{ nell})$

proof –

have 1: $\exists x \in \text{nset } \text{nell}. P x$

using *assms*

by (*metis in-nset-conv-nnth min.cobounded1 min-def nfinite-ntaken nset-nlast ntaken-all*
ntaken-nlast)

have 2: $\text{nfilter } P \text{ nell} = \text{nmap } (\lambda n. \text{nnth } \text{nell } n) (\text{nkfilter } P 0 \text{ nell})$

using 1 *assms* **by** (*simp add: nkfilter-nmap-nfilter*)

have 3: $\text{ntaken } k (\text{nfilter } P \text{ nell}) = \text{ntaken } k (\text{nmap } (\lambda n. \text{nnth } \text{nell } n) (\text{nkfilter } P 0 \text{ nell}))$

using 2 *by simp*

have 4: $\text{ntaken } k (\text{nmap } (\lambda n. \text{nnth } \text{nell } n) (\text{nkfilter } P 0 \text{ nell})) =$

$\text{nmap } (\lambda n. \text{nnth } \text{nell } n) (\text{ntaken } k (\text{nkfilter } P 0 \text{ nell}))$

by *simp*

have 5: $\exists x \in \text{nset } (\text{ntaken } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \text{ nell}). P x$

using *assms* **by** (*metis nfinite-ntaken nset-nlast ntaken-nlast*)

have 6: $(\text{nfilter } P (\text{ntaken } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \text{ nell})) =$

$\text{nmap } (\lambda s. \text{nnth } (\text{ntaken } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \text{ nell}) s)$
 $(\text{nkfilter } P 0 (\text{ntaken } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \text{ nell}))$

using 5 **by** (*simp add: nkfilter-nmap-nfilter*)

have 7: $\text{nmap } (\lambda n. \text{nnth } \text{nell } n) (\text{ntaken } k (\text{nkfilter } P 0 \text{ nell})) =$

$nmap (\lambda n. \text{nnth nell } n) (\text{nkfilter } P \ 0 (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \text{ nell}) \ k) \text{ nell}))$
by (*simp add: 1 2 assms(2) nkfilter-nkfilter-ntaken*)
have 70: $\text{enat } k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
using 2 *assms* **by** *auto*
have 71: $(\bigwedge z. z \in \text{nset } (\text{nkfilter } P \ 0 (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \text{ nell}) \ k) \text{ nell})) \implies$
 $(\lambda n. \text{nnth nell } n) \ z = (\lambda s. \text{nnth } (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \text{ nell}) \ k) \text{ nell}) \ s) \ z)$
using *assms 1 70 ntaken-nkfilter-ntaken-nset-eq[of P nell k]*
 $\text{ntaken-nset[of } k \ (\text{nkfilter } P \ 0 \text{ nell})] \text{ mem-Collect-eq}$
 $\text{nidx-nkfilter-gr-eq[of nell P - - 0]}$
proof *simp*
fix $z :: \text{nat}$
assume $a1: \text{enat } k \leq \text{nlength } (\text{nkfilter } P \ 0 \text{ nell})$
assume $a2: \bigwedge k \ j. \llbracket k \leq j; \text{enat } j \leq \text{nlength } (\text{nkfilter } P \ 0 \text{ nell}) \rrbracket \implies$
 $\text{nnth } (\text{nkfilter } P \ 0 \text{ nell}) \ k \leq \text{nnth } (\text{nkfilter } P \ 0 \text{ nell}) \ j$
assume $a3: z \in \text{nset } (\text{nkfilter } P \ 0 (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \text{ nell}) \ k) \text{ nell}))$
assume $\{\text{nnth } (\text{nkfilter } P \ 0 \text{ nell}) \ i \mid i. i \leq k\} =$
 $\text{nset } (\text{nkfilter } P \ 0 (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \text{ nell}) \ k) \text{ nell}))$
then have $\exists n. z = \text{nnth } (\text{nkfilter } P \ 0 \text{ nell}) \ n \wedge n \leq k$
using $a3$ **by** *blast*
then have $z \leq \text{nnth } (\text{nkfilter } P \ 0 \text{ nell}) \ k$
using $a2 \ a1$ **by** *meson*
then show $\text{nnth nell } z = \text{nnth } (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \text{ nell}) \ k) \text{ nell}) \ z$
by (*simp add: ntaken-nnth*)
qed
have 8: $nmap (\lambda n. \text{nnth nell } n)$
 $(\text{nkfilter } P \ 0 (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \text{ nell}) \ k) \text{ nell})) =$
 $nmap (\lambda s. \text{nnth } (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \text{ nell}) \ k) \text{ nell}) \ s)$
 $(\text{nkfilter } P \ 0 (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \text{ nell}) \ k) \text{ nell}))$
using 1 5 *assms nellist.map-cong0*
using 71 **by** *blast*
show *?thesis*
by (*simp add: 3 6 7 8*)
qed

lemma *nfilter-nkfilter-ntaken-1:*

assumes $\exists x \in \text{nset nell}. P \ x$
 $k \leq \text{nlength } (\text{nkfilter } P \ 0 \text{ nell})$
shows $\text{ntaken } k \ (\text{nfilter } P \text{ nell}) =$
 $\text{nfilter } P \ (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \text{ nell}) \ k) \text{ nell})$
using *assms nfilter-nkfilter-ntaken[of P nell k]*
by (*metis in-nset-conv-nnth nkfilter-holds nkfilter-nnth-n-zero*)

lemma *nkfilter-nmap-shift:*

assumes $\exists x \in \text{nset nell}. P \ x$
shows $nmap (\lambda s. \text{nnth nell } (s+n)) (\text{nkfilter } P \ 0 \text{ nell}) =$
 $nmap (\lambda s. \text{nnth nell } s) (\text{nkfilter } P \ n \text{ nell})$

proof –

have 1: $\text{nlength } (nmap (\lambda s. \text{nnth nell } (s+n)) (\text{nkfilter } P \ 0 \text{ nell})) =$
 $\text{nlength } (nmap (\lambda s. \text{nnth nell } s) (\text{nkfilter } P \ n \text{ nell}))$
by (*simp add: assms nkfilter-nlength*)

have 2: $\bigwedge i. i \leq \text{nlength } (\text{nmap } (\lambda s. \text{nnth } \text{nell } (s+n)) (\text{nkfilter } P \ 0 \ \text{nell})) \longrightarrow$
 $\text{nnth } (\text{nmap } (\lambda s. \text{nnth } \text{nell } (s+n)) (\text{nkfilter } P \ 0 \ \text{nell})) \ i =$
 $\text{nnth } \text{nell } ((\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ i) + n)$
by *simp*
have 3: $\bigwedge i. i \leq \text{nlength } (\text{nmap } (\lambda s. \text{nnth } \text{nell } s) (\text{nkfilter } P \ n \ \text{nell})) \longrightarrow$
 $\text{nnth } (\text{nmap } (\lambda s. \text{nnth } \text{nell } s) (\text{nkfilter } P \ n \ \text{nell})) \ i =$
 $\text{nnth } \text{nell } (\text{nnth } (\text{nkfilter } P \ n \ \text{nell}) \ i)$
by *simp*
have 4: $\bigwedge i. i \leq \text{nlength } (\text{nmap } (\lambda s. \text{nnth } \text{nell } (s+n)) (\text{nkfilter } P \ 0 \ \text{nell})) \longrightarrow$
 $\text{nnth } \text{nell } ((\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ i) + n) = \text{nnth } \text{nell } (\text{nnth } (\text{nkfilter } P \ n \ \text{nell}) \ i)$
using *nkfilter-n-zero[of nell P n]*
by (*simp add: assms*)
show ?thesis **using** 1 4 *nellist-eq-nnth-eq* **by** *force*
qed

lemma *nkfilter-nmap-shift-ndropn*:

assumes $\exists x \in \text{nset } (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ \text{nell}). P \ x$
 $k \leq \text{nlength } (\text{nkfilter } P \ 0 \ \text{nell})$
shows $\text{nmap } (\lambda s. \text{nnth } \text{nell } (s + (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)))$
 $(\text{nkfilter } P \ 0 \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ \text{nell})) =$
 $\text{nmap } (\lambda s. \text{nnth } \text{nell } s) (\text{nkfilter } P \ (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)$
 $(\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ \text{nell}))$

using *assms*

$\text{nellist-eq-nnth-eq}[of \ \text{nmap } (\lambda s. \text{nnth } \text{nell } (s + (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)))$
 $(\text{nkfilter } P \ 0 \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ \text{nell}))$
 $\text{nmap } (\lambda s. \text{nnth } \text{nell } s) (\text{nkfilter } P \ (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)$
 $(\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ \text{nell}))]$

using *nkfilter-n-zero[of (ndropn (nnth (nkfilter P 0 nell) k) nell) P*
 $(\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)]$

by *simp*

lemma *nfilter-nkfilter-ndropn*:

assumes $P \ (\text{nnth } \text{nell } ((\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)))$
 $k \leq \text{nlength } (\text{nkfilter } P \ 0 \ \text{nell})$
shows $\text{ndropn } k \ (\text{nfilter } P \ \text{nell}) =$
 $\text{nfilter } P \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ \text{nell})$

proof –

have 1: $\exists x \in \text{nset } \text{nell}. P \ x$

using *assms*

by (*metis in-nset-conv-nnth min.cobounded1 min-def nfinite-ntaken nset-nlast ntaken-all ntaken-nlast*)

have 2: $(\text{nfilter } P \ \text{nell}) = \text{nmap } (\lambda n. \text{nnth } \text{nell } n) (\text{nkfilter } P \ 0 \ \text{nell})$

by (*simp add: 1 nkfilter-nmap-nfilter*)

have 3: $\text{ndropn } k \ (\text{nfilter } P \ \text{nell}) = \text{ndropn } k \ (\text{nmap } (\lambda n. \text{nnth } \text{nell } n) (\text{nkfilter } P \ 0 \ \text{nell}))$

by (*simp add: 2*)

have 4: $\text{ndropn } k \ (\text{nmap } (\lambda n. \text{nnth } \text{nell } n) (\text{nkfilter } P \ 0 \ \text{nell})) =$

$\text{nmap } (\lambda n. \text{nnth } \text{nell } n) (\text{ndropn } k \ (\text{nkfilter } P \ 0 \ \text{nell}))$

using *ndropn-nmap by blast*

have 5: $\exists x \in \text{nset}(\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ \text{nell}). P \ x$

by (*metis add.commute add.left-neutral assms(1) in-nset-conv-nnth ndropn-nnth zero-enat-def zero-le*)

have 6: $nfilter\ P\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell) =$
 $nmap\ (\lambda s. nnth\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)\ s)$
 $(nkfilter\ P\ 0\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell))$
by (metis 5 nkfilter-nmap-nfilter)
have 7: $nmap\ (\lambda s. nnth\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)\ s)$
 $(nkfilter\ P\ 0\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)) =$
 $nmap\ (\lambda s. nnth\ nell\ (s+(nnth\ (nkfilter\ P\ 0\ nell)\ k)))$
 $(nkfilter\ P\ 0\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell))$
by (simp add: add.commute)
have 8: $nmap\ (\lambda s. nnth\ nell\ (s+(nnth\ (nkfilter\ P\ 0\ nell)\ k)))$
 $(nkfilter\ P\ 0\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)) =$
 $nmap\ (\lambda s. nnth\ nell\ s)$
 $(nkfilter\ P\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell))$
using 5 assms(2) nkfilter-nmap-shift-ndropn **by** fastforce
have 9: $nmap\ (\lambda s. nnth\ nell\ s)$
 $(nkfilter\ P\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)) =$
 $nmap\ (\lambda s. nnth\ nell\ s)$
 $(ndropn\ k\ (nkfilter\ P\ 0\ nell))$
by (simp add: 1 2 assms(2) nkfilter-nkfilter-ndropn)
show ?thesis **using** 3 4 6 7 8 9 **by** auto
qed

lemma *nfilter-nkfilter-ndropn-1*:
assumes $\exists x \in nset\ nell. P\ x$
 $k \leq nlength\ (nkfilter\ P\ 0\ nell)$
shows $ndropn\ k\ (nfilter\ P\ nell) =$
 $nfilter\ P\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)$
using assms *nfilter-nkfilter-ndropn*
by (metis in-nset-conv-nnth minus-nat.diff-0 nkfilter-holds)

lemma *nfilter-nkfilter-nsbn*:
assumes $P\ (nnth\ nell\ ((nnth\ (nkfilter\ P\ 0\ nell)\ i)))$
 $enat\ i \leq nlength\ (nkfilter\ P\ 0\ nell)$
 $P\ (nnth\ nell\ ((nnth\ (nkfilter\ P\ 0\ nell)\ j)))$
 $enat\ j \leq nlength\ (nkfilter\ P\ 0\ nell)$
 $i \leq j$
shows $nsbn\ (nfilter\ P\ nell)\ i\ j =$
 $(nfilter\ P\ (nsbn\ nell$
 $(nnth\ (nkfilter\ P\ 0\ nell)\ i)$
 $(nnth\ (nkfilter\ P\ 0\ nell)\ j)$
 $)$
 $)$

proof –

have 0: $nsbn\ (nfilter\ P\ nell)\ i\ j = ntaken\ (j - i)\ (ndropn\ i\ (nfilter\ P\ nell))$
using nsbn-def1 **by** blast
have 1: $ntaken\ (j - i)\ (ndropn\ i\ (nfilter\ P\ nell)) =$
 $ntaken\ (j - i)\ (nfilter\ P\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ i)\ nell))$

```

  by (simp add: assms(1) assms(2) nfilter-nkfilter-ndropn)
have 4: ((nnth (nkfilter P 0 (ndropn (nnth (nkfilter P 0 nell) i) nell)) (j - i)) +
        (nnth (nkfilter P 0 nell) i)) =
        ((nnth (nkfilter P (nnth (nkfilter P 0 nell) i) (ndropn (nnth (nkfilter P 0 nell) i) nell)) (j - i)) )
  proof -
  have f1:  $\bigwedge n$  ns. (nfirst (ndropn n ns)::nat) = nlast (ntaken n ns)
  by (metis (lifting) ndropn-0 ndropn-nfirst ndropn-nnth ntaken-ndropn-nlast)
  have  $\bigwedge n$  f ns. nlast (ntaken n (nmap f ns)) = (f (nlast (ntaken n ns)::nat)::nat)
  by simp
  then show ?thesis
    using f1 by (metis assms(1) in-nset-conv-nnth ndropn-0 ndropn-nfirst nkfilter-n-zero zero-enat-def
zero-le)
  qed
have 5: ((nnth (nkfilter P (nnth (nkfilter P 0 nell) i) (ndropn (nnth (nkfilter P 0 nell) i) nell)) (j - i)) )
=
        (nnth (nkfilter P 0 nell) j)
using assms nkfilter-nkfilter-ndropn[of nell P i]
  by (metis in-nset-conv-nnth le-add-diff-inverse linorder-le-cases linorder-not-less ndropn-nnth
nfinite-ntaken nkfilter-nlength nnth-beyond nset-nlast ntaken-all)
have 10:  $\exists x \in \text{nset}$  (ndropn (nnth (nkfilter P 0 nell) i) nell). P x
  by (metis assms(1) in-nset-conv-nnth le-zero-eq nle-le nnth-zero-ndropn zero-enat-def)
have 11: ndropn i (nfilter P nell) = nfilter P (ndropn (nnth (nkfilter P 0 nell) i) nell)
  by (simp add: assms(1) assms(2) nfilter-nkfilter-ndropn)
have 12: nlength (ndropn i (nfilter P nell)) = nlength (nfilter P (ndropn (nnth (nkfilter P 0 nell) i) nell))
  by (simp add: 11)
have 13: nlength (nkfilter P 0 (ndropn (nnth (nkfilter P 0 nell) i) nell)) =
        nlength (nfilter P (ndropn (nnth (nkfilter P 0 nell) i) nell))
  by (simp add: 10 nkfilter-nlength)
have 14: enat i  $\leq$  nlength (nfilter P nell)
  by (metis assms(1) assms(2) in-nset-conv-nnth linorder-le-cases nfinite-ntaken nkfilter-nlength nset-nlast
ntaken-all ntaken-nlast)
have 15: enat j  $\leq$  nlength (nfilter P nell)
  by (metis 14 assms(1) assms(4) diff-zero nfilter-nkfilter-ntaken-nlength-0 nkfilter-nlength)
have 16: enat (j - i)  $\leq$  nlength (ndropn i (nfilter P nell))
  by (metis 15 enat-minus-mono1 idiff-enat-enat ndropn-nlength)
have 2: ntaken (j - i) (nfilter P (ndropn (nnth (nkfilter P 0 nell) i) nell)) =
        (nfilter P (ntaken (nnth (nkfilter P 0 (ndropn (nnth (nkfilter P 0 nell) i) nell)) (j - i))
        (ndropn (nnth (nkfilter P 0 nell) i) nell)))
  by (metis 10 12 13 16 nfilter-nkfilter-ntaken-1)
have 3: (nfilter P (ntaken (nnth (nkfilter P 0 (ndropn (nnth (nkfilter P 0 nell) i) nell)) (j - i))
        (ndropn (nnth (nkfilter P 0 nell) i) nell))) =
        (nfilter P (nsubn nell
        (nnth (nkfilter P 0 nell) i)
        ((nnth (nkfilter P 0 (ndropn (nnth (nkfilter P 0 nell) i) nell)) (j - i)) +
        (nnth (nkfilter P 0 nell) i))
        ))
  by (simp add: ntaken-ndropn)
show ?thesis using 0 1 2 3 4 5 by auto
qed

```

lemma *nfilter-nkfilter-nsubn-zero*:

assumes P ($nnth$ $nell$ ($(nnth$ ($nkfilter$ P 0 $nell$) j)))
 $enat$ $j \leq nlength$ ($nkfilter$ P 0 $nell$)
shows $nsubn$ ($nfilter$ P $nell$) 0 $j =$
 $(nfilter$ P ($nsubn$ $nell$
 0
 $(nnth$ ($nkfilter$ P 0 $nell$) j)
 $)$
 $)$

by (*simp add: assms(1) assms(2) nfilter-nkfilter-ntaken nsubn-def1*)

lemma *nkfilter-nnth-aa*:

assumes $\exists x \in nset$ $nell. P$ x
 $n \leq nlength$ ($nfilter$ P $nell$)
shows P ($nnth$ ($nfilter$ P $nell$) n)
using *assms in-nset-conv-nnth nkfilter-holds[of nell P - 0] nkfilter-nfilter[of nell P - 0]*
by (*metis nkfilter-nlength*)

lemma *nfilter-nlength-zero-conv-a*:

assumes $\exists x \in nset$ $nell. P$ x
 $nlength(nfilter$ P $nell$) $= 0$
shows $(\exists k \leq nlength$ $nell. P$ ($nnth$ $nell$ k) \wedge
 $(\forall j \leq nlength$ $nell. j \neq k \longrightarrow \neg P$ ($nnth$ $nell$ j)))
using *assms*
apply *transfer*
proof (*auto simp add: epred-llength min-def split: if-split-asm*)
fix $nella :: 'a$ *llist* **and** $Pa :: 'a \Rightarrow bool$ **and** $x :: 'a$
assume $a1: \neg lnull$ $nella$
assume $a2: Pa$ x
assume $a3: x \in lset$ $nella$
assume $a4: lnull$ ($lfilter$ Pa $nella$)
show $\exists k. (enat$ $k \leq llength$ (ltl $nella$) \longrightarrow
 Pa ($lnth$ $nella$ k) $\wedge (\forall j. enat$ $j \leq llength$ (ltl $nella$) $\longrightarrow j \neq k \longrightarrow \neg Pa$ ($lnth$ $nella$ j))) \wedge
 $enat$ $k \leq llength$ (ltl $nella$)
proof –
have $1: LCons$ (lhd $nella$) (ltl $nella$) $= nella$
by (*metis a2 a3 lfilter-LNil llist.disc(1) llist.exhaust-sel lnull-lfilter*)
have $2: llength$ $nella = eSuc$ ($llength$ (ltl $nella$))
by (*metis 1 llength-LCons*)
have $3: \neg lnull$ ($lfilter$ Pa $nella$)
by (*meson a2 a3 lnull-lfilter*)
show *?thesis*
by (*metis 2 3 a4 iless-Suc-eq lfilter-llength-one-conv-a lhd-LCons-ltl llength-LCons llength-LNil*
 l *list.collapse(1) one-eSuc*)
qed
qed

lemma *nfilter-nlength-zero-conv-c:*

$$(\exists k \leq \text{nlength } \text{nell}. P (\text{nnth } \text{nell } k) \wedge (\forall j \leq \text{nlength } \text{nell}. j \neq k \longrightarrow \neg P (\text{nnth } \text{nell } j))) =$$

$$(\exists k \leq \text{nlength } \text{nell}. P (\text{nnth } \text{nell } k) \wedge (\forall j \leq \text{nlength } \text{nell}. j < k \vee k < j \longrightarrow \neg P (\text{nnth } \text{nell } j)))$$

using *antisym-conv3* **by** *auto*

lemma *nfilter-nlength-zero-conv-d:*

$$(\exists k \leq \text{nlength } \text{nell}. P (\text{nnth } \text{nell } k) \wedge (\forall j \leq \text{nlength } \text{nell}. j < k \vee k < j \longrightarrow \neg P (\text{nnth } \text{nell } j))) \longleftrightarrow$$

$$(\exists k \leq \text{nlength } \text{nell}. P (\text{nnth } \text{nell } k) \wedge (\forall j. j < k \longrightarrow \neg P (\text{nnth } \text{nell } j)) \wedge (\forall j \leq \text{nlength } \text{nell}. k < j \longrightarrow \neg P (\text{nnth } \text{nell } j)))$$

by (*meson enat-ord-simps(2) le-cases le-less-trans*)

lemma *nfilter-nlength-zero-conv-b:*

assumes $(\exists k \leq \text{nlength } \text{nell}. P (\text{nnth } \text{nell } k) \wedge (\forall j \leq \text{nlength } \text{nell}. j \neq k \longrightarrow \neg P (\text{nnth } \text{nell } j)))$

shows $(\exists x \in \text{nset } \text{nell}. P x) \wedge \text{nlength}(\text{nfilter } P \text{ nell}) = 0$

using *assms*

by *transfer*

(*auto split: if-split-asm,*
metis co.enat.exhaust-sel illess-Suc-eq in-lset-conv-lnth llength-eq-0,
metis co.enat.exhaust-sel co.enat.sel(2) illess-Suc-eq lfilter-llength-one-conv-b llength-eq-0
one-eSuc)

lemma *nfilter-nlength-zero-conv:*

$$((\exists x \in \text{nset } \text{nell}. P x) \wedge \text{nlength}(\text{nfilter } P \text{ nell}) = 0) \longleftrightarrow$$

$$(\exists k \leq \text{nlength } \text{nell}. P (\text{nnth } \text{nell } k) \wedge (\forall j \leq \text{nlength } \text{nell}. j \neq k \longrightarrow \neg P (\text{nnth } \text{nell } j)))$$

using *nfilter-nlength-zero-conv-a[of nell P] nfilter-nlength-zero-conv-b[of nell P]*

by *blast*

lemma *nfilter-nlength-zero-conv-1:*

$$((\exists x \in \text{nset } \text{nell}. P x) \wedge \text{nlength}(\text{nfilter } P \text{ nell}) = 0) \longleftrightarrow$$

$$(\exists k \leq \text{nlength } \text{nell}. P (\text{nnth } \text{nell } k) \wedge (\forall j \leq \text{nlength } \text{nell}. j < k \vee k < j \longrightarrow \neg P (\text{nnth } \text{nell } j)))$$

using *nfilter-nlength-zero-conv[of nell P] nfilter-nlength-zero-conv-c[of nell P]*

by *blast*

lemma *nfilter-nlength-zero-conv-2:*

$$((\exists x \in \text{nset } \text{nell}. P x) \wedge \text{nlength}(\text{nfilter } P \text{ nell}) = 0) \longleftrightarrow$$

$$(\exists k \leq \text{nlength } \text{nell}. P (\text{nnth } \text{nell } k) \wedge (\forall j. j < k \longrightarrow \neg P (\text{nnth } \text{nell } j)) \wedge (\forall j \leq \text{nlength } \text{nell}. k < j \longrightarrow \neg P (\text{nnth } \text{nell } j)))$$

using *nfilter-nlength-zero-conv-1[of nell P] nfilter-nlength-zero-conv-d[of nell P]*

by *blast*

lemma *nfilter-ndropns-nmap-help-0:*

```

assumes  $\exists x \in \text{nset } \text{nell}. P x$ 
            $j \leq \text{nnth}(\text{nkfilter } P \ 0 \ \text{nell}) \ 0$ 
shows  $(\text{nfilter } P \ (\text{ndropn} \ (\text{nnth} \ (\text{nkfilter } P \ 0 \ \text{nell}) \ 0) \ \text{nell})) = (\text{nfilter } P \ (\text{ndropn } j \ \text{nell}))$ 
using assms
proof (induction j arbitrary: nell)
case 0
then show ?case
by (metis ndropn-0 nfilter-nkfilter-ndropn-1 zero-enat-def zero-le)
next
case (Suc j)
then show ?case
proof (cases nell)
case (NNil x1)
then show ?thesis
by simp
next
case (NCons x21 x22)
then show ?thesis
proof –
  have 1: is-NNil x22  $\implies$ 
     $\text{nfilter } P \ (\text{ndropn} \ (\text{nnth} \ (\text{nkfilter } P \ 0 \ \text{nell}) \ 0) \ \text{nell}) = \text{nfilter } P \ (\text{ndropn} \ (\text{Suc } j) \ \text{nell})$ 
    by (metis NCons Suc.premis(2) Suc-le-D ndropn-Suc-NCons ndropn-is-NNil)
  have 2:  $P \ x21 \wedge \neg \text{is-NNil } x22 \implies$ 
     $\text{nfilter } P \ (\text{ndropn} \ (\text{nnth} \ (\text{nkfilter } P \ 0 \ \text{nell}) \ 0) \ \text{nell}) = \text{nfilter } P \ (\text{ndropn} \ (\text{Suc } j) \ \text{nell})$ 
    using Suc NCons
    by simp
    (metis Suc.premis(1) dual-order.strict-trans1 nkfilter-not-before nnth-0 zero-less-Suc)
  have 3:  $\neg P \ x21 \wedge \neg \text{is-NNil } x22 \implies$ 
     $\text{nfilter } P \ (\text{ndropn} \ (\text{nnth} \ (\text{nkfilter } P \ 0 \ \text{nell}) \ 0) \ \text{nell}) = \text{nfilter } P \ (\text{ndropn} \ (\text{Suc } j) \ \text{nell})$ 
    using Suc NCons by (simp add: nkfilter-nleast)
  show ?thesis
    using 1 2 3 by blast
qed
qed
qed

```

lemma *nfilter-nappend-ntaken:*

```

assumes  $\exists x \in \text{nset} \ (\text{ntaken } k \ \text{nell}). P x$ 
            $k \leq \text{nlength } \text{nell}$ 
shows  $\text{nfilter } P \ (\text{ntaken } k \ \text{nell}) =$ 
     $\text{ntaken} \ (\text{the-enat} \ (\text{nlength}(\text{nfilter } P \ (\text{ntaken } k \ \text{nell})))) \ (\text{nfilter } P \ \text{nell})$ 
using assms
apply transfer
proof auto
show  $\bigwedge k \ \text{nell} \ P \ x.$ 
   $\neg \text{lnull } \text{nell} \implies$ 
   $\text{enat } k \leq \text{epred} \ (\text{llength } \text{nell}) \implies$ 
   $x \in \text{lset} \ (\text{ltake} \ (\text{enat} \ (\text{Suc } k)) \ \text{nell}) \implies$ 
   $P \ x \implies$ 
   $\forall x \in \text{lset } \text{nell}. \neg P \ x \implies$ 

```

```

    lfilter P (ltake (enat (Suc k)) nell) =
      ltake (enat (Suc (the-enat (epred (llength (lfilter P (ltake (enat (Suc k)) nell)))))) nell
  by (metis dual-order.strict-trans1 in-lset-conv-lnth llength-ltake lprefix-lnthD ltake-is-lprefix
      min.cobounded2)
next
fix k
fix nell :: 'a llist
fix P
fix x
fix xa
assume a0:  $\neg$  lnull nell
assume a1:  $\text{enat } k \leq \text{epred } (\text{llength } nell)$ 
assume a2:  $x \in \text{lset } (\text{ltake } (\text{enat } (\text{Suc } k)) \text{ nell})$ 
assume a3:  $P \ x$ 
assume a4:  $xa \in \text{lset } nell$ 
assume a5:  $P \ xa$ 
show lfilter P (ltake (enat (Suc k)) nell) =
  ltake (enat (Suc (the-enat (epred (llength (lfilter P (ltake (enat (Suc k)) nell)))))) (lfilter P nell)
proof (cases k = epred (llength nell))
case True
then show ?thesis
proof -
have f1:  $\text{eSuc } (\text{enat } k) = \text{llength } nell$ 
by (simp add: True a0)
then have llength (lfilter P nell)  $\neq 0$ 
by (metis (no-types) a2 a3 eSuc-enat llength-eq-0 lnull-lfilter ltake-all order-refl)
then show ?thesis
using f1 by (metis True co.enat.exhaust-sel eSuc-enat enat-the-enat epred-le-epredI infinity-ileE
  llength-lfilter-ile ltake-all order-refl)
qed
next
case False
then show ?thesis
proof -
have f1:  $\text{llength } nell \neq 0$ 
using a0 llength-eq-0 by blast
have g1:  $\neg \text{lnull } (\text{lfilter } P (\text{ltake } (\text{enat } (\text{Suc } k)) \text{ nell}))$ 
by (meson a2 a3 lnull-lfilter)
then show ?thesis
using f1
proof -
have f1:  $\text{enat } k < \text{epred } (\text{llength } nell)$ 
using False a1 order.not-eq-order-implies-strict by blast
have f2:  $\forall e. e = 0 \vee e = \text{eSuc } (\text{epred } e)$ 
by (metis co.enat.exhaust-sel)
then have g2:  $\text{enat } (\text{Suc } k) < \text{llength } nell$ 
using f1 by (metis (no-types) Suc-ile-eq ‹llength nell  $\neq 0$ › ileSS-Suc-eq)
then show ?thesis
using f2
using lfilter-lappend-ltake[of Suc k]

```

```

proof –
  have  $eSuc\ (enat\ k) = \min\ (enat\ (Suc\ k))\ (llength\ nell)$ 
  by  $(metis\ \langle enat\ (Suc\ k) < llength\ nell \rangle\ eSuc-enat\ min.strict-order-iff)$ 
  then have  $eSuc\ (enat\ k) = llength\ (ltake\ (enat\ (Suc\ k))\ nell)$ 
  by simp
  then show ?thesis
  by  $(metis\ g1\ g2\ co.enat.exhaust-sel\ eSuc-enat\ eSuc-infinity\ enat-the-enat\ infinity-ileE$ 
     $lfilter-lappend-ltake\ llength-eq-0\ llength-lfilter-ile)$ 
qed
qed
qed
qed
qed

```

```

lemma nappend-nfilter-nfinite:
  assumes  $\exists x \in nset\ (nellx). P\ x$ 
     $\exists x \in nset\ (nelly). P\ x$ 
    nfinite nellx
  shows  $nfilter\ P\ (nappend\ nellx\ nelly) = nappend\ (nfilter\ P\ nellx)\ (nfilter\ P\ nelly)$ 
using assms
by transfer auto

```

```

lemma nfilter-nappend-ndropn:
  assumes  $\exists x \in nset\ (ndropn\ (Suc\ k)\ nell). P\ x$ 
     $\exists x \in nset\ (ntaken\ k\ nell). P\ x$ 
     $(Suc\ k) \leq nlength\ nell$ 
  shows  $nfilter\ P\ (ndropn\ (Suc\ k)\ nell) =$ 
     $ndropn\ (the-enat\ (eSuc(nlength(nfilter\ P\ (ntaken\ k\ nell)))))\ (nfilter\ P\ nell)$ 
proof –
  have 1: nfinite  $(nfilter\ P\ (ntaken\ k\ nell))$ 
    by  $(metis\ Suc-ile-eq\ assms(3)\ dual-order.strict-implies-order\ enat-ile\ length-nfilter-le$ 
       $min.absorb1\ nfinite-conv-nlength-enat\ ntaken-nlength)$ 
  have 2:  $nfilter\ P\ nell = nappend\ (nfilter\ P\ (ntaken\ k\ nell))\ (nfilter\ P\ (ndropn\ (Suc\ k)\ nell))$ 
    using assms nappend-nfilter-nfinite[of  $(ntaken\ k\ nell)\ P\ (ndropn\ (Suc\ k)\ nell)$ ]
      nappend-ntaken-ndropn[of  $k\ nell$ ] nfinite-ntaken[of  $k\ nell$ ] by simp
  have 3:  $ndropn\ (the-enat\ (eSuc(nlength(nfilter\ P\ (ntaken\ k\ nell)))))$ 
     $(nappend\ (nfilter\ P\ (ntaken\ k\ nell))\ (nfilter\ P\ (ndropn\ (Suc\ k)\ nell)))$ 
     $= (nfilter\ P\ (ndropn\ (Suc\ k)\ nell))$ 
    by  $(metis\ 1\ antisym-conv2\ gr-zeroI\ ile-eSuc\ ile-iless-Suc-eq\ less-imp-le\ ndropn-0$ 
       $ndropn-nappend3\ nfinite-conv-nlength-enat\ one-enat-def\ plus-1-eSuc(2)\ plus-enat-simps(1)$ 
       $the-enat.simps\ zero-less-diff)$ 
  show ?thesis
    by  $(simp\ add: 2\ 3)$ 
qed

```

```

lemma nkfilter-nappend-ntaken:
  assumes  $\exists x \in nset\ (ntaken\ k\ nell). P\ x$ 
     $k \leq nlength\ nell$ 

```



```

shows  $nkfilter\ P\ n\ (ntaken\ k\ nell) =$ 
   $ntaken\ (the-enat\ (llength\ (nkfilter\ P\ n\ (ntaken\ k\ nell))))\ (nkfilter\ P\ n\ nell)$ 
using assms
apply transfer
proof (auto simp add: kfilter-not-lnull-conv kfilter-lnull-conv)
show  $\bigwedge k\ nell\ P\ n\ x.$ 
   $\neg\ lnull\ nell \implies$ 
   $enat\ k \leq epred\ (llength\ nell) \implies$ 
   $x \in lset\ (ltake\ (enat\ (Suc\ k))\ nell) \implies$ 
   $P\ x \implies$ 
   $\forall x \in lset\ nell. \neg\ P\ x \implies$ 
   $kfilter\ P\ n\ (ltake\ (enat\ (Suc\ k))\ nell) =$ 
   $ltake\ (enat\ (Suc\ (the-enat\ (epred\ (llength\ (kfilter\ P\ n\ (ltake\ (enat\ (Suc\ k))\ nell))))))\ (iterates\ Suc\ n)$ 
by (meson lset-ltake subsetD)
next
fix  $ka :: nat$  and  $nella :: 'a\ llist$  and  $Pa :: 'a \Rightarrow bool$  and  $na :: nat$  and  $x :: 'a$  and  $xa :: 'a$ 
assume  $a1: Pa\ x$ 
assume  $a2: Pa\ xa$ 
assume  $a3: xa \in lset\ nella$ 
assume  $a4: x \in lset\ (ltake\ (enat\ (Suc\ ka))\ nella)$ 
have  $f5: \bigwedge e. e = enat\ 0 \vee eSuc\ (epred\ e) = e$ 
by (metis (no-types) co.enat.exhaust-sel zero-enat-def)
have  $f6: \bigwedge as. min\ \infty\ (llength\ (as :: 'a\ llist)) = llength\ as$ 
by simp
have  $f7: eSuc\ \infty = \infty$ 
by simp
have  $f8: (enat\ (Suc\ (the-enat\ (epred\ (llength\ (kfilter\ Pa\ na\ (ltake\ (enat\ (Suc\ ka))\ nella)))))) =$ 
   $(llength\ (kfilter\ Pa\ na\ (ltake\ (enat\ (Suc\ ka))\ nella)))$ 
proof –
  have  $f1: \forall n. \neg\ \infty \leq enat\ n$ 
  by (meson infinity-ileE)
  have  $\neg\ lnull\ (kfilter\ Pa\ na\ (ltake\ (enat\ (Suc\ ka))\ nella))$ 
  by (metis (no-types) a1 a4 kfilter-not-lnull-conv)
  then show ?thesis
  using  $f1$ 
  by (metis co.enat.exhaust-sel dual-order.trans eSuc-enat eSuc-ile-mono enat-the-enat kfilter-llength
    llength-eq-0 llength-lfilter-ile llength-ltake min.cobounded1)
qed
moreover
{ assume  $llength\ nella \neq \infty$ 
  then have  $ltake\ (enat\ (Suc\ ka))\ nella = nella \longrightarrow$ 
     $ltake\ (llength\ (lfilter\ Pa\ (ltake\ (enat\ (Suc\ ka))\ nella)))\ (kfilter\ Pa\ na\ nella) =$ 
     $kfilter\ Pa\ na\ (ltake\ (enat\ (Suc\ ka))\ nella) \wedge$ 
     $epred\ (llength\ (lfilter\ Pa\ (ltake\ (enat\ (Suc\ ka))\ nella))) \neq \infty$ 
  using  $f7\ f6\ f5\ a3\ a2$  by (metis (no-types) kfilter-llength llength-eq-0 llength-lfilter-ile
    lnull-lfilter ltake-all min.orderE not-le-imp-less not-less-iff-gr-or-eq zero-enat-def)
moreover
{ assume  $ltake\ (enat\ (Suc\ ka))\ nella \neq nella$ 
  then have  $enat\ (Suc\ ka) < llength\ nella \wedge min\ (enat\ (Suc\ ka))\ (llength\ nella) \neq \infty \vee$ 
     $enat\ (Suc\ ka) < llength\ nella \wedge llength\ (lfilter\ Pa\ (ltake\ (enat\ (Suc\ ka))\ nella)) \neq eSuc\ \infty$ 

```

```

  by (metis (no-types) enat-ord-code(4) ltake-all min.orderE not-le-imp-less not-less-iff-gr-or-eq) }
ultimately have enat (Suc ka) < llength nella  $\wedge$  min (enat (Suc ka)) (llength nella)  $\neq$   $\infty \vee$ 
  enat (Suc ka) < llength nella  $\wedge$  llength (lfilter Pa (ltake (enat (Suc ka)) nella))  $\neq$  eSuc  $\infty \vee$ 
  ltake (llength (lfilter Pa (ltake (enat (Suc ka)) nella))) (kfilter Pa na nella) =
  kfilter Pa na (ltake (enat (Suc ka)) nella)  $\wedge$ 
  epred (llength (lfilter Pa (ltake (enat (Suc ka)) nella)))  $\neq$   $\infty$ 
by fastforce }
ultimately show kfilter Pa na (ltake (enat (Suc ka)) nella) =
  ltake (enat (Suc (the-enat (epred (llength (kfilter Pa na (ltake (enat (Suc ka)) nella)))))))
  (kfilter Pa na nella)
using f6 f5 a4 a1
kfilter-lappend-ltake[of (enat (Suc ka)) nella Pa na ]
by (metis enat-ord-code(4) kfilter-llength)
qed

```

lemma *nfilter-ndropns-nmap-help-1:*

```

assumes  $\exists x \in \text{nset } nell. P x$ 
   $j \leq \text{nnth}(\text{nkfilter } P \ 0 \ nell) \ (Suc \ 0)$ 
   $\text{nnth}(\text{nkfilter } P \ 0 \ nell) \ 0 < j$ 
   $(Suc \ 0) \leq \text{nlength}(\text{nfilter } P \ nell)$ 
shows  $(\text{nfilter } P \ (\text{ndropn} (\text{nnth} (\text{nkfilter } P \ 0 \ nell) \ (Suc \ 0)) \ nell)) =$ 
   $(\text{nfilter } P \ (\text{ndropn } j \ nell))$ 

```

using *assms*

proof

(induction j arbitrary: nell)

case 0

then show ?case

by blast

next

case (Suc j)

then show ?case

proof (cases nell)

case (NNil x1)

then show ?thesis

by auto

next

case (NCons x21 x22)

then show ?thesis

proof –

have 1: *is-NNil x22* \implies

$\text{nfilter } P \ (\text{ndropn} (\text{nnth} (\text{nkfilter } P \ 0 \ nell) \ (Suc \ 0)) \ nell) =$
 $\text{nfilter } P \ (\text{ndropn} \ (Suc \ j) \ nell)$

by (metis NCons Suc.prem(2) Suc-le-D ndropn-Suc-NCons ndropn-is-NNil)

have 2: $P \ x21 \wedge \neg \text{is-NNil } x22 \implies$

$\text{nfilter } P \ (\text{ndropn} (\text{nnth} (\text{nkfilter } P \ 0 \ nell) \ (Suc \ 0)) \ nell) =$
 $\text{nfilter } P \ (\text{ndropn} \ (Suc \ j) \ nell)$

using Suc NCons

proof simp

assume a1: $P \ x21 \wedge \neg \text{is-NNil } x22$

assume a2: $\text{enat} \ (Suc \ 0) \leq \text{nlength} \ (\text{nfilter } P \ (NCons \ x21 \ x22))$

```

assume a3: nell = NCons x21 x22
assume a4: Suc j ≤ nnth (nkfilter P 0 (NCons x21 x22)) (Suc 0)
have f5: ∀ as p a n. ((∃ a. (a::'a) ∈ nset as ∧ p a) ∨ ¬ p a) ∨
  nkfilter p n (NCons a as) = NNil n
by (metis (no-types) nkfilter-NCons-a)
have f6: ∀ as p n. (∀ a. (a::'a) ∉ nset as ∨ ¬ p a) ∨
  nlength (nkfilter p n as) = nlength (nfilter p as)
by (meson nkfilter-nlength)
have f7: ∀ p as. (∀ a. (a::'a) ∉ nset as ∨ ¬ p a) = (∀ a. a ∉ nset as ∨ ¬ p a)
by meson
have f8: nlength (nkfilter P 0 (NCons x21 x22)) =
  nlength (nfilter P (NCons x21 x22))
by (meson a1 nellist.set-intros(2) nkfilter-nlength)
have f9: ∀ p as. (∀ a. (a::'a) ∉ nset as ∨ ¬ p a) = (∀ a. a ∉ nset as ∨ ¬ p a)
by meson
have f10: (∃ a. a ∈ nset x22 ∧ P a) →
  nkfilter P 0 (NCons x21 x22) = NCons 0 (nkfilter P (Suc 0) x22)
using a1 by (simp add: Bex-def-raw)
then have f11: (∃ a. a ∈ nset x22 ∧ P a) →
  nnth (nkfilter P (Suc 0) x22) 0 − Suc 0 = nnth (nkfilter P 0 x22) 0
using f9 f8 f7 a2 by (metis Suc-ile-eq iless-Suc-eq nkfilter-nnth-n-zero nlength-NCons)
have f14: j < nnth (nkfilter P 0 (NCons x21 x22)) (Suc 0)
using a4 Suc-le-eq by blast
have ∃ a. a ∈ nset x22 ∧ P a
using f8 f5 a2 a1 by (metis Suc-ile-eq gr-implies-not-zero nlength-NNil)
then have (∃ a. a ∈ nset x22 ∧ P a) ∧ nfilter P (ndropn (nnth (nkfilter P 0 x22) 0) x22) =
  nfilter P (ndropn j x22)
using f14 f11 a4
proof −
have j ≤ nnth (nkfilter P 0 x22) 0
using ⟨∃ a. a ∈ nset x22 ∧ P a⟩ f10 f11 f14 by fastforce
then show ?thesis
by (simp add: Bex-def-raw ⟨∃ a. a ∈ nset x22 ∧ P a⟩ nfilter-ndropns-nmap-help-0)
qed
then show nfilter P (ndropn (nnth (nkfilter P 0 (NCons x21 x22)) (Suc 0)) (NCons x21 x22)) =
  nfilter P (ndropn j x22)
using f11 a4
by (metis One-nat-def Suc-le-D diff-Suc-1 f10 ndropn-Suc-NCons nnth-Suc-NCons)
qed
have 3: ¬P x21 ∧ ¬ is-NNil x22 ⇒
  nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc 0)) nell) =
  nfilter P (ndropn (Suc j) nell)
using Suc NCons
proof simp
assume a1: ⋀ nell. [⋀ a∈nset nell. P a; j ≤ nnth (nkfilter P 0 nell) (Suc 0);
  nnth (nkfilter P 0 nell) 0 < j; enat (Suc 0) ≤ nlength (nfilter P nell)] ⇒
  nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc 0)) nell) =
  nfilter P (ndropn j nell)
assume a2: enat (Suc 0) ≤ nlength (nfilter P x22)
assume a3: Suc j ≤ nnth (nkfilter P (Suc 0) x22) (Suc 0)

```

```

assume a4:  $\exists x \in \text{nset } x22. P x$ 
assume a5:  $\text{nnth } (\text{nkfilter } P \text{ (Suc 0) } x22) \text{ 0} < \text{Suc } j$ 
have f12:  $\text{nnth } (\text{nkfilter } P \text{ 0 } x22) \text{ (Suc 0)} =$ 
   $(\text{nnth } (\text{nkfilter } P \text{ (Suc 0) } x22) \text{ (Suc 0)}) - (\text{Suc 0})$ 
using nkfilter-nnth-n-zero[of x22 P Suc 0 Suc 0 ]
by (simp add: a2 a4 nkfilter-nlength)
then have f13:  $j \leq \text{nnth } (\text{nkfilter } P \text{ 0 } x22) \text{ (Suc 0)}$ 
using a3 by linarith
have f14:  $\text{enat } 0 \leq \text{nlength } (\text{nfilter } P \text{ } x22)$ 
using a2 by (metis One-nat-def one-enat-def order.trans zero-enat-def zero-le-one)
have f15:  $\text{nlength } (\text{nkfilter } P \text{ (Suc 0) } x22) = \text{nlength } (\text{nfilter } P \text{ } x22)$ 
by (simp add: a4 nkfilter-nlength)
then have  $\text{Suc } 0 \leq \text{nnth } (\text{nkfilter } P \text{ (Suc 0) } x22) \text{ 0}$ 
by (simp add: a4 f14 nkfilter-lowerbound)
then have  $\text{Suc } (\text{nnth } (\text{nkfilter } P \text{ 0 } x22) \text{ 0}) \leq j$ 
by (metis a4 a5 add-Suc leD nkfilter-nleast not-less-eq-eq)
then have  $\text{nfilter } P \text{ (ndropn } (\text{nnth } (\text{nkfilter } P \text{ 0 } x22) \text{ (Suc 0)}) \text{ } x22) =$ 
   $\text{nfilter } P \text{ (ndropn } j \text{ } x22)$ 
by (simp add: Suc.IH Suc-le-lessD a2 a4 f13)
then show  $\text{nfilter } P \text{ (ndropn } (\text{nnth } (\text{nkfilter } P \text{ (Suc 0) } x22) \text{ (Suc 0)}) \text{ (NCons } x21 \text{ } x22)) =$ 
   $\text{nfilter } P \text{ (ndropn } j \text{ } x22)$ 
using ndropn-Suc-NCons
by (metis One-nat-def a2 a4 f12 f15 le-add-diff-inverse nkfilter-lowerbound plus-1-eq-Suc)
qed
show ?thesis
using 1 2 3 by blast
qed
qed
qed

lemma nfilter-ndropns-nmap-help-j:
assumes  $\exists x \in \text{nset } \text{nell}. P x$ 
   $j \leq \text{nnth}(\text{nkfilter } P \text{ 0 } \text{nell}) \text{ (Suc } i)$ 
   $\text{nnth } (\text{nkfilter } P \text{ 0 } \text{nell}) \text{ } i < j$ 
   $(\text{Suc } i) \leq \text{nlength}(\text{nfilter } P \text{ } \text{nell})$ 
shows  $(\text{nfilter } P \text{ (ndropn } (\text{nnth } (\text{nkfilter } P \text{ 0 } \text{nell}) \text{ (Suc } i)) \text{ } \text{nell})) =$ 
   $(\text{nfilter } P \text{ (ndropn } j \text{ } \text{nell}))$ 
using assms
proof (induction j arbitrary: nell i)
case 0
then show ?case
by blast
next
case (Suc j)
then show ?case
  proof (cases nell)
    case (NNil x1)
    then show ?thesis
    by simp
  next

```

```

case (NCons x21 x22)
then show ?thesis
proof -
  have 1: is-NNil x22  $\implies$ 
    nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc i)) nell) =
    nfilter P (ndropn (Suc j) nell)
  by (metis NCons Suc.prem2(2) Suc-le-D ndropn-Suc-NCons ndropn-is-NNil)
  have 2:  $\neg$  is-NNil x22  $\wedge$  i = 0  $\implies$ 
    nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc i)) nell) =
    nfilter P (ndropn (Suc j) nell)
  using Suc nfilter-ndropns-nmap-help-1[of nell P] by metis
  have 3: P x21  $\wedge$   $\neg$  is-NNil x22  $\wedge$  0 < i  $\implies$ 
    nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc i)) nell) = nfilter P (ndropn (Suc j) nell)
  using Suc NCons
  proof simp
    assume a1:  $\bigwedge$  nell i.  $\llbracket \exists a \in \text{nset } nell. P a; j \leq \text{nnth } (nkfilter P 0 nell) (Suc i);$ 
       $\text{nnth } (nkfilter P 0 nell) i < j; \text{enat } (Suc i) \leq \text{nlength } (nfilter P nell) \rrbracket \implies$ 
      nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc i)) nell) =
      nfilter P (ndropn j nell)
    assume a2: P x21  $\wedge$   $\neg$  is-NNil x22  $\wedge$  0 < i
    assume a3:  $\text{enat } (Suc i) \leq \text{nlength } (nfilter P (NCons x21 x22))$ 
    assume a4: nell = NCons x21 x22
    assume a5: Suc j  $\leq$  nnth (nkfilter P 0 (NCons x21 x22)) (Suc i)
    assume a6: nnth (nkfilter P 0 (NCons x21 x22)) i < Suc j
    show nfilter P (ndropn (nnth (nkfilter P 0 (NCons x21 x22)) (Suc i)) (NCons x21 x22)) =
      nfilter P (ndropn j x22)
  proof -
    have f0:  $\exists a \in \text{nset } x22. P a$ 
    by (metis a2 a3 dual-order.antisym enat.inject nat.distinct(1) nfilter-NCons-a
      nlength-NNil zero-enat-def zero-le)
    have f1: nlength (nkfilter P 0 (NCons x21 x22)) = nlength (nfilter P (NCons x21 x22))
    using a4 by (metis (no-types) Suc(2) nkfilter-nlength)
    have f2: nkfilter P 0 (NCons x21 x22) = NCons 0 (nkfilter P (Suc 0) x22)
    by (metis a2 a3 dual-order.antisym enat.inject f1 nat.distinct(1) nkfilter-NCons
      nkfilter-NCons-a nlength-NNil zero-enat-def zero-le)
    have f3: nnth (nkfilter P 0 (NCons x21 x22)) (Suc i) =
      nnth ( (nkfilter P (Suc 0) x22)) (i)
    by (simp add: f2)
    have f4:  $\text{enat } i \leq \text{nlength } (nkfilter P (Suc 0) x22)$ 
    by (metis Suc-ile-eq a3 f1 f2 illess-Suc-eq nlength-NCons)
    have f5: nnth ( (nkfilter P (Suc 0) x22)) (i) = (Suc 0) + nnth ( (nkfilter P 0 x22)) (i)
    using nkfilter-nnth-n-zero[of x22 P i Suc 0]
    using a5 f0 f3 f4 by linarith
    have f6: ndropn (nnth (nkfilter P 0 (NCons x21 x22)) (Suc i)) (NCons x21 x22) =
      ndropn ((Suc 0) + nnth ( (nkfilter P 0 x22)) (i)) (NCons x21 x22)
    using f3 f5 by presburger
    have f7: ndropn ((Suc 0) + nnth ( (nkfilter P 0 x22)) (i)) (NCons x21 x22) =
      ndropn (nnth ( (nkfilter P 0 x22)) (i)) x22
    using ndropn-Suc-NCons by auto
    have f8: j  $\leq$  nnth (nkfilter P 0 x22) (i)

```

```

    using a5 f3 f5 by linarith
have f11: nnth (nkfilter P 0 (NCons x21 x22)) i = nnth ( (nkfilter P (Suc 0) x22)) (i-1)
  by (metis One-nat-def Suc-pred a2 f2 nnth-Suc-NCons)
have f12: enat (i - 1) ≤ nlength (nkfilter P (Suc 0) x22)
  by (metis One-nat-def Suc-ile-eq Suc-pred a2 f4 less-imp-le)
have f13: (Suc 0) ≤ nnth ( (nkfilter P (Suc 0) x22)) (i-1)
  by (meson f0 f12 nkfilter-lowerbound)
have f14: nnth ( (nkfilter P (Suc 0) x22)) (i-1) = (Suc 0) + nnth ( (nkfilter P 0 x22)) (i-1)
  using nkfilter-nnth-n-zero[of x22 P i-1 Suc 0 ] f12 f0 f13 by (metis le-add-diff-inverse)
have f9: nnth (nkfilter P 0 x22) (i-1) < j
  using a6 f11 f14 by linarith
have f10: i ≤ nlength (nfilter P x22)
  by (metis f0 f4 nkfilter-nlength)
show ?thesis using a1[of x22 i-1]
  by (metis Suc-diff-1 a2 f0 f10 f3 f5 f7 f8 f9)
qed
qed
have 4: ¬P x21 ∧ ¬ is-NNil x22 ∧ 0 < i ⇒
  nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc i)) nell) =
  nfilter P (ndropn (Suc j) nell)
using Suc NCons
proof simp
  assume a0: ¬ P x21 ∧ ¬ is-NNil x22 ∧ 0 < i
  assume a2: ∧ nell i. [∃ a ∈ nset nell. P a; j ≤ nnth (nkfilter P 0 nell) (Suc i);
    nnth (nkfilter P 0 nell) i < j; enat (Suc i) ≤ nlength (nfilter P nell)] ⇒
    nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc i)) nell) =
    nfilter P (ndropn j nell)
  assume a1: ∃ x ∈ nset x22. P x
  assume a4: Suc j ≤ nnth (nkfilter P (Suc 0) x22) (Suc i)
  assume a5: nnth (nkfilter P (Suc 0) x22) i < Suc j
  assume a3: enat (Suc i) ≤ nlength (nfilter P x22)
  assume a6: nell = NCons x21 x22
  show nfilter P (ndropn (nnth (nkfilter P (Suc 0) x22) (Suc i)) (NCons x21 x22)) =
    nfilter P (ndropn j x22)
proof -
  have f1: nlength (nkfilter P 0 (NCons x21 x22)) = nlength (nfilter P (NCons x21 x22))
  by (metis NCons Suc.prem1 nkfilter-nlength)
  have f2: nkfilter P 0 (NCons x21 x22) = (nkfilter P (Suc 0) x22)
  using NCons Suc.prem2 a1 a5 by auto
  have f3: nnth (nkfilter P 0 (NCons x21 x22)) (Suc i) =
    nnth (nkfilter P (Suc 0) x22) (Suc i)
  by (simp add: f2)
  have f4: enat (Suc i) ≤ nlength (nkfilter P (Suc 0) x22)
  using NCons Suc.prem4 f1 f2 by auto
  have f5: nnth ( (nkfilter P (Suc 0) x22)) (Suc i) = (Suc 0) + nnth ( (nkfilter P 0 x22)) (Suc i)
  by (metis One-nat-def Suc-le-D a1 a4 diff-Suc-1 f4 nkfilter-nnth-n-zero plus-1-eq-Suc)
  have f6: (ndropn (nnth (nkfilter P (Suc 0) x22) (Suc i)) (NCons x21 x22)) =
    ndropn ( (Suc 0) + nnth ( (nkfilter P 0 x22)) (Suc i)) (NCons x21 x22)
  by (simp add: f5)
  have f7: ndropn ( (Suc 0) + nnth ( (nkfilter P 0 x22)) (Suc i)) (NCons x21 x22) =

```

```

      ndropn ( nnth ( (nkfilter P 0 x22)) (Suc i)) (x22)
    by simp
    have f8:  $j \leq \text{nnth } (\text{nkfilter } P \ 0 \ x22) \ (\text{Suc } i)$ 
    using a4 f5 by force
    have f11:  $\text{nnth } (\text{nkfilter } P \ 0 \ (\text{NCons } x21 \ x22)) \ i = \text{nnth } ( (\text{nkfilter } P \ (\text{Suc } 0) \ x22)) \ (i)$ 
    by (simp add: f2)
    have f12:  $\text{enat } (i) \leq \text{nlength } (\text{nkfilter } P \ (\text{Suc } 0) \ x22)$ 
    using Suc-ile-eq f4 by auto
    have f13:  $(\text{Suc } 0) \leq \text{nnth } ( (\text{nkfilter } P \ (\text{Suc } 0) \ x22)) \ (i)$ 
    by (simp add: a1 f12 nkfilter-lowerbound)
    have f14:  $\text{nnth } ( (\text{nkfilter } P \ (\text{Suc } 0) \ x22)) \ (i) = (\text{Suc } 0) + \text{nnth } ( (\text{nkfilter } P \ 0 \ x22)) \ (i)$ 
    by (metis a1 f12 f13 le-add-diff-inverse nkfilter-nnth-n-zero)
    have f9:  $\text{nnth } (\text{nkfilter } P \ 0 \ x22) \ (i) < j$ 
    using a5 f14 by auto
    show ?thesis
    using Suc.IH a1 a3 f6 f7 f8 f9 by presburger
  qed
qed
show ?thesis
using 1 2 3 4 by fastforce
qed
qed
qed

```

lemma *nfILTER-ndropns-nmap*:

```

assumes  $\exists x \in \text{nset } (\text{ndropns nell}). P \ x$ 
       $(\text{Suc } i) \leq \text{nlength}(\text{nfILTER } P \ (\text{ndropns nell}))$ 
shows  $\text{ndropn } (\text{Suc } i) \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfILTER } P \ (\text{ndropns nell}))) =$ 
       $(\text{nmap } (\lambda s. \text{nnth } s \ 0)$ 
         $(\text{nfILTER } P \ (\text{ndropns } (\text{ndropn } (\text{Suc } (\text{nnth } (\text{nkfilter } P \ 0 \ (\text{ndropns nell})) \ i)) \ \text{nell}))))$ 
proof –
  have 1:  $\text{ndropn } (\text{Suc } i) \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfILTER } P \ (\text{ndropns nell}))) =$ 
     $\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{ndropn } (\text{Suc } i) \ (\text{nfILTER } P \ (\text{ndropns nell})))$ 
  by (simp add: ndropn-nmap)
  have 2:  $(\text{Suc } (\text{nnth } (\text{nkfilter } P \ 0 \ (\text{ndropns nell})) \ i)) \leq \text{nlength } (\text{ndropns nell})$ 
  using assms nkfilter-nkfilter-ntaken[of  $(\text{ndropns nell}) \ P \ i$ ]
  by (metis Suc-ile-eq antisym-conv2 less-imp-le min.orderE nkfilter-nlength not-le-imp-less
    ntaken-all ntaken-nlength)
  have 3:  $(\text{nfILTER } P \ (\text{ndropns } (\text{ndropn } (\text{Suc } (\text{nnth } (\text{nkfilter } P \ 0 \ (\text{ndropns nell})) \ i)) \ \text{nell}))) =$ 
     $(\text{nfILTER } P \ (\text{ndropn } (\text{Suc } (\text{nnth } (\text{nkfilter } P \ 0 \ (\text{ndropns nell})) \ i)) \ (\text{ndropns nell})))$ 
  by (simp add: 2 ndropn-ndropns)
  have 4:  $(\text{ndropn } (\text{Suc } i) \ (\text{nfILTER } P \ (\text{ndropns nell}))) =$ 
     $(\text{nfILTER } P \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ (\text{ndropns nell})) \ (\text{Suc } i)) \ (\text{ndropns nell})))$ 
  by (simp add: assms(1) assms(2) nfILTER-nkfilter-ndropn-1 nkfilter-nlength)
  have 5:  $(\text{Suc } (\text{nnth } (\text{nkfilter } P \ 0 \ (\text{ndropns nell})) \ i)) \leq (\text{nnth } (\text{nkfilter } P \ 0 \ (\text{ndropns nell})) \ (\text{Suc } i))$ 
  by (simp add: Suc-leI assms(1) assms(2) nidX-nkfilter-expand nkfilter-nlength)
  have 6:  $i < \text{nlength}(\text{nfILTER } P \ (\text{ndropns nell}))$ 
  using Suc-ile-eq assms(2) by blast
  have 7:  $\text{nnth } (\text{nkfilter } P \ 0 \ (\text{ndropns nell})) \ i < (\text{Suc } (\text{nnth } (\text{nkfilter } P \ 0 \ (\text{ndropns nell})) \ i))$ 
  by simp

```

have 8: (nfilter P (ndropn (nnth (nkfilter P 0 (ndropns nell)) (Suc i)) (ndropns nell))) =
 (nfilter P (ndropn (Suc (nnth (nkfilter P 0 (ndropns nell)) i)) (ndropns nell)))
using 5 6 7 *assms*
nfilter-ndropns-nmap-help-j[of ndropns nell P (Suc (nnth (nkfilter P 0 (ndropns nell)) i)) i]
by blast
show ?thesis
using 1 3 4 8 **by** auto
qed

lemma ndropns-nfilter-nnth-exist-ndropn:
assumes $\exists x \in \text{nset } (\text{ndropns nell}). P x$
 $j \leq \text{nlength } (\text{nfilter } P (\text{ndropns nell}))$
shows $(\exists i. \text{enat } i \leq \text{nlength nell} \wedge \text{nnth } (\text{nfilter } P (\text{ndropns nell})) j = \text{ndropn } i \text{ nell})$
proof –
have 1: $\text{nnth } (\text{nfilter } P (\text{ndropns nell})) j \in \text{nset } (\text{ndropns nell})$
using *assms in-nset-conv-nnth nfilter-nnth* **by** fastforce
show ?thesis **using** 1 *using in-nset-ndropns* **by** blast
qed

lemma nfilter-ndropns-nnth-bound:
assumes $(\exists ys \in \text{nset } (\text{ndropns xs}). P ys)$
 $j \leq \text{nlength } (\text{nfilter } P (\text{ndropns xs}))$
shows $\text{nlength } ((\text{nnth } (\text{nfilter } P (\text{ndropns xs})) j)) \leq \text{nlength } xs$
using *assms ndropns-nfilter-nnth-exist-ndropn*[of xs P j]
by (metis *add commute enat.simps*(3) *enat-add-sub-same enat-le-plus-same*(1) *less-eqE ndropn-nlength*)

lemma ndropns-nfilter-ndropn:
assumes $(\text{Suc } k) \leq \text{nlength nell}$
 $\exists x \in \text{nset } (\text{ndropns } (\text{ndropn } (\text{Suc } k) \text{ nell})). P x$
 $\exists x \in \text{nset } (\text{ntaken } k (\text{ndropns nell})). P x$
shows $(\text{nfilter } P (\text{ndropns } (\text{ndropn } (\text{Suc } k) \text{ nell}))) =$
 $(\text{ndropn } (\text{the-enat } (\text{eSuc } (\text{nlength } (\text{nfilter } P (\text{ntaken } k (\text{ndropns nell})))))) (\text{nfilter } P (\text{ndropns nell})))$
using *assms*
ndropn-ndropns[of Suc k nell] *ndropns-nlength*[of nell] *nfilter-nappend-ndropn*[of k ndropns nell P]
by simp

lemma ndropns-nfilter-ndropn-a:
assumes $k \leq \text{nlength } (\text{nfilter } P (\text{ndropns nell}))$
 $\exists x \in \text{nset } (\text{ndropns nell}). P x$
shows $\text{ndropn } k (\text{nfilter } P (\text{ndropns nell})) =$
 $\text{nfilter } P (\text{ndropns } (\text{ndropn } (\text{nnth } (\text{nkfilter } P 0 (\text{ndropns nell})) k) \text{ nell}))$
using *assms ndropn-ndropns nfilter-nkfilter-ndropn-1*[of ndropns nell P k]
nkfilter-upperbound[of ndropns nell P k 0]
by (metis (mono-tags, lifting) *add.left-neutral nkfilter-nlength zero-enat-def*)

lemma nfilter-nlength-imp:
assumes $\exists x \in \text{nset nell}. P x \wedge Q x$
shows $\text{nlength } (\text{nfilter } (\lambda x. P x \wedge Q x) \text{ nell}) \leq \text{nlength } (\text{nfilter } P \text{ nell})$
using *assms* **by** transfer (auto simp *add: lfilter-nlength-imp epred-le-epredI*)


```

lemma nkfilter-chop:
assumes nlast nellx = nfirst nelly
         P (nlast nellx)
         nfinite nellx
shows nkfilter P k (nfuse nellx nelly) =
        nfuse (nkfilter P k nellx) (nkfilter P (k+(the-enat (nlength nellx))) nelly)
using assms
proof (auto simp add: nfuse-def1)
show nlast nellx = nfirst nelly  $\implies$ 
       P (nfirst nelly)  $\implies$ 
       nfinite nellx  $\implies$ 
       is-NNil nelly  $\implies$ 
        $\neg$  is-NNil (nkfilter P (k + the-enat (nlength nellx)) nelly)  $\implies$ 
       nkfilter P k nellx = nappend (nkfilter P k nellx) (ntl (nkfilter P (k + the-enat (nlength nellx)) nelly))
by transfer auto
show nlast nellx = nfirst nelly  $\implies$ 
       P (nfirst nelly)  $\implies$ 
       nfinite nellx  $\implies$   $\neg$  is-NNil nelly  $\implies$ 
       is-NNil (nkfilter P (k + the-enat (nlength nellx)) nelly)  $\implies$ 
       nkfilter P k (nappend nellx (ntl nelly)) = nkfilter P k nellx
apply transfer
proof (auto simp add: kfilter-lappend-lfinite lappend-lnull2)
fix nellxa :: 'a llist and nellya :: 'a llist and Pa :: 'a  $\Rightarrow$  bool and ka :: nat and b :: nat
assume a1: (if lnull (kfilter Pa (ka + the-enat (epred (llength nellxa)))) nellya
         then iterates Suc (ka + the-enat (epred (llength nellxa))))
         else kfilter Pa (ka + the-enat (epred (llength nellxa)))) nellya = LCons b LNil
assume a2:  $\neg$  lnull nellya
assume a3: Pa (lhd nellya)
assume a4:  $\neg$  lnull (kfilter Pa (ka + the-enat (llength nellxa)) (ltl nellya))
have f5: lnull (LNil::nat llist)
using llength-LNil llength-eq-0 by blast
have f6: nellya = LCons (lhd nellya) (ltl nellya)
using a2 by auto
then have Pa (lhd nellya)
using a3 by auto
then have False
by (metis a1 a2 a4 eq-LConsD f6 kfilter-code(2) kfilter-lnull-conv llength-LNil llength-eq-0 llist.set-sel(1))
then show lappend (kfilter Pa ka nellxa) (kfilter Pa (ka + the-enat (llength nellxa)) (ltl nellya)) = iterates
Suc ka
by meson
next
fix nellxa :: 'b llist and nellya :: 'b llist and Pa :: 'b  $\Rightarrow$  bool and ka :: nat and b :: nat
assume a1: (if lnull (kfilter Pa (ka + the-enat (epred (llength nellxa)))) nellya
         then iterates Suc (ka + the-enat (epred (llength nellxa))))
         else kfilter Pa (ka + the-enat (epred (llength nellxa)))) nellya = LCons b LNil
assume a2:  $\neg$  lnull nellya
assume a3: Pa (lhd nellya)
assume a4:  $\neg$  lnull (kfilter Pa (ka + the-enat (llength nellxa)) (ltl nellya))
have f5: lnull (LNil::nat llist)
using llength-LNil llength-eq-0 by blast

```

```

have f6: nellya = LCons (lhd nellya) (ltl nellya)
using a2 by auto
then have Pa (lhd nellya)
using a3 by auto
then have False
using f6 f5 a4 a1 by (metis kfilter-LCons kfilter-llength-n-zero llength-eq-0 ltl-simps(2) not-lnull-conv)
then show lappend (kfilter Pa ka nellxa) (kfilter Pa (ka + the-enat (llength nellxa)) (ltl nellya)) =
      kfilter Pa ka nellxa
  by meson
qed
next
show nlast nellx = nfirst nelly  $\implies$ 
  P (nfirst nelly)  $\implies$ 
  nfinite nellx  $\implies$ 
   $\neg$  is-NNil nelly  $\implies$ 
   $\neg$  is-NNil (nkfilter P (k + the-enat (nlength nellx)) nelly)  $\implies$ 
  nkfilter P k (nappend nellx (ntl nelly)) =
  nappend (nkfilter P k nellx) (ntl (nkfilter P (k + the-enat (nlength nellx)) nelly))
apply transfer
proof (auto simp add: lappend-iterates kfilter-lnull-conv split:if-split-asm)
show f1:  $\bigwedge$  nellx nelly P k x xa.
   $\neg$  lnull nellx  $\implies$ 
   $\neg$  lnull nelly  $\implies$ 
  llast nellx = lhd nelly  $\implies$ 
  P (lhd nelly)  $\implies$ 
  lfinite nellx  $\implies$ 
   $\forall b. \text{nelly} \neq \text{LCons } b \text{ LNil} \implies$ 
   $\forall b. \text{kfilter } P (k + \text{the-enat } (\text{epred } (\text{llength } \text{nellx}))) \text{ nelly} \neq \text{LCons } b \text{ LNil} \implies$ 
   $x \in \text{lset } \text{nelly} \implies P x \implies \forall x \in \text{lset } \text{nellx}. \neg P x \implies P xa \implies xa \in \text{lset } (\text{ltl } \text{nelly}) \implies$ 
  kfilter P k (lappend nellx (ltl nelly)) = iterates Suc k
by (metis in-lset-lappend-iff lappend-lbutlast-llast-id lbutlast-lfinite llist.set-intros(1))
next
show f2:  $\bigwedge$  nellx nelly P k x xa xb.
   $\neg$  lnull nellx  $\implies$ 
   $\neg$  lnull nelly  $\implies$ 
  llast nellx = lhd nelly  $\implies$ 
  P (lhd nelly)  $\implies$ 
  lfinite nellx  $\implies$ 
   $\forall b. \text{nelly} \neq \text{LCons } b \text{ LNil} \implies$ 
   $\forall b. \text{kfilter } P (k + \text{the-enat } (\text{epred } (\text{llength } \text{nellx}))) \text{ nelly} \neq \text{LCons } b \text{ LNil} \implies$ 
   $x \in \text{lset } \text{nelly} \implies$ 
   $P x \implies$ 
   $xa \in \text{lset } \text{nellx} \implies$ 
   $P xa \implies$ 
   $P xb \implies$ 
   $xb \in \text{lset } \text{nellx} \implies$ 
  kfilter P k (lappend nellx (ltl nelly)) =
  lappend (kfilter P k nellx) (ltl (kfilter P (k + the-enat (epred (llength nellx))) nelly))
proof -
fix nellxa :: 'b llist and nellya :: 'b llist and Pa :: 'b  $\Rightarrow$  bool and

```

```

    ka :: nat and x :: 'b and xa :: 'b and xb :: 'b
  assume a1: ¬ lnull nellya
  assume a2: Pa (lhd nellya)
  assume a3: llast nellxa = lhd nellya
  assume a4: ¬ lnull nellxa
  assume a5: lfinite nellxa
  have f6: nellya = LCons (lhd nellya) (ltl nellya)
  using a1 by auto
  have f7: LCons (lhd nellya) (ltl nellya) ≠ LNil ∧ lhd (LCons (lhd nellya) (ltl nellya)) = lhd nellya ∧
    ltl (LCons (lhd nellya) (ltl nellya)) = ltl nellya
  by auto
  then have f8: kfilter Pa (the-enat (epred (llength nellxa)) + ka) (LCons (lhd nellya) (ltl nellya)) =
    LCons (the-enat (epred (llength nellxa)) + ka)
      (kfilter Pa (Suc (the-enat (epred (llength nellxa)) + ka)) (ltl nellya))
  using f6 a2 by (metis (no-types) kfilter-LCons)
  have f9: ∀ bs p n bsa. ¬ lfinite (bs::'b llist) ∨ kfilter p n (lappend bs bsa) =
    lappend (kfilter p n bs) (kfilter p (n + the-enat (llength bs)) bsa)
  by (meson kfilter-lappend-lfinite)
  have f10: lfinite (lbutlast nellxa)
  using a5 by (meson lbutlast-lfinite)
  have f11: lappend (kfilter Pa ka (lbutlast nellxa))
    (kfilter Pa (ka + the-enat (llength (lbutlast nellxa))) LNil) =
    kfilter Pa ka (lbutlast nellxa)
  by simp
  have LCons (the-enat (epred (llength nellxa)) + ka) LNil =
    kfilter Pa (the-enat (epred (llength nellxa)) + ka) (LCons (lhd nellya) LNil)
  using a2 by simp
  then have f12: lappend (lappend (kfilter Pa ka (lbutlast nellxa))
    (kfilter Pa (ka + the-enat (llength (lbutlast nellxa))) LNil))
    (LCons (the-enat (epred (llength nellxa)) + ka) LNil) = kfilter Pa ka nellxa
  using f11 f10 f9 a5 a4 a3 by (metis add.commute lappend-lbutlast-llast-id-lfinite llength-lbutlast)
  have ltl (kfilter Pa (ka + the-enat (epred (llength nellxa))) nellya) =
    kfilter Pa (Suc (the-enat (epred (llength nellxa)) + ka)) (ltl nellya)
  using f8 f6 by (simp add: add.commute)
  then show kfilter Pa ka (lappend nellxa (ltl nellya)) =
    lappend (kfilter Pa ka nellxa) (ltl (kfilter Pa (ka + the-enat (epred (llength nellxa))) nellya))
  using f12 f11 f10 f9 f8 f7 f6 a5 a4 a3
  by (metis add.commute lappend-lbutlast-llast-id-lfinite lappend-snocL1-conv-LCons2 llength-lbutlast)
  qed
show ∧ nellx nelly P k x xa xb.
  ¬ lnull nellx ⇒
  ¬ lnull nelly ⇒
  llast nellx = lhd nelly ⇒
  P (lhd nelly) ⇒
  lfinite nellx ⇒
  ∀ b. nelly ≠ LCons b LNil ⇒
  ∀ b. kfilter P (k + the-enat (epred (llength nellx))) nelly ≠ LCons b LNil ⇒
  x ∈ lset nelly ⇒
  P x ⇒
  xa ∈ lset nellx ⇒

```

```

  P xa  $\implies$ 
  P xb  $\implies$ 
  xb  $\in$  lset (ltl nelly)  $\implies$ 
  kfilter P k (lappend nellx (ltl nelly)) =
  lappend (kfilter P k nellx) (ltl (kfilter P (k + the-enat (epred (llength nellx)))) nelly))
  by (simp add: f2)
qed
qed

```

lemma nleast-conv:

```

assumes  $\exists x \in \text{nset } nellx. P x$ 
shows nleast P nellx = (LEAST na. na  $\leq$  nlength nellx  $\wedge$  P (nnth nellx na))
using assms
by transfer
  (auto simp add: min-def lleast-def,
   metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0)

```

lemma nfilter-chop:

```

assumes nlast nellx = nfirst nelly
          P (nlast nellx)
          nfinite nellx
shows nfilter P (nfuse nellx nelly) = nfuse (nfilter P nellx) (nfilter P nelly)
proof (cases is-NNil nelly)
case True
then show ?thesis by (metis nfilter-NNil nfuse-def1 nfuse-leftneutral)
next
case False
then show ?thesis
  proof (cases is-NNil (nfilter P nelly))
  case True
then show ?thesis unfolding nfuse-def1 using assms nfilter-nappend2[of ntl nelly P nellx]
  nfilter-expand[of nelly P]
  by (auto simp add: nfirst-def nellist.case-eq-if )
  (metis nellist.discI(2) nellist.set-sel(2) nset-nlast)
  next
case False
then show ?thesis
  proof –
  have 1:  $\exists x \in \text{nset } nellx. P x$ 
    using assms nset-nlast by blast
  have 2:  $\exists x \in \text{nset } (ntl nelly). P x$ 
    using False assms
    by (metis nellist.collapse(1) nellist.collapse(2) nellist.disc(1) nfilter-NCons-a
      nfilter-NNil nnth-0 ntaken-0 ntaken-nlast)
  have 3:  $\neg \text{is-NNil } (nfilter P nelly) \implies$ 
    nfilter P (nappend nellx (ntl nelly)) = nappend (nfilter P nellx) (nfilter P (ntl nelly))
    by (simp add: 1 2 assms(3) nappend-nfilter-nfinite)
  have 4:  $\text{is-NNil } (nfilter P nelly) \implies nfilter P nellx = nappend (nfilter P nellx) (nfilter P (ntl nelly))$ 
    by (simp add: False)
  have 5:  $nfilter P (nfuse nellx nelly) = nfilter P (nappend nellx (ntl nelly))$ 

```

```

    unfolding nfuse-def1
  by (metis (full-types) False nellist.collapse(1) nfilter-NNil)
have 6: (ntl (nfilter P nelly)) = (nfilter P (ntl nelly))
  using assms 2 False nfilter-expand[of nelly P]
  by (metis in-nset-ntlD nellist.case-eq-if nellist.sel(5) nfirst-def)
show ?thesis
  by (metis 3 5 6 False nfuse-def1)
qed
qed
qed

```

```

lemma nfilter-chop1:
assumes  $n \leq \text{nlength } nellx$ 
         $P (\text{nlast } (\text{ntaken } n \text{ nellx}))$ 
shows  $\text{nfilter } P \text{ nellx} = \text{nfuse } (\text{nfilter } P (\text{ntaken } n \text{ nellx})) (\text{nfilter } P (\text{ndropn } n \text{ nellx}))$ 
using assms
by (metis nfuse-ntaken-ndropn ndropn-nfirst nfilter-chop nfinite-ntaken ntaken-nlast)

```

```

lemma nfilter-chop1-ntaken:
assumes  $n \leq \text{nlength } nellx$ 
         $P (\text{nlast } (\text{ntaken } n \text{ nellx}))$ 
shows  $\text{ntaken } (\text{the-enat } (\text{nlength } (\text{nfilter } P (\text{ntaken } n \text{ nellx})))) (\text{nfilter } P \text{ nellx}) =$ 
         $(\text{nfilter } P (\text{ntaken } n \text{ nellx}))$ 
using assms nfilter-nappend-ntaken by (metis nfinite-ntaken nset-nlast)

```

```

lemma nfilter-nlast:
assumes  $n \leq \text{nlength } nellx$ 
         $P (\text{nlast } (\text{ntaken } n \text{ nellx}))$ 
shows  $\text{nlast } (\text{nfilter } P (\text{ntaken } n \text{ nellx})) = (\text{nlast } (\text{ntaken } n \text{ nellx}))$ 
using assms
proof (induction n arbitrary: nellx)
case 0
then show ?case by simp
next
case (Suc n)
then show ?case
  proof (cases nellx)
  case (NNil x1)
  then show ?thesis by auto
  next
  case (NCons x21 x22)
  then show ?thesis using Suc
  by auto
  (metis Suc-ile-eq iless-Suc-eq nfilter-NCons nfinite-ntaken nlast-NCons nlength-NCons
    nset-nlast ntaken-Suc-NCons)
qed
qed

```

```

lemma nfilter-nfirst:

```

```

assumes  $P$  ( $nfirst(ndropn\ n\ nellx)$ )
shows  $nfirst(nfilter\ P\ (ndropn\ n\ nellx)) = (nfirst\ (ndropn\ n\ nellx))$ 
using assms
proof (induction n arbitrary: nellx)
case 0
then show ?case
  by (auto simp add: ndropn-nfirst)
  (metis nellist.set-intros(1) nfilter-NNil nfilter-nappend-ntaken nlast-NNil nlength-NNil
    ntaken-0 the-enat-0 zero-enat-def zero-le)
next
case (Suc n)
then show ?case
  by (metis ndropn-ndropn plus-1-eq-Suc)
qed

```

```

lemma nfilter-chop1-ndropn:
assumes  $n \leq nlength\ nellx$ 
   $P\ (nlast\ (ntaken\ n\ nellx))$ 
shows  $ndropn\ (the-enat\ (nlength(nfilter\ P\ (ntaken\ n\ nellx))))\ (nfilter\ P\ nellx) =$ 
   $(nfilter\ P\ (ndropn\ n\ nellx))$ 
proof –
  have 1:  $(nfilter\ P\ nellx) = nfuse\ (nfilter\ P\ (ntaken\ n\ nellx))\ (nfilter\ P\ (ndropn\ n\ nellx))$ 
    by (simp add: assms nfilter-chop1)
  have 2:  $nlast(nfilter\ P\ (ntaken\ n\ nellx)) = nfirst\ (nfilter\ P\ (ndropn\ n\ nellx))$ 
    by (metis assms(1) assms(2) ndropn-nfirst nfilter-nfirst nfilter-nlast ntaken-nlast)
  have 3:  $ndropn\ (the-enat\ (nlength(nfilter\ P\ (ntaken\ n\ nellx))))$ 
     $(nfuse\ (nfilter\ P\ (ntaken\ n\ nellx))\ (nfilter\ P\ (ndropn\ n\ nellx))) =$ 
     $(nfilter\ P\ (ndropn\ n\ nellx))$ 
    by (metis 2 assms(1) enat-the-enat infinity-ileE length-nfilter-le min-absorb1 ndropn-nfuse
      nfinite-conv-nlength-enat ntaken-nlength)
  show ?thesis by (simp add: 1 3)
qed

```

```

lemma nkfilter-chop1:
assumes  $(enat\ n) \leq nlength\ nellx$ 
   $P\ (nlast\ (ntaken\ n\ nellx))$ 
shows  $nkfilter\ P\ k\ nellx = nfuse\ (nkfilter\ P\ k\ (ntaken\ n\ nellx))\ (nkfilter\ P\ (k+n)\ (ndropn\ n\ nellx))$ 
using assms nfuse-ntaken-ndropn[of n nellx] nkfilter-chop[of (ntaken n nellx) (ndropn n nellx) P k]
by (metis min.orderE ndropn-nfirst nfinite-ntaken ntaken-nlast ntaken-nlength the-enat.simps)

```

```

lemma nkfilter-nlast:
assumes  $n \leq nlength\ nellx$ 
   $P\ (nlast\ (ntaken\ n\ nellx))$ 
shows  $nlast(nkfilter\ P\ k\ (ntaken\ n\ nellx)) = k+n$ 
using assms
proof (induction n arbitrary: k nellx)
case 0
then show ?case by simp
next
case (Suc n)

```

```

then show ?case
proof (cases nellx)
case (NNil x1)
then show ?thesis
using Suc.prem1 enat-0-iff(1) by auto
next
case (NCons x21 x22)
then show ?thesis
  using Suc
  proof auto
    assume a1: P (nlast (ntaken n x22))
    assume a2:  $\bigwedge nellx k. \llbracket enat\ n \leq nlength\ nellx; P\ (nlast\ (ntaken\ n\ nellx)) \rrbracket \implies$ 
       $nlast\ (nkfilter\ P\ k\ (ntaken\ n\ nellx)) = k + n$ 
    assume enat (Suc n)  $\leq$  eSuc (nlength x22)
    then have enat n < eSuc (nlength x22)
    using Suc-ile-eq by blast
    then have f3: enat n  $\leq$  nlength x22
    by (meson iless-Suc-eq)
    have  $\exists a. a \in nset\ (ntaken\ n\ x22) \wedge P\ a$ 
    using a1 nfinite-ntaken nset-nlast by blast
    then show nlast (nkfilter P k (NCons x21 (ntaken n x22))) = Suc (k + n)
    using f3 a2 a1 by (simp add: Bex-def-raw)
  qed
qed
qed

lemma nkfilter-nfirst:
  assumes P (nfirst(ndropn n nellx))
  shows nfirst(nkfilter P k (ndropn n nellx)) = k
  using assms
proof (induction n arbitrary: k nellx)
case 0
then show ?case
  by (metis enat-defs(1) nellist.set-intros(1) nkfilter-NNil nkfilter-nappend-ntaken nlength-NNil
    nnth-NNil ntaken-0 the-enat.simps zero-le)
next
case (Suc n)
then show ?case
proof (cases nellx)
case (NNil x1)
then show ?thesis using Suc
  by (metis ndropn-ndropn plus-1-eq-Suc)
next
case (NCons x21 x22)
then show ?thesis using Suc
  by auto
qed
qed

```

lemma nkfilter-chop1-ndropn:

assumes $n \leq \text{nlength } \text{nellx}$
 $P (\text{nlast } (\text{ntaken } n \text{ nellx}))$
shows $\text{ndropn } (\text{the-enat } (\text{nlength}(\text{nkfilter } P \ k \ (\text{ntaken } n \text{ nellx})))) (\text{nkfilter } P \ k \ \text{nellx}) =$
 $(\text{nkfilter } P \ (k+n) \ (\text{ndropn } n \ \text{nellx}))$
proof –
have 1: $(\text{nkfilter } P \ k \ \text{nellx}) = \text{nfuse } (\text{nkfilter } P \ k \ (\text{ntaken } n \ \text{nellx})) \ (\text{nkfilter } P \ (k+n) \ (\text{ndropn } n \ \text{nellx}))$
by (*simp add: assms nkfilter-chop1*)
have 2: $\text{nlast}(\text{nkfilter } P \ k \ (\text{ntaken } n \ \text{nellx})) = \text{nfirst } (\text{nkfilter } P \ (k+n) \ (\text{ndropn } n \ \text{nellx}))$
by (*metis assms(1) assms(2) ndropn-nfirst nkfilter-nfirst nkfilter-nlast ntaken-nlast*)
have 3: $\text{ndropn } (\text{the-enat } (\text{nlength}(\text{nkfilter } P \ k \ (\text{ntaken } n \ \text{nellx}))))$
 $(\text{nfuse } (\text{nkfilter } P \ k \ (\text{ntaken } n \ \text{nellx})) \ (\text{nkfilter } P \ (k+n) \ (\text{ndropn } n \ \text{nellx}))) =$
 $(\text{nkfilter } P \ (k+n) \ (\text{ndropn } n \ \text{nellx}))$
by (*metis 2 assms(2) enat-the-enat infinity-ileE ndropn-nfuse nfinite-conv-nlength-enat*
nfinite-ntaken nlength-nkfilter-le nset-nlast)
show ?thesis **by** (*simp add: 1 3*)
qed

lemma *nkfilter-chop1-ntaken*:

assumes $n \leq \text{nlength } \text{nellx}$
 $P (\text{nlast } (\text{ntaken } n \ \text{nellx}))$
shows $\text{ntaken } (\text{the-enat } (\text{nlength}(\text{nkfilter } P \ k \ (\text{ntaken } n \ \text{nellx})))) (\text{nkfilter } P \ k \ \text{nellx}) =$
 $(\text{nkfilter } P \ k \ (\text{ntaken } n \ \text{nellx}))$
proof –
have 1: $(\text{nkfilter } P \ k \ \text{nellx}) = \text{nfuse } (\text{nkfilter } P \ k \ (\text{ntaken } n \ \text{nellx})) \ (\text{nkfilter } P \ (k+n) \ (\text{ndropn } n \ \text{nellx}))$
by (*simp add: assms nkfilter-chop1*)
have 2: $\text{nlast}(\text{nkfilter } P \ k \ (\text{ntaken } n \ \text{nellx})) = \text{nfirst } (\text{nkfilter } P \ (k+n) \ (\text{ndropn } n \ \text{nellx}))$
by (*metis assms(1) assms(2) ndropn-nfirst nkfilter-nfirst nkfilter-nlast ntaken-nlast*)
have 3: $\text{ntaken } (\text{the-enat } (\text{nlength}(\text{nkfilter } P \ k \ (\text{ntaken } n \ \text{nellx}))))$
 $(\text{nfuse } (\text{nkfilter } P \ k \ (\text{ntaken } n \ \text{nellx})) \ (\text{nkfilter } P \ (k+n) \ (\text{ndropn } n \ \text{nellx}))) =$
 $(\text{nkfilter } P \ k \ (\text{ntaken } n \ \text{nellx}))$
by (*metis 1 assms(1) assms(2) nfinite-ntaken nkfilter-nappend-ntaken nset-nlast*)
show ?thesis **by** (*simp add: 1 3*)
qed

lemma *nfilter-nsubn*:

assumes $\text{enat } j \leq \text{nlength } \text{nellx}$
 $P (\text{nnth } \text{nellx } j)$
 $\text{enat } i \leq \text{nlength } \text{nellx}$
 $P (\text{nnth } \text{nellx } i)$
 $i \leq j$
 $\text{enat } ni = (\text{nlength}(\text{nfilter } P \ (\text{ntaken } i \ \text{nellx})))$
 $\text{enat } nj = (\text{nlength}(\text{nfilter } P \ (\text{ntaken } j \ \text{nellx})))$
shows $(\text{nfilter } P \ (\text{nsubn } \text{nellx } i \ j)) = (\text{nsubn } (\text{nfilter } P \ \text{nellx}) \ ni \ nj)$
proof –
have 1: $(\text{nfilter } P \ (\text{nsubn } \text{nellx } i \ j)) = (\text{nfilter } P \ (\text{ntaken } (j-i) \ (\text{ndropn } i \ \text{nellx})))$
by (*simp add: nsubn-def1*)
have 2: $(\text{nfilter } P \ (\text{ntaken } (j-i) \ (\text{ndropn } i \ \text{nellx}))) =$
 $\text{ntaken } (\text{the-enat } (\text{nlength}(\text{nfilter } P \ (\text{ntaken } (j-i) \ (\text{ndropn } i \ \text{nellx}))))$
 $(\text{nfilter } P \ (\text{ndropn } i \ \text{nellx}))$
by (*metis assms(2) assms(5) enat-ile le-add-diff-inverse2 linorder-le-cases nfilter-chop1-ntaken*)


```

    ntaken-all ntaken-ndropn-nlast)
have 3: ntaken (the-enat (nlength(nfilter P (ntaken (j-i) (ndropn i nellx))))
    (nfilter P (ndropn i nellx)) =
    ntaken (the-enat (nlength(nfilter P (ntaken (j-i) (ndropn i nellx))))
    (ndropn (the-enat (nlength(nfilter P (ntaken i nellx)))) (nfilter P nellx))
by (simp add: assms(3) assms(4) nfilter-chop1-ndropn ntaken-nlast)
have 4: ntaken (the-enat (nlength(nfilter P (ntaken (j-i) (ndropn i nellx))))
    (ndropn (the-enat (nlength(nfilter P (ntaken i nellx)))) (nfilter P nellx)) =
    nsubn (nfilter P nellx)
    (the-enat (nlength(nfilter P (ntaken i nellx))))
    ((the-enat (nlength(nfilter P (ntaken (j-i) (ndropn i nellx)))))+
    (the-enat (nlength(nfilter P (ntaken i nellx)))))
using ntaken-ndropn by blast
have 5: ((the-enat (nlength(nfilter P (ntaken (j-i) (ndropn i nellx)))))+
    (the-enat (nlength(nfilter P (ntaken i nellx))))) =
    ((the-enat (nlength(nfilter P (nsubn nellx i j))))+
    (the-enat (nlength(nfilter P (ntaken i nellx)))))
using 1 by auto
have 6: ntaken j nellx = nfuse (ntaken i nellx) (nsubn nellx i j)
using nsubn-nfuse[of 0 i j nellx]
by (simp add: assms(1) assms(5) nsubn-def1)
have 7: nlength (nfilter P (nfuse (ntaken i nellx) (nsubn nellx i j))) =
    (nlength (nfilter P (ntaken i nellx)) + nlength (nfilter P (nsubn nellx i j)))
by (simp add: assms(4) ndropn-nfirst nfilter-chop nfuse-nlength nsubn-def1 ntaken-nfirst ntaken-nlast)
have 70: nfinite (nfilter P (nfuse (ntaken i nellx) (nsubn nellx i j)))
by (metis 6 assms(1) assms(2) nfilter-chop1-ntaken nfinite-ntaken ntaken-nlast)
have 71: nfinite (nfilter P (ntaken i nellx))
by (metis assms(3) assms(4) nfilter-chop1-ntaken nfinite-ntaken ntaken-nlast)
have 72: nfinite (nsubn nellx i j)
by (simp add: nsubn-def1)
have 8: nj =
    ((the-enat (nlength(nfilter P (nsubn nellx i j))))+ ni)
by (metis 6 7 add.commute assms(6) assms(7) enat-le-plus-same(2) enat-the-enat infinity-ileE
    plus-enat-simps(1) the-enat.simps)
show ?thesis
using 1 2 3 4 8 by (metis assms(6) the-enat.simps)
qed

```

lemma nfilter-nsubn-zero:

assumes enat j ≤ nlength nellx

P (nnth nellx j)

shows (nfilter P (nsubn nellx 0 j)) =

(nsubn (nfilter P nellx)

0

(the-enat (nlength(nfilter P (ntaken j nellx)))))

)

using assms

by (simp add: nfilter-chop1-ntaken nsubn-def1 ntaken-nlast)

1.3.10 nrev

lemma *is-NNil-nrev[simp]*:

is-NNil (nrev xs) = is-NNil xs

apply *transfer*

by (*simp add: is-LEmpty-llength*)

lemma *nrev-NNil[simp]*:

nrev (NNil a) = (NNil a)

apply *transfer*

by *force*

lemma *nrev-NCons[simp]*:

nrev (NCons a xs) = (if nfinite xs then nappend (nrev xs) (NNil a) else (NCons a xs))

apply *transfer*

unfolding *lrev-def*

by *auto*

(*metis lfinite-code(2) list-of-LCons lrev-LCons lrev-def rev.simps(2)*)

lemma *ntl-nrev[simp]*:

nlength xs > 1 \implies

ntl (nrev xs) =

(if nfinite xs then

(nappend (ntl(nrev (ntl xs))) (NNil (nhd xs)))

else ntl xs)

apply *transfer*

by (*auto simp add: is-LEmpty-llength epred-llength*)

(*simp add: llength-ltl*)

lemma *ntl-nrev-one[simp]*:

nlength xs = 1 \implies

ntl (nrev xs) = (NNil (nhd xs))

apply *transfer*

apply (*auto simp add: is-LEmpty-llength*)

apply (*metis epred-1 epred-llength lappend-lnull1 llength-eq-0 llength-lrev*)

apply (*simp add: epred-llength*)

by (*metis co.enat.sel(2) eSuc-infinity llength-eq-infty-conv-lfinite*)

lemma *ntl-nrev-zero[simp]*:

assumes *nlength xs = 0*

shows *ntl (nrev xs) = xs*

by (*metis assms ndropn-0 ndropn-nlast nellist.sel(4) nlength-eq-enat-nfiniteD nrev-NNil the-enat-0 zero-enat-def*)

lemma *nmap-nrev*:

nmap f (nrev xs) = (nrev (nmap f xs))

apply *transfer*

by (*simp add: lmap-lrev*)

lemma *nfinite-nrev*:

$nfinite(nrev\ xs) = nfinite\ xs$

apply *transfer*

by *simp*

lemma *nset-nrev*:

$nset(nrev\ xs) = nset\ xs$

apply *transfer*

by (*simp add: lset-lrev*)

lemma *nlength-nrev*[*simp*]:

$nlength\ (nrev\ xs) = nlength\ xs$

apply *transfer*

by *simp*

lemma *lnth-lrev-help*:

$\neg\ lnull\ xs \implies\ lfinite\ xs \implies\ enat\ i \leq\ epred\ (llength\ xs) \implies\ lnth\ (lrev\ xs)\ i = lnth\ xs\ (the-enat\ (min\ (enat\ (the-enat\ (epred\ (llength\ xs)) - i))\ (epred\ (llength\ xs))))$

using *lnth-lrev[of xs i]*

unfolding *min-def*

apply *auto*

apply (*metis co.enat.exhaust-sel diff-diff-left enat-the-enat epred-enat iless-Suc-eq*

llength-eq-0 llength-eq-inf-conv-lfinite plus-1-eq-Suc the-enat.simps)

by (*metis co.enat.exhaust-sel diff-le-self eSuc-infinity enat-ord-simps(1) enat-the-enat llength-eq-0 llength-eq-inf-conv-lfinite*)

lemma *nnth-nrev*:

assumes *nfinite xs*

$(enat\ i) \leq nlength\ xs$

shows $(nnth\ (nrev\ xs)\ i) = (nnth\ xs\ ((the-enat\ (nlength\ xs)) - i))$

using *assms*

apply *transfer*

apply *auto*

using *lnth-lrev-help* **by** *blast*

lemma *nappend-nrev-nfinite*:

assumes *nfinite xs*

nfinite ys

shows $nrev\ (nappend\ xs\ ys) = nappend\ (nrev\ ys)\ (nrev\ xs)$

using *assms*

apply *transfer*

by (*simp add: lappend-lrev-lfinite*)

lemma *nappend-nrev-infa*:

assumes *nfinite xs*

$\neg nfinite\ ys$

shows $nrev\ (nappend\ xs\ ys) = nappend\ xs\ ys$

using *assms*

apply *transfer*

by *simp*

lemma *nappend-nrev--infb*:

assumes $\neg \text{nfinite } xs$

shows $\text{nrev } (\text{nappend } xs \ y) = \text{nappend } (\text{nrev } xs) (\text{nrev } y)$

using *assms*

apply *transfer*

by (*simp add: lappend-inf*)

lemma *nrev-nrev*:

$\text{nrev } (\text{nrev } xs) = xs$

apply *transfer*

by (*simp add: lrev-lrev*)

lemma *nrev-ntaken*:

assumes $\text{nfinite } xs$

$\text{enat } k \leq \text{nlength } xs$

shows $\text{nrev } (\text{ntaken } k \ xs) = \text{ndropn } (\text{the-enat}(\text{nlength } xs) - k) (\text{nrev } xs)$

using *assms*

proof (*transfer*)

fix $xs \ k$

show $\neg \text{lnull } xs \wedge xs = xs \implies$

$\text{lfinite } xs \implies$

$\text{enat } k \leq \text{epred } (\text{llength } xs) \implies$

$\neg \text{lnull}$

$(\text{if } \text{lfinite } (\text{ltake } (\text{enat } (\text{Suc } k)) \ xs)$

$\text{then } \text{lrev } (\text{ltake } (\text{enat } (\text{Suc } k)) \ xs)$

$\text{else } \text{ltake } (\text{enat } (\text{Suc } k)) \ xs) \wedge$

$(\text{if } \text{lfinite } (\text{ltake } (\text{enat } (\text{Suc } k)) \ xs)$

$\text{then } \text{lrev } (\text{ltake } (\text{enat } (\text{Suc } k)) \ xs)$

$\text{else } \text{ltake } (\text{enat } (\text{Suc } k)) \ xs) =$

ldropn

$(\text{the-enat}$

$(\text{min } (\text{enat}$

$(\text{the-enat } (\text{epred } (\text{llength } xs)) - k))$

$(\text{epred}$

$(\text{llength}$

$(\text{if } \text{lfinite } xs \text{ then } \text{lrev } xs$

$\text{else } xs))))))$

$(\text{if } \text{lfinite } xs \text{ then } \text{lrev } xs \text{ else } xs)$

proof –

assume $a0: \neg \text{lnull } xs \wedge xs = xs$

assume $a1: \text{lfinite } xs$

assume $a2: \text{enat } k \leq \text{epred } (\text{llength } xs)$

show

$\neg \text{lnull}$

$(\text{if } \text{lfinite } (\text{ltake } (\text{enat } (\text{Suc } k)) \ xs) \text{ then } \text{lrev } (\text{ltake } (\text{enat } (\text{Suc } k)) \ xs)$

$\text{else } \text{ltake } (\text{enat } (\text{Suc } k)) \ xs) \wedge$

$(\text{if } \text{lfinite } (\text{ltake } (\text{enat } (\text{Suc } k)) \ xs) \text{ then } \text{lrev } (\text{ltake } (\text{enat } (\text{Suc } k)) \ xs) \text{ else } \text{ltake } (\text{enat } (\text{Suc } k)) \ xs) =$

ldropn

```

  (the-enat
    (min (enat (the-enat (epred (llength xs)) - k)) (epred (llength (if lfinite xs then lrev xs else xs)))))
    (if lfinite xs then lrev xs else xs))
proof -
have 1: enat (Suc k) ≠ 0
  by (metis eSuc-enat zero-ne-eSuc)
have 2: llength xs = 1 ⇒ ?thesis
  using a0 a1 a2 by (simp add: 1 ltake-all one-enat-def)
have 3: enat (Suc k) < llength xs ⇒
  (the-enat (min (enat (the-enat (epred (llength xs)) - k)) (epred (llength xs)))) =
  (enat (the-enat (llength xs) - Suc k))
  using a0 a1 a2 by simp
  (metis diff-diff-left diff-le-self enat-ord-simps(1) enat-the-enat epred-enat
    llength-eq-infity-conv-lfinite min.absorb1 plus-1-eq-Suc the-enat.simps)

have 4: enat (Suc k) < llength xs ⇒ ?thesis
  using a0 a1 a2 lrev-ltake[of xs Suc k] apply simp using 3
  by (metis 1 ldrops-enat llength-lrev lnull-ldrops lnull-lrev ltake.disc(2))
have 5: llength xs = enat (Suc k) ⇒ ?thesis
  using a0 a1 a2 by (simp add: 1 ltake-all)
have 6: llength xs = 1 ∨ enat (Suc k) < llength xs ∨ llength xs = enat (Suc k)
  using a0 a1 a2 apply simp
  by (metis Suc-ile-eq co.enat.exhaust-sel iless-Suc-eq llength-eq-0 order-neq-le-trans)
show ?thesis using a0 1 2 4 5 6 by blast
qed
qed
qed

```

lemma nrev-ndropn:

assumes nfinite xs

enat k ≤ nlength xs

shows nrev (ndropn k xs) = ntaken (the-enat(nlength xs) - k) (nrev xs)

using assms

proof transfer

fix xs k

show ¬ lnull xs ∧ xs = xs ⇒

lfinite xs ⇒

enat k ≤ epred (llength xs) ⇒

¬ lnull

(if lfinite (ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
 then lrev (ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
 else ldrops (the-enat (min (enat k) (epred (llength xs)))) xs) ∧
 (if lfinite (ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
 then lrev (ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
 else ldrops (the-enat (min (enat k) (epred (llength xs)))) xs) =
 ltake (enat (Suc (the-enat (epred (llength xs)) - k))) (if lfinite xs then lrev xs else xs))

proof -

assume a0: ¬ lnull xs ∧ xs = xs

assume a1: lfinite xs

```

assume a2: enat k ≤ epred (llength xs)
show ¬ lnull
  (if lfinite (ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
   then lrev (ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
   else ldropn (the-enat (min (enat k) (epred (llength xs)))) xs) ∧
  (if lfinite (ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
   then lrev (ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
   else ldropn (the-enat (min (enat k) (epred (llength xs)))) xs) =
  ltake (enat (Suc (the-enat (epred (llength xs)) - k))) (if lfinite xs then lrev xs else xs)
proof -
have 1: ¬ lnull
  (if lfinite (ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
   then lrev (ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
   else ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
  using a0 a1 a2 apply simp
  by (metis co.enat.exhaust-sel iless-Suc-eq leD llength-eq-0)
have 2: lfinite (ldropn (the-enat (min (enat k) (epred (llength xs)))) xs)
  using a1 lfinite-ldropn by blast
have 3: (the-enat (min (enat k) (epred (llength xs)))) = enat k
  using a0 a1 a2 by simp
have 4: (enat (the-enat (llength xs) - k)) = (enat (Suc (the-enat (epred (llength xs)) - k)))
  using a0 a1 a2 apply simp
  using 1 llength-eq-inf-conv-lfinite by fastforce
have 5: llength xs = 1 ∨ enat (Suc k) < llength xs ∨ llength xs = enat (Suc k)
  using a0 a1 a2 apply simp
  by (metis Suc-ile-eq co.enat.exhaust-sel iless-Suc-eq llength-eq-0 order-neq-le-trans)
have 6: llength xs = 1 ⇒ ?thesis
  using a0 a1 a2 by (simp add: ltake-all one-enat-def)
have 7: enat (Suc k) < llength xs ⇒ ?thesis
  using lrev-ldrop[of xs k] a0 a1 a2 apply simp
  by (metis 4 Suc-ile-eq dual-order.strict-iff-order ldrop-enat linorder-not-less)
have 8: llength xs = enat (Suc k) ⇒ ?thesis
  using a0 a1 a2 apply simp
  by (metis 4 a2 add-diff-cancel-left' cancel-comm-monoid-add-class.diff-cancel eSuc-enat epred-enat
    iless-Suc-eq ldrop-enat lrev-ldrop plus-1-eq-Suc the-enat.simps)
show ?thesis
using 5 6 7 8 by fastforce
qed
qed
qed

```

lemma nrev-nsbn:

assumes nfinite xs

enat i ≤ enat j

enat j ≤ nlength xs

shows nrev (nsbn xs i j) = nsbn (nrev xs)((the-enat(nlength xs)) - j) ((the-enat(nlength xs)) - i)

using assms

proof transfer

```

fix  $xs\ i\ j$ 
assume  $a0: \neg lnull\ xs \wedge xs = xs$ 
assume  $a1: lfinite\ xs$ 
assume  $a2: enat\ i \leq enat\ j$ 
assume  $a3: enat\ j \leq epred\ (llength\ xs)$ 
show  $\neg lnull$ 
   $(if\ lfinite\ (lsubc\ i\ j\ xs)$ 
     $then\ lrev\ (lsubc\ i\ j\ xs)$ 
     $else\ lsubc\ i\ j\ xs) \wedge$ 
   $(if\ lfinite\ (lsubc\ i\ j\ xs)$ 
     $then\ lrev\ (lsubc\ i\ j\ xs)\ else\ lsubc\ i\ j\ xs) =$ 
   $lsubc\ (the-enat\ (epred\ (llength\ xs)) - j)$ 
   $(the-enat\ (epred\ (llength\ xs)) - i)$ 
   $(if\ lfinite\ xs\ then\ lrev\ xs\ else\ xs)$ 
proof -
have 1:  $\neg lnull\ (if\ lfinite\ (lsubc\ i\ j\ xs)\ then\ lrev\ (lsubc\ i\ j\ xs)\ else\ lsubc\ i\ j\ xs)$ 
  using  $a0\ a1\ a2\ a3$  apply simp
  by (metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0 llength-lsubc zero-ne-eSuc)
have 2:  $lfinite\ (lsubc\ i\ j\ xs)$ 
  using  $a0\ a1\ a2\ a3$  by (simp add: lsubc-def)
have 3:  $(the-enat\ (epred\ (llength\ xs)) - j) = (the-enat\ (llength\ xs) - (j + 1))$ 
  using  $a0\ a1\ a2\ a3\ lfinite-llength-enat$  by fastforce
have 4:  $(the-enat\ (epred\ (llength\ xs)) - i) = (the-enat\ (llength\ xs) - (i + 1))$ 
  using  $a0\ a1\ a2\ a3\ lfinite-llength-enat$  by fastforce
have 5:  $enat\ j < llength\ xs$ 
  using  $a0\ a1\ a2\ a3$  apply simp
  by (metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0)
have 6:  $lrev\ (lsubc\ i\ j\ xs) = lsubc\ (the-enat\ (epred\ (llength\ xs)) - j)$ 
   $(the-enat\ (epred\ (llength\ xs)) - i)$ 
   $(if\ lfinite\ xs\ then\ lrev\ xs\ else\ xs)$ 
  using  $lrev-lsubc[of\ xs\ i\ j]$  5  $a0\ a1\ a2\ a3\ 3\ 4$  by presburger
show ?thesis by (meson 1 2 6)
qed
qed

```

lemma *nlast-nrev*:

shows $nlast\ (nrev\ xs) = (if\ nfinite\ xs\ then\ nfirst\ xs\ else\ nlast\ xs)$
apply *transfer*
by (*simp add: lfirst-def llast-lrev*)

lemma *nfirst-nrev*:

shows $nfirst\ (nrev\ xs) = (if\ nfinite\ xs\ then\ nlast\ xs\ else\ nfirst\ xs)$
apply *transfer*
apply *simp*
by (*metis lfirst-def lfirst-lrev*)

```

lemma nellist-all2-nrev:
  nellist-all2 P (nrev xs) (nrev ys)  $\longleftrightarrow$  nellist-all2 P xs ys
apply transfer
using l1ist-all2-lfiniteD l1ist-all2-lrev apply simp
by fastforce

```

end

1.3.11 Transfer rule

```

context includes lifting-syntax
begin
lemma ndropns-transfer2 [transfer-rule]:
  (nellist-all2 A ==> nellist-all2 (nellist-all2 A)) ndropns ndropns
unfolding rel-fun-def
by (auto intro: nellist-all2-ndropnsI )

```

end

end

1.4 Extended Integers

theory *Extended-Int*

```

imports
  HOL-Library.Extended-Nat

```

begin

```

datatype eint = eint int | PInfty | MInfty

```

```

lemma eint-cong:
   $x = y \implies \text{eint } x = \text{eint } y$ 
by simp

```

```

instantiation eint :: uminus
begin

```

```

fun uminus-eint where
   $\neg(\text{eint } i) = \text{eint } (\neg i)$ 
|  $\neg PInfty = MInfty$ 
|  $\neg MInfty = PInfty$ 

```



```

instance ..

end

instantiation eint :: infinity
begin

definition ( $\infty$  :: eint) = PInfty
instance ..

end

declare [[coercion eint :: int  $\Rightarrow$  eint ]]

lemma eint-uminus-uminus[simp]:
fixes a :: eint
shows  $- (- a) = a$ 
by (cases a) simp-all

lemma
shows PInfty-eq-infinity[simp]:  $PInfty = \infty$ 
and MInfty-eq-minfinity[simp]:  $MInfty = - \infty$ 
and MInfty-neq-PInfty[simp]:  $\infty \neq - (\infty :: eint) - \infty \neq (\infty :: eint)$ 
and MInfty-neq-eint[simp]:  $eint\ r \neq - \infty - \infty \neq eint\ r$ 
and PInfty-neq-eint[simp]:  $eint\ r \neq \infty \infty \neq eint\ r$ 
and PInfty-cases[simp]:  $(case\ \infty\ of\ eint\ r \Rightarrow f\ r \mid PInfty \Rightarrow y \mid MInfty \Rightarrow z) = y$ 
and MInfty-cases[simp]:  $(case\ -\ \infty\ of\ eint\ r \Rightarrow f\ r \mid PInfty \Rightarrow y \mid MInfty \Rightarrow z) = z$ 
by (simp-all add: infinity-eint-def)

declare
PInfty-eq-infinity[code-post]
MInfty-eq-minfinity[code-post]

lemma [code-unfold]:
 $\infty = PInfty$ 
 $- PInfty = MInfty$ 
by simp-all

lemma inj-eint[simp]: inj-on eint A
unfolding inj-on-def by auto

lemma eint-cases[cases type: eint]:
obtains (int) i where  $x = eint\ i$ 
| (PInf)  $x = \infty$ 
| (MInf)  $x = -\infty$ 
by (cases x) auto

lemmas eint2-cases = eint-cases[case-product eint-cases]

```

lemmas *eint3-cases* = *eint2-cases*[*case-product eint-cases*]

lemma *eint-all-split*: $\bigwedge P. (\forall x::eint. P\ x) \longleftrightarrow P\ \infty \wedge (\forall x. P\ (eint\ x)) \wedge P\ (-\infty)$
by (*metis eint-cases*)

lemma *eint-ex-split*: $\bigwedge P. (\exists x::eint. P\ x) \longleftrightarrow P\ \infty \vee (\exists x. P\ (eint\ x)) \vee P\ (-\infty)$
by (*metis eint-cases*)

lemma *eint-uminus-eq-iff*[*simp*]:
fixes *a b* :: *eint*
shows $-a = -b \longleftrightarrow a = b$
by (*cases rule: eint2-cases[of a b]*) *simp-all*

function *int-of-eint* :: *eint* \Rightarrow *int* **where**
int-of-eint (*eint* *r*) = *r*
| *int-of-eint* ∞ = 0
| *int-of-eint* $(-\infty)$ = 0
by (*auto intro: eint-cases*)
termination **by** *standard* (*rule wf-on-bot*)

lemma *int-of-eint*[*simp*]:
int-of-eint ($- x :: eint$) = $-(int-of-eint\ x)$
by (*cases x*) *simp-all*

lemma *range-eint*[*simp*]: *range eint* = *UNIV* $- \{\infty, -\infty\}$
proof *safe*
fix *x*
assume $x \notin range\ eint\ x \neq \infty$
then show $x = -\infty$
by (*cases x*) *auto*
qed *auto*

lemma *eint-range-uminus*[*simp*]: *range uminus* = (*UNIV*::*eint* *set*)
proof *safe*
fix *x* :: *eint*
show $x \in range\ uminus$
by (*intro image-eqI[of - - -x]*) *auto*
qed *auto*

instantiation *eint* :: *abs*
begin

function *abs-eint* **where**
| *eint* *i* | = *eint* |*i*|
| $-\infty$ | = ($\infty::eint$)
| ∞ | = ($\infty::eint$)
by (*auto intro: eint-cases*)
termination **proof** **qed**
(*rule wf-on-bot*)

instance ..

end

lemma *abs-eq-infinity-cases*[*elim!*]:

fixes $x :: \text{eint}$
assumes $|x| = \infty$
obtains $x = \infty \mid x = -\infty$
using *assms* **by** (*cases* x) *auto*

lemma *abs-neq-infinity-cases*[*elim!*]:

fixes $x :: \text{eint}$
assumes $|x| \neq \infty$
obtains r **where** $x = \text{eint } r$
using *assms* **by** (*cases* x) *auto*

lemma *abs-eint-uminus*[*simp*]:

fixes $x :: \text{eint}$
shows $|- x| = |x|$
by (*cases* x) *auto*

lemma *eint-infinity-cases*:

fixes $a :: \text{eint}$
shows $a \neq \infty \implies a \neq -\infty \implies |a| \neq \infty$
by *auto*

Addition

instantiation $\text{eint} :: \{\text{one}, \text{comm-monoid-add}, \text{zero-neq-one}\}$
begin

definition $0 = \text{eint } 0$

definition $1 = \text{eint } 1$

function *plus-eint* **where**

$\text{eint } r + \text{eint } p = \text{eint } (r + p)$
 $|\infty + a = (\infty :: \text{eint})$
 $|a + \infty = (\infty :: \text{eint})$
 $|\text{eint } r + -\infty = -\infty$
 $|- \infty + \text{eint } p = -(\infty :: \text{eint})$
 $|- \infty + -\infty = -(\infty :: \text{eint})$

proof *goal-cases*

case *prems*: ($1 \ P \ x$)
then obtain $a \ b$ **where** $x = (a, b)$
by (*cases* x) *auto*
with *prems* **show** P

```

  by (cases rule: eint2-cases[of a b]) auto
qed auto
termination by standard (rule wf-on-bot )

```

```

lemma Infty-neq-0[simp]:
  ( $\infty::\text{eint}$ )  $\neq$  0 0  $\neq$  ( $\infty::\text{eint}$ )
   $\neg$ ( $\infty::\text{eint}$ )  $\neq$  0 0  $\neq$   $\neg$ ( $\infty::\text{eint}$ )
  by (simp-all add: zero-eint-def)

```

```

lemma eint-eq-0[simp]:
   $\text{eint } r = 0 \longleftrightarrow r = 0$ 
   $0 = \text{eint } r \longleftrightarrow r = 0$ 
  unfolding zero-eint-def by simp-all

```

```

lemma eint-eq-1[simp]:
   $\text{eint } r = 1 \longleftrightarrow r = 1$ 
   $1 = \text{eint } r \longleftrightarrow r = 1$ 
  unfolding one-eint-def by simp-all

```

```

instance
proof
  fix a b c :: eint
  show 0 + a = a
    by (cases a) (simp-all add: zero-eint-def)
  show a + b = b + a
    by (cases rule: eint2-cases[of a b]) simp-all
  show a + b + c = a + (b + c)
    by (cases rule: eint3-cases[of a b c]) simp-all
  show 0  $\neq$  (1::eint)
    by (simp add: one-eint-def zero-eint-def)
qed
end

```

```

lemma eint-0-plus [simp]:  $\text{eint } 0 + x = x$ 
  and plus-eint-0 [simp]:  $x + \text{eint } 0 = x$ 
by (simp-all flip: zero-eint-def)

```

```

instance eint :: numeral ..

```

```

lemma int-of-eint-0[simp]:  $\text{int-of-eint } (0::\text{eint}) = 0$ 
  unfolding zero-eint-def by simp

```

```

lemma abs-eint-zero[simp]:  $|0| = (0::\text{eint})$ 
  unfolding zero-eint-def abs-eint.simps by simp

```

```

lemma eint-uminus-zero[simp]:  $- 0 = (0::\text{eint})$ 

```

by (simp add: zero-eint-def)

lemma *eint-uminus-zero-iff*[simp]:

fixes $a :: \text{eint}$

shows $-a = 0 \longleftrightarrow a = 0$

by (cases a) simp-all

lemma *eint-plus-eq-PInfty*[simp]:

fixes $a\ b :: \text{eint}$

shows $a + b = \infty \longleftrightarrow a = \infty \vee b = \infty$

by (cases rule: eint2-cases[of a b]) auto

lemma *eint-plus-eq-MInfty*[simp]:

fixes $a\ b :: \text{eint}$

shows $a + b = -\infty \longleftrightarrow (a = -\infty \vee b = -\infty) \wedge a \neq \infty \wedge b \neq \infty$

by (cases rule: eint2-cases[of a b]) auto

lemma *eint-add-cancel-left*:

fixes $a\ b :: \text{eint}$

assumes $a \neq -\infty$

shows $a + b = a + c \longleftrightarrow a = \infty \vee b = c$

using assms by (cases rule: eint3-cases[of a b c]) auto

lemma *eint-add-cancel-right*:

fixes $a\ b :: \text{eint}$

assumes $a \neq -\infty$

shows $b + a = c + a \longleftrightarrow a = \infty \vee b = c$

using assms by (cases rule: eint3-cases[of a b c]) auto

lemma *eint-int*: $\text{eint} (\text{int-of-eint } x) = (\text{if } |x| = \infty \text{ then } 0 \text{ else } x)$

by (cases x) simp-all

lemma *int-of-eint-add*:

fixes $a\ b :: \text{eint}$

shows $\text{int-of-eint } (a + b) =$

(if $(|a| = \infty) \wedge (|b| = \infty) \vee (|a| \neq \infty) \wedge (|b| \neq \infty)$ then $\text{int-of-eint } a + \text{int-of-eint } b$ else 0)

by (cases rule: eint2-cases[of a b]) auto

Linear order on *eint*

instantiation *eint* :: *linorder*

begin

function *less-eint*

where

$\text{eint } x < \text{eint } y$	$\longleftrightarrow x < y$
$(\infty :: \text{eint}) < a$	$\longleftrightarrow \text{False}$
$a < -(\infty :: \text{eint})$	$\longleftrightarrow \text{False}$
$\text{eint } x < \infty$	$\longleftrightarrow \text{True}$
$-\infty < \text{eint } r$	$\longleftrightarrow \text{True}$

```

|       $-\infty < (\infty::\text{eint}) \longleftrightarrow \text{True}$ 
proof goal-cases
  case prems: (1  $P$   $x$ )
  then obtain  $a$   $b$  where  $x = (a,b)$  by (cases  $x$ ) auto
  with prems show  $P$  by (cases rule: eint2-cases[of a b]) auto
qed simp-all
termination by (relation  $\{\}$ ) simp

```

definition $x \leq (y::\text{eint}) \longleftrightarrow x < y \vee x = y$

lemma *eint-infity-less[simp]*:
fixes $x :: \text{eint}$
shows $x < \infty \longleftrightarrow (x \neq \infty)$
 $-\infty < x \longleftrightarrow (x \neq -\infty)$
by (*cases* x , *simp-all*) (*cases* x , *simp-all*)

lemma *eint-infity-less-eq[simp]*:
fixes $x :: \text{eint}$
shows $\infty \leq x \longleftrightarrow x = \infty$
and $x \leq -\infty \longleftrightarrow x = -\infty$
by (*auto simp add: less-eq-eint-def*)

lemma *eint-less[simp]*:
 $\text{eint } r < 0 \longleftrightarrow (r < 0)$
 $0 < \text{eint } r \longleftrightarrow (0 < r)$
 $\text{eint } r < 1 \longleftrightarrow (r < 1)$
 $1 < \text{eint } r \longleftrightarrow (1 < r)$
 $0 < (\infty::\text{eint})$
 $-(\infty::\text{eint}) < 0$
by (*simp-all add: zero-eint-def one-eint-def*)

lemma *eint-less-eq[simp]*:
 $x \leq (\infty::\text{eint})$
 $-(\infty::\text{eint}) \leq x$
 $\text{eint } r \leq \text{eint } p \longleftrightarrow r \leq p$
 $\text{eint } r \leq 0 \longleftrightarrow r \leq 0$
 $0 \leq \text{eint } r \longleftrightarrow 0 \leq r$
 $\text{eint } r \leq 1 \longleftrightarrow r \leq 1$
 $1 \leq \text{eint } r \longleftrightarrow 1 \leq r$
by (*auto simp add: less-eq-eint-def zero-eint-def one-eint-def*)

lemma *eint-infity-less-eq2*:
 $a \leq b \implies a = \infty \implies b = (\infty::\text{eint})$
 $a \leq b \implies b = -\infty \implies a = -(\infty::\text{eint})$
by *simp-all*

instance
proof
fix x y $z :: \text{eint}$
show $x \leq x$

```

    by (cases x) simp-all
show  $x < y \longleftrightarrow x \leq y \wedge \neg y \leq x$ 
  by (cases rule: eint2-cases[of x y]) auto
show  $x \leq y \vee y \leq x$ 
  by (cases rule: eint2-cases[of x y]) auto
{
  assume  $x \leq y \wedge y \leq x$ 
  then show  $x = y$ 
    by (cases rule: eint2-cases[of x y]) auto
}
{
  assume  $x \leq y \wedge y \leq z$ 
  then show  $x \leq z$ 
    by (cases rule: eint3-cases[of x y z]) auto
}
qed

```

end

```

instance eint :: ordered-comm-monoid-add
proof
  fix a b c :: eint
  assume  $a \leq b$ 
  then show  $c + a \leq c + b$ 
    by (cases rule: eint3-cases[of a b c]) auto
qed

```

```

lemma eint-one-not-less-zero-eint[simp]:  $\neg 1 < (0::eint)$ 
  by (simp add: zero-eint-def)

```

```

lemma int-of-eint-positive-mono:
  fixes x y :: eint
  shows  $0 \leq x \implies x \leq y \implies y \neq \infty \implies \text{int-of-eint } x \leq \text{int-of-eint } y$ 
  by (cases rule: eint2-cases[of x y]) auto

```

```

lemma eint-MInfty-lessI[intro, simp]:
  fixes a :: eint
  shows  $a \neq -\infty \implies -\infty < a$ 
  by (cases a) auto

```

```

lemma eint-less-PInfty[intro, simp]:
  fixes a :: eint
  shows  $a \neq \infty \implies a < \infty$ 
  by (cases a) auto

```

```

lemma eint-less-eint-Ex:
  fixes a b :: eint
  shows  $x < \text{eint } r \longleftrightarrow x = -\infty \vee (\exists p. p < r \wedge x = \text{eint } p)$ 
  by (cases x) auto

```

lemma *less-PInf-Ex-of-nat*: $x \neq \infty \longleftrightarrow (\exists n::nat. x < eint (int n))$
proof (*cases* x)
 case ($int\ r$)
 then show *?thesis*
 by (*metis* *gt-ex int-ops*(1) *less-eint.simps*(1) *less-eint.simps*(2) *linorder-less-linear of-nat-less-iff* *zero-less-imp-eq-int*)
qed *simp-all*

lemma *eint-add-strict-mono2*:
 fixes $a\ b\ c\ d :: eint$
 assumes $a < b$ **and** $c < d$
 shows $a + c < b + d$
 using *assms*
 by (*cases* a ; *force simp add: elim: less-eint.elims*)

lemma *eint-minus-le-minus[simp]*:
 fixes $a\ b :: eint$
 shows $- a \leq - b \longleftrightarrow b \leq a$
 by (*cases rule: eint2-cases[of a b]*) *auto*

lemma *eint-minus-less-minus[simp]*:
 fixes $a\ b :: eint$
 shows $- a < - b \longleftrightarrow b < a$
 by (*cases rule: eint2-cases[of a b]*) *auto*

lemma *eint-le-int-iff*:
 $x \leq int\text{-of-}eint\ y \longleftrightarrow (|y| \neq \infty \longrightarrow eint\ x \leq y) \wedge (|y| = \infty \longrightarrow x \leq 0)$
 by (*cases* y) *auto*

lemma *int-le-eint-iff*:
 $int\text{-of-}eint\ y \leq x \longleftrightarrow (|y| \neq \infty \longrightarrow y \leq eint\ x) \wedge (|y| = \infty \longrightarrow 0 \leq x)$
 by (*cases* y) *auto*

lemma *eint-less-int-iff*:
 $x < int\text{-of-}eint\ y \longleftrightarrow (|y| \neq \infty \longrightarrow eint\ x < y) \wedge (|y| = \infty \longrightarrow x < 0)$
 by (*cases* y) *auto*

lemma *int-less-eint-iff*:
 $int\text{-of-}eint\ y < x \longleftrightarrow (|y| \neq \infty \longrightarrow y < eint\ x) \wedge (|y| = \infty \longrightarrow 0 < x)$
 by (*cases* y) *auto*

To help with inferences like $\llbracket a < eint\ x; x < y \rrbracket \Longrightarrow a < eint\ y$, where x and y are *int*.

lemma *le-eint-le*: $a \leq eint\ x \Longrightarrow x \leq y \Longrightarrow a \leq eint\ y$
 using *eint-less-eq*(3) *order.trans* **by** *blast*

lemma *le-eint-less*: $a \leq eint\ x \Longrightarrow x < y \Longrightarrow a < eint\ y$
 by (*simp add: le-less-trans*)

lemma *less-eint-le*: $a < eint\ x \Longrightarrow x \leq y \Longrightarrow a < eint\ y$
 using *eint-less-eint-Ex* **by** *auto*

lemma *eint-le-le*: $eint\ y \leq a \implies x \leq y \implies eint\ x \leq a$
by (*simp add: order-subst2*)

lemma *eint-le-less*: $eint\ y \leq a \implies x < y \implies eint\ x < a$
by (*simp add: dual-order.strict-trans1*)

lemma *eint-less-le*: $eint\ y < a \implies x \leq y \implies eint\ x < a$
using *eint-less-eq(3) le-less-trans* **by** *blast*

lemma *int-of-eint-pos*:
fixes $x :: eint$
shows $0 \leq x \implies 0 \leq int\text{-of-}eint\ x$ **by** (*cases x*) *auto*

lemmas *int-of-eint-ord-simps* =
eint-le-int-iff int-le-eint-iff eint-less-int-iff int-less-eint-iff

lemma *abs-eint-ge0[simp]*: $0 \leq x \implies |x :: eint| = x$
by (*cases x*) *auto*

lemma *abs-eint-less0[simp]*: $x < 0 \implies |x :: eint| = -x$
by (*cases x*) *auto*

lemma *abs-eint-pos[simp]*: $0 \leq |x :: eint|$
by (*cases x*) *auto*

lemma *eint-abs-leI*:
fixes $x\ y :: eint$
shows $\llbracket x \leq y; -x \leq y \rrbracket \implies |x| \leq y$
by(*cases x y rule: eint2-cases*)(*simp-all*)

lemma *eint-abs-add*:
fixes $a\ b :: eint$
shows $abs(a+b) \leq abs\ a + abs\ b$
by (*cases rule: eint2-cases[of a b]*) (*auto*)

lemma *int-of-eint-le-0[simp]*: $int\text{-of-}eint\ (x :: eint) \leq 0 \longleftrightarrow x \leq 0 \vee x = \infty$
by (*cases x*) *auto*

lemma *abs-int-of-eint[simp]*: $|int\text{-of-}eint\ (x :: eint)| = int\text{-of-}eint\ |x|$
by (*cases x*) *auto*

lemma *zero-less-int-of-eint*:
fixes $x :: eint$
shows $0 < int\text{-of-}eint\ x \longleftrightarrow 0 < x \wedge x \neq \infty$
by (*cases x*) *auto*

lemma *eint-0-le-uminus-iff[simp]*:
fixes $a :: eint$
shows $0 \leq -a \longleftrightarrow a \leq 0$

by (cases rule: eint2-cases[of a]) auto

lemma eint-uminus-le-0-iff[simp]:

fixes a :: eint
 shows $-a \leq 0 \longleftrightarrow 0 \leq a$
 by (cases rule: eint2-cases[of a]) auto

lemma eint-add-strict-mono:

fixes a b c d :: eint
 assumes $a \leq b$
 and $0 \leq a$
 and $a \neq \infty$
 and $c < d$
 shows $a + c < b + d$
 using assms
 by (cases rule: eint3-cases[case-product eint-cases, of a b c d]) auto

lemma eint-less-add:

fixes a b c :: eint
 shows $|a| \neq \infty \implies c < b \implies a + c < a + b$
 by (cases rule: eint2-cases[of b c]) auto

lemma eint-add-nonneg-eq-0-iff:

fixes a b :: eint
 shows $0 \leq a \implies 0 \leq b \implies a + b = 0 \longleftrightarrow a = 0 \wedge b = 0$
 by (cases a b rule: eint2-cases) auto

lemma eint-uminus-eq-reorder: $-a = b \longleftrightarrow a = (-b::eint)$

by auto

lemma eint-uminus-less-reorder: $-a < b \longleftrightarrow -b < (a::eint)$

by (subst (3) eint-uminus-uminus[symmetric]) (simp only: eint-minus-less-minus)

lemma eint-less-uminus-reorder: $a < -b \longleftrightarrow b < -(a::eint)$

by (subst (3) eint-uminus-uminus[symmetric]) (simp only: eint-minus-less-minus)

lemma eint-uminus-le-reorder: $-a \leq b \longleftrightarrow -b \leq (a::eint)$

by (subst (3) eint-uminus-uminus[symmetric]) (simp only: eint-minus-le-minus)

lemmas eint-uminus-reorder =

eint-uminus-eq-reorder eint-uminus-less-reorder eint-uminus-le-reorder

lemma eint-bot:

fixes x :: eint
 assumes $\bigwedge B. x \leq eint\ B$
 shows $x = -\infty$

proof (cases x)

case (int r)

with assms[of r - 1] show ?thesis

by auto

```

next
  case PInf
  with assms[of 0] show ?thesis
  by auto
next
  case MInf
  then show ?thesis
  by simp
qed

```

```

lemma eint-top:
  fixes  $x :: \text{eint}$ 
  assumes  $\bigwedge B. x \geq \text{eint } B$ 
  shows  $x = \infty$ 
proof (cases  $x$ )
  case (int  $r$ )
  with assms[of  $r + 1$ ] show ?thesis
  by auto
next
  case MInf
  with assms[of 0] show ?thesis
  by auto
next
  case PInf
  then show ?thesis
  by simp
qed

```

```

lemma
  shows eint-max[simp]:  $\text{eint } (\max x y) = \max (\text{eint } x) (\text{eint } y)$ 
  and eint-min[simp]:  $\text{eint } (\min x y) = \min (\text{eint } x) (\text{eint } y)$ 
  by (simp-all add: min-def max-def)

```

```

lemma eint-max-0:  $\max 0 (\text{eint } r) = \text{eint } (\max 0 r)$ 
  by (auto simp: zero-eint-def)

```

```

lemma
  fixes  $f :: \text{nat} \Rightarrow \text{eint}$ 
  shows eint-incseq-uminus[simp]:  $\text{incseq } (\lambda x. - f x) \longleftrightarrow \text{decseq } f$ 
  and eint-decseq-uminus[simp]:  $\text{decseq } (\lambda x. - f x) \longleftrightarrow \text{incseq } f$ 
  unfolding decseq-def incseq-def by auto

```

```

lemma incseq-eint:  $\text{incseq } f \implies \text{incseq } (\lambda x. \text{eint } (f x))$ 
  unfolding incseq-def by auto

```

```

lemma sum-eint[simp]:  $(\sum x \in A. \text{eint } (f x)) = \text{eint } (\sum x \in A. f x)$ 
proof (cases finite  $A$ )
  case True
  then show ?thesis by induct auto
next

```

```

case False
then show ?thesis by simp
qed

```

```

lemma sum-list-eint [simp]: sum-list (map (λx. eint (f x)) xs) = eint (sum-list (map f xs))
by (induction xs) simp-all

```

```

lemma sum-Pinfy:
fixes f :: 'a ⇒ eint
shows (∑ x∈P. f x) = ∞ ⟷ finite P ∧ (∃ i∈P. f i = ∞)

```

```

proof safe
assume *: sum f P = ∞
show finite P
proof (rule ccontr)
assume ¬ finite P
with * show False
by auto
qed
show ∃ i∈P. f i = ∞
proof (rule ccontr)
assume ¬ ?thesis
then have ∧i. i ∈ P ⇒ f i ≠ ∞
by auto
with ⟨finite P⟩ have sum f P ≠ ∞
by induct auto
with * show False
by auto
qed

```

```

next
fix i
assume finite P and i ∈ P and f i = ∞
then show sum f P = ∞
proof induct
case (insert x A)
show ?case using insert by (cases x = i) auto
qed simp
qed

```

```

lemma sum-Inf:
fixes f :: 'a ⇒ eint
shows |sum f A| = ∞ ⟷ finite A ∧ (∃ i∈A. |f i| = ∞)

```

```

proof
assume *: |sum f A| = ∞
have finite A
by (rule ccontr) (insert *, auto)
moreover have ∃ i∈A. |f i| = ∞
proof (rule ccontr)
assume ¬ ?thesis
then have ∀ i∈A. ∃ r. f i = eint r
by auto

```

```

    from bchoice[OF this] obtain r where  $\forall x \in A. f x = \text{eint } (r x)$  ..
    with * show False
      by auto
qed
ultimately show  $\text{finite } A \wedge (\exists i \in A. |f i| = \infty)$ 
  by auto
next
assume  $\text{finite } A \wedge (\exists i \in A. |f i| = \infty)$ 
then obtain i where  $\text{finite } A \ i \in A$  and  $|f i| = \infty$ 
  by auto
then show  $|\text{sum } f A| = \infty$ 
proof induct
  case (insert j A)
  then show ?case
    by (cases rule: eint3-cases[of f i f j sum f A]) auto
qed simp
qed

```

lemma *sum-int-of-eint*:

```

  fixes f :: 'i  $\Rightarrow$  eint
  assumes  $\bigwedge x. x \in S \implies |f x| \neq \infty$ 
  shows  $(\sum x \in S. \text{int-of-eint } (f x)) = \text{int-of-eint } (\text{sum } f S)$ 
proof -
  have  $\forall x \in S. \exists r. f x = \text{eint } r$ 
  proof
    fix x
    assume  $x \in S$ 
    from assms[OF this] show  $\exists r. f x = \text{eint } r$ 
      by (cases f x) auto
  qed
  from bchoice[OF this] obtain r where  $\forall x \in S. f x = \text{eint } (r x)$  ..
  then show ?thesis
    by simp
qed

```

lemma *sum-eint-0*:

```

  fixes f :: 'a  $\Rightarrow$  eint
  assumes  $\text{finite } A$ 
  and  $\bigwedge i. i \in A \implies 0 \leq f i$ 
  shows  $(\sum x \in A. f x) = 0 \iff (\forall i \in A. f i = 0)$ 
proof
  assume  $\text{sum } f A = 0$  with assms show  $\forall i \in A. f i = 0$ 
  proof (induction A)
    case (insert a A)
    then have  $f a = 0 \wedge (\sum a \in A. f a) = 0$ 
      by (subst eint-add-nonneg-eq-0-iff[symmetric]) (simp-all add: sum-nonneg)
    with insert show ?case
      by simp
  qed simp
qed auto

```

Multiplication

instantiation *eint* :: {*comm-monoid-mult*,*sgn*}

begin

function *sgn-eint* :: *eint* \Rightarrow *eint* **where**

sgn (*eint* *r*) = *eint* (*sgn* *r*)

| *sgn* ($\infty::eint$) = 1

| *sgn* ($-\infty::eint$) = -1

by (*auto intro: eint-cases*)

termination by *standard* (*rule wf-on-bot*)

function *times-eint* **where**

eint *r* * *eint* *p* = *eint* (*r* * *p*)

| *eint* *r* * ∞ = (if *r* = 0 then 0 else if *r* > 0 then ∞ else $-\infty$)

| ∞ * *eint* *r* = (if *r* = 0 then 0 else if *r* > 0 then ∞ else $-\infty$)

| *eint* *r* * $-\infty$ = (if *r* = 0 then 0 else if *r* > 0 then $-\infty$ else ∞)

| $-\infty$ * *eint* *r* = (if *r* = 0 then 0 else if *r* > 0 then $-\infty$ else ∞)

| ($\infty::eint$) * ∞ = ∞

| $-(\infty::eint)$ * ∞ = $-\infty$

| ($\infty::eint$) * $-\infty$ = $-\infty$

| $-(\infty::eint)$ * $-\infty$ = ∞

proof *goal-cases*

case *prems*: (1 *P* *x*)

then obtain *a* *b* **where** *x* = (*a*, *b*)

by (*cases* *x*) *auto*

with *prems* **show** *P*

by (*cases rule: eint2-cases*[of *a* *b*]) *auto*

qed *simp-all*

termination by (*relation* {}) *simp*

instance

proof

fix *a* *b* *c* :: *eint*

show 1 * *a* = *a*

by (*cases* *a*) (*simp-all add: one-eint-def*)

show *a* * *b* = *b* * *a*

by (*cases rule: eint2-cases*[of *a* *b*]) *simp-all*

show *a* * *b* * *c* = *a* * (*b* * *c*)

by (*cases rule: eint3-cases*[of *a* *b* *c*])

(*simp-all add: zero-eint-def zero-less-mult-iff*)

qed

end

lemma [*simp*]:

shows *eint-1-times*: *eint* 1 * *x* = *x*

and *times-eint-1*: *x* * *eint* 1 = *x*

by(*simp-all flip: one-eint-def*)

lemma *one-not-le-zero-eint*[*simp*]: $\neg (1 \leq (0::eint))$

by (simp add: one-eint-def zero-eint-def)

lemma *int-eint-1*[simp]: *int-of-eint* (1::eint) = 1
unfolding one-eint-def by simp

lemma *int-of-eint-le-1*:
fixes *a* :: eint
shows $a \leq 1 \implies \text{int-of-eint } a \leq 1$
by (cases *a*) (auto simp: one-eint-def)

lemma *abs-eint-one*[simp]: |1| = (1::eint)
unfolding one-eint-def by simp

lemma *eint-mult-zero*[simp]:
fixes *a* :: eint
shows $a * 0 = 0$
by (cases *a*) (simp-all add: zero-eint-def)

lemma *eint-zero-mult*[simp]:
fixes *a* :: eint
shows $0 * a = 0$
by (cases *a*) (simp-all add: zero-eint-def)

lemma *eint-m1-less-0*[simp]: $-(1::\text{eint}) < 0$
by (simp add: zero-eint-def one-eint-def)

lemma *eint-times*[simp]:
 $1 \neq (\infty::\text{eint})$ $(\infty::\text{eint}) \neq 1$
 $1 \neq -(\infty::\text{eint})$ $-(\infty::\text{eint}) \neq 1$
by (auto simp: one-eint-def)

lemma *eint-plus-1*[simp]:
 $1 + \text{eint } r = \text{eint } (r + 1)$
 $\text{eint } r + 1 = \text{eint } (r + 1)$
 $1 + -(\infty::\text{eint}) = -\infty$
 $-(\infty::\text{eint}) + 1 = -\infty$
unfolding one-eint-def by auto

lemma *eint-zero-times*[simp]:
fixes *a b* :: eint
shows $a * b = 0 \iff a = 0 \vee b = 0$
by (cases rule: eint2-cases[of *a b*]) auto

lemma *eint-mult-eq-PInfty*[simp]:
 $a * b = (\infty::\text{eint}) \iff$
 $(a = \infty \wedge b > 0) \vee (a > 0 \wedge b = \infty) \vee (a = -\infty \wedge b < 0) \vee (a < 0 \wedge b = -\infty)$
by (cases rule: eint2-cases[of *a b*]) auto

lemma *eint-mult-eq-MInfty*[simp]:
 $a * b = -(\infty::\text{eint}) \iff$

$(a = \infty \wedge b < 0) \vee (a < 0 \wedge b = \infty) \vee (a = -\infty \wedge b > 0) \vee (a > 0 \wedge b = -\infty)$
by (*cases rule: eint2-cases[of a b]*) *auto*

lemma *eint-abs-mult*: $|x * y :: \text{eint}| = |x| * |y|$
by (*cases x y rule: eint2-cases*) (*auto simp: abs-mult*)

lemma *eint-0-less-1[simp]*: $0 < (1 :: \text{eint})$
by (*simp-all add: zero-eint-def one-eint-def*)

lemma *eint-mult-minus-left[simp]*:
fixes $a \ b :: \text{eint}$
shows $-a * b = -(a * b)$
by (*cases rule: eint2-cases[of a b]*) *auto*

lemma *eint-mult-minus-right[simp]*:
fixes $a \ b :: \text{eint}$
shows $a * -b = -(a * b)$
by (*cases rule: eint2-cases[of a b]*) *auto*

lemma *eint-mult-infity[simp]*:
 $a * (\infty :: \text{eint}) = (\text{if } a = 0 \text{ then } 0 \text{ else if } 0 < a \text{ then } \infty \text{ else } -\infty)$
by (*cases a*) *auto*

lemma *eint-infity-mult[simp]*:
 $(\infty :: \text{eint}) * a = (\text{if } a = 0 \text{ then } 0 \text{ else if } 0 < a \text{ then } \infty \text{ else } -\infty)$
by (*cases a*) *auto*

lemma *eint-mult-strict-right-mono*:
assumes $a < b$
and $0 < c$
and $c < (\infty :: \text{eint})$
shows $a * c < b * c$
using *assms*
by (*cases rule: eint3-cases[of a b c]*) (*auto simp: zero-le-mult-iff*)

lemma *eint-mult-strict-left-mono*:
 $a < b \implies 0 < c \implies c < (\infty :: \text{eint}) \implies c * a < c * b$
using *eint-mult-strict-right-mono*
by (*simp add: mult.commute[of c]*)

lemma *eint-mult-right-mono*:
fixes $a \ b \ c :: \text{eint}$
assumes $a \leq b \ 0 \leq c$
shows $a * c \leq b * c$
proof (*cases c = 0*)
case *False*
with *assms show ?thesis*
by (*cases rule: eint3-cases[of a b c]*) *auto*
qed *auto*

lemma *eint-mult-left-mono*:

fixes $a\ b\ c :: \text{eint}$

shows $a \leq b \implies 0 \leq c \implies c * a \leq c * b$

using *eint-mult-right-mono*

by (*simp add: mult.commute[of c]*)

lemma *eint-mult-mono*:

fixes $a\ b\ c\ d :: \text{eint}$

assumes $b \geq 0\ c \geq 0\ a \leq b\ c \leq d$

shows $a * c \leq b * d$

by (*metis eint-mult-right-mono mult.commute order-trans assms*)

lemma *eint-mult-mono'*:

fixes $a\ b\ c\ d :: \text{eint}$

assumes $a \geq 0\ c \geq 0\ a \leq b\ c \leq d$

shows $a * c \leq b * d$

by (*metis eint-mult-right-mono mult.commute order-trans assms*)

lemma *eint-mult-mono-strict*:

fixes $a\ b\ c\ d :: \text{eint}$

assumes $b > 0\ c > 0\ a < b\ c < d$

shows $a * c < b * d$

proof –

have $c < \infty$ **using** $\langle c < d \rangle$ **by** *auto*

then have $a * c < b * c$ **by** (*metis eint-mult-strict-left-mono[OF assms(3) assms(2)] mult.commute*)

moreover have $b * c \leq b * d$ **using** *assms(2) assms(4)* **by** (*simp add: assms(1) eint-mult-left-mono less-imp-le*)

ultimately show *?thesis* **by** *simp*

qed

lemma *eint-mult-mono-strict'*:

fixes $a\ b\ c\ d :: \text{eint}$

assumes $a > 0\ c > 0\ a < b\ c < d$

shows $a * c < b * d$

using *assms eint-mult-mono-strict* **by** *auto*

lemma *zero-less-one-eint[simp]*: $0 \leq (1 :: \text{eint})$

by (*simp add: one-eint-def zero-eint-def*)

lemma *eint-0-le-mult[simp]*: $0 \leq a \implies 0 \leq b \implies 0 \leq a * (b :: \text{eint})$

by (*cases rule: eint2-cases[of a b]*) *auto*

lemma *eint-right-distrib*:

fixes $r\ a\ b :: \text{eint}$

shows $0 \leq a \implies 0 \leq b \implies r * (a + b) = r * a + r * b$

by (*cases rule: eint3-cases[of r a b]*) (*simp-all add: field-simps*)

lemma *eint-left-distrib*:

fixes $r\ a\ b :: \text{eint}$

shows $0 \leq a \implies 0 \leq b \implies (a + b) * r = a * r + b * r$

by (cases rule: eint3-cases[of r a b]) (simp-all add: field-simps)

lemma eint-mult-le-0-iff:

fixes a b :: eint

shows $a * b \leq 0 \longleftrightarrow (0 \leq a \wedge b \leq 0) \vee (a \leq 0 \wedge 0 \leq b)$

by (cases rule: eint2-cases[of a b]) (simp-all add: mult-le-0-iff)

lemma eint-zero-le-0-iff:

fixes a b :: eint

shows $0 \leq a * b \longleftrightarrow (0 \leq a \wedge 0 \leq b) \vee (a \leq 0 \wedge b \leq 0)$

by (cases rule: eint2-cases[of a b]) (simp-all add: zero-le-mult-iff)

lemma eint-mult-less-0-iff:

fixes a b :: eint

shows $a * b < 0 \longleftrightarrow (0 < a \wedge b < 0) \vee (a < 0 \wedge 0 < b)$

by (cases rule: eint2-cases[of a b]) (simp-all add: mult-less-0-iff)

lemma eint-zero-less-0-iff:

fixes a b :: eint

shows $0 < a * b \longleftrightarrow (0 < a \wedge 0 < b) \vee (a < 0 \wedge b < 0)$

by (cases rule: eint2-cases[of a b]) (simp-all add: zero-less-mult-iff)

lemma eint-left-mult-cong:

fixes a b c :: eint

shows $c = d \implies (d \neq 0 \implies a = b) \implies a * c = b * d$

by (cases c = 0) simp-all

lemma eint-right-mult-cong:

fixes a b c :: eint

shows $c = d \implies (d \neq 0 \implies a = b) \implies c * a = d * b$

by (cases c = 0) simp-all

lemma eint-distrib:

fixes a b c :: eint

assumes $a \neq \infty \vee b \neq -\infty$

and $a \neq -\infty \vee b \neq \infty$

and $|c| \neq \infty$

shows $(a + b) * c = a * c + b * c$

using assms

by (cases rule: eint3-cases[of a b c]) (simp-all add: field-simps)

lemma numeral-eq-eint [simp]: numeral w = eint (numeral w)

proof (induct w rule: num-induct)

case One

then show ?case

by simp

next

case (inc x)

then show ?case

by (simp add: inc numeral-inc)

qed

lemma *distrib-left-eint-nn*:

$c \geq 0 \implies (x + y) * \text{eint } c = x * \text{eint } c + y * \text{eint } c$
by(cases $x \ y$ rule: *eint2-cases*)(simp-all add: *ring-distrib*)

lemma *sum-eint-right-distrib*:

fixes $f :: 'a \Rightarrow \text{eint}$
shows $(\bigwedge i. i \in A \implies 0 \leq f \ i) \implies r * \text{sum } f \ A = (\sum n \in A. r * f \ n)$
by (*induct A* rule: *infinite-finite-induct*) (*auto simp: eint-right-distrib sum-nonneg*)

lemma *sum-eint-left-distrib*:

$(\bigwedge i. i \in A \implies 0 \leq f \ i) \implies \text{sum } f \ A * r = (\sum n \in A. f \ n * r :: \text{eint})$
using *sum-eint-right-distrib*[of $A \ f \ r$] **by** (*simp add: mult-ac*)

lemma *sum-distrib-right-eint*:

$c \geq 0 \implies \text{sum } f \ A * \text{eint } c = (\sum x \in A. f \ x * c :: \text{eint})$
by(subst *sum-comp-morphism*[**where** $h = \lambda x. x * \text{eint } c$, *symmetric*])(simp-all add: *distrib-left-eint-nn*)

lemma *eint-le-int*:

fixes $x \ y :: \text{eint}$
assumes $\bigwedge z. x \leq \text{eint } z \implies y \leq \text{eint } z$
shows $y \leq x$
by (*metis assms eint-bot eint-cases eint-infty-less-eq(2) eint-less-eq(1) linorder-le-cases*)

lemma *prod-eint-0*:

fixes $f :: 'a \Rightarrow \text{eint}$
shows $(\prod i \in A. f \ i) = 0 \longleftrightarrow \text{finite } A \wedge (\exists i \in A. f \ i = 0)$
proof (*cases finite A*)
 case *True*
 then show *?thesis* **by** (*induct A*) *auto*
qed *auto*

lemma *prod-eint-pos*:

fixes $f :: 'a \Rightarrow \text{eint}$
assumes *pos*: $\bigwedge i. i \in I \implies 0 \leq f \ i$
shows $0 \leq (\prod i \in I. f \ i)$
proof (*cases finite I*)
 case *True*
 from *this pos* **show** *?thesis*
 by *induct auto*
qed *auto*

lemma *prod-PInf*:

fixes $f :: 'a \Rightarrow \text{eint}$

```

assumes  $\bigwedge i. i \in I \implies 0 \leq f\ i$ 
shows  $(\prod_{i \in I}. f\ i) = \infty \iff \text{finite } I \wedge (\exists i \in I. f\ i = \infty) \wedge (\forall i \in I. f\ i \neq 0)$ 
proof (cases finite I)
  case True
  from this assms show ?thesis
proof (induct I)
  case (insert i I)
  then have pos:  $0 \leq f\ i \leq \text{prod } f\ I$ 
    by (auto intro!: prod-eint-pos)
  from insert have  $(\prod_{j \in \text{insert } i\ I}. f\ j) = \infty \iff \text{prod } f\ I * f\ i = \infty$ 
    by auto
  also have ...  $\iff (\text{prod } f\ I = \infty \vee f\ i = \infty) \wedge f\ i \neq 0 \wedge \text{prod } f\ I \neq 0$ 
    using prod-eint-pos[of I f] pos
    by (cases rule: eint2-cases[of f i prod f I]) auto
  also have ...  $\iff \text{finite } (\text{insert } i\ I) \wedge (\exists j \in \text{insert } i\ I. f\ j = \infty) \wedge (\forall j \in \text{insert } i\ I. f\ j \neq 0)$ 
    using insert by (auto simp: prod-eint-0)
  finally show ?case .
qed simp
qed auto

```

```

lemma prod-eint:  $(\prod_{i \in A}. \text{eint } (f\ i)) = \text{eint } (\text{prod } f\ A)$ 
proof (cases finite A)
  case True
  then show ?thesis
    by induct (auto simp: one-eint-def)
next
  case False
  then show ?thesis
    by (simp add: one-eint-def)
qed

```

Power

```

lemma eint-power[simp]:  $(\text{eint } x) \wedge n = \text{eint } (x \wedge n)$ 
  by (induct n) (auto simp: one-eint-def)

```

```

lemma eint-power-PInf[simp]:  $(\infty :: \text{eint}) \wedge n = (\text{if } n = 0 \text{ then } 1 \text{ else } \infty)$ 
  by (induct n) (auto simp: one-eint-def)

```

```

lemma eint-power-uminus[simp]:
  fixes x :: eint
  shows  $(- x) \wedge n = (\text{if even } n \text{ then } x \wedge n \text{ else } - (x \wedge n))$ 
  by (induct n) (auto simp: one-eint-def)

```

```

lemma eint-power-numeral[simp]:
   $(\text{numeral num} :: \text{eint}) \wedge n = \text{eint } (\text{numeral num} \wedge n)$ 
  by (induct n) (auto simp: one-eint-def)

```

```

lemma zero-le-power-eint[simp]:
  fixes a :: eint

```

```

assumes  $0 \leq a$ 
shows  $0 \leq a \wedge n$ 
using assms by (induct n) (auto simp: eint-zero-le-0-iff)

```

Subtraction

```

lemma eint-minus-minus-image[simp]:
  fixes  $S :: \text{eint set}$ 
  shows  $\text{uminus } 'S = S$ 
  by (auto simp: image-iff)

```

```

lemma eint-uminus-lessThan[simp]:
  fixes  $a :: \text{eint}$ 
  shows  $\text{uminus } '\{..<a\} = \{-a<..\}$ 
proof -
  {
    fix  $x$ 
    assume  $-a < x$ 
    then have  $-x < -(-a)$ 
      by (simp del: eint-uminus-uminus)
    then have  $-x < a$ 
      by simp
  }
  then show ?thesis
    by force
qed

```

```

lemma eint-uminus-greaterThan[simp]:  $\text{uminus } '\{(a::\text{eint})<..\} = \{..<-a\}$ 
  by (metis eint-uminus-lessThan eint-uminus-uminus eint-minus-minus-image)

```

```

instantiation eint :: minus
begin

```

```

definition  $x - y = x + -(y::\text{eint})$ 
instance ..

```

```

end

```

```

lemma eint-minus[simp]:
   $\text{eint } r - \text{eint } p = \text{eint } (r - p)$ 
   $-\infty - \text{eint } r = -\infty$ 
   $\text{eint } r - \infty = -\infty$ 
   $(\infty::\text{eint}) - x = \infty$ 
   $-(\infty::\text{eint}) - \infty = -\infty$ 
   $x - -y = x + y$ 
   $x - 0 = x$ 
   $0 - x = -x$ 
  by (simp-all add: minus-eint-def)

```

```

lemma eint-x-minus-x[simp]:  $x - x = (\text{if } |x| = \infty \text{ then } \infty \text{ else } 0::\text{eint})$ 

```

by (cases x) simp-all

lemma *eint-eq-minus-iff*:

fixes $x\ y\ z :: \text{eint}$

shows $x = z - y \longleftrightarrow$

$(|y| \neq \infty \longrightarrow x + y = z) \wedge$

$(y = -\infty \longrightarrow x = \infty) \wedge$

$(y = \infty \longrightarrow z = \infty \longrightarrow x = \infty) \wedge$

$(y = \infty \longrightarrow z \neq \infty \longrightarrow x = -\infty)$

by (cases rule: eint3-cases[of x y z]) auto

lemma *eint-eq-minus*:

fixes $x\ y\ z :: \text{eint}$

shows $|y| \neq \infty \implies x = z - y \longleftrightarrow x + y = z$

by (auto simp: eint-eq-minus-iff)

lemma *eint-less-minus-iff*:

fixes $x\ y\ z :: \text{eint}$

shows $x < z - y \longleftrightarrow$

$(y = \infty \longrightarrow z = \infty \wedge x \neq \infty) \wedge$

$(y = -\infty \longrightarrow x \neq \infty) \wedge$

$(|y| \neq \infty \longrightarrow x + y < z)$

by (cases rule: eint3-cases[of x y z]) auto

lemma *eint-less-minus*:

fixes $x\ y\ z :: \text{eint}$

shows $|y| \neq \infty \implies x < z - y \longleftrightarrow x + y < z$

by (auto simp: eint-less-minus-iff)

lemma *eint-le-minus-iff*:

fixes $x\ y\ z :: \text{eint}$

shows $x \leq z - y \longleftrightarrow (y = \infty \longrightarrow z \neq \infty \longrightarrow x = -\infty) \wedge (|y| \neq \infty \longrightarrow x + y \leq z)$

by (cases rule: eint3-cases[of x y z]) auto

lemma *eint-le-minus*:

fixes $x\ y\ z :: \text{eint}$

shows $|y| \neq \infty \implies x \leq z - y \longleftrightarrow x + y \leq z$

by (auto simp: eint-le-minus-iff)

lemma *eint-minus-less-iff*:

fixes $x\ y\ z :: \text{eint}$

shows $x - y < z \longleftrightarrow y \neq -\infty \wedge (y = \infty \longrightarrow x \neq \infty \wedge z \neq -\infty) \wedge (y \neq \infty \longrightarrow x < z + y)$

by (cases rule: eint3-cases[of x y z]) auto

lemma *eint-minus-less*:

fixes $x\ y\ z :: \text{eint}$

shows $|y| \neq \infty \implies x - y < z \longleftrightarrow x < z + y$

by (auto simp: eint-minus-less-iff)

lemma *eint-minus-le-iff*:

fixes $x\ y\ z :: \text{eint}$
shows $x - y \leq z \longleftrightarrow$
 $(y = -\infty \longrightarrow z = \infty) \wedge$
 $(y = \infty \longrightarrow x = \infty \longrightarrow z = \infty) \wedge$
 $(|y| \neq \infty \longrightarrow x \leq z + y)$
by (*cases rule: eint3-cases[of x y z]*) *auto*

lemma *eint-minus-le:*

fixes $x\ y\ z :: \text{eint}$
shows $|y| \neq \infty \implies x - y \leq z \longleftrightarrow x \leq z + y$
by (*auto simp: eint-minus-le-iff*)

lemma *eint-minus-eq-minus-iff:*

fixes $a\ b\ c :: \text{eint}$
shows $a - b = a - c \longleftrightarrow$
 $b = c \vee a = \infty \vee (a = -\infty \wedge b \neq -\infty \wedge c \neq -\infty)$
by (*cases rule: eint3-cases[of a b c]*) *auto*

lemma *eint-add-le-add-iff:*

fixes $a\ b\ c :: \text{eint}$
shows $c + a \leq c + b \longleftrightarrow$
 $a \leq b \vee c = \infty \vee (c = -\infty \wedge a \neq \infty \wedge b \neq \infty)$
by (*cases rule: eint3-cases[of a b c]*) (*simp-all add: field-simps*)

lemma *eint-add-le-add-iff2:*

fixes $a\ b\ c :: \text{eint}$
shows $a + c \leq b + c \longleftrightarrow a \leq b \vee c = \infty \vee (c = -\infty \wedge a \neq \infty \wedge b \neq \infty)$
by (*cases rule: eint3-cases[of a b c]*) (*simp-all add: field-simps*)

lemma *eint-mult-le-mult-iff:*

fixes $a\ b\ c :: \text{eint}$
shows $|c| \neq \infty \implies c * a \leq c * b \longleftrightarrow (0 < c \longrightarrow a \leq b) \wedge (c < 0 \longrightarrow b \leq a)$
by (*cases rule: eint3-cases[of a b c]*) (*simp-all add: mult-le-cancel-left*)

lemma *eint-minus-mono:*

fixes $A\ B\ C\ D :: \text{eint}$ **assumes** $A \leq B\ D \leq C$
shows $A - C \leq B - D$
using *assms*
by (*cases rule: eint3-cases[case-product eint-cases, of A B C D]*) *simp-all*

lemma *eint-mono-minus-cancel:*

fixes $a\ b\ c :: \text{eint}$
shows $c - a \leq c - b \implies 0 \leq c \implies c < \infty \implies b \leq a$
by (*cases a b c rule: eint3-cases*) *auto*

lemma *int-of-eint-minus:*

fixes $a\ b :: \text{eint}$
shows $\text{int-of-eint } (a - b) = (\text{if } |a| = \infty \vee |b| = \infty \text{ then } 0 \text{ else } \text{int-of-eint } a - \text{int-of-eint } b)$
by (*cases rule: eint2-cases[of a b]*) *auto*

lemma *int-of-eint-minus'*: $|x| = \infty \longleftrightarrow |y| = \infty \implies \text{int-of-eint } x - \text{int-of-eint } y = \text{int-of-eint } (x - y :: \text{eint})$
by(*subst int-of-eint-minus*) *auto*

lemma *eint-diff-positive*:
fixes $a\ b :: \text{eint}$ **shows** $a \leq b \implies 0 \leq b - a$
by (*cases rule: eint2-cases[of a b]*) *auto*

lemma *eint-between*:
fixes $x\ e :: \text{eint}$
assumes $|x| \neq \infty$
and $0 < e$
shows $x - e < x$
and $x < x + e$
using *assms* **by** (*cases x, cases e, auto*)**+**

lemma *eint-minus-eq-PInfty-iff*:
fixes $x\ y :: \text{eint}$
shows $x - y = \infty \longleftrightarrow y = -\infty \vee x = \infty$
by (*cases x y rule: eint2-cases*) *simp-all*

lemma *eint-diff-add-eq-diff-diff-swap*:
fixes $x\ y\ z :: \text{eint}$
shows $|y| \neq \infty \implies x - (y + z) = x - y - z$
by(*cases x y z rule: eint3-cases*) *simp-all*

lemma *eint-diff-add-assoc2*:
fixes $x\ y\ z :: \text{eint}$
shows $x + y - z = x - z + y$
by(*cases x y z rule: eint3-cases*) *simp-all*

lemma *eint-add-uminus-conv-diff*: **fixes** $x\ y\ z :: \text{eint}$ **shows** $-x + y = y - x$
by(*cases x y rule: eint2-cases*) *simp-all*

lemma *eint-minus-diff-eq*:
fixes $x\ y :: \text{eint}$
shows $\llbracket x = \infty \longrightarrow y \neq \infty; x = -\infty \longrightarrow y \neq -\infty \rrbracket \implies -(x - y) = y - x$
by(*cases x y rule: eint2-cases*) *simp-all*

lemma *ediff-le-self* [*simp*]: $x - y \leq (x :: \text{enat})$
by(*cases x y rule: enat.exhaust[case-product enat.exhaust]*) *simp-all*

lemma *eint-abs-diff*:
fixes $a\ b :: \text{eint}$
shows $\text{abs}(a - b) \leq \text{abs } a + \text{abs } b$
by (*cases rule: eint2-cases[of a b]*) (*auto*)

1.4.1 Distributive lattice

instantiation *eint* :: *distrib-lattice*

begin

definition $[simp]: \sup x y = (\max x y :: \text{eint})$

definition $[simp]: \inf x y = (\min x y :: \text{eint})$

instance **by** *standard*

(*auto simp add: inf-int-def sup-int-def max-min-distrib2*)

end

lemma *min-PInf* $[simp]: \min (\infty :: \text{eint}) x = x$
by *simp*

lemma *min-PInf2* $[simp]: \min x (\infty :: \text{eint}) = x$
by *simp*

lemma *max-PInf* $[simp]: \max (\infty :: \text{eint}) x = \infty$
by *simp*

lemma *max-PInf2* $[simp]: \max x (\infty :: \text{eint}) = \infty$
by *simp*

lemma *min-MInf* $[simp]: \min (-\infty :: \text{eint}) x = -\infty$
by *simp*

lemma *min-MInf2* $[simp]: \min x (-\infty :: \text{eint}) = -\infty$
by *simp*

lemma *max-MInf* $[simp]: \max (-\infty :: \text{eint}) x = x$
by *simp*

lemma *max-MInf2* $[simp]: \max x (-\infty :: \text{eint}) = x$
by *simp*

instantiation *eint* :: $\{\text{order-bot}, \text{order-top}\}$
begin

definition *bot-eint* :: *eint* **where** *bot-eint* = $-\infty$

definition *top-eint* :: *eint* **where** *top-eint* = ∞

instance

by *standard (simp-all add: bot-eint-def top-eint-def)*

end

lemma *finite-eint-bounded-below-above*:

assumes *le-fin*: $\bigwedge y. y \in A \implies \text{eint } m \leq y \wedge y \leq \text{eint } n$
shows *finite* *A*

proof (*rule finite-subset*)

show *finite* (*eint* ‘ $\{m..n\}$ ’) **by** *blast*

have $A \subseteq \{\text{eint } m.. \text{eint } n\}$ **using** *le-fin* **by** *fastforce*

also have $\dots \subseteq \text{eint } ‘\{m..n\}$

apply (*rule subsetI*)

subgoal for *x* **by** (*cases x*) *auto*

done

finally show $A \subseteq \text{eint } ‘\{m..n\}$.

qed

lemma *infinite-eint-bounded-above*:

infinite (*eint* ‘ $\{..n\}$ ’)

by (*meson finite-imageD infinite-Iic inj-eint*)

lemma *infinite-eint-bounded-below*:

infinite (*eint* ‘ $\{n.. \}$ ’)

by (*simp add: finite-image-iff infinite-Ici*)

lemma *infinite-eint-unbounded-below-or-above*:

assumes *infinite* *A*

shows $(\exists y \in A. y > \text{eint } n) \vee (\exists y \in A. \text{eint } m > y)$

using *assms finite-eint-bounded-below-above*[*of A m n*]

by *fastforce*

1.4.2 Extended int intervals

lemma *int-greaterThanLessThan-infinity-eq*:

int-of-eint ‘ $\{N::\text{eint} < .. < \infty\}$ ’ =

(*if* $N = \infty$ *then* $\{\}$ *else if* $N = -\infty$ *then* *UNIV* *else* $\{\text{int-of-eint } N < ..\}$)

by (*force simp: int-less-eint-iff intro!: image-eqI*[**where** $x = \text{eint } -$] *elim!: less-eint.elims*)

lemma *int-greaterThanLessThan-minus-infinity-eq*:

int-of-eint ‘ $\{-\infty < .. < N::\text{eint}\}$ ’ =

(*if* $N = \infty$ *then* *UNIV* *else if* $N = -\infty$ *then* $\{\}$ *else* $\{.. < \text{int-of-eint } N\}$)

proof –

have *int-of-eint* ‘ $\{-\infty < .. < N::\text{eint}\}$ ’ = *uminus* ‘*int-of-eint* ‘ $\{-N < .. < \infty\}$ ’

by (*auto simp: eint-uminus-less-reorder intro!: image-eqI*[**where** $x = -x$ **for** x])

also note *int-greaterThanLessThan-infinity-eq*

finally show *?thesis*

using *eint-uminus-eq-reorder int-greaterThanLessThan-infinity-eq* **by** *auto*

qed

lemma *int-greaterThanLessThan-inter*:

int-of-eint ‘ $\{N < .. < M::\text{eint}\}$ ’ = *int-of-eint* ‘ $\{-\infty < .. < M\}$ ’ \cap *int-of-eint* ‘ $\{N < .. < \infty\}$ ’

by (force elim!: less-eint.elims)

lemma *int-atLeastGreaterThan-eq*: *int-of-eint* ‘ $\{N <..< M :: \text{eint}\}$ =

(if $N = \infty$ then $\{\}$ else
 if $N = -\infty$ then
 (if $M = \infty$ then UNIV
 else if $M = -\infty$ then $\{\}$
 else $\{..< \text{int-of-eint } M\}$)
 else if $M = -\infty$ then $\{\}$
 else if $M = \infty$ then $\{\text{int-of-eint } N <..\}$
 else $\{\text{int-of-eint } N <..< \text{int-of-eint } M\}$)

proof (cases $M = -\infty \vee M = \infty \vee N = -\infty \vee N = \infty$)

case *True*

then show ?thesis

by (auto simp: int-greaterThanLessThan-minus-infinity-eq int-greaterThanLessThan-infinity-eq)

next

case *False*

then obtain $p \ q$ **where** $M = \text{eint } p \ N = \text{eint } q$

by (metis MInfty-eq-minfinity eint.distinct(3) uminus-eint.elims)

moreover have $\bigwedge x. \llbracket q < x; x < p \rrbracket \implies x \in \text{int-of-eint } ' \{ \text{eint } q <..< \text{eint } p \}$

by (metis greaterThanLessThan-iff imageI less-eint.simps(1) int-of-eint.simps(1))

ultimately show ?thesis

by (auto elim!: less-eint.elims)

qed

lemma *int-image-eint-ivl*:

fixes $a \ b :: \text{eint}$

shows

int-of-eint ‘ $\{a <..< b\}$ =

(if $a < b$ then (if $a = -\infty$ then if $b = \infty$ then UNIV else $\{..< \text{int-of-eint } b\}$
 else if $b = \infty$ then $\{\text{int-of-eint } a <..\}$ else $\{\text{int-of-eint } a <..< \text{int-of-eint } b\}$) else $\{\}$)

by (cases a; cases b; simp add: int-atLeastGreaterThan-eq not-less)

lemma **fixes** $a \ b \ c :: \text{eint}$

shows *not-inftyI*: $a < b \implies b < c \implies \text{abs } b \neq \infty$

by force

lemma

interval-neqs:

fixes $r \ s \ t :: \text{int}$

shows $\{r <..< s\} \neq \{t <..\}$

and $\{r <..< s\} \neq \{..< t\}$

and $\{r <..< ra\} \neq \text{UNIV}$

and $\{r <..\} \neq \{..< s\}$

and $\{r <..\} \neq \text{UNIV}$

and $\{..< r\} \neq \text{UNIV}$

and $\{\} \neq \{r <..\}$

and $\{\} \neq \{..< r\}$

subgoal

by (metis dual-order.strict-trans greaterThanLessThan-iff greaterThan-iff gt-ex not-le order-refl)

subgoal
by (*metis finite-greaterThanLessThan-int infinite-lto*)
subgoal by force
subgoal
by (*metis greaterThan-iff lessThan-iff lt-ex order-less-le order-less-trans*)
subgoal by force
subgoal by force
subgoal using *greaterThan-non-empty* **by** *blast*
subgoal using *lessThan-non-empty* **by** *blast*
done

lemma *int-greaterThanLessThan-empty-iff*:
fixes *a b :: int*
shows $(\{a <..<< b\} = \{\}) \longleftrightarrow a+1 \geq b$
using *atLeastPlusOneLessThan-greaterThanLessThan-int*
by *fastforce*

lemma *greaterThanLessThan-eq-iff*:
fixes *r s t u :: int*
shows $(\{r <..<< s\} = \{t <..<< u\}) = (r+1 \geq s \wedge u \leq t+1 \vee r = t \wedge s = u)$
using *int-greaterThanLessThan-empty-iff*[*of r s*] *int-greaterThanLessThan-empty-iff*[*of t u*]
apply *auto*
apply (*metis greaterThanLessThan-empty greaterThanLessThan-iff leD order.strict-iff-order verit-la-generic*)
apply (*metis greaterThanLessThan-empty greaterThanLessThan-iff leD order.strict-iff-order verit-la-generic*)
apply (*metis greaterThanLessThan-iff linorder-neqE-linordered-idom order-less-irrefl order-less-trans*)
by (*metis dual-order.strict-trans greaterThanLessThan-iff linorder-neqE-linordered-idom order-less-irrefl*)

lemma *int-of-eint-image-greaterThanLessThan-iff*:
 $\text{int-of-eint } ' \{a <..<< b\} = \text{int-of-eint } ' \{c <..<< d\} \longleftrightarrow (a+1 \geq b \wedge c+1 \geq d \vee a = c \wedge b = d)$
unfolding *int-atLeastGreaterThan-eq*
apply (*cases a; cases b; cases c; cases d*;
simp add: greaterThanLessThan-eq-iff interval-neqs interval-neqs[symmetric])
using *int-greaterThanLessThan-empty-iff* **by** *blast+*

lemma *uminus-image-int-of-eint-image-greaterThanLessThan*:
 $\text{uminus } ' \text{int-of-eint } ' \{l <..<< u\} = \text{int-of-eint } ' \{-u <..<< -l\}$
by (*force simp: algebra-simps eint-less-uminus-reorder*
eint-uminus-less-reorder intro: image-eqI[where x=-x for x])

lemma *add-image-int-of-eint-image-greaterThanLessThan*:
 $(+) c ' \text{int-of-eint } ' \{l <..<< u\} = \text{int-of-eint } ' \{c + l <..<< c + u\}$
apply *safe*
subgoal for *x*
using *eint-less-add*[*of c*]
by (*force simp: int-of-eint-add add commute*)

subgoal for - x

by (force simp: add.commute int-of-eint-minus eint-minus-less eint-less-minus
intro: image-eqI[where $x=x - c$])

done

lemma add2-image-int-of-eint-image-greaterThanLessThan:

($\lambda x. x + c$) ' int-of-eint ' $\{l <.. $u\} = \text{int-of-eint ' } \{l + c <.. $u + c\}$$$

using add-image-int-of-eint-image-greaterThanLessThan[of c l u]

by (metis add.commute image-cong)

lemma minus-image-int-of-eint-image-greaterThanLessThan:

($-$) c ' int-of-eint ' $\{l <.. $u\} = \text{int-of-eint ' } \{c - u <.. $c - l\}$$$

(is ?l = ?r)

proof -

have ?l = (+) c ' uminus ' int-of-eint ' $\{l <.. $u\}$ by auto$

also note uminus-image-int-of-eint-image-greaterThanLessThan

also note add-image-int-of-eint-image-greaterThanLessThan

finally show ?thesis by (simp add: minus-eint-def)

qed

lemma int-eint-bound-lemma-up:

assumes $s \in \text{int-of-eint ' } \{a <.. $b\}$$

assumes $t \notin \text{int-of-eint ' } \{a <.. $b\}$$

assumes $s \leq t$

shows $b \neq \infty$

proof (cases b)

case PInf

then show ?thesis

using assms

apply clarsimp

by (metis UNIV-I assms(1) eint-less-PInfy greaterThan-iff less-eq-eint-def less-le-trans int-image-eint-ivl)

qed auto

lemma int-eint-bound-lemma-down:

assumes $s: s \in \text{int-of-eint ' } \{a <.. $b\}$$

and $t: t \notin \text{int-of-eint ' } \{a <.. $b\}$$

and $t \leq s$

shows $a \neq -\infty$

proof (cases b)

case (int r)

then show ?thesis

using assms int-greaterThanLessThan-minus-infinity-eq by force

next

case PInf

then show ?thesis

using t int-greaterThanLessThan-infinity-eq by auto

next

case MInf

then show ?thesis

using s by auto

qed

lemma *eint-less-le-alt*:

$eint\ i < j \implies eint\ (i+1) \leq j$

using *eint-less-eq*(\mathcal{J})[*of i+1*]

by (*metis eint-less-eint-Ex eint-less-le less-iff-succ-less-eq linorder-not-less order.asym*)

lemma *exists-eint-int*:

assumes $i \neq -\infty$

$i \neq \infty$

shows $\exists n. i = eint\ n$

using *assms*

by (*metis PInfy-eq-infinity eint.exhaust uminus-eint.simps*(2))

1.4.3 Relation to *enat*

definition *eint-of-enat* $n = (\text{case } n \text{ of } enat\ n \Rightarrow eint\ (int\ n) \mid \infty \Rightarrow \infty)$

declare $[[coercion\ eint-of-enat :: enat \Rightarrow eint]]$

declare $[[coercion\ (\lambda n. eint\ (int\ n)) :: nat \Rightarrow eint]]$

lemma *eint-of-enat-simps*[*simp*]:

$eint-of-enat\ (enat\ n) = eint\ n$

$eint-of-enat\ \infty = \infty$

by (*simp-all add: eint-of-enat-def*)

lemma *eint-of-enat-le-iff*[*simp*]:

$eint-of-enat\ m \leq eint-of-enat\ n \longleftrightarrow m \leq n$

by (*cases m n rule: enat2-cases*) *auto*

lemma *eint-of-enat-less-iff*[*simp*]:

$eint-of-enat\ m < eint-of-enat\ n \longleftrightarrow m < n$

by (*cases m n rule: enat2-cases*) *auto*

lemma *numeral-le-eint-of-enat-iff*[*simp*]:

$numeral\ m \leq eint-of-enat\ n \longleftrightarrow numeral\ m \leq n$

by (*cases n*) *auto*

lemma *numeral-less-eint-of-enat-iff*[*simp*]:

$numeral\ m < eint-of-enat\ n \longleftrightarrow numeral\ m < n$

by (*cases n*) *auto*

lemma *eint-of-enat-ge-zero-cancel-iff*[*simp*]:

$0 \leq eint-of-enat\ n \longleftrightarrow 0 \leq n$

by (*cases n*) (*auto simp flip: enat-0*)

lemma *eint-of-enat-gt-zero-cancel-iff*[simp]:

$0 < \text{eint-of-enat } n \iff 0 < n$

by (cases *n*) (auto simp flip: enat-0)

lemma *eint-of-enat-zero*[simp]:

$\text{eint-of-enat } 0 = 0$

by (auto simp flip: enat-0)

lemma *eint-of-enat-inf*[simp]:

$\text{eint-of-enat } n = \infty \iff n = \infty$

by (cases *n*) auto

lemma *eint-of-enat-add*:

$\text{eint-of-enat } (m + n) = \text{eint-of-enat } m + \text{eint-of-enat } n$

by (cases *m n* rule: enat2-cases) auto

lemma *eint-of-enat-sub*:

assumes $n \leq m$

shows $\text{eint-of-enat } (m - n) = \text{eint-of-enat } m - \text{eint-of-enat } n$

using *assms* **by** (cases *m n* rule: enat2-cases) auto

lemma *eint-of-enat-mult*:

$\text{eint-of-enat } (m * n) = \text{eint-of-enat } m * \text{eint-of-enat } n$

by (cases *m n* rule: enat2-cases) auto

lemmas *eint-of-enat-pushin* = *eint-of-enat-add eint-of-enat-sub eint-of-enat-mult*

lemmas *eint-of-enat-pushout* = *eint-of-enat-pushin*[symmetric]

lemma *eint-of-enat-nonneg*: $\text{eint-of-enat } n \geq 0$

by(cases *n*) simp-all

end

Chapter 2

Shallow embedding of Finite and Infinite ITL

2.1 Semantics

```
theory Semantics
imports NELList-Extras HOL-TLA.Intensional
begin
```

This theory mechanises a *shallow* embedding of Finite and Infinite ITL using the *NELList* and *Intensional* theories.

2.1.1 Types of Formulas

To mechanise the Finite and Infinite ITL semantics, the following type abbreviations are used:

```
type-synonym 'a intervals = 'a nellist

type-synonym ('a,'b) formfun    = 'a intervals  $\Rightarrow$  'b
type-synonym 'a formula        = ('a,bool) formfun
type-synonym ('a,'b) stfun     = 'a  $\Rightarrow$  'b
type-synonym 'a stpred         = ('a,bool) stfun
```

```
instance
  fun :: (type,type) world ..
```

```
instance
  prod :: (type,type) world ..
```

```
instance
  sum :: (type,type) world ..
```

```
instance
  nellist :: (type) world ..
```

Pair, function, sum, and interval are instantiated to be of type class *world*. This allows use of the lifted Intensional logic for formulas, and standard logical connectives can therefore be used.

2.1.2 Semantics of ITL

The semantics of ITL is defined.

definition *skip-d* :: ('a :: world) formula
where *skip-d* $\equiv (\lambda s. \text{nlength } s = (\text{enat } (\text{Suc } 0)))$

definition *chop-d* :: ('a :: world) formula \Rightarrow ('a :: world) formula \Rightarrow ('a :: world) formula
where *chop-d* *F1 F2* \equiv
 $(\lambda s.$
 $\quad (\exists n \leq \text{nlength } s. ((\text{ntaken } n \ s) \models F1) \wedge ((\text{ndropn } n \ s) \models F2))$
 $\quad \vee (\neg \text{nfinite } s \wedge (s \models F1))$
 $)$

definition *current-val-d* :: ('a :: world, 'b) stfun \Rightarrow ('a, 'b) formfun
where *current-val-d* *f* = $(\lambda s. ((\text{nfirst } s) \models f))$

definition *next-val-d* :: ('a :: world, 'b) stfun \Rightarrow ('a, 'b) formfun
where *next-val-d* *f* \equiv
 $(\lambda s. (\text{if } \text{nlength } s \neq (\text{enat } 0) \text{ then } ((\text{nnth } s \ 1) \models f) \text{ else } (\epsilon \ (x :: 'b) . x = x))$
 $)$

definition *fin-val-d* :: ('a :: world, 'b) stfun \Rightarrow ('a, 'b) formfun
where *fin-val-d* *f* $\equiv \lambda s. ((\text{if } \text{nfinite } s \text{ then } ((\text{nlast } s) \models f) \text{ else } (\epsilon \ (x :: 'b) . x = x)))$

definition *penult-val-d* :: ('a :: world, 'b) stfun \Rightarrow ('a, 'b) formfun
where *penult-val-d* *f* \equiv
 $(\lambda s.$
 $\quad (\text{if } \text{nfinite } s$
 $\quad \text{then } (\text{if } \text{nlength } s \neq (\text{enat } 0)$
 $\quad \quad \text{then } ((\text{nnth } s \ (\text{the-enat } (\text{epred}(\text{nlength } s)))) \models f)$
 $\quad \quad \text{else } (\epsilon \ (x :: 'b) . x = x))$
 $\quad \text{else } (\epsilon \ (x :: 'b) . x = x)$
 $)$
 $)$

syntax

-skip-d :: lift $((\text{skip}))$
-chop-d :: [lift, lift] \Rightarrow lift $((;-) [84, 84] \ 83)$
-current-val-d :: lift \Rightarrow lift $((\$-) [100] \ 99)$
-next-val-d :: lift \Rightarrow lift $((-\$) [100] \ 99)$
-fin-val-d :: lift \Rightarrow lift $((!-) [100] \ 99)$
-penult-val-d :: lift \Rightarrow lift $((-!) [100] \ 99)$
TEMP :: lift \Rightarrow 'b $((\text{TEMP } -))$

syntax (ASCII)

-skip-d :: lift $((\text{skip}))$
-chop-d :: [lift, lift] \Rightarrow lift $((;-) [84, 84] \ 83)$
-current-val-d :: lift \Rightarrow lift $((\$-) [100] \ 99)$
-next-val-d :: lift \Rightarrow lift $((-\$) [100] \ 99)$

$-fin\text{-}val\text{-}d \quad :: lift \Rightarrow lift \quad ((!-) [100] 99)$
 $-penult\text{-}val\text{-}d \quad :: lift \Rightarrow lift \quad ((-!) [100] 99)$

translations

$-skip\text{-}d \quad \Rightarrow CONST skip\text{-}d$
 $-chop\text{-}d \quad \Rightarrow CONST chop\text{-}d$
 $-current\text{-}val\text{-}d \Rightarrow CONST current\text{-}val\text{-}d$
 $-next\text{-}val\text{-}d \quad \Rightarrow CONST next\text{-}val\text{-}d$
 $-fin\text{-}val\text{-}d \quad \Rightarrow CONST fin\text{-}val\text{-}d$
 $-penult\text{-}val\text{-}d \Rightarrow CONST penult\text{-}val\text{-}d$
 $TEMP F \quad \rightarrow (F :: (- intervals) \Rightarrow -)$

2.1.3 Abbreviations

Some standard temporal abbreviations, with their concrete syntax.

definition $infinite\text{-}d :: ('a :: world) formula$
where $infinite\text{-}d \equiv LIFT(\#True; \#False)$

syntax

$-infinite\text{-}d \quad :: lift \quad (inf)$

syntax (ASCII)

$-infinite\text{-}d \quad :: lift \quad (inf)$

translations

$-infinite\text{-}d \Rightarrow CONST infinite\text{-}d$

definition $finite\text{-}d :: ('a :: world) formula$
where $finite\text{-}d \equiv LIFT(\neg(inf))$

syntax

$-finite\text{-}d \quad :: lift \quad (finite)$

syntax (ASCII)

$-finite\text{-}d \quad :: lift \quad (finite)$

translations

$-finite\text{-}d \Rightarrow CONST finite\text{-}d$

definition $schop\text{-}d :: ('a :: world) formula \Rightarrow 'a formula \Rightarrow 'a formula$
where $schop\text{-}d F1 F2 \equiv LIFT((F1 \wedge finite); F2)$

definition $sometimes\text{-}d :: ('a :: world) formula \Rightarrow 'a formula$
where $sometimes\text{-}d F \equiv LIFT(finite; F)$

definition $di\text{-}d :: ('a :: world) formula \Rightarrow 'a formula$
where $di\text{-}d F \equiv LIFT(F; \#True)$

definition $da\text{-}d :: ('a :: world) formula \Rightarrow 'a formula$

where $da-d\ F \equiv LIFT(finite;(F;\#True))$

definition $next-d :: ('a::world)\ formula \Rightarrow 'a\ formula$
where $next-d\ F \equiv LIFT(skip;F)$

definition $prev-d :: ('a::world)\ formula \Rightarrow 'a\ formula$
where $prev-d\ F \equiv LIFT(F;skip)$

syntax

$-schop-d \quad :: [lift, lift] \Rightarrow lift\ ((- \frown -)\ [84,84]\ 83)$
 $-sometimes-d \quad :: lift \Rightarrow lift\ ((\Diamond -)\ [88]\ 87)$
 $-di-d \quad :: lift \Rightarrow lift\ ((di\ -)\ [88]\ 87)$
 $-da-d \quad :: lift \Rightarrow lift\ ((da\ -)\ [88]\ 87)$
 $-next-d \quad :: lift \Rightarrow lift\ ((\bigcirc -)\ [88]\ 87)$
 $-prev-d \quad :: lift \Rightarrow lift\ ((prev\ -)\ [88]\ 87)$

syntax (ASCII)

$-schop-d \quad :: [lift, lift] \Rightarrow lift\ ((- \frown -)\ [84,84]\ 83)$
 $-sometimes-d \quad :: lift \Rightarrow lift\ ((\langle \rangle -)\ [88]\ 87)$
 $-di-d \quad :: lift \Rightarrow lift\ ((di\ -)\ [88]\ 87)$
 $-da-d \quad :: lift \Rightarrow lift\ ((da\ -)\ [88]\ 87)$
 $-next-d \quad :: lift \Rightarrow lift\ ((next\ -)\ [88]\ 87)$
 $-prev-d \quad :: lift \Rightarrow lift\ ((prev\ -)\ [88]\ 87)$

translations

$-schop-d \quad \Rightarrow CONST\ chop-d$
 $-sometimes-d \Rightarrow CONST\ sometimes-d$
 $-di-d \quad \Rightarrow CONST\ di-d$
 $-da-d \quad \Rightarrow CONST\ da-d$
 $-next-d \quad \Rightarrow CONST\ next-d$
 $-prev-d \quad \Rightarrow CONST\ prev-d$

definition $df-d :: ('a::world)\ formula \Rightarrow 'a\ formula$
where $df-d\ F \equiv LIFT(F \frown \#True)$

definition $sda-d :: ('a::world)\ formula \Rightarrow 'a\ formula$
where $sda-d\ F \equiv LIFT(\#True \frown (F \frown \#True))$

definition $always-d :: ('a::world)\ formula \Rightarrow 'a\ formula$
where $always-d\ F \equiv LIFT(\neg(\Diamond(\neg F)))$

definition $bi-d :: ('a::world)\ formula \Rightarrow 'a\ formula$
where $bi-d\ F \equiv LIFT(\neg(di(\neg F)))$

definition $ba-d :: ('a::world)\ formula \Rightarrow 'a\ formula$
where $ba-d\ F \equiv LIFT(\neg(da(\neg F)))$

definition $wnext-d :: ('a::world)\ formula \Rightarrow 'a\ formula$

where $wnext-d\ F \equiv LIFT(\neg(\bigcirc(\neg F)))$

definition $wprev-d :: ('a::world)\ formula \Rightarrow 'a\ formula$

where $wprev-d\ F \equiv LIFT(\neg(prev(\neg F)))$

definition $more-d :: ('a::world)\ formula$

where $more-d \equiv LIFT(\bigcirc(\#True))$

syntax

$-df-d \quad :: lift \Rightarrow lift\ ((df\ -)\ [88]\ 87)$
 $-sda-d \quad :: lift \Rightarrow lift\ ((sda\ -)\ [88]\ 87)$
 $-always-d \quad :: lift \Rightarrow lift\ ((\square\ -)\ [88]\ 87)$
 $-bi-d \quad :: lift \Rightarrow lift\ ((bi\ -)\ [88]\ 87)$
 $-ba-d \quad :: lift \Rightarrow lift\ ((ba\ -)\ [88]\ 87)$
 $-wnext-d \quad :: lift \Rightarrow lift\ ((wnext\ -)\ [88]\ 87)$
 $-wprev-d \quad :: lift \Rightarrow lift\ ((wprev\ -)\ [88]\ 87)$
 $-more-d \quad :: lift\ \quad\quad\quad ((more))$

syntax (ASCII)

$-df-d \quad :: lift \Rightarrow lift\ ((df\ -)\ [88]\ 87)$
 $-sda-d \quad :: lift \Rightarrow lift\ ((sda\ -)\ [88]\ 87)$
 $-always-d \quad :: lift \Rightarrow lift\ (([]\ -)\ [88]\ 87)$
 $-bi-d \quad :: lift \Rightarrow lift\ ((bi\ -)\ [88]\ 87)$
 $-ba-d \quad :: lift \Rightarrow lift\ ((ba\ -)\ [88]\ 87)$
 $-wnext-d \quad :: lift \Rightarrow lift\ ((wnext\ -)\ [88]\ 87)$
 $-wprev-d \quad :: lift \Rightarrow lift\ ((wprev\ -)\ [88]\ 87)$
 $-more-d \quad :: lift\ \quad\quad\quad ((more))$

translations

$-df-d \quad \Rightarrow CONST\ df-d$
 $-sda-d \quad \Rightarrow CONST\ sda-d$
 $-always-d \Rightarrow CONST\ always-d$
 $-bi-d \quad \Rightarrow CONST\ bi-d$
 $-ba-d \quad \Rightarrow CONST\ ba-d$
 $-wnext-d \Rightarrow CONST\ wnext-d$
 $-wprev-d \Rightarrow CONST\ wprev-d$
 $-more-d \Rightarrow CONST\ more-d$

definition $bf-d :: ('a::world)\ formula \Rightarrow 'a\ formula$

where $bf-d\ F \equiv LIFT(\neg(df(\neg F)))$

definition $sba-d :: ('a::world)\ formula \Rightarrow 'a\ formula$

where $sba-d\ F \equiv LIFT(\neg(sda(\neg F)))$

definition $empty-d :: ('a::world)\ formula$

where $empty-d \equiv LIFT(\neg(more))$

definition $fmore-d :: ('a::world) formula$
where $fmore-d \equiv LIFT(more \wedge finite)$

definition $dm-d :: ('a::world) formula \Rightarrow 'a formula$
where $dm-d F \equiv LIFT(\#True;(more \wedge F))$

syntax

$-bf-d \quad :: lift \Rightarrow lift ((bf -) [88] 87)$
 $-sba-d \quad :: lift \Rightarrow lift ((sba -) [88] 87)$
 $-empty-d \quad :: lift \quad ((empty))$
 $-fmore-d \quad :: lift \quad ((fmore))$
 $-dm-d \quad :: lift \Rightarrow lift ((dm -) [88] 87)$

syntax (ASCII)

$-bf-d \quad :: lift \Rightarrow lift ((bf -) [88] 87)$
 $-sba-d \quad :: lift \Rightarrow lift ((sba -) [88] 87)$
 $-empty-d \quad :: lift \quad ((empty))$
 $-fmore-d \quad :: lift \quad ((fmore))$
 $-dm-d \quad :: lift \Rightarrow lift ((dm -) [88] 87)$

translations

$-bf-d \quad \Rightarrow CONST bf-d$
 $-sba-d \quad \Rightarrow CONST sba-d$
 $-empty-d \Rightarrow CONST empty-d$
 $-fmore-d \Rightarrow CONST fmore-d$
 $-dm-d \quad \Rightarrow CONST dm-d$

definition $bm-d :: ('a::world) formula \Rightarrow 'a formula$
where $bm-d F \equiv LIFT(\neg(dm(\neg F)))$

definition $init-d :: ('a::world) formula \Rightarrow 'a formula$
where $init-d F \equiv LIFT((empty \wedge F);\#True)$

definition $fin-d :: ('a::world) formula \Rightarrow 'a formula$
where $fin-d F \equiv LIFT(\Box(empty \longrightarrow F))$

definition $halt-d :: ('a::world) formula \Rightarrow 'a formula$
where $halt-d F \equiv LIFT(\Box(empty = F))$

definition $initonly-d :: ('a::world) formula \Rightarrow 'a formula$
where $initonly-d F \equiv LIFT(bi(empty = F))$

definition $keep-d :: ('a::world) formula \Rightarrow 'a formula$
where $keep-d F \equiv LIFT(ba(skip \longrightarrow F))$

definition $yields-d :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a formula$
where $yields-d F1 F2 \equiv LIFT(\neg(F1;(\neg F2)))$

definition *syields-d* :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a formula
where *syields-d* F1 F2 \equiv LIFT($\neg(F1 \frown (\neg F2))$)

definition *ifthenelse-d* :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a formula \Rightarrow 'a formula
where *ifthenelse-d* F G H \equiv LIFT($((F \wedge G) \vee (\neg F \wedge H))$)

definition *revyields-d* :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a formula
where *revyields-d* F1 F2 \equiv LIFT($\neg((\neg F1); F2)$)

definition *revsyields-d* :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a formula
where *revsyields-d* F1 F2 \equiv LIFT($\neg((\neg F1) \frown F2)$)

syntax

-bm-d :: lift \Rightarrow lift ((bm -) [88] 87)
-init-d :: lift \Rightarrow lift ((init -) [88] 87)
-fin-d :: lift \Rightarrow lift ((fin -) [88] 87)
-halt-d :: lift \Rightarrow lift ((halt -) [88] 87)
-initonly-d :: lift \Rightarrow lift ((initonly -) [88] 87)
-keep-d :: lift \Rightarrow lift ((keep -) [88] 87)
-yields-d :: [lift, lift] \Rightarrow lift ((- | \sim > -) [88,88] 87)
-syields-d :: [lift, lift] \Rightarrow lift ((- | \simeq > -) [88,88] 87)
-ifthenelse-d :: [lift, lift, lift] \Rightarrow lift ((if_i - then - else -) [88,88,88] 87)
-revyields-d :: [lift, lift] \Rightarrow lift ((- < \sim | -) [84,84] 83)
-revsyields-d :: [lift, lift] \Rightarrow lift ((- < \simeq | -) [84,84] 83)

syntax (ASCII)

-bm-d :: lift \Rightarrow lift ((bm -) [88] 87)
-init-d :: lift \Rightarrow lift ((init -) [88] 87)
-fin-d :: lift \Rightarrow lift ((fin -) [88] 87)
-halt-d :: lift \Rightarrow lift ((halt -) [88] 87)
-initonly-d :: lift \Rightarrow lift ((initonly -) [88] 87)
-keep-d :: lift \Rightarrow lift ((keep -) [88] 87)
-yields-d :: [lift, lift] \Rightarrow lift ((- yields -) [88,88] 87)
-syields-d :: [lift, lift] \Rightarrow lift ((- syields -) [88,88] 87)
-ifthenelse-d :: [lift, lift, lift] \Rightarrow lift ((if_i - then - else -) [88,88,88] 87)
-revyields-d :: [lift, lift] \Rightarrow lift ((- revyields -) [84,84] 83)
-revsyields-d :: [lift, lift] \Rightarrow lift ((- revsyields -) [84,84] 83)

translations

-bm-d \Rightarrow CONST bm-d
-init-d \Rightarrow CONST init-d
-fin-d \Rightarrow CONST fin-d
-halt-d \Rightarrow CONST halt-d
-initonly-d \Rightarrow CONST initonly-d
-keep-d \Rightarrow CONST keep-d
-yields-d \Rightarrow CONST yields-d
-syields-d \Rightarrow CONST syields-d

$\text{-ifthenelse-d} \Rightarrow \text{CONST ifthenelse-d}$
 $\text{-revyields-d} \Rightarrow \text{CONST revyields-d}$
 $\text{-revsyields-d} \Rightarrow \text{CONST revsyields-d}$

definition $\text{sfm-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{sfm-d } F \equiv \text{LIFT}(\neg (\text{fm } (\neg F)))$

definition $\text{ifthen-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{ifthen-d } F \ G \equiv \text{LIFT}(\text{if}_i \ F \ \text{then } G \ \text{else } \# \text{True})$

syntax

$\text{-ifthen-d} :: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{if}_i - \text{then } -) \ [88,88] \ 87)$
 $\text{-sfm-d} \quad \quad \quad :: \text{lift} \Rightarrow \text{lift } ((\text{sfm } -) \ [88] \ 87)$

syntax (ASCII)

$\text{-ifthen-d} :: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{if}_i - \text{then } -) \ [88,88] \ 87)$
 $\text{-sfm-d} \quad \quad \quad :: \text{lift} \Rightarrow \text{lift } ((\text{sfm } -) \ [88] \ 87)$

translations

$\text{-ifthen-d} \Rightarrow \text{CONST ifthen-d}$
 $\text{-sfm-d} \Rightarrow \text{CONST sfm-d}$

definition $\text{afb-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{afb-d } F1 \ F2 \equiv \text{LIFT}(\text{bf}(F1 \longrightarrow \text{fm } F2))$

definition $\text{safb-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{safb-d } F1 \ F2 \equiv \text{LIFT}(\text{bf}(F1 = \text{fm } F2))$

syntax

$\text{-afb-d} \quad \quad \quad :: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((- \mapsto -) \ [84,84] \ 83)$
 $\text{-safb-d} \quad \quad \quad :: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((- \longleftrightarrow -) \ [84,84] \ 83)$

syntax (ASCII)

$\text{-afb-d} \quad \quad \quad :: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((- \text{ afb } -) \ [84,84] \ 83)$
 $\text{-safb-d} \quad \quad \quad :: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((- \text{ safb } -) \ [84,84] \ 83)$

translations

$\text{-afb-d} \Rightarrow \text{CONST afb-d}$
 $\text{-safb-d} \Rightarrow \text{CONST safb-d}$

definition $\text{next-assign-d} :: ('a::\text{world}, 'b) \text{ stfun} \Rightarrow ('a, 'b) \text{ formfun} \Rightarrow 'a \text{ formula}$
where $\text{next-assign-d } v \ e \equiv \text{LIFT}(v\$ = e)$

definition $\text{prev-assign-d} :: ('a::\text{world}, 'b) \text{ stfun} \Rightarrow ('a, 'b) \text{ formfun} \Rightarrow 'a \text{ formula}$
where $\text{prev-assign-d } v \ e \equiv \text{LIFT}(\text{finite} \longrightarrow v! = e)$

definition *always-eq-d* :: ('a::world,'b) stfun \Rightarrow ('a,'b) formfun \Rightarrow 'a formula
where *always-eq-d* v e $\equiv \lambda s. s \models \Box(\$v = e)$

definition *temporal-assign-d* :: ('a::world,'b) stfun \Rightarrow ('a,'b) formfun \Rightarrow 'a formula
where *temporal-assign-d* v e $\equiv \lambda s. s \models \text{finite} \longrightarrow !v = e$

definition *gets-d* :: ('a::world,'b) stfun \Rightarrow ('a,'b) formfun \Rightarrow 'a formula
where *gets-d* v e $\equiv \lambda s. s \models \text{keep}(\text{temporal-assign-d } v \ e)$

definition *stable-d* :: ('a::world,'b) stfun \Rightarrow 'a formula
where *stable-d* v $\equiv \lambda s. s \models \text{gets-d } v \ (\text{current-val-d } v)$

definition *padded-d* :: ('a::world,'b) stfun \Rightarrow 'a formula
where *padded-d* v $\equiv \lambda s. s \models (\text{stable-d } v); \text{skip} \vee \text{empty}$

definition *padded-temp-assign-d* :: ('a::world,'b) stfun \Rightarrow ('a,'b) formfun \Rightarrow 'a formula
where *padded-temp-assign-d* v e $\equiv \lambda s. s \models (\text{temporal-assign-d } v \ e) \wedge (\text{padded-d } v)$

syntax

-next-assign-d :: [lift, lift] \Rightarrow lift ((- := -) [50,51] 50)
-prev-assign-d :: [lift, lift] \Rightarrow lift ((- == -) [50,51] 50)
-always-eq-d :: [lift, lift] \Rightarrow lift ((- \approx -) [50,51] 50)
-temporal-assign-d :: [lift, lift] \Rightarrow lift ((- \leftarrow -) [50,51] 50)
-gets-d :: [lift, lift] \Rightarrow lift ((- gets -) [50,51] 50)
-stable-d :: lift \Rightarrow lift ((stable -) [51] 50)
-padded-d :: lift \Rightarrow lift ((padded -) [51] 50)
-padded-temp-assign-d :: [lift, lift] \Rightarrow lift ((- $<\sim$ -) [50,51] 50)

syntax (ASCII)

-next-assign-d :: [lift, lift] \Rightarrow lift ((- := -) [50,51] 50)
-prev-assign-d :: [lift, lift] \Rightarrow lift ((- == -) [50,51] 50)
-always-eq-d :: [lift, lift] \Rightarrow lift ((- alweqv -) [50,51] 50)
-temporal-assign-d :: [lift, lift] \Rightarrow lift ((- <-- -) [50,51] 50)
-gets-d :: [lift, lift] \Rightarrow lift ((- gets -) [50,51] 50)
-stable-d :: lift \Rightarrow lift ((stable -) [51] 50)
-padded-d :: lift \Rightarrow lift ((padded -) [51] 50)
-padded-temp-assign-d :: [lift, lift] \Rightarrow lift ((- <~ -) [50,51] 50)

translations

-next-assign-d \Rightarrow CONST next-assign-d
-prev-assign-d \Rightarrow CONST prev-assign-d
-always-eq-d \Rightarrow CONST always-eq-d
-temporal-assign-d \Rightarrow CONST temporal-assign-d
-gets-d \Rightarrow CONST gets-d
-stable-d \Rightarrow CONST stable-d
-padded-d \Rightarrow CONST padded-d
-padded-temp-assign-d \Rightarrow CONST padded-temp-assign-d

lemmas *itl-def* = *skip-d-def chop-d-def current-val-d-def next-val-d-def fin-val-d-def penult-val-d-def infinite-d-def finite-d-def chop-d-def sometimes-d-def di-d-def da-d-def next-d-def prev-d-def df-d-def sda-d-def always-d-def bi-d-def ba-d-def wnext-d-def wprev-d-def more-d-def bf-d-def sba-d-def empty-d-def fmore-d-def dm-d-def bm-d-def init-d-def fin-d-def halt-d-def initonly-d-def keep-d-def yields-d-def syields-d-def ifthenelse-d-def sfin-d-def ifthen-d-def next-assign-d-def prev-assign-d-def always-eq-d-def temporal-assign-d-def gets-d-def stable-d-def padded-d-def padded-temp-assign-d-def revyields-d-def revsyields-d-def afb-d-def safb-d-def*

2.1.4 Properties of Operators

The following lemmas show that above operators have the expected semantics.

lemma *skip-defs* :

$(w \models \text{skip}) = (\text{nlength } w = (\text{enat } 1))$

by (*simp add: itl-def*)

lemma *chop-defs* :

$(w \models F1 ; F2) =$

$($
 $(\exists n \leq \text{nlength } w. ((\text{ntaken } n w) \models F1) \wedge ((\text{ndropn } n w) \models F2))$
 $\vee (\neg \text{nfinite } w \wedge (w \models F1))$
 $)$

by (*simp add: itl-def*)

lemma *yields-defs* :

$(w \models F1 \text{ yields } F2) =$

$((\forall n. ((\text{ntaken } n w) \models F1) \longrightarrow \text{enat } n \leq \text{nlength } w \longrightarrow (\text{ndropn } n w \models F2)) \wedge (\text{nfinite } w \vee (w \models \neg F1)))$

by (*simp add: itl-def*)

lemma *revyields-defs* :

$(w \models F1 \text{ revyields } F2) =$

$((\forall n. ((\text{ndropn } n w) \models F2) \longrightarrow \text{enat } n \leq \text{nlength } w \longrightarrow ((\text{ntaken } n w) \models F1)) \wedge (\text{nfinite } w \vee (w \models F1)))$

by (*simp add: itl-def*)

blast

lemma *revsyields-defs* :

$(w \models F1 \text{ revsyields } F2) =$

$(\forall n. ((\text{ndropn } n w) \models F2) \longrightarrow \text{enat } n \leq \text{nlength } w \longrightarrow ((\text{ntaken } n w) \models F1)))$

by (*simp add: itl-def*)

blast

lemma *infinite-defs*:

$(w \models \text{inf}) = (\neg \text{nfinite } w)$

by (*simp add: itl-def*)

lemma *finite-defs* :

$(w \models \text{finite}) = (\text{nfinite } w)$

by (*simp add: itl-def*)

lemma *schop-defs* :

$(w \models F1 \frown F2) =$

(
 $(\exists n \leq \text{nlength } w. ((\text{ntaken } n \ w) \models F1) \wedge ((\text{ndropn } n \ w) \models F2))$
)

by (*auto simp add: itl-def chop-defs finite-defs*)

lemma *syields-defs* :

$(w \models F1 \text{ yields } F2) =$

$(\forall n. ((\text{ntaken } n \ w) \models F1) \longrightarrow \text{enat } n \leq \text{nlength } w \longrightarrow ((\text{ndropn } n \ w) \models F2))$

by (*simp add: itl-def*)

lemma *sometimes-defs* :

$(w \models \Diamond F) = (\exists n \leq \text{nlength } w. ((\text{ndropn } n \ w) \models F))$

by (*simp add: itl-def finite-defs chop-defs*)

lemma *always-defs* :

$(w \models \Box F) =$

$(\forall n \leq \text{nlength } w. ((\text{ndropn } n \ w) \models F))$

by (*simp add: itl-def sometimes-defs*)

lemma *di-defs* :

$(w \models \text{di } F) =$

$((\exists n \leq \text{nlength } w. ((\text{ntaken } n \ w) \models F)) \vee (\neg \text{nfinite } w \wedge (w \models F)))$

by (*simp add: itl-def*)

lemma *df-defs* :

$(w \models \text{df } F) =$

$(\exists n \leq \text{nlength } w. ((\text{ntaken } n \ w) \models F))$

by (*simp add: df-d-def schop-defs*)

lemma *bi-defs* :

$(w \models \text{bi } F) =$

$((\forall n \leq \text{nlength } w. ((\text{ntaken } n \ w) \models F)) \wedge (\text{nfinite } w \vee (w \models F)))$

by (*simp add: itl-def di-defs*)

lemma *bf-defs* :

$(w \models \text{bf } F) =$

$(\forall n \leq \text{nlength } w. ((\text{ntaken } n \ w) \models F))$

by (*simp add: bf-d-def df-defs*)

lemma *da-defs* :

$(w \models \text{da } F) =$

($(\exists n \text{ na. } (n + na \leq \text{nlength } w \wedge ((\text{nsubn } w \text{ n } (n + na)) \models F)) \vee (\neg \text{nfinite } w \wedge ((\text{ndropn } n \text{ w}) \models F)))$)

proof

(*auto simp add: itl-def chop-defs nsubn-def1*)

show $\bigwedge n \text{ na.}$

$\text{enat } n \leq \text{nlength } w \implies$

$\text{enat } na \leq \text{nlength } w - \text{enat } n \implies$

$F (\text{ntaken } na (\text{ndropn } n \text{ w})) \implies \exists n. (\exists na. \text{enat } (n + na) \leq \text{nlength } w \wedge F (\text{ntaken } na (\text{ndropn } n \text{ w})))$

$\vee \neg \text{nfinite } w \wedge F (\text{ndropn } n \text{ w})$

by (*metis add-left-mono enat.simps(3) enat-add-sub-same le-iff-add plus-enat-simps(1)*)

next

fix $n :: \text{nat}$ **and** $na :: \text{nat}$

assume $a1: \text{enat } (n + na) \leq \text{nlength } w$

assume $a2: F (\text{ntaken } na (\text{ndropn } n \text{ w}))$

have $\text{enat } n \leq \text{nlength } w$

using $a1$ **by** (*meson enat-ord-simps(1) le-iff-add order-subst2*)

then show $\exists n. \text{enat } n \leq \text{nlength } w \wedge$

$((\exists na. \text{enat } na \leq \text{nlength } w - \text{enat } n \wedge F (\text{ntaken } na (\text{ndropn } n \text{ w})))$

$\vee \neg \text{nfinite } w \wedge F (\text{ndropn } n \text{ w}))$

using $a2 \ a1$ **by** (*metis (no-types) add-diff-cancel-left' enat-minus-mono1 idiff-enat-enat*)

next

show $\bigwedge n. \neg \text{nfinite } w \implies$

$F (\text{ndropn } n \text{ w}) \implies$

$\exists n. \text{enat } n \leq \text{nlength } w \wedge$

$((\exists na. \text{enat } na \leq \text{nlength } w - \text{enat } n \wedge F (\text{ntaken } na (\text{ndropn } n \text{ w}))) \vee F (\text{ndropn } n \text{ w}))$

by (*meson enat-ile le-cases nfinite-conv-nlength-enat*)

qed

lemma *ba-defs* :

$(w \models \text{ba } F) =$

$(\forall n \text{ na. } (\text{enat } (n + na) \leq \text{nlength } w \longrightarrow ((\text{nsubn } w \text{ n } (n + na)) \models F))$
 $\wedge (\text{nfinite } w \vee ((\text{ndropn } n \text{ w}) \models F)))$

by (*simp add: ba-d-def da-defs*)

lemma *sda-defs* :

$(w \models \text{sda } F) =$

$(\exists n \text{ na. } (n + na \leq \text{nlength } w \wedge ((\text{nsubn } w \text{ n } (n + na)) \models F)))$

proof

(*auto simp add: sda-d-def schop-defs nsubn-def1*)

show $\bigwedge n \text{ na.}$

$\text{enat } n \leq \text{nlength } w \implies$

$\text{enat } na \leq \text{nlength } w - \text{enat } n \implies F (\text{ntaken } na (\text{ndropn } n \text{ w})) \implies$

$\exists n \text{ na. } \text{enat } (n + na) \leq \text{nlength } w \wedge F (\text{ntaken } na (\text{ndropn } n \text{ w}))$

by (*metis add-le-cancel-left enat-ord-simps(1) idiff-enat-enat le-add-diff-inverse le-cases nfinite-nlength-enat nfinite-ntaken ntaken-all*)

next

fix $n :: \text{nat}$ **and** $na :: \text{nat}$

assume $a1: F (\text{ntaken } na (\text{ndropn } n \text{ w}))$

assume $a2: \text{enat } (n + na) \leq \text{nlength } w$

have $enat\ na \leq nlength\ w - enat\ n$
by (*metis a2 add-diff-cancel-left' enat-minus-mono1 idiff-enat-enat*)
then show $\exists n. enat\ n \leq nlength\ w \wedge$
 $(\exists na. enat\ na \leq nlength\ w - enat\ n \wedge F\ (ntaken\ na\ (ndropn\ n\ w)))$
using *a2 a1*
by (*metis add.right-neutral le-cases le-zero-eq ndropn-all ndropn-nlength nlength-NNil*
the-enat.simps the-enat-0)
qed

lemma *sba-defs* :
 $(w \models sba\ F) =$
 $(\forall\ n\ na. n+na \leq nlength\ w \longrightarrow ((nsubn\ w\ n\ (n+na)) \models F))$
by (*simp add: sba-d-def sda-defs*)

lemma *next-defs* :
 $(w \models \bigcirc F) =$
 $(nlength\ w \neq (enat\ 0) \wedge ((ndropn\ 1\ w) \models F))$
by (*simp add: itl-def chop-defs*)
(metis One-nat-def Suc-ile-eq dual-order.order-iff-strict enat-le-plus-same(1) gen-nlength-def
min.orderE min-enat-simps(2) nlength-code nlength-eq-enat-nfiniteD one-enat-def
the-enat.simps zero-enat-def zero-one-enat-neq(1))

lemma *wnext-defs* :
 $(w \models wnext\ F) =$
 $(nlength\ w = (enat\ 0) \vee ((ndropn\ 1\ w) \models F))$
by (*simp add: wnext-d-def next-defs*)

lemma *prev-defs* :
 $(w \models prev\ F) =$
 $((nlength\ w \neq (enat\ 0) \wedge nfinite\ w \wedge$
 $((ntaken\ (the-enat((epred(nlength\ w))))\ w) \models F)) \vee (\neg nfinite\ w \wedge (w \models F)))$

proof (*cases nfinite w*)

case *True*

then show *?thesis*

by (*auto simp add: prev-d-def chop-defs skip-defs*)
(metis One-nat-def diff-diff-cancel enat-ord-simps(1) epred-enat idiff-enat-enat nfinite-nlength-enat
the-enat.simps,
metis One-nat-def diff-le-self eSuc-epred enat.simps(3) enat-add-sub-same enat-ord-simps(1)
epred-enat nfinite-nlength-enat one-enat-def plus-1-eSuc(2) the-enat.simps zero-enat-def)

next

case *False*

then show *?thesis*

by (*auto simp add: prev-d-def chop-defs skip-defs*)
(metis ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD)

qed

lemma *wprev-defs* :
 $(w \models wprev\ F) =$
 $((nfinite\ w \longrightarrow (nlength\ w = (enat\ 0) \vee ((ntaken\ (the-enat(epred(nlength\ w))))\ w) \models F)) \wedge$
 $(nfinite\ w \vee (w \models F)))$

by (*simp add: wprev-d-def prev-defs*)

lemma *more-defs* :

($w \models \text{more}$) = ($0 < \text{nlength } w$)

using *zero-enat-def* **by** (*auto simp add: more-d-def next-defs*)

lemma *fmore-defs* :

($w \models \text{fmore}$) = ($\text{nfinite } w \wedge (0 < \text{nlength } w)$)

by (*auto simp add: fmore-d-def more-defs finite-defs*)

lemma *empty-defs* :

($w \models \text{empty}$) = ($\text{nlength } w = 0$)

by (*simp add: empty-d-def more-defs*)

lemma *init-defs* :

($w \models \text{init } F$) = ($(\text{ntaken } 0 \ w) \models F$)

by (*simp add: init-d-def chop-defs empty-defs min-def*)

(*metis ntaken-0 ntaken-all the-enat.simps zero-enat-def zero-le*)

lemma *initalt-defs* :

($w \models \text{bi}(\text{empty} \longrightarrow F)$) = ($(\text{ntaken } 0 \ w) \models F$)

by (*simp add: bi-defs empty-defs min-def*)

(*metis enat-0-iff(2) nlength-eq-enat-nfiniteD ntaken-0 zero-le*)

lemma *fin-defs* :

($w \models \text{fin } F$) =

($(\text{nfinite } w \wedge ((\text{ndropn } (\text{the-enat}(\text{nlength } w)) \ w) \models F)) \vee (\neg \text{nfinite } w)$)

by (*simp add: fin-d-def empty-defs always-defs*)

(*metis add.right-neutral enat.distinct(2) enat-add-sub-same le-iff-add nfinite-nlength-enat nlength-eq-enat-nfiniteD the-enat.simps*)

lemma *finalt-defs* :

($w \models \# \text{True}; (F \wedge \text{empty})$) =

($(\text{nfinite } w \wedge ((\text{ndropn } (\text{the-enat}(\text{nlength } w)) \ w) \models F)) \vee (\neg \text{nfinite } w)$)

by (*simp add: chop-defs empty-defs*)

(*metis add.right-neutral enat.distinct(2) enat-add-sub-same le-iff-add nfinite-nlength-enat the-enat.simps*)

lemma *sfin-defs* :

($w \models \text{sfin } F$) = ($\text{nfinite } w \wedge ((\text{ndropn } (\text{the-enat}(\text{nlength } w)) \ w) \models F)$)

by (*auto simp add: sfin-d-def fin-defs*)

lemma *afb-defs*:

($w \models \text{f1 } \text{afb } \text{f2}$) =

($\forall n. ((\text{ntaken } n \ w) \models \text{f1}) \wedge \text{enat } n \leq \text{nlength } w \longrightarrow ((\text{NNil } (\text{nnth } w \ n)) \models \text{f2}))$)

unfolding *afb-d-def*

using *bf-defs[of LIFT(f1 \longrightarrow fin f2)] fin-defs[of f2]*

by *simp*

(*metis min.absorb1 ndropn-all ntaken-nlast ntaken-nlength order-refl*)

lemma *safb-defs*:

$(w \models f1 \text{ safb } f2) =$

$(\forall n. \text{ enat } n \leq \text{ nlength } w \longrightarrow ((\text{ntaken } n \ w) \models f1) = ((\text{NNil } (\text{nnth } w \ n)) \models f2))$

unfolding *safb-d-def*

using *bf-defs*[of *LIFT*($f1 = \text{fin } f2$)] *fin-defs*[of *f2*]

by (*simp add: ndropn-all ntaken-nlast*)

lemma *halt-defs* :

$(w \models \text{halt}(F)) = (\forall n. \text{ n} \leq \text{ nlength } w \longrightarrow (\text{ nlength } w = n) = (\text{ (ndropn } n \ w) \models F))$

by (*simp add: halt-d-def empty-defs always-defs*)

(*metis add.right-neutral dual-order.strict-iff-order enat-add-sub-same enat-ord-code*(4)
le-iff-add)

lemma *initonly-defs* :

$(w \models \text{initonly}(F)) =$

$((\forall n. \text{ n} \leq \text{ nlength } w \longrightarrow (n = 0) = ((\text{ntaken } n \ w) \models F)) \wedge$
 $(\text{ nfinite } w \vee (\text{ nlength } w = 0) = F \ w)$
 $)$

by (*simp add: initonly-d-def bi-defs empty-defs zero-enat-def*)

lemma *ifthenelse-defs*:

$(w \models \text{if}_i \ F \ \text{then } G \ \text{else } H) =$

$(((w \models F) \wedge (w \models G)) \vee ((\neg(w \models F) \wedge (w \models H))))$

by (*simp add: itl-def*)

lemma *currentval-defs* :

$(s \models \$v) = (v \ (\text{nfirst } s))$

by (*simp add: itl-def*)

lemma *nextval-defs* :

$(s \models v\$) =$

$(\text{if } \text{ nlength } s \neq (\text{ enat } 0) \ \text{then } (v \ (\text{nnth } s \ 1)) \ \text{else } (\epsilon \ x. x=x))$

by (*simp add: itl-def*)

lemma *finval-defs* :

$(s \models !v) =$

$((\text{if } \text{ nfinite } s \ \text{then } (v \ (\text{nlast } s)) \ \text{else } (\epsilon \ x. x=x))$
 $)$

by (*simp add: itl-def*)

lemma *penultval-defs* :

$(s \models v!) =$

$(\text{if } \text{ nfinite } s \ \text{then}$

$(\text{if } \text{ nlength } s \neq (\text{ enat } 0) \ \text{then } (v \ (\text{nnth } s \ (\text{the-enat}(\text{epred}(\text{ nlength } s)))))) \ \text{else } (\epsilon \ x. x=x))$
 $\text{else } (\epsilon \ x. x=x)$

)
by (*simp add: itl-def*)

lemma *next-assign-defs* :

($s \models v := e$) = ((if $nlength\ s \neq (enat\ 0)$ then $v\ (nnth\ s\ 1)$ else $(\epsilon\ x.\ x=x)$) = $e\ s$)
by (*auto simp: itl-def*)

lemma *prev-assign-defs* :

($s \models v =: e$) =
 (if $nfinite\ s$ then
 (if $nlength\ s \neq (enat\ 0)$ then $(v\ (nnth\ s\ (the-enat(epred(nlength\ s)))) = e\ s$
 else $(\epsilon\ x.\ x=x) = e\ s$)
 else *True*)
)
by (*simp add: itl-def finite-defs*)

lemma *always-equiv-defs* :

($s \models v \approx e$) =
 ($(\forall\ i.\ i \leq nlength\ s \longrightarrow v\ (nnth\ s\ i) = e\ (ndropn\ i\ s))$)
)
by (*simp add: always-eq-d-def always-defs current-val-d-def ndropn-nfirst*)

lemma *temporal-assign-defs* :

($s \models v \leftarrow e$) =
 (if $nfinite\ s$ then $(v\ (nlast\ s)) = e\ s$
 else *True*)
)
by (*simp add: itl-def finite-defs*)

lemma *gets-defs* :

($s \models v\ gets\ e$) =
 ($(\forall\ i.\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = e\ ((nsubn\ s\ i\ (Suc\ i)))$))
)

proof (*auto simp add: min-def finite-defs gets-d-def keep-d-def ba-defs skip-defs*

temporal-assign-d-def fin-val-d-def nsubn-nlength Suc-ile-eq nsubn-def1 ntaken-ndropn-nlast)

show $\bigwedge i.\ \forall n\ na.\ (enat\ na \leq nlength\ s - enat\ n \longrightarrow$

$enat\ (n + na) \leq nlength\ s \longrightarrow$

$na = Suc\ 0 \longrightarrow v\ (nnth\ s\ (Suc\ n)) = e\ (ntaken\ (Suc\ 0)\ (ndropn\ n\ s))) \wedge$

$(\neg enat\ na \leq nlength\ s - enat\ n \longrightarrow$

$enat\ (n + na) \leq nlength\ s \longrightarrow$

$nlength\ s - enat\ n = enat\ (Suc\ 0) \longrightarrow$

$v\ (nnth\ s\ (na + n)) = e\ (ntaken\ na\ (ndropn\ n\ s))) \implies$

$enat\ i < nlength\ s \implies$

$v\ (nnth\ s\ (Suc\ i)) = e\ (ntaken\ (Suc\ 0)\ (ndropn\ i\ s))$

by (*metis One-nat-def add.commute eSuc-enat i0-less ileI1 ileSS-Suc-eq ndropn-Suc-conv-ndropn*
ndropn-nlength nlength-NCons one-eSuc one-enat-def plus-1-eq-Suc zero-le)

show $\bigwedge n\ na.$

$\forall i.\ enat\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = e\ (ntaken\ (Suc\ 0)\ (ndropn\ i\ s)) \implies$

$\neg na \leq Suc\ 0 \implies enat\ (n + na) \leq nlength\ s \implies nlength\ s - enat\ n = enat\ (Suc\ 0) \implies$

$v (nnth\ s\ (na + n)) = e\ (ntaken\ na\ (ndropn\ n\ s))$
by (metis (no-types) add-diff-cancel-left' enat-minus-mono1 enat-ord-simps(1) idiff-enat-enat)
qed

lemma *stable-defs-helpa*:

assumes $(\forall i. i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nnth\ s\ i))$
 $i \leq nlength\ s$
shows $(v\ (nnth\ s\ i) = v\ (nfirst\ s))$
using *assms*
proof (induct i arbitrary:s)
case 0
then show ?case **by** (metis ndropn-0 ndropn-nfirst)
next
case (Suc i)
then show ?case
by (simp add: Suc-ile-eq)
qed

lemma *stable-defs-helpb*:

assumes $(\forall i. i \leq nlength\ s \longrightarrow v\ (nnth\ s\ i) = v\ (nfirst\ s))$
 $i < nlength\ s$
shows $v\ (nnth\ s\ (Suc\ i)) = v\ (nnth\ s\ i)$
using *assms*
proof (induct i arbitrary:s)
case 0
then show ?case **using** Suc-ile-eq **by** auto
next
case (Suc i)
then show ?case **by** (metis eSuc-enat ileI1 less-imp-le)
qed

lemma *stable-defs-help*:

$(\forall i. i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nnth\ s\ i)) =$
 $(\forall i. i \leq nlength\ s \longrightarrow v\ (nnth\ s\ i) = v\ (nfirst\ s))$
using *stable-defs-helpa*[of s v] *stable-defs-helpb*[of s v]
by blast

lemma *stable-defs*:

$(s \models stable\ v) =$
 $(\forall i. i \leq nlength\ s \longrightarrow (v\ (nnth\ s\ i)) = (v\ (nfirst\ s)))$
proof (simp add: stable-d-def gets-defs current-val-d-def)
have 1: $\bigwedge i. i < nlength\ s \longrightarrow v\ (nfirst\ (nsubn\ s\ i\ (Suc\ i))) = v\ (nnth\ s\ i)$
by (simp add: nsubn-def1 ntaken-ndropn-nfirst)
have 2: $(\forall i. enat\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nfirst\ (nsubn\ s\ i\ (Suc\ i)))) =$
 $(\forall i. enat\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nnth\ s\ i))$
by (simp add: 1)
have 3: $(\forall i. enat\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nnth\ s\ i)) =$
 $(\forall i. i \leq nlength\ s \longrightarrow (v\ (nnth\ s\ i)) = (v\ (nfirst\ s)))$
using *stable-defs-help*[of s v] **by** blast
show $(\forall i. enat\ i < nlength\ s \longrightarrow v\ (nnth\ s\ (Suc\ i)) = v\ (nfirst\ (nsubn\ s\ i\ (Suc\ i)))) =$

$(\forall i. \text{enat } i \leq \text{nlength } s \longrightarrow v (\text{nnth } s \ i) = v (\text{nfirst } s))$
 by (simp add: 2 3)
 qed

lemma *padded-defs* :

$(s \models \text{padded } v) =$
 $((\forall i. i < \text{nlength } s \longrightarrow (v (\text{nnth } s \ i)) = (v (\text{nfirst } s)))) \vee \text{nlength } s = (\text{enat } 0)$

proof (cases s)

case (NNil x1)

then show ?thesis

by (auto simp add: padded-d-def stable-defs chop-d-def skip-defs empty-defs ntaken-nfirst
 ntaken-nnth zero-enat-def)

next

case (NCons x21 x22)

then show ?thesis

by (auto simp add: padded-d-def stable-defs chop-d-def skip-defs empty-defs ntaken-nfirst
 ntaken-nnth zero-enat-def)

(metis One-nat-def enat.simps(3) enat-add-sub-same enat-ord-simps(1) iless-Suc-eq le-iff-add
 less-imp-le one-enat-def plus-1-eSuc(2),
 meson iless-Suc-eq less-imp-le,
 metis One-nat-def enat.simps(3) enat-add-sub-same enat-ord-simps(1) ile-eSuc nfinite-nlength-enat
 one-enat-def plus-1-eSuc(2),
 metis One-nat-def antisym-conv2 co.enat.sel(2) diff-le-self enat-add-sub-same enat-ord-code(4)
 enat-ord-simps(1) epred-enat iless-Suc-eq one-enat-def plus-1-eSuc(2))

qed

lemma *padded-temporal-assign-defs* :

$(s \models v < \sim e) =$
 $((s \models \text{padded } v) \wedge$
 $(\text{if } n\text{finite } s \text{ then } (v (\text{nlast } s)) = e \ s \text{ else } \text{True}))$
 $)$

by (auto simp add: padded-temp-assign-d-def padded-defs temporal-assign-defs)

lemma *chop-nfuse-1* :

$(\exists \sigma 1 \sigma 2. \sigma = \text{nfuse } \sigma 1 \sigma 2 \wedge n\text{finite } \sigma 1 \wedge$
 $(\sigma 1 \models f) \wedge (\sigma 2 \models g) \wedge$
 $(\text{nlast } \sigma 1 = \text{nfirst } \sigma 2)) =$
 $((\exists i. 0 \leq i \wedge i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \sigma \models f) \wedge (\text{ndropn } i \sigma \models g)))$

by auto

(metis enat-le-plus-same(1) nfuse-nlength ndropn-nfuse nfinite-conv-nlength-enat ntaken-nfuse
 the-enat.simps,
 metis nfuse-ntaken-ndropn ndropn-nfirst nfinite-ntaken ntaken-nlast)

lemma *chop-nfuse-2* :

$(\exists \sigma 1 \sigma 2. \sigma = \text{nfuse } \sigma 1 \sigma 2 \wedge n\text{finite } \sigma 1 \wedge$
 $(\sigma 1 \in X) \wedge (\sigma 2 \in Y) \wedge$
 $(\text{nlast } \sigma 1 = \text{nfirst } \sigma 2)) =$
 $(\exists i. i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \sigma) \in X \wedge (\text{ndropn } i \sigma) \in Y)$

by auto

(metis enat-le-plus-same(1) ndropn-nfuse nfinite-nlength-enat nfuse-nlength ntaken-nfuse)

the-enat.simps,
metis ndropn-nfirst nfinite-ntaken nfuse-ntaken-ndropn ntaken-nlast)

lemma *chop-nfuse:*

$(\sigma \models f;g) = ($
 $(\exists \sigma 1 \sigma 2. \sigma = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge$
 $(\sigma 1 \models f) \wedge (\sigma 2 \models g) \wedge (\text{nlast } \sigma 1 = \text{nfirst } \sigma 2))$
 $\vee (\neg \text{nfinite } \sigma \wedge (\sigma \models f))$
 $)$

by (*simp add: chop-defs chop-nfuse-1*)

lemmas *itl-defs = skip-defs chop-defs yields-defs infinite-defs finite-defs chop-defs syields-defs*
sometimes-defs always-defs di-defs df-defs bi-defs bf-defs da-defs ba-defs sda-defs sba-defs
next-defs wnext-defs prev-defs wprev-defs more-defs fmore-defs empty-defs init-defs
initalt-defs fin-defs finalt-defs sfin-defs halt-defs initonly-defs ifthenelse-defs
currentval-defs nextval-defs finval-defs penultval-defs next-assign-defs prev-assign-defs
always-eqv-defs temporal-assign-defs gets-defs stable-defs padded-defs
padded-temporal-assign-defs revyields-defs revsyields-defs afb-defs safb-defs

2.1.5 Soundness Axioms

ChopAssoc

lemma *ChopAssocSemHelpa:*

assumes $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $f (\text{ntaken } n \sigma) \wedge$
 $((\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n \sigma)))$
 \vee
 $\neg \text{nfinite } (\text{ndropn } n \sigma) \wedge g (\text{ndropn } n \sigma))) \vee$
 $\neg \text{nfinite } \sigma \wedge f \sigma)$

shows $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge (\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge$
 $f (\text{ntaken } (\min na n) \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \sigma))) \wedge h (\text{ndropn } n \sigma)) \vee$
 $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ntaken } n \sigma) \wedge g (\text{ndropn } n \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \sigma))$

proof –

have 1: $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $f (\text{ntaken } n \sigma) \wedge$
 $((\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n \sigma)))$
 \vee
 $\neg \text{nfinite } (\text{ndropn } n \sigma) \wedge g (\text{ndropn } n \sigma))) \vee$
 $\neg \text{nfinite } \sigma \wedge f \sigma)$

using *assms by auto*

have 2: $\neg \text{nfinite } \sigma \wedge f \sigma \implies$
 $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\min na n) \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \sigma)))$
 $\wedge h (\text{ndropn } n \sigma)) \vee$
 $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ntaken } n \sigma) \wedge g (\text{ndropn } n \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \sigma))$

by *simp*

have 3: $(\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $f (\text{ntaken } n \sigma) \wedge$
 $((\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n \sigma)))$

\vee
 $\neg \text{nfinite } (\text{ndropn } n \ \sigma) \wedge g (\text{ndropn } n \ \sigma)) \implies$
 $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \ \sigma)))$
 $\wedge h (\text{ndropn } n \ \sigma)) \vee$
 $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ntaken } n \ \sigma) \wedge g (\text{ndropn } n \ \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \ \sigma))$
proof –
assume 4: $(\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $f (\text{ntaken } n \ \sigma) \wedge$
 $((\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \ \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n \ \sigma))))$
 \vee
 $\neg \text{nfinite } (\text{ndropn } n \ \sigma) \wedge g (\text{ndropn } n \ \sigma))$
obtain n **where** 5: $\text{enat } n \leq \text{nlength } \sigma \wedge$
 $f (\text{ntaken } n \ \sigma) \wedge$
 $((\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \ \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n \ \sigma))))$
 \vee
 $\neg \text{nfinite } (\text{ndropn } n \ \sigma) \wedge g (\text{ndropn } n \ \sigma))$
using 4 **by** *auto*
have 6: $\text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ntaken } n \ \sigma)$
using 5 **by** *auto*
have 7: $\neg \text{nfinite } (\text{ndropn } n \ \sigma) \wedge g (\text{ndropn } n \ \sigma) \implies$
 $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \ \sigma)))$
 $\wedge h (\text{ndropn } n \ \sigma)) \vee$
 $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ntaken } n \ \sigma) \wedge g (\text{ndropn } n \ \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \ \sigma))$
using 6 *nfinite-ndropn-a* **by** *blast*
have 8: $(\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \ \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n \ \sigma))) \implies$
 $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \ \sigma)))$
 $\wedge h (\text{ndropn } n \ \sigma)) \vee$
 $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ntaken } n \ \sigma) \wedge g (\text{ndropn } n \ \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \ \sigma))$
proof –
assume 9: $(\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \ \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n \ \sigma)))$
show $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \ \sigma)))$
 $\wedge h (\text{ndropn } n \ \sigma)) \vee$
 $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ntaken } n \ \sigma) \wedge g (\text{ndropn } n \ \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \ \sigma))$
proof –
obtain na **where** 10: $\text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \ \sigma)) \wedge$
 $h (\text{ndropn } na (\text{ndropn } n \ \sigma))$
using 9 **by** *auto*
have 11: $h (\text{ndropn } (na+n) \ \sigma)$
by (*metis* 10 *add commute ndropn-ndropn*)
have 12: $na+n \leq \text{nlength } \sigma$
by (*metis* 10 6 *Groups.add-ac(2) dual-order.strict-implies-order enat.simps(3)*
 $\text{enat-add-sub-same enat-less-enat-plusI2 le-iff-add not-le-imp-less}$
 $\text{order.not-eq-order-implies-strict plus-enat-simps(1)})$
have 13: $g (\text{ndropn } n (\text{ntaken } (n+na) \ \sigma))$

```

    by (metis 10 12 add.commute ntaken-ndropn-swap plus-enat-simps(1))
  have 14: f (ntaken (min n (n+na)) σ)
    using 6 by linarith
  show ?thesis
    by (metis 10 12 13 14 6 add.commute le-add1 ndropn-ndropn)
qed
qed
show ?thesis
using 5 7 8 by blast
qed
show ?thesis
using 3 assms by blast
qed

lemma ChopAssocSemHelpb:
assumes ((∃ n. enat n ≤ nlength σ ∧
  (∃ na ≤ n. enat na ≤ nlength σ ∧ f (ntaken (min na n) σ) ∧ g (ndropn na (ntaken n σ)))
  ∧ h (ndropn n σ)) ∨
  ¬ nfinite σ ∧ ((∃ n. enat n ≤ nlength σ ∧ f (ntaken n σ) ∧ g (ndropn n σ)) ∨ ¬ nfinite σ ∧ f σ))
shows ((∃ n. enat n ≤ nlength σ ∧
  f (ntaken n σ) ∧
  ((∃ na. enat na ≤ nlength σ - enat n ∧ g (ntaken na (ndropn n σ)) ∧ h (ndropn na (ndropn n σ)))
  ∨
  ¬ nfinite (ndropn n σ) ∧ g (ndropn n σ))) ∨
  ¬ nfinite σ ∧ f σ)
proof -
  have 1: ((∃ n. enat n ≤ nlength σ ∧
    (∃ na ≤ n. enat na ≤ nlength σ ∧ f (ntaken (min na n) σ) ∧ g (ndropn na (ntaken n σ)))
    ∧ h (ndropn n σ)) ∨
    ¬ nfinite σ ∧ ((∃ n. enat n ≤ nlength σ ∧ f (ntaken n σ) ∧ g (ndropn n σ)) ∨ ¬ nfinite σ ∧ f σ))
    using assms by auto
  have 2: ¬ nfinite σ ∧ ((∃ n. enat n ≤ nlength σ ∧ f (ntaken n σ) ∧ g (ndropn n σ)) ∨ ¬ nfinite σ ∧ f σ)
  ⇒
    ((∃ n. enat n ≤ nlength σ ∧
      f (ntaken n σ) ∧
      ((∃ na. enat na ≤ nlength σ - enat n ∧ g (ntaken na (ndropn n σ)) ∧ h (ndropn na (ndropn n σ)))
      ∨
      ¬ nfinite (ndropn n σ) ∧ g (ndropn n σ))) ∨
      ¬ nfinite σ ∧ f σ)
  using nfinite-ndropn by blast
  have 3: (∃ n. enat n ≤ nlength σ ∧
    (∃ na ≤ n. enat na ≤ nlength σ ∧ f (ntaken (min na n) σ) ∧ g (ndropn na (ntaken n σ)))
    ∧ h (ndropn n σ)) ⇒
    ((∃ n. enat n ≤ nlength σ ∧
      f (ntaken n σ) ∧
      ((∃ na. enat na ≤ nlength σ - enat n ∧ g (ntaken na (ndropn n σ)) ∧ h (ndropn na (ndropn n σ)))
      ∨
      ¬ nfinite (ndropn n σ) ∧ g (ndropn n σ))) ∨
      ¬ nfinite σ ∧ f σ)
  proof -

```

assume 4: $(\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \ \sigma)))$
 $\wedge h (\text{ndropn } n \ \sigma))$
show $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $f (\text{ntaken } n \ \sigma) \wedge$
 $((\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \ \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n \ \sigma)))$
 \vee
 $\neg \text{nfinite } (\text{ndropn } n \ \sigma) \wedge g (\text{ndropn } n \ \sigma))) \vee$
 $\neg \text{nfinite } \sigma \wedge f \ \sigma)$
proof –
obtain n **where** 5: $\text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \ \sigma)))$
 $\wedge h (\text{ndropn } n \ \sigma)$
using 4 **by** *auto*
have 6: $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \ \sigma)))$
using 5 **by** *auto*
obtain na **where** 7: $na \leq n \wedge \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \ \sigma))$
using 6 **by** *auto*
have 8: $na \leq \text{nlength } \sigma$
by (*simp add: 7*)
have 9: $n - na \leq \text{nlength } \sigma - na$
by (*metis 5 enat-minus-mono1 idiff-enat-enat*)
have 10: $f (\text{ntaken } na \ \sigma)$
using 7 **by** *linarith*
have 11: $g (\text{ntaken } (n - na) (\text{ndropn } na \ \sigma))$
by (*simp add: 5 7 ntaken-ndropn-swap*)
have 12: $h (\text{ndropn } ((n - na) + na) \ \sigma)$
by (*simp add: 5 7*)
have 13: $h (\text{ndropn } (n - na) (\text{ndropn } na \ \sigma))$
by (*metis 12 add commute ndropn-ndropn*)
show ?thesis
using 10 11 13 7 9 **by** *auto*
qed
qed
show ?thesis
using 2 3 *assms* **by** *blast*
qed

lemma *ChopAssocSemHelp*:

$((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $f (\text{ntaken } n \ \sigma) \wedge$
 $((\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \ \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n \ \sigma)))$
 \vee
 $\neg \text{nfinite } (\text{ndropn } n \ \sigma) \wedge g (\text{ndropn } n \ \sigma))) \vee$
 $\neg \text{nfinite } \sigma \wedge f \ \sigma) =$
 $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \ \sigma)))$
 $\wedge h (\text{ndropn } n \ \sigma)) \vee$
 $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ntaken } n \ \sigma) \wedge g (\text{ndropn } n \ \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \ \sigma))$

using *ChopAssocSemHelpa*[*of* σ f g h] *ChopAssocSemHelpb*[*of* σ f g h] **by** *blast*

lemma *ChopAssocSemHelp1*:

$((\sigma) \models f ; (g ; h)) = ((\sigma) \models (f;g);h)$

proof –

have $(\sigma \models f ; (g ; h)) = ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $f (\text{ntaken } n \sigma) \wedge$
 $((\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n \sigma))))$

\vee

$\neg \text{nfinite } (\text{ndropn } n \sigma) \wedge g (\text{ndropn } n \sigma))) \vee$

$\neg \text{nfinite } \sigma \wedge f \sigma)$

by (*simp add: chop-defs*)

also have ... =

$((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \sigma)))$
 $\wedge h (\text{ndropn } n \sigma)) \vee$

$\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ntaken } n \sigma) \wedge g (\text{ndropn } n \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \sigma))$

using *ChopAssocSemHelp*[*of* σ f g h] **by** *blast*

also have ... =

$(\sigma \models (f;g);h)$ **by** (*simp add: chop-defs*)

finally show $(\sigma \models f ; (g ; h)) = (\sigma \models (f;g);h)$.

qed

lemma *ChopAssocSem*:

$(\sigma \models f ; (g ; h)) = (f;g);h)$

using *ChopAssocSemHelp1*[*of* f g h σ] **by** *auto*

OrChopImp

lemma *OrChopImpSem*:

$(\sigma \models (f \vee g);h \longrightarrow f;h \vee g;h)$

by (*auto simp add: chop-defs*)

ChopOrImp

lemma *ChopOrImpSem*:

$(\sigma \models f;(g \vee h) \longrightarrow f;g \vee f;h)$

by (*auto simp add: chop-defs*)

EmptyChop

lemma *EmptyChopSem*:

$(\sigma \models \text{empty} ; f = f)$

by (*simp add: chop-defs empty-defs min-def*)

$(\text{metis enat-0-iff}(1) \text{ndropn-0 nlength-eq-enat-nfiniteD zero-le})$

ChopEmpty

lemma *ChopEmptySem*:

$(\sigma \models f;\text{empty} = f)$

by (*simp add: chop-defs empty-defs min-def*)

(metis cancel-comm-monoid-add-class.diff-cancel enat-diff-cancel-left idiff-enat-enat
nfinite-nlength-enat ntaken-all order-refl zero-enat-def)

StateImpBi

lemma *StateImpBiSem:*

($\sigma \models \text{init } f \longrightarrow \text{bi } (\text{init } f)$)
by (simp add: init-defs bi-defs)

NextImpNotNextNot

lemma *NextImpNotNextNotSem:*

($\sigma \models \bigcirc f \longrightarrow \neg (\bigcirc (\neg f))$)
by (simp add: next-defs)

BiBoxChopImpChop

lemma *BiBoxChopImpChopSem:*

($\sigma \models \text{bi } (f \longrightarrow f1) \wedge \Box (g \longrightarrow g1) \longrightarrow f;g \longrightarrow f1;g1$)
by (simp add: bi-defs always-defs chop-defs)
fastforce

BoxInduct

lemma *box-induct-help-1 :*

$\bigwedge j. \forall n. \text{enat } n \leq \text{nlength } \sigma \longrightarrow f (\text{ndropn } n \sigma) \longrightarrow$
 $\text{nlength } \sigma - \text{enat } n = (\text{enat } 0) \vee f (\text{ndropn } (\text{Suc } 0) (\text{ndropn } n \sigma)) \implies$
 $f \sigma \implies$
 $\text{enat } j \leq \text{nlength } \sigma \implies$
 $f (\text{ndropn } j \sigma)$

proof –

fix *j*

show $\forall n. \text{enat } n \leq \text{nlength } \sigma \longrightarrow f (\text{ndropn } n \sigma) \longrightarrow$
 $\text{nlength } \sigma - \text{enat } n = (\text{enat } 0) \vee f (\text{ndropn } (\text{Suc } 0) (\text{ndropn } n \sigma)) \implies$
 $f \sigma \implies$
 $\text{enat } j \leq \text{nlength } \sigma \implies$
 $f (\text{ndropn } j \sigma)$

proof (induct *j* arbitrary: σ)

case 0

then show ?case **by** simp

next

case (Suc *j*)

then show ?case **by** (simp add: ndropn-ndropn)

(metis Suc-ile-eq add.right-neutral enat.simps(3) enat-add-sub-same le-cases
le-iff-add not-less zero-enat-def)

qed

qed

lemma *BoxInductSem:*

($\sigma \models \Box (f \longrightarrow \text{wnext } f) \wedge f \longrightarrow \Box f$)
proof

```

(auto simp add: always-defs wnext-defs )
show  $\bigwedge n. \forall n. \text{enat } n \leq \text{nlength } \sigma \longrightarrow f \text{ (ndropn } n \text{ } \sigma) \longrightarrow$ 
 $\text{nlength } \sigma - \text{enat } n = \text{enat } 0 \vee f \text{ (ndropn (Suc } 0) \text{ (ndropn } n \text{ } \sigma))} \Longrightarrow$ 
 $f \text{ } \sigma \Longrightarrow$ 
 $\text{enat } n \leq \text{nlength } \sigma \Longrightarrow$ 
 $f \text{ (ndropn } n \text{ } \sigma)$ 
using box-induct-help-1 by blast
qed

```

2.1.6 Quantification over State (Flexible) Variables

Quantification in Infinite ITL is done similarly as in Finite ITL.

```
typedecl state
```

```
instance state :: world ..
```

```

type-synonym 'a statefun = (state,'a) stfun
type-synonym statepred = bool statefun
type-synonym 'a tempfun = (state,'a) formfun
type-synonym temporal = state formula

```

2.1.7 Temporal Quantifiers

```

definition exist-state-d :: ('a statefun  $\Rightarrow$  temporal)  $\Rightarrow$  temporal (binder Eex 10)
where exist-state-d F  $\equiv (\lambda s. (\exists x. s \models F x))$ 

```

syntax

```
-Eex :: [idts, lift]  $\Rightarrow$  lift      (( $\exists \exists \exists$  -./ -) [0,10] 10)
```

translations

```
-Eex v A == Eex v. A
```

```

definition forall-state-d :: ('a statefun  $\Rightarrow$  temporal)  $\Rightarrow$  temporal (binder Aall 10)
where forall-state-d F  $\equiv LIFT(\neg(\exists \exists x. \neg(F x)))$ 

```

syntax

```
-Aall :: [idts, lift]  $\Rightarrow$  lift      (( $\exists \forall \forall$  -./ -) [0,10] 10)
```

translations

```
-Aall v A == Aall v. A
```

end

2.2 Axioms and Rules

```
theory ITL
```

```
imports
```


begin

The Finite and Infinite ITL axiom and proof rules are introduced (taken from [12]). The soundness of the rules and axioms are checked using the lemmas of Semantics.thy.

2.2.1 Rules

lemma *MP* :
assumes $\vdash f \longrightarrow g$
 $\vdash f$
shows $\vdash g$
using *assms* **by** *fastforce*

lemma *BoxGen* :
assumes $\vdash f$
shows $\vdash \Box f$
using *assms*
by (*auto simp add: itl-defs Valid-def*)

lemma *BiGen*:
assumes $\vdash f$
shows $\vdash bi\ f$
using *assms*
by (*auto simp add: itl-defs Valid-def*)

2.2.2 Axioms

lemma *ChopAssoc* :
 $\vdash f ; (g ; h) = (f;g);h$
using *ChopAssocSem Valid-def* **by** *blast*

lemma *OrChopImp* :
 $\vdash (f \vee g);h \longrightarrow f;h \vee g;h$
using *OrChopImpSem Valid-def* **by** *blast*

lemma *ChopOrImp* :
 $\vdash f;(g \vee h) \longrightarrow f;g \vee f;h$
using *ChopOrImpSem Valid-def* **by** *blast*

lemma *EmptyChop* :
 $\vdash empty ; f = f$
using *EmptyChopSem Valid-def* **by** *blast*

lemma *ChopEmpty* :
 $\vdash f;empty = f$
using *ChopEmptySem Valid-def* **by** *blast*

lemma *StateImpBi* :
 $\vdash init\ f \longrightarrow bi\ (init\ f)$

using *StateImpBiSem Valid-def* by *blast*

lemma *NextImpNotNextNot* :

$\vdash \bigcirc f \longrightarrow \neg (\bigcirc (\neg f))$

using *NextImpNotNextNotSem Valid-def* by *blast*

lemma *BiBoxChopImpChop* :

$\vdash bi (f \longrightarrow f1) \wedge \Box (g \longrightarrow g1) \longrightarrow f;g \longrightarrow f1;g1$

using *BiBoxChopImpChopSem Valid-def* by *blast*

lemma *BoxInduct* :

$\vdash \Box (f \longrightarrow wnext f) \wedge f \longrightarrow \Box f$

using *BoxInductSem Valid-def* by *blast*

2.2.3 Additional Lemmas

The following is again from [6, 3] but adapted for our need.

lemma *int-eq-true*:

assumes $\vdash P$

shows $\vdash P = \#True$

using *assms* by *auto*

lemma *int-eq*:

assumes $\vdash X = Y$

shows $X = Y$

using *assms* by (*auto simp: inteq-reflection*)

lemma *int-iffI*:

assumes $\vdash F \longrightarrow G$ and $\vdash G \longrightarrow F$

shows $\vdash F = G$

using *assms* by *force*

lemma *int-iffD1*: **assumes** $h: \vdash F = G$ **shows** $\vdash F \longrightarrow G$

using h by *auto*

lemma *int-iffD2*: **assumes** $h: \vdash F = G$ **shows** $\vdash G \longrightarrow F$

using h by *auto*

lemma *lift-imp-trans*:

assumes $\vdash A \longrightarrow B$ and $\vdash B \longrightarrow C$

shows $\vdash A \longrightarrow C$

using *assms* by *force*

lemma *lift-imp-neg*: **assumes** $\vdash A \longrightarrow B$ **shows** $\vdash \neg B \longrightarrow \neg A$

using *assms* by *auto*

lemma *lift-and-com*: $\vdash (A \wedge B) = (B \wedge A)$

by *auto*

2.2.4 Quantification

lemma *EExI* :

$\vdash F\ y \longrightarrow (\exists \exists\ x.\ F\ x)$

by (*auto simp add: exist-state-d-def Valid-def*)

lemma *EExE*:

assumes $\bigwedge x.\ \vdash F\ x \longrightarrow G$

shows $\vdash (\exists \exists\ x.\ F\ x) \longrightarrow G$

using *assms* **by** (*metis (mono-tags, lifting) Valid-def exist-state-d-def unl-lift2*)

lemma *EExVal*:

$((\ w) \models (\exists \exists\ x.\ F\ x)) =$

$(\exists\ x\ (val :: 'a\ nellist). (\ (val = (nmap\ x\ w) \wedge ((\ w) \models F\ x))))$

by (*simp add: exist-state-d-def*)

lemma *AAxDef*:

$\vdash (\forall \forall\ x.\ F\ x) = (\neg(\exists \exists\ x.\ \neg(F\ x)))$

by (*simp add: Valid-def forall-state-d-def exist-state-d-def*)

lemma *ExEqvRule*:

assumes $\bigwedge x.\ \vdash (f\ x) = (g\ x)$

shows $\vdash (\exists\ x.\ f\ x) = (\exists\ x.\ g\ x)$

using *assms* **by** *fastforce*

2.2.5 Lemmas about *current-val*

lemma *current-const*: $\vdash \$(\#c) = \#c$

by (*simp add: current-val-d-def intI*)

lemma *current-fun1*: $\vdash \$ (f\<x>) = f\<\$x>$

by (*simp add: current-val-d-def intI*)

lemma *current-fun2*: $\vdash \$ (f\<x,y>) = f\<\$x,\$y>$

by (*auto simp: current-val-d-def intI*)

lemma *current-fun3*: $\vdash \$ (f\<x,y,z>) = f\<\$x,\$y,\$z>$

by (*auto simp: current-val-d-def intI*)

lemma *current-forall*: $\vdash \$ (\forall\ x.\ P\ x) = (\forall\ x.\ \$ (P\ x))$

by (*auto simp: current-val-d-def intI*)

lemma *current-exists*: $\vdash \$ (\exists\ x.\ P\ x) = (\exists\ x.\ \$ (P\ x))$

by (*auto simp: current-val-d-def intI*)

lemma *current-exists1*: $\vdash \$ (\exists!\ x.\ P\ x) = (\exists!\ x.\ \$ (P\ x))$

by (*auto simp: current-val-d-def intI*)

lemmas *all-current* = *current-const current-fun1 current-fun2 current-fun3*
current-forall current-exists current-exists1

lemmas *all-current-unl* = *all-current*[*THEN intD*]
lemmas *all-current-eq* = *all-current*[*THEN inteq-reflection*]

2.2.6 Lemmas about *next-val*

lemma *next-const*: $\vdash \text{more} \longrightarrow (\#c)\$ = \#c$
by (*auto simp: next-val-d-def more-defs zero-enat-def intI*)

lemma *next-fun1*: $\vdash \text{more} \longrightarrow f\langle x \rangle\$ = f\langle x\$ \rangle$
by (*auto simp: next-val-d-def more-defs zero-enat-def intI*)

lemma *next-fun2*: $\vdash \text{more} \longrightarrow f\langle x, y \rangle\$ = f\langle x\$, y\$ \rangle$
by (*auto simp: next-val-d-def more-defs zero-enat-def intI*)

lemma *next-fun3*: $\vdash \text{more} \longrightarrow f\langle x, y, z \rangle\$ = f\langle x\$, y\$, z\$ \rangle$
by (*auto simp: next-val-d-def more-defs zero-enat-def intI*)

lemma *next-forall*: $\vdash \text{more} \longrightarrow (\forall x. P\ x)\$ = (\forall x. (P\ x)\$)$
by (*auto simp: next-val-d-def intI*)

lemma *next-exists*: $\vdash \text{more} \longrightarrow (\exists x. P\ x)\$ = (\exists x. (P\ x)\$)$
by (*auto simp: next-val-d-def intI*)

lemma *next-exists1*: $\vdash \text{more} \longrightarrow (\exists! x. P\ x)\$ = (\exists! x. (P\ x)\$)$
by (*auto simp: next-val-d-def more-defs zero-enat-def intI*)

lemmas *all-next* = *next-const next-fun1 next-fun2 next-fun3*
next-forall next-exists next-exists1

lemmas *all-next-unl* = *all-next*[*THEN intD*]

2.2.7 Lemmas about *fin-val*

lemma *fin-const*: $\vdash \text{finite} \longrightarrow !(\#c) = \#c$
by (*auto simp: fin-val-d-def finite-defs intI*)

lemma *fin-fun1*: $\vdash \text{finite} \longrightarrow !(f\langle x \rangle) = f\langle !x \rangle$
by (*auto simp: fin-val-d-def finite-defs intI*)

lemma *fin-fun2*: $\vdash \text{finite} \longrightarrow !(f\langle x, y \rangle) = f\langle !x, !y \rangle$
by (*auto simp: fin-val-d-def finite-defs intI*)

lemma *fin-fun3*: $\vdash \text{finite} \longrightarrow !(f\langle x, y, z \rangle) = f\langle !x, !y, !z \rangle$
by (*auto simp: fin-val-d-def finite-defs intI*)

lemma *fin-forall*: $\vdash \text{finite} \longrightarrow !(\forall x. P\ x) = (\forall x. !(P\ x))$
by (*auto simp: fin-val-d-def finite-defs intI*)

lemma *fin-exists*: $\vdash \text{finite} \longrightarrow !(\exists x. P\ x) = (\exists x. !(P\ x))$

by (*auto simp: fin-val-d-def finite-defs intI*)

lemma *fin-exists1*: $\vdash \text{finite} \longrightarrow !(\exists! x. P\ x) = (\exists! x. !(P\ x))$

by (*auto simp: fin-val-d-def finite-defs intI*)

lemmas *all-fin* = *fin-const fin-fun1 fin-fun2 fin-fun3*
fin-forall fin-exists fin-exists1

lemmas *all-fin-unl* = *all-fin[THEN intD]*

2.2.8 Lemmas about *penult-val*

lemma *penult-const*: $\vdash \text{more} \wedge \text{finite} \longrightarrow (\#c)! = \#c$

by (*auto simp: penult-val-d-def more-defs finite-defs zero-enat-def intI*)

lemma *penult-fun1*: $\vdash \text{more} \wedge \text{finite} \longrightarrow f\langle x \rangle! = f\langle x! \rangle$

by (*auto simp: penult-val-d-def more-defs finite-defs zero-enat-def intI*)

lemma *penult-fun2*: $\vdash \text{more} \wedge \text{finite} \longrightarrow f\langle x, y \rangle! = f\langle x!, y! \rangle$

by (*auto simp: penult-val-d-def more-defs finite-defs zero-enat-def intI*)

lemma *penult-fun3*: $\vdash \text{more} \wedge \text{finite} \longrightarrow f\langle x, y, z \rangle! = f\langle x!, y!, z! \rangle$

by (*auto simp: penult-val-d-def more-defs finite-defs zero-enat-def intI*)

lemma *penult-forall*: $\vdash \text{more} \wedge \text{finite} \longrightarrow (\forall x. P\ x)! = (\forall x. (P\ x)!)$

by (*auto simp: penult-val-d-def finite-defs intI*)

lemma *penult-exists*: $\vdash \text{more} \wedge \text{finite} \longrightarrow (\exists x. P\ x)! = (\exists x. (P\ x)!)$

by (*auto simp: penult-val-d-def finite-defs intI*)

lemma *penult-exists1*: $\vdash \text{more} \wedge \text{finite} \longrightarrow (\exists! x. P\ x)! = (\exists! x. (P\ x)!)$

by (*auto simp: penult-val-d-def more-defs finite-defs zero-enat-def intI*)

lemmas *all-penult* = *penult-const penult-fun1 penult-fun2 penult-fun3*
penult-forall penult-exists penult-exists1

lemmas *all-penult-unl* = *all-penult[THEN intD]*

2.2.9 Basic temporal variables properties

lemma *empty-imp-fin-equiv-curr*:

$\vdash \text{empty} \longrightarrow !v = \v

by (*simp add: Valid-def itl-defs nlength-eq-enat-nfiniteD*)

(*metis nlast-NNil nlength-eq-enat-nfiniteD nnth-nlast ntaken-0 ntaken-nlast the-enat-0 zero-enat-def*)

lemma *skip-imp-fin-equiv-next*:

```

  ⊢ skip ⟶ !v = v$
by (simp add: Valid-def itl-defs)
  (metis One-nat-def le-numeral-extra(4) ndropn-0 nlength-eq-enat-nfiniteD
    ntaken-all ntaken-ndropn-nlast one-enat-def plus-1-eq-Suc)

lemma skip-imp-penult-eqv-curr:
  ⊢ skip ⟶ v! = $v
by (simp add: Valid-def itl-defs current-val-d-def nlength-eq-enat-nfiniteD)
  (metis ndropn-0 ndropn-nfirst)

end

```

2.3 Weak chop operator

```

theory Theorems
imports
  ITL
begin

```

We give the proofs of a list of Finite and Infinite ITL theorems. These proofs and theorems are from [11] but adapted for infinite and finite intervals.

2.3.1 Propositional reasoning

This is a list of propositional logic theorems used in the proofs of the ITL theorems.

```

lemma IfThenElseImp:
  ⊢ (ifi g then f else f1) ⟶ ((g ⟶ f) ∧ (¬g ⟶ f1))
by (simp add: itl-def Valid-def)

```

```

lemma Prop01:
  assumes ⊢ f ⟶ ¬g ∨ h
  shows ⊢ g ∧ f ⟶ h
using assms by auto

```

```

lemma Prop02:
  assumes ⊢ f ⟶ g
          ⊢ f1 ⟶ g
  shows ⊢ f ∨ f1 ⟶ g
using assms by fastforce

```

```

lemma Prop03:
  assumes ⊢ f = (g ∨ h)
  shows ⊢ h ⟶ f
using assms by auto

```

```

lemma Prop04:
  assumes ⊢ f = h
          ⊢ f = h1
  shows ⊢ h1 = h

```

using *assms* **using** *int-eq* **by** *auto*

lemma *Prop05*:

assumes $\vdash f \longrightarrow g$
shows $\vdash f \longrightarrow h \vee g$
using *assms* **by** *auto*

lemma *Prop06*:

assumes $\vdash f = (g \vee h)$
 $\vdash h = h1$
shows $\vdash f = (g \vee h1)$
using *assms* **by** *fastforce*

lemma *Prop07*:

assumes $\vdash f \longrightarrow g \vee h$
shows $\vdash f \wedge \neg g \longrightarrow h$
using *assms* **by** *auto*

lemma *Prop08*:

assumes $\vdash f \longrightarrow g \vee h$
 $\vdash h \longrightarrow h1$
shows $\vdash f \longrightarrow g \vee h1$
using *assms* **by** *fastforce*

lemma *Prop09*:

assumes $\vdash f \wedge g \longrightarrow h$
shows $\vdash f \longrightarrow (g \longrightarrow h)$
using *assms* **by** *auto*

lemma *Prop10*:

assumes $\vdash f \longrightarrow g$
shows $\vdash f = (f \wedge g)$
using *assms* **by** *auto*

lemma *Prop11*:

$(\vdash f = f1) = ((\vdash f \longrightarrow f1) \wedge (\vdash f1 \longrightarrow f))$
by (*auto simp: Valid-def*)

lemma *Prop12*:

$(\vdash f \longrightarrow (f1 \wedge f2)) = ((\vdash f \longrightarrow f1) \wedge (\vdash f \longrightarrow f2))$
by (*auto simp: Valid-def*)

lemma *Prop13*:

assumes $\vdash f \longrightarrow g \vee h$
shows $\vdash f \wedge \neg h \longrightarrow g$
using *assms* **by** (*auto simp: Valid-def*)

2.3.2 State formulas

The *init* operator denotes state formulas, i.e., ITL formula that only constrain the first state of an interval.

lemma *Initprop* :

$\vdash ((init\ f) \wedge (init\ g)) = init(f \wedge g)$
 $\vdash (\neg (init\ f)) = init(\neg f)$
 $\vdash ((init\ f) \vee (init\ g)) = init(f \vee g)$
 $\vdash init\ \#True$

by (*auto simp: itl-defs*)

lemma *Finprop* :

$\vdash ((\#True;(f \wedge empty)) \wedge (\#True;(g \wedge empty))) = (\#True;((f \wedge g) \wedge empty))$
 $\vdash ((\#True;(f \wedge empty)) \vee (\#True;(g \wedge empty))) = (\#True;((f \vee g) \wedge empty))$
 $\vdash (\#True;((\#True) \wedge empty))$
 $\vdash finite \longrightarrow (\neg (\#True;(f \wedge empty))) = (\#True;(\neg f \wedge empty))$
 $\vdash (\neg (\#True;(f \wedge empty))) = ((\#True;(\neg f \wedge empty)) \wedge finite)$

using *nfinite-nlength-enat*

by (*auto simp: finalt-defs finite-defs zero-enat-def,*
auto simp add: itl-defs nfinite-nlength-enat zero-enat-def, force)

2.3.3 finite and inf properties

lemma *EmptyImpFinite*:

$\vdash empty \longrightarrow finite$

using *nlength-eq-enat-nfiniteD* **by** (*auto simp add: itl-defs zero-enat-def*)

lemma *SkipChopFiniteImpFinite*:

$\vdash skip;finite \longrightarrow finite$

using *nfinite-ndropn nlength-eq-enat-nfiniteD*

by (*simp add: Valid-def itl-defs, force*)

lemma *FiniteChopSkipImpFinite*:

$\vdash finite;skip \longrightarrow finite$

using *nlength-eq-enat-nfiniteD*

by (*simp add: Valid-def itl-defs, force*)

lemma *FiniteChopSkipImpMore*:

$\vdash finite;skip \longrightarrow more$

using *nlength-eq-enat-nfiniteD one-enat-def*

by (*simp add: Valid-def itl-defs, force*)

lemma *FiniteAndMoreImpFiniteChopSkip*:

$\vdash finite \wedge more \longrightarrow finite;skip$

by (*auto simp add: Valid-def itl-defs zero-enat-def*)

(*metis Suc-ile-eq diff-diff-cancel diff-le-self enat-ord-simps(1) idiff-enat-enat nfinite-nlength-enat*)

lemma *FiniteChopSkipEqvFiniteAndMore*:

$\vdash finite;skip = (finite \wedge more)$

by (*simp add: FiniteAndMoreImpFiniteChopSkip FiniteChopSkipImpFinite FiniteChopSkipImpMore*)

Prop12 int-iffI)

lemma *FiniteChopSkipEqvSkipChopFinite:*

$\vdash \text{finite};\text{skip} = \text{skip};\text{finite}$

by (*auto simp add: Valid-def itl-defs*)

(metis enat.distinct(1) enat-add-sub-same enat-le-plus-same(2) le-iff-add ,
metis eSuc-enat enat.simps(3) enat-add-sub-same idiff-enat-0-right illess-Suc-eq le-zero-eq
less-eqE min.orderE nlength-eq-enat-nfiniteD not-le one-eSuc plus-1-eSuc(2),
metis add commute enat.simps(3) enat-add-sub-same idiff-enat-enat le-iff-add
nfinite-nlength-enat)

lemma *FiniteAndEmptyEqvEmpty:*

$\vdash (\text{finite} \wedge \text{empty}) = \text{empty}$

by (*auto simp add: Valid-def itl-defs nlength-eq-enat-nfiniteD
zero-enat-def*)

lemma *FiniteChopFiniteEqvFinite:*

$\vdash \text{finite};\text{finite} = \text{finite}$

by (*auto simp add: Valid-def itl-defs*) (metis zero-enat-def zero-le)

lemma *InfChopInfEqvInf:*

$\vdash \text{inf};\text{inf} = \text{inf}$

by (*simp add: Valid-def itl-defs*)

lemma *InfChopFiniteEqvInf:*

$\vdash \text{inf};\text{finite} = \text{inf}$

by (*simp add: Valid-def itl-defs*)

lemma *FiniteChopInfEqvInf:*

$\vdash \text{finite};\text{inf} = \text{inf}$

by (*auto simp add: Valid-def itl-defs*) (metis zero-enat-def zero-le)

lemma *InfEqvNotFinite:*

$\vdash \text{inf} = (\neg \text{finite})$

by (*simp add: Valid-def itl-defs*)

lemma *FiniteEqvNotInf:*

$\vdash \text{finite} = (\neg \text{inf})$

by (*simp add: Valid-def itl-defs*)

lemma *ChopTrueAndFiniteEqvAndFiniteChopFinite:*

$\vdash ((f;\# \text{True}) \wedge \text{finite}) = (f \wedge \text{finite});\text{finite}$

by (*auto simp add: Valid-def itl-defs*)

lemma *TrueChopAndFiniteEqvAndFiniteChopFinite:*

$\vdash ((\# \text{True};f) \wedge \text{finite}) = \text{finite};(f \wedge \text{finite})$

by (*auto simp add: Valid-def itl-defs*)

lemma *FiniteChopMoreEqvMore:*

$\vdash \text{finite};\text{more} = \text{more}$

by (*auto simp add: Valid-def itl-defs*)
 (*metis idiff-0-right zero-enat-def zero-le*)

lemma *ChopAndFiniteDist*:
 $\vdash ((f;g) \wedge \text{finite}) = (f \wedge \text{finite});(g \wedge \text{finite})$
by (*auto simp add: Valid-def itl-defs*)

lemma *FiniteOrInfinite*:
 $\vdash \text{finite} \vee \text{inf}$
by (*simp add: Valid-def itl-defs*)

lemma *FiniteImpAnd*:
assumes $\vdash \text{finite} \longrightarrow f = g$
shows $\vdash (f \wedge \text{finite}) = (g \wedge \text{finite})$
using *assms* **by** (*auto simp add: Valid-def itl-defs*)

lemma *FmoreEqvSkipChopFinite*:
 $\vdash \text{fmore} = \text{skip};\text{finite}$
by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite*
 fmore-d-def inteq-reflection lift-and-com)

lemma *FiniteImp*:
 $\vdash (f \wedge \text{finite} \longrightarrow g) = (f \wedge \text{finite} \longrightarrow g \wedge \text{finite})$
by (*simp add: itl-defs Valid-def*)

lemma *ChopAndInf*:
 $\vdash ((f;g) \wedge \text{inf}) = (f; (g \wedge \text{inf}))$
by (*auto simp add: Valid-def itl-defs*)

lemma *ChopAndInfB*:
 $\vdash ((f;g) \wedge \text{inf}) = ((f \wedge \text{inf}) \vee (f \wedge \text{finite});(g \wedge \text{inf}))$
by (*auto simp add: Valid-def itl-defs*)

lemma *MoreAndInfEqvInf*:
 $\vdash (\text{more} \wedge \text{inf}) = \text{inf}$
by (*metis ChopAndInfEmptyImpFinite FiniteChopMoreEqvMore InfEqvNotFinite Prop11 Prop12 empty-d-def*
 finite-d-def int-simps(32) inteq-reflection)

lemma *AndInfChopAndInfEqvAndInf*:
 $\vdash (f \wedge \text{inf});(f \wedge \text{inf}) = (f \wedge \text{inf})$
by (*auto simp add: Valid-def itl-defs*)

lemma *AndInfChopEqvAndInf*:
 $\vdash (f \wedge \text{inf});g = (f \wedge \text{inf})$
by (*auto simp add: Valid-def itl-defs*)

lemma *AndMoreAndInfEqvAndInf*:
 $\vdash ((f \wedge \text{more}) \wedge \text{inf}) = (f \wedge \text{inf})$
by (*auto simp add: Valid-def itl-defs nlength-eq-enat-nfiniteD zero-enat-def*)
 (*metis gr-zeroI nlength-eq-enat-nfiniteD zero-enat-def*)

lemma *AndMoreAndFiniteEqvAndFmore*:

$\vdash ((f \wedge \text{more}) \wedge \text{finite}) = (f \wedge \text{fmore})$

by (*auto simp add: Valid-def itl-defs*)

lemma *NotFmoreAndEmpty*:

$\vdash \neg (\text{empty} \wedge \text{fmore})$

by (*auto simp add: fmore-d-def empty-d-def*)

lemma *NotFmoreAndInf*:

$\vdash \neg ((f \wedge \text{inf}) \wedge \text{fmore})$

by (*auto simp add: fmore-d-def finite-d-def infinite-d-def*)

lemma *FmoreChopAnd*:

$\vdash (((f \wedge \text{more}); g) \wedge \text{fmore}) = ((f \wedge \text{fmore}); (g \wedge \text{finite}))$

by (*auto simp add: Valid-def itl-defs*)

lemma *NotEmptyAndInf*:

$\vdash \neg (\text{empty} \wedge \text{inf})$

by (*auto simp add: Valid-def itl-defs nlength-eq-enat-nfiniteD zero-enat-def*)

lemma *OrFiniteInf*:

$\vdash f = ((f \wedge \text{finite}) \vee (f \wedge \text{inf}))$

by (*auto simp add: Valid-def itl-defs*)

lemma *AndInfEqvChopFalse*:

$\vdash (f \wedge \text{inf}) = f; \# \text{False}$

by (*auto simp add: Valid-def itl-defs*)

lemma *EmptyOrMoreSplit*:

$\vdash f = ((f \wedge \text{empty}) \vee (f \wedge \text{more}))$

unfolding *empty-d-def* **by** *auto*

2.3.4 Basic Theorems

lemma *BiChopImpChop* :

$\vdash \text{bi } (f \longrightarrow f1) \longrightarrow f; g \longrightarrow f1; g$

proof –

have 1: $\vdash g \longrightarrow g$ **by** *auto*

hence 2: $\vdash \Box (g \longrightarrow g)$ **by** (*rule BoxGen*)

have 3: $\vdash \text{bi } (f \longrightarrow f1) \wedge \Box (g \longrightarrow g) \longrightarrow f; g \longrightarrow f1; g$ **by** (*rule BiBoxChopImpChop*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *AndChopA*:

$\vdash (f \wedge f1); g \longrightarrow f; g$

proof –

have 1: $\vdash f \wedge f1 \longrightarrow f$ **by** *auto*

hence 2: $\vdash \text{bi } (f \wedge f1 \longrightarrow f)$ **by** (*rule BiGen*)

have 3: $\vdash bi (f \wedge f1 \longrightarrow f) \longrightarrow (f \wedge f1);g \longrightarrow f;g$ **by** (rule *BiChopImpChop*)
from 2 3 **show** ?thesis **using** MP **by** blast
qed

lemma *AndChopB*:

$\vdash (f \wedge f1);g \longrightarrow f1;g$

proof –

have 1: $\vdash f \wedge f1 \longrightarrow f1$ **by** auto

hence 2: $\vdash bi (f \wedge f1 \longrightarrow f1)$ **by** (rule *BiGen*)

have 3: $\vdash bi (f \wedge f1 \longrightarrow f1) \longrightarrow (f \wedge f1);g \longrightarrow f1;g$ **by** (rule *BiChopImpChop*)

from 2 3 **show** ?thesis **using** MP **by** blast

qed

lemma *NextChop*:

$\vdash (\bigcirc f);g = \bigcirc(f;g)$

proof –

have 1: $\vdash skip;(f;g) = (skip;f);g$ **by** (rule *ChopAssoc*)

show ?thesis **by** (metis 1 int-eq next-d-def)

qed

lemma *BoxChopImpChop* :

$\vdash \Box (g \longrightarrow g1) \longrightarrow f;g \longrightarrow f;g1$

proof –

have 1: $\vdash g \longrightarrow g$ **by** auto

hence 2: $\vdash bi (g \longrightarrow g)$ **by** (rule *BiGen*)

have 3: $\vdash bi (f \longrightarrow f) \wedge \Box(g \longrightarrow g1) \longrightarrow f;g \longrightarrow f;g1$ **by** (rule *BiBoxChopImpChop*)

from 2 3 **show** ?thesis **by** fastforce

qed

lemma *LeftChopImpChop*:

assumes $\vdash f \longrightarrow f1$

shows $\vdash f;g \longrightarrow f1;g$

proof –

have 1: $\vdash f \longrightarrow f1$ **using** assms **by** auto

hence 2: $\vdash bi (f \longrightarrow f1)$ **by** (rule *BiGen*)

have 3: $\vdash bi (f \longrightarrow f1) \longrightarrow f;g \longrightarrow f1;g$ **by** (rule *BiChopImpChop*)

from 2 3 **show** ?thesis **using** MP **by** blast

qed

lemma *RightChopImpChop*:

assumes $\vdash g \longrightarrow g1$

shows $\vdash f;g \longrightarrow f;g1$

proof –

have 1: $\vdash g \longrightarrow g1$ **using** assms **by** auto

hence 2: $\vdash \Box (g \longrightarrow g1)$ **by** (rule *BoxGen*)

have 3: $\vdash \Box (g \longrightarrow g1) \longrightarrow f;g \longrightarrow f;g1$ **by** (rule *BoxChopImpChop*)

from 2 3 **show** ?thesis **using** MP **by** blast

qed

lemma *RightChopEqvChop*:

assumes $\vdash g = g1$
shows $\vdash (f;g) = (f;g1)$
using *assms RightChopImpChop[of g g1 f] RightChopImpChop[of g1 g f]*
by *fastforce*

lemma *InfRightChopEqvChop*:
assumes $\vdash inf \longrightarrow g = g1$
shows $\vdash inf \longrightarrow (f;g) = (f;g1)$
using *assms*
by (*auto simp add: Valid-def itl-defs*)

lemma *ChopOrEqv*:
 $\vdash f;(g \vee g1) = (f;g \vee f;g1)$
proof –
have $1: \vdash g \longrightarrow g \vee g1$ **by** *auto*
hence $2: \vdash f;g \longrightarrow f;(g \vee g1)$ **by** (*rule RightChopImpChop*)
have $3: \vdash g1 \longrightarrow g \vee g1$ **by** *auto*
hence $4: \vdash f;g1 \longrightarrow f;(g \vee g1)$ **by** (*rule RightChopImpChop*)
from $2\ 4$ **show** *?thesis* **by** (*meson ChopOrImp Prop02 Prop11*)
qed

lemma *OrChopEqv*:
 $\vdash (f \vee f1);g = (f;g \vee f1;g)$
proof –
have $1: \vdash f \longrightarrow f \vee f1$ **by** *auto*
hence $2: \vdash f;g \longrightarrow (f \vee f1);g$ **by** (*rule LeftChopImpChop*)
have $3: \vdash f1 \longrightarrow f \vee f1$ **by** *auto*
hence $4: \vdash f1;g \longrightarrow (f \vee f1);g$ **by** (*rule LeftChopImpChop*)
from $2\ 4$ **show** *?thesis*
by (*meson OrChopImp int-iffI Prop02*)
qed

lemma *OrChopImpRule*:
assumes $\vdash f \longrightarrow f1 \vee f2$
shows $\vdash f;g \longrightarrow (f1;g) \vee (f2;g)$
proof –
have $1: \vdash f \longrightarrow f1 \vee f2$ **using** *assms* **by** *auto*
hence $2: \vdash f;g \longrightarrow (f1 \vee f2);g$ **by** (*rule LeftChopImpChop*)
have $3: \vdash (f1 \vee f2);g = (f1;g \vee f2;g)$ **by** (*rule OrChopEqv*)
from $2\ 3$ **show** *?thesis* **by** *fastforce*
qed

lemma *LeftChopEqvChop*:
assumes $\vdash f = f1$
shows $\vdash f;g = (f1;g)$
proof –
have $1: \vdash f = f1$ **using** *assms* **by** *auto*
hence $2: \vdash f \longrightarrow f1$ **by** *auto*
hence $3: \vdash f;g \longrightarrow f1;g$ **by** (*rule LeftChopImpChop*)
have $\vdash f1 \longrightarrow f$ **using** 1 **by** *auto*

hence 4: $\vdash f1;g \longrightarrow f;g$ **by** (rule *LeftChopImpChop*)
 from 3 4 **show** ?thesis **by** (simp add: int-iffI)
 qed

lemma *OrChopEqvRule*:

assumes $\vdash f = (f1 \vee f2)$
shows $\vdash f;g = ((f1;g) \vee (f2;g))$

proof –

have 1: $\vdash f = (f1 \vee f2)$ **using** *assms* **by** *auto*
hence 2: $\vdash f;g = ((f1 \vee f2);g)$ **by** (rule *LeftChopEqvChop*)
have 3: $\vdash (f1 \vee f2);g = (f1;g \vee f2;g)$ **by** (rule *OrChopEqv*)
from 2 3 **show** ?thesis **by** *fastforce*

qed

lemma *NextImpNext*:

assumes $\vdash f \longrightarrow g$
shows $\vdash \bigcirc f \longrightarrow \bigcirc g$

proof –

have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence 2: $\vdash \Box (f \longrightarrow g)$ **by** (rule *BoxGen*)
have 3: $\vdash \Box (f \longrightarrow g) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **by** (rule *BoxChopImpChop*)
have 4: $\vdash (skip;f) \longrightarrow (skip;g)$ **by** (metis 2 3 MP)
from 4 **show** ?thesis **by** (metis next-d-def)

qed

lemma *ChopOrImpRule*:

assumes $\vdash g \longrightarrow g1 \vee g2$
shows $\vdash f;g \longrightarrow (f;g1) \vee (f;g2)$

proof –

have 1: $\vdash g \longrightarrow g1 \vee g2$ **using** *assms* **by** *auto*
hence 2: $\vdash f;g \longrightarrow f;(g1 \vee g2)$ **by** (rule *RightChopImpChop*)
have 3: $\vdash f;(g1 \vee g2) = (f;g1 \vee f;g2)$ **by** (rule *ChopOrEqv*)
from 2 3 **show** ?thesis **by** *fastforce*

qed

lemma *NextImpDist*:

$\vdash \bigcirc (f \longrightarrow g) \longrightarrow \bigcirc f \longrightarrow \bigcirc g$

proof –

have 1: $\vdash (\neg (f \longrightarrow g)) = (f \wedge \neg g)$ **by** *auto*
hence 2: $\vdash skip;(\neg (f \longrightarrow g)) = skip;(f \wedge \neg g)$ **by** (rule *RightChopEqvChop*)
have 3: $\vdash f \longrightarrow g \vee (f \wedge \neg g)$ **by** *auto*
hence 4: $\vdash skip;f \longrightarrow (skip;g) \vee (skip;(f \wedge \neg g))$ **by** (rule *ChopOrImpRule*)
hence 5: $\vdash \neg (skip;(f \wedge \neg g)) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **by** *auto*
have 6: $\vdash \neg (skip;(\neg (f \longrightarrow g))) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **using** 2 5 **by** *fastforce*
hence 7: $\vdash \neg (\bigcirc(\neg (f \longrightarrow g))) \longrightarrow (\bigcirc f) \longrightarrow (\bigcirc g)$ **by** (simp add: next-d-def)
have 8: $\vdash \bigcirc(f \longrightarrow g) \longrightarrow \neg (\bigcirc(\neg (f \longrightarrow g)))$ **by** (rule *NextImpNotNextNot*)
from 7 8 **show** ?thesis **using** *lift-imp-trans* **by** *blast*

qed

lemma *FiniteChopImpDiamond*:

$\vdash (f \wedge \text{finite});g \longrightarrow \Diamond g$
proof –
have 1: $\vdash f \wedge \text{finite} \longrightarrow \text{finite}$ **by** *auto*
hence 2: $\vdash (f \wedge \text{finite});g \longrightarrow \text{finite};g$ **by** (*rule LeftChopImpChop*)
from 2 **show** ?thesis **by** (*simp add: sometimes-d-def*)
qed

lemma *NowImpDiamond*:
 $\vdash f \longrightarrow \Diamond f$
proof –
have 1: $\vdash \text{empty};f = f$ **by** (*rule EmptyChop*)
have 2: $\vdash \text{empty} \longrightarrow \text{finite}$ **by** (*rule EmptyImpFinite*)
hence 3: $\vdash \text{empty};f \longrightarrow \text{finite};f$ **by** (*rule LeftChopImpChop*)
have 4: $\vdash f \longrightarrow \text{finite};f$ **using** 1 3 **by** *fastforce*
from 4 **show** ?thesis **by** (*simp add: sometimes-d-def*)
qed

lemma *BoxElim*:
 $\vdash \Box f \longrightarrow f$
proof –
have 1: $\vdash \neg f \longrightarrow \Diamond (\neg f)$ **by** (*rule NowImpDiamond*)
hence 2: $\vdash \neg (\Diamond (\neg f)) \longrightarrow f$ **by** *auto*
from 2 **show** ?thesis **by** (*metis always-d-def*)
qed

lemma *NextDiamondImpDiamond*:
 $\vdash \Diamond (\Diamond f) \longrightarrow \Diamond f$
proof –
have 1: $\vdash \text{skip};(\text{finite};f) = ((\text{skip};\text{finite});f)$ **by** (*rule ChopAssoc*)
hence 2: $\vdash (\text{skip};\text{finite});f = \text{skip};(\text{finite};f)$ **by** *auto*
hence 3: $\vdash (\text{skip};\text{finite});f = \Diamond (\Diamond f)$ **by** (*simp add: next-d-def sometimes-d-def*)
have 4: $\vdash (\text{skip};\text{finite});f \longrightarrow \Diamond f$
by (*simp add: SkipChopFiniteImpFinite LeftChopImpChop sometimes-d-def*)
from 3 4 **show** ?thesis **by** *fastforce*
qed

lemma *BoxImpNowAndWeakNext*:
 $\vdash \Box f \longrightarrow (f \wedge \text{wnext} (\Box f))$
proof –
have 1: $\vdash \neg f \longrightarrow \Diamond (\neg f)$ **by** (*rule NowImpDiamond*)
hence 2: $\vdash \neg (\Diamond (\neg f)) \longrightarrow f$ **by** *auto*
hence 3: $\vdash \Box f \longrightarrow f$ **by** (*metis always-d-def*)
have 4: $\vdash \Diamond (\Diamond (\neg f)) \longrightarrow \Diamond (\neg f)$ **by** (*rule NextDiamondImpDiamond*)
have 5: $\vdash \neg \neg (\Diamond (\neg f)) \longrightarrow \Diamond (\neg f)$ **by** *auto*
hence 6: $\vdash \Diamond (\neg \neg (\Diamond (\neg f))) \longrightarrow \Diamond (\neg f)$ **by** (*rule NextImpNext*)
have 7: $\vdash \Diamond (\neg \neg (\Diamond (\neg f))) \longrightarrow \Diamond (\neg f)$ **using** 4 6 **by** *auto*
hence 8: $\vdash \Diamond (\neg (\Box f)) \longrightarrow \Diamond (\neg f)$ **by** (*simp add: always-d-def*)
hence 9: $\vdash \neg (\Diamond (\neg f)) \longrightarrow \neg (\Diamond (\neg (\Box f)))$ **by** *auto*
hence 10: $\vdash \Box f \longrightarrow \text{wnext} (\Box f)$ **by** (*simp add: always-d-def wnext-d-def*)

from 3 10 show ?thesis by fastforce
qed

lemma *BoxImpBoxRule*:

assumes $\vdash f \longrightarrow g$

shows $\vdash \Box f \longrightarrow \Box g$

proof –

have 1: $\vdash f \longrightarrow g$ using *assms* by *auto*

hence 2: $\vdash \neg g \longrightarrow \neg f$ by *auto*

hence 3: $\vdash \Box(\neg g \longrightarrow \neg f)$ by (rule *BoxGen*)

have 4: $\vdash \Box(\neg g \longrightarrow \neg f) \longrightarrow (\text{finite}; \neg g) \longrightarrow (\text{finite}; \neg f)$ by (rule *BoxChopImpChop*)

have 5: $\vdash (\text{finite}; \neg g) \longrightarrow (\text{finite}; \neg f)$ using 3 4 MP by *blast*

hence 6: $\vdash \Diamond(\neg g) \longrightarrow \Diamond(\neg f)$ by (simp add: *sometimes-d-def*)

hence 7: $\vdash \neg(\Diamond(\neg f)) \longrightarrow \neg(\Diamond(\neg g))$ by *auto*

from 7 show ?thesis by (simp add: *always-d-def*)

qed

lemma *BoxImpDist*:

$\vdash \Box(f \longrightarrow g) \longrightarrow \Box f \longrightarrow \Box g$

proof –

have 1: $\vdash (f \longrightarrow g) \longrightarrow (\neg g \longrightarrow \neg f)$

by *auto*

hence 2: $\vdash \Box(f \longrightarrow g) \longrightarrow \Box(\neg g \longrightarrow \neg f)$

by (rule *BoxImpBoxRule*)

have 3: $\vdash \Box((\neg g) \longrightarrow \neg f) \longrightarrow (\text{finite}; (\neg g)) \longrightarrow (\text{finite}; (\neg f))$

by (rule *BoxChopImpChop*)

have 4: $\vdash \Box(f \longrightarrow g) \longrightarrow (\text{finite}; (\neg g)) \longrightarrow (\text{finite}; (\neg f))$

using 2 3 *lift-imp-trans* by *blast*

hence 5: $\vdash \Box(f \longrightarrow g) \longrightarrow \Diamond(\neg g) \longrightarrow \Diamond(\neg f)$

by (simp add: *sometimes-d-def*)

hence 6: $\vdash \Box(f \longrightarrow g) \longrightarrow \neg(\Diamond(\neg f)) \longrightarrow \neg(\Diamond(\neg g))$

by *auto*

from 6 show ?thesis by (simp add: *always-d-def*)

qed

lemma *DiamondEmptyEqvFinite*:

$\vdash \Diamond \text{empty} = \text{finite}$

proof –

have 1: $\vdash \text{finite}; \text{empty} = \text{finite}$ by (rule *ChopEmpty*)

from 1 show ?thesis by (simp add: *sometimes-d-def*)

qed

lemma *NextEqvNext*:

assumes $\vdash f = g$

shows $\vdash \bigcirc f = \bigcirc g$

proof –

have 1: $\vdash f = g$ using *assms* by *auto*

hence 2: $\vdash \text{skip}; f = \text{skip}; g$ by (rule *RightChopEqvChop*)

from 1 show ?thesis by (metis 2 *next-d-def*)

qed

lemma *NextAndNextImpNextRule*:

assumes $\vdash (f \wedge g) \longrightarrow h$

shows $\vdash (\bigcirc f \wedge \bigcirc g) \longrightarrow \bigcirc h$

using *assms*

by (*simp add: Valid-def itl-defs*)

lemma *NextAndNextEqvNextRule*:

assumes $\vdash (f \wedge g) = h$

shows $\vdash (\bigcirc f \wedge \bigcirc g) = \bigcirc h$

using *assms*

by (*simp add: NextAndNextImpNextRule NextImpNext Prop11 Prop12*)

lemma *WeakNextEqvWeakNext*:

assumes $\vdash f = g$

shows $\vdash \text{wnext } f = \text{wnext } g$

using *assms* **using** *inteq-reflection* **by** *force*

lemma *DiamondImpDiamond*:

assumes $\vdash f \longrightarrow g$

shows $\vdash \Diamond f \longrightarrow \Diamond g$

using *assms* **by** (*simp add: RightChopImpChop sometimes-d-def*)

lemma *DiamondEqvDiamond*:

assumes $\vdash f = g$

shows $\vdash \Diamond f = \Diamond g$

using *assms* **using** *int-eq* **by** *force*

lemma *BoxEqvBox*:

assumes $\vdash f = g$

shows $\vdash \Box f = \Box g$

using *assms* **using** *inteq-reflection* **by** *force*

lemma *BoxAndBoxImpBoxRule*:

assumes $\vdash f \wedge g \longrightarrow h$

shows $\vdash \Box f \wedge \Box g \longrightarrow \Box h$

using *assms* **by** (*auto simp: itl-defs Valid-def*)

lemma *BoxAndBoxEqvBoxRule*:

assumes $\vdash (f \wedge g) = h$

shows $\vdash (\Box f \wedge \Box g) = \Box h$

using *assms* *BoxAndBoxImpBoxRule* *BoxImpBoxRule* **by** (*metis int-iffD1 int-iffD2 int-iffI Prop12*)

lemma *ImpBoxRule*:

assumes $\vdash f \longrightarrow g$

shows $\vdash \Box f \longrightarrow \Box g$

using *assms* **by** (*simp add: BoxImpBoxRule*)

lemma *WnextEqvEmptyOrNext*:

$\vdash \text{wnext } f = (\text{empty} \vee \bigcirc f)$

by (auto simp: Valid-def itl-defs zero-enat-def)

lemma *BoxIntro*:

assumes $\vdash f \longrightarrow g$
 $\vdash \text{more} \wedge f \longrightarrow \bigcirc f$

shows $\vdash f \longrightarrow \Box g$

proof –

have 1: $\vdash \text{more} \wedge f \longrightarrow \bigcirc f$

using *assms* **by** *auto*

hence 2: $\vdash f \longrightarrow (\text{empty} \vee \bigcirc f)$

unfolding *empty-d-def* **by** *fastforce*

hence 3: $\vdash f \longrightarrow \text{wnext } f$

by (*metis WnextEqvEmptyOrNext inteq-reflection*)

hence 4: $\vdash \Box(f \longrightarrow \text{wnext } f)$

by (*rule BoxGen*)

have 5: $\vdash (\Box(f \longrightarrow \text{wnext } f)) \wedge f \longrightarrow \Box f$

by (*rule BoxInduct*)

hence 6: $\vdash (\Box(f \longrightarrow \text{wnext } f)) \longrightarrow (f \longrightarrow \Box f)$

by *fastforce*

have 7: $\vdash f \longrightarrow \Box f$

using 4 6 *MP* **by** *blast*

have 8: $\vdash \Box f \longrightarrow f$

by (*rule BoxElim*)

have 9: $\vdash f = \Box f$

using 7 8 **by** *fastforce*

have 10: $\vdash f \longrightarrow g$

using *assms* **by** *auto*

hence 11: $\vdash \Box f \longrightarrow \Box g$

by (*rule ImpBoxRule*)

from 7 9 11 **show** *?thesis*

using *lift-imp-trans* **by** *blast*

qed

lemma *NextLoop*:

assumes $\vdash f \longrightarrow \bigcirc f$

shows $\vdash \text{finite} \longrightarrow \neg f$

proof –

have 1: $\vdash f \longrightarrow \bigcirc f$

using *assms* **by** *auto*

hence 2: $\vdash f \longrightarrow (\text{more} \wedge \text{wnext } f)$

unfolding *more-d-def wnext-d-def*

by (*metis NextImpNext NextImpNotNextNot Prop05 Prop10 Prop12 int-iffD1 int-simps(29) lift-imp-trans*)

hence 3: $\vdash f \longrightarrow \text{wnext } f$

by *auto*

hence 4: $\vdash \Box(f \longrightarrow \text{wnext } f)$

by (*rule BoxGen*)

have 5: $\vdash \Box(f \longrightarrow \text{wnext } f) \wedge f \longrightarrow \Box f$

by (*rule BoxInduct*)

hence 6: $\vdash \Box(f \longrightarrow \text{wnext } f) \longrightarrow (f \longrightarrow \Box f)$

by *fastforce*

have 7: $\vdash f \longrightarrow \Box f$
using 4 6 MP **by** blast
have 8: $\vdash \Box f \longrightarrow f$
by (rule BoxElim)
have 9: $\vdash f = \Box f$
using 7 8 **by** fastforce
have 10: $\vdash f \longrightarrow \text{more}$
using 2 **by** auto
hence 11: $\vdash \Box f \longrightarrow \Box \text{more}$
by (rule ImpBoxRule)
have 12: $\vdash \text{finite} = (\neg(\Box \text{more}))$
by (metis DiamondEmptyEqvFinite InfEqvNotFinite always-d-def empty-d-def finite-d-def int-eq)
from 7 9 11 12 **show** ?thesis
by fastforce
qed

lemma NotEmptyAndNext:
 $\vdash \neg(\text{empty} \wedge \bigcirc f)$
by (auto simp: Valid-def itl-defs zero-enat-def)

lemma BoxEqvAndWnextBox:
 $\vdash \Box f = (f \wedge \text{wnext } (\Box f))$
proof –
have 1: $\vdash \Box f \longrightarrow f \wedge \text{wnext } (\Box f)$
using BoxImpNowAndWeakNext **by** blast
have 2: $\vdash f \wedge \text{wnext } (\Box f) \longrightarrow f$
by auto
have 3: $\vdash \text{more} \wedge (f \wedge \text{wnext } (\Box f)) \longrightarrow \bigcirc (f \wedge \text{wnext } (\Box f))$
using 1 NextImpNext WnextEqvEmptyOrNext empty-d-def int-iffD1
by (metis Prop01 Prop05 Prop08)
have 4: $\vdash f \wedge \text{wnext } (\Box f) \longrightarrow \Box f$
using 2 3 BoxIntro **by** blast
from 1 4 **show** ?thesis **by** fastforce
qed

lemma BoxEqvAndEmptyOrNextBox:
 $\vdash \Box f = (f \wedge (\text{empty} \vee \bigcirc(\Box f)))$
using BoxEqvAndWnextBox WnextEqvEmptyOrNext **by** (metis int-eq)

lemma BoxEqvBoxBox:
 $\vdash \Box f = \Box(\Box f)$
using BoxGen BoxInduct
by (metis BoxImpNowAndWeakNext MP int-iffI Prop09 Prop12)

lemma BoxBoxImpBox:
 $\vdash \Box(\Box h) \longrightarrow \Box h$
by (simp add: BoxElim)

lemma BoxImpBoxBox:
 $\vdash \Box h \longrightarrow \Box(\Box h)$

by (simp add: BoxEqvBoxBox int-iffD1)

lemma *DiamondIntroC*:

assumes $\vdash f \longrightarrow \bigcirc g$

shows $\vdash f \longrightarrow \Diamond g$

using *assms*

by (*metis* (*no-types*, *lifting*) *ChopAssoc FiniteChopSkipEqvSkipChopFinite NextChop*
NextDiamondImpDiamond NowImpDiamond inteq-reflection lift-imp-trans next-d-def
sometimes-d-def)

lemma *DiamondIntro*:

assumes $\vdash (f \wedge \neg g) \longrightarrow \bigcirc f$

shows $\vdash f \wedge \text{finite} \longrightarrow \Diamond g$

proof –

have 1: $\vdash f \wedge \neg g \longrightarrow \bigcirc f$

using *assms* **by** *auto*

hence 2: $\vdash f \wedge \neg g \wedge (\Box (\neg g)) \longrightarrow (\bigcirc f) \wedge (\Box (\neg g))$

by *auto*

have 3: $\vdash (\Box (\neg g)) \longrightarrow \neg g$

by (*rule* *BoxElim*)

hence 4: $\vdash \Box (\neg g) = ((\Box (\neg g)) \wedge \neg g)$

using *BoxImpBoxBox BoxBoxImpBox* **by** *fastforce*

have 5: $\vdash f \wedge (\Box (\neg g)) \longrightarrow \bigcirc f \wedge \Box (\neg g)$

using 2 4 **by** *fastforce*

have 6: $\vdash \Box (\neg g) = ((\neg g) \wedge \text{wnext}(\Box (\neg g)))$

using *BoxEqvAndWnextBox* **by** *metis*

have 7: $\vdash \bigcirc f \wedge \Box (\neg g) \longrightarrow \bigcirc f \wedge \text{wnext}(\Box (\neg g))$

using 6 **by** *auto*

have 8: $\vdash f \wedge (\Box (\neg g)) \longrightarrow \bigcirc f \wedge \text{wnext}(\Box (\neg g))$

using 5 7 **using** *lift-imp-trans* **by** *blast*

hence 9: $\vdash f \wedge (\Box (\neg g)) \longrightarrow \text{more} \wedge \text{wnext } f \wedge \text{wnext}(\Box (\neg g))$

using *zero-enat-def* **by** (*auto* *simp*: *Valid-def itl-defs*)

hence 10: $\vdash f \wedge (\Box (\neg g)) \longrightarrow \text{wnext } f \wedge \text{wnext}(\Box (\neg g))$

by *auto*

hence 11: $\vdash f \wedge (\Box (\neg g)) \longrightarrow \text{wnext } (f \wedge \Box (\neg g))$

by (*auto* *simp*: *Valid-def itl-defs*)

hence 12: $\vdash \Box (f \wedge (\Box (\neg g))) \longrightarrow \text{wnext } (f \wedge \Box (\neg g))$

by (*rule* *BoxGen*)

have 13: $\vdash \Box (f \wedge (\Box (\neg g))) \longrightarrow \text{wnext } (f \wedge \Box (\neg g)) \wedge f \wedge (\Box (\neg g)) \longrightarrow \Box (f \wedge (\Box (\neg g)))$

by (*rule* *BoxInduct*)

hence 14: $\vdash \Box (f \wedge (\Box (\neg g))) \longrightarrow \text{wnext } (f \wedge \Box (\neg g)) \longrightarrow ((f \wedge (\Box (\neg g))) \longrightarrow \Box (f \wedge (\Box (\neg g))))$

by *fastforce*

have 15: $\vdash ((f \wedge (\Box (\neg g))) \longrightarrow \Box (f \wedge (\Box (\neg g))))$

using 12 14 *MP* **by** *blast*

have 16: $\vdash \Box (f \wedge (\Box (\neg g))) \longrightarrow (f \wedge (\Box (\neg g)))$

by (*rule* *BoxElim*)

have 17: $\vdash \Box (f \wedge (\Box (\neg g))) = (f \wedge (\Box (\neg g)))$

using 16 15 **by** *fastforce*

have 18: $\vdash (f \wedge (\Box (\neg g))) \longrightarrow \text{more}$

using 9 **by** *auto*

hence 19: $\vdash \Box(f \wedge (\Box(\neg g))) \longrightarrow \Box \text{ more}$
by (*rule ImpBoxRule*)
have 20: $\vdash \text{finite} = (\neg(\Box \text{ more}))$
by (*metis DiamondEmptyEqvFinite InfEqvNotFinite always-d-def empty-d-def finite-d-def int-eq*)
have 21: $\vdash \text{finite} \longrightarrow \neg(f \wedge (\Box(\neg g)))$
using 17 19 20 by *fastforce*
hence 22: $\vdash \text{finite} \longrightarrow \neg f \vee \neg(\Box(\neg g))$
by *auto*
have 23: $\vdash (\neg(\Box(\neg g))) = \Diamond g$
by (*auto simp: always-d-def*)
from 22 23 show ?thesis by *fastforce*
qed

lemma *DiamondIntroB:*

assumes $\vdash (f \wedge \neg g) \longrightarrow \bigcirc(f \wedge \neg g)$
shows $\vdash f \wedge \text{finite} \longrightarrow \Diamond g$
proof –
have 1: $\vdash (f \wedge \neg g) \longrightarrow \bigcirc(f \wedge \neg g)$ **using** *assms* **by** *auto*
hence 2: $\vdash \text{finite} \longrightarrow \neg(f \wedge \neg g)$ **by** (*rule NextLoop*)
hence 3: $\vdash f \wedge \text{finite} \longrightarrow g$ **by** *auto*
have 4: $\vdash g \longrightarrow \Diamond g$ **by** (*rule NowImpDiamond*)
from 3 4 show ?thesis using *lift-imp-trans* **by** *blast*
qed

lemma *NextContra :*

assumes $\vdash (f \wedge \neg g) \longrightarrow (\bigcirc f \wedge \neg(\bigcirc g))$
shows $\vdash f \wedge \text{finite} \longrightarrow g$
proof –
have 1: $\vdash (f \wedge \neg g) \longrightarrow (\bigcirc f \wedge \neg(\bigcirc g))$ **using** *assms* **by** *auto*
hence 2: $\vdash \neg(f \longrightarrow g) \longrightarrow \bigcirc(\neg(f \longrightarrow g))$ **by** (*auto simp: itl-defs Valid-def*)
hence 3: $\vdash \text{finite} \longrightarrow \neg\neg(f \longrightarrow g)$ **by** (*rule NextLoop*)
from 3 show ?thesis by *auto*
qed

lemma *DiamondDiamondEqvDiamond:*

$\vdash \Diamond(\Diamond f) = \Diamond f$
proof –
have 1: $\vdash \text{finite}; \text{finite} = \text{finite}$ **by** (*simp add: FiniteChopFiniteEqvFinite*)
hence 2: $\vdash (\text{finite}; \text{finite}); f = \text{finite}; f$ **using** *LeftChopEqvChop* **by** *blast*
have 3: $\vdash (\text{finite}; \text{finite}); f = \text{finite}; (\text{finite}; f)$ **using** *ChopAssoc* **by** *fastforce*
from 2 3 show ?thesis by (*metis inteq-reflection sometimes-d-def*)
qed

lemma *WeakNextDiamondInduct:*

assumes $\vdash \text{wnext } (\Diamond f) \longrightarrow f$
shows $\vdash \text{finite} \longrightarrow f$
proof –
have 1: $\vdash \text{wnext } (\Diamond f) \longrightarrow f$ **using** *assms* **by** *blast*
hence 2: $\vdash \neg f \longrightarrow \neg(\text{wnext } (\Diamond f))$ **by** *fastforce*
hence 3: $\vdash \neg f \longrightarrow \bigcirc(\neg(\Diamond f))$ **by** (*simp add: wnext-d-def*)

have 4: $\vdash f \longrightarrow \Diamond f$ **by** (rule *NowImpDiamond*)
hence 5: $\vdash \neg(\Diamond f) \longrightarrow \neg f$ **by** *auto*
have 6: $\vdash \neg f \longrightarrow \bigcirc(\neg f)$ **using** 3 5 **using** *NextImpNext lift-imp-trans* **by** *blast*
hence 7: $\vdash \text{finite} \longrightarrow \neg\neg f$ **by** (rule *NextLoop*)
from 7 **show** ?thesis **by** *auto*
qed

lemma *EmptyNextInducta*:

assumes $\vdash \text{empty} \longrightarrow f$
 $\vdash \bigcirc f \longrightarrow f$
shows $\vdash \text{finite} \longrightarrow f$
proof –
have 1: $\vdash \text{empty} \longrightarrow f$ **using** *assms* **by** *auto*
have 2: $\vdash \bigcirc f \longrightarrow f$ **using** *assms* **by** *blast*
have 3: $\vdash (\text{empty} \vee \bigcirc f) \longrightarrow f$ **using** 1 2 **by** *fastforce*
have 4: $\vdash \text{wnext } f = (\text{empty} \vee \bigcirc f)$ **by** (rule *WnextEqvEmptyOrNext*)
hence 5: $\vdash \text{wnext } f \longrightarrow f$ **using** 3 **by** *fastforce*
hence 6: $\vdash \neg f \longrightarrow \neg(\text{wnext } f)$ **by** *auto*
hence 7: $\vdash \neg f \longrightarrow \bigcirc(\neg f)$ **by** (*auto simp: wnext-d-def*)
hence 8: $\vdash \text{finite} \longrightarrow \neg\neg f$ **by** (rule *NextLoop*)
from 8 **show** ?thesis **by** *auto*
qed

lemma *EmptyNextInductb*:

assumes $\vdash \text{empty} \wedge f \longrightarrow g$
 $\vdash \bigcirc(f \longrightarrow g) \wedge f \longrightarrow g$
shows $\vdash f \wedge \text{finite} \longrightarrow g$
proof –
have 1: $\vdash \text{empty} \wedge f \longrightarrow g$ **using** *assms* **by** *auto*
have 2: $\vdash \bigcirc(f \longrightarrow g) \wedge f \longrightarrow g$ **using** *assms* **by** *blast*
have 3: $\vdash (\text{empty} \vee \bigcirc(f \longrightarrow g)) \wedge f \longrightarrow g$ **using** 1 2 **by** *fastforce*
hence 4: $\vdash \text{wnext } (f \longrightarrow g) \wedge f \longrightarrow g$ **using** *WnextEqvEmptyOrNext* **by** *fastforce*
hence 5: $\vdash \text{wnext } (f \longrightarrow g) \longrightarrow (f \longrightarrow g)$ **by** *fastforce*
hence 6: $\vdash \neg(f \longrightarrow g) \longrightarrow \neg(\text{wnext } (f \longrightarrow g))$ **by** *fastforce*
hence 7: $\vdash \neg(f \longrightarrow g) \longrightarrow \bigcirc(\neg(f \longrightarrow g))$ **by** (*simp add: wnext-d-def*)
hence 8: $\vdash \text{finite} \longrightarrow \neg\neg(f \longrightarrow g)$ **by** (rule *NextLoop*)
from 8 **show** ?thesis **by** *auto*
qed

lemma *FinImpFin*:

assumes $\vdash f \longrightarrow g$
shows $\vdash \text{fin } f \longrightarrow \text{fin } g$
using *ImpBoxRule*[of *LIFT* ($\text{empty} \longrightarrow f$) *LIFT* ($\text{empty} \longrightarrow g$)] *assms*
 fin-d-def [of *f*] fin-d-def [of *g*] **by** *fastforce*

lemma *FinEqvFin*:

assumes $\vdash f = g$
shows $\vdash \text{fin } f = \text{fin } g$
using *assms* **by** (*simp add: FinImpFin Prop11*)

lemma *FinAndFinImpFinRule*:

assumes $\vdash f \wedge g \longrightarrow h$

shows $\vdash \text{fin } f \wedge \text{fin } g \longrightarrow \text{fin } h$

using *assms* **by** (*auto simp add: itl-defs Valid-def*)

lemma *FinAndFinEqvFinRule*:

assumes $\vdash (f \wedge g) = h$

shows $\vdash (\text{fin } f \wedge \text{fin } g) = \text{fin } h$

using *assms*

by (*simp add: FinAndFinImpFinRule FinImpFin Prop11 Prop12*)

lemma *HaltEqvHalt*:

assumes $\vdash f = g$

shows $\vdash \text{halt } f = \text{halt } g$

proof –

have 1: $\vdash f = g$ **using** *assms* **by** *auto*

hence 2: $\vdash (\text{empty} = f) = (\text{empty} = g)$ **by** *auto*

hence 3: $\vdash \Box(\text{empty} = f) = \Box(\text{empty} = g)$ **by** (*rule BoxEqvBox*)

from 3 **show** *?thesis* **by** (*simp add: halt-d-def*)

qed

lemma *BiImpDiImpDi*:

$\vdash \text{bi } (f \longrightarrow g) \longrightarrow \text{di } f \longrightarrow \text{di } g$

proof –

have 1: $\vdash \text{bi } (f \longrightarrow g) \longrightarrow (f; \# \text{True}) \longrightarrow (g; \# \text{True})$ **by** (*rule BiChopImpChop*)

from 1 **show** *?thesis* **by** (*simp add: di-d-def*)

qed

lemma *DiImpDi*:

assumes $\vdash f \longrightarrow g$

shows $\vdash \text{di } f \longrightarrow \text{di } g$

proof –

have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*

hence 2: $\vdash f; \# \text{True} \longrightarrow g; \# \text{True}$ **by** (*rule LeftChopImpChop*)

from 2 **show** *?thesis* **by** (*simp add: di-d-def*)

qed

lemma *BiImpBiRule*:

assumes $\vdash f \longrightarrow g$

shows $\vdash \text{bi } f \longrightarrow \text{bi } g$

proof –

have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*

hence 2: $\vdash \neg g \longrightarrow \neg f$ **by** *auto*

hence 3: $\vdash \text{di } (\neg g) \longrightarrow \text{di } (\neg f)$ **by** (*rule DiImpDi*)

hence 4: $\vdash \neg (\text{di } (\neg f)) \longrightarrow \neg (\text{di } (\neg g))$ **by** *auto*

from 4 **show** *?thesis* **by** (*simp add: bi-d-def*)

qed

lemma *DiEqvDi*:

assumes $\vdash f = g$
shows $\vdash di\ f = di\ g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash f; \#True = g; \#True$ **by** (rule *LeftChopEqvChop*)
from 2 **show** *?thesis* **by** (simp add: *di-d-def*)
qed

lemma *BiEqvBi*:
assumes $\vdash f = g$
shows $\vdash bi\ f = bi\ g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash (\neg f) = (\neg g)$ **by** *auto*
hence 3: $\vdash di\ (\neg f) = di\ (\neg g)$ **by** (rule *DiEqvDi*)
hence 4: $\vdash (\neg (di\ (\neg f))) = (\neg (di\ (\neg g)))$ **by** *auto*
from 4 **show** *?thesis* **by** (simp add: *bi-d-def*)
qed

lemma *LeftChopChopImpChopRule*:
assumes $\vdash (f; g) \longrightarrow g$
shows $\vdash (f; g); h \longrightarrow (g; h)$
proof –
have 1: $\vdash (f; g) \longrightarrow g$ **using** *assms* **by** *blast*
hence 2: $\vdash (f; g); h \longrightarrow g; h$ **by** (rule *LeftChopImpChop*)
have 3: $\vdash f; (g; h) = (f; g); h$ **by** (rule *ChopAssoc*)
from 2 3 **show** *?thesis* **by** *auto*
qed

lemma *AndChopCommute* :
 $\vdash (f \wedge f1); g = (f1 \wedge f); g$
proof –
have 1: $\vdash (f \wedge f1) = (f1 \wedge f)$ **by** *auto*
from 1 **show** *?thesis* **by** (rule *LeftChopEqvChop*)
qed

lemma *BiAndChopImport*:
 $\vdash bi\ f \wedge (f1; g) \longrightarrow (f \wedge f1); g$
proof –
have 1: $\vdash f \longrightarrow (f1 \longrightarrow f \wedge f1)$ **by** *auto*
hence 2: $\vdash bi\ f \longrightarrow bi\ (f1 \longrightarrow f \wedge f1)$ **by** (rule *BiImpBiRule*)
have 3: $\vdash bi\ (f1 \longrightarrow (f \wedge f1)) \longrightarrow f1; g \longrightarrow (f \wedge f1); g$ **by** (rule *BiChopImpChop*)
from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *StateAndChopImport*:
 $\vdash (init\ w) \wedge (f; g) \longrightarrow ((init\ w) \wedge f); g$
proof –
have 1: $\vdash (init\ w) \longrightarrow bi\ (init\ w)$ **by** (rule *StateImpBi*)
hence 2: $\vdash (init\ w) \wedge (f; g) \longrightarrow bi\ (init\ w) \wedge (f; g)$ **by** *auto*

have 3: $\vdash bi \ (init \ w) \wedge (f; g) \longrightarrow ((init \ w) \wedge f); g$ **by** (rule *BiAndChopImport*)
from 2 3 **show** ?thesis **using** MP **by** fastforce
qed

2.3.5 Further Properties Di and Bi

lemma *ImpDi*:

$\vdash f \longrightarrow di \ f$
proof –
have 1: $\vdash f; empty = f$ **by** (rule *ChopEmpty*)
have 2: $\vdash empty \longrightarrow \#True$ **by** auto
hence 3: $\vdash f; empty \longrightarrow f; \#True$ **by** (rule *RightChopImpChop*)
have 4: $\vdash f \longrightarrow f; \#True$ **using** 1 3 **by** fastforce
from 4 **show** ?thesis **by** (simp add: di-d-def)
qed

lemma *DiState*:

$\vdash di \ (init \ w) = (init \ w)$
proof –
have 0: $\vdash (init \ (\neg w)) \longrightarrow bi \ (init \ (\neg w))$ **using** *StateImpBi* **by** fastforce
hence 1: $\vdash \neg(init \ w) \longrightarrow bi \ (\neg \ (init \ w))$ **using** *Initprop(2)* **by** (metis *inteq-reflection*)
hence 2: $\vdash (\neg \ (init \ w)) \longrightarrow \neg \ (di \ (\neg \neg \ (init \ w)))$ **by** (simp add: bi-d-def)
have 3: $\vdash (\neg \ (init \ w) \longrightarrow \neg \ (di \ (\neg \neg \ (init \ w)))) \longrightarrow$
 $(di \ (\neg \neg \ (init \ w)) \longrightarrow (init \ w))$ **by** auto
have 4: $\vdash di \ (\neg \neg \ (init \ w)) \longrightarrow (init \ w)$ **using** 2 3 MP **by** blast
have 5: $\vdash (init \ w) \longrightarrow \neg \neg \ (init \ w)$ **by** auto
hence 6: $\vdash di \ (init \ w) \longrightarrow di \ (\neg \neg \ (init \ w))$ **by** (rule *DiImpDi*)
have 7: $\vdash di \ (init \ w) \longrightarrow (init \ w)$ **using** 6 4 **using** *lift-imp-trans* **by** metis
have 8: $\vdash (init \ w) \longrightarrow di \ (init \ w)$ **by** (rule *ImpDi*)
from 7 8 **show** ?thesis **by** fastforce
qed

lemma *StateChop*:

$\vdash (init \ w); f \longrightarrow (init \ w)$
by (metis *ChopAssoc Prop12 RightChopImpChop TrueW init-d-def int-simps(13) int-simps(17) inteq-reflection*)

lemma *StateChopExportA*:

$\vdash ((init \ w) \wedge f); g \longrightarrow (init \ w)$
using *DiState AndChopA StateChop* **by** fastforce

lemma *StateAndChop*:

$\vdash ((init \ w) \wedge f); g = ((init \ w) \wedge (f; g))$
by (simp add: *AndChopB StateAndChopImport StateChopExportA Prop11 Prop12*)

lemma *StateAndChopImpChopRule*:

assumes $\vdash (init \ w) \wedge f \longrightarrow f1$
shows $\vdash (init \ w) \wedge (f; g) \longrightarrow (f1; g)$
proof –
have 1: $\vdash (init \ w) \wedge f \longrightarrow f1$ **using** *assms* **by** auto

hence 2: $\vdash ((init\ w) \wedge f); g \longrightarrow f1; g$ **by** (rule *LeftChopImpChop*)
have 3: $\vdash ((init\ w) \wedge f); g = ((init\ w) \wedge (f; g))$ **by** (rule *StateAndChop*)
from 2 3 **show** ?thesis **by** fastforce
qed

lemma *StateImpChopEqvChop* :
assumes $\vdash (init\ w) \longrightarrow (f = f1)$
shows $\vdash (init\ w) \longrightarrow ((f; g) = (f1; g))$
proof –
have 1: $\vdash (init\ w) \longrightarrow (f = f1)$ **using** *assms* **by** *auto*
hence 2: $\vdash (init\ w) \wedge f \longrightarrow f1$ **by** *auto*
hence 3: $\vdash (init\ w) \wedge (f; g) \longrightarrow (f1; g)$ **by** (rule *StateAndChopImpChopRule*)
have 4: $\vdash (init\ w) \wedge f1 \longrightarrow f$ **using** 1 **by** *auto*
hence 5: $\vdash (init\ w) \wedge (f1; g) \longrightarrow (f; g)$ **by** (rule *StateAndChopImpChopRule*)
from 3 5 **show** ?thesis **by** fastforce
qed

lemma *ChopEqvStateAndChop*:
assumes $\vdash f = (init\ w) \wedge f1$
shows $\vdash (f; g) = ((init\ w) \wedge (f1; g))$
proof –
have 1: $\vdash f = ((init\ w) \wedge f1)$ **using** *assms* **by** *auto*
hence 2: $\vdash f; g = (((init\ w) \wedge f1); g)$ **by** (rule *LeftChopEqvChop*)
have 3: $\vdash ((init\ w) \wedge f1); g = ((init\ w) \wedge (f1; g))$ **by** (rule *StateAndChop*)
from 2 3 **show** ?thesis **by** fastforce
qed

lemma *DiIntro*:
 $\vdash f \longrightarrow di\ f$
proof –
have 1: $\vdash f; empty = f$ **by** (rule *ChopEmpty*)
have 2: $\vdash empty \longrightarrow \#True$ **by** *auto*
hence 3: $\vdash \Box(empty \longrightarrow \#True)$ **by** (rule *BoxGen*)
have 4: $\vdash \Box(empty \longrightarrow \#True) \longrightarrow (f; empty \longrightarrow f; \#True)$ **by** (rule *BoxChopImpChop*)
have 5: $\vdash f; empty \longrightarrow f; \#True$ **using** 3 4 *MP* **by** fastforce
hence 6: $\vdash f; empty \longrightarrow di\ f$ **by** (*simp* *add: di-d-def*)
from 1 6 **show** ?thesis **by** fastforce
qed

lemma *BiElim*:
 $\vdash bi\ f \longrightarrow f$
proof –
have 1: $\vdash \neg f \longrightarrow di\ (\neg f)$ **by** (rule *DiIntro*)
have 2: $\vdash (\neg f \longrightarrow di\ (\neg f)) \longrightarrow (\neg(di\ (\neg f)) \longrightarrow f)$ **by** *auto*
have 3: $\vdash \neg (di\ (\neg f)) \longrightarrow f$ **using** 1 2 *MP* **by** *blast*
from 3 **show** ?thesis **by** (*metis* *bi-d-def*)
qed

lemma *BiContraPosImpDist*:
 $\vdash bi\ (\neg g \longrightarrow \neg f) \longrightarrow (bi\ f) \longrightarrow (bi\ g)$

proof –

have 1: $\vdash bi (\neg g \longrightarrow \neg f) \longrightarrow (di (\neg g)) \longrightarrow (di (\neg f))$ **by** (rule BiImpDiImpDi)

hence 2: $\vdash bi (\neg g \longrightarrow \neg f) \longrightarrow (\neg (di (\neg f))) \longrightarrow (\neg (di (\neg g)))$ **by** auto

from 2 **show** ?thesis **by** (metis bi-d-def)

qed

lemma BiImpDist:

$\vdash bi (f \longrightarrow g) \longrightarrow (bi f) \longrightarrow (bi g)$

proof –

have 1: $\vdash (f \longrightarrow g) \longrightarrow (\neg g \longrightarrow \neg f)$ **by** auto

hence 2: $\vdash \neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g)$ **by** auto

hence 3: $\vdash bi (\neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g))$ **by** (rule BiGen)

have 4: $\vdash bi (\neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g))$

\longrightarrow

$bi (f \longrightarrow g) \longrightarrow bi (\neg g \longrightarrow \neg f)$ **by** (rule BiContraPosImpDist)

have 5: $\vdash bi (f \longrightarrow g) \longrightarrow bi (\neg g \longrightarrow \neg f)$ **using** 3 4 **MP** **by** blast

have 6: $\vdash bi (\neg g \longrightarrow \neg f) \longrightarrow (bi f) \longrightarrow (bi g)$ **by** (rule BiContraPosImpDist)

from 5 6 **show** ?thesis **using** lift-imp-trans **by** blast

qed

lemma IfChopEqvRule:

assumes $\vdash f = if_i (init w) \text{ then } f1 \text{ else } f2$

shows $\vdash f; g = if_i (init w) \text{ then } (f1; g) \text{ else } (f2; g)$

proof –

have 1: $\vdash f = if_i (init w) \text{ then } f1 \text{ else } f2$

using assms **by** auto

hence 2: $\vdash f = (((init w) \wedge f1) \vee ((init (\neg w)) \wedge f2))$

by (metis Initprop(2) ifthenelse-d-def inteq-reflection)

hence 3: $\vdash f; g = (((init w) \wedge f1); g \vee ((init (\neg w)) \wedge f2); g)$

by (rule OrChopEqvRule)

have 4: $\vdash ((init w) \wedge f1); g = ((init w) \wedge (f1; g))$

by (rule StateAndChop)

have 5: $\vdash ((init (\neg w)) \wedge f2); g = ((init (\neg w)) \wedge (f2; g))$

by (rule StateAndChop)

have 6: $\vdash f; g = (((init w) \wedge f1; g) \vee ((init (\neg w)) \wedge f2; g))$

using 3 4 5 **by** fastforce

from 6 **show** ?thesis

by (metis Initprop(2) ifthenelse-d-def inteq-reflection)

qed

lemma ChopOrEqvRule:

assumes $\vdash g = (g1 \vee g2)$

shows $\vdash f; g = (f; g1) \vee (f; g2)$

proof –

have 1: $\vdash g = (g1 \vee g2)$ **using** assms **by** auto

hence 2: $\vdash f; g = (f; (g1 \vee g2))$ **by** (rule RightChopEqvChop)

have 3: $\vdash f; (g1 \vee g2) = (f; g1 \vee f; g2)$ **by** (rule ChopOrEqv)

from 2 3 **show** ?thesis **by** fastforce

qed

lemma *EmptyOrChopEqv*:

$\vdash (\text{empty} \vee f); g = (g \vee (f; g))$

proof –

have 1: $\vdash (\text{empty} \vee f); g = ((\text{empty}; g) \vee (f; g))$ **by** (rule *OrChopEqv*)

have 2: $\vdash \text{empty}; g = g$ **by** (rule *EmptyChop*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *EmptyOrNextChopEqv*:

$\vdash (\text{empty} \vee \circ f); g = (g \vee \circ(f; g))$

proof –

have 1: $\vdash (\text{empty} \vee \circ f); g = (g \vee ((\circ f); g))$ **by** (rule *EmptyOrChopEqv*)

have 2: $\vdash (\circ f); g = \circ(f; g)$ **by** (rule *NextChop*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *EmptyOrChopImpRule*:

assumes $\vdash f \longrightarrow \text{empty} \vee f1$

shows $\vdash f; g \longrightarrow g \vee (f1; g)$

proof –

have 1: $\vdash f \longrightarrow \text{empty} \vee f1$ **using** assms **by** auto

hence 2: $\vdash f; g \longrightarrow (\text{empty} \vee f1); g$ **by** (rule *LeftChopImpChop*)

have 3: $\vdash (\text{empty} \vee f1); g = (g \vee (f1; g))$ **by** (rule *EmptyOrChopEqv*)

from 2 3 **show** ?thesis **by** fastforce

qed

lemma *EmptyOrChopEqvRule*:

assumes $\vdash f = (\text{empty} \vee f1)$

shows $\vdash f; g = (g \vee (f1; g))$

proof –

have 1: $\vdash f = (\text{empty} \vee f1)$ **using** assms **by** auto

hence 2: $\vdash f; g = ((\text{empty} \vee f1); g)$ **by** (rule *LeftChopEqvChop*)

have 3: $\vdash (\text{empty} \vee f1); g = (g \vee (f1; g))$ **by** (rule *EmptyOrChopEqv*)

from 2 3 **show** ?thesis **by** fastforce

qed

lemma *EmptyOrNextChopImpRule*:

assumes $\vdash f \longrightarrow \text{empty} \vee \circ f1$

shows $\vdash f; g \longrightarrow g \vee \circ(f1; g)$

proof –

have 1: $\vdash f \longrightarrow \text{empty} \vee \circ f1$ **using** assms **by** auto

hence 2: $\vdash f; g \longrightarrow (\text{empty} \vee \circ f1); g$ **by** (rule *LeftChopImpChop*)

have 3: $\vdash (\text{empty} \vee \circ f1); g = (g \vee \circ(f1; g))$ **by** (rule *EmptyOrNextChopEqv*)

from 2 3 **show** ?thesis **by** fastforce

qed

lemma *EmptyOrNextChopEqvRule*:

assumes $\vdash f = (\text{empty} \vee \circ f1)$

shows $\vdash f; g = (g \vee \circ(f1; g))$

proof –

have 1: $\vdash f = (\text{empty} \vee \circ f1)$ **using** *assms* **by** *auto*
hence 2: $\vdash f; g = ((\text{empty} \vee \circ f1); g)$ **by** (rule *LeftChopEqvChop*)
have 3: $\vdash (\text{empty} \vee \circ f1); g = (g \vee \circ(f1; g))$ **by** (rule *EmptyOrNextChopEqv*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *ChopEmptyOrImpRule*:

assumes $\vdash g \longrightarrow \text{empty} \vee g1$
shows $\vdash f; g \longrightarrow f \vee (f; g1)$
proof –
have 1: $\vdash g \longrightarrow \text{empty} \vee g1$ **using** *assms* **by** *auto*
hence 2: $\vdash f; g \longrightarrow (f; \text{empty}) \vee (f; g1)$ **by** (rule *ChopOrImpRule*)
have 3: $\vdash f; \text{empty} = f$ **by** (rule *ChopEmpty*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *StateAndEmptyImpBoxState*:

$\vdash (\text{init } w) \wedge \text{empty} \longrightarrow \Box (\text{init } w)$
using *BoxEqvAndEmptyOrNextBox* **by** *fastforce*

lemma *BoxEqvAndBox*:

$\vdash \Box f = (f \wedge \Box f)$
using *BoxElim* **by** *fastforce*

lemma *NotBoxImpNotOrNotNextBox*:

$\vdash \neg(\Box f) \longrightarrow \neg f \vee \neg(\circ(\Box f))$
proof –
have 1: $\vdash f \wedge (\circ(\Box f)) \longrightarrow \Box f$ **using** *BoxEqvAndEmptyOrNextBox* **by** *fastforce*
hence 2: $\vdash \neg(\Box f) \longrightarrow \neg(f \wedge (\circ(\Box f)))$ **by** *fastforce*
have 3: $\vdash (\neg(f \wedge (\circ(\Box f)))) = (\neg f \vee \neg(\circ(\Box f)))$ **by** *auto*
from 2 3 **show** *?thesis* **by** *auto*
qed

lemma *BoxStateChopBoxAndInfImpBox*:

$\vdash \Box (\text{init } w); \Box (\text{init } w) \wedge \text{inf} \longrightarrow \Box (\text{init } w)$
by (*auto simp add: Valid-def itl-defs nfirst-eq-nnth-zero*)
(metis enat-ile le-add-diff-inverse linorder-le-cases min-def ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD ntaken-nnth)

lemma *BoxStateChopBoxEqvBox*:

$\vdash \Box (\text{init } w); \Box (\text{init } w) = \Box (\text{init } w)$
proof –
have 1: $\vdash (\Box (\text{init } w)) = ((\text{init } w) \wedge (\text{empty} \vee \circ(\Box (\text{init } w))))$
by (rule *BoxEqvAndEmptyOrNextBox*)
hence 2: $\vdash (\Box (\text{init } w); \Box (\text{init } w)) =$
 $((\text{init } w) \wedge ((\text{empty} \vee \circ(\Box (\text{init } w))); \Box (\text{init } w)))$
by (*metis StateAndChop inteq-reflection*)
have 3: $\vdash ((\text{empty} \vee \circ(\Box (\text{init } w))); \Box (\text{init } w)) =$

$(\Box (init\ w) \vee \bigcirc(\Box (init\ w); \Box (init\ w)))$
by (rule *EmptyOrNextChopEqv*)
have 4: $\vdash (\Box (init\ w); \Box (init\ w)) =$
 $((init\ w) \wedge (\Box (init\ w) \vee \bigcirc(\Box (init\ w); \Box (init\ w))))$
using 2 3 **by** *fastforce*
have 5: $\vdash \neg (\Box (init\ w)) \longrightarrow \neg (init\ w) \vee \neg(\bigcirc(\Box (init\ w)))$
by (rule *NotBoxImpNotOrNotNextBox*)
have 6: $\vdash (\Box (init\ w); \Box (init\ w)) \wedge \neg(\Box (init\ w)) \longrightarrow$
 $\bigcirc(\Box (init\ w); \Box (init\ w)) \wedge \neg(\bigcirc(\Box (init\ w)))$
using 4 5 **by** *fastforce*
hence 7: $\vdash \Box (init\ w); \Box (init\ w) \wedge finite \longrightarrow \Box (init\ w)$
by (rule *NextContra*)
have 8: $\vdash \Box (init\ w); \Box (init\ w) \wedge inf \longrightarrow \Box (init\ w)$
by (rule *BoxStateChopBoxAndInfImpBox*)
have 9: $\vdash \Box (init\ w); \Box (init\ w) \wedge (finite \vee inf) \longrightarrow \Box (init\ w)$
using 7 8 **by** *fastforce*
hence 10: $\vdash \Box (init\ w); \Box (init\ w) \longrightarrow \Box (init\ w)$
using *FiniteOrInfinite* **by** *fastforce*
have 11: $\vdash \Box (init\ w) = ((init\ w) \wedge \Box (init\ w))$
by (rule *BoxEqvAndBox*)
have 12: $\vdash empty ; \Box (init\ w) = \Box (init\ w)$
by (rule *EmptyChop*)
have 13: $\vdash ((init\ w) \wedge empty) ; \Box (init\ w) = ((init\ w) \wedge (empty ; \Box (init\ w)))$
by (rule *StateAndChop*)
have 14: $\vdash \Box (init\ w) = ((init\ w) \wedge empty) ; \Box (init\ w)$
using 11 12 13 **by** *fastforce*
have 15: $\vdash (init\ w) \wedge empty \longrightarrow \Box (init\ w)$
by (rule *StateAndEmptyImpBoxState*)
hence 16: $\vdash ((init\ w) \wedge empty) ; \Box (init\ w) \longrightarrow \Box (init\ w); \Box (init\ w)$
by (rule *LeftChopImpChop*)
have 17: $\vdash \Box (init\ w) \longrightarrow \Box (init\ w); \Box (init\ w)$
using 14 16 **by** *fastforce*
from 10 17 **show** *?thesis* **by** *fastforce*
qed

lemma *NotBoxStateImpBoxYieldsNotBox*:

$\vdash \neg(\Box (init\ w)) \longrightarrow (\Box (init\ w))\ yields\ (\neg(\Box (init\ w)))$

proof –

have 1: $\vdash \Box (init\ w); \Box (init\ w) = \Box (init\ w)$ **by** (rule *BoxStateChopBoxEqvBox*)

have 2: $\vdash \Box (init\ w) = (\neg \neg(\Box (init\ w)))$ **by** *auto*

hence 3: $\vdash \Box (init\ w); \Box (init\ w) = \Box (init\ w); (\neg \neg(\Box (init\ w)))$ **by** (rule *RightChopEqvChop*)

have 4: $\vdash \neg(\Box (init\ w)) \longrightarrow \neg(\Box (init\ w); (\neg \neg(\Box (init\ w))))$ **using** 1 3 **by** *auto*

from 4 **show** *?thesis* **by** (*simp add: yields-d-def*)

qed

lemma *StateEqvBi*:

$\vdash (init\ w) = bi\ (init\ w)$

proof –

have 1: $\vdash (init\ w) \longrightarrow bi\ (init\ w)$ **by** (rule *StateImpBi*)

have 2: $\vdash bi\ (init\ w) \longrightarrow (init\ w)$ **by** (rule *BiElim*)

from 1 2 show ?thesis by fastforce
qed

lemma *FiniteChopEqvDiamond*:
 $\vdash \text{finite}; f = \Diamond f$
 by (simp add: sometimes-d-def)

2.3.6 Properties of Da and Ba

lemma *DaEqvDtDi*:
 $\vdash da\ f = \Diamond (di\ f)$
 proof –
 have 1: $\vdash \text{finite}; (f; \#True) = \text{finite}; (f; \#True)$ by auto
 hence 2: $\vdash \text{finite}; (f; \#True) = \text{finite}; di\ f$ by (simp add: di-d-def)
 have 3: $\vdash \text{finite}; di\ f = \Diamond (di\ f)$ by (rule *FiniteChopEqvDiamond*)
 have 4: $\vdash \text{finite}; (f; \#True) = \Diamond (di\ f)$ using 2 3 by fastforce
 from 4 show ?thesis by (simp add: da-d-def)
 qed

lemma *DaEqvDiDt*:
 $\vdash da\ f = di\ (\Diamond f)$
 proof –
 have 1: $\vdash \text{finite}; f = \Diamond f$ by (rule *FiniteChopEqvDiamond*)
 hence 2: $\vdash (\text{finite}; f); \#True = (\Diamond f); \#True$ by (rule *LeftChopEqvChop*)
 hence 3: $\vdash (\text{finite}; f); \#True = di\ (\Diamond f)$ by (simp add: di-d-def)
 have 4: $\vdash \text{finite}; (f; \#True) = (\text{finite}; f); \#True$ by (rule *ChopAssoc*)
 have 5: $\vdash \text{finite}; (f; \#True) = di\ (\Diamond f)$ using 3 4 by fastforce
 from 5 show ?thesis by (simp add: da-d-def)
 qed

lemma *DtDiEqvDiDt*:
 $\vdash \Diamond (di\ f) = di\ (\Diamond f)$
 by (metis *ChopAssoc* di-d-def sometimes-d-def)

lemma *DiamondNotEqvNotBox*:
 $\vdash \Diamond (\neg f) = (\neg (\Box f))$
 by (simp add: always-d-def)

lemma *BaEqvBiBt*:
 $\vdash ba\ f = bi\ (\Box f)$
 proof –
 have 1: $\vdash da\ (\neg f) = di\ (\Diamond (\neg f))$ by (rule *DaEqvDiDt*)
 have 2: $\vdash \Diamond (\neg f) = (\neg (\Box f))$ by (rule *DiamondNotEqvNotBox*)
 hence 3: $\vdash di\ (\Diamond (\neg f)) = di\ (\neg (\Box f))$ by (rule *DiEqvDi*)
 have 4: $\vdash da\ (\neg f) = di\ (\neg (\Box f))$ using 1 3 by fastforce
 hence 5: $\vdash (\neg (da\ (\neg f))) = (\neg (di\ (\neg (\Box f))))$ by auto
 hence 6: $\vdash (\neg (da\ (\neg f))) = bi\ (\Box f)$ by (simp add: bi-d-def)
 from 6 show ?thesis by (simp add: ba-d-def)
 qed

lemma *DiNotEqvNotBi*:

$\vdash \text{di } (\neg f) = (\neg (bi \ f))$

proof –

have 1: $\vdash bi \ f = (\neg (di \ (\neg f)))$ **by** (*simp add: bi-d-def*)

from 1 **show** *?thesis* **by** *auto*

qed

lemma *NotDiamondNotEqvBox*:

$\vdash (\neg (\Diamond (\neg f))) = \Box f$

by (*simp add: always-d-def*)

lemma *BaEqvBtBi*:

$\vdash ba \ f = \Box (bi \ f)$

proof –

have 1: $\vdash da \ (\neg f) = \Diamond (di \ (\neg f))$ **by** (*rule DaEqvDtDi*)

have 2: $\vdash di \ (\neg f) = (\neg (bi \ f))$ **by** (*rule DiNotEqvNotBi*)

hence 3: $\vdash \Diamond (di \ (\neg f)) = \Diamond (\neg (bi \ f))$ **by** (*rule DiamondEqvDiamond*)

have 4: $\vdash (\neg (\Diamond (\neg (bi \ f)))) = \Box (bi \ f)$ **by** (*rule NotDiamondNotEqvBox*)

have 5: $\vdash (\neg (da \ (\neg f))) = \Box (bi \ f)$ **using** 1 2 3 4 **by** *fastforce*

from 5 **show** *?thesis* **by** (*simp add: ba-d-def*)

qed

lemma *BtBiEqvBiBt*:

$\vdash \Box (bi \ f) = bi (\Box f)$

proof –

have 1: $\vdash ba \ f = \Box (bi \ f)$ **by** (*rule BaEqvBtBi*)

have 2: $\vdash ba \ f = bi (\Box f)$ **by** (*rule BaEqvBiBt*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *BoxStateEqvBaBoxState*:

$\vdash \Box (\text{init } w) = ba (\Box (\text{init } w))$

proof –

have 1: $\vdash (\text{init } w) = bi \ (\text{init } w)$ **by** (*rule StateEqvBi*)

hence 2: $\vdash \Box (\text{init } w) = \Box (bi \ (\text{init } w))$ **by** (*rule BoxEqvBox*)

have 3: $\vdash \Box (bi \ (\text{init } w)) = bi (\Box (\text{init } w))$ **by** (*rule BtBiEqvBiBt*)

have 4: $\vdash \Box (\text{init } w) = \Box (\Box (\text{init } w))$ **by** (*rule BoxEqvBoxBox*)

hence 5: $\vdash bi (\Box (\text{init } w)) = bi (\Box (\Box (\text{init } w)))$ **by** (*rule BiEqvBi*)

have 6: $\vdash ba (\Box (\text{init } w)) = bi (\Box (\Box (\text{init } w)))$ **by** (*rule BaEqvBiBt*)

from 2 3 5 6 **show** *?thesis* **by** *fastforce*

qed

lemma *BaImpBi*:

$\vdash ba \ f \longrightarrow bi \ f$

proof –

have 1: $\vdash ba \ f = \Box (bi \ f)$ **by** (*rule BaEqvBtBi*)

have 2: $\vdash \Box (bi \ f) \longrightarrow bi \ f$ **by** (*rule BoxElim*)

from 1 2 **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*

qed

lemma *BaImpBt*:

$\vdash \text{ba } f \longrightarrow \Box f$

proof –

have 1: $\vdash \text{ba } f = \text{bi}(\Box f)$ **by** (rule *BaEqvBiBt*)

have 2: $\vdash \text{bi}(\Box f) \longrightarrow \Box f$ **by** (rule *BiElim*)

from 1 2 **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*

qed

lemma *DiamondImpDa*:

$\vdash \Diamond f \longrightarrow \text{da } f$

by (metis *DiIntro DiamondImpDiamond da-d-def di-d-def sometimes-d-def*)

lemma *DiImpDa*:

$\vdash \text{di } f \longrightarrow \text{da } f$

by (metis *NowImpDiamond da-d-def di-d-def sometimes-d-def*)

lemma *BoxAndChopImport*:

$\vdash \Box h \wedge f; g \longrightarrow f; (h \wedge g)$

proof –

have 1: $\vdash h \longrightarrow g \longrightarrow (h \wedge g)$ **by** *auto*

hence 2: $\vdash \Box h \longrightarrow \Box(g \longrightarrow (h \wedge g))$ **by** (rule *ImpBoxRule*)

have 3: $\vdash \Box(g \longrightarrow (h \wedge g)) \longrightarrow f; g \longrightarrow f; (h \wedge g)$ **by** (rule *BoxChopImpChop*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *BaAndChopImport*:

$\vdash \text{ba } f \wedge (g; g1) \longrightarrow (f \wedge g); (f \wedge g1)$

proof –

have 1: $\vdash \text{ba } f \longrightarrow \text{bi } f$ **by** (rule *BaImpBi*)

have 2: $\vdash \text{bi } f \wedge (g; g1) \longrightarrow (f \wedge g); g1$ **by** (rule *BiAndChopImport*)

have 3: $\vdash \text{ba } f \longrightarrow \Box f$ **by** (rule *BaImpBt*)

have 4: $\vdash \Box f \wedge (f \wedge g); g1 \longrightarrow (f \wedge g); (f \wedge g1)$ **by** (rule *BoxAndChopImport*)

from 1 2 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *ChopAndCommute*:

$\vdash f; (g \wedge g1) = f; (g1 \wedge g)$

proof –

have 1: $\vdash (g \wedge g1) = (g1 \wedge g)$ **by** *auto*

from 1 **show** *?thesis* **by** (rule *RightChopEqvChop*)

qed

lemma *ChopAndA*:

$\vdash f; (g \wedge g1) \longrightarrow f; g$

proof –

have 1: $\vdash (g \wedge g1) \longrightarrow g$ **by** *auto*

from 1 **show** *?thesis* **by** (rule *RightChopImpChop*)

qed

lemma *ChopAndB*:

$\vdash f; (g \wedge g1) \longrightarrow f; g1$

proof –

have 1: $\vdash (g \wedge g1) \longrightarrow g1$ **by** *auto*

from 1 **show** *?thesis* **by** (rule *RightChopImpChop*)

qed

lemma *BoxStateAndChopEqvChop*:

$\vdash (\Box (init\ w) \wedge (f; g)) = ((\Box (init\ w) \wedge f); (\Box (init\ w) \wedge g))$

proof –

have 1: $\vdash \Box (init\ w) = ba(\Box (init\ w))$

by (rule *BoxStateEqvBaBoxState*)

have 2: $\vdash ba(\Box (init\ w)) \wedge (f; g) \longrightarrow (\Box (init\ w) \wedge f); (\Box (init\ w) \wedge g)$

by (rule *BaAndChopImport*)

have 3: $\vdash \Box (init\ w) \wedge (f; g) \longrightarrow (\Box (init\ w) \wedge f); (\Box (init\ w) \wedge g)$

using 1 2 **by** *fastforce*

have 11: $\vdash (\Box (init\ w) \wedge f); (\Box (init\ w) \wedge g) \longrightarrow (\Box (init\ w)); (\Box (init\ w) \wedge g)$

by (rule *AndChopA*)

have 12: $\vdash (\Box (init\ w)); (\Box (init\ w) \wedge g) \longrightarrow (\Box (init\ w)); (\Box (init\ w))$

by (rule *ChopAndA*)

have 13: $\vdash (\Box (init\ w)); (\Box (init\ w)) = \Box (init\ w)$

by (rule *BoxStateChopBoxEqvBox*)

have 14: $\vdash (\Box (init\ w) \wedge f); (\Box (init\ w) \wedge g) \longrightarrow f; (\Box (init\ w) \wedge g)$

by (rule *AndChopB*)

have 15: $\vdash f; (\Box (init\ w) \wedge g) \longrightarrow f; g$

by (rule *ChopAndB*)

have 16: $\vdash (\Box (init\ w) \wedge f); (\Box (init\ w) \wedge g) \longrightarrow \Box (init\ w) \wedge (f; g)$

using 11 12 13 14 15 **by** *fastforce*

from 3 16 **show** *?thesis* **by** *fastforce*

qed

lemma *DiEqvNotBiNot*:

$\vdash di\ f = (\neg (bi\ (\neg f)))$

proof –

have 1: $\vdash bi\ (\neg f) = (\neg (di\ (\neg \neg f)))$ **by** (simp add: *bi-d-def*)

hence 2: $\vdash di\ (\neg \neg f) = (\neg (bi\ (\neg f)))$ **by** *auto*

have 3: $\vdash f = (\neg \neg f)$ **by** *auto*

hence 4: $\vdash di\ f = di\ (\neg \neg f)$ **by** (rule *DiEqvDi*)

from 2 4 **show** *?thesis* **by** *auto*

qed

lemma *ChopAndBoxImport*:

$\vdash f; g \wedge \Box h \longrightarrow f; (g \wedge h)$

proof –

have 1: $\vdash \Box h \wedge f; g \longrightarrow f; (h \wedge g)$ **by** (rule *BoxAndChopImport*)

have 2: $\vdash f; (h \wedge g) = f; (g \wedge h)$ **by** (rule *ChopAndCommute*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *AndChopAndCommute*:

$\vdash (f \wedge g); (f1 \wedge g1) = (g \wedge f); (g1 \wedge f1)$

proof –
have 1: $\vdash (f \wedge g); (f1 \wedge g1) = (g \wedge f); (f1 \wedge g1)$ **by** (*rule AndChopCommute*)
have 2: $\vdash (g \wedge f); (f1 \wedge g1) = (g \wedge f); (g1 \wedge f1)$ **by** (*rule ChopAndCommute*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *ChopImpChop*:
assumes $\vdash f \longrightarrow f1$
 $\vdash g \longrightarrow g1$
shows $\vdash f;g \longrightarrow f1;g1$
proof –
have 1: $\vdash f \longrightarrow f1$ **using** *assms* **by** *auto*
hence 2: $\vdash f; g \longrightarrow f1; g$ **by** (*rule LeftChopImpChop*)
have 3: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*
hence 4: $\vdash f1; g \longrightarrow f1; g1$ **by** (*rule RightChopImpChop*)
from 2 4 **show** ?thesis **by** fastforce
qed

lemma *ChopEqvChop*:
assumes $\vdash f = f1$
 $\vdash g = g1$
shows $\vdash f;g = f1;g1$
proof –
have 1: $\vdash f = f1$ **using** *assms* **by** *auto*
hence 2: $\vdash f; g = f1; g$ **by** (*rule LeftChopEqvChop*)
have 3: $\vdash g = g1$ **using** *assms* **by** *auto*
hence 4: $\vdash f1; g = f1; g1$ **by** (*rule RightChopEqvChop*)
from 2 4 **show** ?thesis **by** fastforce
qed

lemma *BoxImpBoxImpBox*:
 $\vdash \Box h \longrightarrow \Box(g \longrightarrow \Box h \wedge g)$
proof –
have 1: $\vdash \Box h \longrightarrow (g \longrightarrow \Box h \wedge g)$ **by** *auto*
hence 2: $\vdash \Box(\Box h) \longrightarrow \Box(g \longrightarrow \Box h \wedge g)$ **by** (*rule ImpBoxRule*)
have 3: $\vdash \Box h = \Box(\Box h)$ **by** (*rule BoxEqvBoxBox*)
from 2 3 **show** ?thesis **by** fastforce
qed

lemma *BoxChopImpChopBox*:
 $\vdash \Box h \longrightarrow f; g \longrightarrow f; (\Box h \wedge g)$
proof –
have 1: $\vdash \Box h \longrightarrow \Box(g \longrightarrow \Box h \wedge g)$ **by** (*rule BoxImpBoxImpBox*)
have 2: $\vdash \Box(g \longrightarrow \Box h \wedge g) \longrightarrow f; g \longrightarrow f; (\Box h \wedge g)$ **by** (*rule BoxChopImpChop*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *NotChopEqvYieldsNot*:
 $\vdash (\neg (f; g)) = f \text{ yields } (\neg g)$
proof –

have 1: $\vdash g = (\neg \neg g)$ **by** *auto*
hence 2: $\vdash f; g = f; (\neg \neg g)$ **by** (rule *RightChopEqvChop*)
hence 3: $\vdash (\neg (f; g)) = (\neg (f; (\neg \neg g)))$ **by** *auto*
from 3 **show** *?thesis* **by** (simp add: *yields-d-def*)
qed

lemma *NotDiFalse*:

$\vdash \neg (di \# False)$

proof –

have 1: $\vdash (init \# True) \longrightarrow bi (init \# True)$ **by** (rule *StateImpBi*)
hence 2: $\vdash \# True \longrightarrow bi \# True$ **by** (simp add: *BiGen*)
have 3: $\vdash \# True$ **by** *auto*
have 4: $\vdash bi \# True$ **using** 2 3 *MP* **by** *auto*
hence 5: $\vdash \neg (di (\neg \# True))$ **by** (simp add: *bi-d-def*)
have 6: $\vdash (\neg \# True) = \# False$ **by** *auto*
hence 7: $\vdash di (\neg \# True) = di \# False$ **by** (rule *DiEqvDi*)
from 5 7 **show** *?thesis* **by** *auto*

qed

lemma *StateAndEmptyChop*:

$\vdash ((init\ w) \wedge empty); f = ((init\ w) \wedge f)$

proof –

have 1: $\vdash ((init\ w) \wedge empty); f = ((init\ w) \wedge empty; f)$ **by** (rule *StateAndChop*)
have 2: $\vdash empty; f = f$ **by** (rule *EmptyChop*)
from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *StateAndNextChop*:

$\vdash ((init\ w) \wedge \bigcirc f); g = ((init\ w) \wedge \bigcirc(f; g))$

proof –

have 1: $\vdash ((init\ w) \wedge \bigcirc f); g = ((init\ w) \wedge (\bigcirc f); g)$ **by** (rule *StateAndChop*)
have 2: $\vdash (\bigcirc f); g = \bigcirc(f; g)$ **by** (rule *NextChop*)
from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *NextAndEqvNextAndNext*:

$\vdash \bigcirc (f \wedge g) = (\bigcirc f \wedge \bigcirc g)$

by (metis *NextAndNextEqvNextRule int-eq lift-and-com*)

lemma *NextStateAndChop*:

$\vdash \bigcirc(((init\ w) \wedge f); g) = (\bigcirc (init\ w) \wedge \bigcirc(f; g))$

proof –

have 1: $\vdash ((init\ w) \wedge f); g = ((init\ w) \wedge f; g)$ **by** (rule *StateAndChop*)
hence 2: $\vdash \bigcirc(((init\ w) \wedge f); g) = \bigcirc((init\ w) \wedge f; g)$ **by** (rule *NextEqvNext*)
have 3: $\vdash \bigcirc((init\ w) \wedge f; g) = (\bigcirc (init\ w) \wedge \bigcirc(f; g))$ **by** (rule *NextAndEqvNextAndNext*)
from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *StateYieldsEqv*:

$\vdash ((init\ w) \longrightarrow (f\ yields\ g)) = ((init\ w) \wedge f)\ yields\ g$

proof –

have 1: $\vdash ((init\ w) \wedge f); (\neg\ g) = ((init\ w) \wedge f; (\neg\ g))$ **by** (rule *StateAndChop*)

hence 2: $\vdash ((init\ w) \longrightarrow \neg (f; (\neg\ g))) = (\neg (((init\ w) \wedge f); (\neg\ g)))$ **by** *auto*

from 2 **show** ?thesis **by** (simp add: yields-d-def)

qed

lemma *StateAndDi*:

$\vdash ((init\ w) \wedge\ di\ f) = di\ ((init\ w) \wedge f)$

proof –

have 1: $\vdash ((init\ w) \wedge f); \#True = ((init\ w) \wedge f; \#True)$ **by** (rule *StateAndChop*)

from 1 **show** ?thesis **by** (metis di-d-def inteq-reflection)

qed

lemma *DiNext*:

$\vdash\ di(\bigcirc f) = \bigcirc (di\ f)$

proof –

have 1: $\vdash (\bigcirc f); \#True = \bigcirc(f; \#True)$ **by** (rule *NextChop*)

from 1 **show** ?thesis **by** (simp add: di-d-def)

qed

lemma *DiNextState*:

$\vdash\ di(\bigcirc (init\ w)) = \bigcirc (init\ w)$

proof –

have 1: $\vdash\ di(\bigcirc (init\ w)) = \bigcirc(di\ (init\ w))$ **by** (rule *DiNext*)

have 2: $\vdash\ di\ (init\ w) = (init\ w)$ **by** (rule *DiState*)

hence 3: $\vdash\ \bigcirc(di\ (init\ w)) = \bigcirc (init\ w)$ **by** (rule *NextEqvNext*)

from 1 3 **show** ?thesis **by** *fastforce*

qed

lemma *StateImpBiGen*:

assumes $\vdash (init\ w) \longrightarrow f$

shows $\vdash (init\ w) \longrightarrow bi\ f$

proof –

have 1: $\vdash (init\ w) \longrightarrow f$ **using** *assms* **by** *auto*

hence 2: $\vdash \neg f \longrightarrow \neg (init\ w)$ **by** *auto*

hence 3: $\vdash\ di(\neg f) \longrightarrow di(\neg (init\ w))$ **by** (rule *DiImpDi*)

hence 4: $\vdash\ di(\neg f) \longrightarrow di\ (init\ (\neg w))$ **by** (metis *Initprop*(2) *inteq-reflection*)

have 5: $\vdash\ di\ (init\ (\neg w)) = (init\ (\neg w))$ **by** (rule *DiState*)

have 6: $\vdash\ di(\neg f) \longrightarrow \neg (init\ w)$ **using** 4 5 **using** *Initprop*(2) **by** *fastforce*

hence 7: $\vdash (init\ w) \longrightarrow \neg (di(\neg f))$ **by** *auto*

from 7 **show** ?thesis **by** (simp add: bi-d-def)

qed

lemma *ChopAndNotChopImp*:

$\vdash\ f; g \wedge \neg (f; g1) \longrightarrow f; (g \wedge \neg\ g1)$

proof –

have 1: $\vdash g \longrightarrow (g \wedge \neg\ g1) \vee g1$ **by** *auto*

hence 2: $\vdash f; g \longrightarrow f; ((g \wedge \neg\ g1) \vee g1)$ **by** (rule *RightChopImpChop*)

have 3: $\vdash f; ((g \wedge \neg\ g1) \vee g1) \longrightarrow (f; (g \wedge \neg\ g1)) \vee (f; g1)$ **by** (rule *ChopOrImp*)

have 4: $\vdash f; g \longrightarrow f; (g \wedge \neg\ g1) \vee f; g1$ **using** 2 3 *MP* **by** *fastforce*

from 4 show ?thesis by auto
qed

lemma ChopAndYieldsImp:

$\vdash f; g \wedge f \text{ yields } g1 \longrightarrow f; (g \wedge g1)$

proof –

have 1: $\vdash g \longrightarrow (g \wedge g1) \vee \neg g1$ by auto

hence 2: $\vdash f; g \longrightarrow f; ((g \wedge g1) \vee \neg g1)$ by (rule RightChopImpChop)

have 3: $\vdash f; ((g \wedge g1) \vee \neg g1) \longrightarrow (f; (g \wedge g1)) \vee (f; (\neg g1))$ by (rule ChopOrImp)

have 4: $\vdash f; g \longrightarrow f; (g \wedge g1) \vee f; (\neg g1)$ using 2 3 MP by fastforce

hence 5: $\vdash f; g \wedge \neg (f; (\neg g1)) \longrightarrow f; (g \wedge g1)$ by auto

from 5 show ?thesis by (simp add: yields-d-def)

qed

lemma ChopAndYieldsMP:

$\vdash f; g \wedge f \text{ yields } (g \longrightarrow g1) \longrightarrow f; g1$

proof –

have 1: $\vdash f; g \wedge f \text{ yields } (g \longrightarrow g1) \longrightarrow f; (g \wedge (g \longrightarrow g1))$ by (rule ChopAndYieldsImp)

have 2: $\vdash g \wedge (g \longrightarrow g1) \longrightarrow g1$ by auto

hence 3: $\vdash f; (g \wedge (g \longrightarrow g1)) \longrightarrow f; g1$ by (rule RightChopImpChop)

from 1 3 show ?thesis by fastforce

qed

lemma OrYieldsImp:

$\vdash (f \vee f1) \text{ yields } g = ((f \text{ yields } g) \wedge (f1 \text{ yields } g))$

proof –

have 1: $\vdash ((f \vee f1); (\neg g)) = ((f; (\neg g)) \vee (f1; (\neg g)))$ by (rule OrChopEqv)

hence 2: $\vdash (\neg ((f \vee f1); (\neg g))) = (\neg (f; (\neg g)) \wedge \neg (f1; (\neg g)))$ by auto

from 2 show ?thesis by (simp add: yields-d-def)

qed

lemma LeftYieldsImpYields:

assumes $\vdash f \longrightarrow f1$

shows $\vdash (f1 \text{ yields } g) \longrightarrow (f \text{ yields } g)$

proof –

have 1: $\vdash f \longrightarrow f1$ using assms by auto

hence 2: $\vdash f; (\neg g) \longrightarrow f1; (\neg g)$ by (rule LeftChopImpChop)

hence 3: $\vdash \neg (f1; (\neg g)) \longrightarrow \neg (f; (\neg g))$ by auto

from 3 show ?thesis by (simp add: yields-d-def)

qed

lemma LeftYieldsEqvYields:

assumes $\vdash f = f1$

shows $\vdash (f \text{ yields } g) = (f1 \text{ yields } g)$

proof –

have 1: $\vdash f = f1$ using assms by auto

hence 2: $\vdash f; (\neg g) = f1; (\neg g)$ by (rule LeftChopEqvChop)

hence 3: $\vdash (\neg (f; (\neg g))) = (\neg (f1; (\neg g)))$ by auto

from 3 show ?thesis by (simp add: yields-d-def)

qed

2.3.7 Properties of Fin

lemma *FinEqvTrueChopAndEmpty*:

$\vdash \text{fin } f = \# \text{True}; (f \wedge \text{empty})$
proof –
have 1: $\vdash \text{fin } f = \Box(\text{empty} \longrightarrow f)$
by (*simp add: fin-d-def*)
have 2: $\vdash \Box(\text{empty} \longrightarrow f) = (\neg(\Diamond(\neg(\text{empty} \longrightarrow f))))$
by (*simp add: always-d-def*)
have 3: $\vdash (\neg(\text{empty} \longrightarrow f)) = (\neg f \wedge \text{empty})$
by *auto*
hence 4: $\vdash \Diamond(\neg(\text{empty} \longrightarrow f)) = \Diamond(\neg f \wedge \text{empty})$
using *DiamondEqvDiamond* **by** *blast*
hence 5: $\vdash \neg(\Diamond(\neg(\text{empty} \longrightarrow f))) = (\neg(\Diamond(\neg f \wedge \text{empty})))$
by *auto*
have 51: $\vdash \text{finite}; ((\neg f \wedge \text{empty}) \wedge \text{finite}) \longrightarrow \text{finite} ; (\neg f \wedge \text{empty})$
by (*simp add: ChopAndA*)
have 52: $\vdash (\# \text{True}; (\neg f \wedge \text{empty}) \wedge \text{finite}) \longrightarrow \text{finite} ; (\neg f \wedge \text{empty})$
by (*metis 51 TrueChopAndFiniteEqvAndFiniteChopFinite int-eq*)
have 53: $\vdash \neg(\# \text{True}; (f \wedge \text{empty})) \longrightarrow \text{finite} ; (\neg f \wedge \text{empty})$
by (*metis 52 Finprop(5) int-eq*)
have 54: $\vdash \neg \text{finite}; (\neg f \wedge \text{empty}) \longrightarrow \# \text{True}; (f \wedge \text{empty})$
using 53 **by** *auto*
have 6: $\vdash (\neg(\Diamond(\neg f \wedge \text{empty}))) \longrightarrow \# \text{True}; (f \wedge \text{empty})$
unfolding *sometimes-d-def* **using** 54 **by** *auto*
have 61: $\vdash \neg f \wedge \text{empty} \longrightarrow \text{finite}$
by (*metis ChopAndB DiamondEmptyEqvFinite NowImpDiamond inteq-reflection lift-imp-trans sometimes-d-def*)
have 62: $\vdash (\neg \# \text{True}; (f \wedge \text{empty})) = \text{finite}; (\neg f \wedge \text{empty})$
using 61
by (*metis (no-types) Finprop(5) Prop10 TrueChopAndFiniteEqvAndFiniteChopFinite inteq-reflection*)
have 7: $\vdash \# \text{True}; (f \wedge \text{empty}) \longrightarrow (\neg(\Diamond(\neg f \wedge \text{empty})))$
unfolding *sometimes-d-def* **using** *TrueChopAndFiniteEqvAndFiniteChopFinite*[of *LIFT(f \wedge empty)*]
using 62 **by** *auto*
have 8: $\vdash (\neg(\Diamond(\neg f \wedge \text{empty}))) = \# \text{True}; (f \wedge \text{empty})$
by (*simp add: 6 7 int-iffI*)
from 1 2 5 8 **show** *?thesis* **by** *fastforce*
qed

lemma *DiamondFin*:

$\vdash \Diamond(\text{fin } w) = \text{fin } w$
by (*metis (no-types, lifting) ChopAssoc ChopOrEqv FinEqvTrueChopAndEmpty FiniteChopFiniteEqvFinite FiniteChopInfEqvInf FiniteOrInfinite int-eq-true inteq-reflection sometimes-d-def*)

lemma *FiniteChopFinExportA*:

$\vdash (f \wedge \text{finite}); (g \wedge \text{fin } w) \longrightarrow \text{fin } w$
using *DiamondFin*
by (*metis ChopAndB FiniteChopImpDiamond inteq-reflection lift-imp-trans*)

lemma *FinImpBox*:

$\vdash \text{fin } w \longrightarrow \square(\text{fin } w)$
by (*metis* *BoxImpBoxBox* *fin-d-def*)

lemma *FinAndChopImport*:

$\vdash (\text{fin } w) \wedge (f;g) \longrightarrow f;((\text{fin } w) \wedge g)$

proof –

have 1: $\vdash \text{fin } w \longrightarrow \square(\text{fin } w)$ **by** (*rule* *FinImpBox*)

hence 2: $\vdash \text{fin } w \wedge f;g \longrightarrow \square(\text{fin } w) \wedge (f;g)$ **by** *auto*

have 3: $\vdash \square(\text{fin } w) \wedge (f;g) \longrightarrow f;((\text{fin } w) \wedge g)$ **using** *BoxAndChopImport* **by** *blast*
from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*

qed

lemma *FinAndChop*:

$\vdash ((f \wedge \text{finite});(g \wedge \text{fin } w)) = (\text{fin } w \wedge (f \wedge \text{finite});g)$

using *FinAndChopImport* *FiniteChopFinExportA* *ChopAndA* *ChopAndCommute*
by *fastforce*

lemma *ChopAndEmptyEqvEmptyChopEmpty*:

$\vdash ((f;g) \wedge \text{empty}) = (f \wedge \text{empty});(g \wedge \text{empty})$

by (*auto simp: itl-defs min-absorb1*)

lemma *FinAndEmpty*:

$\vdash ((\text{fin } w) \wedge \text{empty}) = (w \wedge \text{empty})$

proof –

have 1: $\vdash ((\text{fin } w) \wedge \text{empty}) = (\# \text{True};(w \wedge \text{empty}) \wedge \text{empty})$

using *FinEqvTrueChopAndEmpty* **by** *fastforce*

have 2: $\vdash (\# \text{True};(w \wedge \text{empty}) \wedge \text{empty}) = ((\# \text{True} \wedge \text{empty});(w \wedge \text{empty}))$

using *ChopAndEmptyEqvEmptyChopEmpty*[of *LIFT*($\# \text{True}$) *LIFT*($w \wedge \text{empty}$)]

by (*metis* *AndChopA* *ChopAndA* *ChopAndEmptyEqvEmptyChopEmpty* *Prop11* *Prop12* *int-eq*)

have 3: $\vdash (\# \text{True} \wedge \text{empty});(w \wedge \text{empty}) = (\text{empty};(w \wedge \text{empty}))$

using *LeftChopEqvChop* **by** *fastforce*

have 4: $\vdash (\text{empty};(w \wedge \text{empty})) = (w \wedge \text{empty})$

using *EmptyChop* **by** *blast*

from 1 2 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *AndFinEqvChopAndEmpty*:

$\vdash ((f \wedge \text{finite}) \wedge \text{fin } g) = (f \wedge \text{finite});(g \wedge \text{empty})$

proof –

have 1: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } g) = ((f \wedge \text{finite}) ; \text{empty} \wedge \text{fin } g)$

using *ChopEmpty* **by** (*metis* *inteq-reflection*)

have 2: $\vdash (\text{fin } g \wedge (f \wedge \text{finite});\text{empty}) = ((f \wedge \text{finite});(\text{empty} \wedge \text{fin } g))$

using *FinAndChop* **by** *fastforce*

have 3: $\vdash (\text{empty} \wedge \text{fin } g) = (\text{fin } g \wedge \text{empty})$

by *auto*

have 4: $\vdash (\text{fin } g \wedge \text{empty}) = (g \wedge \text{empty})$

using *FinAndEmpty* **by** *metis*

have 5: $\vdash (\text{empty} \wedge \text{fin } g) = (g \wedge \text{empty})$

using 3 4 **by** *auto*

hence 6: $\vdash (f \wedge \text{finite});(\text{empty} \wedge \text{fin } g) = (f \wedge \text{finite});(g \wedge \text{empty})$

using *RightChopEqvChop* by *blast*
 from 1 2 5 show ?thesis by (metis integ-reflection lift-and-com)
 qed

lemma *AndFinEqvChopStateAndEmpty*:
 $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)) = (f \wedge \text{finite}); ((\text{init } w) \wedge \text{empty})$
 using *AndFinEqvChopAndEmpty* by *blast*

lemma *FinStateEqvStateAndEmptyOrNextFinState*:
 $\vdash \text{fin } (\text{init } w) = (((\text{init } w) \wedge \text{empty}) \vee \bigcirc(\text{fin } (\text{init } w)))$
proof –
 have 1: $\vdash \text{fin } (\text{init } w) = \Box(\text{empty} \longrightarrow \text{init } w)$
 by (simp add: fin-d-def)
 have 2: $\vdash \Box(\text{empty} \longrightarrow \text{init } w) =$
 $((\text{empty} \longrightarrow \text{init } w) \wedge \text{wnext } (\Box(\text{empty} \longrightarrow \text{init } w)))$
 by (rule BoxEqvAndWnextBox)
 have 3: $\vdash \text{fin } (\text{init } w) = ((\text{empty} \longrightarrow \text{init } w) \wedge \text{wnext } (\text{fin } (\text{init } w)))$
 using 1 2 by (simp add: fin-d-def)
 have 4: $\vdash \text{wnext } (\text{fin } (\text{init } w)) = (\text{empty} \vee \bigcirc(\text{fin } (\text{init } w)))$
 by (rule WnextEqvEmptyOrNext)
 have 5: $\vdash \text{fin } (\text{init } w) = ((\text{empty} \longrightarrow \text{init } w) \wedge (\text{empty} \vee \bigcirc(\text{fin } (\text{init } w))))$
 using 3 4 by fastforce
 have 6: $\vdash ((\text{empty} \longrightarrow \text{init } w) \wedge (\text{empty} \vee \bigcirc(\text{fin } (\text{init } w)))) =$
 $((\text{empty} \longrightarrow \text{init } w) \wedge \text{empty}) \vee ((\text{empty} \longrightarrow \text{init } w) \wedge \bigcirc(\text{fin } (\text{init } w)))$
 by auto
 have 7: $\vdash ((\text{empty} \longrightarrow \text{init } w) \wedge \text{empty}) = ((\text{init } w) \wedge \text{empty})$
 by auto
 have 8: $\vdash ((\text{empty} \longrightarrow \text{init } w) \wedge \bigcirc(\text{fin } (\text{init } w))) = \bigcirc(\text{fin } (\text{init } w))$
 by (metis (no-types, lifting) 5 DiamondFin NextDiamondImpDiamond Prop10 Prop12 int-eq lift-and-com)
 have 9: $\vdash (((\text{empty} \longrightarrow \text{init } w) \wedge \text{empty}) \vee ((\text{empty} \longrightarrow \text{init } w) \wedge \bigcirc(\text{fin } (\text{init } w)))) =$
 $((\text{init } w) \wedge \text{empty}) \vee \bigcirc(\text{fin } (\text{init } w))$
 using 7 8 by auto
 from 5 6 8 9 show ?thesis by fastforce
 qed

lemma *FinChopEqvOr*:
 $\vdash (\text{fin } (\text{init } w)); f = (((\text{init } w) \wedge f) \vee \bigcirc((\text{fin } (\text{init } w)); f))$
proof –
 have 1: $\vdash \text{fin } (\text{init } w) = (((\text{init } w) \wedge \text{empty}) \vee \bigcirc(\text{fin } (\text{init } w)))$
 by (rule FinStateEqvStateAndEmptyOrNextFinState)
 hence 2: $\vdash (\text{fin } (\text{init } w)); f = (((\text{init } w) \wedge \text{empty}) \vee \bigcirc(\text{fin } (\text{init } w))); f$
 by (rule LeftChopEqvChop)
 have 3: $\vdash (((\text{init } w) \wedge \text{empty}) \vee \bigcirc(\text{fin } (\text{init } w))); f =$
 $((\text{init } w) \wedge \text{empty}); f \vee (\bigcirc(\text{fin } (\text{init } w))); f$
 by (rule OrChopEqv)
 have 4: $\vdash ((\text{init } w) \wedge \text{empty}); f = ((\text{init } w) \wedge f)$
 by (rule StateAndEmptyChop)
 have 5: $\vdash (\bigcirc(\text{fin } (\text{init } w))); f = \bigcirc((\text{fin } (\text{init } w)); f)$
 by (rule NextChop)

from 2 3 4 5 show ?thesis by fastforce
qed

lemma *FinChopEqvDiamond*:

$\vdash (\text{fin } (\text{init } w) \wedge \text{finite}); f = \Diamond ((\text{init } w) \wedge f)$

proof –

have 1: $\vdash (\text{fin } (\text{init } w) \wedge \text{finite}) = (\text{finite}; ((\text{init } w) \wedge \text{empty}))$

by (*metis AndFinEqvChopAndEmpty int-simps(17) inteq-reflection lift-and-com*)

hence 2: $\vdash (\text{fin } (\text{init } w) \wedge \text{finite}); f = (\text{finite}; ((\text{init } w) \wedge \text{empty}); f)$

by (*rule LeftChopEqvChop*)

have 3: $\vdash \text{finite}; ((\text{init } w) \wedge \text{empty}); f = (\text{finite}; ((\text{init } w) \wedge \text{empty}); f)$

by (*rule ChopAssoc*)

have 4: $\vdash \text{finite}; ((\text{init } w) \wedge \text{empty}); f = \Diamond ((\text{init } w) \wedge \text{empty}); f)$

by (*simp add: sometimes-d-def*)

have 5: $\vdash ((\text{init } w) \wedge \text{empty}); f = ((\text{init } w) \wedge f)$

using *StateAndEmptyChop* **by** *blast*

hence 6: $\vdash \Diamond ((\text{init } w) \wedge \text{empty}); f = \Diamond ((\text{init } w) \wedge f)$

by (*rule DiamondEqvDiamond*)

from 2 3 4 6 show ?thesis by fastforce

qed

lemma *NotDiamondAndNot*:

$\vdash \neg(\Diamond (f \wedge \neg f))$

proof –

have 1: $\vdash (\neg(\Diamond (f \wedge \neg f))) = \Box(\neg(f \wedge \neg f))$ **using** *NotDiamondNotEqvBox* **by** *fastforce*

have 2: $\vdash \neg(f \wedge \neg f)$ **by** *simp*

have 3: $\vdash \Box(\neg(f \wedge \neg f))$ **using** 2 **by** (*simp add: BoxGen*)

from 1 3 show ?thesis by fastforce

qed

lemma *FinYields*:

$\vdash (\text{fin } (\text{init } w) \wedge \text{finite}) \text{ yields } (\text{init } w)$

proof –

have 1: $\vdash (\text{fin } (\text{init } w) \wedge \text{finite}); (\neg(\text{init } w)) = \Diamond((\text{init } w) \wedge \neg(\text{init } w))$

by (*rule FinChopEqvDiamond*)

have 2: $\vdash \neg(\Diamond((\text{init } w) \wedge \neg(\text{init } w)))$

by (*rule NotDiamondAndNot*)

have 3: $\vdash \neg((\text{fin } (\text{init } w) \wedge \text{finite}); (\neg(\text{init } w)))$

using 1 2 **by** *fastforce*

from 3 show ?thesis by (*simp add: yields-d-def*)

qed

lemma *ImpAndFinStateOrFinNotState*:

$\vdash f \longrightarrow (f \wedge \text{fin } (\text{init } w)) \vee (f \wedge \text{fin } (\neg(\text{init } w)))$

by (*simp add: itl-defs Valid-def*)

lemma *AndFinChopEqvStateAndChop*:

$\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); g = (f \wedge \text{finite}); ((\text{init } w) \wedge g)$

proof –

have 1: $\vdash (\text{fin } (\text{init } w) \wedge \text{finite}) \text{ yields } (\text{init } w)$

by (rule *FinYields*)
have 2: $\vdash (f \wedge \text{finite}) \wedge \text{fin } (\text{init } w) \longrightarrow \text{fin } (\text{init } w)$
 by *auto*
hence 3: $\vdash (\text{fin } (\text{init } w) \wedge \text{finite}) \text{ yields } (\text{init } w) \longrightarrow$
 $((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)) \text{ yields } (\text{init } w)$
 using *LeftYieldsImpYields*
 by (metis *AndFinEqvChopAndEmpty Prop11 Prop12 inteq-reflection*)
have 4: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)) \text{ yields } (\text{init } w)$
 using 1 3 *MP* by *fastforce*
have 5: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); g \wedge ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)) \text{ yields } (\text{init } w)$
 $\longrightarrow ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); (g \wedge (\text{init } w))$
 by (rule *ChopAndYieldsImp*)
have 6: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); g \longrightarrow$
 $((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); (g \wedge (\text{init } w))$
 using 4 5 by *fastforce*
have 7: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); (g \wedge (\text{init } w)) \longrightarrow (f \wedge \text{finite}); (g \wedge (\text{init } w))$
 by (rule *AndChopA*)
have 8: $\vdash g \wedge (\text{init } w) \longrightarrow (\text{init } w) \wedge g$
 by *auto*
hence 9: $\vdash (f \wedge \text{finite}); (g \wedge (\text{init } w)) \longrightarrow (f \wedge \text{finite}); ((\text{init } w) \wedge g)$
 by (rule *RightChopImpChop*)
have 10: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); g \longrightarrow (f \wedge \text{finite}); ((\text{init } w) \wedge g)$
 using 6 7 9 by *fastforce*
have 11: $\vdash (f \wedge \text{finite}) \longrightarrow ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)) \vee ((f \wedge \text{finite}) \wedge \text{fin } (\neg (\text{init } w)))$
 using *ImpAndFinStateOrFinNotState* by *blast*
hence 12: $\vdash (f \wedge \text{finite}); ((\text{init } w) \wedge g) \longrightarrow$
 $((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)) \vee$
 $((\text{finite} \wedge f) \wedge \text{fin } (\neg (\text{init } w))) ; ((\text{init } w) \wedge g)$
 using *LeftChopImpChop*
 by (metis *inteq-reflection lift-and-com*)
have 13: $\vdash (((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)) \vee ((f \wedge \text{finite}) \wedge \text{fin } (\neg (\text{init } w)))) ; ((\text{init } w) \wedge g)$
 $=$
 $((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w));$
 $((\text{init } w) \wedge g) \vee ((f \wedge \text{finite}) \wedge \text{fin } (\neg (\text{init } w))); ((\text{init } w) \wedge g))$
 by (rule *OrChopEqv*)
have 14: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } (\neg w))); ((\text{init } w) \wedge g) \longrightarrow$
 $\Diamond((\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g))$
 using *FinChopEqvDiamond*
 by (metis *AndFinEqvChopAndEmpty ChopEmpty FiniteChopImpDiamond LeftChopImpChop int-eq*)
have 141: $\vdash \neg(\Diamond((\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g))) \longrightarrow$
 $\neg((f \wedge \text{finite}) \wedge \text{fin } (\text{init } (\neg w))); ((\text{init } w) \wedge g))$
 using 14 by *fastforce*
have 142: $\vdash ((\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g)) = \#False$
 using *Initprop(2)* by *fastforce*
have 15: $\vdash \neg(\Diamond((\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g)))$
 by (metis 142 *NotDiamondAndNot int-simps(21) inteq-reflection*)
have 151: $\vdash \neg((f \wedge \text{finite}) \wedge \text{fin } (\text{init } (\neg w))); ((\text{init } w) \wedge g))$
 using 15 141 by *fastforce*
have 1511: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\neg (\text{init } w))); ((\text{init } w) \wedge g) \longrightarrow \#False$
 using 151 by (metis *Initprop(2) int-simps(14) inteq-reflection*)

have 152: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w));$
 $((\text{init } w) \wedge g) \vee ((f \wedge \text{finite}) \wedge \text{fin } (\neg (\text{init } w))); ((\text{init } w) \wedge g) \longrightarrow$
 $((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); ((\text{init } w) \wedge g)$
using 1511 **by** *fastforce*
have 16: $\vdash (f \wedge \text{finite}); ((\text{init } w) \wedge g) \longrightarrow ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); ((\text{init } w) \wedge g)$
using 12 13 152
proof –
have $\vdash (f \wedge \text{finite}); (\text{init } w \wedge g) \longrightarrow$
 $((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w) \vee (f \wedge \text{finite}) \wedge \text{fin } (\neg \text{init } w)); (\text{init } w \wedge g)$
by (*metis* 12 *inteq-reflection lift-and-com*)
then show *?thesis*
using 13 152 **by** *fastforce*
qed
have 17: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); ((\text{init } w) \wedge g) \longrightarrow ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); g$
by (*rule ChopAndB*)
have 18: $\vdash (f \wedge \text{finite}); ((\text{init } w) \wedge g) \longrightarrow ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); g$
using 16 17 **by** *fastforce*
from 10 18 **show** *?thesis* **by** *fastforce*
qed

lemma *DiAndFinEqvChopState*:

$\vdash \text{di } ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)) = (f \wedge \text{finite}); (\text{init } w)$

proof –

have 1: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); \# \text{True} = (f \wedge \text{finite}); ((\text{init } w) \wedge \# \text{True})$

by (*rule AndFinChopEqvStateAndChop*)

have 2: $\vdash ((\text{init } w) \wedge \# \text{True}) = (\text{init } w)$

by *auto*

hence 3: $\vdash ((f \wedge \text{finite}); ((\text{init } w) \wedge \# \text{True})) = ((f \wedge \text{finite}); (\text{init } w))$

by (*rule RightChopEqvChop*)

have 4: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); \# \text{True} = (f \wedge \text{finite}); (\text{init } w)$

using 1 3 **by** *auto*

from 4 **show** *?thesis* **by** (*simp add: di-d-def*)

qed

lemma *FinNotStateEqvNotFinState*:

$\vdash (\neg (\text{fin } (\text{init } w)) \wedge \text{finite}) = (\text{fin } (\text{init } (\neg w)) \wedge \text{finite})$

using *FinEqvTrueChopAndEmpty Finprop(4) Initprop(2) FiniteImpAnd* **by** (*metis inteq-reflection*)

lemma *BiImpFinEqvYieldsState*:

$\vdash \text{bi } (f \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w)) = (f \wedge \text{finite}) \text{yields } (\text{init } w)$

proof –

have 1: $\vdash \text{di } ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } (\neg w))) = (f \wedge \text{finite}); (\text{init } (\neg w))$

by (*rule DiAndFinEqvChopState*)

have 2: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } (\neg w))) = ((f \wedge \text{finite}) \wedge \neg (\text{fin } (\text{init } w)))$

using *FinNotStateEqvNotFinState* **by** *fastforce*

have 3: $\vdash ((f \wedge \text{finite}) \wedge \neg (\text{fin } (\text{init } w))) = (\neg (f \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w)))$

by *auto*

have 4: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } (\neg w))) = (\neg (f \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w)))$

using 2 3 **by** *fastforce*

hence 5: $\vdash \text{di } ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } (\neg w))) = \text{di } (\neg (f \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w)))$

by (rule DiEqvDi)
 have 6: $\vdash di (\neg (f \wedge finite \longrightarrow fin (init w))) = (\neg (bi (f \wedge finite \longrightarrow fin (init w))))$
 by (rule DiNotEqvNotBi)
 have 7: $\vdash \neg (bi (f \wedge finite \longrightarrow fin (init w))) = (f \wedge finite); (init (\neg w))$
 using 1 5 6 Initprop by fastforce
 hence 8: $\vdash bi (f \wedge finite \longrightarrow fin (init w)) = (\neg ((f \wedge finite); (\neg (init w))))$
 by (metis Initprop(2) int-eq int-simps(7))
 from 8 show ?thesis by (simp add: yields-d-def)
 qed

lemma StateImpYields:

assumes $\vdash (init w) \wedge f \wedge finite \longrightarrow fin (init w1)$
 shows $\vdash (init w) \longrightarrow ((f \wedge finite) yields (init w1))$
 proof –
 have 1: $\vdash (init w) \wedge f \wedge finite \longrightarrow fin (init w1)$
 using assms by auto
 hence 2: $\vdash (init w) \longrightarrow (f \wedge finite \longrightarrow fin (init w1))$
 by auto
 hence 3: $\vdash (init w) \longrightarrow bi (f \wedge finite \longrightarrow fin (init w1))$
 by (rule StateImpBiGen)
 have 4: $\vdash bi (f \wedge finite \longrightarrow fin (init w1)) = (f \wedge finite) yields (init w1)$
 by (rule BiImpFinEqvYieldsState)
 from 3 4 show ?thesis by fastforce
 qed

lemma StateAndYieldsImpYields:

assumes $\vdash (init w) \wedge f \longrightarrow f1$
 shows $\vdash (init w) \wedge (f1 yields g) \longrightarrow (f yields g)$
 proof –
 have 1: $\vdash (init w) \wedge f \longrightarrow f1$ using assms by auto
 hence 2: $\vdash (init w) \wedge (f; (\neg g)) \longrightarrow f1; (\neg g)$ by (rule StateAndChopImpChopRule)
 hence 3: $\vdash (init w) \wedge \neg (f1; (\neg g)) \longrightarrow \neg (f; (\neg g))$ by auto
 from 3 show ?thesis by (simp add: yields-d-def)
 qed

lemma AndYieldsA:

$\vdash f yields g \longrightarrow (f \wedge f1) yields g$
 proof –
 have 1: $\vdash f \wedge f1 \longrightarrow f$ by auto
 from 1 show ?thesis by (rule LeftYieldsImpYields)
 qed

lemma AndYieldsB:

$\vdash f1 yields g \longrightarrow (f \wedge f1) yields g$
 proof –
 have 1: $\vdash f \wedge f1 \longrightarrow f1$ by auto
 from 1 show ?thesis by (rule LeftYieldsImpYields)
 qed

lemma RightYieldsImpYields:

assumes $\vdash g \longrightarrow g1$
shows $\vdash (f \text{ yields } g) \longrightarrow (f \text{ yields } g1)$
proof –
have 1: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*
hence 2: $\vdash \neg g1 \longrightarrow \neg g$ **by** *auto*
hence 3: $\vdash f; (\neg g1) \longrightarrow f; (\neg g)$ **by** (*rule RightChopImpChop*)
hence 4: $\vdash \neg (f; (\neg g)) \longrightarrow \neg (f; (\neg g1))$ **by** *auto*
from 4 **show** *?thesis* **by** (*simp add: yields-d-def*)
qed

lemma *RightYieldsEqvYields*:

assumes $\vdash g = g1$
shows $\vdash (f \text{ yields } g) = (f \text{ yields } g1)$
proof –
have 1: $\vdash g = g1$ **using** *assms* **by** *auto*
hence 2: $\vdash (\neg g) = (\neg g1)$ **by** *auto*
hence 3: $\vdash f; (\neg g) = f; (\neg g1)$ **by** (*rule RightChopEqvChop*)
hence 4: $\vdash (\neg (f; (\neg g))) = (\neg (f; (\neg g1)))$ **by** *auto*
from 4 **show** *?thesis* **by** (*simp add: yields-d-def*)
qed

lemma *BoxImpYields*:

$\vdash \Box g \longrightarrow (f \wedge \text{finite}) \text{ yields } g$
proof –
have 1: $\vdash (f \wedge \text{finite}); (\neg g) \longrightarrow \Diamond(\neg g)$ **by** (*rule FiniteChopImpDiamond*)
hence 2: $\vdash \neg (\Diamond(\neg g)) \longrightarrow \neg ((f \wedge \text{finite}); (\neg g))$ **by** *auto*
from 2 **show** *?thesis* **by** (*simp add: yields-d-def always-d-def*)
qed

lemma *BoxEqvFiniteYields*:

$\vdash \Box f = \text{finite yields } f$
proof –
have 1: $\vdash \text{finite}; (\neg f) = \Diamond(\neg f)$ **by** (*rule FiniteChopEqvDiamond*)
hence 2: $\vdash (\neg (\text{finite}; (\neg f))) = (\neg (\Diamond(\neg f)))$ **by** *auto*
have 3: $\vdash \Box f = (\neg (\Diamond(\neg f)))$ **by** (*simp add: always-d-def*)
have 4: $\vdash \Box f = (\neg (\text{finite}; (\neg f)))$ **using** 2 3 **by** *fastforce*
from 4 **show** *?thesis* **by** (*simp add: yields-d-def*)
qed

lemma *YieldsGen*:

assumes $\vdash g$
shows $\vdash (f \wedge \text{finite}) \text{ yields } g$
proof –
have 1: $\vdash g$ **using** *assms* **by** *auto*
hence 2: $\vdash \Box g$ **by** (*rule BoxGen*)
have 3: $\vdash \Box g \longrightarrow (f \wedge \text{finite}) \text{ yields } g$ **by** (*rule BoxImpYields*)
from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *YieldsAndYieldsEqvYieldsAnd*:

$\vdash ((f \text{ yields } g) \wedge (f \text{ yields } g1)) = f \text{ yields } (g \wedge g1)$
proof –
have 1: $\vdash f; (\neg g \vee \neg g1) = ((f; (\neg g)) \vee (f; (\neg g1)))$
by (rule ChopOrEqv)
hence 2: $\vdash ((f; (\neg g)) \vee (f; (\neg g1))) = f; (\neg g \vee \neg g1)$
by auto
have 3: $\vdash (\neg g \vee \neg g1) = (\neg (g \wedge g1))$
by auto
hence 4: $\vdash f; (\neg g \vee \neg g1) = f; (\neg (g \wedge g1))$
by (rule RightChopEqvChop)
have 5: $\vdash (f; (\neg g)) \vee (f; (\neg g1)) = f; (\neg (g \wedge g1))$
using 2 4 **by** fastforce
hence 6: $\vdash (\neg (f; (\neg g)) \wedge \neg (f; (\neg g1))) = (\neg (f; (\neg (g \wedge g1))))$
by (metis 1 3 int-simps(14) int-simps(33) inteq-reflection)
from 6 **show** ?thesis **by** (simp add: yields-d-def)
qed

lemma YieldsAndYieldsImpAndYieldsAnd:
 $\vdash (f \text{ yields } g) \wedge (f1 \text{ yields } g1) \longrightarrow (f \wedge f1) \text{ yields } (g \wedge g1)$
proof –
have 1: $\vdash f \text{ yields } g \longrightarrow (f \wedge f1) \text{ yields } g$
by (rule AndYieldsA)
have 2: $\vdash f1 \text{ yields } g1 \longrightarrow (f \wedge f1) \text{ yields } g1$
by (rule AndYieldsB)
have 3: $\vdash ((f \wedge f1) \text{ yields } g \wedge (f \wedge f1) \text{ yields } g1) = (f \wedge f1) \text{ yields } (g \wedge g1)$
by (rule YieldsAndYieldsEqvYieldsAnd)
from 1 2 3 **show** ?thesis **by** fastforce
qed

lemma YieldsYieldsEqvChopYields:
 $\vdash f \text{ yields } (g \text{ yields } h) = (f; g) \text{ yields } h$
proof –
have 1: $\vdash f; (g; (\neg h)) = (f; g); (\neg h)$ **by** (rule ChopAssoc)
hence 2: $\vdash f; (g; (\neg h)) = (f; g); (\neg h)$ **by** auto
have 3: $\vdash g; (\neg h) = (\neg \neg (g; (\neg h)))$ **by** auto
hence 4: $\vdash f; (g; (\neg h)) = f; (\neg \neg (g; (\neg h)))$ **by** (rule RightChopEqvChop)
have 5: $\vdash f; (\neg \neg (g; (\neg h))) = (f; g); (\neg h)$ **using** 2 4 **by** auto
hence 6: $\vdash f; (\neg (g \text{ yields } h)) = (f; g); (\neg h)$ **by** (simp add: yields-d-def)
hence 7: $\vdash (\neg (f; (\neg (g \text{ yields } h)))) = (\neg ((f; g); (\neg h)))$ **by** auto
from 7 **show** ?thesis **by** (simp add: yields-d-def)
qed

lemma EmptyYields:
 $\vdash \text{empty} \text{ yields } f = f$
proof –
have 1: $\vdash \text{empty}; (\neg f) = (\neg f)$ **by** (rule EmptyChop)
hence 2: $\vdash (\neg (\text{empty}; (\neg f))) = f$ **by** auto
from 2 **show** ?thesis **by** (simp add: yields-d-def)
qed

lemma *NextYields*:

$\vdash (\bigcirc f) \text{ yields } g = \text{wnext } (f \text{ yields } g)$

proof –

have 1: $\vdash (\bigcirc f); (\neg g) = \bigcirc(f; (\neg g))$ **by** (rule *NextChop*)

hence 2: $\vdash (\neg ((\bigcirc f); (\neg g))) = (\neg (\bigcirc(f; (\neg g))))$ **by** *auto*

hence 3: $\vdash (\bigcirc f) \text{ yields } g = (\neg (\bigcirc(f; (\neg g))))$ **by** (simp add: *yields-d-def*)

have 4: $\vdash (\neg (\bigcirc(f; (\neg g)))) = \text{wnext } (\neg (f; (\neg g)))$ **by** (auto simp: *wnext-d-def*)

have 5: $\vdash (\bigcirc f) \text{ yields } g = \text{wnext } (\neg (f; (\neg g)))$ **using** 3 4 **by** *fastforce*

from 5 **show** ?thesis **by** (simp add: *yields-d-def*)

qed

lemma *SkipChopEqvNext*:

$\vdash \text{skip}; f = \bigcirc f$

by (simp add: *next-d-def*)

lemma *SkipYieldsEqvWeakNext*:

$\vdash \text{skip} \text{ yields } f = \text{wnext } f$

proof –

have 1: $\vdash \text{skip}; (\neg f) = \bigcirc(\neg f)$ **by** (rule *SkipChopEqvNext*)

hence 2: $\vdash (\neg (\text{skip}; (\neg f))) = (\neg (\bigcirc(\neg f)))$ **by** *auto*

have 3: $\vdash (\neg (\bigcirc(\neg f))) = \text{wnext } f$ **by** (auto simp: *wnext-d-def*)

have 4: $\vdash (\neg (\text{skip}; (\neg f))) = \text{wnext } f$ **using** 2 3 **by** *fastforce*

from 4 **show** ?thesis **by** (simp add: *yields-d-def*)

qed

lemma *NextImpSkipYields*:

$\vdash \bigcirc f \longrightarrow \text{skip} \text{ yields } f$

proof –

have 1: $\vdash \bigcirc f \longrightarrow \text{wnext } f$ **using** *WnextEqvEmptyOrNext* **by** *fastforce*

have 2: $\vdash \text{skip} \text{ yields } f = \text{wnext } f$ **by** (rule *SkipYieldsEqvWeakNext*)

from 1 2 **show** ?thesis **by** *fastforce*

qed

lemma *MoreEqvSkipChopTrue*:

$\vdash \text{more} = \text{skip}; \# \text{True}$

proof –

have 1: $\vdash \text{skip}; \# \text{True} = \bigcirc \# \text{True}$ **by** (rule *SkipChopEqvNext*)

hence 2: $\vdash \bigcirc \# \text{True} = \text{skip}; \# \text{True}$ **by** *auto*

from 2 **show** ?thesis **by** (simp add: *more-d-def*)

qed

lemma *MoreChopImpMore*:

$\vdash \text{more}; f \longrightarrow \text{more}$

proof –

have 1: $\vdash (\bigcirc \# \text{True}); f = \bigcirc(\# \text{True}; f)$ **by** (rule *NextChop*)

have 2: $\vdash \bigcirc(\# \text{True}; f) \longrightarrow \text{more}$ **by** (simp add: *NextImpNext more-d-def*)

have 3: $\vdash (\bigcirc \# \text{True}; f) \longrightarrow \text{more}$ **using** 1 2 **by** *fastforce*

from 3 **show** ?thesis **by** (metis *more-d-def*)

qed

lemma *FmoreChopImpFmore*:

$\vdash fmore ; (f \wedge finite) \longrightarrow fmore$

proof –

have 1: $\vdash fmore ; (f \wedge finite) = \bigcirc(finite ; (f \wedge finite))$

using *FmoreEqvSkipChopFinite* **by** (*metis NextChop inteq-reflection next-d-def*)

have 2: $\vdash \bigcirc(finite ; (f \wedge finite)) \longrightarrow fmore$

by (*metis ChopAndB FiniteChopFiniteEqvFinite FmoreEqvSkipChopFinite RightChopImpChop inteq-reflection next-d-def*)

have 3: $\vdash (\bigcirc finite ; (f \wedge finite)) \longrightarrow fmore$ **using** 1 2

by (*metis FmoreEqvSkipChopFinite inteq-reflection next-d-def*)

from 1 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *ChopMoreImpMore*:

$\vdash f ; more \longrightarrow more$

proof –

have 1: $\vdash (f \wedge finite) ; more \longrightarrow \Diamond more$

by (*rule FiniteChopImpDiamond*)

have 11: $\vdash (f \wedge inf) ; more \longrightarrow more$

by (*metis AndInfChopEqvAndInf MoreAndInfEqvInf Prop11 Prop12 lift-imp-trans*)

have 2: $\vdash \Diamond more \longrightarrow more$

by (*metis FiniteChopMoreEqvMore NowImpDiamond inteq-reflection sometimes-d-def*)

have 3: $\vdash (f \wedge finite) ; more \longrightarrow more$

using 1 2 **by** *fastforce*

have 4: $\vdash f = ((f \wedge finite) \vee (f \wedge inf))$

by (*simp add: OrFiniteInf*)

hence 5: $\vdash f ; more = ((f \wedge finite) ; more \vee (f \wedge inf) ; more)$

by (*simp add: OrChopEqvRule*)

from 11 3 5 **show** *?thesis* **by** *fastforce*

qed

lemma *MoreChopEqvNextDiamond*:

$\vdash fmore ; f = \bigcirc(\Diamond f)$

proof –

have 1: $\vdash fmore ; f = (\bigcirc finite) ; f$

by (*simp add: FmoreEqvSkipChopFinite LeftChopEqvChop next-d-def*)

have 2: $\vdash (\bigcirc finite) ; f = \bigcirc(finite ; f)$

by (*rule NextChop*)

have 3: $\vdash fmore ; f = \bigcirc(finite ; f)$

using 1 2 **by** *fastforce*

from 3 **show** *?thesis* **by** (*simp add: sometimes-d-def*)

qed

lemma *WeakNextBoxImpMoreYields*:

$\vdash fmore \text{ yields } f = wnext(\Box f)$

proof –

have 1: $\vdash fmore ; (\neg f) = \bigcirc(\Diamond (\neg f))$ **by** (*rule MoreChopEqvNextDiamond*)

have 2: $\vdash \bigcirc(\Diamond (\neg f)) = \bigcirc(\neg(\Box f))$ **by** (*auto simp: always-d-def*)

have 3: $\vdash \bigcirc(\neg(\Box f)) = (\neg (wnext(\Box f)))$ **by** (*auto simp: wnext-d-def*)

have 4: $\vdash fmore ; (\neg f) = (\neg(fmore \text{ yields } f))$ **by** (*simp add: yields-d-def*)

from 1 2 3 4 show ?thesis by fastforce
qed

lemma *NotEqvYieldsMore*:

$\vdash (\neg f) = f \text{ yields more}$

proof –

have 1: $\vdash f; \text{ empty} = f$ by (rule *ChopEmpty*)

hence 2: $\vdash (\neg (f; \text{ empty})) = (\neg f)$ by auto

have 3: $\vdash \text{ empty} = (\neg \text{ more})$ by (auto simp: *empty-d-def*)

hence 4: $\vdash f; \text{ empty} = f; (\neg \text{ more})$ by (rule *RightChopEqvChop*)

hence 5: $\vdash (\neg (f; \text{ empty})) = (\neg (f; (\neg \text{ more})))$ by auto

have 6: $\vdash (\neg f) = (\neg (f; (\neg \text{ more})))$ using 2 5 by fastforce

from 6 show ?thesis by (metis *yields-d-def*)

qed

lemma *LeftChopImpMoreRule*:

assumes $\vdash f \longrightarrow \text{ more}$

shows $\vdash f; g \longrightarrow \text{ more}$

proof –

have 1: $\vdash f \longrightarrow \text{ more}$ using assms by auto

hence 2: $\vdash f; g \longrightarrow \text{ more} ; g$ by (rule *LeftChopImpChop*)

have 3: $\vdash \text{ more} ; g \longrightarrow \text{ more}$ by (rule *MoreChopImpMore*)

from 2 3 show ?thesis using *lift-imp-trans* by blast

qed

lemma *LeftChopImpFMoreRule*:

assumes $\vdash f \longrightarrow \text{ fmore}$

shows $\vdash f; (g \wedge \text{ finite}) \longrightarrow \text{ fmore}$

proof –

have 1: $\vdash f \longrightarrow \text{ fmore}$ using assms by auto

hence 2: $\vdash f; (g \wedge \text{ finite}) \longrightarrow \text{ fmore} ; (g \wedge \text{ finite})$ by (rule *LeftChopImpChop*)

have 3: $\vdash \text{ fmore} ; (g \wedge \text{ finite}) \longrightarrow \text{ fmore}$ using *FmoreChopImpFmore* by fastforce

from 2 3 show ?thesis using *lift-imp-trans* by blast

qed

lemma *RightChopImpMoreRule*:

assumes $\vdash g \longrightarrow \text{ more}$

shows $\vdash f; g \longrightarrow \text{ more}$

proof –

have 1: $\vdash g \longrightarrow \text{ more}$ using assms by auto

hence 2: $\vdash f; g \longrightarrow f; \text{ more}$ by (rule *RightChopImpChop*)

have 3: $\vdash f; \text{ more} \longrightarrow \text{ more}$ by (rule *ChopMoreImpMore*)

from 2 3 show ?thesis using *lift-imp-trans* by blast

qed

lemma *NotDiEqvBiNot*:

$\vdash (\neg (di f)) = bi (\neg f)$

proof –

have 1: $\vdash f = (\neg \neg f)$ by auto

hence 2: $\vdash di f = di (\neg \neg f)$ by (rule *DiEqvDi*)

hence 3: $\vdash (\neg (di\ f)) = (\neg (di\ (\neg \neg\ f)))$ **by** *auto*
 from 3 **show** *?thesis* **by** (*simp add: bi-d-def*)
qed

lemma *ChopImpDi*:
 $\vdash f; g \longrightarrow di\ f$
proof –
 have 1: $\vdash g \longrightarrow \#True$ **by** *auto*
 hence 2: $\vdash f; g \longrightarrow f; \#True$ **by** (*rule RightChopImpChop*)
 from 2 **show** *?thesis* **by** (*simp add: di-d-def*)
qed

lemma *TrueEqvTrueChopTrue*:
 $\vdash \#True = \#True; \#True$
proof –
 have 1: $\vdash \#True; \#True \longrightarrow \#True$ **by** *auto*
 have 2: $\vdash \#True \longrightarrow di\ \#True$ **by** (*rule DiIntro*)
 hence 3: $\vdash \#True \longrightarrow \#True; \#True$ **by** (*simp add: di-d-def*)
 from 1 3 **show** *?thesis* **by** *auto*
qed

lemma *DiEqvDiDi*:
 $\vdash di\ f = di\ (di\ f)$
proof –
 have 1: $\vdash \#True = \#True; \#True$ **by** (*rule TrueEqvTrueChopTrue*)
 hence 2: $\vdash f; \#True = f; (\#True; \#True)$ **by** (*rule RightChopEqvChop*)
 have 3: $\vdash f; (\#True; \#True) = (f; \#True); \#True$ **by** (*rule ChopAssoc*)
 have 4: $\vdash f; \#True = (f; \#True); \#True$ **using** 2 3 **by** *fastforce*
 from 4 **show** *?thesis* **by** (*metis di-d-def*)
qed

lemma *BiEqvBiBi*:
 $\vdash bi\ f = bi\ (bi\ f)$
proof –
 have 1: $\vdash di\ (\neg\ f) = di\ (di\ (\neg\ f))$ **by** (*rule DiEqvDiDi*)
 have 2: $\vdash di\ (\neg\ f) = (\neg\ (bi\ f))$ **by** (*rule DiNotEqvNotBi*)
 hence 3: $\vdash di\ (di\ (\neg\ f)) = di\ (\neg\ (bi\ f))$ **by** (*rule DiEqvDi*)
 have 4: $\vdash di\ (\neg\ f) = di\ (\neg\ (bi\ f))$ **using** 1 3 **by** *fastforce*
 hence 5: $\vdash (\neg\ (di\ (\neg\ f))) = (\neg\ (di\ (\neg\ (bi\ f))))$ **by** *fastforce*
 from 5 **show** *?thesis* **by** (*metis bi-d-def*)
qed

lemma *DiOrEqv*:
 $\vdash di\ (f \vee g) = (di\ f \vee di\ g)$
proof –
 have 1: $\vdash (f \vee g); \#True = (f; \#True \vee g; \#True)$ **by** (*rule OrChopEqv*)
 from 1 **show** *?thesis* **by** (*simp add: di-d-def*)
qed

lemma *DiAndA*:

$\vdash \text{di } (f \wedge g) \longrightarrow \text{di } f$

proof –

have 1: $\vdash (f \wedge g); \#True \longrightarrow f; \#True$ **by** (rule AndChopA)
from 1 **show** ?thesis **by** (simp add: di-d-def)

qed

lemma DiAndB:

$\vdash \text{di } (f \wedge g) \longrightarrow \text{di } g$

proof –

have 1: $\vdash (f \wedge g); \#True \longrightarrow g; \#True$ **by** (rule AndChopB)
from 1 **show** ?thesis **by** (simp add: di-d-def)

qed

lemma DiAndImpAnd:

$\vdash \text{di } (f \wedge g) \longrightarrow \text{di } f \wedge \text{di } g$

proof –

have 1: $\vdash \text{di } (f \wedge g) \longrightarrow \text{di } f$ **by** (rule DiAndA)
have 2: $\vdash \text{di } (f \wedge g) \longrightarrow \text{di } g$ **by** (rule DiAndB)
from 1 2 **show** ?thesis **by** fastforce

qed

lemma DiSkipEqvMore:

$\vdash \text{di } \text{skip} = \text{more}$

proof –

have 1: $\vdash \text{skip}; \#True = \bigcirc \#True$ **by** (rule SkipChopEqvNext)
have 2: $\vdash \bigcirc \#True = \text{more}$ **by** (auto simp: more-d-def)
have 3: $\vdash \text{skip}; \#True = \text{more}$ **using** 1 2 **by** fastforce
from 3 **show** ?thesis **by** (simp add: di-d-def)

qed

lemma DiMoreEqvMore:

$\vdash \text{di } \text{more} = \text{more}$

proof –

have 1: $\vdash \text{di } (\bigcirc \#True) = \bigcirc(\text{di } \#True)$
by (rule DiNext)
have 2: $\vdash \bigcirc(\text{di } \#True) \longrightarrow \text{more}$
by (metis 1 ChopImpDi TrueEqvTrueChopTrue di-d-def int-eq more-d-def)
have 3: $\vdash \text{di } (\bigcirc \#True) \longrightarrow \text{more}$
using 1 2 **by** fastforce
hence 4: $\vdash \text{di } \text{more} \longrightarrow \text{more}$
by (simp add: more-d-def)
have 5: $\vdash \text{more} \longrightarrow \text{di } \text{more}$
by (rule ImpDi)
from 4 5 **show** ?thesis **by** fastforce

qed

lemma DiIfEqvRule:

assumes $\vdash f = \text{if}_i (\text{init } w) \text{ then } g \text{ else } h$

shows $\vdash \text{di } f = \text{if}_i (\text{init } w) \text{ then } (\text{di } g) \text{ else } (\text{di } h)$

proof –

have 1: $\vdash f = \text{if}_i (\text{init } w) \text{ then } g \text{ else } h$ **using** *assms* **by** *auto*
hence 2: $\vdash f; \# \text{True} = \text{if}_i (\text{init } w) \text{ then } (g; \# \text{True}) \text{ else } (h; \# \text{True})$ **by** (*rule IfChopEqvRule*)
from 2 **show** *?thesis* **by** (*simp add: di-d-def*)
qed

lemma *DiEmpty*:

$\vdash \text{di empty}$

proof –

have 1: $\vdash \# \text{True}$ **by** *auto*
have 2: $\vdash \text{empty}; \# \text{True} = \# \text{True}$ **by** (*rule EmptyChop*)
have 3: $\vdash \text{empty}; \# \text{True}$ **using** 1 2 **by** *auto*
from 3 **show** *?thesis* **by** (*simp add: di-d-def*)
qed

lemma *DaNotEqvNotBa*:

$\vdash \text{da } (\neg f) = (\neg (\text{ba } f))$

proof –

have 1: $\vdash \text{ba } f = (\neg (\text{da } (\neg f)))$ **by** (*simp add: ba-d-def*)
from 1 **show** *?thesis* **by** *fastforce*
qed

lemma *DaEqvDa*:

assumes $\vdash f = g$

shows $\vdash \text{da } f = \text{da } g$

using *assms* **using** *int-eq* **by** *force*

lemma *DaEqvNotBaNot*:

$\vdash \text{da } f = (\neg (\text{ba } (\neg f)))$

proof –

have 1: $\vdash \text{ba } (\neg f) = (\neg (\text{da } (\neg \neg f)))$ **by** (*simp add: ba-d-def*)
hence 2: $\vdash \text{da } (\neg \neg f) = (\neg (\text{ba } (\neg f)))$ **by** *fastforce*
have 3: $\vdash f = (\neg \neg f)$ **by** *simp*
hence 4: $\vdash \text{da } f = \text{da } (\neg \neg f)$ **by** (*rule DaEqvDa*)
from 2 4 **show** *?thesis* **by** *simp*
qed

lemma *BaElim*:

$\vdash \text{ba } f \longrightarrow f$

proof –

have 1: $\vdash \text{ba } f = \Box(\text{bi } f)$ **by** (*rule BaEqvBtBi*)
have 2: $\vdash \text{bi } f \longrightarrow f$ **by** (*rule BiElim*)
hence 3: $\vdash \Box(\text{bi } f \longrightarrow f)$ **by** (*rule BoxGen*)
have 4: $\vdash \Box(\text{bi } f \longrightarrow f) \longrightarrow \Box(\text{bi } f) \longrightarrow \Box f$ **by** (*rule BoxImpDist*)
have 5: $\vdash \Box(\text{bi } f) \longrightarrow \Box f$ **using** 3 4 *MP* **by** *fastforce*
have 6: $\vdash \Box f \longrightarrow f$ **by** (*rule BoxElim*)
from 1 5 6 **show** *?thesis* **using** *BaImpBt lift-imp-trans* **by** *metis*
qed

lemma *DaIntro*:

$\vdash f \longrightarrow \text{da } f$

proof –
have 1: $\vdash ba (\neg f) \longrightarrow (\neg f)$ **by** (*rule BaElim*)
hence 2: $\vdash \neg \neg f \longrightarrow \neg (ba (\neg f))$ **by** *fastforce*
have 3: $\vdash f = (\neg \neg f)$ **by** *simp*
have 4: $\vdash da f = (\neg (ba (\neg f)))$ **by** (*rule DaEqvNotBaNot*)
from 2 3 4 **show** *?thesis* **by** *fastforce*
qed

lemma *BaGen*:
assumes $\vdash f$
shows $\vdash ba f$
proof –
have 1: $\vdash f$ **using** *assms* **by** *auto*
hence 2: $\vdash \Box f$ **by** (*rule BoxGen*)
hence 3: $\vdash bi(\Box f)$ **by** (*rule BiGen*)
have 4: $\vdash ba f = bi(\Box f)$ **by** (*rule BaEqvBiBt*)
from 3 4 **show** *?thesis* **by** *fastforce*
qed

lemma *BaImpDist*:
 $\vdash ba (f \longrightarrow g) \longrightarrow ba f \longrightarrow ba g$
proof –
have 1: $\vdash bi (f \longrightarrow g) \longrightarrow (bi f \longrightarrow bi g)$
by (*rule BiImpDist*)
hence 2: $\vdash \Box (bi (f \longrightarrow g) \longrightarrow (bi f \longrightarrow bi g))$
by (*rule BoxGen*)
have 3: $\vdash \Box (bi (f \longrightarrow g) \longrightarrow (bi f \longrightarrow bi g))$
 \longrightarrow
 $(\Box (bi (f \longrightarrow g)) \longrightarrow (\Box (bi f) \longrightarrow \Box (bi g)))$
by (*meson 2 BoxImpDist MP lift-imp-trans Prop01 Prop05 Prop09*)
have 4: $\vdash \Box (bi (f \longrightarrow g)) \longrightarrow (\Box (bi f) \longrightarrow \Box (bi g))$
using 2 3 *MP* **by** *fastforce*
have 5: $\vdash ba (f \longrightarrow g) = \Box (bi (f \longrightarrow g))$
by (*rule BaEqvBtBi*)
have 6: $\vdash ba f = \Box (bi f)$
by (*rule BaEqvBtBi*)
have 7: $\vdash ba g = \Box (bi g)$
by (*rule BaEqvBtBi*)
from 4 5 6 7 **show** *?thesis* **by** *fastforce*
qed

lemma *BiAndEqv*:
 $\vdash bi (f \wedge g) = (bi f \wedge bi g)$
proof –
have 1: $\vdash di (\neg f \vee \neg g) = (di (\neg f) \vee di (\neg g))$
by (*simp add: DiOrEqv*)
have 2: $\vdash (\neg (di (\neg f \vee \neg g))) = (\neg di (\neg f) \wedge \neg di (\neg g))$
using 1 **by** *auto*
have 3: $\vdash (f \wedge g) = (\neg (\neg f \vee \neg g))$
by *fastforce*

have 4: $\vdash bi (f \wedge g) = (\neg (di (\neg f \vee \neg g)))$
unfolding *bi-d-def* **using** 3 **by** (*metis int-simps*(4) *inteq-reflection*)
from 2 4 **show** ?thesis **unfolding** *bi-d-def* **by** (*metis inteq-reflection*)
qed

lemma *BaAndEqv*:

$\vdash ba (f \wedge g) = (ba f \wedge ba g)$

proof –

have 1: $\vdash ba (f \wedge g) = \Box(bi (f \wedge g))$

by (*rule BaEqvBtBi*)

have 2: $\vdash bi (f \wedge g) = (bi f \wedge bi g)$

by (*simp add: BiAndEqv*)

hence 3: $\vdash \Box(bi (f \wedge g)) = \Box(bi f \wedge bi g)$

using *BoxEqvBox* **by** *blast*

have 4: $\vdash \Box(bi f \wedge bi g) = (\Box(bi f) \wedge \Box(bi g))$

by (*metis 2 BoxAndBoxEqvBoxRule inteq-reflection*)

have 5: $\vdash ba f = \Box(bi f)$

by (*rule BaEqvBtBi*)

have 6: $\vdash ba g = \Box(bi g)$

by (*rule BaEqvBtBi*)

from 1 3 4 5 6 **show** ?thesis **by** *fastforce*

qed

lemma *BaImpBaEqvBa*:

$\vdash ba (f \longrightarrow g) \longrightarrow (ba f \longrightarrow ba g)$

proof –

have 1: $\vdash ba (f \longrightarrow g) \longrightarrow ba f \longrightarrow ba g$ **by** (*rule BaImpDist*)

have 2: $\vdash ba (g \longrightarrow f) \longrightarrow ba g \longrightarrow ba f$ **by** (*rule BaImpDist*)

have 25: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** *fastforce*

have 3: $\vdash ba (f = g) = ba ((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** (*metis 25 BaAndEqv inteq-reflection*)

have 4: $\vdash ba ((f \longrightarrow g) \wedge (g \longrightarrow f)) = (ba((f \longrightarrow g)) \wedge ba((g \longrightarrow f)))$ **by** (*rule BaAndEqv*)

have 5: $\vdash ((ba f \longrightarrow ba g) \wedge (ba g \longrightarrow ba f)) = (ba f \longrightarrow ba g)$ **by** *auto*

from 1 2 3 4 5 **show** ?thesis **by** *fastforce*

qed

lemma *BaImpBa*:

assumes $\vdash f \longrightarrow g$

shows $\vdash ba f \longrightarrow ba g$

using *BaGen BaImpDist MP assms* **by** *metis*

lemma *BaEqvBa*:

assumes $\vdash f = g$

shows $\vdash ba f = ba g$

using *BaGen BaImpBaEqvBa MP assms* **by** *metis*

lemma *DaImpDa*:

assumes $\vdash f \longrightarrow g$

shows $\vdash da f \longrightarrow da g$

using *assms* **by** (*metis DaEqvDtDi DiAndB DiamondImpDiamond inteq-reflection Prop10*)

lemma *DiamondEqvDiamondDiamond*:

$\vdash \Diamond f = \Diamond (\Diamond f)$

proof –

have 1: $\vdash \Diamond (\Diamond f) = \text{finite};(\text{finite};f)$

by (*simp add: sometimes-d-def*)

have 2: $\vdash \text{finite};(\text{finite};f) = (\text{finite};\text{finite});f$

by (*rule ChopAssoc*)

have 3: $\vdash (\text{finite};\text{finite});f = \text{finite};f$

by (*simp add: LeftChopEqvChop FiniteChopFiniteEqvFinite*)

have 4: $\vdash \text{finite};f = \Diamond f$

by (*simp add: sometimes-d-def*)

from 1 2 3 4 **show** ?thesis **by** *fastforce*

qed

lemma *DaEqvDaDa*:

$\vdash da\ f = da\ (da\ f)$

proof –

have 1: $\vdash da\ f = \Diamond (di\ f)$

by (*rule DaEqvDtDi*)

have 2: $\vdash di\ f = (di\ (di\ f))$

by (*rule DiEqvDiDi*)

hence 3: $\vdash \Diamond (di\ f) = \Diamond (di\ (di\ f))$

by (*rule DiamondEqvDiamond*)

have 4: $\vdash \Diamond (di\ f) = \Diamond (\Diamond (di\ (di\ f)))$

using *DiamondEqvDiamondDiamond DiEqvDiDi* **using** 3 **by** *fastforce*

have 5: $\vdash \Diamond (di\ (di\ f)) = di\ (\Diamond (di\ f))$

by (*rule DtDiEqvDiDt*)

hence 6: $\vdash \Diamond (\Diamond (di\ (di\ f))) = \Diamond (di\ (\Diamond (di\ f)))$

by (*rule DiamondEqvDiamond*)

have 7: $\vdash da\ f = \Diamond (di\ (\Diamond (di\ f)))$

using 1 3 4 6 **by** *fastforce*

have 8: $\vdash da\ (\Diamond (di\ f)) = \Diamond (di\ (\Diamond (di\ f)))$

by (*rule DaEqvDtDi*)

have 9: $\vdash da\ (da\ f) = da\ (\Diamond (di\ f))$

using 1 **by** (*rule DaEqvDa*)

from 7 8 9 **show** ?thesis **by** *fastforce*

qed

lemma *BaEqvBaBa*:

$\vdash ba\ f = ba\ (ba\ f)$

proof –

have 1: $\vdash da\ (\neg f) = da\ (da\ (\neg f))$ **by** (*rule DaEqvDaDa*)

have 2: $\vdash da\ (da\ (\neg f)) = (\neg (ba\ (\neg (da\ (\neg f)))))$ **by** (*rule DaEqvNotBaNot*)

have 3: $\vdash (\neg (da\ (da\ (\neg f)))) = ba\ (\neg (da\ (\neg f)))$ **by** (*auto simp: ba-d-def*)

have 4: $\vdash (\neg (da\ (\neg f))) = ba\ (\neg (da\ (\neg f)))$ **using** 1 2 3 **by** *fastforce*

from 4 **show** ?thesis **by** (*metis ba-d-def*)

qed

lemma *BaLeftChopImpChop*:

$\vdash ba\ (f \longrightarrow f1) \longrightarrow f; g \longrightarrow f1; g$

proof –
have 1: $\vdash \text{ba } (f \longrightarrow f1) \longrightarrow \text{bi } (f \longrightarrow f1)$ **by** (rule *BaImpBi*)
have 2: $\vdash \text{bi } (f \longrightarrow f1) \longrightarrow f; g \longrightarrow f1; g$ **by** (rule *BiChopImpChop*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *BaRightChopImpChop*:

$\vdash \text{ba } (g \longrightarrow g1) \longrightarrow f; g \longrightarrow f; g1$

proof –
have 1: $\vdash \text{ba } (g \longrightarrow g1) \longrightarrow \Box(g \longrightarrow g1)$ **by** (rule *BaImpBt*)
have 2: $\vdash \Box(g \longrightarrow g1) \longrightarrow f; g \longrightarrow f; g1$ **by** (rule *BoxChopImpChop*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *ChopAndBaImport*:

$\vdash (f; f1) \wedge \text{ba } g \longrightarrow (f \wedge g); (f1 \wedge g)$

proof –
have 1: $\vdash \text{ba } g \wedge (f; f1) \longrightarrow (g \wedge f); (g \wedge f1)$ **by** (rule *BaAndChopImport*)
have 2: $\vdash (g \wedge f); (g \wedge f1) = (f \wedge g); (f1 \wedge g)$ **by** (rule *AndChopAndCommute*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *BaAndChopImportA*:

$\vdash \text{ba } f \wedge g; g1 \longrightarrow (f \wedge g); g1$

by (meson *BaAndChopImport ChopAndB lift-imp-trans*)

lemma *BaAndChopImportB*:

$\vdash \text{ba } f \wedge g; g1 \longrightarrow (f \wedge g); (\text{ba } f \wedge g1)$

proof –
have 1: $\vdash \text{ba } f = \text{ba } (\text{ba } f)$
by (simp add: *BaEqvBaBa*)
have 2: $\vdash \text{ba } (\text{ba } f) \wedge g; g1 \longrightarrow g; (\text{ba } f \wedge g1)$
by (metis *AndChopB BaAndChopImport lift-imp-trans*)
have 3: $\vdash \text{ba } f \wedge g; (\text{ba } f \wedge g1) \longrightarrow (f \wedge g); (\text{ba } f \wedge g1)$
by (simp add: *BaAndChopImportA*)
from 1 2 3 **show** ?thesis **by** fastforce
qed

lemma *BaImpBaImpBaAnd*:

$\vdash \text{ba } h \longrightarrow \text{ba}(g \longrightarrow \text{ba } h \wedge g)$

proof –
have 1: $\vdash \text{ba } h \longrightarrow (g \longrightarrow \text{ba } h \wedge g)$ **by** fastforce
hence 2: $\vdash \text{ba}(\text{ba } h) \longrightarrow \text{ba}(g \longrightarrow \text{ba } h \wedge g)$ **by** (rule *BaImpBa*)
have 3: $\vdash \text{ba } h = \text{ba}(\text{ba } h)$ **by** (rule *BaEqvBaBa*)
from 2 3 **show** ?thesis **by** fastforce
qed

lemma *BaChopImpChopBa*:

$\vdash \text{ba } f \longrightarrow g; g1 \longrightarrow g; ((\text{ba } f) \wedge g1)$

proof –

have 1: $\vdash ba\ f \longrightarrow ba\ (g1 \longrightarrow (ba\ f) \wedge g1)$ **by** (rule *BaImpBaImpBaAnd*)
have 2: $\vdash ba\ (g1 \longrightarrow ba\ f \wedge g1) \longrightarrow g; g1 \longrightarrow g; (ba\ f \wedge g1)$ **by** (rule *BaRightChopImpChop*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *DiNotBaImpNotBa*:

$\vdash di\ (\neg (ba\ f)) \longrightarrow \neg (ba\ f)$

proof –

have 1: $\vdash ba\ f = ba\ (ba\ f)$ **by** (rule *BaEqvBaBa*)
have 2: $\vdash ba\ (ba\ f) \longrightarrow bi\ (ba\ f)$ **by** (rule *BaImpBi*)
have 3: $\vdash ba\ f \longrightarrow bi\ (ba\ f)$ **using** 1 2 **by** fastforce
hence 4: $\vdash ba\ f \longrightarrow \neg (di\ (\neg (ba\ f)))$ **by** (simp add: *bi-d-def*)
from 4 **show** ?thesis **by** fastforce
qed

lemma *NotBaChopImpNotBa*:

$\vdash (\neg (ba\ f)); g \longrightarrow \neg (ba\ f)$

proof –

have 1: $\vdash (\neg (ba\ f)); g \longrightarrow di\ (\neg (ba\ f))$ **by** (rule *ChopImpDi*)
have 2: $\vdash di\ (\neg (ba\ f)) \longrightarrow \neg (ba\ f)$ **by** (rule *DiNotBaImpNotBa*)
from 1 2 **show** ?thesis **using** lift-imp-trans **by** blast
qed

lemma *DiamondFinImpFin*:

$\vdash \Diamond (fin\ f) \longrightarrow fin\ f$

proof –

have 1: $\vdash fin\ f = \#True;(f \wedge empty)$
by (rule *FinEqvTrueChopAndEmpty*)
hence 2: $\vdash \Diamond (fin\ f) = finite;(\#True;(f \wedge empty))$
by (metis *FiniteChopFiniteEqvFinite LeftChopEqvChop inteq-reflection sometimes-d-def*)
have 3: $\vdash finite;(\#True;(f \wedge empty)) = (finite;\#True);(f \wedge empty)$
by (rule *ChopAssoc*)
have 4: $\vdash (finite;\#True);(f \wedge empty) \longrightarrow \#True;(f \wedge empty)$
using 1 2 3 *DiamondFin* **by** fastforce
from 1 2 3 4 **show** ?thesis **by** fastforce
qed

lemma *ChopFinImpFin*:

$\vdash (f \wedge finite); fin\ (init\ w) \longrightarrow fin\ (init\ w)$

proof –

have 1: $\vdash (f \wedge finite); fin\ (init\ w) \longrightarrow \Diamond (fin\ (init\ w))$ **by** (rule *FiniteChopImpDiamond*)
have 2: $\vdash \Diamond (fin\ (init\ w)) \longrightarrow fin\ (init\ w)$ **by** (rule *DiamondFinImpFin*)
from 1 2 **show** ?thesis **using** lift-imp-trans **by** blast
qed

lemma *FiniteRightChopEqvChop*:

assumes $\vdash finite \longrightarrow g = g1$

shows $\vdash finite \longrightarrow f;g = f;g1$

using assms **by** (auto simp add: *Valid-def itl-defs*)

lemma *FinImpYieldsFin*:

$\vdash \text{fin } (\text{init } w) \wedge \text{finite} \longrightarrow (f \wedge \text{finite}) \text{ yields } (\text{fin } (\text{init } w) \wedge \text{finite})$

proof –

have 1: $\vdash (f \wedge \text{finite}); (\text{fin } (\text{init } (\neg w)) \wedge \text{finite}) \longrightarrow (\text{fin } (\text{init } (\neg w)) \wedge \text{finite})$

by (*metis* (*no-types*, *lifting*) *ChopAndB FiniteChopEqvDiamond FiniteChopFinExportA FiniteChopFiniteEqvFinite FiniteChopImpDiamond Prop12 inteq-reflection lift-and-com lift-imp-trans*)

have 2: $\vdash \text{finite} \longrightarrow (\neg (\text{fin } (\text{init } w) \wedge \text{finite})) = (\text{fin } (\text{init } (\neg w)) \wedge \text{finite})$

using *FinNotStateEqvNotFinState* **by** *fastforce*

hence 3: $\vdash \text{finite} \longrightarrow (f \wedge \text{finite}); (\neg (\text{fin } (\text{init } w) \wedge \text{finite})) = (f \wedge \text{finite}); (\text{fin } (\text{init } (\neg w)) \wedge \text{finite})$

using *FiniteRightChopEqvChop*[*of LIFT*($\neg (\text{fin } (\text{init } w) \wedge \text{finite}))$]
LIFT($\text{fin } (\text{init } (\neg w)) \wedge \text{finite}$) *LIFT*($f \wedge \text{finite}$)]

by *blast*

have 4: $\vdash (f \wedge \text{finite}); (\neg (\text{fin } (\text{init } w) \wedge \text{finite})) \longrightarrow (\neg (\text{fin } (\text{init } w) \wedge \text{finite}))$

using 1 2 3 **by** *fastforce*

hence 5: $\vdash \text{fin } (\text{init } w) \wedge \text{finite} \longrightarrow \neg ((f \wedge \text{finite}); (\neg (\text{fin } (\text{init } w) \wedge \text{finite})))$

by *fastforce*

from 5 **show** *?thesis* **by** (*simp add: yields-d-def*)

qed

lemma *ChopAndFin*:

$\vdash ((f; g) \wedge (\text{fin } (\text{init } w) \wedge \text{finite})) = (f \wedge \text{finite}); (g \wedge (\text{fin } (\text{init } w) \wedge \text{finite}))$

proof –

have 1: $\vdash \text{fin } (\text{init } w) \wedge \text{finite} \longrightarrow (f \wedge \text{finite}) \text{ yields } (\text{fin } (\text{init } w) \wedge \text{finite})$

by (*rule FinImpYieldsFin*)

have 10: $\vdash ((f; g) \wedge (\text{fin } (\text{init } w) \wedge \text{finite})) = (((f \wedge \text{finite}); (g \wedge \text{finite})) \wedge \text{fin } (\text{init } w) \wedge \text{finite})$

using *ChopAndFiniteDist*[*of f g*] **by** *auto*

have 2: $\vdash (f; g) \wedge (\text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow ((f \wedge \text{finite}); (g \wedge \text{finite})) \wedge (f \wedge \text{finite}) \text{ yields } (\text{fin } (\text{init } w) \wedge \text{finite})$

using 1 10 **by** *fastforce*

have 3: $\vdash ((f \wedge \text{finite}); (g \wedge \text{finite})) \wedge (f \wedge \text{finite}) \text{ yields } (\text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow (f \wedge \text{finite}); ((g \wedge \text{finite}) \wedge (\text{fin } (\text{init } w) \wedge \text{finite}))$

using *ChopAndYieldsImp* **by** *blast*

have 30: $\vdash ((g \wedge \text{finite}) \wedge (\text{fin } (\text{init } w) \wedge \text{finite})) = (g \wedge \text{fin } (\text{init } w) \wedge \text{finite})$

by *auto*

have 4: $\vdash (f; g) \wedge (\text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow (f \wedge \text{finite}); (g \wedge \text{fin } (\text{init } w) \wedge \text{finite})$

using 2 3 30 **by** (*metis* (*mono-tags*, *lifting*) *inteq-reflection lift-imp-trans*)

have 11: $\vdash (f \wedge \text{finite}); (g \wedge \text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow (f \wedge \text{finite}); (g \wedge \text{finite})$

using *ChopAndA* **by** (*metis* 30 *inteq-reflection*)

have 12: $\vdash (f \wedge \text{finite}); (g \wedge \text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow (f \wedge \text{finite}); (\text{fin } (\text{init } w) \wedge \text{finite})$

by (*rule ChopAndB*)

have 13: $\vdash (f \wedge \text{finite}); (\text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow \Diamond (\text{fin } (\text{init } w) \wedge \text{finite})$

using *FiniteChopImpDiamond* **by** *blast*

have 14: $\vdash \Diamond (\text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow \text{fin } (\text{init } w)$

by (*metis* *ChopAndA DiamondFin inteq-reflection sometimes-d-def*)

have 15: $\vdash (f \wedge \text{finite}); (g \wedge \text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow$

$((f \wedge \text{finite}); (g \wedge \text{finite})) \wedge \text{fin } (\text{init } w)$
using 11 12 13 14 **by** fastforce
from 4 15 **show** ?thesis **by** (metis ChopAndFiniteDist Prop12 int-iffI inteq-reflection)
qed

lemma ChopAndNotFin:

$\vdash (f; g \wedge \neg (\text{fin } (\text{init } w)) \wedge \text{finite}) = (f \wedge \text{finite}); (g \wedge \neg (\text{fin } (\text{init } w)) \wedge \text{finite})$
proof –
have 1: $\vdash (f; g \wedge \text{fin } (\text{init } (\neg w)) \wedge \text{finite}) =$
 $(f \wedge \text{finite}); (g \wedge \text{fin } (\text{init } (\neg w)) \wedge \text{finite})$
by (rule ChopAndFin)
have 2: $\vdash (\text{fin } (\text{init } (\neg w)) \wedge \text{finite}) = (\neg (\text{fin } (\text{init } w))) \wedge \text{finite}$
using FinNotStateEqvNotFinState **by** fastforce
show ?thesis **by** (metis 1 2 int-eq)
qed

lemma FinChopChain:

$\vdash (((\text{init } w) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w1)));$
 $((\text{init } w1) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w2)))$
 $\wedge \text{finite}$
 $\longrightarrow (((\text{init } w) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w2)))$
proof –
have 1: $\vdash (\text{init } w) \wedge \text{finite} \wedge$
 $((\text{init } w) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w1)); ((\text{init } w1) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w2))$
 \longrightarrow
 $((\text{init } w) \wedge \text{finite} \wedge ((\text{init } w) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w1)));$
 $((\text{init } w1) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w2)) \wedge \text{finite}$
using ChopAndFiniteDist StateAndChopImport
by (metis (no-types, opaque-lifting) inteq-reflection lift-and-com)
have 2: $\vdash (\text{init } w) \wedge \text{finite} \wedge ((\text{init } w) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w1)) \longrightarrow \text{fin } (\text{init } w1) \wedge \text{finite}$
by auto
have 3: $\vdash ((\text{init } w) \wedge \text{finite} \wedge ((\text{init } w) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w1)));$
 $((\text{init } w1) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w2)) \wedge \text{finite}$
 \longrightarrow
 $(\text{fin } (\text{init } w1) \wedge \text{finite}); (((\text{init } w1) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w2)) \wedge \text{finite})$
using 2 LeftChopImpChop **by** blast
have 4: $\vdash (\text{fin } (\text{init } w1) \wedge \text{finite}); (((\text{init } w1) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w2)) \wedge \text{finite}) =$
 $\Diamond((\text{init } w1) \wedge ((\text{init } w1) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w2)) \wedge \text{finite})$
using FinChopEqvDiamond **by** blast
have 41: $\vdash ((\text{init } w1) \wedge \text{finite} \wedge ((\text{init } w1) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w2))) \longrightarrow \text{fin } (\text{init } w2)$
by auto
have 42: $\vdash \Diamond((\text{init } w1) \wedge \text{finite} \wedge ((\text{init } w1) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w2))) \longrightarrow \Diamond(\text{fin } (\text{init } w2))$
using 41 DiamondImpDiamond **by** blast
have 5: $\vdash \Diamond(\text{fin } (\text{init } w2)) \longrightarrow \text{fin } (\text{init } w2)$
using DiamondFinImpFin **by** blast
have 6: $\vdash (\text{init } w) \wedge \text{finite} \wedge ((\text{init } w) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w1));$
 $((\text{init } w1) \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w2))$
 $\longrightarrow \text{fin } (\text{init } w2)$
using 1 3 4 5 42
using ChopAndCommute FinChopEqvDiamond **by** fastforce

from 6 show ?thesis by fastforce
qed

lemma ChopRule:

assumes $\vdash (init\ w) \wedge f \wedge finite \longrightarrow fin\ (init\ w1)$
 $\vdash (init\ w1) \wedge f1 \wedge finite \longrightarrow fin\ (init\ w2)$
shows $\vdash (init\ w) \wedge (f; f1) \wedge finite \longrightarrow fin\ (init\ w2)$
proof –
have 1: $\vdash (init\ w) \wedge (f; f1) \wedge finite \longrightarrow ((init\ w) \wedge f \wedge finite); (f1 \wedge finite)$
using *StateAndChopImport*
by (*metis ChopAndFiniteDist inteq-reflection*)
have 2: $\vdash (init\ w) \wedge f \wedge finite \longrightarrow fin\ (init\ w1) \wedge finite$
using *assms* **by** *auto*
hence 3: $\vdash ((init\ w) \wedge f \wedge finite); (f1 \wedge finite) \longrightarrow (fin\ (init\ w1) \wedge finite); (f1 \wedge finite)$
by (*rule LeftChopImpChop*)
have 4: $\vdash (fin\ (init\ w1) \wedge finite); (f1 \wedge finite) = \Diamond((init\ w1) \wedge f1 \wedge finite)$
by (*rule FinChopEqvDiamond*)
have 5: $\vdash (init\ w1) \wedge f1 \wedge finite \longrightarrow fin\ (init\ w2)$
using *assms* **by** *auto*
hence 6: $\vdash \Diamond((init\ w1) \wedge f1 \wedge finite) \longrightarrow \Diamond(fin\ (init\ w2))$
by (*rule DiamondImpDiamond*)
have 7: $\vdash \Diamond(fin\ (init\ w2)) \longrightarrow fin\ (init\ w2)$
using *DiamondFinImpFin* **by** *blast*
from 1 3 4 6 7 **show** ?thesis **by** *fastforce*
qed

lemma ChopRep:

assumes $\vdash (init\ w) \wedge f \wedge finite \longrightarrow f1 \wedge fin\ (init\ w1)$
 $\vdash (init\ w1) \wedge g \wedge finite \longrightarrow g1$
shows $\vdash (init\ w) \wedge (f; g) \wedge finite \longrightarrow ((f1 \wedge finite); g1)$
proof –
have 1: $\vdash (init\ w) \wedge f \wedge finite \longrightarrow ((f1 \wedge finite) \wedge fin\ (init\ w1))$
using *assms* **by** *auto*
hence 2: $\vdash (init\ w) \wedge ((f \wedge finite); (g \wedge finite)) \longrightarrow$
 $((f1 \wedge finite) \wedge fin\ (init\ w1)); (g \wedge finite)$
using *StateAndChopImpChopRule* **by** *blast*
have 3: $\vdash ((f1 \wedge finite) \wedge fin\ (init\ w1)); (g \wedge finite) =$
 $(f1 \wedge finite); ((init\ w1) \wedge (g \wedge finite))$
using *AndFinChopEqvStateAndChop* **by** *blast*
have 4: $\vdash (init\ w1) \wedge g \wedge finite \longrightarrow g1$
using *assms* **by** *auto*
hence 5: $\vdash (f1 \wedge finite); ((init\ w1) \wedge g \wedge finite) \longrightarrow (f1 \wedge finite); g1$
using *RightChopImpChop* **by** *blast*
from 2 3 5 **show** ?thesis **using** *ChopAndFiniteDist* **by** *fastforce*
qed

lemma ChopRepAndFin:

assumes $\vdash (init\ w) \wedge f \wedge finite \longrightarrow f1 \wedge fin\ (init\ w1)$
 $\vdash (init\ w1) \wedge g \wedge finite \longrightarrow g1 \wedge fin\ (init\ w2)$
shows $\vdash (init\ w) \wedge (f; g) \wedge finite \longrightarrow ((f1 \wedge finite); g1) \wedge fin\ (init\ w2)$

proof –

have 1: $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow f1 \wedge \text{fin } (\text{init } w1)$
using *assms* **by** *auto*
have 2: $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1 \wedge \text{fin } (\text{init } w2)$
using *assms* **by** *auto*
have 3: $\vdash (\text{init } w) \wedge (f; g) \wedge \text{finite} \longrightarrow (f1 \wedge \text{finite}); (g1 \wedge \text{fin } (\text{init } w2))$
using 1 2 **by** (*rule ChopRep*)
have 4: $\vdash (f1 \wedge \text{finite}); (g1 \wedge \text{fin } (\text{init } w2)) \longrightarrow (f1 \wedge \text{finite}); g1$
by (*rule ChopAndA*)
have 5: $\vdash (f1 \wedge \text{finite}); (g1 \wedge \text{fin } (\text{init } w2)) \longrightarrow (f1 \wedge \text{finite}); \text{fin } (\text{init } w2)$
by (*rule ChopAndB*)
have 6: $\vdash (f1 \wedge \text{finite}); \text{fin } (\text{init } w2) \longrightarrow \text{fin } (\text{init } w2)$
by (*rule ChopFinImpFin*)
from 1 2 3 4 5 6 **show** ?thesis **by** (*meson Prop12 lift-imp-trans*)
qed

lemma *TrueChopMoreEqvMore*:

$\vdash \# \text{True} ; \text{more} = \text{more}$
by (*metis ChopMoreImpMore EmptyImpFinite FiniteAndEmptyEqvEmpty FiniteChopMoreEqvMore LeftChopImpChop Prop09 int-eq-true int-iffI inteq-reflection*)

lemma *FiniteChopFmoreEqvFmore*:

$\vdash \text{finite}; \text{fmore} = \text{fmore}$
by (*metis TrueChopAndFiniteEqvAndFiniteChopFinite TrueChopMoreEqvMore fmore-d-def inteq-reflection*)

lemma *MoreChopLoop*:

assumes $\vdash f \longrightarrow \text{fmore} ; f$
shows $\vdash \text{finite} \longrightarrow \neg f$

proof –

have 1: $\vdash f \longrightarrow \text{fmore} ; f$
using *assms* **by** *auto*
hence 11: $\vdash \Diamond (f) \longrightarrow \Diamond (\text{fmore}; f)$
using *DiamondImpDiamond* **by** *blast*
have 12: $\vdash \Diamond (\text{fmore}; f) = \text{finite}; (\text{fmore}; f)$
by (*simp add: sometimes-d-def*)
have 13: $\vdash \text{finite}; (\text{fmore}; f) = (\text{finite}; \text{fmore}); f$
by (*rule ChopAssoc*)
have 14: $\vdash \Diamond (\text{fmore}; f) = \text{fmore}; f$
using *FiniteChopFmoreEqvFmore* 12 13 **by** (*metis int-eq*)
have 2: $\vdash \text{fmore} ; f = \bigcirc (\Diamond f)$
using *MoreChopEqvNextDiamond* **by** *blast*
have 3: $\vdash \Diamond (f) \longrightarrow \bigcirc (\Diamond f)$
using 11 14 2 **by** *fastforce*
hence 4: $\vdash \text{finite} \longrightarrow \neg (\Diamond f)$
using *NextLoop* **by** *blast*
have 5: $\vdash \neg (\Diamond f) \longrightarrow \neg f$
using *NowImpDiamond* **by** *fastforce*
from 4 5 **show** ?thesis **using** *lift-imp-trans* **by** *blast*
qed

lemma *MoreChopContra*:

assumes $\vdash f \wedge \neg g \longrightarrow (fmore ; (f \wedge \neg g))$

shows $\vdash f \wedge finite \longrightarrow g$

proof –

have 1: $\vdash f \wedge \neg g \longrightarrow (fmore ; (f \wedge \neg g))$ **using** *assms* **by** *auto*

hence 2: $\vdash finite \longrightarrow \neg (f \wedge \neg g)$ **by** (*rule MoreChopLoop*)

from 2 **show** *?thesis* **by** *auto*

qed

lemma *MoreChopLoopFinite*:

assumes $\vdash f \wedge finite \longrightarrow fmore ; f$

shows $\vdash finite \longrightarrow \neg f$

proof –

have 1: $\vdash f \wedge finite \longrightarrow fmore ; f$

using *assms* **by** *auto*

hence 11: $\vdash \Diamond (f \wedge finite) \longrightarrow \Diamond (fmore;f)$

using *DiamondImpDiamond* **by** *blast*

have 12: $\vdash \Diamond (fmore;f) = finite;(fmore;f)$

by (*simp add: sometimes-d-def*)

have 13: $\vdash finite;(fmore;f) = (finite;fmore);f$

by (*rule ChopAssoc*)

have 14: $\vdash \Diamond (fmore;f) = fmore;f$

using *FiniteChopFmoreEqvFmore* 12 13 **by** (*metis int-eq*)

have 2: $\vdash fmore ; f = \bigcirc(\Diamond f)$

using *MoreChopEqvNextDiamond* **by** *blast*

have 3: $\vdash \Diamond (f \wedge finite) \longrightarrow \bigcirc(\Diamond f)$

using 11 14 2 **by** *fastforce*

have 31: $\vdash \Diamond (f \wedge finite) = ((\Diamond f) \wedge finite)$

by (*metis (no-types, lifting) 3 ChopAndB ChopAndNotChopImp DiamondDiamondEqvDiamond DiamondIntroC FiniteChopFiniteEqvFinite FiniteChopInfEqvInf Prop11 Prop12 finite-d-def inteq-reflection sometimes-d-def*)

have 32: $\vdash (\Diamond f) \wedge finite \longrightarrow \bigcirc(\Diamond f)$

using 3 31 **by** *fastforce*

hence 4: $\vdash finite \longrightarrow \neg (\Diamond f)$

by (*metis (no-types, lifting) DiamondIntro FiniteChopInfEqvInf InfEqvNotFinite Prop09 finite-d-def int-simps(15) int-simps(32) inteq-reflection sometimes-d-def*)

have 5: $\vdash \neg (\Diamond f) \longrightarrow \neg f$

by (*simp add: NowImpDiamond*)

from 4 5 **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*

qed

lemma *MoreChopEqvFmoreOrInf*:

$\vdash more ; f = (fmore;f) \vee inf)$

by (*metis AndInfChopEqvAndInf MoreAndInfEqvInf OrChopEqv OrFiniteInf fmore-d-def int-eq*)

lemma *MoreChopLoopFiniteB*:

assumes $\vdash f \longrightarrow more ; f$

shows $\vdash finite \longrightarrow \neg f$

proof –

have 1: $\vdash f \longrightarrow more ; f$

```

using assms by auto
have 10:  $\vdash f \longrightarrow (fmore;f) \vee inf$ 
using MoreChopEqvFmoreOrInf assms by fastforce
hence 100:  $\vdash f \wedge finite \longrightarrow (fmore;f)$ 
by (simp add: Prop13 finite-d-def)
hence 11:  $\vdash \Diamond (f \wedge finite) \longrightarrow \Diamond (fmore;f)$ 
using DiamondImpDiamond by blast
have 12:  $\vdash \Diamond (fmore;f) = finite;(fmore;f)$ 
by (simp add: sometimes-d-def)
have 13:  $\vdash finite;(fmore;f) = (finite;fmore);f$ 
by (rule ChopAssoc)
have 14:  $\vdash \Diamond (fmore;f) = fmore;f$ 
using FiniteChopFmoreEqvFmore 12 13 by (metis int-eq)
have 2:  $\vdash fmore ; f = \bigcirc(\Diamond f)$ 
using MoreChopEqvNextDiamond by blast
have 3:  $\vdash \Diamond (f \wedge finite) \longrightarrow \bigcirc(\Diamond f)$ 
using 11 14 2 by fastforce
have 31:  $\vdash \Diamond (f \wedge finite) = ((\Diamond f) \wedge finite)$ 
by (metis (no-types, opaque-lifting) ChopAndA ChopAndB ChopAndNotChopImp FiniteChopFiniteEqvFi-
nite
FiniteChopInfEqvInf Prop11 Prop12 finite-d-def inteq-reflection sometimes-d-def)
have 32:  $\vdash (\Diamond f) \wedge finite \longrightarrow \bigcirc(\Diamond f)$ 
using 3 31 by fastforce
hence 4:  $\vdash finite \longrightarrow \neg (\Diamond f)$ 
by (metis (no-types, lifting) DiamondIntro FiniteChopInfEqvInf InfEqvNotFinite Prop09
finite-d-def int-simps(15) int-simps(32) inteq-reflection sometimes-d-def)
have 5:  $\vdash \neg (\Diamond f) \longrightarrow \neg f$ 
by (simp add: NowImpDiamond)
from 4 5 show ?thesis using lift-imp-trans by fastforce
qed

```

lemma *MoreChopContraFinite*:

```

assumes  $\vdash (f \wedge \neg g) \wedge finite \longrightarrow (fmore ; (f \wedge \neg g))$ 
shows  $\vdash f \wedge finite \longrightarrow g$ 
proof –
have 1:  $\vdash (f \wedge \neg g) \wedge finite \longrightarrow (fmore ; (f \wedge \neg g))$  using assms by auto
hence 2:  $\vdash finite \longrightarrow \neg (f \wedge \neg g)$  using MoreChopLoopFinite by (simp add: MoreChopLoopFinite)
from 2 show ?thesis by (simp add: Valid-def)
qed

```

lemma *MoreChopContraFiniteB*:

```

assumes  $\vdash (f \wedge \neg g) \longrightarrow (more ; (f \wedge \neg g))$ 
shows  $\vdash f \wedge finite \longrightarrow g$ 
proof –
have 1:  $\vdash (f \wedge \neg g) \longrightarrow (more ; (f \wedge \neg g))$  using assms by auto
hence 2:  $\vdash finite \longrightarrow \neg (f \wedge \neg g)$  using MoreChopLoopFinite by (simp add: MoreChopLoopFiniteB)
from 2 show ?thesis by (simp add: Valid-def)
qed

```

lemma *ChopLoop*:

assumes $\vdash f \longrightarrow g;f$
 $\vdash g \longrightarrow fmore$
shows $\vdash finite \longrightarrow \neg f$
proof –
have 1: $\vdash f \longrightarrow g; f$ **using** *assms* **by** *auto*
have 2: $\vdash g \longrightarrow fmore$ **using** *assms* **by** *auto*
hence 3: $\vdash g; f \longrightarrow fmore ; f$ **by** (*rule LeftChopImpChop*)
have 4: $\vdash f \longrightarrow fmore ; f$ **using** 1 3 **by** *fastforce*
from 4 **show** ?thesis **using** *MoreChopLoop* **by** *auto*
qed

lemma *ChopLoopB*:
assumes $\vdash f \longrightarrow g;f$
 $\vdash g \longrightarrow more$
shows $\vdash finite \longrightarrow \neg f$
proof –
have 1: $\vdash f \longrightarrow g; f$ **using** *assms* **by** *auto*
have 2: $\vdash g \longrightarrow more$ **using** *assms* **by** *auto*
hence 3: $\vdash g; f \longrightarrow more ; f$ **by** (*rule LeftChopImpChop*)
have 4: $\vdash f \longrightarrow more ; f$ **using** 1 3 **by** *fastforce*
from 4 **show** ?thesis **using** *MoreChopLoopFiniteB* **by** *auto*
qed

lemma *ChopContra*:
assumes $\vdash f \wedge \neg g \longrightarrow h; f \wedge \neg (h; g)$
 $\vdash h \longrightarrow fmore$
shows $\vdash f \wedge finite \longrightarrow g$
proof –
have 1: $\vdash f \wedge \neg g \longrightarrow h; f \wedge \neg (h; g)$ **using** *assms* **by** *auto*
have 2: $\vdash h \longrightarrow fmore$ **using** *assms* **by** *auto*
have 3: $\vdash h; f \wedge \neg (h; g) \longrightarrow h; (f \wedge \neg g)$ **by** (*rule ChopAndNotChopImp*)
have 4: $\vdash h; (f \wedge \neg g) \longrightarrow fmore ; (f \wedge \neg g)$ **using** 2 **by** (*rule LeftChopImpChop*)
have 5: $\vdash f \wedge \neg g \longrightarrow fmore ; (f \wedge \neg g)$ **using** 1 3 4 **by** *fastforce*
from 5 **show** ?thesis **using** *MoreChopContra* **by** *auto*
qed

lemma *ChopContraB*:
assumes $\vdash f \wedge \neg g \longrightarrow h; f \wedge \neg (h; g)$
 $\vdash h \longrightarrow more$
shows $\vdash f \wedge finite \longrightarrow g$
proof –
have 1: $\vdash f \wedge \neg g \longrightarrow h; f \wedge \neg (h; g)$ **using** *assms* **by** *auto*
have 2: $\vdash h \longrightarrow more$ **using** *assms* **by** *auto*
have 3: $\vdash h; f \wedge \neg (h; g) \longrightarrow h; (f \wedge \neg g)$ **by** (*rule ChopAndNotChopImp*)
have 4: $\vdash h; (f \wedge \neg g) \longrightarrow more ; (f \wedge \neg g)$ **using** 2 **by** (*rule LeftChopImpChop*)
have 5: $\vdash f \wedge \neg g \longrightarrow more ; (f \wedge \neg g)$ **using** 1 3 4 **by** *fastforce*
from 5 **show** ?thesis **using** *MoreChopContraFiniteB* **by** *auto*
qed

2.3.8 Properties of Halt

lemma *WnextAndMoreEqvNext*:

$\vdash (wnext\ f \wedge more) = \bigcirc f$

proof –

have 1: $\vdash wnext\ f = (empty \vee \bigcirc f)$

by (*simp add: WnextEqvEmptyOrNext*)

have 2: $\vdash \bigcirc f \longrightarrow more$

by (*metis DiIntro LeftChopImpMoreRule di-d-def more-d-def next-d-def*)

have 3: $\vdash ((empty \vee \bigcirc f) \wedge more) = \bigcirc f$

unfolding *empty-d-def* **using** 2 **by** *auto*

show ?thesis **by** (*metis 1 3 int-eq*)

qed

lemma *BoxStateAndEmptyEqvStateAndEmpty*:

$\vdash (\Box(empty = (init\ w)) \wedge empty) = ((init\ w) \wedge empty)$

proof –

have 1: $\vdash ((empty = (init\ w)) \wedge empty) = ((init\ w) \wedge empty)$

by *force*

have 2: $\vdash (\Box(empty = (init\ w)) \wedge empty) \longrightarrow ((init\ w) \wedge empty)$

using *BoxElim* **by** *fastforce*

have 3: $\vdash ((init\ w) \wedge empty) \longrightarrow (\Box(empty = (init\ w)) \wedge empty)$

using *BoxEqvAndEmptyOrNextBox* **by** *fastforce*

show ?thesis

by (*simp add: 2 3 int-iffI*)

qed

lemma *BoxEmptyEqvIStateqvEmptyAndStateOrNotStateNext*:

$\vdash \Box(empty = (init\ w)) = ((empty \wedge init\ w) \vee (\neg(init\ w) \wedge \bigcirc(\Box(empty = (init\ w)))))$

proof –

have 1: $\vdash \Box(empty = (init\ w)) =$

$((\Box(empty = (init\ w)) \wedge empty) \vee (\Box(empty = (init\ w)) \wedge more))$

by (*auto simp: empty-d-def*)

have 2: $\vdash (\Box(empty = (init\ w)) \wedge empty) = ((init\ w) \wedge empty)$

using *BoxStateAndEmptyEqvStateAndEmpty* **by** *blast*

have 3: $\vdash \Box(empty = (init\ w)) = ((empty = (init\ w)) \wedge wnext(\Box(empty = (init\ w))))$

using *BoxEqvAndWnextBox* **by** *blast*

hence 4: $\vdash (\Box(empty = (init\ w)) \wedge more) =$

$((\Box(empty = (init\ w)) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more))$

by *auto*

have 5: $\vdash ((\Box(empty = (init\ w)) \wedge more) = (\neg(init\ w) \wedge more))$

by (*auto simp: empty-d-def*)

have 6: $\vdash (wnext(\Box(empty = (init\ w))) \wedge more) = \bigcirc(\Box(empty = (init\ w)))$

using *WnextAndMoreEqvNext* **by** *metis*

have 7: $\vdash (\Box(empty = (init\ w)) \wedge more) =$

$((\neg(init\ w) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more))$

using 4 5 **by** *fastforce*

have 8: $\vdash ((\neg(init\ w) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)) =$

$((\neg(init\ w)) \wedge (wnext(\Box(empty = (init\ w))) \wedge more))$

by *auto*

have 9: $\vdash ((\neg(init\ w)) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)) =$

```

      ((¬ (init w)) ∧ ○(□(empty = (init w))))
    using 8 6 by auto
  have 10: ⊢ □(empty = (init w)) = (((init w) ∧ empty) ∨ (□(empty = (init w)) ∧ more) )
    using 1 2 by fastforce
  show ?thesis
    using 10 7 9 by fastforce
qed

```

lemma *HaltStateEqvIfStateThenEmptyElseNext*:

```

⊢ halt ( init w ) = ifi (init w) then empty else ( ○( halt ( init w ) ) )
by (metis BoxEmptyEqvIStateEqvEmptyAndStateOrNotStateNext halt-d-def ifthenelse-d-def inteq-reflection
    lift-and-com)

```

lemma *HaltChopEqv*:

```

⊢ ((halt ( init w ) ) ; f) = (ifi (init w) then ( f ) else (○( halt ( init w ) ; f )))
proof -
  have 1: ⊢ halt (init w) =
    (ifi (init w) then empty else ( ○( halt ( init w ) )))
    by (rule HaltStateEqvIfStateThenEmptyElseNext)
  hence 2: ⊢ ((halt (init w)) ; f) =
    (ifi (init w) then (empty ; f) else ( ○( halt ( init w ) ; f )))
    by (rule IfChopEqvRule)
  have 3: ⊢ empty ; f = f
    by (rule EmptyChop)
  have 4: ⊢ (○( halt ( init w ) )) ; f = ○( halt ( init w ) ; f)
    by (rule NextChop)
  from 2 3 4 show ?thesis by (metis inteq-reflection)
qed

```

lemma *AndHaltChopImp*:

```

⊢ init w ∧ ( halt ( init w ) ; f ) ⟶ f
proof -
  have 1: ⊢ halt ( init w ) ; f = ifi (init w) then f else ( ○( halt ( init w ) ; f ) )
    by (rule HaltChopEqv)
  have 2: ⊢ init w ∧ ifi (init w) then f else ( ○( halt ( init w ) ; f ) ) ⟶ f
    by (auto simp: ifthenelse-d-def)
  from 1 2 show ?thesis by fastforce
qed

```

lemma *NotAndHaltChopImpNext*:

```

⊢ ¬ ( init w ) ∧ ( halt ( init w ) ; f ) ⟶ ○( halt ( init w ) ; f )
proof -
  have 1: ⊢ halt ( init w ) ; f = ifi (init w) then f else ( ○( halt ( init w ) ; f ) )
    by (rule HaltChopEqv)
  have 2: ⊢ ¬ ( init w ) ∧ ifi (init w) then f else ( ○( halt ( init w ) ; f ) ) ⟶
    ○( halt ( init w ) ; f )
    by (auto simp: ifthenelse-d-def)
  from 1 2 show ?thesis by fastforce
qed

```

lemma *NotAndHaltChopImpSkipYields*:
 $\vdash \neg (\text{init } w) \wedge (\text{halt } (\text{init } w); f) \longrightarrow \text{skip yields } (\text{halt } (\text{init } w); f)$
proof –
have 1: $\vdash \neg (\text{init } w) \wedge (\text{halt } (\text{init } w); f) \longrightarrow \bigcirc (\text{halt } (\text{init } w); f)$
by (*rule NotAndHaltChopImpNext*)
have 2: $\vdash \bigcirc (\text{halt } (\text{init } w); f) \longrightarrow \text{skip yields } (\text{halt } (\text{init } w); f)$
by (*rule NextImpSkipYields*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *FiniteChopAndEmptyEqvChopAndEmpty*:
 $\vdash ((\text{finite};(f \wedge \text{empty})) \wedge g) = ((g \wedge \text{finite});(f \wedge \text{empty}))$
proof –
have 1: $\vdash g \wedge \text{finite};(f \wedge \text{empty}) \longrightarrow \text{fin } f$
by (*metis ChopAndA DiamondFin FinAndEmpty Prop01 Prop05 inteq-reflection sometimes-d-def*)
have 2: $\vdash g \wedge \text{finite};(f \wedge \text{empty}) \longrightarrow (\text{finite} \wedge g) \wedge \text{fin } f$
using 1 **by** (*metis (no-types, lifting) ChopAndB ChopEmpty Prop10 Prop12 int-iffD1 inteq-reflection*)
have 3: $\vdash ((\text{finite};(f \wedge \text{empty})) \wedge g) \longrightarrow ((g \wedge \text{finite});(f \wedge \text{empty}))$
using 2 **using** *AndFinEqvChopAndEmpty* **by** fastforce
have 4: $\vdash ((g \wedge \text{finite});(f \wedge \text{empty})) \longrightarrow ((\text{finite};(f \wedge \text{empty})) \wedge g)$
by (*metis AndChopB ChopAndB ChopEmpty Prop12 inteq-reflection*)
from 3 4 **show** ?thesis **by** fastforce
qed

lemma *WprevEqvEmptyOrPrev*:
 $\vdash \text{wprev } f = (\text{empty} \vee \text{prev } f)$
using *nlength-eq-enat-nfiniteD* **by** (*auto simp add: Valid-def itl-defs zero-enat-def*)

lemma *NotChopSkipEqvMoreAndNotChopSkip*:
 $\vdash (\neg f); \text{skip} = (\text{more} \wedge \neg(f; \text{skip}))$
proof –
have 1: $\vdash \text{wprev } f = (\text{empty} \vee \text{prev } f)$ **using** *WprevEqvEmptyOrPrev* **by** auto
hence 2: $\vdash (\neg(\text{wprev } f)) = (\neg(\text{empty} \vee \text{prev } f))$ **by** auto
have 3: $\vdash \neg(\text{wprev } f) = ((\neg f); \text{skip})$ **by** (*simp add: wprev-d-def prev-d-def*)
have 31: $\vdash (\text{empty} \vee \text{prev } f) = (\text{empty} \vee (f; \text{skip}))$ **by** (*simp add: prev-d-def*)
have 32: $\vdash (\text{empty} \vee (f; \text{skip})) = (\neg \text{more} \vee \neg \neg(f; \text{skip}))$ **by** (*simp add: empty-d-def*)
have 33: $\vdash (\neg \text{more} \vee \neg \neg(f; \text{skip})) = (\neg(\text{more} \wedge \neg \neg(f; \text{skip})))$ **by** fastforce
have 34: $\vdash (\text{empty} \vee \text{prev } f) = (\neg(\text{more} \wedge \neg \neg(f; \text{skip})))$ **using** 31 32 33 **by** (*metis int-eq*)
have 4: $\vdash \neg(\text{empty} \vee \text{prev } f) = (\text{more} \wedge \neg \neg(f; \text{skip}))$ **using** 34 **by** fastforce
from 2 3 4 **show** ?thesis **by** fastforce
qed

lemma *HaltChopImpNotHaltChopNot*:
 $\vdash \text{halt } (\text{init } w); f \wedge \text{finite} \longrightarrow \neg (\text{halt } (\text{init } w); (\neg f))$
proof –
have 1: $\vdash \text{halt } (\text{init } w); f = \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc (\text{halt } (\text{init } w); f))$
by (*rule HaltChopEqv*)
have 2: $\vdash \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc (\text{halt } (\text{init } w); f)) \longrightarrow$
 $((\text{init } w) \longrightarrow f) \wedge (\neg (\text{init } w) \longrightarrow (\bigcirc (\text{halt } (\text{init } w); f)))$

by (rule *IfThenElseImp*)
 have 3: $\vdash \text{halt } (\text{init } w); (\neg f) =$
 $\text{if}_i (\text{init } w) \text{ then } (\neg f) \text{ else } (\bigcirc(\text{halt } (\text{init } w); (\neg f)))$
 by (rule *HaltChopEqv*)
 have 4: $\vdash \text{if}_i (\text{init } w) \text{ then } (\neg f) \text{ else } (\bigcirc(\text{halt } (\text{init } w); (\neg f))) \longrightarrow$
 $(((\text{init } w) \longrightarrow \neg f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt } (\text{init } w); (\neg f))))))$
 by (rule *IfThenElseImp*)
 have 5: $\vdash \text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); (\neg f) \longrightarrow$
 $((\text{init } w) \longrightarrow f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt } (\text{init } w); f))) \wedge$
 $((\text{init } w) \longrightarrow \neg f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt } (\text{init } w); (\neg f))))$
 using 1 2 3 4 by fastforce
 have 6: $\vdash ((\text{init } w) \longrightarrow f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt } (\text{init } w); f))) \wedge$
 $((\text{init } w) \longrightarrow \neg f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt } (\text{init } w); (\neg f)))) \longrightarrow$
 $(\bigcirc(\text{halt } (\text{init } w); f)) \wedge (\bigcirc(\text{halt } (\text{init } w); (\neg f)))$
 by auto
 have 7: $\vdash \text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); (\neg f) \longrightarrow$
 $(\bigcirc(\text{halt } (\text{init } w); f)) \wedge (\bigcirc(\text{halt } (\text{init } w); (\neg f)))$
 using 5 6 lift-imp-trans by blast
 have 8: $\vdash ((\bigcirc(\text{halt } (\text{init } w); f)) \wedge (\bigcirc(\text{halt } (\text{init } w); (\neg f)))) =$
 $\bigcirc(\text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); (\neg f))$
 using NextAndEqvNextAndNext by fastforce
 have 9: $\vdash \text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); (\neg f) \longrightarrow$
 $\bigcirc(\text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); (\neg f))$
 using 7 8 by fastforce
 hence 10: $\vdash \text{finite} \longrightarrow \neg(\text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); (\neg f))$
 using NextLoop by blast
 from 10 show ?thesis by auto
 qed

lemma *HaltChopImpHaltYields*:

$\vdash \text{halt } (\text{init } w); f \wedge \text{finite} \longrightarrow (\text{halt } (\text{init } w)) \text{ yields } f$

proof –

have 1: $\vdash \text{halt } (\text{init } w); f \wedge \text{finite} \longrightarrow \neg (\text{halt } (\text{init } w); (\neg f))$

by (rule *HaltChopImpNotHaltChopNot*)

from 1 show ?thesis by (simp add: yields-d-def)

qed

lemma *HaltChopAnd*:

$\vdash (\text{halt } (\text{init } w)); f \wedge (\text{halt } (\text{init } w)); g \wedge \text{finite} \longrightarrow (\text{halt } (\text{init } w)); (f \wedge g)$

proof –

have 1: $\vdash (\text{halt } (\text{init } w)); g \wedge \text{finite} \longrightarrow (\text{halt } (\text{init } w)) \text{ yields } g$

by (rule *HaltChopImpHaltYields*)

hence 2: $\vdash (\text{halt } (\text{init } w)); f \wedge (\text{halt } (\text{init } w)); g \wedge \text{finite} \longrightarrow$

$(\text{halt } (\text{init } w)); f \wedge (\text{halt } (\text{init } w)) \text{ yields } g$

by auto

have 3: $\vdash (\text{halt } (\text{init } w)); f \wedge (\text{halt } (\text{init } w)) \text{ yields } g \longrightarrow$

$(\text{halt } (\text{init } w)); (f \wedge g)$

by (rule *ChopAndYieldsImp*)

from 2 3 show ?thesis by fastforce

qed

lemma *HaltAndChopAndHaltChopImpHaltAndChopAnd:*

$\vdash (\text{halt } (\text{init } w) \wedge f); f1 \wedge (\text{halt } (\text{init } w); g) \wedge \text{finite} \longrightarrow (\text{halt } (\text{init } w) \wedge f); (f1 \wedge g)$

proof –

have 1: $\vdash f1 \longrightarrow \neg g \vee (f1 \wedge g)$

by *auto*

hence 2: $\vdash (\text{halt } (\text{init } w) \wedge f); f1 \longrightarrow$
 $(\text{halt } (\text{init } w) \wedge f); (\neg g) \vee ((\text{halt } (\text{init } w) \wedge f); (f1 \wedge g))$

by (*rule ChopOrImpRule*)

have 3: $\vdash (\text{halt } (\text{init } w) \wedge f); (\neg g) \longrightarrow \text{halt } (\text{init } w); (\neg g)$

by (*rule AndChopA*)

have 31: $\vdash (\text{halt } (\text{init } w) \wedge f); f1 \longrightarrow$
 $\text{halt } (\text{init } w); (\neg g) \vee ((\text{halt } (\text{init } w) \wedge f); (f1 \wedge g))$

using 23 **by** *fastforce*

have 4: $\vdash \text{halt } (\text{init } w); g \wedge \text{finite} \longrightarrow \neg (\text{halt } (\text{init } w); (\neg g))$

by (*rule HaltChopImpNotHaltChopNot*)

hence 41: $\vdash (\text{halt } (\text{init } w); (\neg g)) \wedge \text{finite} \longrightarrow \neg(\text{halt } (\text{init } w); g)$

by *auto*

have 42: $\vdash (\text{halt } (\text{init } w) \wedge f); f1 \wedge \text{finite} \longrightarrow$
 $\neg(\text{halt } (\text{init } w); g) \vee ((\text{halt } (\text{init } w) \wedge f); (f1 \wedge g))$

using 31 41 **by** *fastforce*

from 42 **show** *?thesis* **by** *auto*

qed

lemma *HaltImpBoxYields:*

$\vdash (\text{halt } (\text{init } w)); f \wedge \text{finite} \longrightarrow (\Box(\neg (\text{init } w))) \text{ yields } ((\text{halt } (\text{init } w)); f)$

proof –

have 1: $\vdash (\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)) \longrightarrow \text{di } (\Box(\neg (\text{init } w)))$

by (*rule ChopImpDi*)

have 2: $\vdash \Box(\neg (\text{init } w)) \longrightarrow \neg (\text{init } w)$

by (*rule BoxElim*)

hence 3: $\vdash \text{di } (\Box(\neg (\text{init } w))) \longrightarrow \text{di } (\neg (\text{init } w))$

by (*rule DiImpDi*)

have 4: $\vdash \text{di } (\text{init } (\neg w)) = (\text{init } (\neg w))$

by (*rule DiState*)

have 41: $\vdash (\text{init } (\neg w)) = (\neg (\text{init } w))$

using *Initprop(2)* **by** *fastforce*

have 42: $\vdash \text{di } (\neg (\text{init } w)) = (\neg(\text{init } w))$

using 4 41 **by** (*metis inteq-reflection*)

have 5: $\vdash ((\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f))) \longrightarrow \neg (\text{init } w)$

using 1 2 42 **using** 3 **by** *fastforce*

hence 51: $\vdash (\text{halt } (\text{init } w); f) \wedge ((\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f))) \longrightarrow$
 $(\text{halt } (\text{init } w); f) \wedge \neg (\text{init } w)$

by *fastforce*

have 6: $\vdash \text{halt } (\text{init } w); f = \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\Box(\text{halt } (\text{init } w); f))$

by (*rule HaltChopEqv*)

hence 61: $\vdash (\text{halt } (\text{init } w); f \wedge \neg (\text{init } w)) =$
 $((\text{if}_i (\text{init } w) \text{ then } f \text{ else } (\Box(\text{halt } (\text{init } w); f))) \wedge \neg (\text{init } w))$

using 6 **by** *auto*

have 62: $\vdash (\text{if}_i (\text{init } w) \text{ then } f \text{ else } (\Box(\text{halt } (\text{init } w); f))) \wedge$

$\neg (init\ w) \longrightarrow (\bigcirc(halt\ (init\ w); f))$
by (*auto simp: ifthenelse-d-def*)
have 63: $\vdash halt\ (init\ w); f \wedge \neg (init\ w) \longrightarrow (\bigcirc(halt\ (init\ w); f))$
using 61 62 **by** *fastforce*
have 7: $\vdash (halt\ (init\ w); f) \wedge (\Box(\neg (init\ w))); (\neg (halt\ (init\ w); f)) \longrightarrow$
 $\bigcirc((halt\ (init\ w)); f)$
using 51 63 **using** *lift-imp-trans* **by** *blast*
have 8: $\vdash \Box(\neg (init\ w)) \longrightarrow empty \vee \bigcirc(\Box(\neg (init\ w)))$
by (*metis BoxImpYields WeakNextBoxImpMoreYields WnextEqvEmptyOrNext fmore-d-def int-eq*)
hence 9: $\vdash ((\Box(\neg (init\ w))); (\neg (halt\ (init\ w); f))) \longrightarrow$
 $\neg (halt\ (init\ w); f) \vee \bigcirc((\Box(\neg (init\ w))); (\neg (halt\ (init\ w); f)))$
by (*rule EmptyOrNextChopImpRule*)
hence 10: $\vdash ((halt\ (init\ w)); f) \wedge (\Box(\neg (init\ w))); (\neg (halt\ (init\ w); f)) \longrightarrow$
 $\bigcirc((\Box(\neg (init\ w))); (\neg (halt\ (init\ w); f)))$
by *fastforce*
have 11: $\vdash (halt\ (init\ w)); f \wedge (\Box(\neg (init\ w))); (\neg (halt\ (init\ w); f)) \longrightarrow$
 $\bigcirc((halt\ (init\ w)); f) \wedge \bigcirc((\Box(\neg (init\ w))); (\neg (halt\ (init\ w); f)))$
using 7 10 **by** *fastforce*
have 12: $\vdash \bigcirc((halt\ (init\ w)); f) \wedge \bigcirc((\Box(\neg (init\ w))); (\neg (halt\ (init\ w); f)))$
 $\longrightarrow \bigcirc(((halt\ (init\ w)); f) \wedge ((\Box(\neg (init\ w))); (\neg (halt\ (init\ w); f))))$
using *NextAndEqvNextAndNext* **by** *fastforce*
have 13: $\vdash (halt\ (init\ w)); f \wedge (\Box(\neg (init\ w))); (\neg (halt\ (init\ w); f)) \longrightarrow$
 $\bigcirc(((halt\ (init\ w)); f) \wedge ((\Box(\neg (init\ w))); (\neg (halt\ (init\ w); f))))$
using 11 12 **by** *fastforce*
hence 14: $\vdash finite \longrightarrow \neg ((halt\ (init\ w)); f \wedge (\Box(\neg (init\ w))); (\neg (halt\ (init\ w); f)))$
using *NextLoop* **by** *blast*
hence 15: $\vdash (halt\ (init\ w)); f \wedge finite \longrightarrow \neg ((\Box(\neg (init\ w))); (\neg (halt\ (init\ w); f)))$
by *auto*
from 15 **show** *?thesis* **by** (*simp add: yields-d-def*)
qed

2.3.9 Properties of Groups of chops

lemma *NestedChopImpChop*:

assumes $\vdash init\ w \wedge f \longrightarrow g; (init\ w1 \wedge f1)$
 $\vdash init\ w1 \wedge f1 \longrightarrow g1; (init\ w2 \wedge f2)$
shows $\vdash init\ w \wedge f \longrightarrow g; (g1; (init\ w2 \wedge f2))$
proof –
have 1: $\vdash init\ w \wedge f \longrightarrow g; (init\ w1 \wedge f1)$ **using** *assms(1)* **by** *auto*
have 2: $\vdash init\ w1 \wedge f1 \longrightarrow g1; (init\ w2 \wedge f2)$ **using** *assms(2)* **by** *auto*
hence 3: $\vdash g; (init\ w1 \wedge f1) \longrightarrow g; (g1; (init\ w2 \wedge f2))$ **by** (*rule RightChopImpChop*)
from 1 3 **show** *?thesis* **by** *fastforce*
qed

end

2.4 Strong chop operator

theory *SChopTheorems*

```

imports
  Theorems
begin

```

We give the proofs of a list of Finite and Infinite ITL theorems but now using the strong chop.

2.4.1 Strong Chop axioms

```

lemma SChopAssoc:
   $\vdash f \frown (g \frown h) = (f \frown g) \frown h$ 
proof –
  have 1:  $\vdash f \frown (g \frown h) = (f \wedge \text{finite}); ((g \wedge \text{finite}); h)$ 
    by (simp add: schop-d-def)
  have 2:  $\vdash (f \wedge \text{finite}); ((g \wedge \text{finite}); h) = ((f \wedge \text{finite}); (g \wedge \text{finite})); h$ 
    using ChopAssoc by blast
  have 3:  $\vdash ((f \wedge \text{finite}); (g \wedge \text{finite})); h = (f \frown (g \wedge \text{finite})); h$ 
    by (simp add: schop-d-def)
  have 4:  $\vdash f \frown (g \wedge \text{finite}) = (f \frown g \wedge \text{finite})$ 
    by (simp add: schop-d-def)
    (metis AndChopA ChopAndA ChopAndFiniteDist Prop11 Prop12 inteq-reflection)
  have 5:  $\vdash (f \frown (g \wedge \text{finite})); h = (f \frown g \wedge \text{finite}); h$ 
    using 4 by (simp add: LeftChopEqvChop)
  have 6:  $\vdash (f \frown g \wedge \text{finite}); h = (f \frown g) \frown h$ 
    by (simp add: schop-d-def)
from 1 2 3 5 6 show ?thesis by fastforce
qed

```

```

lemma FiniteOr:
   $\vdash ((f \vee g) \wedge \text{finite}) = ((f \wedge \text{finite}) \vee (g \wedge \text{finite}))$ 
by auto

```

```

lemma OrSChopImp :
   $\vdash (f \vee g) \frown h \longrightarrow f \frown h \vee g \frown h$ 
unfolding s chop-d-def
by (simp add: FiniteOr OrChopImpRule int-iffD1)

```

```

lemma SChopOrImp :
   $\vdash f \frown (g \vee h) \longrightarrow f \frown g \vee f \frown h$ 
unfolding s chop-d-def by (simp add: ChopOrImp)

```

```

lemma EmptySChop :
   $\vdash \text{empty} \frown f = f$ 
by (metis EmptyChopSem FiniteAndEmptyEqvEmpty intI inteq-reflection lift-and-com schop-d-def)

```

```

lemma SChopEmpty :
   $\vdash \text{finite} \longrightarrow f \frown \text{empty} = f$ 
unfolding s chop-d-def
proof –
have f1:  $\vdash (f \wedge \text{finite}); \text{empty} = (f \wedge \text{finite})$ 
  by (simp add: ChopEmpty int-eq)

```


then show $\vdash \text{finite} \longrightarrow (f \wedge \text{finite}); \text{empty} = f$
by *fastforce*
qed

lemma *StateImpBf* :
 $\vdash \text{init } f \longrightarrow \text{bf } (\text{init } f)$
unfolding *bf-d-def df-d-def schop-d-def*
by (*metis (no-types) AndChopA StateImpBi bi-d-def di-d-def lift-imp-neg lift-imp-trans*)

lemma *BfBoxSChopImpSChop* :
 $\vdash \text{bf } (f \longrightarrow f1) \wedge \Box(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f1 \frown g1$
by (*auto simp add: Valid-def itl-defs*)

lemma *AndMoreSChopEqvAndFmoreChop*:
 $\vdash (f \wedge \text{more}) \frown g = (f \wedge \text{fmore}); g$
by (*simp add: LeftChopEqvChop AndMoreAndFiniteEqvAndFmore schop-d-def*)

lemma *FiniteBfGen*:
assumes $\vdash \text{finite} \longrightarrow f$
shows $\vdash \text{bf } f$
using *assms*
by (*simp add: Valid-def itl-defs*)

lemma *BfGen*:
assumes $\vdash f$
shows $\vdash \text{bf } f$
using *assms*
by (*metis EmptyImpFinite FiniteAndEmptyEqvEmpty FiniteBfGen Prop09 int-eq-true inteq-reflection*)

2.4.2 ITL operators in terms of SChop

lemma *NextSChopdef*:
 $\vdash \bigcirc f = \text{skip} \frown f$
by (*metis FiniteChopSkipEqvSkipChopFinite NowImpDiamond Prop10 SkipChopFiniteImpFinite inteq-reflection lift-imp-trans next-d-def schop-d-def sometimes-d-def*)

lemma *DiamondSChopdef*:
 $\vdash \Diamond f = \# \text{True} \frown f$
by (*simp add: schop-d-def sometimes-d-def*)

lemma *FiniteSChopdef*:
 $\vdash \text{finite} = \Diamond \text{empty}$
by (*simp add: DiamondEmptyEqvFinite int-iffD1 int-iffD2 int-iffI*)

lemma *ChopSChopdef*:
 $\vdash f;g = ((f \frown g) \vee (f \wedge \text{inf}))$
by (*metis AndInfChopEqvAndInf OrChopEqv OrFiniteInf inteq-reflection schop-d-def*)

lemma *SFinprop* :

$\vdash ((\# \text{True} \neg (f \wedge \text{empty})) \wedge (\# \text{True} \neg (g \wedge \text{empty}))) = (\# \text{True} \neg ((f \wedge g) \wedge \text{empty}))$
 $\vdash ((\# \text{True} \neg (f \wedge \text{empty})) \vee (\# \text{True} \neg (g \wedge \text{empty}))) = (\# \text{True} \neg ((f \vee g) \wedge \text{empty}))$
 $\vdash \text{finite} \longrightarrow (\neg (\# \text{True} \neg (f \wedge \text{empty}))) = (\# \text{True} \neg (\neg f \wedge \text{empty}))$
 $\vdash (\neg (\# \text{True} \neg (f \wedge \text{empty}))) = ((\# \text{True} \neg (\neg f \wedge \text{empty})) \vee \text{inf})$

by (*auto simp add: Valid-def itl-defs zero-enat-def*)

(*metis add.right-neutral enat.distinct(2) enat-add-sub-same less-eqE the-enat.simps zero-enat-def,*
metis ndropn-eq-NNil ndropn-nlast ndropn-nlength nfinite-nlength-enat nlength-NNil the-enat.simps
zero-enat-def,
metis add.right-neutral enat.distinct(2) enat-add-sub-same le-iff-add the-enat.simps zero-enat-def,
metis ndropn-eq-NNil ndropn-nlast ndropn-nlength nfinite-nlength-enat nlength-NNil the-enat.simps
zero-enat-def,
metis add.right-neutral enat.distinct(2) enat-add-sub-same le-iff-add the-enat.simps zero-enat-def,
metis ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD)

2.4.3 Basic Theorems

lemma *BfSChopImpSChop* :

$\vdash \text{bf } (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$

proof –

have 1: $\vdash g \longrightarrow g$ **by** *auto*

hence 2: $\vdash \square (g \longrightarrow g)$ **by** (*rule BoxGen*)

have 3: $\vdash \text{bf } (f \longrightarrow f1) \wedge \square(g \longrightarrow g) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (*rule BfBoxSChopImpSChop*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *BiImpBf*:

$\vdash \text{bi } f \longrightarrow \text{bf } f$

unfolding *bi-d-def bf-d-def di-d-def df-d-def schop-d-def*

by (*simp add: AndChopA*)

lemma *BiSChopImpSChop* :

$\vdash \text{bi } (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$

proof –

have 1: $\vdash g \longrightarrow g$

by *auto*

hence 2: $\vdash \square (g \longrightarrow g)$

by (*rule BoxGen*)

have 3: $\vdash \text{bi } (f \longrightarrow f1) \wedge \square(g \longrightarrow g) \longrightarrow f \frown g \longrightarrow f1 \frown g$

using *BiImpBf BfBoxSChopImpSChop* **using** *BfSChopImpSChop* **by** *fastforce*

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *AndSChopA*:

$\vdash (f \wedge f1) \frown g \longrightarrow f \frown g$

proof –

have 1: $\vdash f \wedge f1 \longrightarrow f$ **by** *auto*

hence 2: $\vdash \text{bf } (f \wedge f1 \longrightarrow f)$ **by** (*rule BfGen*)

have 3: $\vdash \text{bf } (f \wedge f1 \longrightarrow f) \longrightarrow (f \wedge f1) \frown g \longrightarrow f \frown g$ **by** (rule BfSChopImpSChop)
from 2 3 **show** ?thesis **using** MP **by** blast
qed

lemma AndSChopB:

$\vdash (f \wedge f1) \frown g \longrightarrow f1 \frown g$

proof –

have 1: $\vdash f \wedge f1 \longrightarrow f1$ **by** auto

hence 2: $\vdash \text{bf } (f \wedge f1 \longrightarrow f1)$ **by** (rule BfGen)

have 3: $\vdash \text{bf } (f \wedge f1 \longrightarrow f1) \longrightarrow (f \wedge f1) \frown g \longrightarrow f1 \frown g$ **by** (rule BfSChopImpSChop)

from 2 3 **show** ?thesis **using** MP **by** blast

qed

lemma NextSChop:

$\vdash (\bigcirc f) \frown g = \bigcirc(f \frown g)$

proof –

have 1: $\vdash \text{skip} \frown (f \frown g) = (\text{skip} \frown f) \frown g$ **by** (rule SChopAssoc)

from 1 **show** ?thesis **using** NextSChopdef **by** (metis integ-reflection)

qed

lemma BoxSChopImpSChop :

$\vdash \Box (g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$

proof –

have 1: $\vdash g \longrightarrow g$ **by** auto

hence 2: $\vdash \text{bf } (g \longrightarrow g)$ **by** (rule BfGen)

have 3: $\vdash \text{bf } (f \longrightarrow f) \wedge \Box(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **by** (rule BfBoxSChopImpSChop)

from 2 3 **show** ?thesis **by** fastforce

qed

lemma LeftSChopImpSChop:

assumes $\vdash f \longrightarrow f1$

shows $\vdash f \frown g \longrightarrow f1 \frown g$

proof –

have 1: $\vdash f \longrightarrow f1$ **using** assms **by** auto

hence 2: $\vdash \text{bf } (f \longrightarrow f1)$ **by** (rule BfGen)

have 3: $\vdash \text{bf } (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (rule BfSChopImpSChop)

from 2 3 **show** ?thesis **using** MP **by** blast

qed

lemma RightSChopImpSChop:

assumes $\vdash g \longrightarrow g1$

shows $\vdash f \frown g \longrightarrow f \frown g1$

proof –

have 1: $\vdash g \longrightarrow g1$ **using** assms **by** auto

hence 2: $\vdash \Box (g \longrightarrow g1)$ **by** (rule BoxGen)

have 3: $\vdash \Box (g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **by** (rule *BoxSChopImpSChop*)
from 2 3 **show** ?thesis **using** MP **by** blast
qed

lemma *RightSChopEqvSChop*:

assumes $\vdash g = g1$
shows $\vdash (f \frown g) = (f \frown g1)$
proof –
have 1: $\vdash g = g1$ **using** assms **by** auto
have 2: $(\vdash g \longrightarrow g1) \implies (\vdash f \frown g \longrightarrow f \frown g1)$ **by** (rule *RightSChopImpSChop*)
have 3: $(\vdash g1 \longrightarrow g) \implies (\vdash f \frown g1 \longrightarrow f \frown g)$ **by** (rule *RightSChopImpSChop*)
from 1 2 3 **show** ?thesis **by** fastforce
qed

lemma *BoxRightSChopEqvSChop*:

$\vdash \Box (g = g1) \longrightarrow (f \frown g) = (f \frown g1)$
proof –
have 0: $\vdash (g = g1) = ((g \longrightarrow g1) \wedge (g1 \longrightarrow g))$
by fastforce
have 1: $\vdash \Box (g = g1) = (\Box (g \longrightarrow g1) \wedge \Box (g1 \longrightarrow g))$
by (metis 0 BoxAndBoxEqvBoxRule inteq-reflection)
have 2: $\vdash \Box (g \longrightarrow g1) \longrightarrow (f \frown g) \longrightarrow (f \frown g1)$
by (simp add: BoxSChopImpSChop)
have 3: $\vdash \Box (g1 \longrightarrow g) \longrightarrow (f \frown g1) \longrightarrow (f \frown g)$
by (simp add: BoxSChopImpSChop)
from 1 2 3 **show** ?thesis **by** fastforce
qed

lemma *FiniteRightSChopEqvSChop*:

assumes $\vdash \text{finite} \longrightarrow g = g1$
shows $\vdash \text{finite} \longrightarrow (f \frown g) = (f \frown g1)$
using assms **unfolding** schop-d-def
by (simp add: FiniteRightChopEqvChop)

lemma *SChopOrEqv*:

$\vdash f \frown (g \vee g1) = (f \frown g \vee f \frown g1)$
proof –
have 1: $\vdash g \longrightarrow g \vee g1$ **by** auto
hence 2: $\vdash f \frown g \longrightarrow f \frown (g \vee g1)$ **by** (rule *RightSChopImpSChop*)
have 3: $\vdash g1 \longrightarrow g \vee g1$ **by** auto
hence 4: $\vdash f \frown g1 \longrightarrow f \frown (g \vee g1)$ **by** (rule *RightSChopImpSChop*)
from 2 4 **show** ?thesis **by** (meson SChopOrImp Prop02 Prop11)
qed

lemma *OrSChopEqv*:

$\vdash (f \vee f1) \frown g = (f \frown g \vee f1 \frown g)$
proof –

have 1: $\vdash f \longrightarrow f \vee f1$ **by** *auto*
hence 2: $\vdash f \wedge g \longrightarrow (f \vee f1) \wedge g$ **by** (*rule LeftSChopImpSChop*)
have 3: $\vdash f1 \longrightarrow f \vee f1$ **by** *auto*
hence 4: $\vdash f1 \wedge g \longrightarrow (f \vee f1) \wedge g$ **by** (*rule LeftSChopImpSChop*)
from 2 4 **show** *?thesis*
by (*meson OrSChopImp int-iffI Prop02*)
qed

lemma *OrSChopImpRule*:
assumes $\vdash f \longrightarrow f1 \vee f2$
shows $\vdash f \wedge g \longrightarrow (f1 \wedge g) \vee (f2 \wedge g)$
proof –
have 1: $\vdash f \longrightarrow f1 \vee f2$ **using** *assms* **by** *auto*
hence 2: $\vdash f \wedge g \longrightarrow (f1 \vee f2) \wedge g$ **by** (*rule LeftSChopImpSChop*)
have 3: $\vdash (f1 \vee f2) \wedge g = (f1 \wedge g \vee f2 \wedge g)$ **by** (*rule OrSChopEqv*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *LeftSChopEqvSChop*:
assumes $\vdash f = f1$
shows $\vdash f \wedge g = (f1 \wedge g)$
proof –
have 1: $\vdash f = f1$ **using** *assms* **by** *auto*
hence 2: $\vdash f \longrightarrow f1$ **by** *auto*
hence 3: $\vdash f \wedge g \longrightarrow f1 \wedge g$ **by** (*rule LeftSChopImpSChop*)
have $\vdash f1 \longrightarrow f$ **using** 1 **by** *auto*
hence 4: $\vdash f1 \wedge g \longrightarrow f \wedge g$ **by** (*rule LeftSChopImpSChop*)
from 3 4 **show** *?thesis* **by** (*simp add: int-iffI*)
qed

lemma *OrSChopEqvRule*:
assumes $\vdash f = (f1 \vee f2)$
shows $\vdash f \wedge g = ((f1 \wedge g) \vee (f2 \wedge g))$
proof –
have 1: $\vdash f = (f1 \vee f2)$ **using** *assms* **by** *auto*
hence 2: $\vdash f \wedge g = ((f1 \vee f2) \wedge g)$ **by** (*rule LeftSChopEqvSChop*)
have 3: $\vdash (f1 \vee f2) \wedge g = (f1 \wedge g \vee f2 \wedge g)$ **by** (*rule OrSChopEqv*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *SChopOrImpRule*:
assumes $\vdash g \longrightarrow g1 \vee g2$
shows $\vdash f \wedge g \longrightarrow (f \wedge g1) \vee (f \wedge g2)$
proof –
have 1: $\vdash g \longrightarrow g1 \vee g2$ **using** *assms* **by** *auto*
hence 2: $\vdash f \wedge g \longrightarrow f \wedge (g1 \vee g2)$ **by** (*rule RightSChopImpSChop*)
have 3: $\vdash f \wedge (g1 \vee g2) = (f \wedge g1 \vee f \wedge g2)$ **by** (*rule SChopOrEqv*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *SChopImpDiamond*:

$\vdash f \frown g \longrightarrow \Diamond g$

proof –

have 1: $\vdash f \longrightarrow \#True$ **by** *auto*

hence 2: $\vdash f \frown g \longrightarrow \#True \frown g$ **by** (*rule LeftSChopImpSChop*)

from 2 **show** *?thesis* **using** *DiamondSChopdef* **by** *fastforce*

qed

lemma *BfImpDfImpDf*:

$\vdash bf (f \longrightarrow g) \longrightarrow df f \longrightarrow df g$

proof –

have 1: $\vdash bf (f \longrightarrow g) \longrightarrow (f \frown \#True) \longrightarrow (g \frown \#True)$ **by** (*rule BfSChopImpSChop*)

from 1 **show** *?thesis* **by** (*simp add: df-d-def*)

qed

lemma *DfImpDf*:

assumes $\vdash f \longrightarrow g$

shows $\vdash df f \longrightarrow df g$

proof –

have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*

hence 2: $\vdash f \frown \#True \longrightarrow g \frown \#True$ **by** (*rule LeftSChopImpSChop*)

from 2 **show** *?thesis* **by** (*simp add: df-d-def*)

qed

lemma *BfImpBfRule*:

assumes $\vdash f \longrightarrow g$

shows $\vdash bf f \longrightarrow bf g$

proof –

have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*

hence 2: $\vdash \neg g \longrightarrow \neg f$ **by** *auto*

hence 3: $\vdash df (\neg g) \longrightarrow df (\neg f)$ **by** (*rule DfImpDf*)

hence 4: $\vdash \neg (df (\neg f)) \longrightarrow \neg (df (\neg g))$ **by** *auto*

from 4 **show** *?thesis* **by** (*simp add: bf-d-def*)

qed

lemma *DfEqvDf*:

assumes $\vdash f = g$

shows $\vdash df f = df g$

proof –

have 1: $\vdash f = g$ **using** *assms* **by** *auto*

hence 2: $\vdash f \frown \#True = g \frown \#True$ **by** (*rule LeftSChopEqvSChop*)

from 2 **show** *?thesis* **by** (*simp add: df-d-def*)

qed

lemma *BfEqvBf*:
assumes $\vdash f = g$
shows $\vdash bff = bf\ g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash (\neg f) = (\neg g)$ **by** *auto*
hence 3: $\vdash df\ (\neg f) = df\ (\neg g)$ **by** (*rule DfEqvDf*)
hence 4: $\vdash (\neg (df\ (\neg f))) = (\neg (df\ (\neg g)))$ **by** *auto*
from 4 **show** *?thesis* **by** (*simp add: bf-d-def*)
qed

lemma *LeftSChopSChopImpSChopRule*:
assumes $\vdash (f \frown g) \longrightarrow g$
shows $\vdash (f \frown g) \frown h \longrightarrow (g \frown h)$
proof –
have 1: $\vdash (f \frown g) \longrightarrow g$ **using** *assms* **by** *blast*
hence 2: $\vdash (f \frown g) \frown h \longrightarrow g \frown h$ **by** (*rule LeftSChopImpSChop*)
have 3: $\vdash f \frown (g \frown h) = (f \frown g) \frown h$ **by** (*rule SChopAssoc*)
from 2 3 **show** *?thesis* **by** *auto*
qed

lemma *AndSChopCommutate* :
 $\vdash (f \wedge f1) \frown g = (f1 \wedge f) \frown g$
proof –
have 1: $\vdash (f \wedge f1) = (f1 \wedge f)$ **by** *auto*
from 1 **show** *?thesis* **by** (*rule LeftSChopEqvSChop*)
qed

lemma *BfAndSChopImport*:
 $\vdash bff \wedge (f1 \frown g) \longrightarrow (f \wedge f1) \frown g$
proof –
have 1: $\vdash f \longrightarrow (f1 \longrightarrow f \wedge f1)$ **by** *auto*
hence 2: $\vdash bf\ f \longrightarrow bf\ (f1 \longrightarrow f \wedge f1)$ **by** (*rule BfImpBfRule*)
have 3: $\vdash bf\ (f1 \longrightarrow (f \wedge f1)) \longrightarrow f1 \frown g \longrightarrow (f \wedge f1) \frown g$ **by** (*rule BfSChopImpSChop*)
from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *BiAndSChopImport*:
 $\vdash bif \wedge (f1 \frown g) \longrightarrow (f \wedge f1) \frown g$
proof –
have 1: $\vdash f \longrightarrow (f1 \longrightarrow f \wedge f1)$ **by** *auto*
hence 2: $\vdash bi\ f \longrightarrow bi\ (f1 \longrightarrow f \wedge f1)$ **by** (*rule BiImpBiRule*)
have 3: $\vdash bi\ (f1 \longrightarrow (f \wedge f1)) \longrightarrow f1 \frown g \longrightarrow (f \wedge f1) \frown g$ **by** (*rule BiSChopImpSChop*)
from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *StateAndSChopImport*:

$\vdash (init\ w) \wedge (f \frown g) \longrightarrow ((init\ w) \wedge f) \frown g$
proof –
have 1: $\vdash (init\ w) \longrightarrow bf\ (init\ w)$ **by** (rule StateImpBf)
hence 2: $\vdash (init\ w) \wedge (f \frown g) \longrightarrow bf\ (init\ w) \wedge (f \frown g)$ **by** auto
have 3: $\vdash bf\ (init\ w) \wedge (f \frown g) \longrightarrow ((init\ w) \wedge f) \frown g$ **by** (rule BfAndSChopImport)
from 2 3 **show** ?thesis **using** MP **by** fastforce
qed

2.4.4 Further Properties Df and Bf

lemma AndFiniteImpDf:

$\vdash f \wedge finite \longrightarrow df\ f$
proof –
have 1: $\vdash finite \longrightarrow f \frown empty = f$ **by** (rule SChopEmpty)
have 2: $\vdash empty \longrightarrow \#True$ **by** auto
hence 3: $\vdash f \frown empty \longrightarrow f \frown \#True$ **by** (rule RightSChopImpSChop)
have 4: $\vdash f \wedge finite \longrightarrow f \frown \#True$ **using** 1 3 **by** fastforce
from 4 **show** ?thesis **by** (simp add: df-d-def)
qed

lemma DfState:

$\vdash df\ (init\ w) = (init\ w)$
proof –
have 0: $\vdash (init\ (\neg w)) \longrightarrow bf\ (init\ (\neg w))$ **using** StateImpBf **by** fastforce
hence 1: $\vdash \neg(init\ w) \longrightarrow bf\ (\neg (init\ w))$ **using** Initprop(2) **by** (metis inteq-reflection)
hence 2: $\vdash (\neg (init\ w)) \longrightarrow \neg (df\ (\neg \neg (init\ w)))$ **by** (simp add: bf-d-def)
have 3: $\vdash (\neg (init\ w) \longrightarrow \neg (df\ (\neg \neg (init\ w)))) \longrightarrow (df\ (\neg \neg (init\ w)) \longrightarrow (init\ w))$ **by** auto
have 4: $\vdash df\ (\neg \neg (init\ w)) \longrightarrow (init\ w)$ **using** 2 3 MP **by** blast
have 5: $\vdash (init\ w) \longrightarrow \neg \neg (init\ w)$ **by** auto
hence 6: $\vdash df\ (init\ w) \longrightarrow df\ (\neg \neg (init\ w))$ **by** (rule DfImpDf)
have 7: $\vdash df\ (init\ w) \longrightarrow (init\ w)$ **using** 6 4 **using** lift-imp-trans **by** metis
have 8: $\vdash (init\ w) \wedge finite \longrightarrow df\ (init\ w)$ **by** (rule AndFiniteImpDf)
from 7 8 **show** ?thesis
by (metis NowImpDiamond Prop10 StateAndChop df-d-def int-simps(17) inteq-reflection
lift-and-com schop-d-def sometimes-d-def)
qed

lemma StateSChop:

$\vdash (init\ w) \frown f \longrightarrow (init\ w)$
by (simp add: StateChopExportA schop-d-def)

lemma StateSChopExportA:

$\vdash ((init\ w) \wedge f) \frown g \longrightarrow (init\ w)$
by (meson AndSChopA StateSChop lift-imp-trans)

lemma StateAndSChop:

$\vdash ((init\ w) \wedge f) \frown g = ((init\ w) \wedge (f \frown g))$
by (*simp add: AndSChopB StateAndSChopImport StateSChopExportA Prop11 Prop12*)

lemma *StateAndSChopImpSChopRule*:

assumes $\vdash (init\ w) \wedge f \longrightarrow f1$
shows $\vdash (init\ w) \wedge (f \frown g) \longrightarrow (f1 \frown g)$
proof –
have 1: $\vdash (init\ w) \wedge f \longrightarrow f1$ **using** *assms* **by** *auto*
hence 2: $\vdash ((init\ w) \wedge f) \frown g \longrightarrow f1 \frown g$ **by** (*rule LeftSChopImpSChop*)
have 3: $\vdash ((init\ w) \wedge f) \frown g = ((init\ w) \wedge (f \frown g))$ **by** (*rule StateAndSChop*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *StateImpSChopEqvSChop* :

assumes $\vdash (init\ w) \longrightarrow (f = f1)$
shows $\vdash (init\ w) \longrightarrow ((f \frown g) = (f1 \frown g))$
proof –
have 1: $\vdash (init\ w) \longrightarrow (f = f1)$ **using** *assms* **by** *auto*
hence 2: $\vdash (init\ w) \wedge f \longrightarrow f1$ **by** *auto*
hence 3: $\vdash (init\ w) \wedge (f \frown g) \longrightarrow (f1 \frown g)$ **by** (*rule StateAndSChopImpSChopRule*)
have 4: $\vdash (init\ w) \wedge f1 \longrightarrow f$ **using** 1 **by** *auto*
hence 5: $\vdash (init\ w) \wedge (f1 \frown g) \longrightarrow (f \frown g)$ **by** (*rule StateAndSChopImpSChopRule*)
from 3 5 **show** *?thesis* **by** *fastforce*
qed

lemma *ChopEqvStateAndSChop*:

assumes $\vdash f = (init\ w) \wedge f1$
shows $\vdash (f \frown g) = ((init\ w) \wedge (f1 \frown g))$
proof –
have 1: $\vdash f = ((init\ w) \wedge f1)$ **using** *assms* **by** *auto*
hence 2: $\vdash f \frown g = (((init\ w) \wedge f1) \frown g)$ **by** (*rule LeftSChopEqvSChop*)
have 3: $\vdash ((init\ w) \wedge f1) \frown g = ((init\ w) \wedge (f1 \frown g))$ **by** (*rule StateAndSChop*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *DfIntro*:

$\vdash f \wedge finite \longrightarrow df\ f$
proof –
have 1: $\vdash finite \longrightarrow f \frown empty = f$ **by** (*rule SChopEmpty*)
have 2: $\vdash empty \longrightarrow \#True$ **by** *auto*
hence 3: $\vdash \Box(empty \longrightarrow \#True)$ **by** (*rule BoxGen*)
have 4: $\vdash \Box(empty \longrightarrow \#True) \longrightarrow (f; empty \longrightarrow f; \#True)$ **by** (*rule BoxChopImpChop*)
have 5: $\vdash f \frown empty \longrightarrow f \frown \#True$ **using** 3 4 *MP* **by** (*simp add: RightSChopImpSChop*)
hence 6: $\vdash f \frown empty \longrightarrow df\ f$ **by** (*simp add: df-d-def*)
from 1 6 **show** *?thesis* **using** *AndFiniteImpDf* **by** *blast*
qed

lemma *BfElim*:

$\vdash bf\ f \wedge finite \longrightarrow f$
proof –

have 1: $\vdash \neg f \wedge \text{finite} \longrightarrow \text{df } (\neg f)$ **by** (rule DfIntro)
have 2: $\vdash (\neg f \wedge \text{finite} \longrightarrow \text{df } (\neg f)) \longrightarrow (\neg(\text{df } (\neg f)) \longrightarrow \neg(\neg f \wedge \text{finite}))$ **by** simp
have 21: $\vdash \neg(\neg f \wedge \text{finite}) = (f \vee \text{inf})$ **by** (simp add: Valid-def finite-d-def)
have 3: $\vdash \neg (\text{df } (\neg f)) \longrightarrow f \vee \text{inf}$ **using** 1 2 21 **by** fastforce
from 3 **show** ?thesis **by** (simp add: Prop13 bf-d-def finite-d-def)
qed

lemma BfContraPosImpDist:

$\vdash \text{bf } (\neg g \longrightarrow \neg f) \longrightarrow (\text{bf } f) \longrightarrow (\text{bf } g)$
proof –
have 1: $\vdash \text{bf } (\neg g \longrightarrow \neg f) \longrightarrow (\text{df } (\neg g)) \longrightarrow (\text{df } (\neg f))$ **by** (rule BfImpDfImpDf)
hence 2: $\vdash \text{bf } (\neg g \longrightarrow \neg f) \longrightarrow (\neg(\text{df } (\neg f))) \longrightarrow (\neg(\text{df } (\neg g)))$ **by** auto
from 2 **show** ?thesis **by** (metis bf-d-def)
qed

lemma BfImpDist:

$\vdash \text{bf } (f \longrightarrow g) \longrightarrow (\text{bf } f) \longrightarrow (\text{bf } g)$
proof –
have 1: $\vdash (f \longrightarrow g) \longrightarrow (\neg g \longrightarrow \neg f)$ **by** auto
hence 2: $\vdash \neg(\neg g \longrightarrow \neg f) \longrightarrow \neg(f \longrightarrow g)$ **by** auto
hence 3: $\vdash \text{bf } (\neg(\neg g \longrightarrow \neg f) \longrightarrow \neg(f \longrightarrow g))$ **by** (rule BfGen)
have 4: $\vdash \text{bf } (\neg(\neg g \longrightarrow \neg f) \longrightarrow \neg(f \longrightarrow g))$
 \longrightarrow
 $\text{bf } (f \longrightarrow g) \longrightarrow \text{bf } (\neg g \longrightarrow \neg f)$ **by** (rule BfContraPosImpDist)
have 5: $\vdash \text{bf } (f \longrightarrow g) \longrightarrow \text{bf } (\neg g \longrightarrow \neg f)$ **using** 3 4 **MP** **by** blast
have 6: $\vdash \text{bf } (\neg g \longrightarrow \neg f) \longrightarrow (\text{bf } f) \longrightarrow (\text{bf } g)$ **by** (rule BfContraPosImpDist)
from 5 6 **show** ?thesis **using** lift-imp-trans **by** blast
qed

lemma FiniteImpBfImpBfRule:

assumes $\vdash \text{finite} \longrightarrow (f \longrightarrow g)$
shows $\vdash \text{bf } f \longrightarrow \text{bf } g$
proof –
have 1: $\vdash \text{finite} \longrightarrow f \longrightarrow g$ **using** assms **by** auto
have 2: $\vdash \text{bf}(f \longrightarrow g)$ **using** 1 **by** (simp add: FiniteBfGen)
have 3: $\vdash \text{bf}(f \longrightarrow g) \longrightarrow \text{bf } f \longrightarrow \text{bf } g$ **using** BfImpDist **by** blast
from 2 3 **show** ?thesis **by** fastforce
qed

lemma FiniteImpBfEqvRule:

assumes $\vdash \text{finite} \longrightarrow (f = g)$
shows $\vdash \text{bf } f = \text{bf } g$
proof –
have 1: $\vdash \text{finite} \longrightarrow (f = g)$ **using** assms **by** blast
have 2: $\vdash \text{finite} \longrightarrow (f \longrightarrow g)$ **using** 1 **by** auto
have 3: $\vdash \text{bf } f \longrightarrow \text{bf } g$ **by** (simp add: 2 FiniteImpBfImpBfRule)
qed

have 4: $\vdash \text{finite} \longrightarrow (g \longrightarrow f)$ **using** 1 **by** *auto*
have 5: $\vdash \text{bf } g \longrightarrow \text{bf } f$ **by** (*simp add: 4 FiniteImpBfImpBfRule*)
from 3 5 **show** ?thesis **by** *fastforce*
qed

lemma *IfSChopEqvRule*:

assumes $\vdash f = \text{if}_i (\text{init } w) \text{ then } f1 \text{ else } f2$
shows $\vdash f \frown g = \text{if}_i (\text{init } w) \text{ then } (f1 \frown g) \text{ else } (f2 \frown g)$
proof –
have 1: $\vdash f = \text{if}_i (\text{init } w) \text{ then } f1 \text{ else } f2$
using *assms* **by** *auto*
hence 2: $\vdash f = (((\text{init } w) \wedge f1) \vee ((\text{init } (\neg w)) \wedge f2))$
unfolding *ifthenelse-d-def* **by** (*metis Initprop(2) int-eq*)
hence 3: $\vdash f \frown g = (((\text{init } w) \wedge f1) \frown g \vee ((\text{init } (\neg w)) \wedge f2) \frown g)$
by (*rule OrSChopEqvRule*)
have 4: $\vdash ((\text{init } w) \wedge f1) \frown g = ((\text{init } w) \wedge (f1 \frown g))$
by (*rule StateAndSChop*)
have 5: $\vdash ((\text{init } (\neg w)) \wedge f2) \frown g = ((\text{init } (\neg w)) \wedge (f2 \frown g))$
by (*rule StateAndSChop*)
have 6: $\vdash f \frown g = (((\text{init } w) \wedge f1 \frown g) \vee ((\text{init } (\neg w)) \wedge f2 \frown g))$
using 3 4 5 **by** *fastforce*
from 6 **show** ?thesis **unfolding** *ifthenelse-d-def* **by** (*metis Initprop(2) integ-reflection*)
qed

lemma *SChopOrEqvRule*:

assumes $\vdash g = (g1 \vee g2)$
shows $\vdash f \frown g = ((f \frown g1) \vee (f \frown g2))$
proof –
have 1: $\vdash g = (g1 \vee g2)$ **using** *assms* **by** *auto*
hence 2: $\vdash f \frown g = (f \frown (g1 \vee g2))$ **by** (*rule RightSChopEqvSChop*)
have 3: $\vdash f \frown (g1 \vee g2) = (f \frown g1 \vee f \frown g2)$ **by** (*rule SChopOrEqv*)
from 2 3 **show** ?thesis **by** *fastforce*
qed

lemma *EmptyOrSChopEqv*:

$\vdash (\text{empty} \vee f) \frown g = (g \vee (f \frown g))$
proof –
have 1: $\vdash (\text{empty} \vee f) \frown g = ((\text{empty} \frown g) \vee (f \frown g))$ **by** (*rule OrSChopEqv*)
have 2: $\vdash \text{empty} \frown g = g$ **by** (*rule EmptySChop*)
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *EmptyOrNextSChopEqv*:

$\vdash (\text{empty} \vee \circ f) \frown g = (g \vee \circ(f \frown g))$
proof –
have 1: $\vdash (\text{empty} \vee \circ f) \frown g = (g \vee ((\circ f) \frown g))$ **by** (*rule EmptyOrSChopEqv*)
have 2: $\vdash (\circ f) \frown g = \circ(f \frown g)$ **by** (*rule NextSChop*)
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *EmptyOrSChopImpRule*:
assumes $\vdash f \longrightarrow \text{empty} \vee f1$
shows $\vdash f \frown g \longrightarrow g \vee (f1 \frown g)$
proof –
have 1: $\vdash f \longrightarrow \text{empty} \vee f1$ **using** *assms* **by** *auto*
hence 2: $\vdash f \frown g \longrightarrow (\text{empty} \vee f1) \frown g$ **by** (*rule LeftSChopImpSChop*)
have 3: $\vdash (\text{empty} \vee f1) \frown g = (g \vee (f1 \frown g))$ **by** (*rule EmptyOrSChopEqv*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrSChopEqvRule*:
assumes $\vdash f = (\text{empty} \vee f1)$
shows $\vdash f \frown g = (g \vee (f1 \frown g))$
proof –
have 1: $\vdash f = (\text{empty} \vee f1)$ **using** *assms* **by** *auto*
hence 2: $\vdash f \frown g = ((\text{empty} \vee f1) \frown g)$ **by** (*rule LeftSChopEqvSChop*)
have 3: $\vdash (\text{empty} \vee f1) \frown g = (g \vee (f1 \frown g))$ **by** (*rule EmptyOrSChopEqv*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextSChopImpRule*:
assumes $\vdash f \longrightarrow \text{empty} \vee \circ f1$
shows $\vdash f \frown g \longrightarrow g \vee \circ(f1 \frown g)$
proof –
have 1: $\vdash f \longrightarrow \text{empty} \vee \circ f1$ **using** *assms* **by** *auto*
hence 2: $\vdash f \frown g \longrightarrow (\text{empty} \vee \circ f1) \frown g$ **by** (*rule LeftSChopImpSChop*)
have 3: $\vdash (\text{empty} \vee \circ f1) \frown g = (g \vee \circ(f1 \frown g))$ **by** (*rule EmptyOrNextSChopEqv*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextSChopEqvRule*:
assumes $\vdash f = (\text{empty} \vee \circ f1)$
shows $\vdash f \frown g = (g \vee \circ(f1 \frown g))$
proof –
have 1: $\vdash f = (\text{empty} \vee \circ f1)$ **using** *assms* **by** *auto*
hence 2: $\vdash f \frown g = ((\text{empty} \vee \circ f1) \frown g)$ **by** (*rule LeftSChopEqvSChop*)
have 3: $\vdash (\text{empty} \vee \circ f1) \frown g = (g \vee \circ(f1 \frown g))$ **by** (*rule EmptyOrNextSChopEqv*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *SChopEmptyOrImpRule*:
assumes $\vdash g \longrightarrow \text{empty} \vee g1$
shows $\vdash f \frown g \wedge \text{finite} \longrightarrow f \vee (f \frown g1)$
proof –
have 1: $\vdash g \longrightarrow \text{empty} \vee g1$ **using** *assms* **by** *auto*
hence 2: $\vdash f \frown g \longrightarrow (f \frown \text{empty}) \vee (f \frown g1)$ **by** (*rule SChopOrImpRule*)
have 3: $\vdash \text{finite} \longrightarrow f \frown \text{empty} = f$ **by** (*rule SChopEmpty*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateSChopBoxAndInfImpBox:*

$\vdash \Box (init\ w) \frown \Box (init\ w) \wedge inf \longrightarrow \Box (init\ w)$

by (*metis AndChopA BoxStateChopBoxEqvBox OrFiniteInf Prop03 int-eq lift-imp-trans schop-d-def*)

lemma *BoxStateSChopBoxEqvBox:*

$\vdash \Box (init\ w) \frown \Box (init\ w) = \Box (init\ w)$

proof –

have 1: $\vdash (\Box (init\ w)) = ((init\ w) \wedge (empty \vee \bigcirc(\Box (init\ w))))$

by (*rule BoxEqvAndEmptyOrNextBox*)

hence 2: $\vdash (\Box (init\ w) \frown \Box (init\ w)) =$

$((init\ w) \wedge ((empty \vee \bigcirc(\Box (init\ w))) \frown \Box (init\ w)))$

by (*metis StateAndSChop integ-reflection*)

have 3: $\vdash ((empty \vee \bigcirc(\Box (init\ w))) \frown \Box (init\ w)) =$

$(\Box (init\ w) \vee \bigcirc(\Box (init\ w) \frown \Box (init\ w)))$

by (*rule EmptyOrNextSChopEqv*)

have 4: $\vdash (\Box (init\ w) \frown \Box (init\ w)) =$

$((init\ w) \wedge (\Box (init\ w) \vee \bigcirc(\Box (init\ w) \frown \Box (init\ w))))$

using 2 3 **by** *fastforce*

have 5: $\vdash \neg(\Box (init\ w)) \longrightarrow \neg (init\ w) \vee \neg(\bigcirc(\Box (init\ w)))$

by (*rule NotBoxImpNotOrNotNextBox*)

have 6: $\vdash (\Box (init\ w) \frown \Box (init\ w)) \wedge \neg(\Box (init\ w)) \longrightarrow$

$\bigcirc(\Box (init\ w) \frown \Box (init\ w)) \wedge \neg(\bigcirc(\Box (init\ w)))$

using 4 5 **by** *fastforce*

hence 7: $\vdash \Box (init\ w) \frown \Box (init\ w) \wedge finite \longrightarrow \Box (init\ w)$

by (*rule NextContra*)

have 8: $\vdash \Box (init\ w) \frown \Box (init\ w) \wedge inf \longrightarrow \Box (init\ w)$

by (*rule BoxStateSChopBoxAndInfImpBox*)

have 9: $\vdash \Box (init\ w) \frown \Box (init\ w) \wedge (finite \vee inf) \longrightarrow \Box (init\ w)$

using 7 8 **by** *fastforce*

hence 10: $\vdash \Box (init\ w) \frown \Box (init\ w) \longrightarrow \Box (init\ w)$

using *FiniteOrInfinite* **by** *fastforce*

have 11: $\vdash \Box (init\ w) = ((init\ w) \wedge \Box (init\ w))$

by (*rule BoxEqvAndBox*)

have 12: $\vdash empty \frown \Box (init\ w) = \Box (init\ w)$

by (*rule EmptySChop*)

have 13: $\vdash ((init\ w) \wedge empty) \frown \Box (init\ w) = ((init\ w) \wedge (empty \frown \Box (init\ w)))$

by (*rule StateAndSChop*)

have 14: $\vdash \Box (init\ w) = ((init\ w) \wedge empty) \frown \Box (init\ w)$

using 11 12 13 **by** *fastforce*

have 15: $\vdash (init\ w) \wedge empty \longrightarrow \Box (init\ w)$

by (*rule StateAndEmptyImpBoxState*)

hence 16: $\vdash ((init\ w) \wedge empty) \frown \Box (init\ w) \longrightarrow \Box (init\ w) \frown \Box (init\ w)$

by (*rule LeftSChopImpSChop*)

have 17: $\vdash \Box (init\ w) \longrightarrow \Box (init\ w) \frown \Box (init\ w)$

using 14 16 **by** *fastforce*

from 10 17 **show** *?thesis* **by** *fastforce*

qed

lemma *NotBoxStateImpBoxSYieldsNotBox:*

$\vdash \neg(\Box (init\ w)) \longrightarrow (\Box (init\ w))\ syields\ (\neg(\Box (init\ w)))$

proof –
have 1: $\vdash \Box (init\ w) \frown \Box (init\ w) = \Box (init\ w)$ **by** (rule *BoxStateSChopBoxEqvBox*)
have 2: $\vdash \Box (init\ w) = (\neg \neg (\Box (init\ w)))$ **by** *auto*
hence 3: $\vdash \Box (init\ w) \frown \Box (init\ w) = \Box (init\ w) \frown (\neg \neg (\Box (init\ w)))$ **by** (rule *RightSChopEqvSChop*)
have 4: $\vdash \neg (\Box (init\ w)) \longrightarrow \neg (\Box (init\ w) \frown (\neg \neg (\Box (init\ w))))$ **using** 1 3 **by** *auto*
from 4 **show** ?thesis **by** (simp add: syields-d-def)
qed

lemma *StateEqvBf*:

$\vdash (init\ w) = bf\ (init\ w)$

proof –

have 1: $\vdash (init\ w) \longrightarrow bf\ (init\ w)$ **by** (rule *StateImpBf*)
have 2: $\vdash bf\ (init\ w) \wedge finite \longrightarrow (init\ w)$ **by** (rule *BfElim*)
from 1 2 **show** ?thesis
by (metis (no-types) *DfState Initprop(2) bf-d-def int-simps(4) inteq-reflection*)

qed

lemma *TrueSChopEqvDiamond*:

$\vdash \#True \frown f = \Diamond f$

using *DiamondSChopdef* **by** *fastforce*

lemma *BfAndEqvBfAndBf*:

$\vdash bf(f \wedge g) = (bf\ f \wedge bf\ g)$

proof –

have 1: $\vdash f \wedge g \longrightarrow f$ **by** *auto*
have 2: $\vdash bf(f \wedge g) \longrightarrow bf\ f$ **by** (simp add: 1 *BfImpBfRule*)
have 3: $\vdash f \wedge g \longrightarrow g$ **by** *auto*
have 4: $\vdash bf(f \wedge g) \longrightarrow bf\ g$ **by** (simp add: 3 *BfImpBfRule*)
have 5: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ **by** *auto*
have 6: $\vdash bf\ f \longrightarrow bf\ (g \longrightarrow f \wedge g)$ **by** (simp add: 5 *BfImpBfRule*)
have 7: $\vdash bf\ (g \longrightarrow f \wedge g) \longrightarrow (bf\ g \longrightarrow bf(f \wedge g))$ **by** (simp add: *BfImpDist*)
have 8: $\vdash bf\ f \wedge bf\ g \longrightarrow bf\ (f \wedge g)$ **using** 6 7 **by** *fastforce*
from 2 4 8 **show** ?thesis **by** *fastforce*

qed

lemma *BfEqvBfImpAndBfImp*:

$\vdash bf(f = g) = (bf\ (f \longrightarrow g) \wedge bf(g \longrightarrow f))$

proof –

have 1: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** *auto*
have 2: $\vdash bf(f = g) = bf((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** (simp add: 1 *BfEqvBf*)
have 3: $\vdash bf((f \longrightarrow g) \wedge (g \longrightarrow f)) = (bf(f \longrightarrow g) \wedge bf(g \longrightarrow f))$ **by** (simp add: *BfAndEqvBfAndBf*)
from 2 3 **show** ?thesis **by** *fastforce*

qed

lemma *BfEqvImpSChopEqvSChop*:

$\vdash bf(f = f1) \longrightarrow f \frown g = f1 \frown g$

```

proof –
  have 1:  $\vdash \text{bf}(f = f1) = (\text{bf } (f \longrightarrow f1) \wedge \text{bf}(f1 \longrightarrow f))$  by (simp add: BfEqvBfImpAndBfImp)
  have 2:  $\vdash \text{bf } (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$  by (simp add: BfSCHopImpSCHop)
  have 3:  $\vdash \text{bf}(f1 \longrightarrow f) \longrightarrow f1 \frown g \longrightarrow f \frown g$  by (simp add: BfSCHopImpSCHop)
  from 1 2 3 show ?thesis by fastforce
qed

```

```

lemma BfEqvDfEqvDf:
   $\vdash \text{bf}(f = g) \longrightarrow (\text{df } f = \text{df } g)$ 
proof –
  have 1:  $\vdash \text{bf}(f = g) \longrightarrow (f \frown \#True) = (g \frown \#True)$ 
    using BfEqvImpSCHopEqvSCHop by fastforce
  from 1 show ?thesis by (simp add: df-d-def)
qed

```

```

lemma FiniteImpEqvDfImpRule:
  assumes  $\vdash \text{finite} \longrightarrow f = g$ 
  shows  $\vdash \text{df } f = \text{df } g$ 
proof –
  have 1:  $\vdash \text{finite} \longrightarrow f = g$  using assms by auto
  have 2:  $\vdash \text{bf}(f = g)$  using 1 by (simp add: FiniteBfGen)
  have 3:  $\vdash \text{bf}(f = g) \longrightarrow (\text{df } f = \text{df } g)$  by (simp add: BfEqvDfEqvDf)
  from 2 3 show ?thesis by fastforce
qed

```

```

lemma DfEmpty:
   $\vdash \text{df } \text{empty}$ 
proof –
  have 1:  $\vdash \#True$  by auto
  have 2:  $\vdash \text{empty} \frown \#True = \#True$  by (rule EmptySCHop)
  have 3:  $\vdash \text{empty} \frown \#True$  using 1 2 by auto
  from 3 show ?thesis by (simp add: df-d-def)
qed

```

```

lemma BfImpDf:
   $\vdash \text{bf } f \longrightarrow \text{df } f$ 
proof –
  have 1:  $\vdash f \longrightarrow (\text{empty} \longrightarrow f)$  by auto
  have 2:  $\vdash \text{bf } f \longrightarrow \text{bf}(\text{empty} \longrightarrow f)$  by (simp add: 1 BfImpBfRule)
  have 3:  $\vdash \text{bf}(\text{empty} \longrightarrow f) \longrightarrow \text{df } \text{empty} \longrightarrow \text{df } f$  by (simp add: BfImpDfImpDf)
  have 4:  $\vdash \text{bf } f \longrightarrow \text{df } \text{empty} \longrightarrow \text{df } f$  using 2 3 lift-imp-trans by blast
  have 5:  $\vdash \text{df } \text{empty}$  by (simp add: DfEmpty)
  from 4 5 show ?thesis by fastforce
qed

```

2.4.5 Properties of SDa and SBa

lemma *SDaEqvDtDf*:

$\vdash \text{sda } f = \Diamond (df \ f)$

proof –

have 1: $\vdash \#True \frown (f \frown \#True) = \#True \frown (f \frown \#True)$ **by** *auto*

hence 2: $\vdash \#True \frown (f \frown \#True) = \#True \frown df \ f$ **by** (*simp add: df-d-def*)

have 3: $\vdash \#True \frown (df \ f) = \Diamond (df \ f)$ **by** (*simp add: TrueSChopEqvDiamond*)

have 4: $\vdash \#True \frown (f \frown \#True) = \Diamond (df \ f)$ **using** 2 3 **by** *fastforce*

from 4 **show** *?thesis* **by** (*simp add: sda-d-def*)

qed

lemma *SDaEqvDfDt*:

$\vdash \text{sda } f = df (\Diamond f)$

proof –

have 1: $\vdash \#True \frown f = \Diamond f$ **by** (*rule TrueSChopEqvDiamond*)

hence 2: $\vdash (\#True \frown f) \frown \#True = (\Diamond f) \frown \#True$ **by** (*rule LeftSChopEqvSChop*)

hence 3: $\vdash (\#True \frown f) \frown \#True = df (\Diamond f)$ **by** (*simp add: df-d-def*)

have 4: $\vdash \#True \frown (f \frown \#True) = (\#True \frown f) \frown \#True$ **by** (*rule SChopAssoc*)

have 5: $\vdash \#True \frown (f \frown \#True) = df (\Diamond f)$ **using** 3 4 **by** *fastforce*

from 5 **show** *?thesis* **by** (*simp add: sda-d-def*)

qed

lemma *DtDfEqvDfDt*:

$\vdash \Diamond (df \ f) = df (\Diamond f)$

by (*meson Prop04 SDaEqvDfDt SDaEqvDtDf*)

lemma *SBaEqvBfBt*:

$\vdash \text{sba } f = bf (\Box f)$

proof –

have 1: $\vdash \text{sda } (\neg f) = df (\Diamond (\neg f))$ **by** (*rule SDaEqvDfDt*)

have 2: $\vdash \Diamond (\neg f) = (\neg (\Box f))$ **by** (*rule DiamondNotEqvNotBox*)

hence 3: $\vdash df (\Diamond (\neg f)) = df (\neg (\Box f))$ **by** (*rule DfEqvDf*)

have 4: $\vdash \text{sda } (\neg f) = df (\neg (\Box f))$ **using** 1 3 **by** *fastforce*

hence 5: $\vdash (\neg (\text{sda } (\neg f))) = (\neg (df (\neg (\Box f))))$ **by** *auto*

hence 6: $\vdash (\neg (\text{sda } (\neg f))) = bf (\Box f)$ **by** (*simp add: bf-d-def*)

from 6 **show** *?thesis* **by** (*simp add: sba-d-def*)

qed

lemma *DfNotEqvNotBf*:

$\vdash df (\neg f) = (\neg (bf \ f))$

proof –

have 1: $\vdash bf \ f = (\neg (df (\neg f)))$ **by** (*simp add: bf-d-def*)

from 1 **show** *?thesis* **by** *auto*

qed

lemma *DfDfNotEqvNotBfBf*:

$\vdash df (df (\neg f)) = (\neg (bf (bf \ f)))$

proof –

have 1: $\vdash df (\neg f) = (\neg bf f)$ **by** (*simp add: DfNotEqvNotBf*)
have 2: $\vdash df (df (\neg f)) = df (\neg bf f)$ **by** (*simp add: 1 DfEqvDf*)
have 3: $\vdash df (\neg bf f) = (\neg bf (bf f))$ **by** (*simp add: DfNotEqvNotBf*)
from 2 3 **show** ?thesis **by** fastforce
qed

lemma *DfDtEqvDtDf*:
 $\vdash df(\Diamond f) = \Diamond(df f)$
proof –
have 1: $\vdash (\#True \frown f) \frown \#True = \#True \frown (f \frown \#True)$
using *SChopAssoc* **by** fastforce
have 2: $\vdash (\Diamond f) \frown \#True = \Diamond(f \frown \#True)$
using 1 **by** (*metis TrueSChopEqvDiamond int-eq*)
from 1 2 **show** ?thesis **by** (*simp add: df-d-def*)
qed

lemma *DfDtNotEqvNotBfBt*:
 $\vdash df(\Diamond(\neg f)) = (\neg(bf(\Box f)))$
proof –
have 1: $\vdash \Diamond(\neg f) = (\neg(\Box f))$ **by** (*simp add: DiamondNotEqvNotBox*)
have 2: $\vdash df(\Diamond(\neg f)) = df(\neg(\Box f))$ **by** (*simp add: 1 DfEqvDf*)
have 3: $\vdash df(\neg(\Box f)) = (\neg(bf(\Box f)))$ **by** (*simp add: DfNotEqvNotBf*)
from 2 3 **show** ?thesis **by** fastforce
qed

lemma *DtDfNotEqvNotBtBf*:
 $\vdash \Diamond(df(\neg f)) = (\neg(\Box(bf f)))$
proof –
have 1: $\vdash df(\neg f) = (\neg(bf f))$ **using** *DfNotEqvNotBf* **by** blast
have 2: $\vdash \Diamond(df(\neg f)) = \Diamond(\neg(bf f))$ **by** (*simp add: 1 DiamondEqvDiamond*)
have 3: $\vdash \Diamond(\neg(bf f)) = (\neg \Box(bf f))$ **by** (*simp add: DiamondNotEqvNotBox*)
from 2 3 **show** ?thesis **by** fastforce
qed

lemma *SBaEqvBtBf*:
 $\vdash sba f = \Box(bf f)$
proof –
have 1: $\vdash sda(\neg f) = \Diamond(df(\neg f))$ **by** (*rule SDaEqvDtDf*)
have 2: $\vdash df(\neg f) = (\neg(bf f))$ **by** (*rule DfNotEqvNotBf*)
hence 3: $\vdash \Diamond(df(\neg f)) = \Diamond(\neg(bf f))$ **by** (*rule DiamondEqvDiamond*)
have 4: $\vdash (\neg(\Diamond(\neg(bf f)))) = \Box(bf f)$ **by** (*rule NotDiamondNotEqvBox*)
have 5: $\vdash (\neg(sda(\neg f))) = \Box(bf f)$ **using** 1 2 3 4 **by** fastforce
from 5 **show** ?thesis **by** (*simp add: sba-d-def*)
qed

lemma *BaImpSBa*:
 $\vdash ba f \longrightarrow sba f$

using *BaEqvBiBt BiImpBf SBaEqvBfBt* **by** *fastforce*

lemma *SDaImpDa*:

$\vdash sda\ f \longrightarrow da\ f$

proof $-$

have 1: $\vdash ba\ (\neg f) \longrightarrow sba\ (\neg f)$

using *BaImpSBa* **by** *blast*

have 2: $\vdash \neg sba\ (\neg f) \longrightarrow \neg ba\ (\neg f)$

using 1 **by** *fastforce*

from 2 **show** *?thesis* **by** (*simp add: sba-d-def ba-d-def*)

qed

lemma *BtBfEqvBfBt*:

$\vdash \Box (bf\ f) = bf(\Box\ f)$

proof $-$

have 1: $\vdash sba\ f = \Box (bf\ f)$ **by** (*rule SBaEqvBtBf*)

have 2: $\vdash sba\ f = bf(\Box\ f)$ **by** (*rule SBaEqvBfBt*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *BoxStateEqvSBaBoxState*:

$\vdash \Box (init\ w) = sba\ (\Box (init\ w))$

proof $-$

have 1: $\vdash (init\ w) = bf\ (init\ w)$ **by** (*rule StateEqvBf*)

hence 2: $\vdash \Box (init\ w) = \Box (bf\ (init\ w))$ **by** (*rule BoxEqvBox*)

have 3: $\vdash \Box (bf\ (init\ w)) = bf(\Box (init\ w))$ **by** (*rule BtBfEqvBfBt*)

have 4: $\vdash \Box (init\ w) = \Box(\Box (init\ w))$ **by** (*rule BoxEqvBoxBox*)

hence 5: $\vdash bf(\Box (init\ w)) = bf(\Box(\Box (init\ w)))$ **by** (*rule BfEqvBf*)

have 6: $\vdash sba(\Box (init\ w)) = bf(\Box(\Box (init\ w)))$ **by** (*rule SBaEqvBfBt*)

from 2 3 5 6 **show** *?thesis* **by** *fastforce*

qed

lemma *SBaImpBf*:

$\vdash sba\ f \longrightarrow bf\ f$

proof $-$

have 1: $\vdash sba\ f = \Box(bf\ f)$ **by** (*rule SBaEqvBtBf*)

have 2: $\vdash \Box(bf\ f) \longrightarrow bf\ f$ **by** (*rule BoxElim*)

from 1 2 **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*

qed

lemma *BaImpBf*:

$\vdash ba\ f \longrightarrow bf\ f$

proof $-$

have 1: $\vdash ba\ f = \Box(bi\ f)$ **by** (*rule BaEqvBtBi*)

have 2: $\vdash \Box(bi\ f) \longrightarrow bi\ f$ **by** (*rule BoxElim*)

have 3: $\vdash bi\ f \longrightarrow bf\ f$ **by** (*simp add: BiImpBf*)

from 1 2 3 **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*

qed

lemma *SBaImpBt*:

$\vdash \text{ sba } f \wedge \text{ finite } \longrightarrow \Box f$

proof –

have 1: $\vdash \text{ sba } f = \text{ bf } (\Box f)$ **by** (rule *SBaEqvBfBt*)

have 2: $\vdash \text{ bf } (\Box f) \wedge \text{ finite } \longrightarrow \Box f$ **by** (rule *BfElim*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *DiamondImpSDa*:

$\vdash \Diamond f \wedge \text{ finite } \longrightarrow \text{ sda } f$

by (metis *AndFiniteImpDf SDaEqvDfDt inteq-reflection*)

lemma *DfImpSDa*:

$\vdash \text{ df } f \longrightarrow \text{ sda } f$

using *NowImpDiamond SDaEqvDtDf* **by** fastforce

lemma *BoxAndSChopImport*:

$\vdash \Box h \wedge f \frown g \longrightarrow f \frown (h \wedge g)$

proof –

have 1: $\vdash h \longrightarrow g \longrightarrow (h \wedge g)$ **by** auto

hence 2: $\vdash \Box h \longrightarrow \Box(g \longrightarrow (h \wedge g))$ **by** (rule *ImpBoxRule*)

have 3: $\vdash \Box(g \longrightarrow (h \wedge g)) \longrightarrow f \frown g \longrightarrow f \frown (h \wedge g)$ **by** (rule *BoxSChopImpSChop*)

from 2 3 **show** ?thesis **by** fastforce

qed

lemma *SBaAndSChopImport*:

$\vdash \text{ sba } f \wedge \text{ finite } \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown (f \wedge g1)$

proof –

have 1: $\vdash \text{ sba } f \longrightarrow \text{ bf } f$ **by** (rule *SBaImpBf*)

have 2: $\vdash \text{ bf } f \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown g1$ **by** (rule *BfAndSChopImport*)

have 3: $\vdash \text{ sba } f \wedge \text{ finite } \longrightarrow \Box f$ **by** (rule *SBaImpBt*)

have 4: $\vdash \Box f \wedge (f \wedge g) \frown g1 \longrightarrow (f \wedge g) \frown (f \wedge g1)$ **by** (rule *BoxAndSChopImport*)

from 1 2 3 4 **show** ?thesis **by** fastforce

qed

lemma *BaAndSChopImport*:

$\vdash \text{ ba } f \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown (f \wedge g1)$

proof –

have 1: $\vdash \text{ ba } f \longrightarrow \text{ bi } f$ **by** (rule *BaImpBi*)

have 2: $\vdash \text{ bi } f \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown g1$ **by** (rule *BiAndSChopImport*)

have 3: $\vdash \text{ ba } f \longrightarrow \Box f$ **by** (rule *BaImpBt*)

have 4: $\vdash \Box f \wedge (f \wedge g) \frown g1 \longrightarrow (f \wedge g) \frown (f \wedge g1)$ **by** (rule *BoxAndSChopImport*)

from 1 2 3 4 **show** ?thesis **by** fastforce

qed

lemma *SChopAndCommute*:

$\vdash f \frown (g \wedge g1) = f \frown (g1 \wedge g)$

proof –

have 1: $\vdash (g \wedge g1) = (g1 \wedge g)$ **by** auto

from 1 **show** ?thesis **by** (rule *RightSChopEqvSChop*)

qed

lemma *SChopAndA*:

$\vdash f \frown (g \wedge g1) \longrightarrow f \frown g$

proof –

have 1: $\vdash (g \wedge g1) \longrightarrow g$ **by** *auto*

from 1 **show** *?thesis* **by** (rule *RightSChopImpSChop*)

qed

lemma *SChopAndB*:

$\vdash f \frown (g \wedge g1) \longrightarrow f \frown g1$

proof –

have 1: $\vdash (g \wedge g1) \longrightarrow g1$ **by** *auto*

from 1 **show** *?thesis* **by** (rule *RightSChopImpSChop*)

qed

lemma *BoxStateAndSChopEqvSChop*:

$\vdash (\Box (init\ w) \wedge finite \wedge (f \frown g)) = ((\Box (init\ w) \wedge f) \frown (\Box (init\ w) \wedge g) \wedge finite)$

proof –

have 1: $\vdash \Box (init\ w) = sba(\Box (init\ w))$

by (rule *BoxStateEqvSBaBoxState*)

have 2: $\vdash sba(\Box (init\ w)) \wedge finite \wedge (f \frown g) \longrightarrow (\Box (init\ w) \wedge f) \frown (\Box (init\ w) \wedge g)$

by (rule *SBaAndSChopImport*)

have 3: $\vdash \Box (init\ w) \wedge finite \wedge (f \frown g) \longrightarrow (\Box (init\ w) \wedge f) \frown (\Box (init\ w) \wedge g)$

using 1 2 **by** *fastforce*

have 11: $\vdash (\Box (init\ w) \wedge f) \frown (\Box (init\ w) \wedge g) \longrightarrow (\Box (init\ w)) \frown (\Box (init\ w) \wedge g)$

by (rule *AndSChopA*)

have 12: $\vdash (\Box (init\ w)) \frown (\Box (init\ w) \wedge g) \longrightarrow (\Box (init\ w)) \frown (\Box (init\ w))$

by (rule *SChopAndA*)

have 13: $\vdash (\Box (init\ w)) \frown (\Box (init\ w)) = \Box (init\ w)$

by (rule *BoxStateSChopBoxEqvBox*)

have 14: $\vdash (\Box (init\ w) \wedge f) \frown (\Box (init\ w) \wedge g) \longrightarrow f \frown (\Box (init\ w) \wedge g)$

by (rule *AndSChopB*)

have 15: $\vdash f \frown (\Box (init\ w) \wedge g) \longrightarrow f \frown g$

by (rule *SChopAndB*)

have 16: $\vdash (\Box (init\ w) \wedge f) \frown (\Box (init\ w) \wedge g) \longrightarrow \Box (init\ w) \wedge (f \frown g)$

using 11 12 13 14 15 **by** *fastforce*

from 3 16 **show** *?thesis* **by** *fastforce*

qed

lemma *DfEqvNotBfNot*:

$\vdash df\ f = (\neg (bf\ (\neg\ f)))$

proof –

have 1: $\vdash bf\ (\neg\ f) = (\neg (df\ (\neg\ \neg\ f)))$ **by** (*simp add: bf-d-def*)

hence 2: $\vdash df\ (\neg\ \neg\ f) = (\neg (bf\ (\neg\ f)))$ **by** *auto*

have 3: $\vdash f = (\neg\ \neg\ f)$ **by** *auto*

hence 4: $\vdash df\ f = df\ (\neg\ \neg\ f)$ **by** (rule *DfEqvDf*)

from 2 4 **show** *?thesis* **by** *auto*

qed

lemma *SChopAndBoxImport*:

$\vdash f \frown g \wedge \Box h \longrightarrow f \frown (g \wedge h)$

proof –

have 1: $\vdash \Box h \wedge f \frown g \longrightarrow f \frown (h \wedge g)$ **by** (rule *BoxAndSChopImport*)

have 2: $\vdash f \frown (h \wedge g) = f \frown (g \wedge h)$ **by** (rule *SChopAndCommute*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *AndSChopAndCommute*:

$\vdash (f \wedge g) \frown (f1 \wedge g1) = (g \wedge f) \frown (g1 \wedge f1)$

proof –

have 1: $\vdash (f \wedge g) \frown (f1 \wedge g1) = (g \wedge f) \frown (f1 \wedge g1)$ **by** (rule *AndSChopCommute*)

have 2: $\vdash (g \wedge f) \frown (f1 \wedge g1) = (g \wedge f) \frown (g1 \wedge f1)$ **by** (rule *SChopAndCommute*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *SChopImpSChop*:

assumes $\vdash f \longrightarrow f1$

$\vdash g \longrightarrow g1$

shows $\vdash f \frown g \longrightarrow f1 \frown g1$

proof –

have 1: $\vdash f \longrightarrow f1$ **using** *assms* **by** auto

hence 2: $\vdash f \frown g \longrightarrow f1 \frown g$ **by** (rule *LeftSChopImpSChop*)

have 3: $\vdash g \longrightarrow g1$ **using** *assms* **by** auto

hence 4: $\vdash f1 \frown g \longrightarrow f1 \frown g1$ **by** (rule *RightSChopImpSChop*)

from 2 4 **show** ?thesis **by** fastforce

qed

lemma *SChopEqvSChop*:

assumes $\vdash f = f1$

$\vdash g = g1$

shows $\vdash f \frown g = f1 \frown g1$

proof –

have 1: $\vdash f = f1$ **using** *assms* **by** auto

hence 2: $\vdash f \frown g = f1 \frown g$ **by** (rule *LeftSChopEqvSChop*)

have 3: $\vdash g = g1$ **using** *assms* **by** auto

hence 4: $\vdash f1 \frown g = f1 \frown g1$ **by** (rule *RightSChopEqvSChop*)

from 2 4 **show** ?thesis **by** fastforce

qed

lemma *BoxSChopImpSChopBox*:

$\vdash \Box h \longrightarrow f \frown g \longrightarrow f \frown (\Box h \wedge g)$

proof –

have 1: $\vdash \Box h \longrightarrow \Box(g \longrightarrow \Box h \wedge g)$ **by** (rule *BoxImpBoxImpBox*)

have 2: $\vdash \Box(g \longrightarrow \Box h \wedge g) \longrightarrow f \frown g \longrightarrow f \frown (\Box h \wedge g)$ **by** (rule *BoxSChopImpSChop*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *NotChopEqvSYieldsNot*:

$\vdash (\neg (f \frown g)) = f \text{ syields } (\neg g)$

proof –

have 1: $\vdash g = (\neg \neg g)$ **by** *auto*

hence 2: $\vdash f \frown g = f \frown (\neg \neg g)$ **by** (*rule RightSCHopEqvSCHop*)

hence 3: $\vdash (\neg (f \frown g)) = (\neg (f \frown (\neg \neg g)))$ **by** *auto*

from 3 **show** *?thesis* **by** (*simp add: syields-d-def*)

qed

lemma *NotDfFalse*:

$\vdash \neg (df \#False)$

proof –

have 1: $\vdash (init \#True) \longrightarrow bf (init \#True)$ **by** (*rule StateImpBf*)

hence 2: $\vdash \#True \longrightarrow bf \#True$ **by** (*simp add: BfGen*)

have 3: $\vdash \#True$ **by** *auto*

have 4: $\vdash bf \#True$ **using** 2 3 *MP* **by** *auto*

hence 5: $\vdash \neg (df (\neg \#True))$ **by** (*simp add: bf-d-def*)

have 6: $\vdash (\neg \#True) = \#False$ **by** *auto*

hence 7: $\vdash df (\neg \#True) = df \#False$ **by** (*rule DfEqvDf*)

from 5 7 **show** *?thesis* **by** *auto*

qed

lemma *StateAndEmptySCHop*:

$\vdash ((init\ w) \wedge empty) \frown f = ((init\ w) \wedge f)$

proof –

have 1: $\vdash ((init\ w) \wedge empty) \frown f = ((init\ w) \wedge empty \frown f)$ **by** (*rule StateAndSCHop*)

have 2: $\vdash empty \frown f = f$ **by** (*rule EmptySCHop*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *StateAndNextSCHop*:

$\vdash ((init\ w) \wedge \bigcirc f) \frown g = ((init\ w) \wedge \bigcirc(f \frown g))$

proof –

have 1: $\vdash ((init\ w) \wedge \bigcirc f) \frown g = ((init\ w) \wedge (\bigcirc f) \frown g)$ **by** (*rule StateAndSCHop*)

have 2: $\vdash (\bigcirc f) \frown g = \bigcirc(f \frown g)$ **by** (*rule NextSCHop*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *NextStateAndSCHop*:

$\vdash \bigcirc(((init\ w) \wedge f) \frown g) = (\bigcirc (init\ w) \wedge \bigcirc(f \frown g))$

proof –

have 1: $\vdash ((init\ w) \wedge f) \frown g = ((init\ w) \wedge f \frown g)$ **by** (*rule StateAndSCHop*)

hence 2: $\vdash \bigcirc(((init\ w) \wedge f) \frown g) = \bigcirc((init\ w) \wedge f \frown g)$ **by** (*rule NextEqvNext*)

have 3: $\vdash \bigcirc((init\ w) \wedge f \frown g) = (\bigcirc (init\ w) \wedge \bigcirc(f \frown g))$ **by** (*rule NextAndEqvNextAndNext*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *StateSYieldsEqv*:

$\vdash ((init\ w) \longrightarrow (f \text{ syields } g)) = ((init\ w) \wedge f) \text{ syields } g$

proof –
have 1: $\vdash ((init\ w) \wedge f) \frown (\neg\ g) = ((init\ w) \wedge f \frown (\neg\ g))$ **by** (rule *StateAndSChop*)
hence 2: $\vdash ((init\ w) \longrightarrow \neg (f \frown (\neg\ g))) = (\neg (((init\ w) \wedge f) \frown (\neg\ g)))$ **by** *auto*
from 2 **show** ?thesis **by** (simp add: syields-d-def)
qed

lemma *StateAndDf*:

$\vdash ((init\ w) \wedge df\ f) = df\ ((init\ w) \wedge f)$

proof –
have 1: $\vdash ((init\ w) \wedge f) \frown \#True = ((init\ w) \wedge f \frown \#True)$ **by** (rule *StateAndSChop*)
from 1 **show** ?thesis **by** (metis df-d-def integ-reflection)
qed

lemma *DfNext*:

$\vdash df(\bigcirc f) = \bigcirc(df\ f)$

proof –
have 1: $\vdash (\bigcirc f) \frown \#True = \bigcirc(f \frown \#True)$ **by** (rule *NextSChop*)
from 1 **show** ?thesis **by** (simp add: df-d-def)
qed

lemma *DfNextState*:

$\vdash df(\bigcirc (init\ w)) = \bigcirc (init\ w)$

proof –
have 1: $\vdash df(\bigcirc (init\ w)) = \bigcirc(df\ (init\ w))$ **by** (rule *DfNext*)
have 2: $\vdash df\ (init\ w) = (init\ w)$ **by** (rule *DfState*)
hence 3: $\vdash \bigcirc(df\ (init\ w)) = \bigcirc (init\ w)$ **by** (rule *NextEqvNext*)
from 1 3 **show** ?thesis **by** fastforce
qed

lemma *DfStateAndNextStateEqvStateAndNextState*:

$\vdash df(init\ w \wedge \bigcirc (init\ w1)) = (init\ w \wedge \bigcirc (init\ w1))$

proof –
have 1: $\vdash (init\ w \wedge \bigcirc (init\ w1)) \frown \#True = (init\ w \wedge \bigcirc ((init\ w1) \frown \#True))$
using *StateAndNextSChop* **by** blast
have 2: $\vdash df(init\ w \wedge \bigcirc (init\ w1)) = (init\ w \wedge \bigcirc ((init\ w1) \frown \#True))$
using 1 **by** (simp add: df-d-def)
have 3: $\vdash df(init\ w1) = init\ w1$
by (simp add: *DfState*)
have 4: $\vdash skip \frown df(init\ w1) = skip \frown (init\ w1)$
by (simp add: 3 *RightSChopEqvSChop*)
have 5: $\vdash \bigcirc(df\ (init\ w1)) = \bigcirc (init\ w1)$
by (simp add: 3 *NextEqvNext*)
from 2 5 **show** ?thesis **by** (metis df-d-def int-eq)
qed

lemma *StateImpBfGen*:

assumes $\vdash (init\ w) \longrightarrow f$

shows $\vdash (init\ w) \longrightarrow bf\ f$

proof –

have 1: $\vdash (\text{init } w) \longrightarrow f$ **using** *assms* **by** *auto*
hence 2: $\vdash \neg f \longrightarrow \neg (\text{init } w)$ **by** *auto*
hence 3: $\vdash df (\neg f) \longrightarrow df (\neg (\text{init } w))$ **by** (*rule DfImpDf*)
hence 4: $\vdash df (\neg f) \longrightarrow df (\text{init } (\neg w))$ **by** (*metis Initprop(2) inteq-reflection*)
have 5: $\vdash df (\text{init } (\neg w)) = (\text{init } (\neg w))$ **by** (*rule DfState*)
have 6: $\vdash df (\neg f) \longrightarrow \neg (\text{init } w)$ **using** 4 5 **using** *Initprop(2)* **by** *fastforce*
hence 7: $\vdash (\text{init } w) \longrightarrow \neg (df (\neg f))$ **by** *auto*
from 7 **show** *?thesis* **by** (*simp add: bf-d-def*)
qed

lemma *SChopAndNotSChopImp*:

$\vdash f \frown g \wedge \neg (f \frown g1) \longrightarrow f \frown (g \wedge \neg g1)$

proof –

have 1: $\vdash g \longrightarrow (g \wedge \neg g1) \vee g1$ **by** *auto*
hence 2: $\vdash f \frown g \longrightarrow f \frown ((g \wedge \neg g1) \vee g1)$ **by** (*rule RightSChopImpSChop*)
have 3: $\vdash f \frown ((g \wedge \neg g1) \vee g1) \longrightarrow (f \frown (g \wedge \neg g1)) \vee (f \frown g1)$ **by** (*rule SChopOrImp*)
have 4: $\vdash f \frown g \longrightarrow f \frown (g \wedge \neg g1) \vee f \frown g1$ **using** 2 3 *MP* **by** *fastforce*
from 4 **show** *?thesis* **by** *auto*
qed

lemma *SChopAndSYieldsImp*:

$\vdash f \frown g \wedge f \text{ syields } g1 \longrightarrow f \frown (g \wedge g1)$

proof –

have 1: $\vdash g \longrightarrow (g \wedge g1) \vee \neg g1$ **by** *auto*
hence 2: $\vdash f \frown g \longrightarrow f \frown ((g \wedge g1) \vee \neg g1)$ **by** (*rule RightSChopImpSChop*)
have 3: $\vdash f \frown ((g \wedge g1) \vee \neg g1) \longrightarrow (f \frown (g \wedge g1)) \vee (f \frown (\neg g1))$ **by** (*rule SChopOrImp*)
have 4: $\vdash f \frown g \longrightarrow f \frown (g \wedge g1) \vee f \frown (\neg g1)$ **using** 2 3 *MP* **by** *fastforce*
hence 5: $\vdash f \frown g \wedge \neg (f \frown (\neg g1)) \longrightarrow f \frown (g \wedge g1)$ **by** *auto*
from 5 **show** *?thesis* **by** (*simp add: syields-d-def*)
qed

lemma *SChopAndSYieldsMP*:

$\vdash f \frown g \wedge f \text{ syields } (g \longrightarrow g1) \longrightarrow f \frown g1$

proof –

have 1: $\vdash f \frown g \wedge f \text{ syields } (g \longrightarrow g1) \longrightarrow f \frown (g \wedge (g \longrightarrow g1))$ **by** (*rule SChopAndSYieldsImp*)
have 2: $\vdash g \wedge (g \longrightarrow g1) \longrightarrow g1$ **by** *auto*
hence 3: $\vdash f \frown (g \wedge (g \longrightarrow g1)) \longrightarrow f \frown g1$ **by** (*rule RightSChopImpSChop*)
from 1 3 **show** *?thesis* **by** *fastforce*
qed

lemma *OrSYieldsImp*:

$\vdash (f \vee f1) \text{ syields } g = ((f \text{ syields } g) \wedge (f1 \text{ syields } g))$

proof –

have 1: $\vdash ((f \vee f1) \frown (\neg g)) = ((f \frown (\neg g)) \vee (f1 \frown (\neg g)))$ **by** (*rule OrSChopEqv*)
hence 2: $\vdash \neg ((f \vee f1) \frown (\neg g)) = \neg (f \frown (\neg g)) \wedge \neg (f1 \frown (\neg g))$ **by** *auto*
from 2 **show** *?thesis* **by** (*simp add: syields-d-def*)
qed

lemma *LeftSYieldsImpSYields*:

assumes $\vdash f \longrightarrow f1$

shows $\vdash (f1 \text{ syields } g) \longrightarrow (f \text{ syields } g)$
proof –
have 1: $\vdash f \longrightarrow f1$ **using** *assms* **by** *auto*
hence 2: $\vdash f \frown (\neg g) \longrightarrow f1 \frown (\neg g)$ **by** (*rule LeftSChopImpSChop*)
hence 3: $\vdash \neg (f1 \frown (\neg g)) \longrightarrow \neg (f \frown (\neg g))$ **by** *auto*
from 3 **show** *?thesis* **by** (*simp add: syields-d-def*)
qed

lemma *LeftSYieldsEqvSYields*:
assumes $\vdash f = f1$
shows $\vdash (f \text{ syields } g) = (f1 \text{ syields } g)$
proof –
have 1: $\vdash f = f1$ **using** *assms* **by** *auto*
hence 2: $\vdash f \frown (\neg g) = f1 \frown (\neg g)$ **by** (*rule LeftSChopEqvSChop*)
hence 3: $\vdash \neg (f \frown (\neg g)) = \neg (f1 \frown (\neg g))$ **by** *auto*
from 3 **show** *?thesis* **by** (*simp add: syields-d-def*)
qed

2.4.6 Properties of SFin

lemma *SFinEqvTrueSChopAndEmpty*:
 $\vdash \text{sfin } f = \# \text{True} \frown (f \wedge \text{empty})$
proof –
have 01: $\vdash \text{sfin } f = (\neg \text{fin } (\neg f))$
by (*simp add: sfin-d-def*)
have 02: $\vdash (\neg \text{fin } (\neg f)) = (\neg (\Box (\text{empty} \longrightarrow \neg f)))$
by (*simp add: fin-d-def*)
have 03: $\vdash (\neg (\Box (\text{empty} \longrightarrow \neg f))) = \Diamond (\neg (\text{empty} \longrightarrow \neg f))$
by (*simp add: always-d-def*)
have 04: $\vdash \neg (\text{empty} \longrightarrow \neg f) = (\text{empty} \wedge f)$
by *auto*
have 05: $\vdash \Diamond (\neg (\text{empty} \longrightarrow \neg f)) = \Diamond (\text{empty} \wedge f)$
using 04 *inteq-reflection* **by** *fastforce*
from 01 02 03 05 **show** *?thesis*
by (*metis SChopAndCommute TrueSChopEqvDiamond inteq-reflection*)
qed

lemma *DiamondSFin*:
 $\vdash \Diamond (\text{sfin } w) = \text{sfin } w$
by (*metis (no-types, lifting) ChopAssoc FiniteChopFiniteEqvFinite FiniteOr FiniteOrInfinite InfEqvNotFinite OrFiniteInf SFinEqvTrueSChopAndEmpty finite-d-def int-eq-true int-simps(21) inteq-reflection schop-d-def sometimes-d-def*)

lemma *SChopSFinExportA*:
 $\vdash f \frown (g \wedge \text{sfin } w) \longrightarrow \text{sfin } w$
using *DiamondSFin*
by (*metis SChopAndB SChopImpDiamond inteq-reflection lift-imp-trans*)

lemma *SFinImpBox*:
 $\vdash \text{sfin } w \longrightarrow \Box (\text{sfin } w)$

by (metis (mono-tags, lifting) DiamondFin always-d-def intI int-eq int-simps(4) sfin-d-def unl-lift2)

lemma *SFinAndSChopImport*:

$\vdash (sfin\ w) \wedge (f \frown g) \longrightarrow f \frown ((sfin\ w) \wedge g)$

proof –

have 1: $\vdash sfin\ w \longrightarrow \Box(sfin\ w)$ **by** (rule *SFinImpBox*)

hence 2: $\vdash sfin\ w \wedge (f \frown g) \longrightarrow \Box(sfin\ w) \wedge (f \frown g)$ **by** *auto*

have 3: $\vdash \Box(sfin\ w) \wedge (f \frown g) \longrightarrow f \frown ((sfin\ w) \wedge g)$ **using** *BoxAndSChopImport* **by** *blast*
from 2 3 **show** ?thesis **using** *MP* **by** *fastforce*

qed

lemma *SFinAndSChop*:

$\vdash (f \frown (g \wedge sfin\ w)) = (sfin\ w \wedge f \frown g)$

using *SFinAndSChopImport* *SChopSFinExportA* *SChopAndA* *SChopAndCommutate*

by *fastforce*

lemma *SChopAndEmptyEqvEmptySChopEmpty*:

$\vdash ((f \frown g) \wedge empty) = (f \wedge empty) \frown (g \wedge empty)$

by (*auto simp: itl-defs zero-enat-def*)

lemma *SFinAndEmpty*:

$\vdash ((sfin\ w) \wedge empty) = (w \wedge empty)$

proof –

have 1: $\vdash ((sfin\ w) \wedge empty) = (\#True \frown (w \wedge empty) \wedge empty)$

using *SFinEqvTrueSChopAndEmpty* **by** *fastforce*

have 2: $\vdash (\#True \frown (w \wedge empty) \wedge empty) = ((\#True \wedge empty) \frown (w \wedge empty))$

by (*simp add: FiniteChopAndEmptyEqvChopAndEmpty schop-d-def*)

have 3: $\vdash (\#True \wedge empty) \frown (w \wedge empty) = (empty \frown (w \wedge empty))$

using *LeftSChopEqvSChop* **by** *fastforce*

have 4: $\vdash (empty \frown (w \wedge empty)) = (w \wedge empty)$

using *EmptySChop* **by** *blast*

from 1 2 3 4 **show** ?thesis **by** *fastforce*

qed

lemma *AndSFinEqvSChopAndEmpty*:

$\vdash ((f \wedge finite) \wedge sfin\ g) = f \frown (g \wedge empty)$

proof –

have 1: $\vdash ((f \wedge finite) \wedge sfin\ g) = (f \frown empty \wedge sfin\ g)$

using *SChopEmpty*

by (metis (no-types, lifting) DiamondEmptyEqvFinite FiniteImpAnd Prop10 SChopImpDiamond
inteq-reflection lift-and-com)

have 2: $\vdash (sfin\ g \wedge f \frown empty) = (f \frown (empty \wedge sfin\ g))$

using *SFinAndSChop* **by** *fastforce*

have 3: $\vdash (empty \wedge sfin\ g) = (sfin\ g \wedge empty)$

by *auto*

have 4: $\vdash (sfin\ g \wedge empty) = (g \wedge empty)$

using *SFinAndEmpty* **by** *metis*

have 5: $\vdash (empty \wedge sfin\ g) = (g \wedge empty)$

using 3 4 **by** *auto*

hence 6: $\vdash f \frown (empty \wedge sfin\ g) = f \frown (g \wedge empty)$

using *RightSChopEqvSChop* by *blast*
 from 1 2 5 show ?thesis by (metis *inteq-reflection lift-and-com*)
 qed

lemma *AndSFinEqvSChopStateAndEmpty*:
 $\vdash ((f \wedge \text{finite}) \wedge \text{sfin } (\text{init } w)) = f \frown ((\text{init } w) \wedge \text{empty})$
 using *AndSFinEqvSChopAndEmpty* by *blast*

lemma *DiamondEqvEmptyOrNextDiamond*:
 $\vdash \Diamond f = (f \vee \bigcirc(\Diamond f))$
proof –
 have 1: $\vdash \Box (\neg f) = ((\neg f) \wedge \text{wnext}(\Box (\neg f)))$
 by (simp add: *BoxEqvAndWnextBox*)
 have 2: $\vdash (\neg \Diamond f) = ((\neg f) \wedge \text{wnext}(\Box (\neg f)))$
 using 1 by (simp add: *always-d-def*)
 have 3: $\vdash \Diamond f = (f \vee \neg(\text{wnext}(\Box (\neg f))))$
 using 2 by *auto*
 have 4: $\vdash (\neg(\text{wnext}(\Box (\neg f)))) = \bigcirc(\neg\Box(\neg f))$
 by (simp add: *wnext-d-def*)
 have 5: $\vdash \neg\Box(\neg f) = \Diamond f$
 by (simp add: *always-d-def*)
 have 6: $\vdash \bigcirc(\neg\Box(\neg f)) = \bigcirc(\Diamond f)$
 using 5 using *inteq-reflection* by *force*
 from 3 4 6 show ?thesis by *fastforce*
 qed

lemma *SFinStateEqvStateAndEmptyOrNextSFinState*:
 $\vdash \text{sfin } (\text{init } w) = (((\text{init } w) \wedge \text{empty}) \vee \bigcirc(\text{sfin } (\text{init } w)))$
proof –
 have 01: $\vdash \text{sfin } (\text{init } w) = \# \text{True} \frown ((\text{init } w) \wedge \text{empty})$
 by (simp add: *SFinEqvTrueSChopAndEmpty*)
 have 02: $\vdash \# \text{True} \frown ((\text{init } w) \wedge \text{empty}) = \Diamond((\text{init } w) \wedge \text{empty})$
 by (simp add: *TrueSChopEqvDiamond*)
 have 03: $\vdash \Diamond((\text{init } w) \wedge \text{empty}) = (((\text{init } w) \wedge \text{empty}) \vee \bigcirc(\text{sfin } (\text{init } w)))$
 using *DiamondEqvEmptyOrNextDiamond* 02 01 by (metis *inteq-reflection*)
 from 01 02 03 show ?thesis by *fastforce*
 qed

lemma *SFinSChopEqvOr*:
 $\vdash (\text{sfin } (\text{init } w)) \frown f = (((\text{init } w) \wedge f) \vee \bigcirc((\text{sfin } (\text{init } w)) \frown f))$
proof –
 have 1: $\vdash \text{sfin } (\text{init } w) = (((\text{init } w) \wedge \text{empty}) \vee \bigcirc(\text{sfin } (\text{init } w)))$
 by (rule *SFinStateEqvStateAndEmptyOrNextSFinState*)
 hence 2: $\vdash (\text{sfin } (\text{init } w)) \frown f = (((\text{init } w) \wedge \text{empty}) \vee \bigcirc(\text{sfin } (\text{init } w))) \frown f$
 by (rule *LeftSChopEqvSChop*)
 have 3: $\vdash (((\text{init } w) \wedge \text{empty}) \vee \bigcirc(\text{sfin } (\text{init } w))) \frown f$
 $= (((\text{init } w) \wedge \text{empty}) \frown f \vee (\bigcirc(\text{sfin } (\text{init } w))) \frown f)$
 by (rule *OrSChopEqv*)
 have 4: $\vdash ((\text{init } w) \wedge \text{empty}) \frown f = ((\text{init } w) \wedge f)$
 by (rule *StateAndEmptySChop*)

have 5: $\vdash (\bigcirc (\text{sfin } (\text{init } w))) \frown f = \bigcirc ((\text{sfin } (\text{init } w)) \frown f)$
by (rule NextSChop)
from 2 3 4 5 **show** ?thesis **by** fastforce
qed

lemma SFinSChopEqvDiamond:

$\vdash (\text{sfin } (\text{init } w)) \frown f = \Diamond ((\text{init } w) \wedge f)$

proof –

have 1: $\vdash (\text{sfin } (\text{init } w)) = (\# \text{True} \frown ((\text{init } w) \wedge \text{empty}))$
by (simp add: SFinEqvTrueSChopAndEmpty)
hence 2: $\vdash (\text{sfin } (\text{init } w)) \frown f = (\# \text{True} \frown ((\text{init } w) \wedge \text{empty})) \frown f$
by (rule LeftSChopEqvSChop)
have 3: $\vdash \# \text{True} \frown ((\text{init } w) \wedge \text{empty}) \frown f = (\# \text{True} \frown ((\text{init } w) \wedge \text{empty})) \frown f$
by (rule SChopAssoc)
have 4: $\vdash \# \text{True} \frown ((\text{init } w) \wedge \text{empty}) \frown f = \Diamond ((\text{init } w) \wedge \text{empty}) \frown f$
using TrueSChopEqvDiamond **by** blast
have 5: $\vdash ((\text{init } w) \wedge \text{empty}) \frown f = ((\text{init } w) \wedge f)$
using StateAndEmptySChop **by** blast
hence 6: $\vdash \Diamond ((\text{init } w) \wedge \text{empty}) \frown f = \Diamond ((\text{init } w) \wedge f)$
by (rule DiamondEqvDiamond)
from 2 3 4 6 **show** ?thesis **by** fastforce
qed

lemma SFinSYields:

$\vdash (\text{sfin } (\text{init } w)) \text{ syields } (\text{init } w)$

proof –

have 1: $\vdash (\text{sfin } (\text{init } w)) \frown (\neg (\text{init } w)) = \Diamond ((\text{init } w) \wedge \neg (\text{init } w))$
by (rule SFinSChopEqvDiamond)
have 2: $\vdash \neg (\Diamond ((\text{init } w) \wedge \neg (\text{init } w)))$
by (rule NotDiamondAndNot)
have 3: $\vdash \neg ((\text{sfin } (\text{init } w)) \frown (\neg (\text{init } w)))$
using 1 2 **by** fastforce
from 3 **show** ?thesis **by** (simp add: syields-d-def)
qed

lemma SFinEqvFinAndFinite:

$\vdash (\text{finite} \wedge \text{fin } f) = \text{sfin } f$

by (metis AndFinEqvChopAndEmpty DiamondSChopdef SFinEqvTrueSChopAndEmpty int-simps(20)
 inteq-reflection sometimes-d-def)

lemma AndFiniteImpAndSFinStateOrSFinNotState:

$\vdash f \wedge \text{finite} \longrightarrow (f \wedge \text{sfin } (\text{init } w)) \vee (f \wedge \text{sfin } (\neg (\text{init } w)))$

proof –

have 1: $\vdash (\neg \text{fin } (\text{init } (\neg w)) \wedge \text{finite}) = (\text{fin } (\text{init } (\neg \neg w)) \wedge \text{finite})$
using FinNotStateEqvNotFinState **by** blast
have 2: $\vdash (\text{fin } (\text{init } (\neg \neg w)) \wedge \text{finite}) = (\text{fin } (\text{init } (w)) \wedge \text{finite})$
by simp
have 3: $\vdash f \wedge \text{finite} \longrightarrow (f \wedge \text{finite}) \wedge \text{fin } (\text{init } w) \vee (f \wedge \text{finite}) \wedge \text{fin } (\neg \text{init } w)$
by (simp add: ImpAndFinStateOrFinNotState)
have 4: $\vdash (\text{finite} \wedge \text{fin } (\text{init } w)) = \text{sfin } (\text{init } w)$

using *SFinEqvFinAndFinite*[of *LIFT*(*init w*)] by *fastforce*
 have 5: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)) = (f \wedge \text{sfin } (\text{init } w))$
 using 4 by *auto*
 have 6: $\vdash (\text{finite} \wedge \text{fin } (\neg(\text{init } w))) = \text{sfin}(\neg(\text{init } w))$
 using *SFinEqvFinAndFinite*[of *LIFT*($\neg(\text{init } w)$)] by *fastforce*
 have 7: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\neg(\text{init } w))) = (f \wedge \text{sfin } (\neg(\text{init } w)))$
 using 6 by *auto*
 show ?thesis
 using 3 5 7 by *fastforce*
 qed

lemma *AndSFinSChopEqvStateAndSChop*:

$\vdash (f \wedge \text{sfin } (\text{init } w)) \frown g = f \frown ((\text{init } w) \wedge g)$

proof –

have 1: $\vdash (\text{sfin } (\text{init } w)) \text{ syields } (\text{init } w)$
 by (rule *SFinSYields*)
 have 2: $\vdash f \wedge \text{sfin } (\text{init } w) \longrightarrow \text{sfin } (\text{init } w)$
 by *auto*
 hence 3: $\vdash (\text{sfin } (\text{init } w)) \text{ syields } (\text{init } w) \longrightarrow (f \wedge \text{sfin } (\text{init } w)) \text{ syields } (\text{init } w)$
 using *LeftSYieldsImpSYields* by *metis*
 have 4: $\vdash (f \wedge \text{sfin } (\text{init } w)) \text{ syields } (\text{init } w)$
 using 1 3 *MP* by *fastforce*
 have 5: $\vdash (f \wedge \text{sfin } (\text{init } w)) \frown g \wedge (f \wedge \text{sfin } (\text{init } w)) \text{ syields } (\text{init } w) \longrightarrow (f \wedge \text{sfin } (\text{init } w)) \frown (g \wedge (\text{init } w))$
 by (rule *SChopAndSYieldsImp*)
 have 6: $\vdash (f \wedge \text{sfin } (\text{init } w)) \frown g \longrightarrow (f \wedge \text{sfin } (\text{init } w)) \frown (g \wedge (\text{init } w))$
 using 4 5 by *fastforce*
 have 7: $\vdash (f \wedge \text{sfin } (\text{init } w)) \frown (g \wedge (\text{init } w)) \longrightarrow f \frown (g \wedge (\text{init } w))$
 by (rule *AndSChopA*)
 have 8: $\vdash g \wedge (\text{init } w) \longrightarrow (\text{init } w) \wedge g$
 by *auto*
 hence 9: $\vdash f \frown (g \wedge (\text{init } w)) \longrightarrow f \frown ((\text{init } w) \wedge g)$
 by (rule *RightSChopImpSChop*)
 have 10: $\vdash (f \wedge \text{sfin } (\text{init } w)) \frown g \longrightarrow f \frown ((\text{init } w) \wedge g)$
 using 6 7 9 by *fastforce*
 have 11: $\vdash (f \wedge \text{finite}) \longrightarrow (f \wedge \text{sfin } (\text{init } w)) \vee (f \wedge \text{sfin } (\neg(\text{init } w)))$
 using *AndFiniteImpAndSFinStateOrSFinNotState* by *blast*
 hence 12: $\vdash f \frown ((\text{init } w) \wedge g) \longrightarrow ((f \wedge \text{sfin } (\text{init } w)) \vee (f \wedge \text{sfin } (\neg(\text{init } w)))) \frown ((\text{init } w) \wedge g)$
 by (metis *FiniteImp LeftChopImpChop inteq-reflection schop-d-def*)
 have 13: $\vdash ((f \wedge \text{sfin } (\text{init } w)) \vee (f \wedge \text{sfin } (\neg(\text{init } w)))) \frown ((\text{init } w) \wedge g) = ((f \wedge \text{sfin } (\text{init } w)) \frown ((\text{init } w) \wedge g)) \vee (f \wedge \text{sfin } (\neg(\text{init } w))) \frown ((\text{init } w) \wedge g)$
 by (rule *OrSChopEqv*)
 have 14: $\vdash (f \wedge \text{sfin } (\text{init } (\neg w))) \frown ((\text{init } w) \wedge g) \longrightarrow \Diamond((\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g))$
 using *SFinSChopEqvDiamond*
 by (metis *SChopImpSChop Prop12 int-iffD1 inteq-reflection lift-and-com*)
 have 141: $\vdash \neg(\Diamond((\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g))) \longrightarrow \neg((f \wedge \text{sfin } (\text{init } (\neg w))) \frown ((\text{init } w) \wedge g))$

using 14 by fastforce
 have 150: $\vdash ((init (\neg w)) \wedge ((init w) \wedge g)) = \#False$
 using Initprop(2) by fastforce
 have 15: $\vdash \neg(\Diamond((init (\neg w)) \wedge ((init w) \wedge g)))$
 by (metis 150 NotDiamondAndNot int-eq int-simps(21))
 have 151: $\vdash \neg((f \wedge sfin (init (\neg w))) \frown ((init w) \wedge g))$
 using 15 141 by fastforce
 have 1511: $\vdash (f \wedge sfin (\neg (init w))) \frown ((init w) \wedge g) \longrightarrow \#False$
 using 151 by (metis Initprop(2) int-simps(14) inteq-reflection)
 have 152: $\vdash (f \wedge sfin (init w)) \frown ((init w) \wedge g) \vee (f \wedge sfin (\neg (init w))) \frown ((init w) \wedge g) \longrightarrow$
 $(f \wedge sfin (init w)) \frown ((init w) \wedge g)$
 using 1511 by fastforce
 have 16: $\vdash f \frown ((init w) \wedge g) \longrightarrow (f \wedge sfin (init w)) \frown ((init w) \wedge g)$
 using 12 13 152 by fastforce
 have 17: $\vdash (f \wedge sfin (init w)) \frown ((init w) \wedge g) \longrightarrow (f \wedge sfin (init w)) \frown g$
 by (rule SChopAndB)
 have 18: $\vdash f \frown ((init w) \wedge g) \longrightarrow (f \wedge sfin (init w)) \frown g$
 using 16 17 by fastforce
 from 10 18 show ?thesis by fastforce
 qed

lemma DfAndSFinEqvSChopState:

$\vdash df (f \wedge sfin (init w)) = f \frown (init w)$

proof –

have 1: $\vdash (f \wedge sfin (init w)) \frown \#True = f \frown ((init w) \wedge \#True)$

by (rule AndSFinSChopEqvStateAndSChop)

have 2: $\vdash ((init w) \wedge \#True) = (init w)$

by auto

hence 3: $\vdash (f \frown ((init w) \wedge \#True)) = (f \frown (init w))$

by (rule RightSChopEqvSChop)

have 4: $\vdash (f \wedge sfin (init w)) \frown \#True = f \frown (init w)$

using 1 3 by auto

from 4 show ?thesis by (simp add: df-d-def)

qed

lemma SFinNotStateEqvNotSFinState:

$\vdash finite \longrightarrow (\neg(sfin (init w))) = (sfin (init (\neg w)))$

using SFinEqvTrueSChopAndEmpty

by (metis Initprop(2) SFinprop(3) int-eq)

lemma BfImpSFinEqvSYieldsState:

$\vdash bf (f \longrightarrow sfin (init w)) = f \ syields (init w)$

proof –

have 1: $\vdash df (f \wedge sfin (init (\neg w))) = f \frown (init (\neg w))$

by (rule DfAndSFinEqvSChopState)

have 2: $\vdash finite \longrightarrow (f \wedge sfin (init (\neg w))) = (f \wedge \neg(sfin (init w)))$

using SFinNotStateEqvNotSFinState by fastforce

have 3: $\vdash (f \wedge \neg(sfin (init w))) = (\neg(f \longrightarrow sfin (init w)))$

by auto

have 4: $\vdash finite \longrightarrow (f \wedge sfin (init (\neg w))) = (\neg(f \longrightarrow sfin (init w)))$

using 2 3 by fastforce
 hence 5: $\vdash df (f \wedge sfin (init (\neg w))) = df (\neg (f \longrightarrow sfin(init w)))$
 by (metis DfEqvNotBfNot FiniteImpAnd df-d-def inteq-reflection schop-d-def)
 have 6: $\vdash df (\neg (f \longrightarrow sfin (init w))) = (\neg (bf (f \longrightarrow sfin(init w))))$
 by (rule DfNotEqvNotBf)
 have 7: $\vdash \neg (bf (f \longrightarrow sfin (init w))) = f \frown (init (\neg w))$
 using 1 5 6 Initprop by fastforce
 hence 8: $\vdash bf (f \longrightarrow sfin (init w)) = (\neg (f \frown (\neg (init w))))$
 by (metis Initprop(2) int-eq int-simps(7))
 from 8 show ?thesis by (simp add: syields-d-def)
 qed

lemma StateImpSYields:

assumes $\vdash (init w) \wedge f \longrightarrow sfin (init w1)$
 shows $\vdash (init w) \longrightarrow (f syields (init w1))$
 proof –
 have 1: $\vdash (init w) \wedge f \longrightarrow sfin (init w1)$ using assms by auto
 hence 2: $\vdash (init w) \longrightarrow (f \longrightarrow sfin (init w1))$ by auto
 hence 3: $\vdash (init w) \longrightarrow bf (f \longrightarrow sfin (init w1))$ using StateImpBfGen by auto
 have 4: $\vdash bf (f \longrightarrow sfin (init w1)) = f syields (init w1)$ by (rule BfImpSFinEqvSYieldsState)
 from 3 4 show ?thesis by fastforce
 qed

lemma StateAndSYieldsImpSYields:

assumes $\vdash (init w) \wedge f \longrightarrow f1$
 shows $\vdash (init w) \wedge (f1 syields g) \longrightarrow (f syields g)$
 proof –
 have 1: $\vdash (init w) \wedge f \longrightarrow f1$ using assms by auto
 hence 2: $\vdash (init w) \wedge (f \frown (\neg g)) \longrightarrow f1 \frown (\neg g)$ by (rule StateAndSChopImpSChopRule)
 hence 3: $\vdash (init w) \wedge \neg (f1 \frown (\neg g)) \longrightarrow \neg (f \frown (\neg g))$ by auto
 from 3 show ?thesis by (simp add: syields-d-def)
 qed

lemma AndSYieldsA:

$\vdash f syields g \longrightarrow (f \wedge f1) syields g$
 proof –
 have 1: $\vdash f \wedge f1 \longrightarrow f$ by auto
 from 1 show ?thesis by (rule LeftSYieldsImpSYields)
 qed

lemma AndSYieldsB:

$\vdash f1 syields g \longrightarrow (f \wedge f1) syields g$
 proof –
 have 1: $\vdash f \wedge f1 \longrightarrow f1$ by auto
 from 1 show ?thesis by (rule LeftSYieldsImpSYields)
 qed

lemma RightSYieldsImpSYields:

assumes $\vdash g \longrightarrow g1$
 shows $\vdash (f syields g) \longrightarrow (f syields g1)$

proof –
have 1: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*
hence 2: $\vdash \neg g1 \longrightarrow \neg g$ **by** *auto*
hence 3: $\vdash f \frown (\neg g1) \longrightarrow f \frown (\neg g)$ **by** (*rule RightSChopImpSChop*)
hence 4: $\vdash \neg (f \frown (\neg g)) \longrightarrow \neg (f \frown (\neg g1))$ **by** *auto*
from 4 **show** *?thesis* **by** (*simp add: syields-d-def*)
qed

lemma *RightSYieldsEqvSYields*:

assumes $\vdash g = g1$
shows $\vdash (f \text{ syields } g) = (f \text{ syields } g1)$

proof –
have 1: $\vdash g = g1$ **using** *assms* **by** *auto*
hence 2: $\vdash (\neg g) = (\neg g1)$ **by** *auto*
hence 3: $\vdash f \frown (\neg g) = f \frown (\neg g1)$ **by** (*rule RightSChopEqvSChop*)
hence 4: $\vdash (\neg (f \frown (\neg g))) = (\neg (f \frown (\neg g1)))$ **by** *auto*
from 4 **show** *?thesis* **by** (*simp add: syields-d-def*)
qed

lemma *BoxImpSYields*:

$\vdash \Box g \longrightarrow f \text{ syields } g$

proof –
have 1: $\vdash f \frown (\neg g) \longrightarrow \Diamond(\neg g)$ **by** (*rule SChopImpDiamond*)
hence 2: $\vdash \neg (\Diamond(\neg g)) \longrightarrow \neg (f \frown (\neg g))$ **by** *auto*
from 2 **show** *?thesis* **by** (*simp add: syields-d-def always-d-def*)
qed

lemma *BoxEqvTrueSYields*:

$\vdash \Box f = \# \text{True syields } f$

proof –
have 1: $\vdash \# \text{True} \frown (\neg f) = \Diamond(\neg f)$ **by** (*rule TrueSChopEqvDiamond*)
hence 2: $\vdash (\neg (\# \text{True} \frown (\neg f))) = (\neg (\Diamond(\neg f)))$ **by** *auto*
have 3: $\vdash \Box f = (\neg (\Diamond(\neg f)))$ **by** (*simp add: always-d-def*)
have 4: $\vdash \Box f = (\neg (\# \text{True} \frown (\neg f)))$ **using** 2 3 **by** *fastforce*
from 4 **show** *?thesis* **by** (*simp add: syields-d-def*)
qed

lemma *SYieldsGen*:

assumes $\vdash g$
shows $\vdash f \text{ syields } g$

proof –
have 1: $\vdash g$ **using** *assms* **by** *auto*
hence 2: $\vdash \Box g$ **by** (*rule BoxGen*)
have 3: $\vdash \Box g \longrightarrow f \text{ syields } g$ **by** (*rule BoxImpSYields*)
from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *SYieldsAndSYieldsEqvSYieldsAnd*:

$\vdash ((f \text{ syields } g) \wedge (f \text{ syields } g1)) = f \text{ syields } (g \wedge g1)$

proof –

have 1: $\vdash f \frown (\neg g \vee \neg g1) = ((f \frown (\neg g)) \vee (f \frown (\neg g1)))$ **by** (rule *SChopOrEqv*)
hence 2: $\vdash ((f \frown (\neg g)) \vee (f \frown (\neg g1))) = f \frown (\neg g \vee \neg g1)$ **by** *auto*
have 3: $\vdash (\neg g \vee \neg g1) = (\neg (g \wedge g1))$ **by** *auto*
hence 4: $\vdash f \frown (\neg g \vee \neg g1) = f \frown (\neg (g \wedge g1))$ **by** (rule *RightSChopEqvSChop*)
have 5: $\vdash (f \frown (\neg g)) \vee (f \frown (\neg g1)) = f \frown (\neg (g \wedge g1))$ **using** 2 4 **by** *fastforce*
hence 6: $\vdash (\neg (f \frown (\neg g)) \wedge \neg (f \frown (\neg g1))) = (\neg (f \frown (\neg (g \wedge g1))))$ **using** 1 4 **by** *fastforce*
from 6 **show** ?thesis **by** (simp add: *syields-d-def*)
qed

lemma *SYieldsAndSYieldsImpAndSYieldsAnd*:

$\vdash (f \text{ yields } g) \wedge (f1 \text{ yields } g1) \longrightarrow (f \wedge f1) \text{ yields } (g \wedge g1)$

proof –

have 1: $\vdash f \text{ yields } g \longrightarrow (f \wedge f1) \text{ yields } g$

by (rule *AndSYieldsA*)

have 2: $\vdash f1 \text{ yields } g1 \longrightarrow (f \wedge f1) \text{ yields } g1$

by (rule *AndSYieldsB*)

have 3: $\vdash ((f \wedge f1) \text{ yields } g \wedge (f \wedge f1) \text{ yields } g1) = (f \wedge f1) \text{ yields } (g \wedge g1)$

by (rule *SYieldsAndSYieldsEqvSYieldsAnd*)

from 1 2 3 **show** ?thesis **by** *fastforce*

qed

lemma *SYieldsSYieldsEqvSChopSYields*:

$\vdash f \text{ yields } (g \text{ yields } h) = (f \frown g) \text{ yields } h$

proof –

have 1: $\vdash f \frown (g \frown (\neg h)) = (f \frown g) \frown (\neg h)$ **by** (rule *SChopAssoc*)

hence 2: $\vdash f \frown (g \frown (\neg h)) = (f \frown g) \frown (\neg h)$ **by** *auto*

have 3: $\vdash g \frown (\neg h) = (\neg \neg (g \frown (\neg h)))$ **by** *auto*

hence 4: $\vdash f \frown (g \frown (\neg h)) = f \frown (\neg \neg (g \frown (\neg h)))$ **by** (rule *RightSChopEqvSChop*)

have 5: $\vdash f \frown (\neg \neg (g \frown (\neg h))) = (f \frown g) \frown (\neg h)$ **using** 2 4 **by** *auto*

hence 6: $\vdash f \frown (\neg (g \text{ yields } h)) = (f \frown g) \frown (\neg h)$ **by** (simp add: *syields-d-def*)

hence 7: $\vdash (\neg (f \frown (\neg (g \text{ yields } h)))) = (\neg ((f \frown g) \frown (\neg h)))$ **by** *auto*

from 7 **show** ?thesis **by** (simp add: *syields-d-def*)

qed

lemma *EmptySYields*:

$\vdash \text{empty} \text{ yields } f = f$

proof –

have 1: $\vdash \text{empty} \frown (\neg f) = (\neg f)$ **by** (rule *EmptySChop*)

hence 2: $\vdash (\neg (\text{empty} \frown (\neg f))) = f$ **by** *auto*

from 2 **show** ?thesis **by** (simp add: *syields-d-def*)

qed

lemma *NextSYields*:

$\vdash (\circ f) \text{ yields } g = \text{wnext } (f \text{ yields } g)$

proof –

have 1: $\vdash (\circ f) \frown (\neg g) = \circ (f \frown (\neg g))$ **by** (rule *NextSChop*)

hence 2: $\vdash (\neg ((\circ f) \frown (\neg g))) = (\neg (\circ (f \frown (\neg g))))$ **by** *auto*

hence 3: $\vdash (\circ f) \text{ yields } g = (\neg (\circ (f \frown (\neg g))))$ **by** (simp add: *syields-d-def*)

have 4: $\vdash (\neg (\circ (f \frown (\neg g)))) = \text{wnext } (\neg (f \frown (\neg g)))$ **by** (auto simp: *wnext-d-def*)

have 5: $\vdash (\circ f) \text{ yields } g = \text{wnext } (\neg (f \frown (\neg g)))$ **using** 3 4 **by** *fastforce*

from 5 show ?thesis by (simp add: syields-d-def)
qed

lemma SkipSChopEqvNext:
 $\vdash \text{skip} \frown f = \bigcirc f$
 by (meson NextSChopdef Prop11)

lemma SkipSYieldsEqvWeakNext:
 $\vdash \text{skip syields } f = \text{wnext } f$
 proof –
 have 1: $\vdash \text{skip} \frown (\neg f) = \bigcirc(\neg f)$ by (rule SkipSChopEqvNext)
 hence 2: $\vdash (\neg (\text{skip} \frown (\neg f))) = (\neg (\bigcirc(\neg f)))$ by auto
 have 3: $\vdash (\neg (\bigcirc(\neg f))) = \text{wnext } f$ by (auto simp: wnext-d-def)
 have 4: $\vdash (\neg (\text{skip} \frown (\neg f))) = \text{wnext } f$ using 2 3 by fastforce
 from 4 show ?thesis by (simp add: syields-d-def)
 qed

lemma NextImpSkipSYields:
 $\vdash \bigcirc f \longrightarrow \text{skip syields } f$
 proof –
 have 1: $\vdash \bigcirc f \longrightarrow \text{wnext } f$ using WnextEqvEmptyOrNext by fastforce
 have 2: $\vdash \text{skip syields } f = \text{wnext } f$ by (rule SkipSYieldsEqvWeakNext)
 from 1 2 show ?thesis by fastforce
 qed

lemma MoreEqvSkipSChopTrue:
 $\vdash \text{more} = \text{skip} \frown \# \text{True}$
 proof –
 have 1: $\vdash \text{skip} \frown \# \text{True} = \bigcirc \# \text{True}$ by (rule SkipSChopEqvNext)
 hence 2: $\vdash \bigcirc \# \text{True} = \text{skip} \frown \# \text{True}$ by auto
 from 2 show ?thesis by (simp add: more-d-def)
 qed

lemma MoreSChopImpMore:
 $\vdash \text{more} \frown f \longrightarrow \text{more}$
 proof –
 have 1: $\vdash (\bigcirc \# \text{True}) \frown f = \bigcirc(\# \text{True} \frown f)$
 by (rule NextSChop)
 have 2: $\vdash \bigcirc(\# \text{True} \frown f) \longrightarrow \text{more}$
 by (metis DiIntro LeftChopImpMoreRule di-d-def more-d-def next-d-def)
 have 3: $\vdash (\bigcirc \# \text{True} \frown f) \longrightarrow \text{more}$
 using 1 2 by fastforce
 from 3 show ?thesis by (metis more-d-def)
 qed

lemma MoreSChopImpFmore:
 $\vdash \text{more} \frown (f \wedge \text{finite}) \longrightarrow \text{fmore}$
 proof –
 have 1: $\vdash \text{more} \frown (f \wedge \text{finite}) = \bigcirc(\# \text{True} \frown (f \wedge \text{finite}))$
 by (simp add: NextSChop more-d-def)

have 2: $\vdash \bigcirc(\#True \frown (f \wedge \text{finite})) \longrightarrow f\text{more}$
by (metis 1 FmoreChopImpFmore fmore-d-def int-eq schop-d-def)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *SChopMoreImpMore*:

$\vdash f \frown \text{more} \longrightarrow \text{more}$

proof –

have 1: $\vdash f \frown \text{more} \longrightarrow \Diamond \text{more}$ **by** (rule *SChopImpDiamond*)
have 2: $\vdash \Diamond \text{more} \longrightarrow \text{more}$ **by** (simp add: *FiniteChopMoreEqvMore int-iffD1 sometimes-d-def*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *MoreSChopEqvNextDiamond*:

$\vdash \text{more} \frown f = \bigcirc(\Diamond f)$

proof –

have 1: $\vdash \text{more} \frown f = (\bigcirc \#True) \frown f$ **by** (simp add: *more-d-def*)
have 2: $\vdash (\bigcirc \#True) \frown f = \bigcirc(\#True \frown f)$ **by** (rule *NextSChop*)
have 3: $\vdash \text{more} \frown f = \bigcirc(\#True \frown f)$ **using** 1 2 **by** fastforce
from 3 **show** ?thesis **by** (metis *TrueSChopEqvDiamond inteq-reflection*)
qed

lemma *WeakNextBoxImpMoreSYields*:

$\vdash \text{more} \text{ syields } f = \text{wnext}(\Box f)$

proof –

have 1: $\vdash \text{more} \frown (\neg f) = \bigcirc(\Diamond (\neg f))$ **by** (rule *MoreSChopEqvNextDiamond*)
have 2: $\vdash \bigcirc(\Diamond (\neg f)) = \bigcirc(\neg(\Box f))$ **by** (auto simp: *always-d-def*)
have 3: $\vdash \bigcirc(\neg(\Box f)) = (\neg(\text{wnext}(\Box f)))$ **by** (auto simp: *wnext-d-def*)
have 4: $\vdash \text{more} \frown (\neg f) = (\neg(\text{more} \text{ syields } f))$ **by** (simp add: *syields-d-def*)
from 1 2 3 4 **show** ?thesis **by** fastforce
qed

lemma *NotEqvSYieldsMore*:

$\vdash \text{finite} \longrightarrow (\neg f) = f \text{ syields } \text{more}$

proof –

have 1: $\vdash \text{finite} \longrightarrow f \frown \text{empty} = f$ **by** (rule *SChopEmpty*)
hence 2: $\vdash \text{finite} \longrightarrow (\neg(f \frown \text{empty})) = (\neg f)$ **by** auto
have 3: $\vdash \text{empty} = (\neg \text{more})$ **by** (auto simp: *empty-d-def*)
hence 4: $\vdash f \frown \text{empty} = f \frown (\neg \text{more})$ **by** (rule *RightSChopEqvSChop*)
hence 5: $\vdash (\neg(f \frown \text{empty})) = (\neg(f \frown (\neg \text{more})))$ **by** auto
have 6: $\vdash \text{finite} \longrightarrow (\neg f) = (\neg(f \frown (\neg \text{more})))$ **using** 2 5 **by** fastforce
from 6 **show** ?thesis **by** (metis *syields-d-def*)
qed

lemma *LeftSChopImpMoreRule*:

assumes $\vdash f \longrightarrow \text{more}$

shows $\vdash f \frown g \longrightarrow \text{more}$

proof –

have 1: $\vdash f \longrightarrow \text{more}$ **using** assms **by** auto
hence 2: $\vdash f \frown g \longrightarrow \text{more} \frown g$ **by** (rule *LeftSChopImpSChop*)

have 3: $\vdash \text{more} \frown g \longrightarrow \text{more}$ **by** (rule *MoreSChopImpMore*)
from 2 3 **show** ?thesis **using** lift-imp-trans **by** blast
qed

lemma *LeftSChopImpFMoreRule*:

assumes $\vdash f \longrightarrow \text{fmore}$
shows $\vdash f \frown (g \wedge \text{finite}) \longrightarrow \text{fmore}$
proof –
have 1: $\vdash f \longrightarrow \text{fmore}$
using assms **by** auto
hence 2: $\vdash f \frown (g \wedge \text{finite}) \longrightarrow \text{more} \frown (g \wedge \text{finite})$
by (metis *FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite FmoreEqvSkipChopFinite LeftSChopImpSChop Prop12 inteq-reflection*)
have 3: $\vdash \text{more} \frown (g \wedge \text{finite}) \longrightarrow \text{fmore}$
using *MoreSChopImpFmore* **by** fastforce
from 2 3 **show** ?thesis **using** lift-imp-trans **by** blast
qed

lemma *RightSChopImpMoreRule*:

assumes $\vdash g \longrightarrow \text{more}$
shows $\vdash f \frown g \longrightarrow \text{more}$
proof –
have 1: $\vdash g \longrightarrow \text{more}$ **using** assms **by** auto
hence 2: $\vdash f \frown g \longrightarrow f \frown \text{more}$ **by** (rule *RightSChopImpSChop*)
have 3: $\vdash f \frown \text{more} \longrightarrow \text{more}$ **by** (rule *SChopMoreImpMore*)
from 2 3 **show** ?thesis **using** lift-imp-trans **by** blast
qed

lemma *NotDfEqvBfNot*:

$\vdash (\neg (df\ f)) = bf\ (\neg\ f)$
proof –
have 1: $\vdash f = (\neg \neg\ f)$ **by** auto
hence 2: $\vdash df\ f = df\ (\neg \neg\ f)$ **by** (rule *DfEqvDf*)
hence 3: $\vdash (\neg (df\ f)) = (\neg (df\ (\neg \neg\ f)))$ **by** auto
from 3 **show** ?thesis **by** (simp add: bf-d-def)
qed

lemma *SChopImpDf*:

$\vdash f \frown g \longrightarrow df\ f$
proof –
have 1: $\vdash g \longrightarrow \#True$ **by** auto
hence 2: $\vdash f \frown g \longrightarrow f \frown \#True$ **by** (rule *RightSChopImpSChop*)
from 2 **show** ?thesis **by** (simp add: df-d-def)
qed

lemma *TrueEqvTrueSChopTrue*:

$\vdash \#True = \#True \frown \#True$
proof –
have 1: $\vdash \#True \frown \#True \longrightarrow \#True$

by auto
 have 2: $\vdash \#True \longrightarrow \#True \frown \#True$
 by (metis DfState Initprop(4) df-d-def int-eq-true int-iffD1 inteq-reflection)
 from 1 2 show ?thesis by auto
 qed

lemma DfEqvDfDf:
 $\vdash df\ f = df\ (df\ f)$
proof –
 have 1: $\vdash \#True = \#True \frown \#True$ by (rule TrueEqvTrueSChopTrue)
 hence 2: $\vdash f \frown \#True = f \frown (\#True \frown \#True)$ by (rule RightSChopEqvSChop)
 have 3: $\vdash f \frown (\#True \frown \#True) = (f \frown \#True) \frown \#True$ by (rule SChopAssoc)
 have 4: $\vdash f \frown \#True = (f \frown \#True) \frown \#True$ using 2 3 by fastforce
 from 4 show ?thesis by (metis df-d-def)
 qed

lemma BfEqvBfBf:
 $\vdash bf\ f = bf\ (bf\ f)$
proof –
 have 1: $\vdash df\ (\neg f) = df\ (df\ (\neg f))$ by (rule DfEqvDfDf)
 have 2: $\vdash df\ (\neg f) = \neg (bf\ f)$ by (rule DfNotEqvNotBf)
 hence 3: $\vdash df\ (df\ (\neg f)) = df\ (\neg (bf\ f))$ by (rule DfEqvDf)
 have 4: $\vdash df\ (\neg f) = df\ (\neg (bf\ f))$ using 1 3 by fastforce
 hence 5: $\vdash (\neg (df\ (\neg f))) = (\neg (df\ (\neg (bf\ f))))$ by fastforce
 from 5 show ?thesis by (metis bf-d-def)
 qed

lemma BfImpBfBf:
 $\vdash bf\ f \longrightarrow bf\ (bf\ f)$
proof –
 have 1: $\vdash bf\ (bf\ f) = bf\ f$ using BfEqvBfBf by fastforce
 from 1 show ?thesis by (simp add: int-iffD2)
 qed

lemma DfOrEqv:
 $\vdash df\ (f \vee g) = (df\ f \vee df\ g)$
proof –
 have 1: $\vdash (f \vee g) \frown \#True = (f \frown \#True \vee g \frown \#True)$ by (rule OrSChopEqv)
 from 1 show ?thesis by (simp add: df-d-def)
 qed

lemma DfAndA:
 $\vdash df\ (f \wedge g) \longrightarrow df\ f$
proof –
 have 1: $\vdash (f \wedge g) \frown \#True \longrightarrow f \frown \#True$ by (rule AndSChopA)
 from 1 show ?thesis by (simp add: df-d-def)

qed

lemma *DfAndB*:

$\vdash df (f \wedge g) \longrightarrow df g$

proof –

have 1: $\vdash (f \wedge g) \frown \#True \longrightarrow g \frown \#True$ **by** (rule *AndSChopB*)

from 1 **show** ?thesis **by** (simp add: df-d-def)

qed

lemma *DfAndImpAnd*:

$\vdash df (f \wedge g) \longrightarrow df f \wedge df g$

proof –

have 1: $\vdash df (f \wedge g) \longrightarrow df f$ **by** (rule *DfAndA*)

have 2: $\vdash df (f \wedge g) \longrightarrow df g$ **by** (rule *DfAndB*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *DfSkipEqvMore*:

$\vdash df skip = more$

proof –

have 1: $\vdash skip \frown \#True = \bigcirc \#True$ **by** (rule *SkipSChopEqvNext*)

have 2: $\vdash \bigcirc \#True = more$ **by** (auto simp: more-d-def)

have 3: $\vdash skip \frown \#True = more$ **using** 1 2 **by** fastforce

from 3 **show** ?thesis **by** (simp add: df-d-def)

qed

lemma *DfMoreEqvMore*:

$\vdash df more = more$

proof –

have 1: $\vdash df (\bigcirc \#True) = \bigcirc(df \#True)$ **by** (rule *DfNext*)

have 2: $\vdash \bigcirc(df \#True) \longrightarrow more$

by (metis *ChopImpDi di-d-def more-d-def next-d-def*)

have 3: $\vdash df (\bigcirc \#True) \longrightarrow more$ **using** 1 2 **by** fastforce

hence 4: $\vdash df more \longrightarrow more$ **by** (simp add: more-d-def)

have 5: $\vdash more \longrightarrow df more$

by (metis 1 4 *TrueEqvTrueSChopTrue df-d-def inteq-reflection more-d-def*)

from 4 5 **show** ?thesis **by** fastforce

qed

lemma *DfIfEqvRule*:

assumes $\vdash f = if_i (init w) then g else h$

shows $\vdash df f = if_i (init w) then (df g) else (df h)$

proof –

have 1: $\vdash f = if_i (init w) then g else h$ **using** assms **by** auto

hence 2: $\vdash f \frown \#True = if_i (init w) then (g \frown \#True) else (h \frown \#True)$ **by** (rule *IfSChopEqvRule*)

from 2 **show** ?thesis **by** (simp add: df-d-def)

qed

lemma *SDaNotEqvNotSBa*:

$\vdash sda (\neg f) = (\neg (sba f))$

proof –
have 1: $\vdash \text{ sba } f = (\neg (\text{ sda } (\neg f)))$ **by** (*simp add: sba-d-def*)
from 1 **show** ?thesis **by** fastforce
qed

lemma *SDaEqvSDa*:
assumes $\vdash f = g$
shows $\vdash \text{ sda } f = \text{ sda } g$
using *assms* **using** *int-eq* **by** force

lemma *SDaEqvNotSBaNot*:
 $\vdash \text{ sda } f = (\neg (\text{ sba } (\neg f)))$
proof –
have 1: $\vdash \text{ sba } (\neg f) = (\neg (\text{ sda } (\neg \neg f)))$ **by** (*simp add: sba-d-def*)
hence 2: $\vdash \text{ sda } (\neg \neg f) = (\neg (\text{ sba } (\neg f)))$ **by** fastforce
have 3: $\vdash f = (\neg \neg f)$ **by** *simp*
hence 4: $\vdash \text{ sda } f = \text{ sda } (\neg \neg f)$ **by** (*rule SDaEqvSDa*)
from 2 4 **show** ?thesis **by** *simp*
qed

lemma *SBaElim*:
 $\vdash \text{ sba } f \wedge \text{ finite} \longrightarrow f$
proof –
have 1: $\vdash \text{ sba } f = \Box(\text{bf } f)$
by (*rule SBaEqvBtBf*)
have 2: $\vdash \text{bf } f \wedge \text{ finite} \longrightarrow f$
by (*rule BfElim*)
hence 3: $\vdash \Box(\text{bf } f \wedge \text{ finite} \longrightarrow f)$
by (*rule BoxGen*)
have 4: $\vdash \Box(\text{bf } f \wedge \text{ finite} \longrightarrow f) \longrightarrow \Box(\text{bf } f \wedge \text{ finite}) \longrightarrow \Box f$
by (*rule BoxImpDist*)
have 5: $\vdash \Box(\text{bf } f \wedge \text{ finite}) \longrightarrow \Box f$
using 3 4 *MP* **by** fastforce
have 6: $\vdash \Box(\text{bf } f \wedge \text{ finite}) = (\Box(\text{bf } f) \wedge \text{ finite})$
by (*metis (no-types, lifting) BoxEqvFiniteYields FiniteChopInfEqvInf NotChopEqvYieldsNot YieldsAndYieldsEqvYieldsAnd finite-d-def inteq-reflection*)
have 7: $\vdash \Box f \longrightarrow f$
by (*rule BoxElim*)
from 1 5 6 7 **show** ?thesis **using** *SBaImpBt lift-imp-trans* **by** *metis*
qed

lemma *SDaIntro*:
 $\vdash f \wedge \text{ finite} \longrightarrow \text{ sda } f$
proof –
have 1: $\vdash \text{ sba } (\neg f) \wedge \text{ finite} \longrightarrow (\neg f)$ **by** (*rule SBaElim*)
hence 2: $\vdash \neg \neg f \longrightarrow \neg (\text{ sba } (\neg f) \wedge \text{ finite})$ **by** fastforce
have 3: $\vdash f = (\neg \neg f)$ **by** *simp*
have 4: $\vdash \text{ sda } f = (\neg (\text{ sba } (\neg f)))$ **by** (*rule SDaEqvNotSBaNot*)
from 2 3 4 **show** ?thesis **by** fastforce
qed

lemma *SBaGen*:

assumes $\vdash f$
shows $\vdash sba\ f$

proof –

have 1: $\vdash f$ **using** *assms* **by** *auto*
hence 2: $\vdash \Box f$ **by** (*rule BoxGen*)
hence 3: $\vdash bf(\Box f)$ **by** (*rule BfGen*)
have 4: $\vdash sba\ f = bf(\Box f)$ **by** (*rule SBaEqvBfBt*)
from 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *SBaImpDist*:

$\vdash sba\ (f \longrightarrow g) \longrightarrow sba\ f \longrightarrow sba\ g$

proof –

have 1: $\vdash bf\ (f \longrightarrow g) \longrightarrow (bf\ f \longrightarrow bf\ g)$
by (*rule BfImpDist*)
hence 2: $\vdash \Box(bf\ (f \longrightarrow g) \longrightarrow (bf\ f \longrightarrow bf\ g))$
by (*rule BoxGen*)
have 3: $\vdash \Box(bf\ (f \longrightarrow g) \longrightarrow (bf\ f \longrightarrow bf\ g))$
 \longrightarrow
 $(\Box(bf\ (f \longrightarrow g)) \longrightarrow (\Box(bf\ f) \longrightarrow \Box(bf\ g)))$
by (*meson 2 BoxImpDist MP lift-imp-trans Prop01 Prop05 Prop09*)
have 4: $\vdash \Box(bf\ (f \longrightarrow g)) \longrightarrow (\Box(bf\ f) \longrightarrow \Box(bf\ g))$
using 2 3 *MP* **by** *fastforce*
have 5: $\vdash sba\ (f \longrightarrow g) = \Box(bf\ (f \longrightarrow g))$
by (*rule SBaEqvBtBf*)
have 6: $\vdash sba\ f = \Box(bf\ f)$
by (*rule SBaEqvBtBf*)
have 7: $\vdash sba\ g = \Box(bf\ g)$
by (*rule SBaEqvBtBf*)
from 4 5 6 7 **show** *?thesis* **by** *fastforce*

qed

lemma *SBaAndEqv*:

$\vdash sba\ (f \wedge g) = (sba\ f \wedge sba\ g)$

proof –

have 1: $\vdash sba\ (f \wedge g) = \Box(bf\ (f \wedge g))$
by (*rule SBaEqvBtBf*)
have 2: $\vdash bf\ (f \wedge g) = (bf\ f \wedge bf\ g)$
by (*simp add: BfAndEqvBfAndBf*)
hence 3: $\vdash \Box(bf\ (f \wedge g)) = \Box(bf\ f \wedge bf\ g)$
using *BoxEqvBox* **by** *blast*
have 4: $\vdash \Box(bf\ f \wedge bf\ g) = (\Box(bf\ f) \wedge \Box(bf\ g))$
by (*metis 2 BoxAndBoxEqvBoxRule inteq-reflection*)
have 5: $\vdash sba\ f = \Box(bf\ f)$
by (*rule SBaEqvBtBf*)
have 6: $\vdash sba\ g = \Box(bf\ g)$
by (*rule SBaEqvBtBf*)
from 1 3 4 5 6 **show** *?thesis* **by** *fastforce*

qed

lemma *SBaImpSBaEqvSBa*:

$\vdash \text{ sba } (f = g) \longrightarrow (\text{ sba } f = \text{ sba } g)$

proof –

have 1: $\vdash \text{ sba } (f \longrightarrow g) \longrightarrow \text{ sba } f \longrightarrow \text{ sba } g$

by (*rule SBaImpDist*)

have 2: $\vdash \text{ sba } (g \longrightarrow f) \longrightarrow \text{ sba } g \longrightarrow \text{ sba } f$

by (*rule SBaImpDist*)

have 3: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$

by *auto*

hence 31: $\vdash \text{ sba}(f = g) = \text{ sba } ((f \longrightarrow g) \wedge (g \longrightarrow f))$

using *inteq-reflection* **by** *force*

have 4: $\vdash \text{ sba } ((f \longrightarrow g) \wedge (g \longrightarrow f)) = (\text{ sba }((f \longrightarrow g)) \wedge \text{ sba }((g \longrightarrow f)))$

by (*rule SBaAndEqv*)

have 5: $\vdash ((\text{ sba } f \longrightarrow \text{ sba } g) \wedge (\text{ sba } g \longrightarrow \text{ sba } f)) = (\text{ sba } f = \text{ sba } g)$

by *auto*

from 1 2 31 4 5 **show** *?thesis* **by** *fastforce*

qed

lemma *SBaImpSBa*:

assumes $\vdash f \longrightarrow g$

shows $\vdash \text{ sba } f \longrightarrow \text{ sba } g$

using *SBaGen SBaImpDist MP assms* **by** *metis*

lemma *SBaEqvSBa*:

assumes $\vdash f = g$

shows $\vdash \text{ sba } f = \text{ sba } g$

using *SBaGen SBaImpSBaEqvSBa MP assms* **by** *metis*

lemma *SDaImpSDa*:

assumes $\vdash f \longrightarrow g$

shows $\vdash \text{ sda } f \longrightarrow \text{ sda } g$

using *assms* **by** (*metis SDaEqvDtDf DfAndB DiamondImpDiamond inteq-reflection Prop10*)

lemma *SDaEqvSDaSDa*:

$\vdash \text{ sda } f = \text{ sda } (\text{ sda } f)$

proof –

have 1: $\vdash \text{ sda } f = \Diamond (df f)$

by (*rule SDaEqvDtDf*)

have 2: $\vdash df f = (df (\text{ sda } f))$

by (*rule DfEqvDfDf*)

hence 3: $\vdash \Diamond (\text{ sda } f) = \Diamond (df (\text{ sda } f))$

by (*rule DiamondEqvDiamond*)

have 4: $\vdash \Diamond (df f) = \Diamond (\Diamond (df (\text{ sda } f)))$

using *DiamondEqvDiamondDiamond DfEqvDfDf* **using** 3 **by** *fastforce*

have 5: $\vdash \Diamond (df (\text{ sda } f)) = df (\Diamond (\text{ sda } f))$

by (*rule DtDfEqvDfDt*)

hence 6: $\vdash \Diamond (\Diamond (df (\text{ sda } f))) = \Diamond (df (\Diamond (\text{ sda } f)))$

by (*rule DiamondEqvDiamond*)

have 7: $\vdash sda\ f = \Diamond (df\ (\Diamond\ (df\ f)))$
using 1 3 4 6 **by** *fastforce*
have 8: $\vdash sda\ (\Diamond\ (df\ f)) = \Diamond\ (df\ (\Diamond\ (df\ f)))$
by (rule *SDaEqvDtDf*)
have 9: $\vdash sda\ (sda\ f) = sda\ (\Diamond\ (df\ f))$
using 1 **by** (rule *SDaEqvSDa*)
from 7 8 9 **show** *?thesis* **by** *fastforce*
qed

lemma *SBaEqvSBaSBa*:

$\vdash sba\ f = sba\ (sba\ f)$

proof –

have 1: $\vdash sda\ (\neg\ f) = sda\ (sda\ (\neg\ f))$ **by** (rule *SDaEqvSDaSDa*)
have 2: $\vdash sda\ (sda\ (\neg\ f)) = (\neg\ (sba\ (\neg\ (sda\ (\neg\ f)))))$ **by** (rule *SDaEqvNotSBaNot*)
have 3: $\vdash (\neg\ (sda\ (sda\ (\neg\ f)))) = sba\ (\neg\ (sda\ (\neg\ f)))$ **by** (auto simp: *sba-d-def*)
have 4: $\vdash (\neg\ (sda\ (\neg\ f))) = sba\ (\neg\ (sda\ (\neg\ f)))$ **using** 1 2 3 **by** *fastforce*
from 4 **show** *?thesis* **by** (metis *sba-d-def*)
qed

lemma *SBaLeftSChopImpSChop*:

$\vdash sba\ (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$

proof –

have 1: $\vdash sba\ (f \longrightarrow f1) \longrightarrow bf\ (f \longrightarrow f1)$ **by** (rule *SBaImpBf*)
have 2: $\vdash bf\ (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (rule *BfSChopImpSChop*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *BaLeftSChopImpSChop*:

$\vdash ba\ (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$

proof –

have 1: $\vdash ba\ (f \longrightarrow f1) \longrightarrow bf\ (f \longrightarrow f1)$ **by** (rule *BaImpBf*)
have 2: $\vdash bf\ (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (rule *BfSChopImpSChop*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *SBaRightSChopImpSChop*:

$\vdash sba\ (g \longrightarrow g1) \wedge finite \longrightarrow f \frown g \longrightarrow f \frown g1$

proof –

have 1: $\vdash sba\ (g \longrightarrow g1) \wedge finite \longrightarrow \Box(g \longrightarrow g1)$ **by** (rule *SBaImpBt*)
have 2: $\vdash \Box(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **by** (rule *BoxSChopImpSChop*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *BaRightSChopImpSChop*:

$\vdash ba\ (g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$

proof –

have 1: $\vdash ba\ (g \longrightarrow g1) \longrightarrow \Box(g \longrightarrow g1)$ **by** (rule *BaImpBt*)
have 2: $\vdash \Box(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **by** (rule *BoxSChopImpSChop*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *SChopAndSBaImport*:

$\vdash (f \frown f1) \wedge sba\ g \wedge finite \longrightarrow (f \wedge g) \frown (f1 \wedge g)$

proof –

have 1: $\vdash sba\ g \wedge finite \wedge (f \frown f1) \longrightarrow (g \wedge f) \frown (g \wedge f1)$ **by** (rule *SBaAndSChopImport*)

have 2: $\vdash (g \wedge f) \frown (g \wedge f1) = (f \wedge g) \frown (f1 \wedge g)$ **by** (rule *AndSChopAndCommute*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *SChopAndBaImport*:

$\vdash (f \frown f1) \wedge ba\ g \longrightarrow (f \wedge g) \frown (f1 \wedge g)$

proof –

have 1: $\vdash ba\ g \wedge (f \frown f1) \longrightarrow (g \wedge f) \frown (g \wedge f1)$ **by** (rule *BaAndSChopImport*)

have 2: $\vdash (g \wedge f) \frown (g \wedge f1) = (f \wedge g) \frown (f1 \wedge g)$ **by** (rule *AndSChopAndCommute*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *BaAndSChopImportA*:

$\vdash ba\ f \wedge g \frown g1 \longrightarrow (f \wedge g) \frown g1$

by (meson *BaAndSChopImport SChopAndB lift-imp-trans*)

lemma *BaAndSChopImportB*:

$\vdash ba\ f \wedge g \frown g1 \longrightarrow (f \wedge g) \frown (ba\ f \wedge g1)$

proof –

have 1: $\vdash ba\ f = ba\ (ba\ f)$

by (simp add: *BaEqvBaBa*)

have 2: $\vdash ba\ (ba\ f) \wedge g \frown g1 \longrightarrow g \frown (ba\ f \wedge g1)$

by (metis *AndSChopB BaAndSChopImport lift-imp-trans*)

have 3: $\vdash ba\ f \wedge g \frown (ba\ f \wedge g1) \longrightarrow (f \wedge g) \frown (ba\ f \wedge g1)$

by (simp add: *BaAndSChopImportA*)

from 1 2 3 **show** ?thesis **by** fastforce

qed

lemma *SBaImpSBaImpSBaAnd*:

$\vdash sba\ h \longrightarrow sba(g \longrightarrow sba\ h \wedge g)$

proof –

have 1: $\vdash sba\ h \longrightarrow (g \longrightarrow sba\ h \wedge g)$ **by** fastforce

hence 2: $\vdash sba(sba\ h) \longrightarrow sba(g \longrightarrow sba\ h \wedge g)$ **by** (rule *SBaImpSBa*)

have 3: $\vdash sba\ h = sba(sba\ h)$ **by** (rule *SBaEqvSBaSBa*)

from 2 3 **show** ?thesis **by** fastforce

qed

lemma *SBaSChopImpSChopSBa*:

$\vdash sba\ f \wedge finite \longrightarrow g \frown g1 \longrightarrow g \frown ((sba\ f) \wedge g1)$

proof –

have 1: $\vdash sba\ f \longrightarrow sba\ (g1 \longrightarrow (sba\ f) \wedge g1)$

by (rule *SBaImpSBaImpSBaAnd*)

have 2: $\vdash sba\ (g1 \longrightarrow sba\ f \wedge g1) \wedge finite \longrightarrow g \frown g1 \longrightarrow g \frown (sba\ f \wedge g1)$

by (rule *SBaRightSChopImpSChop*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *BaSChopImpSChopBa*:

$\vdash \text{ba } f \longrightarrow g \frown g1 \longrightarrow g \frown ((\text{ba } f) \wedge g1)$

proof –

have 1: $\vdash \text{ba } f \longrightarrow \text{ba } (g1 \longrightarrow (\text{ba } f) \wedge g1)$

by (rule *BaImpBaImpBaAnd*)

have 2: $\vdash \text{ba } (g1 \longrightarrow \text{ba } f \wedge g1) \longrightarrow g \frown g1 \longrightarrow g \frown (\text{ba } f \wedge g1)$

by (rule *BaRightSChopImpSChop*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *DfNotSBaImpNotSBa*:

$\vdash \text{df } (\neg (\text{sba } f)) \longrightarrow \neg (\text{sba } f)$

proof –

have 1: $\vdash \text{sba } f = \text{sba } (\text{sba } f)$ **by** (rule *SBaEqvSBaSBa*)

have 2: $\vdash \text{sba } (\text{sba } f) \longrightarrow \text{bf } (\text{sba } f)$ **by** (rule *SBaImpBf*)

have 3: $\vdash \text{sba } f \longrightarrow \text{bf } (\text{sba } f)$ **using** 1 2 **by** fastforce

hence 4: $\vdash \text{sba } f \longrightarrow \neg (\text{df } (\neg (\text{sba } f)))$ **by** (simp add: bf-d-def)

from 4 **show** ?thesis **by** fastforce

qed

lemma *DfNotBaImpNotBa*:

$\vdash \text{df } (\neg (\text{ba } f)) \longrightarrow \neg (\text{ba } f)$

proof –

have 1: $\vdash \text{ba } f = \text{ba } (\text{ba } f)$ **by** (rule *BaEqvBaBa*)

have 2: $\vdash \text{ba } (\text{ba } f) \longrightarrow \text{bf } (\text{ba } f)$ **by** (rule *BaImpBf*)

have 3: $\vdash \text{ba } f \longrightarrow \text{bf } (\text{ba } f)$ **using** 1 2 **by** fastforce

hence 4: $\vdash \text{ba } f \longrightarrow \neg (\text{df } (\neg (\text{ba } f)))$ **by** (simp add: bf-d-def)

from 4 **show** ?thesis **by** fastforce

qed

lemma *NotSBaSChopImpNotSBa*:

$\vdash (\neg (\text{sba } f)) \frown g \longrightarrow \neg (\text{sba } f)$

proof –

have 1: $\vdash (\neg (\text{sba } f)) \frown g \longrightarrow \text{df } (\neg (\text{sba } f))$ **by** (rule *SChopImpDf*)

have 2: $\vdash \text{df } (\neg (\text{sba } f)) \longrightarrow \neg (\text{sba } f)$ **by** (rule *DfNotSBaImpNotSBa*)

from 1 2 **show** ?thesis **using** lift-imp-trans **by** blast

qed

lemma *NotBaSChopImpNotSBa*:

$\vdash (\neg (\text{ba } f)) \frown g \longrightarrow \neg (\text{ba } f)$

proof –

have 1: $\vdash (\neg (\text{ba } f)) \frown g \longrightarrow \text{df } (\neg (\text{ba } f))$ **by** (rule *SChopImpDf*)

have 2: $\vdash \text{df } (\neg (\text{ba } f)) \longrightarrow \neg (\text{ba } f)$ **by** (rule *DfNotBaImpNotBa*)

from 1 2 **show** ?thesis **using** lift-imp-trans **by** blast

qed

lemma *DiamondSFinImpSFin*:

$\vdash \Diamond (\text{sfin } f) \longrightarrow \text{sfin } f$

proof –

have 1: $\vdash \text{sfm } f = \# \text{True} \frown (f \wedge \text{empty})$
by (rule *SFinEqvTrueSChopAndEmpty*)
hence 2: $\vdash \Diamond (\text{sfm } f) = \# \text{True} \frown (\# \text{True} \frown (f \wedge \text{empty}))$
by (metis *DiamondSChopdef inteq-reflection*)
have 3: $\vdash \# \text{True} \frown (\# \text{True} \frown (f \wedge \text{empty})) = (\# \text{True} \frown \# \text{True}) \frown (f \wedge \text{empty})$
by (rule *SChopAssoc*)
have 4: $\vdash (\# \text{True} \frown \# \text{True}) \frown (f \wedge \text{empty}) \longrightarrow \# \text{True} \frown (f \wedge \text{empty})$
using 1 2 3
by (metis *SChopImpDiamond TrueEqvTrueSChopTrue inteq-reflection*)
from 1 2 3 4 **show** ?thesis **by** fastforce
qed

lemma *SChopSFinImpSFin*:

$\vdash f \frown \text{sfm } (\text{init } w) \longrightarrow \text{sfm } (\text{init } w)$

proof –

have 1: $\vdash f \frown \text{sfm } (\text{init } w) \longrightarrow \Diamond (\text{sfm } (\text{init } w))$ **by** (rule *SChopImpDiamond*)
have 2: $\vdash \Diamond (\text{sfm } (\text{init } w)) \longrightarrow \text{sfm } (\text{init } w)$ **by** (rule *DiamondSFinImpSFin*)
from 1 2 **show** ?thesis **using** lift-imp-trans **by** blast
qed

lemma *SFinImpSYieldsSFin*:

$\vdash \text{sfm } (\text{init } w) \longrightarrow f \text{syields } (\text{sfm } (\text{init } w))$

proof –

have 1: $\vdash f \frown (\text{sfm } (\text{init } (\neg w))) \longrightarrow (\text{sfm } (\text{init } (\neg w)))$
by (simp add: *SChopSFinImpSFin*)
have 2: $\vdash \text{finite} \longrightarrow (\neg (\text{sfm } (\text{init } w))) = (\text{sfm } (\text{init } (\neg w)))$
using *SFinNotStateEqvNotSFinState* **by** fastforce
hence 3: $\vdash \text{finite} \longrightarrow f \frown (\neg (\text{sfm } (\text{init } w))) = f \frown (\text{sfm } (\text{init } (\neg w)))$
using *FiniteRightSChopEqvSChop* **by** blast
have 4: $\vdash f \frown (\neg (\text{sfm } (\text{init } w))) \wedge \text{finite} \longrightarrow (\neg (\text{sfm } (\text{init } w)))$
using 1 2 3 **by** fastforce
hence 5: $\vdash \text{sfm } (\text{init } w) \longrightarrow \neg (f \frown (\neg (\text{sfm } (\text{init } w))))$
by (metis (no-types, lifting) *DiamondFin SChopImpDiamond int-simps(32) int-simps(4) inteq-reflection sfm-d-def*)
from 5 **show** ?thesis **by** (simp add: *syields-d-def*)
qed

lemma *SChopAndSFin*:

$\vdash ((f \frown g) \wedge (\text{sfm } (\text{init } w))) = f \frown (g \wedge (\text{sfm } (\text{init } w)))$

proof –

have 1: $\vdash \text{sfm } (\text{init } w) \longrightarrow f \text{syields } (\text{sfm } (\text{init } w))$
by (rule *SFinImpSYieldsSFin*)
have 2: $\vdash (f \frown g) \wedge (\text{sfm } (\text{init } w)) \longrightarrow (f \frown g) \wedge f \text{syields } (\text{sfm } (\text{init } w))$
using 1 **by** fastforce
have 3: $\vdash f \frown g \wedge f \text{syields } (\text{sfm } (\text{init } w)) \longrightarrow$
 $f \frown (g \wedge (\text{sfm } (\text{init } w)))$
using *SChopAndSYieldsImp* **by** blast
have 4: $\vdash (f \frown g) \wedge (\text{sfm } (\text{init } w)) \longrightarrow f \frown (g \wedge \text{sfm } (\text{init } w))$
using 2 3 **by** (metis (mono-tags, lifting) lift-imp-trans)

from 4 **show** ?thesis

by (simp add: Prop12 SChopAndA SChopSFinExportA int-iffI)

qed

lemma SChopAndNotSFin:

$\vdash (f \frown g \wedge \neg (sfin (init w)) \wedge finite) = f \frown (g \wedge \neg (sfin (init w)) \wedge finite)$

proof –

have 1: $\vdash (f \frown g \wedge sfin (init (\neg w))) = f \frown (g \wedge sfin (init (\neg w)))$

by (rule SChopAndSFin)

have 2: $\vdash (sfin (init (\neg w)) \wedge finite) = (\neg (sfin (init w))) \wedge finite$

using SFinNotStateEqvNotSFinState **by** fastforce

hence 3: $\vdash (g \wedge sfin (init (\neg w))) = (g \wedge \neg (sfin (init w)) \wedge finite)$

using DiamondEmptyEqvFinite SChopAndB SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond **by** fastforce

hence 4: $\vdash f \frown (g \wedge sfin (init (\neg w))) = f \frown (g \wedge \neg (sfin (init w)) \wedge finite)$

using RightSChopEqvSChop **by** blast

from 1 2 4 **show** ?thesis

by (metis DiamondEmptyEqvFinite Prop10 SChopAndB SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond
inteq-reflection)

qed

lemma SFinSChopChain:

$\vdash (((init w) \wedge finite \longrightarrow sfin (init w1))) \frown$
 $((init w1) \wedge finite \longrightarrow sfin (init w2)))$
 $\wedge finite$
 $\longrightarrow (((init w) \wedge finite \longrightarrow sfin (init w2)))$

proof –

have 01: $\vdash (init w \wedge finite \wedge$
 $((init w \wedge finite \longrightarrow sfin (init w1)) \wedge finite); (init w1 \wedge finite \longrightarrow sfin (init w2))) =$
 $(init w \wedge empty) \frown (((init w \wedge finite \longrightarrow sfin (init w1)) \wedge finite);$
 $(init w1 \wedge finite \longrightarrow sfin (init w2)) \wedge finite)$

by (meson Prop04 SChopAndCommute StateAndEmptySChop)

have 02: $\vdash (finite \wedge init w) = (init w \wedge empty) \frown finite$

by (metis StateAndEmptySChop inteq-reflection lift-and-com)

have 03: $\vdash init w \wedge finite \wedge$
 $((init w \wedge finite \longrightarrow sfin (init w1)) \wedge finite);$
 $(init w1 \wedge finite \longrightarrow sfin (init w2)) \longrightarrow finite \wedge init w$

by force

have 04: $\vdash (init w \wedge finite \wedge (init w \wedge finite \longrightarrow sfin (init w1)));$
 $(init w1 \wedge finite \longrightarrow sfin (init w2)) =$
 $(init w \wedge (finite \wedge (init w \wedge finite \longrightarrow sfin (init w1)));$
 $(init w1 \wedge finite \longrightarrow sfin (init w2)))$

by (simp add: StateAndChop)

have 05: $\vdash init w \wedge finite \wedge ((init w \wedge finite \longrightarrow sfin (init w1)) \wedge finite);$
 $(init w1 \wedge finite \longrightarrow sfin (init w2)) \longrightarrow$
 $(init w \wedge finite \wedge (init w \wedge finite \longrightarrow sfin (init w1)));$
 $(init w1 \wedge finite \longrightarrow sfin (init w2))$

by (metis 04 AndChopCommute ChopAndB StateAndEmptyChop int-eq)

have 06: $\vdash init w \wedge finite \wedge ((init w \wedge finite \longrightarrow sfin (init w1)) \wedge finite);$

$$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \longrightarrow$$

$$(init\ w \wedge finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)));$$

$$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite$$

by (meson 03 05 Prop12)

have 1: $\vdash (init\ w) \wedge finite \wedge$
 $((init\ w) \wedge finite \longrightarrow sfin\ (init\ w1)) \frown$
 $((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2))$
 \longrightarrow
 $((init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow sfin\ (init\ w1))) \frown$
 $((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite$

unfolding schop-d-def using 06 ChopAndFiniteDist by fastforce

have 2: $\vdash (init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow sfin\ (init\ w1)) \longrightarrow$
 $sfin\ (init\ w1)$

by auto

have 3: $\vdash ((init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow sfin\ (init\ w1))) \frown$
 $((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite$
 \longrightarrow
 $(sfin\ (init\ w1)) \frown (((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite)$

using 2 LeftSChopImpSChop by blast

have 4: $\vdash (sfin\ (init\ w1)) \frown (((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2))) =$
 $\Diamond((init\ w1) \wedge ((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2)))$

using SFinSChopEqvDiamond by blast

have 41: $\vdash ((init\ w1) \wedge finite \wedge ((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2))) \longrightarrow sfin\ (init\ w2)$

by auto

have 42: $\vdash \Diamond((init\ w1) \wedge finite \wedge ((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2))) \longrightarrow \Diamond(sfin\ (init\ w2))$

using 41 DiamondImpDiamond by blast

have 5: $\vdash \Diamond(sfin\ (init\ w2)) \longrightarrow sfin\ (init\ w2)$

using DiamondSFinImpSFin by blast

have 51: $\vdash \Diamond(init\ w1 \wedge finite \wedge (init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2))) \longrightarrow sfin\ (init\ w2)$

using 42 5 lift-imp-trans by blast

have 52: $\vdash init\ w \wedge finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \frown (init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2))$
 $\longrightarrow sfin\ (init\ w1) \frown ((init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite)$

by (meson 1 3 lift-imp-trans)

have 53: $\vdash init\ w \wedge finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \frown (init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2))$
 $\longrightarrow \Diamond(init\ w1 \wedge (init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite)$

by (metis 52 SFinSChopEqvDiamond inteq-reflection)

have 6: $\vdash (init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow sfin\ (init\ w1)) \frown$
 $((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2))$
 $\longrightarrow sfin\ (init\ w2)$

by (metis 42 5 53 inteq-reflection lift-and-com lift-imp-trans)

from 6 show ?thesis by fastforce

qed

lemma SChopRule:

assumes $\vdash (init\ w) \wedge f \wedge finite \longrightarrow sfin\ (init\ w1)$

$\vdash (init\ w1) \wedge f1 \wedge finite \longrightarrow sfin\ (init\ w2)$

shows $\vdash (init\ w) \wedge (f \frown f1) \wedge finite \longrightarrow sfin\ (init\ w2)$

proof –

have 1: $\vdash (init\ w) \wedge (f \frown f1) \wedge finite \longrightarrow ((init\ w) \wedge f) \frown (f1 \wedge finite)$

by (metis ChopEmpty SChopAssoc StateAndSChopImport inteq-reflection schop-d-def)

have 2: $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w1)$
using *assms* **by** *auto*
have 21: $\vdash (\text{init } w \wedge f \wedge \text{finite}); (f1 \wedge \text{finite}) = ((\text{init } w \wedge f); f1 \wedge \text{finite})$
by (*metis ChopAndFiniteDist StateAndChop StateAndSChop inteq-reflection schop-d-def*)
have 22: $\vdash (\text{init } w \wedge f \wedge \text{finite}) = ((\text{init } w \wedge f \wedge \text{finite}) \wedge \text{sfm } (\text{init } w1))$
using 2 *Prop10* **by** *blast*
hence 3: $\vdash ((\text{init } w) \wedge f) \frown (f1 \wedge \text{finite}) \longrightarrow (\text{sfm } (\text{init } w1)) \frown (f1 \wedge \text{finite})$
using 21 22
by (*metis (no-types, opaque-lifting) 2 ChopEmpty EmptySChop SChopAssoc StateAndSChop StateAndSChopImpSChopRule int-eq schop-d-def*)
have 4: $\vdash (\text{sfm } (\text{init } w1)) \frown (f1 \wedge \text{finite}) = \Diamond((\text{init } w1) \wedge f1 \wedge \text{finite})$
by (*rule SFinSChopEqvDiamond*)
have 5: $\vdash (\text{init } w1) \wedge f1 \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w2)$
using *assms* **by** *auto*
hence 6: $\vdash \Diamond((\text{init } w1) \wedge f1 \wedge \text{finite}) \longrightarrow \Diamond(\text{sfm } (\text{init } w2))$
by (*rule DiamondImpDiamond*)
have 7: $\vdash \Diamond(\text{sfm } (\text{init } w2)) \longrightarrow \text{sfm } (\text{init } w2)$
using *DiamondSFinImpSFin* **by** *blast*
from 1 3 4 6 7 **show** ?thesis **by** *fastforce*
qed

lemma *SChopRep*:

assumes $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow f1 \wedge \text{sfm } (\text{init } w1)$
 $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1$
shows $\vdash (\text{init } w) \wedge ((f \frown g) \wedge \text{finite}) \longrightarrow (f1 \frown g1)$
proof –
have 1: $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow (f1 \wedge \text{sfm } (\text{init } w1))$
using *assms* **by** *auto*
hence 2: $\vdash (\text{init } w) \wedge (f \frown (g \wedge \text{finite})) \longrightarrow (f1 \wedge \text{sfm } (\text{init } w1)) \frown (g \wedge \text{finite})$
by (*metis DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty Prop12 SChopSFinExportA SFinEqvTrueSChopAndEmpty StateAndChopImpChopRule StateAndEmptySChop TrueSChopEqvDiamond inteq-reflection schop-d-def*)
have 3: $\vdash (f1 \wedge \text{sfm } (\text{init } w1)) \frown (g \wedge \text{finite}) = f1 \frown ((\text{init } w1) \wedge (g \wedge \text{finite}))$
using *AndSFinSChopEqvStateAndSChop* **by** *blast*
have 31: $\vdash (\text{init } w) \wedge ((f \frown g) \wedge \text{finite}) \longrightarrow f1 \frown ((\text{init } w1) \wedge (g \wedge \text{finite}))$
using 2 3 **by** (*metis ChopEmpty SChopAssoc inteq-reflection schop-d-def*)
have 4: $\vdash (\text{init } w1) \wedge (g \wedge \text{finite}) \longrightarrow g1$
using *assms* **by** *auto*
hence 5: $\vdash f1 \frown ((\text{init } w1) \wedge (g \wedge \text{finite})) \longrightarrow f1 \frown g1$
using *RightSChopImpSChop* **by** *blast*
show ?thesis **using** 31 5 **by** *fastforce*
qed

lemma *SChopRepAndSFin*:

assumes $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow f1 \wedge \text{sfm } (\text{init } w1)$
 $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1 \wedge \text{sfm } (\text{init } w2)$
shows $\vdash (\text{init } w) \wedge (f \frown g) \wedge \text{finite} \longrightarrow (f1 \frown g1) \wedge \text{sfm } (\text{init } w2)$
proof –
have 1: $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow f1 \wedge \text{sfm } (\text{init } w1)$
using *assms* **by** *auto*


```

have 2:  $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1 \wedge \text{sfine } (\text{init } w2)$ 
  using assms by auto
have 3:  $\vdash (\text{init } w) \wedge (f \frown g) \wedge \text{finite} \longrightarrow f1 \frown (g1 \wedge \text{sfine } (\text{init } w2))$ 
  using 1 2 by (rule SChopRep)
have 4:  $\vdash f1 \frown (g1 \wedge \text{sfine } (\text{init } w2)) \longrightarrow f1 \frown g1$ 
  by (rule SChopAndA)
have 5:  $\vdash f1 \frown (g1 \wedge \text{sfine } (\text{init } w2)) \longrightarrow f1 \frown \text{sfine } (\text{init } w2)$ 
  by (rule SChopAndB)
have 6:  $\vdash f1 \frown \text{sfine } (\text{init } w2) \longrightarrow \text{sfine } (\text{init } w2)$ 
  by (rule SChopSFinImpSFin)
show ?thesis
  by (metis 3 4 5 6 Prop12 lift-imp-trans)
qed

```

lemma *TrueSChopMoreEqvMore*:

$\vdash \# \text{True} \frown \text{more} = \text{more}$

by (*metis* *ChopAssoc TrueChopMoreEqvMore TrueEqvTrueSChopTrue inteq-reflection schop-d-def*)

lemma *SChopFmoreEqvFmore*:

$\vdash \# \text{True} \frown \text{fmore} = \text{fmore}$

by (*simp* *add: FiniteChopFmoreEqvFmore schop-d-def*)

lemma *MoreSChopLoop*:

assumes $\vdash f \longrightarrow \text{more} \frown f$

shows $\vdash \text{finite} \longrightarrow \neg f$

proof –

have 1: $\vdash f \longrightarrow \text{more} \frown f$

using *assms* **by** *auto*

hence 11: $\vdash \Diamond (f) \longrightarrow \Diamond (\text{more} \frown f)$

using *DiamondImpDiamond* **by** *blast*

have 12: $\vdash \Diamond (\text{more} \frown f) = \# \text{True} \frown (\text{more} \frown f)$

by (*simp* *add: DiamondSChopdef*)

have 13: $\vdash \# \text{True} \frown (\text{more} \frown f) = (\# \text{True} \frown \text{more}) \frown f$

by (*rule* *SChopAssoc*)

have 14: $\vdash \Diamond (\text{more} \frown f) = \text{more} \frown f$

using 12 13 **by** (*metis* *TrueSChopMoreEqvMore inteq-reflection*)

have 2: $\vdash \text{more} \frown f = \bigcirc (\Diamond f)$

using *MoreSChopEqvNextDiamond* **by** *blast*

have 3: $\vdash \Diamond (f) \longrightarrow \bigcirc (\Diamond f)$

using 11 14 2 **by** *fastforce*

hence 4: $\vdash \text{finite} \longrightarrow \neg (\Diamond f)$

using *NextLoop* **by** *blast*

have 5: $\vdash \neg (\Diamond f) \longrightarrow \neg f$

using *NowImpDiamond* **by** *fastforce*

from 4 5 **show** ?thesis **using** *lift-imp-trans* **by** *blast*

qed

lemma *MoreSChopContra*:

assumes $\vdash f \wedge \neg g \longrightarrow (\text{more} \frown (f \wedge \neg g))$

shows $\vdash f \wedge \text{finite} \longrightarrow g$

proof –
have 1: $\vdash f \wedge \neg g \longrightarrow (more \smallfrown (f \wedge \neg g))$ **using** *assms* **by** *auto*
hence 2: $\vdash finite \longrightarrow \neg (f \wedge \neg g)$ **by** (rule *MoreSChopLoop*)
from 2 **show** *?thesis* **by** *fastforce*
qed

lemma *MoreSChopLoopFinite*:

assumes $\vdash f \wedge finite \longrightarrow more \smallfrown f$
shows $\vdash finite \longrightarrow \neg f$
proof –
have 1: $\vdash f \wedge finite \longrightarrow more \smallfrown f$
using *assms* **by** *auto*
hence 11: $\vdash \Diamond (f \wedge finite) \longrightarrow \Diamond (more \smallfrown f)$
using *DiamondImpDiamond* **by** *blast*
have 12: $\vdash \Diamond (more \smallfrown f) = \#True \smallfrown (more \smallfrown f)$
by (*simp add: DiamondSChopdef*)
have 13: $\vdash \#True \smallfrown (more \smallfrown f) = (\#True \smallfrown more) \smallfrown f$
by (rule *SChopAssoc*)
have 14: $\vdash \Diamond (more \smallfrown f) = more \smallfrown f$
using 12 13 **by** (*metis TrueSChopMoreEqvMore inteq-reflection*)
have 2: $\vdash more \smallfrown f = \bigcirc(\Diamond f)$
using *MoreSChopEqvNextDiamond* **by** *blast*
have 3: $\vdash \Diamond (f \wedge finite) \longrightarrow \bigcirc(\Diamond f)$
using 11 14 2 **by** *fastforce*
have 31: $\vdash \Diamond (f \wedge finite) = ((\Diamond f) \wedge finite)$
by (*metis (no-types, lifting) DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty SFinAndSChop SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection lift-and-com*)
have 32: $\vdash (\Diamond f) \wedge finite \longrightarrow \bigcirc(\Diamond f)$
using 3 31 **by** *fastforce*
hence 4: $\vdash finite \longrightarrow \neg (\Diamond f)$
by (*metis (no-types, lifting) DiamondIntro FiniteChopInfEqvInf InfEqvNotFinite Prop09 finite-d-def int-simps(15) int-simps(32) inteq-reflection sometimes-d-def*)
have 5: $\vdash \neg (\Diamond f) \longrightarrow \neg f$
by (*simp add: NowImpDiamond*)
from 4 5 **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *MoreSChopContraFinite*:

assumes $\vdash (f \wedge \neg g) \wedge finite \longrightarrow (more \smallfrown (f \wedge \neg g))$
shows $\vdash f \wedge finite \longrightarrow g$
proof –
have 1: $\vdash (f \wedge \neg g) \wedge finite \longrightarrow (more \smallfrown (f \wedge \neg g))$ **using** *assms* **by** *auto*
hence 2: $\vdash finite \longrightarrow \neg (f \wedge \neg g)$ **by** (*simp add: MoreSChopLoopFinite*)
from 2 **show** *?thesis* **by** (*simp add: Valid-def*)
qed

lemma *SChopLoop*:

assumes $\vdash f \longrightarrow g \smallfrown f$
 $\vdash g \longrightarrow f \smallfrown g$
shows $\vdash finite \longrightarrow \neg f$

```

proof –
have 1:  $\vdash f \longrightarrow g \frown f$  using assms by auto
have 2:  $\vdash g \longrightarrow \text{more}$  using assms by (simp add: Prop12 fmore-d-def)
hence 3:  $\vdash g \frown f \longrightarrow \text{more} \frown f$  by (rule LeftSChopImpSChop)
have 4:  $\vdash f \longrightarrow \text{more} \frown f$  using 1 3 by fastforce
from 4 show ?thesis using MoreSChopLoop by auto
qed

```

```

lemma SChopLoopB:
  assumes  $\vdash f \longrightarrow g \frown f$ 
            $\vdash g \longrightarrow \text{more}$ 
  shows  $\vdash \text{finite} \longrightarrow \neg f$ 
proof –
have 1:  $\vdash f \longrightarrow g \frown f$  using assms by auto
have 2:  $\vdash g \longrightarrow \text{more}$  using assms by auto
hence 3:  $\vdash g \frown f \longrightarrow \text{more} \frown f$  by (rule LeftSChopImpSChop)
have 4:  $\vdash f \longrightarrow \text{more} \frown f$  using 1 3 by fastforce
from 4 show ?thesis using MoreSChopLoop by blast
qed

```

```

lemma SChopContra:
  assumes  $\vdash f \wedge \neg g \longrightarrow h \frown f \wedge \neg (h \frown g)$ 
            $\vdash h \longrightarrow \text{fmore}$ 
  shows  $\vdash f \wedge \text{finite} \longrightarrow g$ 
proof –
have 1:  $\vdash f \wedge \neg g \longrightarrow h \frown f \wedge \neg (h \frown g)$  using assms by auto
have 2:  $\vdash h \longrightarrow \text{more}$  using assms by (simp add: Prop12 fmore-d-def)
have 3:  $\vdash h \frown f \wedge \neg (h \frown g) \longrightarrow h \frown (f \wedge \neg g)$  by (rule SChopAndNotSChopImp)
have 4:  $\vdash h \frown (f \wedge \neg g) \longrightarrow \text{more} \frown (f \wedge \neg g)$  using 2 by (rule LeftSChopImpSChop)
have 5:  $\vdash f \wedge \neg g \longrightarrow \text{more} \frown (f \wedge \neg g)$  using 1 3 4 by fastforce
from 5 show ?thesis using MoreSChopContra by auto
qed

```

```

lemma SChopContraB:
  assumes  $\vdash f \wedge \neg g \longrightarrow h \frown f \wedge \neg (h \frown g)$ 
            $\vdash h \longrightarrow \text{more}$ 
  shows  $\vdash f \wedge \text{finite} \longrightarrow g$ 
proof –
have 1:  $\vdash f \wedge \neg g \longrightarrow h \frown f \wedge \neg (h \frown g)$  using assms by auto
have 2:  $\vdash h \longrightarrow \text{more}$  using assms by auto
have 3:  $\vdash h \frown f \wedge \neg (h \frown g) \longrightarrow h \frown (f \wedge \neg g)$  by (rule SChopAndNotSChopImp)
have 4:  $\vdash h \frown (f \wedge \neg g) \longrightarrow \text{more} \frown (f \wedge \neg g)$  using 2 by (rule LeftSChopImpSChop)
have 5:  $\vdash f \wedge \neg g \longrightarrow \text{more} \frown (f \wedge \neg g)$  using 1 3 4 by fastforce
from 5 show ?thesis using MoreSChopContra by blast
qed

```

2.4.7 Properties of Halt

```

lemma HaltSChopEqv:
   $\vdash ((\text{halt } (\text{init } w)) \frown f) = (\text{if}_i \text{ (init } w) \text{ then } (f) \text{ else } (\bigcirc (\text{halt } (\text{init } w)) \frown f))$ 

```

proof –

have 1: $\vdash \text{halt}(\text{init } w) =$
 $(\text{if}_i (\text{init } w) \text{ then } \text{empty} \text{ else } (\bigcirc(\text{halt} (\text{init } w))))$
by (rule *HaltStateEqvIfStateThenEmptyElseNext*)
hence 2: $\vdash ((\text{halt}(\text{init } w)) \frown f) =$
 $(\text{if}_i (\text{init } w) \text{ then } (\text{empty} \frown f) \text{ else } (\bigcirc(\text{halt} (\text{init } w)) \frown f))$
by (rule *IfSCHopEqvRule*)
have 3: $\vdash \text{empty} \frown f = f$
by (rule *EmptySCHop*)
have 4: $\vdash (\bigcirc(\text{halt} (\text{init } w))) \frown f = \bigcirc(\text{halt} (\text{init } w) \frown f)$
by (rule *NextSCHop*)
from 2 3 4 **show** ?thesis **by** (metis *inteq-reflection*)

qed

lemma *AndHaltSCHopImp*:

$\vdash \text{init } w \wedge (\text{halt} (\text{init } w) \frown f) \longrightarrow f$

proof –

have 1: $\vdash \text{halt} (\text{init } w) \frown f = \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt} (\text{init } w) \frown f))$
by (rule *HaltSCHopEqv*)
have 2: $\vdash \text{init } w \wedge \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt} (\text{init } w) \frown f)) \longrightarrow f$
by (auto simp: *ifthenelse-d-def*)
from 1 2 **show** ?thesis **by** fastforce

qed

lemma *NotAndHaltSCHopImpNext*:

$\vdash \neg (\text{init } w) \wedge (\text{halt} (\text{init } w) \frown f) \longrightarrow \bigcirc(\text{halt} (\text{init } w) \frown f)$

proof –

have 1: $\vdash \text{halt} (\text{init } w) \frown f = \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt} (\text{init } w) \frown f))$
by (rule *HaltSCHopEqv*)
have 2: $\vdash \neg (\text{init } w) \wedge \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt} (\text{init } w) \frown f)) \longrightarrow$
 $\bigcirc(\text{halt} (\text{init } w) \frown f)$
by (auto simp: *ifthenelse-d-def*)
from 1 2 **show** ?thesis **by** fastforce

qed

lemma *NotAndHaltSCHopImpSkipSYields*:

$\vdash \neg (\text{init } w) \wedge (\text{halt} (\text{init } w) \frown f) \longrightarrow \text{skip } \text{syields} (\text{halt} (\text{init } w) \frown f)$

proof –

have 1: $\vdash \neg (\text{init } w) \wedge (\text{halt} (\text{init } w) \frown f) \longrightarrow \bigcirc(\text{halt} (\text{init } w) \frown f)$
by (rule *NotAndHaltSCHopImpNext*)
have 2: $\vdash \bigcirc(\text{halt} (\text{init } w) \frown f) \longrightarrow \text{skip } \text{syields} (\text{halt} (\text{init } w) \frown f)$
by (rule *NextImpSkipSYields*)
from 1 2 **show** ?thesis **by** fastforce

qed

lemma *SCHopAndEmptyEqvSCHopAndEmpty*:

$\vdash ((\# \text{True} \frown (f \wedge \text{empty})) \wedge g) = (g \frown (f \wedge \text{empty}))$

proof –

have 1: $\vdash (\# \text{True} \frown (f \wedge \text{empty})) \wedge g \longrightarrow g \frown (f \wedge \text{empty})$
by (simp add: *FiniteCHopAndEmptyEqvCHopAndEmpty int-iffD1 schop-d-def*)

have 2: $\vdash g \frown (f \wedge \text{empty}) \longrightarrow (\# \text{True} \frown (f \wedge \text{empty})) \wedge g$
by (*metis AndSFinEqvSChopAndEmpty Prop12 SFinEqvTrueSChopAndEmpty int-iffD1 inteq-reflection*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *NotSChopSkipEqvFmoreAndNotSChopSkip*:

$\vdash (\neg f) \frown \text{skip} = (f\text{more} \wedge \neg(f \frown \text{skip}))$

proof –

have 1: $\vdash (\neg f) \frown \text{skip} = ((\neg f \wedge \text{finite}); \text{skip})$

by (*simp add: schop-d-def*)

have 2: $\vdash (\neg f \wedge \text{finite}); \text{skip} = (\neg(f \vee \text{inf})); \text{skip}$

by (*metis (no-types, lifting) LeftChopEqvChop finite-d-def int-simps(14) int-simps(33) inteq-reflection*)

have 3: $\vdash (\neg(f \vee \text{inf})); \text{skip} = (\text{more} \wedge \neg((f \vee \text{inf}); \text{skip}))$

using *NotChopSkipEqvMoreAndNotChopSkip* **by** blast

have 4: $\vdash (f \vee \text{inf}); \text{skip} = (f; \text{skip} \vee \text{inf})$

by (*metis AndInfChopEqvAndInf MoreAndInfEqvInf OrChopEqv inteq-reflection*)

have 5: $\vdash (\text{more} \wedge \neg((f \vee \text{inf}); \text{skip})) = (\text{more} \wedge \neg(f; \text{skip} \vee \text{inf}))$

using 4 **by** auto

have 6: $\vdash (\text{more} \wedge \neg(f; \text{skip} \vee \text{inf})) = (\text{more} \wedge \neg(f; \text{skip}) \wedge \text{finite})$

unfolding *finite-d-def* **by** fastforce

have 7: $\vdash (\text{more} \wedge \neg(f; \text{skip}) \wedge \text{finite}) = (\text{more} \wedge \neg(f \frown \text{skip} \vee (f \wedge \text{inf})) \wedge \text{finite})$

by (*metis ChopEmpty ChopSChopdef inteq-reflection*)

have 8: $\vdash (\text{more} \wedge \neg(f \frown \text{skip} \vee (f \wedge \text{inf})) \wedge \text{finite}) =$
 $(\text{more} \wedge \neg(f \frown \text{skip}) \wedge \neg(f \wedge \text{inf}) \wedge \text{finite})$

by auto

have 9: $\vdash (\neg(f \wedge \text{inf}) \wedge \text{finite}) = \text{finite}$

unfolding *finite-d-def* **by** force

have 10: $\vdash (\text{more} \wedge \neg(f \frown \text{skip}) \wedge \neg(f \wedge \text{inf}) \wedge \text{finite}) =$
 $(\text{more} \wedge \neg(f \frown \text{skip}) \wedge \text{finite})$

using 9 **by** fastforce

have 11: $\vdash (\text{more} \wedge \neg(f \frown \text{skip}) \wedge \text{finite}) = (f\text{more} \wedge \neg(f \frown \text{skip}))$

using *fmore-d-def* **by** (*metis Prop11 Prop12 lift-and-com*)

from 1 2 3 5 6 7 8 10 11 **show** ?thesis **by** (*metis inteq-reflection*)

qed

lemma *HaltSChopImpNotHaltSChopNot*:

$\vdash \text{halt } (\text{init } w) \frown f \wedge \text{finite} \longrightarrow \neg (\text{halt } (\text{init } w) \frown (\neg f))$

proof –

have 1: $\vdash \text{halt } (\text{init } w) \frown f = \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc (\text{halt } (\text{init } w) \frown f))$

by (*rule HaltSChopEqv*)

have 2: $\vdash \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc (\text{halt } (\text{init } w) \frown f)) \longrightarrow$
 $((\text{init } w) \longrightarrow f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc (\text{halt } (\text{init } w) \frown f)))$

by (*rule IfThenElseImp*)

have 3: $\vdash \text{halt } (\text{init } w) \frown (\neg f) =$
 $\text{if}_i (\text{init } w) \text{ then } (\neg f) \text{ else } (\bigcirc (\text{halt } (\text{init } w) \frown (\neg f)))$

by (*rule HaltSChopEqv*)

have 4: $\vdash \text{if}_i (\text{init } w) \text{ then } (\neg f) \text{ else } (\bigcirc (\text{halt } (\text{init } w) \frown (\neg f))) \longrightarrow$
 $((\text{init } w) \longrightarrow \neg f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc (\text{halt } (\text{init } w) \frown (\neg f))))$

by (*rule IfThenElseImp*)

have 5: $\vdash \text{halt } (init\ w) \frown f \wedge \text{halt } (init\ w) \frown (\neg f) \longrightarrow$
 $(((init\ w) \longrightarrow f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(\text{halt } (init\ w) \frown f)))) \wedge$
 $(((init\ w) \longrightarrow \neg f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(\text{halt } (init\ w) \frown (\neg f))))))$
using 1 2 3 4 **by** *fastforce*
have 6: $\vdash (((init\ w) \longrightarrow f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(\text{halt } (init\ w) \frown f)))) \wedge$
 $(((init\ w) \longrightarrow \neg f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(\text{halt } (init\ w) \frown (\neg f)))))) \longrightarrow$
 $(\bigcirc(\text{halt } (init\ w) \frown f)) \wedge (\bigcirc(\text{halt } (init\ w) \frown (\neg f)))$
by *auto*
have 7: $\vdash \text{halt } (init\ w) \frown f \wedge \text{halt } (init\ w) \frown (\neg f) \longrightarrow$
 $(\bigcirc(\text{halt } (init\ w) \frown f)) \wedge (\bigcirc(\text{halt } (init\ w) \frown (\neg f)))$
using 5 6 *lift-imp-trans* **by** *blast*
have 8: $\vdash ((\bigcirc(\text{halt } (init\ w) \frown f)) \wedge (\bigcirc(\text{halt } (init\ w) \frown (\neg f)))) =$
 $\bigcirc(\text{halt } (init\ w) \frown f \wedge \text{halt } (init\ w) \frown (\neg f))$
using *NextAndEqvNextAndNext* **by** *fastforce*
have 9: $\vdash \text{halt } (init\ w) \frown f \wedge \text{halt } (init\ w) \frown (\neg f) \longrightarrow$
 $\bigcirc(\text{halt } (init\ w) \frown f \wedge \text{halt } (init\ w) \frown (\neg f))$
using 7 8 **by** *fastforce*
hence 10: $\vdash \text{finite} \longrightarrow \neg(\text{halt } (init\ w) \frown f \wedge \text{halt } (init\ w) \frown (\neg f))$
using *NextLoop* **by** *blast*
from 10 **show** ?thesis **by** *auto*
qed

lemma *HaltSCHopImpHaltSYields*:

$\vdash \text{halt } (init\ w) \frown f \wedge \text{finite} \longrightarrow (\text{halt } (init\ w)) \text{ syields } f$

proof –

have 1: $\vdash \text{halt } (init\ w) \frown f \wedge \text{finite} \longrightarrow \neg(\text{halt } (init\ w) \frown (\neg f))$
by (*rule HaltSCHopImpNotHaltSCHopNot*)
from 1 **show** ?thesis **by** (*simp add: syields-d-def*)
qed

lemma *HaltSCHopAnd*:

$\vdash (\text{halt } (init\ w) \frown f \wedge (\text{halt } (init\ w) \frown g) \wedge \text{finite}) \longrightarrow (\text{halt } (init\ w) \frown (f \wedge g))$

proof –

have 1: $\vdash (\text{halt } (init\ w) \frown g) \wedge \text{finite} \longrightarrow (\text{halt } (init\ w)) \text{ syields } g$
by (*rule HaltSCHopImpHaltSYields*)
hence 2: $\vdash (\text{halt } (init\ w) \frown f \wedge (\text{halt } (init\ w) \frown g) \wedge \text{finite}) \longrightarrow$
 $(\text{halt } (init\ w) \frown f \wedge (\text{halt } (init\ w) \text{ syields } g))$
by *auto*
have 3: $\vdash (\text{halt } (init\ w) \frown f \wedge (\text{halt } (init\ w) \text{ syields } g)) \longrightarrow$
 $(\text{halt } (init\ w) \frown (f \wedge g))$
by (*rule SCHopAndSYieldsImp*)
from 2 3 **show** ?thesis **by** *fastforce*
qed

lemma *HaltAndSCHopAndHaltSCHopImpHaltAndSCHopAnd*:

$\vdash (\text{halt } (init\ w) \frown f) \frown f1 \wedge (\text{halt } (init\ w) \frown g) \wedge \text{finite}$
 $\longrightarrow (\text{halt } (init\ w) \frown f) \frown (f1 \wedge g)$

proof –

have 1: $\vdash f1 \longrightarrow \neg g \vee (f1 \wedge g)$
by *auto*

hence 2: $\vdash (\text{halt } (\text{init } w) \wedge f) \frown f1 \longrightarrow$
 $(\text{halt } (\text{init } w) \wedge f) \frown (\neg g) \vee ((\text{halt } (\text{init } w) \wedge f) \frown (f1 \wedge g))$
by (rule *SChopOrImpRule*)
have 3: $\vdash (\text{halt } (\text{init } w) \wedge f) \frown (\neg g) \longrightarrow \text{halt } (\text{init } w) \frown (\neg g)$
by (rule *AndSChopA*)
have 31: $\vdash (\text{halt } (\text{init } w) \wedge f) \frown f1 \longrightarrow$
 $\text{halt } (\text{init } w) \frown (\neg g) \vee ((\text{halt } (\text{init } w) \wedge f) \frown (f1 \wedge g))$
using 23 **by** *fastforce*
have 4: $\vdash \text{halt } (\text{init } w) \frown g \wedge \text{finite} \longrightarrow \neg (\text{halt } (\text{init } w) \frown (\neg g))$
by (rule *HaltSChopImpNotHaltSChopNot*)
hence 41: $\vdash (\text{halt } (\text{init } w) \frown (\neg g)) \wedge \text{finite} \longrightarrow \neg(\text{halt } (\text{init } w) \frown g)$
by *auto*
have 42: $\vdash (\text{halt } (\text{init } w) \wedge f) \frown f1 \wedge \text{finite} \longrightarrow$
 $\neg(\text{halt } (\text{init } w) \frown g) \vee ((\text{halt } (\text{init } w) \wedge f) \frown (f1 \wedge g))$
using 31 41 **by** *fastforce*
from 42 **show** ?thesis **by** *auto*
qed

lemma *HaltImpBoxSYields*:

$\vdash (\text{halt } (\text{init } w)) \frown f \wedge \text{finite} \longrightarrow (\Box(\neg (\text{init } w))) \text{ syields } ((\text{halt } (\text{init } w)) \frown f)$

proof –

have 1: $\vdash (\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)) \longrightarrow \text{df } (\Box(\neg (\text{init } w)))$
by (rule *SChopImpDf*)
have 2: $\vdash \Box(\neg (\text{init } w)) \longrightarrow \neg (\text{init } w)$
by (rule *BoxElim*)
hence 3: $\vdash \text{df } (\Box(\neg (\text{init } w))) \longrightarrow \text{df } (\neg (\text{init } w))$
by (rule *DfImpDf*)
have 4: $\vdash \text{df } (\text{init } (\neg w)) = (\text{init } (\neg w))$
by (rule *DfState*)
have 41: $\vdash (\text{init } (\neg w)) = (\neg (\text{init } w))$
using *Initprop(2)* **by** *fastforce*
have 42: $\vdash \text{df } (\neg (\text{init } w)) = (\neg(\text{init } w))$
using 4 41 **by** (*metis integ-reflection*)
have 5: $\vdash ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))) \longrightarrow \neg (\text{init } w)$
using 1 2 42 **using** 3 **by** *fastforce*
hence 51: $\vdash (\text{halt } (\text{init } w) \frown f) \wedge ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))) \longrightarrow$
 $(\text{halt } (\text{init } w) \frown f) \wedge \neg (\text{init } w)$
by *fastforce*
have 6: $\vdash \text{halt } (\text{init } w) \frown f = \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\Box(\text{halt } (\text{init } w) \frown f))$
by (rule *HaltSChopEqv*)
hence 61: $\vdash (\text{halt } (\text{init } w) \frown f \wedge \neg (\text{init } w)) =$
 $((\text{if}_i (\text{init } w) \text{ then } f \text{ else } (\Box(\text{halt } (\text{init } w) \frown f))) \wedge \neg (\text{init } w))$
using 6 **by** *auto*
have 62: $\vdash (\text{if}_i (\text{init } w) \text{ then } f \text{ else } (\Box(\text{halt } (\text{init } w) \frown f))) \wedge$
 $\neg (\text{init } w) \longrightarrow (\Box(\text{halt } (\text{init } w) \frown f))$
by (*auto simp: ifthenelse-d-def*)
have 63: $\vdash \text{halt } (\text{init } w) \frown f \wedge \neg (\text{init } w) \longrightarrow (\Box(\text{halt } (\text{init } w) \frown f))$
using 61 62 **by** *fastforce*
have 7: $\vdash (\text{halt } (\text{init } w) \frown f) \wedge (\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)) \longrightarrow$
 $\Box((\text{halt } (\text{init } w) \frown f))$

using 51 63 using lift-imp-trans by blast
 have 8: $\vdash \Box (\neg (\text{init } w)) \longrightarrow \text{empty} \vee \bigcirc(\Box(\neg (\text{init } w)))$
 by (metis BoxImpYields WeakNextBoxImpMoreYields WnextEqvEmptyOrNext fmore-d-def int-eq)
 hence 9: $\vdash ((\Box (\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))) \longrightarrow$
 $\neg (\text{halt } (\text{init } w) \frown f) \vee \bigcirc((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)))$
 by (rule EmptyOrNextSChopImpRule)
 hence 10: $\vdash ((\text{halt } (\text{init } w) \frown f) \wedge (\Box (\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))) \longrightarrow$
 $\bigcirc((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)))$
 by fastforce
 have 11: $\vdash (\text{halt } (\text{init } w) \frown f \wedge (\Box (\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))) \longrightarrow$
 $\bigcirc((\text{halt } (\text{init } w) \frown f) \wedge \bigcirc((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))))$
 using 7 10 by fastforce
 have 12: $\vdash \bigcirc((\text{halt } (\text{init } w) \frown f) \wedge \bigcirc((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))))$
 $\longrightarrow \bigcirc(((\text{halt } (\text{init } w) \frown f) \wedge ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))))$
 using NextAndEqvNextAndNext by fastforce
 have 13: $\vdash (\text{halt } (\text{init } w) \frown f \wedge (\Box (\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))) \longrightarrow$
 $\bigcirc(((\text{halt } (\text{init } w) \frown f) \wedge ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))))$
 using 11 12 by fastforce
 hence 14: $\vdash \text{finite} \longrightarrow \neg ((\text{halt } (\text{init } w) \frown f \wedge (\Box (\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))))$
 using NextLoop by blast
 hence 15: $\vdash (\text{halt } (\text{init } w) \frown f \wedge \text{finite} \longrightarrow \neg ((\Box (\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))))$
 by auto
 from 15 show ?thesis by (simp add: syields-d-def)
 qed

2.4.8 Properties of Groups of strong chops

lemma NestedSChopImpSChop:

assumes $\vdash \text{init } w \wedge f \longrightarrow g \frown (\text{init } w1 \wedge f1)$
 $\vdash \text{init } w1 \wedge f1 \longrightarrow g1 \frown (\text{init } w2 \wedge f2)$
 shows $\vdash \text{init } w \wedge f \longrightarrow g \frown (g1 \frown (\text{init } w2 \wedge f2))$
 proof –
 have 1: $\vdash \text{init } w \wedge f \longrightarrow g \frown (\text{init } w1 \wedge f1)$ using assms(1) by auto
 have 2: $\vdash \text{init } w1 \wedge f1 \longrightarrow g1 \frown (\text{init } w2 \wedge f2)$ using assms(2) by auto
 hence 3: $\vdash g \frown (\text{init } w1 \wedge f1) \longrightarrow g \frown (g1 \frown (\text{init } w2 \wedge f2))$ by (rule RightSChopImpSChop)
 from 1 3 show ?thesis by fastforce
 qed

end

2.5 Chopstar and variants

theory Chopstar
 imports
 SChopTheorems
 begin

This theory defines the chopstar operator for infinite ITL and provides a library of lemmas. We also define the strong version schopstar, the weak version wpowerstar and a semantic version achopstar. The wpowerstar corresponds to the Kleene star operator from Kleene Algebra [1]. We provide lemmas that express various relationships between them. We also ported the numerous Kleene algebra lemmas from [1] to ITL.

2.5.1 Definitions

primrec *wpower-d* :: ('a::world) formula \Rightarrow nat \Rightarrow 'a formula
where *wpow-0* : (*wpower-d* *F* 0) = *LIFT*(*empty*)
| *wpow-Suc*: (*wpower-d* *F* (*Suc* *n*)) = *LIFT*((*F*);(*wpower-d* *F* *n*))

syntax

-*wpower-d* :: [*lift*,*nat*] \Rightarrow *lift* ((*wpower* - -) [88,88] 87)

syntax (*ASCII*)

-*wpower-d* :: [*lift*,*nat*] \Rightarrow *lift* ((*wpower* - -) [88,88] 87)

translations

-*wpower-d* \Rightarrow *CONST* *wpower-d*

definition *fpower-d* :: ('a::world) formula \Rightarrow nat \Rightarrow 'a formula

where

fpower-d *F* *n* \equiv *LIFT*(*wpower* (*F* \wedge *finite*) *n*)

syntax

-*fpower-d* :: [*lift*,*nat*] \Rightarrow *lift* ((*fpower* - -) [88,88] 87)

syntax (*ASCII*)

-*fpower-d* :: [*lift*,*nat*] \Rightarrow *lift* ((*fpower* - -) [88,88] 87)

translations

-*fpower-d* \Rightarrow *CONST* *fpower-d*

definition *len-d* :: nat \Rightarrow ('a::world) formula

where *len-d* *n* \equiv *LIFT*(*wpower* *skip* *n*)

definition *wpowerstar-d* :: ('a::world) formula \Rightarrow 'a formula

where *wpowerstar-d* *F* \equiv *LIFT*((\exists *k*. *wpower* *F* *k*))

definition *fpowerstar-d* :: ('a::world) formula \Rightarrow 'a formula

where *fpowerstar-d* *F* \equiv *LIFT*(\exists *k*. *fpower* *F* *k*)

syntax

-*len-d* :: nat \Rightarrow *lift* ((*len* -) [88] 87)

-*wpowerstar-d* :: *lift* \Rightarrow *lift* ((*wpowerstar* -) [85] 85)

-*fpowerstar-d* :: *lift* \Rightarrow *lift* ((*fpowerstar* -) [85] 85)

syntax (*ASCII*)

$-len-d \quad :: nat \Rightarrow lift \quad ((len -) [88] 87)$
 $-wpowerstar-d \quad :: lift \Rightarrow lift \quad ((wpowerstar -) [85] 85)$
 $-fpowerstar-d \quad :: lift \Rightarrow lift \quad ((fpowerstar -) [85] 85)$

translations

$-len-d \quad \Rightarrow CONST len-d$
 $-wpowerstar-d \quad \Rightarrow CONST wpowerstar-d$
 $-fpowerstar-d \quad \Rightarrow CONST fpowerstar-d$

definition $powerstar-d :: ('a::world) formula \Rightarrow 'a formula$

where $powerstar-d F \equiv LIFT(fpowerstar F; (empty \vee (F \wedge inf)))$

syntax

$-powerstar-d \quad :: lift \Rightarrow lift \quad ((powerstar -) [85] 85)$

syntax (*ASCII*)

$-powerstar-d \quad :: lift \Rightarrow lift \quad ((powerstar -) [85] 85)$

translations

$-powerstar-d \quad \Rightarrow CONST powerstar-d$

definition $chopstar-d :: ('a::world) formula \Rightarrow 'a formula$

where $chopstar-d F \equiv LIFT(powerstar (F \wedge more))$

definition $schopstar-d :: ('a::world) formula \Rightarrow 'a formula$

where $schopstar-d F \equiv LIFT(fpowerstar (F \wedge more))$

syntax

$-chopstar-d \quad :: lift \Rightarrow lift \quad ((-^*) [85] 85)$
 $-schopstar-d \quad :: lift \Rightarrow lift \quad ((schopstar -) [85] 85)$

syntax (*ASCII*)

$-chopstar-d \quad :: lift \Rightarrow lift \quad ((chopstar -) [85] 85)$
 $-schopstar-d \quad :: lift \Rightarrow lift \quad ((schopstar -) [85] 85)$

translations

$-chopstar-d \quad \Rightarrow CONST chopstar-d$
 $-schopstar-d \quad \Rightarrow CONST schopstar-d$

definition $while-d :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a formula$

where $while\text{-}d\ F\ G \equiv LIFT((F \wedge G)^* \wedge (fin\ ((\neg F))))$

syntax

$\text{-}while\text{-}d :: [lift, lift] \Rightarrow lift\ ((while\ \text{-}\ do\ \text{-}\)\ [88,88]\ 87)$

syntax (*ASCII*)

$\text{-}while\text{-}d :: [lift, lift] \Rightarrow lift\ ((while\ \text{-}\ do\ \text{-}\)\ [88,88]\ 87)$

translations

$\text{-}while\text{-}d \Rightarrow CONST\ while\text{-}d$

definition $swhile\text{-}d :: ('a::world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$

where $swhile\text{-}d\ F\ G \equiv LIFT(schopstar(F \wedge G) \wedge (sfin\ ((\neg F))))$

definition $repeat\text{-}d :: ('a::world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$

where $repeat\text{-}d\ F\ G \equiv LIFT(F;while\ (\neg G)\ do\ F)$

syntax

$\text{-}swhile\text{-}d :: [lift, lift] \Rightarrow lift\ ((swhile\ \text{-}\ do\ \text{-}\)\ [88,88]\ 87)$

$\text{-}repeat\text{-}d :: [lift, lift] \Rightarrow lift\ ((repeat\ \text{-}\ until\ \text{-}\)\ [88,88]\ 87)$

syntax (*ASCII*)

$\text{-}swhile\text{-}d :: [lift, lift] \Rightarrow lift\ ((swhile\ \text{-}\ do\ \text{-}\)\ [88,88]\ 87)$

$\text{-}repeat\text{-}d :: [lift, lift] \Rightarrow lift\ ((repeat\ \text{-}\ until\ \text{-}\)\ [88,88]\ 87)$

translations

$\text{-}swhile\text{-}d \Rightarrow CONST\ swhile\text{-}d$

$\text{-}repeat\text{-}d \Rightarrow CONST\ repeat\text{-}d$

definition $srepeat\text{-}d :: ('a::world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$

where $srepeat\text{-}d\ F\ G \equiv LIFT(F \frown swhile\ (\neg G)\ do\ F)$

syntax

$\text{-}srepeat\text{-}d :: [lift, lift] \Rightarrow lift\ ((srepeat\ \text{-}\ until\ \text{-}\)\ [88,88]\ 87)$

syntax (*ASCII*)

$\text{-}srepeat\text{-}d :: [lift, lift] \Rightarrow lift\ ((srepeat\ \text{-}\ until\ \text{-}\)\ [88,88]\ 87)$

translations

$\text{-}srepeat\text{-}d \Rightarrow CONST\ srepeat\text{-}d$

definition $aschopstar\text{-}d :: ('a::world)\ formula \Rightarrow 'a\ formula$

where $aschopstar\text{-}d\ F \equiv$

$\lambda s. (\exists\ (l::nat\ nellist).$

$(nnth\ l\ 0) = 0 \wedge nfinite\ l \wedge nidx\ l \wedge$

$(enat\ (nlast\ l)) = (nlength\ s) \wedge nfinite\ s \wedge$

$(\forall\ (i::nat) . (enat\ i) < (nlength\ l)) \longrightarrow$

$((nsbn\ s\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models F))$

)

syntax

$-aschopstar-d \quad :: lift \Rightarrow lift \quad ((aschopstar -) [85] 85)$

syntax (ASCII)

$-aschopstar-d \quad :: lift \Rightarrow lift \quad ((aschopstar -) [85] 85)$

translations

$-aschopstar-d \quad \Rightarrow CONST aschopstar-d$

lemma FChopSem-var [mono]:

$(w \models f;g) =$
 $((\exists n. enat\ n \leq nlength\ w \wedge f\ ((ntaken\ n\ w)) \wedge g\ (ndropn\ n\ w)) \vee$
 $(\neg nfinite\ w \wedge f\ w)) \vee$

by (*simp add: itl-defs*)

inductive istar-d :: ('a::world) formula \Rightarrow 'a formula

for F where

$(s \models empty) \Longrightarrow (s \models (istar-d\ F))$
 $| (s \models F; (istar-d\ F)) \Longrightarrow (s \models (istar-d\ F))$

syntax

$-istar-d \quad :: lift \Rightarrow lift \quad ((istar -) [85] 85)$

syntax (ASCII)

$-istar-d \quad :: lift \Rightarrow lift \quad ((istar -) [85] 85)$

translations

$-istar-d \quad \Rightarrow CONST istar-d$

2.5.2 Semantic lemmas

lemma ChopExist:

$\vdash (\exists k. f;g\ k) = f;(\exists k. g\ k)$

by (*auto simp add: itl-defs Valid-def*)

lemma SChopExist:

$\vdash (\exists k. f \frown g\ k) = f \frown (\exists k. g\ k)$

by (*auto simp add: itl-defs Valid-def*)

lemma ExistChop:

$\vdash (\exists k. (g\ k);f) = (\exists k. g\ k);f$

by (*auto simp add: itl-defs Valid-def*)

lemma ExistSChop:

$\vdash (\exists k. (g\ k) \frown f) = (\exists k. g\ k) \frown f$

by (*auto simp add: itl-defs Valid-def*)

lemma *wpowersem1*:

$(\sigma \models (\exists k. \text{wpower } f \ k) = (\text{empty} \vee (\exists k. \text{wpower } f \ (\text{Suc } k))))$

proof (*auto*)

show $\bigwedge x. \sigma \models (\text{wpower } f \ x) \implies \forall k. \neg (\sigma \models f; \text{wpower } f \ k) \implies \sigma \models \text{empty}$

by (*metis not0-implies-Suc wpow-0 wpow-Suc*)

show $\sigma \models \text{empty} \implies \exists x. \sigma \models (\text{wpower } f \ x)$

by (*metis wpow-0*)

show $\bigwedge k. \sigma \models (f; \text{wpower } f \ k) \implies \exists x. \sigma \models (\text{wpower } f \ x)$

by (*metis wpow-Suc*)

qed

lemma *fpowersem1*:

$(\sigma \models (\exists k. \text{fpower } f \ k) = (\text{empty} \vee (\exists k. \text{fpower } f \ (\text{Suc } k))))$

unfolding *fpower-d-def* **using** *wpowersem1*[*of LIFT (f ∧ finite) σ*]

by *blast*

lemma *wpowersem*:

$\vdash (\exists k. \text{wpower } f \ k) = (\text{empty} \vee f; (\exists k. (\text{wpower } f \ k)))$

proof –

have 1: $\vdash (\exists k. \text{wpower } f \ k) = (\text{empty} \vee (\exists k. \text{wpower } f \ (\text{Suc } k)))$

using *wpowersem1* **by** *blast*

have 2: $\vdash (\exists k. \text{wpower } f \ (\text{Suc } k)) = (\exists k. f; \text{wpower } f \ k)$

by *simp*

have 3: $\vdash (\exists k. f; (\text{wpower } f \ k)) = f; (\exists k. (\text{wpower } f \ k))$

using *ChopExist* **by** *blast*

from 1 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *fpowersem*:

$\vdash (\exists k. \text{fpower } f \ k) = (\text{empty} \vee (f \wedge \text{finite}); (\exists k. (\text{fpower } f \ k)))$

unfolding *fpower-d-def* **using** *wpowersem*[*of LIFT (f ∧ finite)*]

by *blast*

lemma *finite-nidx-bounded-nlast*:

assumes *nfinite l*

nidx l

$(\text{enat } (\text{nlast } l)) = (\text{nlength } s)$

$(\text{enat } i) \leq (\text{nlength } l)$

shows $(\text{nnth } l \ i) \leq \text{nlength } s$

using *assms*

by (*metis enat-ord-simps(1) nfinite-nlength-enat nidx-less-eq nnth-nlast*

the-enat.simps verit-comp-simplify1(2))

lemma *aschopstar-wpower-chain-a*:

assumes $(\exists (l:: \text{nat nellist}).$

$(\text{nlength } l) = (\text{Suc } n) \wedge \text{nfinite } l \wedge \text{nidx } l \wedge (\text{nnth } l \ 0) = 0 \wedge$

```

    (enat (nlast l)) = (nlength σ) ∧ nfinite σ ∧
    (∀ (i::nat). i < (nlength l) →
      ((nsubn σ (nnth l i) (nnth l (Suc i))) ⊨ f)
    )
  )
)
shows (∃ k. 0 ≤ k ∧ k ≤ nlength σ ∧ 0 < k ∧ (nsubn σ 0 k ⊨ f) ∧
  (∃ ls. nfinite ls ∧ (nlength ls) = n ∧ nidx ls ∧ (nnth ls 0) = 0 ∧
    (enat (nlast ls)) = (nlength (ndropn k σ)) ∧ nfinite(ndropn k σ) ∧
    (∀ (i::nat). i < (nlength ls) →
      ((nsubn (ndropn k σ) (nnth ls i) (nnth ls (Suc i))) ⊨ f)
    )
  ))
)
proof -
obtain l where 1: (nlength l) = (Suc n) ∧ nfinite l ∧ nidx l ∧ (nnth l 0) = 0 ∧
  (enat (nlast l)) = (nlength σ) ∧ nfinite σ ∧
  (∀ (i::nat). i < (nlength l) →
    ((nsubn σ (nnth l i) (nnth l (Suc i))) ⊨ f)
  )
)
using assms by auto
have 2: nlength l > 0
using 1 by (simp add: enat-0-iff(1))
have 3: l = NCons (nnth l 0) (ndropn 1 l)
using 2 by (simp add: ndropn-Suc-conv-ndropn zero-enat-def)
have 4: nlength (ndropn 1 l) = n
by (simp add: 1)
have 5: 0 ≤ (nnth l 0)
by simp
have 6: (nnth l 0) ≤ nlength σ
using 1 using i0-lb zero-enat-def by presburger
have 7: (nnth l 0) = 0
using 1 by blast
have 71: (nnth (ndropn 1 l) 0) = (nnth l 1)
by auto
have 8: nnth l 0 < nnth (ndropn 1 l) 0
by (metis 1 71 One-nat-def eSuc-enat enat-ord-simps(2) ileI1 nidx-gr-first zero-less-Suc)
have 9: nidx (ndropn 1 l)
by (metis 1 2 One-nat-def Suc-eq-plus1 Suc-ile-eq enat-min-eq ndropn-nlength ndropn-nnth
  nidx-expand plus-1-eq-Suc plus-enat-simps(1) zero-enat-def)
have 10: (enat (nlast (ndropn 1 l))) = (nlength σ)
using 1 3 by (metis nlast-NCons)
have 101:  $\bigwedge j. j \leq nlength (ndropn 1 l) \rightarrow$ 
   $nnth (nmap (\lambda x. x - (nnth l 1)) (ndropn 1 l)) j =$ 
   $(nnth l (Suc j)) - (nnth l 1)$ 
by simp
have 102:  $\bigwedge j. j \leq nlength (ndropn 1 l) \rightarrow$ 
   $(nnth l 1) \leq (nnth l (Suc j))$ 
by (simp add: 1 nidx-less-eq)
have 103:  $\bigwedge j. j < nlength (ndropn 1 l) \rightarrow$ 
   $(nnth l (Suc j)) - (nnth l 1) <$ 
   $(nnth l (Suc (Suc j))) - (nnth l 1)$ 

```

using 1 *nidx-expand*[of *l*]
by (*metis* 102 4 *diff-less-mono eSuc-enat ileI1 illess-Suc-eq order-less-imp-le*)
have 11: *nidx* (*nmap* ($\lambda x. x - (\text{nnth } l \ 1)$) (*ndropn* 1 *l*))
using 103 101 *nidx-expand*[of (*nmap* ($\lambda x. x - (\text{nnth } l \ 1)$) (*ndropn* 1 *l*))]
using *Suc-ile-eq* **by** *force*
have 12: *nlength* (*nmap* ($\lambda x. x - (\text{nnth } l \ 1)$) (*ndropn* 1 *l*)) = *n*
using 4 **by** *auto*
have 13: (*enat* (*nlast* (*nmap* ($\lambda x. x - (\text{nnth } l \ 1)$) (*ndropn* 1 *l*))))
=
(*nlength* (*ndropn* (*nnth* *l* 1) σ))
by (*metis* 1 10 *idiff-enat-enat ndropn-nlength nfinite-ndropn nlast-nmap*)
have 14: (*nsubn* $\sigma \ 0$ (*nnth* *l* 1) $\models f$)
by (*metis* 1 *One-nat-def enat-ord-simps*(2) *zero-less-Suc*)
have 15: ($\forall (i::\text{nat}). i < \text{nlength } (\text{ndropn } 1 \ l) \longrightarrow$
(*nsubn* σ (*nnth* (*ndropn* 1 *l*) *i*) (*nnth* (*ndropn* 1 *l*) (*Suc* *i*)) $\models f$))
using 1 3 *eSuc-enat ileI1 illess-Suc-eq ndropn-nnth nlength-NCons plus-1-eq-Suc* **by** *metis*

have 16: ($\forall (i::\text{nat}). i < \text{nlength } (\text{nmap } (\lambda x. x - (\text{nnth } l \ 1)) (\text{ndropn } 1 \ l)) \longrightarrow$
(*nsubn* σ ((*nnth* (*nmap* ($\lambda x. x - (\text{nnth } l \ 1)$) (*ndropn* 1 *l*)) *i*) + (*nnth* *l* 1))
((*nnth* (*nmap* ($\lambda x. x - (\text{nnth } l \ 1)$) (*ndropn* 1 *l*)) (*Suc* *i*)) + (*nnth* *l* 1)) $\models f$))
using 102 12 15 4 **by** *force*
have 17: ($\forall (i::\text{nat}). i < \text{nlength } (\text{nmap } (\lambda x. x - (\text{nnth } l \ 1)) (\text{ndropn } 1 \ l)) \longrightarrow$
((*nsubn* (*ndropn* (*nnth* *l* 1) σ)
(*nnth* (*nmap* ($\lambda x. x - (\text{nnth } l \ 1)$) (*ndropn* 1 *l*)) *i*)
(*nnth* (*nmap* ($\lambda x. x - (\text{nnth } l \ 1)$) (*ndropn* 1 *l*)) (*Suc* *i*))) $\models f$))
by (*metis* 11 16 *eSuc-enat ileI1 nidx-expand nsubn-ndropn*)
have 18: *nfinite* (*nmap* ($\lambda x. x - (\text{nnth } l \ 1)$) (*ndropn* 1 *l*))
using 12 *nlength-eq-enat-nfiniteD* **by** *blast*
have 19: *nfinite* (*ndropn* (*nnth* *l* 1) σ)
using 1 *nfinite-ndropn-a* **by** *blast*
have 20: ($\exists \text{ ls. } n\text{finite } \text{ls} \wedge (\text{nlength } \text{ls}) = n \wedge \text{nidx } \text{ls} \wedge (\text{nnth } \text{ls } 0) = 0 \wedge$
(*enat* (*nlast* *ls*)) = (*nlength* (*ndropn* (*nnth* *l* 1) σ)) \wedge *nfinite*(*ndropn* (*nnth* *l* 1) σ) \wedge
($\forall (i::\text{nat}). i < (\text{nlength } \text{ls}) \longrightarrow$
((*nsubn* (*ndropn* (*nnth* *l* 1) σ) (*nnth* *ls* *i*) (*nnth* *ls* (*Suc* *i*))) $\models f$)
))
by (*metis* 11 12 13 17 18 19 71 *diff-self-eq-0 enat-le-plus-same*(1) *gen-nlength-def*
nlength-code nnth-nmap)
show ?thesis
by (*metis* 1 2 20 71 8 *One-nat-def Suc-ile-eq finite-nidx-bounded-nlast zero-enat-def*
zero-order(1))
qed

lemma *aschopstar-wpower-chain-b*:

assumes ($\exists k. 0 \leq k \wedge k \leq \text{nlength } \sigma \wedge 0 < k \wedge$
(*nsubn* $\sigma \ 0 \ k \models f$) \wedge
($\exists \text{ ls. } n\text{finite } \text{ls} \wedge (\text{nlength } \text{ls}) = n \wedge \text{nidx } \text{ls} \wedge (\text{nnth } \text{ls } 0) = 0 \wedge$
(*enat* (*nlast* *ls*)) = (*nlength* (*ndropn* *k* σ)) \wedge *nfinite*(*ndropn* *k* σ) \wedge
($\forall (i::\text{nat}). i < (\text{nlength } \text{ls}) \longrightarrow$
((*nsubn* (*ndropn* *k* σ) (*nnth* *ls* *i*) (*nnth* *ls* (*Suc* *i*))) $\models f$)
))

$$\text{shows } (\exists (l :: \text{nat } \text{nellist}).$$

$$\begin{aligned} & (\text{nlength } l) = (\text{Suc } n) \wedge \text{nfinite } l \wedge \text{nidx } l \wedge (\text{nnth } l \ 0) = 0 \wedge \\ & (\text{enat } (\text{nlast } l)) = (\text{nlength } \sigma) \wedge \text{nfinite } \sigma \wedge \\ & (\forall (i :: \text{nat}). i < (\text{nlength } l) \longrightarrow \\ & \quad ((\text{nsubn } \sigma (\text{nnth } l \ i) (\text{nnth } l (\text{Suc } i))) \models f) \\ &) \\ &) \end{aligned}$$

proof –

obtain k **where** $1: 0 \leq k \wedge k \leq \text{nlength } \sigma \wedge k > 0 \wedge$
 $(\text{nsubn } \sigma \ 0 \ k \models f) \wedge$
 $(\exists \text{ ls. } \text{nfinite } \text{ls} \wedge (\text{nlength } \text{ls}) = n \wedge \text{nidx } \text{ls} \wedge (\text{nnth } \text{ls} \ 0) = 0 \wedge$
 $(\text{enat } (\text{nlast } \text{ls})) = (\text{nlength } (\text{ndropn } k \ \sigma)) \wedge \text{nfinite}(\text{ndropn } k \ \sigma) \wedge$
 $(\forall (i :: \text{nat}). i < (\text{nlength } \text{ls}) \longrightarrow$
 $\quad ((\text{nsubn } (\text{ndropn } k \ \sigma) (\text{nnth } \text{ls} \ i) (\text{nnth } \text{ls} (\text{Suc } i))) \models f)$
 $\quad))$

using *assms* **by** *auto*

have $2: (\exists \text{ ls. } \text{nfinite } \text{ls} \wedge (\text{nlength } \text{ls}) = n \wedge \text{nidx } \text{ls} \wedge (\text{nnth } \text{ls} \ 0) = 0 \wedge$
 $(\text{enat } (\text{nlast } \text{ls})) = (\text{nlength } (\text{ndropn } k \ \sigma)) \wedge \text{nfinite}(\text{ndropn } k \ \sigma) \wedge$
 $(\forall (i :: \text{nat}). i < (\text{nlength } \text{ls}) \longrightarrow$
 $\quad ((\text{nsubn } (\text{ndropn } k \ \sigma) (\text{nnth } \text{ls} \ i) (\text{nnth } \text{ls} (\text{Suc } i))) \models f)$
 $\quad))$

using 1 **by** *auto*

obtain ls **where** $3: \text{nfinite } \text{ls} \wedge (\text{nlength } \text{ls}) = n \wedge \text{nidx } \text{ls} \wedge (\text{nnth } \text{ls} \ 0) = 0 \wedge$
 $(\text{enat } (\text{nlast } \text{ls})) = (\text{nlength } (\text{ndropn } k \ \sigma)) \wedge \text{nfinite}(\text{ndropn } k \ \sigma) \wedge$
 $(\forall (i :: \text{nat}). i < (\text{nlength } \text{ls}) \longrightarrow$
 $\quad ((\text{nsubn } (\text{ndropn } k \ \sigma) (\text{nnth } \text{ls} \ i) (\text{nnth } \text{ls} (\text{Suc } i))) \models f)$
 $\quad))$

using 2 **by** *auto*

have $4: \text{nidx } (\text{nmap } (\lambda x. x + k) \ \text{ls})$
using 3
by (*simp add: nidx-expand*)

have $41: \bigwedge j. j \leq \text{nlength } (\text{nmap } (\lambda x. x + k) \ \text{ls}) \longrightarrow$
 $0 < (\text{nnth } (\text{nmap } (\lambda x. x + k) \ \text{ls}) \ j)$
by (*simp add: 1*)

have $5: \text{nidx } (\text{NCons } 0 \ (\text{nmap } (\lambda x. x + k) \ \text{ls}))$

using $3 \ 4$ *nidx-expand*[*of* ls] *nidx-expand*[*of* $(\text{nmap } (\lambda x. x + k) \ \text{ls})$]
nidx-expand[*of* $(\text{NCons } 0 \ (\text{nmap } (\lambda x. x + k) \ \text{ls}))$]

by (*metis* (*no-types*, *lifting*) 1 *Suc-ile-eq* *diff-zero* *iless-Suc-eq* *le-add-diff-inverse* *lessI*
less-Suc-eq-0-disj *nlength-NCons* *nlength-nmap* *nnth-0* *nnth-Suc-NCons* *nnth-nmap*)

have $6: (\text{nlength } ((\text{NCons } 0 \ (\text{nmap } (\lambda x. x + k) \ \text{ls})))) = (\text{Suc } n)$
by (*simp add: 3 eSuc-enat*)

have $7: (\text{enat } (\text{nlast } ((\text{NCons } 0 \ (\text{nmap } (\lambda x. x + k) \ \text{ls})))) = (\text{nlength } \sigma)$
by (*metis* $1 \ 3$ *add.commute* *enat.distinct*(2) *enat-add-sub-same* *less-eqE* *ndropn-nlength*
nlast-NCons *nlast-nmap* *plus-enat-simps*(1))

have $8: (\text{nsubn } \sigma \ 0 \ k \models f)$
using 1 **by** *auto*

have $9: (\forall (i :: \text{nat}). i < (\text{nlength } \text{ls}) \longrightarrow$

$((\text{nsubn } (\text{ndropn } k \ \sigma) \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)) \) \models f)$
 $)$
using 3 **by** *auto*
have 10: $(\forall (i::nat). \ i < (\text{nlength } ls) \longrightarrow$
 $((\text{nsubn } \sigma \ ((\text{nnth } ls \ i)+k) \ ((\text{nnth } ls \ ((i)+1))+k) \) \models f)$
 $)$
by (*metis* 3 *Suc-ile-eq* *add.commute* *add.right-neutral* *nidx-less* *nsubn-ndropn* *plus-1-eq-Suc*)

have 11: $(\forall (i::nat). \ i < \text{nlength } (((\text{nmap } (\lambda x. \ x + k) \ ls)))) \longrightarrow$
 $(\text{nsubn } \sigma \ (\text{nnth } (((\text{nmap } (\lambda x. \ x + k) \ ls)))) \ i) \ (\text{nnth } (((\text{nmap } (\lambda x. \ x + k) \ ls)))) \ (\text{Suc } i) \models f))$
by (*metis* 10 *add.commute* *eSuc-enat* *ileI1* *nlength-nmap* *nnth-nmap* *order-less-imp-le* *plus-1-eq-Suc*)
have 12: $(\forall i. \ (0 < i \wedge i < 1 + (\text{nlength } (\text{nmap } (\lambda x. \ x + k) \ ls))) \longrightarrow$
 $((\text{nsubn } \sigma \ ((\text{nnth } (\text{nmap } (\lambda x. \ x + k) \ ls) \ (i-1)))$
 $((\text{nnth } (\text{nmap } (\lambda x. \ x + k) \ ls) \ ((i)))) \) \models f)$
 $)$
using 11
by (*metis* 3 *Suc-diff-1* *Suc-ile-eq* *eSuc-enat* *iless-Suc-eq* *nlength-nmap* *one-enat-def* *plus-1-eq-Suc*
plus-enat-simps(1))
have 13: $(\forall (i::nat). \ i < \text{nlength } ((\text{NCons } 0 \ (\text{nmap } (\lambda x. \ x + k) \ ls)))) \longrightarrow$
 $(\text{nsubn } \sigma \ (\text{nnth } (((\text{NCons } 0 \ (\text{nmap } (\lambda x. \ x + k) \ ls)))) \ i)$
 $(\text{nnth } (((\text{NCons } 0 \ (\text{nmap } (\lambda x. \ x + k) \ ls)))) \ (\text{Suc } i)) \models f))$
using 12
using 1 11 3 6 *less-Suc-eq-0-disj* **by** *auto*

have 14: $(\text{nnth } (\text{NCons } 0 \ (\text{nmap } (\lambda x. \ x + k) \ ls)) \ 0) = 0$
by *simp*
have 15: $(\text{nnth } (\text{NCons } 0 \ (\text{nmap } (\lambda x. \ x + k) \ ls)) \ (1)) = k$
using 4 **by** (*simp* *add*: 3)
have 16: $(\text{nsubn } \sigma \ (\text{nnth } (\text{NCons } 0 \ (\text{nmap } (\lambda x. \ x + k) \ ls)) \ 0) \ k \models f)$
by (*simp* *add*: 8)
have 17: $\text{nfinite } (\text{NCons } 0 \ (\text{nmap } (\lambda x. \ x + k) \ ls))$
using 6 *nlength-eq-enat-nfiniteD* **by** *blast*
show ?thesis
by (*metis* 13 3 5 6 7 *nfinite-NCons* *nfinite-ndropn* *nfinite-nmap* *nnth-0*)
qed

lemma *chop-wpower-equiv-sem*:

$(\sigma \models (\exists n. \ (\text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) \ n))) =$
 $((\sigma \models \text{empty}) \vee (\sigma \models ((f \wedge \text{more}) \wedge \text{finite}); (\exists n. \ (\text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) \ n))))$
using *wpowersem* **by** *fastforce*

lemma *aschopstar-equiv-wpower-chop-help*:

$(\sigma \models \text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) \ n) =$
 $(\exists (l::nat \ \text{nellist}).$
 $(\text{nlength } l) = n \wedge \text{nfinite } l \wedge \text{nidx } l \wedge (\text{nnth } l \ 0) = 0 \wedge$
 $(\text{enat } (\text{nlast } l)) = (\text{nlength } \sigma) \wedge \text{nfinite } \sigma \wedge$
 $(\forall (i::nat). \ i < (\text{nlength } l) \longrightarrow$
 $((\text{nsubn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) \models f)$
 $)$

)

proof

(*induct n arbitrary: σ*)

case 0

then show ?case

by (*auto simp add: empty-defs nidx-expand nnth-nlast zero-enat-def*)

(*metis eSuc-enat enat-0-iff(1) iless-Suc-eq leD nfinite-conv-nlength-enat nlast-NNil nlength-NNil nnth-NNil zero-le*)

next

case (*Suc n*)

then show ?case

proof –

have ($\sigma \models \text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) \text{ (Suc } n) =$

$(\sigma \models (((f \wedge \text{more}) \wedge \text{finite}); (\text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) \text{ } n)))$)

by *simp*

also have ... =

($\exists k. 0 \leq k \wedge k \leq \text{nlength } (\sigma) \wedge k > 0 \wedge$

$(\text{ntaken } k \text{ } (\sigma) \models f) \wedge$

$(\text{ndropn } k \text{ } (\sigma) \models \text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) \text{ } n)$

)

using *enat-0-iff(1) le-zero-eq* **by** (*auto simp add: more-defs chop-defs finite-defs*)

also have ... =

($\exists k. 0 \leq k \wedge k \leq \text{nlength } (\sigma) \wedge k > 0 \wedge$

$(\text{nsubn } \sigma \text{ } 0 \text{ } k \models f) \wedge$

$(\text{ndropn } k \text{ } (\sigma) \models \text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) \text{ } n)$

)

by (*metis One-nat-def Suc-diff-1 Suc-diff-Suc diff-add ndropn-0 ntaken-ndropn*)

also have ... =

($\exists k. 0 \leq k \wedge k \leq \text{nlength } (\sigma) \wedge k > 0 \wedge$

$(\text{nsubn } \sigma \text{ } 0 \text{ } k \models f) \wedge$

$(\exists l. \text{nfinite } l \wedge (\text{nlength } l) = n \wedge \text{nidx } l \wedge (\text{nnth } l \text{ } 0) = 0 \wedge$

$(\text{enat } (\text{nlast } l)) = (\text{nlength } (\text{ndropn } k \text{ } \sigma)) \wedge \text{nfinite}(\text{ndropn } k \text{ } \sigma) \wedge$

$(\forall (i::\text{nat}). i < (\text{nlength } l) \longrightarrow$

$((\text{nsubn } (\text{ndropn } k \text{ } \sigma) \text{ } (\text{nnth } l \text{ } i) \text{ } (\text{nnth } l \text{ } (\text{Suc } i))) \models f)$

$))$

)

using *Suc.hyps* **by** *blast*

also have ... =

($\exists (l::\text{nat nellist}).$

$(\text{nlength } l) = (\text{Suc } n) \wedge \text{nfinite } l \wedge \text{nidx } l \wedge (\text{nnth } l \text{ } 0) = 0 \wedge$

$(\text{enat } (\text{nlast } l)) = (\text{nlength } \sigma) \wedge \text{nfinite } \sigma \wedge$

$(\forall (i::\text{nat}). i < (\text{nlength } l) \longrightarrow$

$((\text{nsubn } \sigma \text{ } (\text{nnth } l \text{ } i) \text{ } (\text{nnth } l \text{ } (\text{Suc } i))) \models f)$

$))$

)

using *aschopstar-wpower-chain-a*[*of n σ f*]

aschopstar-wpower-chain-b [*of σ f n*] **by** *auto*

finally show $(\sigma \models \text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) (\text{Suc } n)) =$
 $(\exists (l :: \text{nat nellist}).$
 $(\text{nlength } l) = (\text{Suc } n) \wedge \text{nfinite } l \wedge \text{nid } l \wedge (\text{nnth } l \ 0) = 0 \wedge$
 $(\text{enat } (\text{nlast } l)) = (\text{nlength } \sigma) \wedge \text{nfinite } \sigma \wedge$
 $(\forall (i :: \text{nat}). i < (\text{nlength } l) \longrightarrow$
 $((\text{nsubn } \sigma (\text{nnth } l \ i) (\text{nnth } l (\text{Suc } i))) \models f)$
 $)$
 $)$.
qed
qed

lemma *aschopstar-equiv-power-chop*:

$(\sigma \models \text{aschopstar } f) = (\ (\sigma \models (\exists k. \text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) k)))$
using *nfinite-conv-nlength-enat* **by** (*simp add: aschopstar-d-def aschopstar-equiv-wpower-chop-help*)
blast

lemma *ASChopstarEqvSem*:

$(\sigma \models (\text{aschopstar } f = (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{finite}); (\text{aschopstar } f))))$
proof –
have 1: $(\sigma \models \text{aschopstar } f) = (\ (\sigma \models (\exists k. \text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) k)))$
using *aschopstar-equiv-power-chop* **by** *simp*
have 2: $(\ (\sigma \models (\exists k. \text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) k))) =$
 $(\ (\sigma \models \text{empty}) \vee (\sigma \models ((f \wedge \text{more}) \wedge \text{finite}); (\exists n. \text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) n)))$
using *chop-wpower-equiv-sem* **by** *simp*
have 3: $(\sigma \models ((f \wedge \text{more}) \wedge \text{finite}); (\exists n. \text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) n)) =$
 $(\ (\exists n. n \leq \text{nlength } \sigma \wedge ((\text{ntaken } n \ \sigma) \models (f \wedge \text{more}) \wedge \text{finite}) \wedge$
 $((\text{ndropn } n \ \sigma) \models (\exists x. (\text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) x))))$
by (*simp add: chop-defs finite-defs*) *blast*
have 4: $(\ (\exists n. n \leq \text{nlength } \sigma \wedge ((\text{ntaken } n \ \sigma) \models (f \wedge \text{more}) \wedge \text{finite}) \wedge$
 $((\text{ndropn } n \ \sigma) \models (\exists x. (\text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) x)))) =$
 $(\ (\exists n. n \leq \text{nlength } \sigma \wedge ((\text{ntaken } n \ \sigma) \models (f \wedge \text{more}) \wedge \text{finite}) \wedge ((\text{ndropn } n \ \sigma) \models \text{aschopstar } f)))$
by (*simp add: aschopstar-equiv-power-chop*)
have 5: $(\ (\exists n. n \leq \text{nlength } \sigma \wedge ((\text{ntaken } n \ \sigma) \models (f \wedge \text{more}) \wedge \text{finite}) \wedge ((\text{ndropn } n \ \sigma) \models \text{aschopstar } f))) =$
 $(\sigma \models ((f \wedge \text{more}) \wedge \text{finite}); (\text{aschopstar } f))$
by (*simp add: chop-defs finite-defs*) *blast*
have 6: $(\ (\sigma \models \text{empty}) \vee (\sigma \models ((f \wedge \text{more}) \wedge \text{finite}); (\exists n. \text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) n))) =$
 $(\ (\sigma \models (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{finite}); (\text{aschopstar } f))))$
using 3 4 5 **by** *auto*
show *?thesis* **using** 1 2 6 **by** *auto*
qed

lemma *ASChopstarEqvSChopstar*:

$\vdash (\text{aschopstar } f) = (\text{s chopstar } f)$
by (*simp add: Valid-def schopstar-d-def aschopstar-equiv-power-chop fpowerstar-d-def fpower-d-def*)

lemma *len-defs* :

$(w \models \text{len } n) = (\text{nlength } w = n)$

```

proof
  (simp add: len-d-def )
  show ( $w \models (\text{wpower skip } n) = (\text{nlength } w = n)$ )
  proof (induct n arbitrary:w)
  case 0
  then show ?case by (simp add: empty-defs zero-enat-def)
  next
  case (Suc n)
  then show ?case
  by (auto simp add: min-def len-d-def empty-defs chop-defs skip-defs finite-defs nlength-eq-enat-nfiniteD)
    (metis One-nat-def enat.distinct(2) enat-add-sub-same le-iff-add plus-1-eq-Suc plus-enat-simps(1))
  qed
qed

```

```

lemma PowerstarEqvSemhelp1:
   $\vdash \text{empty};(\text{empty} \vee (f \wedge \text{inf})) = (\text{empty} \vee (f \wedge \text{inf}))$ 
using EmptyChopSem by blast

```

```

lemma PowerstarEqvSemhelp2:
   $\vdash (f \wedge \text{inf}) = (f \wedge \text{inf});g$ 
by (auto simp add: Valid-def itl-defs)

```

```

lemma PowerstarEqvSemhelp3:
   $\vdash ((f \wedge \text{inf});g \vee (f \wedge \text{finite});g) = (f ;g)$ 
by (auto simp add: Valid-def itl-defs)

```

```

lemma WPowerstarEqvSem:
  ( $\sigma \models (\text{wpowerstar } f) = (\text{empty} \vee f;(\text{wpowerstar } f))$ )
by (metis intD wpowersem wpowerstar-d-def)

```

```

lemma FPowerstarEqvSem:
  ( $\sigma \models (\text{fpowerstar } f) = (\text{empty} \vee (f \wedge \text{finite});(\text{fpowerstar } f))$ )
by (metis fpowersem fpowerstar-d-def intD)

```

```

lemma PowerstarEqvSem:
  ( $\sigma \models (\text{powerstar } f) = (\text{empty} \vee f;(\text{powerstar } f))$ )
proof –
  have 1: ( $\sigma \models (\text{powerstar } f) =$ 
    ( $\sigma \models (\exists k. \text{fpower } f k);(\text{empty} \vee f \wedge \text{inf}))$ )
  by (simp add: powerstar-d-def fpowerstar-d-def)
  have 2: ( $\sigma \models (\exists k. \text{fpower } f k);(\text{empty} \vee f \wedge \text{inf}) =$ 
    ( $\sigma \models (\text{empty} \vee (f \wedge \text{finite});(\exists k. (\text{fpower } f k)));(\text{empty} \vee f \wedge \text{inf}))$ )
  using fpowersem by (metis inteq-reflection)

```

have 3: $(\sigma \models (\text{empty} \vee (f \wedge \text{finite}); (\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf}))) =$
 $(\sigma \models \text{empty}; (\text{empty} \vee (f \wedge \text{inf})) \vee$
 $((f \wedge \text{finite}); (\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf})))$
by (*metis OrChopEqv inteq-reflection*)
have 4: $(\sigma \models \text{empty}; (\text{empty} \vee (f \wedge \text{inf})) \vee$
 $((f \wedge \text{finite}); (\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf}))) =$
 $(\sigma \models (\text{empty} \vee (f \wedge \text{inf}) \vee ((f \wedge \text{finite}); (\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf}))))$
using *PowerstarEqvSemhelp1*
by (*metis (mono-tags, lifting) inteq-reflection unl-lift2*)
have 5: $(\sigma \models (\text{empty} \vee (f \wedge \text{inf}) \vee ((f \wedge \text{finite}); (\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf})))) =$
 $(\sigma \models (\text{empty} \vee (f \wedge \text{inf}); ((\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf})) \vee$
 $((f \wedge \text{finite}); (\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf}))))$
using *PowerstarEqvSemhelp2*
by (*metis (mono-tags, lifting) inteq-reflection*)
have 51: $(\sigma \models ((f \wedge \text{finite}); (\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf})) =$
 $(f \wedge \text{finite}); ((\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf})))$
by (*auto simp add: ChopAssocSemHelp1*)
have 6: $(\sigma \models (\text{empty} \vee (f \wedge \text{inf}); (\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf})) \vee$
 $((f \wedge \text{finite}); (\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf}))) =$
 $(\sigma \models (\text{empty} \vee (f \wedge \text{inf}); ((\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf})) \vee$
 $(f \wedge \text{finite}); ((\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf}))))$

using 51 **by** *auto*
have 7 : $(\sigma \models (\text{empty} \vee (f \wedge \text{inf}); ((\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf})) \vee$
 $(f \wedge \text{finite}); ((\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf})))) =$
 $(\sigma \models (\text{empty} \vee f; ((\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf}))))$
using *PowerstarEqvSemhelp3* **by** *fastforce*
have 8: $(\sigma \models (\text{empty} \vee f; ((\exists k. (\text{fpower } f \ k)); (\text{empty} \vee f \wedge \text{inf})))) =$
 $(\sigma \models (\text{empty} \vee f; (\text{powerstar } f)))$
by (*simp add: powerstar-d-def fpowerstar-d-def*)
from 1 2 3 4 5 6 7 8 **show** ?thesis **by** *fastforce*
qed

lemma *wpowerchopsem*:

$\vdash (\exists k. \text{wpower } (f \wedge \text{more}) \ k) =$
 $(\text{empty} \vee (f \wedge \text{more}); (\exists k. (\text{wpower } (f \wedge \text{more}) \ k)))$
 $)$

by (*simp add: wpowersem*)

lemma *powerchopsem*:

$\vdash (\exists k. \text{fpower } (f \wedge \text{more}) \ k) =$
 $(\text{empty} \vee ((f \wedge \text{more}) \wedge \text{finite}); (\exists k. (\text{fpower } (f \wedge \text{more}) \ k)))$
 $)$

using *fpowersem* **by** *auto*

lemma *ChopstarEqvSem*:

$(\sigma \models f^\star = (\text{empty} \vee (f \wedge \text{more}); f^\star))$

by (*metis PowerstarEqvSem chopstar-d-def*)

lemma *SChopstarEqvSem*:

$(\sigma \models (\text{schopstar } f) = (\text{empty} \vee (f \wedge \text{more}) \frown (\text{schopstar } f)))$

by (*metis FPowerstarEqvSem schop-d-def schopstar-d-def*)

lemma *ChopstarEqv* :

$\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$

using *ChopstarEqvSem Valid-def* **by** *blast*

lemma *IStarIntros*:

$\vdash \text{empty} \vee f;(\text{istar } f) \longrightarrow (\text{istar } f)$

unfolding *Valid-def* **using** *istar-d.intros* **by** *fastforce*

lemma *IStarCases*:

$(w \models (\text{istar } F)) \implies$

$((w \models \text{empty} \vee (F; \text{istar } F)) \implies P) \implies P$

using *istar-d.cases*[*of F w P*] **by** *auto*

lemma *IStarEqvIStarSem*:

$(w \models (\text{istar } f)) = (w \models \text{empty} \vee f;(\text{istar } f))$

using *IStarCases*[*of f*]

by (*metis istar-d.intros(1) istar-d.intros(2) unl-lift2*)

lemma *IStarEqvIStar*:

$\vdash (\text{istar } f) = (\text{empty} \vee f;(\text{istar } f))$

using *IStarEqvIStarSem*[*of f*] **unfolding** *Valid-def* **by** *auto*

lemma *IStarInductSem*:

assumes $(\bigwedge s. (s \models \text{empty}) \implies P s)$

$(\bigwedge s. (s \models (F;(\text{istar } F) \wedge P))) \implies P s)$

shows $(w \models (\text{istar } F) \longrightarrow P)$

using *assms istar-d.induct*[*of F w P*]

chop-defs[*of F LIFT ((istar F) \wedge P)*]

unfolding *chop-d-def* **by** (*metis intensional-rews(3)*)

lemma *IStarInduct*:

assumes $\vdash \text{empty} \vee f;((\text{istar } f) \wedge g) \longrightarrow g$

shows $\vdash (\text{istar } f) \longrightarrow g$

using *assms IStarInductSem*[*of g*] **unfolding** *Valid-def* **by** (*metis intensional-rews(3)*)

lemma *IStarWeakInductSem*:

assumes $(\bigwedge s. (s \models \text{empty}) \implies P s)$

$(\bigwedge s. (s \models (F;P)) \implies P s)$

shows $(w \models (\text{istar } F) \longrightarrow P)$

using *assms istar-d.induct*[*of F w P*] **using** *chop-defs*[*of F P*]

chop-defs[*of F LIFT ((istar F) \wedge P)*] **unfolding** *chop-d-def*

by (*metis intensional-rews(3)*)

lemma *IStarWeakInduct*:
assumes $\vdash \text{empty} \vee f; g \longrightarrow g$
shows $\vdash (\text{istar } f) \longrightarrow g$
using *assms IStarWeakInductSem*[of *g*] **unfolding** *Valid-def*
by (*metis intensional-rews*(3))

2.5.3 Helper lemmas

lemma *AndEmptyChopAndEmptyEqvAndEmpty*:
 $\vdash (f \wedge \text{empty}); (f \wedge \text{empty}) = (f \wedge \text{empty})$
proof –
have 1: $\vdash (f \wedge \text{empty}); (f \wedge \text{empty}) \longrightarrow (f \wedge \text{empty})$
by (*metis ChopAndB ChopEmpty int-eq*)
have 2: $\vdash (f \wedge \text{empty}) \longrightarrow (f \wedge \text{empty}); (f \wedge \text{empty})$
by (*auto simp add: Valid-def itl-defs zero-enat-def*)
(*metis iless-Suc-eq less-numeral-extra*(1) *ntake-0 ntake-all one-eSuc zero-enat-def*)
show *?thesis*
by (*simp add: 1 2 int-iffI*)
qed

2.5.4 Properties of Chopstar and Chopplus

lemma *FPowerstardef*:
 $\vdash \text{fpowerstar } f = (\exists n. \text{fpower } f n)$
by (*simp add: fpowerstar-d-def*)

lemma *Powerstardef*:
 $\vdash \text{powerstar } f = (\text{fpowerstar } f); (\text{empty} \vee (f \wedge \text{inf}))$
by (*simp add: fpowerstar-d-def powerstar-d-def*)

lemma *Chopstardef*:
 $\vdash \text{chopstar } f = \text{powerstar } (f \wedge \text{more})$
by (*simp add: chopstar-d-def*)

lemma *SChopstardef*:
 $\vdash \text{schopstar } f = \text{fpowerstar } (f \wedge \text{more})$
by (*simp add: schopstar-d-def*)

lemma *WPowerEqRule*:
assumes $\vdash f = g$
shows $\vdash \text{wpower } f n = \text{wpower } g n$
using *assms*
by (*metis int-eq int-simps*(20))

lemma *WPowerCommute*:
 $\vdash (f); (\text{wpower } f n) = (\text{wpower } f n); (f)$
proof
(*induct n*)

```

case 0
then show ?case
by (metis ChopEmpty EmptyChop inteq-reflection wpow-0)
next
case (Suc n)
then show ?case
  by (metis ChopAssoc inteq-reflection wpow-Suc)
qed

```

```

lemma FPowerCommute:
   $\vdash (f \wedge \text{finite}) ; \text{fpower } f \ n = \text{fpower } f \ n ; (f \wedge \text{finite})$ 
unfolding fpower-d-def using WPowerCommute[of LIFT (f  $\wedge$  finite)]
by blast

```

```

lemma WPowerChopInductL:
  assumes  $\vdash g \vee f ; h \longrightarrow h$ 
  shows  $\vdash (\text{wpower } f \ n) ; g \longrightarrow h$ 
using assms
proof
  (induct n)
  case 0
  then show ?case using EmptyChop
  by (metis MP Prop12 int-eq int-iffD1 int-simps(33) wpow-0)
  next
  case (Suc n)
  then show ?case
  by (metis ChopAssoc Prop05 Prop11 RightChopImpChop lift-imp-trans wpow-Suc)
qed

```

```

lemma FPowerChopInductL:
  assumes  $\vdash g \vee f ; h \longrightarrow h$ 
  shows  $\vdash (\text{fpower } f \ n) ; g \longrightarrow h$ 
unfolding fpower-d-def using assms WPowerChopInductL[of g LIFT (f  $\wedge$  finite) h n]
using AndChopA by fastforce

```

```

lemma FPowerChopInductFiniteL:
  assumes  $\vdash g \vee (f \wedge \text{finite}) ; h \longrightarrow h$ 
  shows  $\vdash (\text{fpower } f \ n) ; g \longrightarrow h$ 
unfolding fpower-d-def using assms WPowerChopInductL[of g LIFT (f  $\wedge$  finite) h n]
by blast

```

```

lemma WPowerChopInductMoreL:
  assumes  $\vdash g \vee (f \wedge \text{more}) ; h \longrightarrow h$ 
  shows  $\vdash (\text{wpower } f \ n) ; g \longrightarrow h$ 
using assms
proof
  (induct n)
  case 0
  then show ?case

```



```

by (metis WPowerChopInductL wpow-0)
next
case (Suc n)
then show ?case
proof -
have 1:  $\vdash \text{wpower } f \text{ (Suc } n); g = (f; \text{wpower } f \text{ } n); g$ 
by simp
have 2:  $\vdash (f; \text{wpower } f \text{ } n); g = f; ((\text{wpower } f \text{ } n); g)$ 
by (meson ChopAssoc Prop11)
have 3:  $\vdash f; ((\text{wpower } f \text{ } n); g) \longrightarrow f; h$ 
using RightChopImpChop Suc.hyps Suc.prem by blast
have 31:  $\vdash f = ((f \wedge \text{more}) \vee (f \wedge \text{empty}))$ 
unfolding empty-d-def by fastforce
have 4:  $\vdash f; h = ((f \wedge \text{more}); h \vee ((f \wedge \text{empty}); h))$ 
using 31 OrChopEqvRule by blast
have 5:  $\vdash ((f \wedge \text{more}); h) \longrightarrow h$ 
by (metis Prop03 Prop10 assms inteq-reflection lift-imp-trans)
have 6:  $\vdash ((f \wedge \text{empty}); h) \longrightarrow h$ 
by (metis AndChopB EmptyChop inteq-reflection)
from 5 6 4 3 2 1 show ?thesis
by (metis Prop02 inteq-reflection lift-imp-trans)
qed
qed

```

```

lemma FPowerChopInductFiniteMoreL:
assumes  $\vdash g \vee ((f \wedge \text{finite}) \wedge \text{more}); h \longrightarrow h$ 
shows  $\vdash (\text{fpower } f \text{ } n); g \longrightarrow h$ 
unfolding fpower-d-def using assms
WPowerChopInductMoreL[of g LIFT (f  $\wedge$  finite) h n]
by blast

```

```

lemma FPowerChopInductInfL:
assumes  $\vdash g \vee f; h \longrightarrow h$ 
shows  $\vdash ((\text{fpower } f \text{ } n); (f \wedge \text{inf})); g \longrightarrow h$ 
using assms
proof
(induct n)
case 0
then show ?case
by (metis (no-types, lifting) AndInfChopEqvAndInf ChopAssoc FPowerChopInductFiniteL PowerstarE-
qvSemhelp3
Prop03 Prop10 Prop12 inteq-reflection)
next
case (Suc n)
then show ?case
proof -
have  $\vdash (f \wedge \text{finite}); (\text{fpower } f \text{ } n; ((f \wedge \text{inf}); g)) = (\text{fpower } f \text{ (Suc } n); (f \wedge \text{inf})); g$ 
by (metis ChopAssoc fpower-d-def int-eq wpow-Suc)

```

then show *?thesis*
by (*metis* (*no-types*, *lifting*) *AndChopA ChopAndB ChopAssoc Prop03 Prop10 Suc assms int-eq lift-imp-trans*)

qed

qed

lemma *FChopInductInfMoreL*:

assumes $\vdash g \vee f; h \longrightarrow h$

shows $\vdash ((f\text{power } f \ n); (f \wedge \text{more}) \wedge \text{inf}); g \longrightarrow h$

using *FPowerChopInductInfL*

by (*metis AndMoreAndInfEqvAndInf assms inteq-reflection*)

lemma *WPowerChopInductR*:

assumes $\vdash g \vee h; f \longrightarrow h$

shows $\vdash g; (w\text{power } f \ n) \longrightarrow h$

using *assms*

proof

(*induct n*)

case 0

then show *?case using ChopEmpty*

by (*metis MP Prop12 int-iffD2 int-simps(33) inteq-reflection wpow-0*)

next

case (*Suc n*)

then show *?case*

by (*metis AndChopB ChopAssoc Prop03 Prop10 WPowerCommute inteq-reflection lift-imp-trans wpow-Suc*)

qed

lemma *FPowerChopInductR*:

assumes $\vdash g \vee h; f \longrightarrow h$

shows $\vdash g; (f\text{power } f \ n) \longrightarrow h$

unfolding *fpower-d-def* **using** *assms WPowerChopInductR[of g h LIFT (f \wedge finite) n]*

using *ChopAndA* **by** *fastforce*

lemma *FpowerChopInductInfR*:

assumes $\vdash g \vee h; f \longrightarrow h$

shows $\vdash g; ((f\text{power } f \ n); (f \wedge \text{inf})) \longrightarrow h$

using *assms*

by (*metis ChopAndA ChopAssoc FPowerChopInductR LeftChopImpChop Prop05 int-iffD1 lift-imp-trans*)

lemma *WPowerStarCommute*:

$\vdash f; (\exists n. w\text{power } f \ n) = (\exists n. w\text{power } f \ n); f$

proof –

have 1: $\vdash f; (\exists n. w\text{power } f \ n) = (\exists n. f ; w\text{power } f \ n)$

by (*metis ChopExist Prop11*)

have 2: $\vdash (\exists n. f ; w\text{power } f \ n) = (\exists n. (w\text{power } f \ n); f)$

using *WPowerCommute* **by** (*metis ExEqvRule*)

have 3: $\vdash (\exists n. (w\text{power } f \ n); f) = (\exists n. (w\text{power } f \ n)); f$

by (*simp add: ExistChop*)

from 1 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *FPowerStarCommute*:

$\vdash (f \wedge \text{finite}); (\exists n. \text{fpower } f \ n) = (\exists n. \text{fpower } f \ n); (f \wedge \text{finite})$

unfolding *fpower-d-def* **using** *WPowerStarCommute*[of *LIFT* ($f \wedge \text{finite}$)]

by *blast*

lemma *WPowerSucAndEmptyEqvAndEmpty*:

$\vdash (\text{wpower } (f \wedge \text{empty}) (\text{Suc } n)) = (f \wedge \text{empty})$

proof

(*induct n*)

case 0

then show ?*case*

by (*metis ChopEmpty wpow-0 wpow-Suc*)

next

case (*Suc n*)

then show ?*case*

by (*metis AndEmptyChopAndEmptyEqvAndEmpty int-eq wpow-Suc*)

qed

lemma *FPowerSucAndEmptyEqvAndEmpty*:

$\vdash (\text{fpower } (f \wedge \text{empty}) (\text{Suc } n)) = (f \wedge \text{empty})$

unfolding *fpower-d-def* **using** *WPowerSucAndEmptyEqvAndEmpty*[of *LIFT* ($f \wedge \text{finite}$) *n*]

WPowerEqRule[of *LIFT* ($(f \wedge \text{empty}) \wedge \text{finite}$) *LIFT* ($(f \wedge \text{finite}) \wedge \text{empty}$) (*Suc n*)]

by (*meson EmptyImpFinite Prop01 Prop04 Prop05 Prop10 WPowerEqRule WPowerSucAndEmptyEqvAndEmpty*)

lemma *WPowerOr*:

$\vdash (\text{wpower } (f \vee g) (\text{Suc } n)) = ((f; \text{wpower } (f \vee g) \ n) \vee (g; \text{wpower } (f \vee g) \ n))$

by (*simp add: OrChopEqv*)

lemma *FPowerOr*:

$\vdash (\text{fpower } (f \vee g) (\text{Suc } n)) = (((f \wedge \text{finite}); \text{fpower } (f \vee g) \ n) \vee ((g \wedge \text{finite}); \text{fpower } (f \vee g) \ n))$

by (*simp add: FiniteOr OrChopEqvRule fpower-d-def*)

lemma *WPowerEmptyOrMore*:

$\vdash (\text{wpower } ((f \wedge \text{empty}) \vee (f \wedge \text{more})) (\text{Suc } n)) = ((f \wedge \text{empty}); (\text{wpower } ((f \wedge \text{empty}) \vee (f \wedge \text{more})) \ n) \vee (f \wedge \text{more}); (\text{wpower } ((f \wedge \text{empty}) \vee (f \wedge \text{more})) \ n))$

using *WPowerOr* **by** *blast*

lemma *FPowerEmptyOrMore*:

$\vdash (\text{fpower } ((f \wedge \text{empty}) \vee (f \wedge \text{more})) (\text{Suc } n)) = ((f \wedge \text{empty}); (\text{fpower } ((f \wedge \text{empty}) \vee (f \wedge \text{more})) \ n) \vee (f \wedge \text{more}); (\text{fpower } ((f \wedge \text{empty}) \vee (f \wedge \text{more})) \ n))$

using *FPowerOr*[of *LIFT* ($f \wedge \text{empty}$) *LIFT* ($f \wedge \text{more}$) *n*]

by (*metis* (*no-types*, *lifting*) *AndMoreAndFiniteEqvAndFmore EmptyImpFinite Prop01 Prop05 Prop10 int-eq*)

lemma *WPowerstarInductL*:
assumes $\vdash g \vee f; h \longrightarrow h$
shows $\vdash (\text{wpowerstar } f); g \longrightarrow h$
proof –
have 1: $\vdash (\text{wpowerstar } f); g = ((\exists n. \text{wpower } (f) \ n)); g$
by (*simp add: wpowerstar-d-def LeftChopEqvChop*)
have 2: $\vdash (\exists n. \text{wpower } (f) \ n); g =$
 $(\exists n. (\text{wpower } (f) \ n); g)$
by (*metis ExistChop inteq-reflection*)
have 3: $\bigwedge n. \vdash (\text{wpower } (f) \ n); g \longrightarrow h$
using *WPowerChopInductL*[of *g LIFT(f) h*] *assms* **by** *auto*
have 4: $\vdash (\exists n. ((\text{wpower } (f) \ n)); g) \longrightarrow h$
by (*metis (mono-tags, lifting) 3 Prop10 intI int-eq unl-Rex unl-lift2*)
from 1 2 4 **show** *?thesis*
by (*metis inteq-reflection*)
qed

lemma *FPowerstar-WPowerstar*:
 $\vdash \text{fpowerstar } f = \text{wpowerstar } (f \wedge \text{finite})$
unfolding *fpowerstar-d-def wpowerstar-d-def fpower-d-def* **by** *simp*

lemma *FPowerstarInductL*:
assumes $\vdash g \vee (f \wedge \text{finite}); h \longrightarrow h$
shows $\vdash (\text{fpowerstar } f); g \longrightarrow h$
using *assms WPowerstarInductL*[of *g LIFT (f \wedge finite) h*]
FPowerstar-WPowerstar[of *f*] **by** (*metis int-eq*)

lemma *WPowerstarInductR*:
assumes $\vdash g \vee h; f \longrightarrow h$
shows $\vdash g; (\text{wpowerstar } f) \longrightarrow h$
proof –
have 1: $\vdash g; (\text{wpowerstar } f) = g; (\exists n. \text{wpower } f \ n)$
by (*simp add: wpowerstar-d-def*)
have 2: $\vdash (g; (\exists n. \text{wpower } f \ n)) = (\exists n. g; (\text{wpower } f \ n))$
by (*metis Prop04 ChopExist int-simps(31)*)
have 3: $\bigwedge n. \vdash g; (\text{wpower } f \ n) \longrightarrow h$
using *WPowerChopInductR assms* **by** *blast*
have 4: $\vdash (\exists n. g; (\text{wpower } f \ n)) \longrightarrow h$
by (*metis (mono-tags, lifting) 3 Prop10 intI int-eq unl-Rex unl-lift2*)
from 1 2 4 **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *FPowerstarInductR*:
assumes $\vdash g \vee h; f \longrightarrow h$
shows $\vdash g; (\text{fpowerstar } f) \longrightarrow h$
proof –
have 1: $\vdash g \vee h; (f \wedge \text{finite}) \longrightarrow h$
using *assms using ChopAndA* **by** *fastforce*
show *?thesis*

```

using 1 WPowerstarInductR[of g h LIFT (f ∧ finite) ]
FPowerstar-WPowerstar[of f]
by (metis integ-reflection)
qed

```

```

lemma WPowerstarEqv :
  ⊢ (wpowerstar f) = (empty ∨ f; (wpowerstar f))
using WPowerstarEqvSem by blast

```

```

lemma FPowerstarEqv :
  ⊢ (fpowerstar f) = (empty ∨ (f ∧ finite); (fpowerstar f))
by (simp add: fpowersem fpowerstar-d-def)

```

```

lemma SChopstarEqv :
  ⊢ (schopstar f) = (empty ∨ ((f ∧ more) ∧ finite); (schopstar f))
by (simp add: FPowerstarEqv chopstar-d-def)

```

```

lemma WPowerstar-more-absorb:
  ⊢ (wpowerstar (f ∧ more)) = (wpowerstar f)
proof –
  have 1: ⊢ (wpowerstar (f ∧ more)) ⟶ (wpowerstar f)
    using WPowerstarInductL[of LIFT empty LIFT f ∧ more LIFT (wpowerstar f)]
    by (metis AndChopA ChopEmpty Prop02 Prop05 WPowerChopInductL WPowerstarEqv int-iffD2 integ-reflection wpow-0)
  have 2: ⊢ empty ⟶ wpowerstar (f ∧ more)
    using WPowerstarEqv[of LIFT f ∧ more] by fastforce
  have 20: ⊢ (f ∧ more); wpowerstar (f ∧ more) ⟶ wpowerstar (f ∧ more)
    by (meson Prop03 WPowerstarEqv)
  have 21: ⊢ (f ∧ empty); wpowerstar (f ∧ more) ⟶ wpowerstar (f ∧ more)
    by (metis AndChopB EmptyChop int-eq)
  have 22: ⊢ ((f ∧ more) ∨ (f ∧ empty)); wpowerstar (f ∧ more) =
    ((f ∧ more); wpowerstar (f ∧ more) ∨ (f ∧ empty); wpowerstar (f ∧ more))
    by (simp add: OrChopEqv)
  have 23: ⊢ ((f ∧ more) ∨ (f ∧ empty)) = f
    unfolding empty-d-def by fastforce
  have 3: ⊢ f; wpowerstar (f ∧ more) ⟶ wpowerstar (f ∧ more)
    by (metis 20 21 22 23 Prop02 integ-reflection)
  have 4: ⊢ empty ∨ f; wpowerstar (f ∧ more) ⟶ wpowerstar (f ∧ more)
    using 2 3 by fastforce
  have 5: ⊢ (wpowerstar f) ⟶ (wpowerstar (f ∧ more))
    using 4 WPowerstarInductL[of LIFT empty f LIFT (wpowerstar (f ∧ more))]
    by (metis ChopEmpty integ-reflection)
  show ?thesis using 1 5 by fastforce
qed

```

```

lemma FPowerstar-more-absorb:
  ⊢ (fpowerstar (f ∧ more)) = (fpowerstar f)
proof –
  have 1: ⊢ ((f ∧ more) ∧ finite) = ((f ∧ finite) ∧ more)

```

by *fastforce*
 show *?thesis*
 using *FPowerstar-WPowerstar[of f] FPowerstar-WPowerstar[of LIFT (f ∧ more)]*
WPowerstar-more-absorb[of LIFT (f ∧ finite)] 1
 by (*metis int-eq*)
 qed

lemma *SChopstar-WPowerstar*:
 $\vdash (\text{schopstar } f) = (\text{wpowerstar } (f \wedge \text{finite}))$
 by (*metis FPowerstar-WPowerstar FPowerstar-more-absorb inteq-reflection chopstar-d-def*)

lemma *SChopstar-and-more*:
 $\vdash (\text{schopstar } (f \wedge \text{more})) = (\text{schopstar } f)$
 by (*simp add: FPowerstar-more-absorb chopstar-d-def*)

lemma *IStarWPowerstar*:
 $\vdash (\text{istar } f) = (\text{wpowerstar } f)$
proof –
 have 1: $\vdash (\text{istar } f) \longrightarrow (\text{wpowerstar } f)$
 by (*metis IStarWeakInduct WPowerstarEqv int-iffD2*)
 have 2: $\vdash (\text{wpowerstar } f) \longrightarrow (\text{istar } f)$
 using *WPowerstarInductL[of LIFT empty f LIFT istar f]*
 by (*metis ChopEmpty IStarIntros inteq-reflection*)
 show *?thesis*
 by (*simp add: 1 2 Prop11*)
 qed

2.5.5 Kleene Algebra

lemma *WPowerstar-imp-empty*:
 $\vdash \text{empty} \longrightarrow (\text{wpowerstar } f)$
 using *WPowerstarEqv[of f]* by *fastforce*

lemma *SChopstar-imp-empty*:
 $\vdash \text{empty} \longrightarrow (\text{schopstar } f)$
 using *SChopstarEqv[of f]* by *fastforce*

lemma *WPowerstar-swap*:
 $\vdash (\text{wpowerstar } (f \vee g)) = (\text{wpowerstar } (g \vee f))$
proof –
 have 1: $\vdash (f \vee g) = (g \vee f)$
 by *fastforce*
 show *?thesis*
 by (*metis 1 WPowerstarEqv inteq-reflection*)
 qed

lemma *SChopstar-swap*:
 $\vdash (\text{schopstar } (f \vee g)) = (\text{schopstar } (g \vee f))$
proof –

have 1: $\vdash (f \vee g) = (g \vee f)$
by *fastforce*
show *?thesis*
by (*metis* 1 *SChopstardef int-eq*)
qed

lemma *WPowerstar-1L*:
 $\vdash f;(\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f)$
by (*meson Prop03 WPowerstarEqv*)

lemma *SChopstar-1L*:
 $\vdash (f) \neg (\text{schopstar } f) \longrightarrow (\text{schopstar } f)$
by (*metis SChopstar-WPowerstar WPowerstar-1L inteq-reflection schop-d-def*)

lemma *SChopstarMore-1L*:
 $\vdash (f \wedge \text{more}) \neg (\text{schopstar } f) \longrightarrow (\text{schopstar } f)$
by (*meson AndSChopA SChopstar-1L lift-imp-trans*)

lemma *WPowerstar-trans-eq*:
 $\vdash (\text{wpowerstar } f);(\text{wpowerstar } f) = (\text{wpowerstar } f)$
proof –
have 1: $\vdash (\text{wpowerstar } f) \vee f;(\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f)$
by (*simp add: WPowerstar-1L*)
have 2: $\vdash (\text{wpowerstar } f);(\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f)$
using 1 *WPowerstarInductL* **by** *blast*
have 3: $\vdash (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f);(\text{wpowerstar } f)$
by (*metis AndChopB EmptyChop Prop10 WPowerstar-imp-empty inteq-reflection*)
show *?thesis*
by (*simp add: 2 3 int-iffI*)
qed

lemma *SChopstar-trans-eq*:
 $\vdash (\text{schopstar } f);(\text{schopstar } f) = (\text{schopstar } f)$
by (*metis SChopstar-WPowerstar WPowerstar-trans-eq inteq-reflection*)

lemma *WPowerstar-trans*:
 $\vdash (\text{wpowerstar } f);(\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f)$
using *WPowerstar-trans-eq* **by** *fastforce*

lemma *SChopstar-trans*:
 $\vdash (\text{schopstar } f);(\text{schopstar } f) \longrightarrow (\text{schopstar } f)$
using *SChopstar-trans-eq* **by** *fastforce*

lemma *WPowerstar-induct-lvar*:
assumes $\vdash f;g \longrightarrow g$
shows $\vdash (\text{wpowerstar } f);g \longrightarrow g$
using *assms*
by (*simp add: WPowerstarInductL*)

lemma *SChopstar-induct-lvar*:

assumes $\vdash (f) \frown g \longrightarrow g$
shows $\vdash (\text{schopstar } f) \frown g \longrightarrow g$
using *assms*
by (*metis AndChopA SChopstar-WPowerstar WPowerstar-induct-lvar inteq-reflection lift-imp-trans schop-d-def*)

lemma *SChopstarMore-induct-lvar*:

assumes $\vdash (f \wedge \text{more}) \frown g \longrightarrow g$
shows $\vdash (\text{schopstar } f) \frown g \longrightarrow g$
using *assms*
by (*metis FPowerstar-WPowerstar SChopstar-induct-lvar SChopstar-WPowerstar inteq-reflection schopstar-d-def*)

lemma *WPowerstar-inductL-var-equiv*:

$(\vdash (\text{wpowerstar } f); g \longrightarrow g) = (\vdash f; g \longrightarrow g)$
proof –
have 1: $(\vdash f; g \longrightarrow g) \implies (\vdash (\text{wpowerstar } f); g \longrightarrow g)$
by (*simp add: WPowerstar-induct-lvar*)
have 2: $(\vdash (\text{wpowerstar } f); g \longrightarrow g) \implies (\vdash f; g \longrightarrow g)$
by (*metis (no-types, lifting) AndChopB ChopAssoc EmptyChop Prop10 WPowerstar-1L WPowerstar-imp-empty int-eq lift-and-com*)
show ?thesis
using 1 2 **by** blast
qed

lemma *SChopstar-inductL-var-equiv*:

$(\vdash (\text{schopstar } f) \frown g \longrightarrow g) = (\vdash (f) \frown g \longrightarrow g)$
proof –
have 1: $(\vdash (f) \frown g \longrightarrow g) \implies (\vdash (\text{schopstar } f) \frown g \longrightarrow g)$
by (*simp add: SChopstar-induct-lvar*)
have 10: $\vdash (\text{schopstar } f) \longrightarrow (\text{empty} \vee (f \wedge \text{finite}) \vee f \frown (\text{schopstar } f))$
by (*metis AndSChopA FPowerstarEqv Prop05 Prop08 Prop11 schop-d-def schopstar-d-def*)
have 101: $\vdash (f \wedge \text{finite}) \longrightarrow (\text{schopstar } f)$
by (*metis ChopEmpty RightChopImpChop SChopstar-1L SChopstar-imp-empty int-eq lift-imp-trans schop-d-def*)
have 11: $\vdash (\text{empty} \vee (f \wedge \text{finite}) \vee f \frown (\text{schopstar } f)) \longrightarrow (\text{schopstar } f)$
by (*meson 101 Prop02 SChopstar-1L SChopstar-imp-empty*)
have 12: $\vdash (\text{schopstar } f) = (\text{empty} \vee (f \wedge \text{finite}) \vee f \frown (\text{schopstar } f))$
using 10 11 *int-iffI* **by** blast
have 2: $(\vdash (\text{schopstar } f) \frown g \longrightarrow g) \implies (\vdash (f) \frown g \longrightarrow g)$
using 12
by (*metis (no-types, opaque-lifting) EmptySChop LeftSChopImpSChop SChopAssoc SChopstar-1L SChopstar-imp-empty int-iffI inteq-reflection*)
show ?thesis
using 1 2 **by** blast
qed

lemma *SChopstarMore-induct-lvar-equiv*:

$(\vdash (\text{schopstar } f) \frown g \longrightarrow g) = (\vdash (f \wedge \text{more}) \frown g \longrightarrow g)$
using *SChopstar-inductL-var-equiv* [of *LIFT* $f \wedge \text{more}$ g]
SChopstar-and-more [of f] **by** (*metis int-eq*)

lemma *WPowerstar-induct-lvar-eq*:
assumes $\vdash f;g = g$
shows $\vdash (\text{wpowerstar } f);g \longrightarrow g$
using *assms*
using *WPowerstar-induct-lvar int-iffD1* **by** *blast*

lemma *SChopstar-induct-lvar-eq*:
assumes $\vdash (f) \frown g = g$
shows $\vdash (\text{schopstar } f) \frown g \longrightarrow g$
using *assms*
using *SChopstar-induct-lvar int-iffD1* **by** *blast*

lemma *SChopstarMore-induct-lvar-eq*:
assumes $\vdash (f \wedge \text{more}) \frown g = g$
shows $\vdash (\text{schopstar } f) \frown g \longrightarrow g$
using *assms SChopstar-induct-lvar-eq[of LIFT f \wedge more g] SChopstar-and-more[of f]* **by** (*metis int-eq*)

lemma *WPowerstar-induct-lvar-eq2*:
assumes $\vdash f;g = g$
shows $\vdash (\text{wpowerstar } f);g = g$
using *assms*
by (*meson ChopImpChop EmptyChop Prop11 WPowerstar-imp-empty WPowerstar-induct-lvar-eq lift-imp-trans*)

lemma *SChopstar-induct-lvar-eq2*:
assumes $\vdash (f) \frown g = g$
shows $\vdash (\text{schopstar } f) \frown g = g$
using *assms*
by (*metis AndSChopB EmptySChop Prop10 SChopstar-imp-empty SChopstar-induct-lvar int-eq int-iffD1 int-iffI*)

lemma *SChopstarMore-induct-lvar-eq2*:
assumes $\vdash (f \wedge \text{more}) \frown g = g$
shows $\vdash (\text{schopstar } f) \frown g = g$
using *assms SChopstar-induct-lvar-eq2[of LIFT f \wedge more g] SChopstar-and-more[of f]* **by** (*metis int-eq*)

lemma *WPowerstar-induct-lvar-empty*:
assumes $\vdash \text{empty} \vee f ; g \longrightarrow g$
shows $\vdash (\text{wpowerstar } f) \longrightarrow g$
using *assms*
by (*metis ChopEmpty WPowerstarInductL inteq-reflection*)

lemma *SChopstar-induct-lvar-empty*:
assumes $\vdash \text{empty} \vee (f) \frown g \longrightarrow g$
shows $\vdash (\text{schopstar } f) \longrightarrow g$
using *assms*
by (*metis FPowerstar-WPowerstar FPowerstar-more-absorb WPowerstar-induct-lvar-empty inteq-reflection*
schop-d-def chopstar-d-def)

lemma *SChopstarMore-induct-lvar-empty*:
assumes $\vdash \text{empty} \vee (f \wedge \text{more}) \frown g \longrightarrow g$
shows $\vdash (\text{schopstar } f) \longrightarrow g$
using *assms SChopstar-induct-lvar-empty*[of *LIFT* $f \wedge \text{more } g$] *SChopstar-and-more*[of f] **by** (*metis int-eq*)

lemma *WPowerstar-induct-lvar-star*:
assumes $\vdash f ; (\text{wpowerstar } g) \longrightarrow (\text{wpowerstar } g)$
shows $\vdash (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } g)$
using *assms*
by (*meson Prop02 WPowerstar-imp-empty WPowerstar-induct-lvar-empty*)

lemma *SChopstar-induct-lvar-star*:
assumes $\vdash (f) \frown (\text{schopstar } g) \longrightarrow (\text{schopstar } g)$
shows $\vdash (\text{schopstar } f) \longrightarrow (\text{schopstar } g)$
using *assms*
by (*meson Prop02 SChopstar-imp-empty SChopstar-induct-lvar-empty*)

lemma *SChopstarMore-induct-lvar-star*:
assumes $\vdash (f \wedge \text{more}) \frown (\text{schopstar } g) \longrightarrow (\text{schopstar } g)$
shows $\vdash (\text{schopstar } f) \longrightarrow (\text{schopstar } g)$
using *assms using SChopstar-induct-lvar-star*[of *LIFT* $f \wedge \text{more } g$] *SChopstar-and-more*[of f] **by** (*metis int-eq*)

lemma *WPowerstar-induct-leq*:
assumes $\vdash (h \vee f;g) = g$
shows $\vdash (\text{wpowerstar } f);h \longrightarrow g$
using *assms*
using *WPowerstarInductL int-iffD1* **by** *blast*

lemma *SChopstar-induct-leq*:
assumes $\vdash (h \vee (f)\frown g) = g$
shows $\vdash (\text{schopstar } f)\frown h \longrightarrow g$
using *assms*
by (*metis AndChopA SChopstar-WPowerstar WPowerstar-induct-leq inteq-reflection lift-imp-trans schop-d-def*)

lemma *SChopstarMore-induct-leq*:
assumes $\vdash (h \vee (f \wedge \text{more})\frown g) = g$
shows $\vdash (\text{schopstar } f)\frown h \longrightarrow g$
using *assms SChopstar-induct-leq*[of h *LIFT* $f \wedge \text{more } g$] *SChopstar-and-more*[of f] **by** (*metis int-eq*)

lemma *WPowerstar-subid*:
assumes $\vdash f \longrightarrow \text{empty}$
shows $\vdash (\text{wpowerstar } f) = \text{empty}$

using *assms*
by (*meson ChopEmpty Prop02 Prop11 WPowerstar-imp-empty WPowerstar-induct-lvar-empty lift-imp-trans*)

lemma *SChopstar-subid*:

assumes $\vdash f \longrightarrow \text{empty}$

shows $\vdash (\text{schopstar } f) = \text{empty}$

using *assms*

by (*metis EmptyImpFinite FPowerstar-WPowerstar FPowerstar-more-absorb Prop10 WPowerstar-subid int-eq*

lift-imp-trans chopstar-d-def)

lemma *WPowerstar-subdist*:

$\vdash (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } (f \vee g))$

proof –

have 1: $\vdash f;(\text{wpowerstar } (f \vee g)) \longrightarrow (f \vee g);(\text{wpowerstar } (f \vee g))$

using *OrChopEqv* **by** *fastforce*

have 2: $\vdash (f \vee g);(\text{wpowerstar } (f \vee g)) \longrightarrow (\text{wpowerstar } (f \vee g))$

by (*simp add: WPowerstar-1L*)

show *?thesis*

using 1 2 *WPowerstar-induct-lvar-star lift-imp-trans* **by** *blast*

qed

lemma *SChopstar-subdist*:

$\vdash (\text{schopstar } f) \longrightarrow (\text{schopstar } (f \vee g))$

by (*metis FPowerstar-WPowerstar FPowerstar-more-absorb FiniteOr WPowerstar-subdist int-eq chopstar-d-def*)

lemma *WPowerstar-subdist-var*:

$\vdash (\text{wpowerstar } f) \vee (\text{wpowerstar } g) \longrightarrow (\text{wpowerstar } (f \vee g))$

by (*metis Prop02 WPowerstar-subdist WPowerstar-swap inteq-reflection*)

lemma *SChopstar-subdist-var*:

$\vdash (\text{schopstar } f) \vee (\text{schopstar } g) \longrightarrow (\text{schopstar } (f \vee g))$

by (*metis Prop02 SChopstar-subdist SChopstar-swap inteq-reflection*)

lemma *WPowerstar-iso*:

assumes $\vdash f \longrightarrow g$

shows $\vdash (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } g)$

by (*metis AndChopB Prop05 Prop10 WPowerstar-induct-lvar-star WPowerstarEqv assms inteq-reflection*)

lemma *SChopstar-iso*:

assumes $\vdash f \longrightarrow g$

shows $\vdash (\text{schopstar } f) \longrightarrow (\text{schopstar } g)$

using *assms*

by (*metis AndSChopB Prop10 SChopstar-1L SChopstar-induct-lvar-star inteq-reflection lift-imp-trans*)

lemma *WPowerstar-invol*:

$\vdash (\text{wpowerstar } (\text{wpowerstar } f)) = (\text{wpowerstar } f)$

proof –

```

have 1:  $\vdash (wpowerstar\ f);(wpowerstar\ f) = (wpowerstar\ f)$ 
  by (simp add: WPowerstar-trans-eq)
have 2:  $\vdash (wpowerstar\ (wpowerstar\ f)) \longrightarrow (wpowerstar\ f)$ 
  using 1 WPowerstar-induct-lvar-star int-iffD1 by blast
have 3:  $\vdash (wpowerstar\ (wpowerstar\ f));(wpowerstar\ (wpowerstar\ f)) \longrightarrow (wpowerstar\ (wpowerstar\ f))$ 
  by (simp add: WPowerstar-trans)
have 4:  $\vdash f;(wpowerstar\ (wpowerstar\ f)) \longrightarrow (wpowerstar\ (wpowerstar\ f))$ 
  using WPowerstar-1L WPowerstar-inductL-var-equiv by blast
have 5:  $\vdash (wpowerstar\ f) \longrightarrow (wpowerstar\ (wpowerstar\ f))$ 
  by (simp add: 4 WPowerstar-induct-lvar-star)
show ?thesis
by (simp add: 2 5 Prop11)
qed

```

lemma *SChopstar-invol*:

```

 $\vdash (schopstar\ (schopstar\ f)) = (schopstar\ f)$ 
by (meson Prop11 SChopstar-1L SChopstar-inductL-var-equiv SChopstar-induct-lvar-star)

```

lemma *WPowerstar-star2*:

```

 $\vdash (wpowerstar\ (empty\ \vee\ f)) = (wpowerstar\ f)$ 
by (meson EmptyOrChopEqv Prop02 Prop03 Prop11 WPowerstar-imp-empty WPowerstar-induct-lvar-empty
  WPowerstarEqv lift-imp-trans)

```

lemma *SChopstar-star2*:

```

 $\vdash (schopstar\ (empty\ \vee\ f)) = (schopstar\ f)$ 
by (metis EmptyImpFinite FiniteOr Prop10 SChopstar-WPowerstar WPowerstar-star2 int-eq)

```

lemma *Chop-WPowerstar-Closure*:

```

assumes  $\vdash f \longrightarrow (wpowerstar\ h)$ 
   $\vdash g \longrightarrow (wpowerstar\ h)$ 
shows  $\vdash f;g \longrightarrow (wpowerstar\ h)$ 
proof –
have 1:  $\vdash g\ \vee\ (wpowerstar\ h);(wpowerstar\ h) \longrightarrow (wpowerstar\ h)$ 
  by (metis Prop02 WPowerstar-1L WPowerstar-induct-lvar assms(2))
have 2:  $\vdash (wpowerstar\ h); g \longrightarrow (wpowerstar\ h)$ 
  by (meson Prop02 WPowerstarInductL WPowerstar-1L assms(2))
have 3:  $\vdash f;g \longrightarrow (wpowerstar\ h);g$ 
  by (simp add: LeftChopImpChop assms(1))
show ?thesis
using 2 3 lift-imp-trans by blast
qed

```

lemma *SChop-SChopstar-Closure*:

```

assumes  $\vdash f \longrightarrow (schopstar\ h)$ 
   $\vdash g \longrightarrow (schopstar\ h)$ 
shows  $\vdash f \frown g \longrightarrow (schopstar\ h)$ 
using assms
by (metis AndSChopA Prop10 SChopAndB SChopstarMore-1L SChopstarMore-induct-lvar integ-reflection)

```

lift-and-com lift-imp-trans)

lemma *WPowerstar-wpowerstar-closure*:
assumes $\vdash (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } h)$
shows $\vdash (\text{wpowerstar } (\text{wpowerstar } f)) \longrightarrow (\text{wpowerstar } h)$
using *assms*
by (*simp add: Chop-WPowerstar-Closure WPowerstar-induct-lvar-star*)

lemma *SChopstar-SChopstar-closure*:
assumes $\vdash (\text{schopstar } f) \longrightarrow (\text{schopstar } h)$
shows $\vdash (\text{schopstar } (\text{schopstar } f)) \longrightarrow (\text{schopstar } h)$
using *assms*
by (*metis SChopstar-invol inteq-reflection*)

lemma *WPowerstar-closed-unfold*:
assumes $\vdash (\text{wpowerstar } f) = f$
shows $\vdash f = (\text{empty} \vee f;f)$
using *assms*
by (*metis WPowerstarEqv int-eq*)

lemma *SChopstar-closed-unfold*:
assumes $\vdash (\text{schopstar } f) = f$
shows $\vdash f = (\text{empty} \vee (f) \frown f)$
using *assms*
by (*metis SChopstar-WPowerstar WPowerstarEqv inteq-reflection schop-d-def*)

lemma *SChopstarMore-closed-unfold*:
assumes $\vdash (\text{schopstar } f) = f$
shows $\vdash f = (\text{empty} \vee (f \wedge \text{more}) \frown f)$
using *assms*
by (*metis SChopstarEqv int-eq schop-d-def*)

lemma *WPowerstar-ext*:
 $\vdash f \longrightarrow (\text{wpowerstar } f)$
proof –
have *1*: $\vdash f \longrightarrow f;(\text{wpowerstar } f)$
by (*metis ChopEmpty RightChopImpChop WPowerstar-imp-empty int-eq*)
show *?thesis* **by** (*meson 1 WPowerstar-1L lift-imp-trans*)
qed

lemma *SChopstar-ext*:
 $\vdash f \wedge \text{finite} \longrightarrow (\text{schopstar } f)$
by (*metis SChopstar-WPowerstar WPowerstar-ext inteq-reflection*)

lemma *SChopstarMore-ext*:
 $\vdash f \wedge \text{more} \wedge \text{finite} \longrightarrow (\text{schopstar } f)$
by (*metis AndMoreAndFiniteEqvAndFmore FPowerstar-WPowerstar SChopstar-ext SChopstar-WPowerstar fmore-d-def int-eq schopstar-d-def*)

lemma *WPowerstar-1R*:

$\vdash (wpowerstar\ f) ; f \longrightarrow (wpowerstar\ f)$

by (*simp add: Chop-WPowerstar-Closure WPowerstar-ext*)

lemma *SChopstar-1R*:

$\vdash (schopstar\ f) \frown (f \wedge finite) \longrightarrow (schopstar\ f)$

by (*simp add: SChop-SChopstar-Closure SChopstar-ext*)

lemma *SChopstarMore-1R*:

$\vdash (schopstar\ f) \frown (f \wedge fmore) \longrightarrow (schopstar\ f)$

by (*simp add: SChop-SChopstar-Closure SChopstarMore-ext fmore-d-def*)

lemma *WPowerstar-unfoldR*:

$\vdash empty \vee (wpowerstar\ f) ; f \longrightarrow (wpowerstar\ f)$

by (*meson Prop02 WPowerstar-1R WPowerstar-imp-empty*)

lemma *SChopstar-unfoldR*:

$\vdash empty \vee (schopstar\ f) \frown (f \wedge finite) \longrightarrow (schopstar\ f)$

by (*meson Prop02 SChopstar-1R SChopstar-imp-empty*)

lemma *SChopstarMore-unfoldR*:

$\vdash empty \vee (schopstar\ f) \frown (f \wedge fmore) \longrightarrow (schopstar\ f)$

by (*meson Prop02 SChopstarMore-1R SChopstar-imp-empty*)

lemma *WPowerstar-sim1*:

assumes $\vdash f ; h \longrightarrow h ; g$

shows $\vdash (wpowerstar\ f) ; h \longrightarrow h ; (wpowerstar\ g)$

proof –

have 1: $\vdash (f ; h) ; (wpowerstar\ g) \longrightarrow (h ; g) ; (wpowerstar\ g)$

by (*simp add: LeftChopImpChop assms*)

have 2: $\vdash (h ; g) ; (wpowerstar\ g) \longrightarrow h ; (wpowerstar\ g)$

by (*metis ChopAssoc RightChopImpChop WPowerstar-1L inteq-reflection*)

have 3: $\vdash (f ; h) ; (wpowerstar\ g) \longrightarrow h ; (wpowerstar\ g)$

using 1 2 *lift-imp-trans* **by** *blast*

have 4: $\vdash (wpowerstar\ g) = (empty \vee g ; (wpowerstar\ g))$

by (*simp add: WPowerstarEqv*)

have 41: $\vdash h ; (empty \vee g ; (wpowerstar\ g)) = (h ; empty \vee h ; (g ; (wpowerstar\ g)))$

by (*simp add: ChopOrEqv*)

have 42: $\vdash h ; (g ; (wpowerstar\ g)) = (h ; g) ; (wpowerstar\ g)$

by (*simp add: ChopAssoc*)

have 5: $\vdash h ; (wpowerstar\ g) = (h \vee (h ; g) ; (wpowerstar\ g))$

by (*metis 4 41 42 ChopEmpty inteq-reflection*)

have 6: $\vdash h \longrightarrow h ; (wpowerstar\ g)$

using 5 **by** *fastforce*

have 7: $\vdash h \vee (f ; h) ; (wpowerstar\ g) \longrightarrow h ; (wpowerstar\ g)$

using 3 6 *Prop02* **by** *blast*

show *?thesis*

using *WPowerstarInductL*[of *LIFT h f LIFT h ; (wpowerstar g)*]

by (metis 3 6 ChopAssoc Prop02 inteq-reflection)
qed

lemma *SChopstar-sim1*:

assumes $\vdash f \frown h \longrightarrow h \frown g$

shows $\vdash (\text{schopstar } f) \frown (h \wedge \text{finite}) \longrightarrow h \frown (\text{schopstar } g)$

proof –

have 1: $\vdash (f \frown h) \frown (\text{schopstar } g) \longrightarrow (h \frown g) \frown (\text{schopstar } g)$

by (simp add: LeftSChopImpSChop assms)

have 2: $\vdash (h \frown g) \frown (\text{schopstar } g) \longrightarrow h \frown (\text{schopstar } g)$

by (metis RightSChopImpSChop SChopAssoc SChopstar-1L inteq-reflection)

have 3: $\vdash (f \frown h) \frown (\text{schopstar } g) \longrightarrow h \frown (\text{schopstar } g)$

using 1 2 lift-imp-trans by blast

have 4: $\vdash (\text{schopstar } g) = (\text{empty} \vee (g \wedge \text{more}) \frown (\text{schopstar } g))$

by (simp add: SChopstarEqv schop-d-def)

have 5: $\vdash h \frown (\text{schopstar } g) = ((h \wedge \text{finite}) \vee (h \frown (g \wedge \text{more})) \frown (\text{schopstar } g))$

by (metis ChopEmpty SChopAssoc SChopOrEqv SChopstarEqv int-eq schop-d-def)

have 6: $\vdash h \wedge \text{finite} \longrightarrow h \frown (\text{schopstar } g)$

using 5 by fastforce

have 7: $\vdash (h \wedge \text{finite}) \vee (f \frown h) \frown (\text{schopstar } g) \longrightarrow h \frown (\text{schopstar } g)$

using 3 6 Prop02 by blast

show ?thesis using 7

by (metis 3 6 Prop10 SChopAndB SChopAssoc SChopstar-induct-lvar inteq-reflection lift-imp-trans)

qed

lemma *SChopstarMore-sim1*:

assumes $\vdash (f \wedge \text{more}) \frown h \longrightarrow h \frown (g \wedge \text{more})$

shows $\vdash (\text{schopstar } f) \frown (h \wedge \text{finite}) \longrightarrow h \frown (\text{schopstar } g)$

using assms SChopstar-sim1[of LIFT $f \wedge \text{more}$ h LIFT $g \wedge \text{more}$]

by (metis SChopstar-and-more int-eq)

lemma *WPowerstar-Quasicomm-varA*:

assumes $\vdash (g; f \longrightarrow f; (\text{wpowerstar } (f \vee g)))$

shows $\vdash ((\text{wpowerstar } g); f \longrightarrow f; (\text{wpowerstar } (f \vee g)))$

proof –

have 0: $\vdash (\text{wpowerstar } (\text{wpowerstar } (f \vee g))) = (\text{wpowerstar } (f \vee g))$

by (meson WPowerstar-1L WPowerstar-inductL-var-equiv WPowerstar-induct-lvar-star int-iffI)

have 2: $\vdash f; \text{wpowerstar } \text{wpowerstar } (f \vee g) =$
 $f; (\text{wpowerstar } (f \vee g))$

by (simp add: 0 RightChopEqvChop)

have 4: $\vdash (\text{wpowerstar } g; f \longrightarrow$

$f; \text{wpowerstar } (f \vee g)) =$

$(\text{wpowerstar } g; f \longrightarrow$

$f; \text{wpowerstar } \text{wpowerstar } (f \vee g))$

using 2 by fastforce

show ?thesis

using 4 WPowerstar-sim1[of LIFT g LIFT f LIFT $(\text{wpowerstar } (f \vee g))$]

by (metis 0 assms int-eq)

qed

lemma *SChopstar-Quasicomm-varA*:

assumes $\vdash ((g) \frown ((f) \wedge \text{finite}) \longrightarrow (f) \frown (\text{schopstar } (f \vee g)))$

shows $\vdash ((\text{schopstar } g) \frown ((f) \wedge \text{finite}) \longrightarrow (f) \frown (\text{schopstar } (f \vee g)))$

proof –

have 0: $\vdash (\text{schopstar } (\text{schopstar } (f \vee g))) = (\text{schopstar } (f \vee g))$

by (*simp add: SChopstar-invol*)

have 2: $\vdash ((f) \wedge \text{finite}) \frown \text{schopstar } \text{schopstar } (f \vee g) =$
 $(f) \frown (\text{schopstar } (f \vee g))$

by (*metis (no-types, lifting) 0 AndSCHopA Prop11 Prop12 inteq-reflection itl-def(9) lift-and-com*)

have 3: $\vdash (\text{schopstar } (g) \frown (((f) \wedge \text{finite}) \wedge \text{finite})) =$
 $((\text{schopstar } g) \frown ((f) \wedge \text{finite}))$

by (*metis Prop12 RightSCHopImpSCHop int-iffD2 int-iffI lift-and-com*)

have 4: $\vdash (\text{schopstar } (g) \frown (((f) \wedge \text{finite})) \longrightarrow$
 $(f) \frown \text{schopstar } (f \vee g)) =$
 $(\text{schopstar } (g) \frown (((f) \wedge \text{finite}) \wedge \text{finite}) \longrightarrow$
 $((f) \wedge \text{finite}) \frown \text{schopstar } \text{schopstar } (f \vee g))$

using 2 3 **by** *fastforce*

show *?thesis*

using 4 *SChopstar-sim1*[of *LIFT g LIFT (f) ∧ finite LIFT (schopstar (f ∨ g))*]

by (*metis 0 2 assms int-eq*)

qed

lemma *SChopstarMore-or-and*:

$\vdash \text{schopstar } (f \wedge \text{more} \vee g \wedge \text{more}) = (\text{schopstar } ((f \vee g) \wedge \text{more}))$

proof –

have 1: $\vdash (f \wedge \text{more} \vee g \wedge \text{more}) = ((f \vee g) \wedge \text{more})$

by *fastforce*

show *?thesis*

by (*metis 1 SChopstardef int-eq*)

qed

lemma *SChopstar-QuasicommMore-varA*:

assumes $\vdash ((g \wedge \text{more}) \frown ((f \wedge \text{more}) \wedge \text{finite}) \longrightarrow (f \wedge \text{more}) \frown (\text{schopstar } (f \vee g)))$

shows $\vdash ((\text{schopstar } g) \frown ((f \wedge \text{more}) \wedge \text{finite}) \longrightarrow (f \wedge \text{more}) \frown (\text{schopstar } (f \vee g)))$

using *assms SChopstar-Quasicomm-varA*[of *LIFT g ∧ more LIFT f ∧ more*]

SChopstar-and-more[of *f*] *SChopstar-and-more*[of *g*] *SChopstar-and-more*[of *LIFT (f ∨ g)*]

AndMoreAndFiniteEqvAndFmore[of *f*]

by (*metis SChopstarMore-or-and inteq-reflection*)

lemma *WPowerstar-Quasicomm-varB*:

assumes $\vdash ((\text{wpowerstar } g); f \longrightarrow f ; (\text{wpowerstar } (f \vee g)))$

shows $\vdash (g; f \longrightarrow f ; (\text{wpowerstar } (f \vee g)))$

proof –

have 1: $\vdash g; f \longrightarrow (\text{wpowerstar } g); f$

by (*simp add: LeftChopImpChop WPowerstar-ext*)

show *?thesis*

using 1 *assms lift-imp-trans* **by** *blast*

qed

lemma *SChopstar-Quasicommm-varB*:

assumes $\vdash ((\text{schopstar } g) \frown ((f) \wedge \text{finite}) \longrightarrow (f) \frown (\text{schopstar } (f \vee g)))$

shows $\vdash ((g) \frown ((f) \wedge \text{finite}) \longrightarrow (f) \frown (\text{schopstar } (f \vee g)))$

proof –

have $1: \vdash (g) \frown ((f) \wedge \text{finite}) \longrightarrow (\text{schopstar } g) \frown ((f) \wedge \text{finite})$

by (*metis LeftChopImpChop Prop12 SChopstar-ext int-iffD2 lift-and-com chop-d-def*)

show *?thesis*

using *1 assms lift-imp-trans* **by** *blast*

qed

lemma *SChopstar-QuasicommmMore-varB*:

assumes $\vdash ((\text{schopstar } g) \frown ((f \wedge \text{more}) \wedge \text{finite}) \longrightarrow (f \wedge \text{more}) \frown (\text{schopstar } (f \vee g)))$

shows $\vdash ((g \wedge \text{more}) \frown ((f \wedge \text{more}) \wedge \text{finite}) \longrightarrow (f \wedge \text{more}) \frown (\text{schopstar } (f \vee g)))$

using *assms SChopstar-Quasicommm-varB[of LIFT g \wedge more LIFT f \wedge more]*

SChopstar-and-more[of f] SChopstar-and-more[of g] SChopstar-and-more[of LIFT (f \vee g)]

AndMoreAndFiniteEqvAndFmore[of f]

by (*metis SChopstarMore-or-and inteq-reflection*)

lemma *WPowerstar-Quasicommm-var*:

$(\vdash (g; f \longrightarrow f ; (\text{wpowerstar } (f \vee g)))) =$

$(\vdash ((\text{wpowerstar } g); f \longrightarrow f ; (\text{wpowerstar } (f \vee g))))$

using *WPowerstar-Quasicommm-varA WPowerstar-Quasicommm-varB* **by** *blast*

lemma *SChopstar-Quasicommm-var*:

$(\vdash ((g) \frown ((f) \wedge \text{finite}) \longrightarrow (f) \frown (\text{schopstar } (f \vee g)))) =$

$(\vdash ((\text{schopstar } g) \frown ((f) \wedge \text{finite}) \longrightarrow (f) \frown (\text{schopstar } (f \vee g))))$

using *SChopstar-Quasicommm-varA SChopstar-Quasicommm-varB* **by** *blast*

lemma *SChopstar-QuasicommmMore-var*:

$(\vdash ((g \wedge \text{more}) \frown ((f \wedge \text{more}) \wedge \text{finite}) \longrightarrow (f \wedge \text{more}) \frown (\text{schopstar } (f \vee g)))) =$

$(\vdash ((\text{schopstar } g) \frown ((f \wedge \text{more}) \wedge \text{finite}) \longrightarrow (f \wedge \text{more}) \frown (\text{schopstar } (f \vee g))))$

using *SChopstar-QuasicommmMore-varA SChopstar-QuasicommmMore-varB* **by** *blast*

lemma *WPowerstar-slide1*:

$\vdash (\text{wpowerstar } (f ; g); f \longrightarrow f; (\text{wpowerstar } (g; f)))$

by (*simp add: ChopAssoc WPowerstar-sim1 int-iffD2*)

lemma *SChopstar-slide1*:

$\vdash (\text{schopstar } (f \frown g) \frown (f \wedge \text{finite}) \longrightarrow f \frown (\text{schopstar } (g \frown f)))$

using *SChopstar-sim1*

by (*metis SChopAssoc int-iffD2*)

lemma *WPowerstar-slide1-var1*:

$\vdash (\text{wpowerstar } f); f \longrightarrow f; (\text{wpowerstar } f)$

by (*meson Prop04 WPowerstar-sim1 int-iffD1 int-simps(27)*)

lemma *SChopstar-slide1-var1*:

$\vdash (schopstar\ f) \frown (f \wedge finite) \longrightarrow f \frown (schopstar\ f)$

by (*simp add: SChopstar-sim1*)

lemma *SChopstarMore-slide1-var1*:

$\vdash (schopstar\ f) \frown ((f \wedge more) \wedge finite) \longrightarrow (f \wedge more) \frown (schopstar\ f)$

using *SChopstar-slide1-var1*[*of LIFT f \wedge more*] *SChopstar-and-more*[*of f*]

by (*metis inteq-reflection*)

lemma *wpowerstar-unfoldl-eq*:

$\vdash (empty \vee f; (wpowerstar\ f)) = (wpowerstar\ f)$

by (*meson Prop04 WPowerstar-1L WPowerstar-induct-lvar-star WPowerstarEqv int-iffI*)

lemma *SChopstar-unfoldl-eq*:

$\vdash (empty \vee f \frown (schopstar\ f)) = (schopstar\ f)$

by (*metis SChopstar-WPowerstar WPowerstarEqv inteq-reflection schop-d-def*)

lemma *SChopstarMore-unfoldl-eq*:

$\vdash (empty \vee (f \wedge more) \frown (schopstar\ f)) = (schopstar\ f)$

using *SChopstar-unfoldl-eq*[*of LIFT f \wedge more*] *SChopstar-and-more*[*of f*]

by (*metis inteq-reflection*)

lemma *WPowerstar-rtc1-eq*:

$\vdash (empty \vee f \vee (wpowerstar\ f); (wpowerstar\ f)) = (wpowerstar\ f)$

by (*meson Prop02 Prop05 Prop11 WPowerstar-ext WPowerstar-imp-empty WPowerstar-trans-eq*)

lemma *SChopstar-rtc1-eq*:

$\vdash (empty \vee (f \wedge finite) \vee (schopstar\ f) \frown (schopstar\ f)) = (schopstar\ f)$

by (*meson EmptySChop LeftSChopImpSChop Prop02 Prop04 Prop05 Prop08 Prop11 SChop-SChopstar-Closure*

SChopstar-ext SChopstar-imp-empty)

lemma *SChopstarMore-rtc1-eq*:

$\vdash (empty \vee ((f \wedge more) \wedge finite) \vee (schopstar\ f) \frown (schopstar\ f)) = (schopstar\ f)$

using *SChopstar-rtc1-eq*[*of LIFT f \wedge more*] *SChopstar-and-more*[*of f*]

by (*metis inteq-reflection*)

lemma *WPowerstar-rtc1*:

$\vdash (empty \vee f \vee (wpowerstar\ f); (wpowerstar\ f)) \longrightarrow (wpowerstar\ f)$

by (*meson WPowerstar-rtc1-eq int-iffD1*)

lemma *SChopstar-rtc1*:

$\vdash (empty \vee (f \wedge finite) \vee (schopstar\ f) \frown (schopstar\ f)) \longrightarrow (schopstar\ f)$

by (*meson SChopstar-rtc1-eq int-iffD1*)

lemma *SChopstarMore-rtc1*:

$\vdash (empty \vee ((f \wedge more) \wedge finite) \vee (schopstar\ f) \frown (schopstar\ f)) \longrightarrow (schopstar\ f)$

using *SChopstar-rtc1*[*of LIFT f \wedge more*] *SChopstar-and-more*[*of f*]

by (metis inteq-reflection)

lemma WPowerstar-rtc2:

$(\vdash \text{empty} \vee f;f \longrightarrow f) = (\vdash f = (\text{wpowerstar } f))$

proof –

have 1: $(\vdash \text{empty} \vee f;f \longrightarrow f) \implies (\vdash f = (\text{wpowerstar } f))$

using WPowerstar-induct-lvar-empty[of f LIFT f]

by (simp add: WPowerstar-ext int-iffI)

have 2: $(\vdash f = (\text{wpowerstar } f)) \implies (\vdash \text{empty} \vee f;f \longrightarrow f)$

by (metis WPowerstar-unfoldR inteq-reflection)

show ?thesis

by (metis 1 2 inteq-reflection)

qed

lemma SChopstar-rtc2:

$(\vdash \text{empty} \vee (f) \frown (f \wedge \text{finite}) \longrightarrow (f \wedge \text{finite})) = (\vdash (f \wedge \text{finite}) = (\text{schopstar } f))$

proof –

have 1: $(\vdash \text{empty} \vee (f) \frown (f \wedge \text{finite}) \longrightarrow (f \wedge \text{finite})) \implies (\vdash (f \wedge \text{finite}) = (\text{schopstar } f))$

using SChopstar-induct-lvar-empty[of f LIFT f \wedge finite]

by (simp add: Prop11 SChopstar-ext)

have 2: $(\vdash (f \wedge \text{finite}) = (\text{schopstar } f)) \implies (\vdash \text{empty} \vee (f) \frown (f \wedge \text{finite}) \longrightarrow (f \wedge \text{finite}))$

by (metis Prop02 SChopstar-1L SChopstar-imp-empty inteq-reflection)

show ?thesis

by (metis 1 2 inteq-reflection)

qed

lemma SChopstarMore-rtc2:

$(\vdash \text{empty} \vee (f \wedge \text{more}) \frown ((f \wedge \text{more}) \wedge \text{finite}) \longrightarrow ((f \wedge \text{more}) \wedge \text{finite})) =$

$(\vdash ((f \wedge \text{more}) \wedge \text{finite}) = (\text{schopstar } f))$

using SChopstar-rtc2[of LIFT f \wedge more] SChopstar-and-more[of f]

by (metis inteq-reflection)

lemma SChopstarMore-rtc2-alt:

$(\vdash \text{empty} \vee (f \wedge \text{more}) \frown (f \wedge \text{finite}) \longrightarrow (f \wedge \text{finite})) = (\vdash (f \wedge \text{finite}) = (\text{schopstar } f))$

proof –

have 1: $(\vdash \text{empty} \vee (f \wedge \text{more}) \frown (f \wedge \text{finite}) \longrightarrow (f \wedge \text{finite})) \implies (\vdash (f \wedge \text{finite}) = (\text{schopstar } f))$

using SChopstarMore-induct-lvar-empty[of f LIFT f \wedge finite]

by (simp add: SChopstar-ext int-iffI)

have 2: $(\vdash (f \wedge \text{finite}) = (\text{schopstar } f)) \implies (\vdash \text{empty} \vee (f \wedge \text{more}) \frown (f \wedge \text{finite}) \longrightarrow (f \wedge \text{finite}))$

by (metis SChopstarMore-unfoldl-eq int-eq int-iffD1)

show ?thesis

by (metis 1 2 inteq-reflection)

qed

lemma WPowerstar-rtc3:

$(\vdash (\text{empty} \vee f;f) = f) = (\vdash f = (\text{wpowerstar } f))$

by (metis WPowerstar-rtc2 int-iffD1 inteq-reflection wpowerstar-unfoldl-eq)

lemma *SChopstar-rtc3*:

$(\vdash (\text{empty} \vee (f) \frown (f \wedge \text{finite})) = (f \wedge \text{finite})) = (\vdash (f \wedge \text{finite}) = (\text{schopstar } f))$
by (*metis SChopstar-rtc2 SChopstar-unfoldl-eq int-iffD1 inteq-reflection*)

lemma *SChopstarMore-rtc3*:

$(\vdash (\text{empty} \vee (f \wedge \text{more}) \frown ((f \wedge \text{more}) \wedge \text{finite})) = ((f \wedge \text{more}) \wedge \text{finite})) =$
 $(\vdash ((f \wedge \text{more}) \wedge \text{finite}) = (\text{schopstar } f))$
using *SChopstar-rtc3*[of *LIFT* $f \wedge \text{more}$] *SChopstar-and-more*[of f]
by (*metis inteq-reflection*)

lemma *SChopstarMore-rtc3-alt*:

$(\vdash (\text{empty} \vee (f \wedge \text{more}) \frown (f \wedge \text{finite})) = (f \wedge \text{finite})) = (\vdash (f \wedge \text{finite}) = (\text{schopstar } f))$
by (*metis SChopstarMore-induct-lvar-empty SChopstarMore-unfoldl-eq SChopstar-ext int-iffD1 int-iffI inteq-reflection*)

lemma *WPowerstar-rtc-least*:

assumes $\vdash \text{empty} \vee f \vee g;g \longrightarrow g$
shows $\vdash (\text{wpowerstar } f) \longrightarrow g$
proof –
have 1: $\vdash f \longrightarrow g$
using *assms* **by** *fastforce*
have 2: $\vdash g;g \longrightarrow g$
using *assms* **by** *fastforce*
have 3: $\vdash f;g \longrightarrow g;g$
by (*metis 1 LeftChopImpChop*)
have 4: $\vdash f;g \longrightarrow g$
using 2 3 *lift-imp-trans* **by** *blast*
have 5: $\vdash \text{empty} \longrightarrow g$
using *assms* **by** *fastforce*
show *?thesis*
by (*meson 4 5 Prop02 WPowerstar-induct-lvar-empty*)
qed

lemma *SChopstar-rtc-least*:

assumes $\vdash \text{empty} \vee (f \wedge \text{finite}) \vee (g) \frown (g) \longrightarrow (g)$
shows $\vdash (\text{schopstar } f) \longrightarrow g$
proof –
have 1: $\vdash (f \wedge \text{finite}) \longrightarrow (g \wedge \text{finite})$
using *assms* **by** *fastforce*
have 2: $\vdash (g) \frown (g) \longrightarrow g$
using *assms* **by** *fastforce*
have 3: $\vdash (f) \frown (g) \longrightarrow (g) \frown (g)$
by (*metis 1 LeftChopImpChop schop-d-def*)
have 4: $\vdash (f) \frown (g) \longrightarrow g$
using 2 3 *lift-imp-trans* **by** *blast*
have 5: $\vdash \text{empty} \longrightarrow g$
using *assms* **by** *fastforce*
show *?thesis*
by (*meson 4 5 Prop02 SChopstar-induct-lvar-empty*)

qed

lemma *SChopstarMore-rtc-least*:

assumes $\vdash \text{empty} \vee ((f \wedge \text{more}) \wedge \text{finite}) \vee (g) \frown (g) \longrightarrow (g)$

shows $\vdash (\text{schopstar } f) \longrightarrow g$

using *assms SChopstar-rtc-least[of LIFT f \wedge more] SChopstar-and-more[of f]*

by (*metis inteq-reflection*)

lemma *WPowerstar-rtc-least-eq*:

assumes $\vdash (\text{empty} \vee f \vee g;g) = g$

shows $\vdash (\text{wpowerstar } f) \longrightarrow g$

using *assms*

using *WPowerstar-rtc-least int-iffD1* **by** *blast*

lemma *SChopstar-rtc-least-eq*:

assumes $\vdash (\text{empty} \vee (f \wedge \text{finite}) \vee (g) \frown (g)) = (g)$

shows $\vdash (\text{schopstar } f) \longrightarrow g$

using *SChopstar-rtc-least assms int-iffD1* **by** *blast*

lemma *SChopstarMore-rtc-least-eq*:

assumes $\vdash (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{finite}) \vee (g) \frown (g)) = (g)$

shows $\vdash (\text{schopstar } f) \longrightarrow g$

using *assms SChopstar-rtc-least-eq[of LIFT f \wedge more] SChopstar-and-more[of f]*

by (*metis inteq-reflection*)

lemma *WPowerstar-subdist-var-1*:

$\vdash f \longrightarrow (\text{wpowerstar } (f \vee g))$

by (*meson WPowerstar-ext WPowerstar-subdist lift-imp-trans*)

lemma *SChopstar-subdist-var-1*:

$\vdash f \wedge \text{finite} \longrightarrow (\text{schopstar } (f \vee g))$

by (*meson SChopstar-ext SChopstar-subdist lift-imp-trans*)

lemma *WPowerstar-subdist-var-2*:

$\vdash f;g \longrightarrow (\text{wpowerstar } (f \vee g))$

by (*metis Chop-WPowerstar-Closure WPowerstar-subdist-var-1 WPowerstar-swap inteq-reflection*)

lemma *SChopstar-subdist-var-2*:

$\vdash (f) \frown (g \wedge \text{finite}) \longrightarrow (\text{schopstar } (f \vee g))$

by (*metis Chop-WPowerstar-Closure SChopstar-swap SChopstar-WPowerstar SChopstar-subdist-var-1 inteq-reflection schop-d-def*)

lemma *SChopstarMore-subdist-var-2*:

$\vdash (f \wedge \text{more}) \frown ((g \wedge \text{more}) \wedge \text{finite}) \longrightarrow (\text{schopstar } (f \vee g))$

using *SChopstar-subdist-var-2[of LIFT f \wedge more LIFT g \wedge more]*

SChopstar-and-more[of f] SChopstar-and-more[of g]

SChopstar-and-more[of LIFT (f \vee g)]

SChopstarMore-or-and[of f g]

by (*metis inteq-reflection*)

lemma *WPowerstar-subdist-var-3*:

$\vdash (wpowerstar\ f); (wpowerstar\ g) \longrightarrow (wpowerstar\ (f \vee g))$

by (*metis Chop-WPowerstar-Closure WPowerstar-subdist WPowerstar-swap inteq-reflection*)

lemma *SChopstar-subdist-var-3*:

$\vdash (schopstar\ f) \frown (schopstar\ g) \longrightarrow (schopstar\ (f \vee g))$

by (*metis SChop-SChopstar-Closure SChopstar-subdist SChopstar-swap inteq-reflection*)

lemma *WPowerstar-denest*:

$\vdash (wpowerstar\ (f \vee g)) = (wpowerstar\ ((wpowerstar\ f);(wpowerstar\ g)))$

proof –

have 1: $\vdash f \longrightarrow (wpowerstar\ f)$

by (*simp add: WPowerstar-ext*)

have 2: $\vdash (wpowerstar\ f) \longrightarrow (wpowerstar\ f);(wpowerstar\ g)$

by (*metis ChopEmpty RightChopImpChop WPowerstar-imp-empty int-eq*)

have 3: $\vdash f \longrightarrow (wpowerstar\ f);(wpowerstar\ g)$

using 1 2 **by** *fastforce*

have 4: $\vdash g \longrightarrow (wpowerstar\ g)$

by (*simp add: WPowerstar-ext*)

have 5: $\vdash (wpowerstar\ g) \longrightarrow (wpowerstar\ f);(wpowerstar\ g)$

by (*meson EmptyChop LeftChopImpChop Prop11 WPowerstar-imp-empty lift-imp-trans*)

have 6: $\vdash g \longrightarrow (wpowerstar\ f);(wpowerstar\ g)$

using 4 5 **by** *fastforce*

have 7: $\vdash f \vee g \longrightarrow (wpowerstar\ f);(wpowerstar\ g)$

using 3 6 **by** *fastforce*

have 9: $\vdash (wpowerstar\ (f \vee g)) \longrightarrow (wpowerstar\ ((wpowerstar\ f);(wpowerstar\ g)))$

using 7 *WPowerstar-iso* **by** *blast*

have 10: $\vdash (wpowerstar\ f);(wpowerstar\ g) \longrightarrow (wpowerstar\ (f \vee g))$

by (*simp add: WPowerstar-subdist-var-3*)

have 11: $\vdash (wpowerstar\ ((wpowerstar\ f);(wpowerstar\ g))) \longrightarrow (wpowerstar\ (f \vee g))$

by (*simp add: 10 Chop-WPowerstar-Closure WPowerstar-induct-lvar-star*)

show *?thesis* **using** 11 9 *int-iffI* **by** *blast*

qed

lemma *SChopstar-denest*:

$\vdash (schopstar\ (f \vee g)) = (schopstar\ ((schopstar\ f) \frown (schopstar\ g)))$

proof –

have 1: $\vdash (f \wedge finite) \longrightarrow (schopstar\ f)$

by (*simp add: SChopstar-ext*)

have 2: $\vdash (schopstar\ f) \longrightarrow (schopstar\ f) \frown (schopstar\ g)$

by (*metis (no-types, lifting) AndEmptyChopAndEmptyEqvAndEmpty ChopImpChop EmptyImpFinite FiniteAndEmptyEqvEmpty Prop02 Prop12 SChopAssoc SChopImpSChop SChopstar-1L SChopstar-imp-empty*

SChopstar-induct-lvar-empty inteq-reflection schop-d-def)

have 3: $\vdash (f \wedge finite) \longrightarrow (schopstar\ f) \frown (schopstar\ g)$

using 1 2 **by** *fastforce*

have 4: $\vdash (g \wedge finite) \longrightarrow (schopstar\ g)$

by (*simp add: SChopstar-ext*)

have 5: $\vdash (schopstar\ g) \longrightarrow (schopstar\ f) \frown (schopstar\ g)$

by (meson EmptySChop LeftSChopImpSChop Prop11 SChopstar-imp-empty lift-imp-trans)
 have 6: $\vdash (g \wedge \text{finite}) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)$
 using 4 5 by fastforce
 have 7: $\vdash (f \wedge \text{finite}) \vee (g \wedge \text{finite}) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)$
 using 3 6 by fastforce
 have 8: $\vdash ((f \wedge \text{finite}) \vee (g \wedge \text{finite})) = ((f \vee g) \wedge \text{finite})$
 by fastforce
 have 9: $\vdash (\text{schopstar } (f \vee g)) \longrightarrow (\text{schopstar } ((\text{schopstar } f) \frown (\text{schopstar } g)))$
 by (metis (no-types, opaque-lifting) 7 8 ChopEmpty EmptySChop FPowerstar-WPowerstar
 FPowerstar-more-absorb SChopAssoc SChopstar-iso inteq-reflection schop-d-def schopstar-d-def)
 have 10: $\vdash (\text{schopstar } f) \frown (\text{schopstar } g) \longrightarrow (\text{schopstar } (f \vee g))$
 by (simp add: SChopstar-subdist-var-3)
 have 11: $\vdash (\text{schopstar } ((\text{schopstar } f) \frown (\text{schopstar } g))) \longrightarrow (\text{schopstar } (f \vee g))$
 by (simp add: 10 SChop-SChopstar-Closure SChopstar-induct-lvar-star)
 show ?thesis using 11 9 int-iffI by blast
 qed

lemma WPowerstar-or-var:

$\vdash (\text{wpowerstar } ((\text{wpowerstar } f) \vee (\text{wpowerstar } g))) = (\text{wpowerstar } (f \vee g))$
 using WPowerstar-denest[of f g]
 WPowerstar-denest[of LIFT (wpowerstar f)]
 WPowerstar-invol[of f] WPowerstar-invol[of g]
 by (metis int-eq)

lemma SChopstar-or-var:

$\vdash (\text{schopstar } ((\text{schopstar } f) \vee (\text{schopstar } g))) = (\text{schopstar } (f \vee g))$
 by (metis (no-types, opaque-lifting) FPowerstar-WPowerstar FPowerstar-more-absorb FiniteOr
 SChopstar-invol WPowerstar-denest inteq-reflection schopstar-d-def)

lemma WPowerstar-denest-var:

$\vdash (\text{wpowerstar } f) ; (\text{wpowerstar } (g; (\text{wpowerstar } f))) = (\text{wpowerstar } (f \vee g))$

proof –

have 1: $\vdash \text{empty} \longrightarrow (\text{wpowerstar } f); (\text{wpowerstar } (g; (\text{wpowerstar } f)))$
 by (metis AndEmptyChopAndEmptyEqvAndEmpty ChopImpChop FiniteAndEmptyEqvEmpty WPower-
 star-imp-empty
 inteq-reflection)
 have 2: $\vdash (f; (\text{wpowerstar } f)); (\text{wpowerstar } (g; (\text{wpowerstar } f))) \longrightarrow$
 $(\text{wpowerstar } f); (\text{wpowerstar } (g; (\text{wpowerstar } f)))$
 by (simp add: LeftChopImpChop WPowerstar-1L)
 have 3: $\vdash (g; (\text{wpowerstar } f)); (\text{wpowerstar } (g; (\text{wpowerstar } f))) \longrightarrow$
 $(\text{wpowerstar } f); (\text{wpowerstar } (g; (\text{wpowerstar } f)))$
 by (metis EmptyChop LeftChopImpChop WPowerstar-1L WPowerstar-imp-empty inteq-reflection lift-imp-trans)
 have 4: $\vdash ((f; (\text{wpowerstar } f)); (\text{wpowerstar } (g; (\text{wpowerstar } f)))) \vee$
 $(g; (\text{wpowerstar } f)); (\text{wpowerstar } (g; (\text{wpowerstar } f)))) =$
 $((f \vee g); (\text{wpowerstar } f)); (\text{wpowerstar } (g; (\text{wpowerstar } f))))$
 by (metis OrChopEqv int-eq)
 have 5: $\vdash \text{empty} \vee ((f \vee g); (\text{wpowerstar } f)); (\text{wpowerstar } (g; (\text{wpowerstar } f))) \longrightarrow$
 $(\text{wpowerstar } f); (\text{wpowerstar } (g; (\text{wpowerstar } f)))$
 using 1 2 3 4 by (metis Prop02 int-eq)

```

have 6:  $\vdash (\text{wpowerstar } (f \vee g)) \longrightarrow (\text{wpowerstar } f);(\text{wpowerstar } (g;(\text{wpowerstar } f)))$ 
  by (metis 5 ChopAssoc WPowerstar-induct-lvar-empty int-eq)
have 7:  $\vdash (\text{wpowerstar } (g ;(\text{wpowerstar } f))) \longrightarrow (\text{wpowerstar } ((\text{wpowerstar } g);(\text{wpowerstar } f)))$ 
  by (simp add: LeftChopImpChop WPowerstar-ext WPowerstar-iso)
have 8:  $\vdash (\text{wpowerstar } (g ;(\text{wpowerstar } f))) \longrightarrow (\text{wpowerstar } (f \vee g))$ 
  by (metis 7 WPowerstar-denest WPowerstar-swap inteq-reflection)
have 9:  $\vdash (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } (f \vee g))$ 
  by (simp add: WPowerstar-subdist)
have 10:  $\vdash (\text{wpowerstar } f) ; (\text{wpowerstar } (g;(\text{wpowerstar } f))) \longrightarrow (\text{wpowerstar } (f \vee g))$ 
  using 8 9 Chop-WPowerstar-Closure by blast
show ?thesis
by (simp add: 10 6 int-iffI)
qed

```

lemma *SChopstar-denest-var*:

$\vdash (\text{schopstar } f) \frown (\text{schopstar } ((g) \frown (\text{schopstar } f))) = (\text{schopstar } (f \vee g))$

proof –

```

have 1:  $\vdash \text{empty} \longrightarrow (\text{schopstar } ((g) \frown (\text{schopstar } f)))$ 
  by (simp add: SChopstar-imp-empty)
have 11:  $\vdash \text{empty} \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } ((g) \frown (\text{schopstar } f)))$ 
  by (metis AndEmptyChopAndEmptyEqvAndEmpty ChopImpChop EmptyImpFinite FiniteAndEmptyEqvEmpty)
  Prop12 SChopstar-imp-empty inteq-reflection itl-def(9)
have 2:  $\vdash ((f) \frown (\text{schopstar } f)) \frown (\text{schopstar } ((g) \frown (\text{schopstar } f))) \longrightarrow$ 
   $(\text{schopstar } f) \frown (\text{schopstar } ((g) \frown (\text{schopstar } f)))$ 
  using LeftSChopImpSChop SChopstar-1L by blast
have 3:  $\vdash ((g) \frown (\text{schopstar } f)) \frown (\text{schopstar } ((g) \frown (\text{schopstar } f))) \longrightarrow$ 
   $(\text{schopstar } f) \frown (\text{schopstar } ((g) \frown (\text{schopstar } f)))$ 
  by (metis EmptySChop LeftSChopImpSChop SChopstar-1L SChopstar-imp-empty inteq-reflection lift-imp-trans)
have 4:  $\vdash ((f \vee g) \frown (\text{schopstar } f)) \frown (\text{schopstar } ((g) \frown (\text{schopstar } f))) =$ 
   $((f) \frown (\text{schopstar } f)) \frown (\text{schopstar } ((g) \frown (\text{schopstar } f))) \vee$ 
   $((g) \frown (\text{schopstar } f)) \frown (\text{schopstar } ((g) \frown (\text{schopstar } f)))$ 
  by (metis OrSChopEqv inteq-reflection)
have 5:  $\vdash \text{empty} \vee ((f \vee g) \frown (\text{schopstar } f)) \frown (\text{schopstar } ((g) \frown (\text{schopstar } f))) \longrightarrow$ 
   $(\text{schopstar } f) \frown (\text{schopstar } ((g) \frown (\text{schopstar } f)))$ 
  using 11 2 3 4
  by (metis Prop02 inteq-reflection)
have 6:  $\vdash (\text{schopstar } (f \vee g)) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } ((g) \frown (\text{schopstar } f)))$ 
  by (metis 5 SChopAssoc SChopstar-induct-lvar-empty inteq-reflection)
have 7:  $\vdash (\text{schopstar } (g \frown (\text{schopstar } f))) \longrightarrow (\text{schopstar } ((\text{schopstar } g) \frown (\text{schopstar } f)))$ 
  by (metis AndChopB ChopEmpty LeftChopImpChop Prop12 SChopstar-ext SChopstar-iso inteq-reflection chop-d-def)
have 8:  $\vdash (\text{schopstar } (g \frown (\text{schopstar } f))) \longrightarrow (\text{schopstar } (f \vee g))$ 
  by (metis 7 SChopstar-denest SChopstar-swap inteq-reflection)
have 9:  $\vdash (\text{schopstar } f) \longrightarrow (\text{schopstar } (f \vee g))$ 
  by (simp add: SChopstar-subdist)
have 10:  $\vdash (\text{schopstar } f) \frown (\text{schopstar } ((g) \frown (\text{schopstar } f))) \longrightarrow (\text{schopstar } (f \vee g))$ 
  by (simp add: 8 9 SChop-SChopstar-Closure)
show ?thesis

```


by (*simp add: 10 6 int-iffI*)
qed

lemma *SChopstarMore-denest-var*:

$\vdash (\text{schopstar } f) \frown (\text{schopstar } ((g \wedge \text{more}) \frown (\text{schopstar } f))) = (\text{schopstar } (f \vee g))$
using *SChopstar-denest-var*[*of LIFT f \wedge more LIFT g \wedge more*]
SChopstar-and-more[*of f*] *SChopstar-and-more*[*of g*]
SChopstar-and-more[*of LIFT (f \vee g)*]
SChopstarMore-or-and[*of f g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-denest-var-2*:

$\vdash (\text{wpowerstar } f); (\text{wpowerstar } (g; (\text{wpowerstar } f))) =$
 $(\text{wpowerstar } ((\text{wpowerstar } f); (\text{wpowerstar } g)))$
by (*metis WPowerstar-denest WPowerstar-denest-var int-eq*)

lemma *SChopstar-denest-var-2*:

$\vdash (\text{schopstar } f) \frown (\text{schopstar } (g \frown (\text{schopstar } f))) = (\text{schopstar } ((\text{schopstar } f) \frown (\text{schopstar } g)))$
by (*metis SChopstar-denest SChopstar-denest-var inteq-reflection*)

lemma *SChopstarMore-denest-var-2*:

$\vdash (\text{schopstar } f) \frown (\text{schopstar } ((g \wedge \text{more}) \frown (\text{schopstar } f))) = (\text{schopstar } ((\text{schopstar } f) \frown (\text{schopstar } g)))$
using *SChopstar-denest-var-2*[*of f LIFT g \wedge more*] *SChopstar-and-more*[*of g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-denest-var-3*:

$\vdash (\text{wpowerstar } f); (\text{wpowerstar } ((\text{wpowerstar } g); (\text{wpowerstar } f))) =$
 $(\text{wpowerstar } ((\text{wpowerstar } f); (\text{wpowerstar } g)))$
by (*metis WPowerstar-denest WPowerstar-denest-var WPowerstar-ext WPowerstar-induct-lvar-star*
WPowerstar-trans int-iffI inteq-reflection)

lemma *SChopstar-denest-var-3*:

$\vdash (\text{schopstar } f) \frown (\text{schopstar } ((\text{schopstar } g) \frown (\text{schopstar } f))) = (\text{schopstar } ((\text{schopstar } f) \frown (\text{schopstar } g)))$
by (*metis SChopstar-denest-var-2 SChopstar-invol int-eq*)

lemma *WPowerstar-denest-var-4*:

$\vdash (\text{wpowerstar } ((\text{wpowerstar } g); (\text{wpowerstar } f))) =$
 $(\text{wpowerstar } ((\text{wpowerstar } f); (\text{wpowerstar } g)))$
by (*metis WPowerstar-denest WPowerstar-swap inteq-reflection*)

lemma *SChopstar-denest-var-4*:

$\vdash (\text{schopstar } ((\text{schopstar } g) \frown (\text{schopstar } f))) = (\text{schopstar } ((\text{schopstar } f) \frown (\text{schopstar } g)))$
by (*metis SChopstar-denest SChopstar-swap inteq-reflection*)

lemma *WPowerstar-denest-var-5*:

$\vdash (\text{wpowerstar } f); (\text{wpowerstar } (g; (\text{wpowerstar } f))) =$
 $(\text{wpowerstar } g); (\text{wpowerstar } (f; (\text{wpowerstar } g)))$

by (metis WPowerstar-denest-var WPowerstar-swap int-eq)

lemma *SChopstar-denest-var-5*:

$\vdash (\text{schopstar } f) \frown (\text{schopstar } (g \frown (\text{schopstar } f))) = (\text{schopstar } g) \frown (\text{schopstar } (f \frown (\text{schopstar } g)))$

by (metis SChopstar-denest-var SChopstar-swap inteq-reflection)

lemma *SChopstarMore-denest-var-5*:

$\vdash (\text{schopstar } f) \frown (\text{schopstar } ((g \wedge \text{more}) \frown (\text{schopstar } f))) = (\text{schopstar } g) \frown (\text{schopstar } (f \frown (\text{schopstar } g)))$

using *SChopstar-denest-var-5*[of *f LIFT g \wedge more*]

SChopstar-and-more[of *g*]

by (metis inteq-reflection)

lemma *WPowerstar-denest-var-6*:

$\vdash ((\text{wpowerstar } f); (\text{wpowerstar } g)); (\text{wpowerstar } (f \vee g)) = (\text{wpowerstar } (f \vee g))$

by (metis ChopAssoc WPowerstar-denest WPowerstar-denest-var-3 inteq-reflection)

lemma *SChopstar-denest-var-6*:

$\vdash ((\text{schopstar } f) \frown (\text{schopstar } g)) \frown (\text{schopstar } (f \vee g)) = (\text{schopstar } (f \vee g))$

by (metis SChopAssoc SChopstar-denest SChopstar-denest-var-3 inteq-reflection)

lemma *WPowerstar-denest-var-7*:

$\vdash (\text{wpowerstar } (f \vee g)) ; ((\text{wpowerstar } f); (\text{wpowerstar } g)) = (\text{wpowerstar } (f \vee g))$

proof –

have 1: $\vdash (\text{wpowerstar } (f \vee g)) ; ((\text{wpowerstar } f); (\text{wpowerstar } g)) \longrightarrow$
 $(\text{wpowerstar } (f \vee g)) ; (\text{wpowerstar } ((\text{wpowerstar } f); (\text{wpowerstar } g)))$

by (simp add: RightChopImpChop WPowerstar-ext)

have 2: $\vdash (\text{wpowerstar } (f \vee g)) ; ((\text{wpowerstar } f); (\text{wpowerstar } g)) \longrightarrow (\text{wpowerstar } (f \vee g))$

by (simp add: Chop-WPowerstar-Closure WPowerstar-subdist-var-3)

have 3: $\vdash \text{empty} \longrightarrow (\text{wpowerstar } (f \vee g)) ; ((\text{wpowerstar } f); (\text{wpowerstar } g))$

by (metis ChopEmpty ChopImpChop WPowerstar-imp-empty inteq-reflection)

have 4: $\vdash (f \vee g) ; ((\text{wpowerstar } (f \vee g)) ; ((\text{wpowerstar } f); (\text{wpowerstar } g))) \longrightarrow$
 $((\text{wpowerstar } (f \vee g)) ; ((\text{wpowerstar } f); (\text{wpowerstar } g)))$

by (metis ChopAssoc LeftChopImpChop WPowerstar-1L inteq-reflection)

have 5: $\vdash \text{empty} \vee (f \vee g) ; ((\text{wpowerstar } (f \vee g)) ; ((\text{wpowerstar } f); (\text{wpowerstar } g))) \longrightarrow$
 $((\text{wpowerstar } (f \vee g)) ; ((\text{wpowerstar } f); (\text{wpowerstar } g)))$

using 3 4 Prop02 by blast

have 6: $\vdash (\text{wpowerstar } (f \vee g)) \longrightarrow ((\text{wpowerstar } (f \vee g)) ; ((\text{wpowerstar } f); (\text{wpowerstar } g)))$

using 5 WPowerstar-induct-lvar-empty by blast

show ?thesis using 2 6 int-iffI by blast

qed

lemma *SChopstar-denest-var-7*:

$\vdash (\text{schopstar } (f \vee g)) \frown ((\text{schopstar } f) \frown (\text{schopstar } g)) = (\text{schopstar } (f \vee g))$

proof –

have 1: $\vdash (\text{schopstar } (f \vee g)) \frown ((\text{schopstar } f) \frown (\text{schopstar } g)) \longrightarrow$
 $(\text{schopstar } (f \vee g)) \frown (\text{schopstar } ((\text{schopstar } f) \frown (\text{schopstar } g)))$

by (meson RightSChopImpSChop SChopstar-denest SChopstar-subdist-var-3 int-iffD1 lift-imp-trans)

have 2: $\vdash (\text{schopstar } (f \vee g)) \wedge ((\text{schopstar } f) \wedge (\text{schopstar } g)) \longrightarrow (\text{schopstar } (f \vee g))$
by (*simp add: SChop-SChopstar-Closure SChopstar-subdist-var-3*)
have 3: $\vdash \text{empty} \longrightarrow (\text{schopstar } (f \vee g)) \wedge ((\text{schopstar } f) \wedge (\text{schopstar } g))$
by (*metis EmptySChop SChopImpSChop SChopstar-imp-empty inteq-reflection*)
have 4: $\vdash (f \vee g) \wedge ((\text{schopstar } (f \vee g)) \wedge ((\text{schopstar } f) \wedge (\text{schopstar } g))) \longrightarrow$
 $((\text{schopstar } (f \vee g)) \wedge ((\text{schopstar } f) \wedge (\text{schopstar } g)))$
by (*metis LeftSChopImpSChop SChopAssoc SChopstar-1L inteq-reflection*)
have 5: $\vdash \text{empty} \vee (f \vee g) \wedge ((\text{schopstar } (f \vee g)) \wedge ((\text{schopstar } f) \wedge (\text{schopstar } g))) \longrightarrow$
 $((\text{schopstar } (f \vee g)) \wedge ((\text{schopstar } f) \wedge (\text{schopstar } g)))$
using 3 4 *Prop02* **by** *blast*
have 6: $\vdash (\text{schopstar } (f \vee g)) \longrightarrow ((\text{schopstar } (f \vee g)) \wedge ((\text{schopstar } f) \wedge (\text{schopstar } g)))$
using 5 *SChopstar-induct-lvar-empty* **by** *blast*
show ?thesis **using** 2 6 *int-iffI* **by** *blast*
qed

lemma *WPowerstar-denest-var-8:*

$\vdash ((\text{wpowerstar } f); (\text{wpowerstar } g)); (\text{wpowerstar } ((\text{wpowerstar } f); (\text{wpowerstar } g))) =$
 $(\text{wpowerstar } ((\text{wpowerstar } f); (\text{wpowerstar } g)))$
by (*metis ChopAssoc WPowerstar-denest-var-3 int-eq*)

lemma *SChopstar-denest-var-8:*

$\vdash ((\text{schopstar } f) \wedge (\text{schopstar } g)) \wedge (\text{schopstar } ((\text{schopstar } f) \wedge (\text{schopstar } g))) =$
 $(\text{schopstar } ((\text{schopstar } f) \wedge (\text{schopstar } g)))$
by (*metis SChopstar-denest SChopstar-denest-var-6 inteq-reflection*)

lemma *WPowerstar-denest-var-9:*

$\vdash (\text{wpowerstar } ((\text{wpowerstar } f); (\text{wpowerstar } g))); ((\text{wpowerstar } f); (\text{wpowerstar } g)) =$
 $(\text{wpowerstar } ((\text{wpowerstar } f); (\text{wpowerstar } g)))$
by (*metis WPowerstar-denest WPowerstar-denest-var-7 inteq-reflection*)

lemma *SChopstar-denest-var-9:*

$\vdash (\text{schopstar } ((\text{schopstar } f) \wedge (\text{schopstar } g))) \wedge ((\text{schopstar } f) \wedge (\text{schopstar } g)) =$
 $(\text{schopstar } ((\text{schopstar } f) \wedge (\text{schopstar } g)))$
by (*metis SChopstar-denest SChopstar-denest-var-7 inteq-reflection*)

lemma *WPowerstar-confluence:*

$(\vdash g ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f); (\text{wpowerstar } g)) =$
 $(\vdash (\text{wpowerstar } g) ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f); (\text{wpowerstar } g))$

proof –

have 1: $(\vdash g ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f); (\text{wpowerstar } g)) \implies$
 $(\vdash (\text{wpowerstar } g) ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f); (\text{wpowerstar } g))$
by (*metis Prop02 WPowerstar-imp-empty WPowerstar-rtc2 WPowerstar-sim1 WPowerstar-trans in-*
teq-reflection)
have 2: $(\vdash (\text{wpowerstar } g) ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f); (\text{wpowerstar } g)) \implies$
 $(\vdash g ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f); (\text{wpowerstar } g))$
by (*metis AndChopB Prop10 WPowerstar-ext inteq-reflection lift-imp-trans*)
show ?thesis **using** 1 2 **by** *blast*
qed

lemma *SChopstar-finite*:

$\vdash \text{schopstar } f \longrightarrow \text{finite}$

by (*metis EmptyImpFinite FiniteChopEqvDiamond FiniteChopFiniteEqvFinite Prop02 SChopImpDiamond*

SChopstar-induct-lvar-empty inteq-reflection)

lemma *SChopstar-confluence*:

$(\vdash g \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)) =$

$(\vdash (\text{schopstar } g) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g))$

proof –

have 1: $(\vdash g \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)) \Longrightarrow$

$(\vdash (\text{schopstar } g) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g))$

using *SChopstar-sim1*[of *g LIFT (schopstar f) LIFT (schopstar g)*]

by (*metis Prop10 SChopstar-finite SChopstar-invol inteq-reflection*)

have 2: $(\vdash (\text{schopstar } g) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)) \Longrightarrow$

$(\vdash g \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g))$

by (*metis AndChopB Prop10 SChopstar-ext SChopstar-finite int-eq lift-imp-trans schop-d-def*)

show *?thesis* **using** 1 2 **by** *blast*

qed

lemma *SChopstarMore-confluence*:

$(\vdash (g \wedge \text{more}) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)) =$

$(\vdash (\text{schopstar } g) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g))$

using *SChopstar-confluence*[of *LIFT g \wedge more f*]

SChopstar-and-more[of *g*]

by (*metis inteq-reflection*)

lemma *WPowerstar-church-rosser*:

assumes $\vdash (\text{wpowerstar } g) ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f) ; (\text{wpowerstar } g)$

shows $\vdash (\text{wpowerstar } (f \vee g)) = (\text{wpowerstar } f) ; (\text{wpowerstar } g)$

proof –

have 1: $\vdash ((\text{wpowerstar } f) ; (\text{wpowerstar } g)); ((\text{wpowerstar } f) ; (\text{wpowerstar } g)) \longrightarrow$

$((\text{wpowerstar } f) ; (\text{wpowerstar } f)); ((\text{wpowerstar } g) ; (\text{wpowerstar } g))$

by (*metis (no-types, lifting) ChopAssoc ChopImpChop WPowerstar-trans WPowerstar-trans-eq assms int-eq*)

have 2: $\vdash ((\text{wpowerstar } f) ; (\text{wpowerstar } g)); ((\text{wpowerstar } f) ; (\text{wpowerstar } g)) \longrightarrow$

$((\text{wpowerstar } f) ; (\text{wpowerstar } g))$

by (*metis 1 WPowerstar-trans-eq inteq-reflection*)

have 3: $\vdash \text{empty} \longrightarrow ((\text{wpowerstar } f) ; (\text{wpowerstar } g))$

by (*metis 2 WPowerstar-denest-var-9 WPowerstar-imp-empty WPowerstar-induct-lvar int-eq lift-imp-trans*)

have 4: $\vdash \text{empty} \vee ((\text{wpowerstar } f) ; (\text{wpowerstar } g)); ((\text{wpowerstar } f) ; (\text{wpowerstar } g)) \longrightarrow$

$((\text{wpowerstar } f) ; (\text{wpowerstar } g))$

using 2 3 *Prop02* **by** *blast*

have 5: $\vdash (\text{wpowerstar } ((\text{wpowerstar } f) ; (\text{wpowerstar } g))) \longrightarrow$

$((\text{wpowerstar } f) ; (\text{wpowerstar } g))$

using 4 *WPowerstar-induct-lvar-empty* **by** *blast*

have 6: $\vdash (\text{wpowerstar } (f \vee g)) \longrightarrow (\text{wpowerstar } f) ; (\text{wpowerstar } g)$

by (*metis 5 WPowerstar-denest inteq-reflection*)

have 7: $\vdash (\text{wpowerstar } f) ; (\text{wpowerstar } g) \longrightarrow (\text{wpowerstar } (f \vee g))$

by (simp add: WPowerstar-subdist-var-3)
 show ?thesis
 by (simp add: 6 7 int-iffI)
 qed

lemma *SChopstar-church-rosser*:

assumes $\vdash (\text{schopstar } g) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)$

shows $\vdash (\text{schopstar } (f \vee g)) = (\text{schopstar } f) \frown (\text{schopstar } g)$

proof –

have 0: $\vdash ((\text{schopstar } f) \frown (\text{schopstar } g)) \frown ((\text{schopstar } f)) \longrightarrow$
 $((\text{schopstar } f) \frown (\text{schopstar } f)) \frown ((\text{schopstar } g))$
 by (metis RightSChopImpSChop SChopAssoc assms inteq-reflection)
have 01: $\vdash (((\text{schopstar } f) \frown (\text{schopstar } g)) \frown ((\text{schopstar } f))) \frown (\text{schopstar } g) \longrightarrow$
 $((((\text{schopstar } f) \frown (\text{schopstar } f)) \frown ((\text{schopstar } g))) \frown (\text{schopstar } g))$
 by (simp add: 0 LeftSChopImpSChop)
have 1: $\vdash ((\text{schopstar } f) \frown (\text{schopstar } g)) \frown ((\text{schopstar } f) \frown (\text{schopstar } g)) \longrightarrow$
 $((\text{schopstar } f) \frown (\text{schopstar } f)) \frown ((\text{schopstar } g) \frown (\text{schopstar } g))$
 by (metis 01 SChopAssoc inteq-reflection)
have 2: $\vdash ((\text{schopstar } f) \frown (\text{schopstar } g)) \frown ((\text{schopstar } f) \frown (\text{schopstar } g)) \longrightarrow$
 $((\text{schopstar } f) \frown (\text{schopstar } g))$
 by (metis (no-types, lifting) 1 SChopstar-denest SChopstar-denest-var-3 int-eq int-simps(27))
have 3: $\vdash \text{empty} \longrightarrow ((\text{schopstar } f) \frown (\text{schopstar } g))$
 by (meson EmptySChop Prop11 SChopImpSChop SChopstar-imp-empty lift-imp-trans)
have 4: $\vdash \text{empty} \vee ((\text{schopstar } f) \frown (\text{schopstar } g)) \frown ((\text{schopstar } f) \frown (\text{schopstar } g)) \longrightarrow$
 $((\text{schopstar } f) \frown (\text{schopstar } g))$
 using 2 3 Prop02 by blast
have 5: $\vdash (\text{schopstar } ((\text{schopstar } f) \frown (\text{schopstar } g))) \longrightarrow ((\text{schopstar } f) \frown (\text{schopstar } g))$
 using 4 SChopstar-induct-lvar-empty by blast
have 6: $\vdash (\text{schopstar } (f \vee g)) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)$
 by (metis 5 SChopstar-denest inteq-reflection)
have 7: $\vdash (\text{schopstar } f) \frown (\text{schopstar } g) \longrightarrow (\text{schopstar } (f \vee g))$
 by (simp add: SChopstar-subdist-var-3)
 show ?thesis
 by (simp add: 6 7 int-iffI)
 qed

lemma *WPowerstar-church-rosser-var*:

assumes $\vdash g ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f) ; (\text{wpowerstar } g)$

shows $\vdash (\text{wpowerstar } (f \vee g)) = (\text{wpowerstar } f) ; (\text{wpowerstar } g)$

using *assms*

by (simp add: WPowerstar-church-rosser WPowerstar-confluence)

lemma *SChopstar-church-rosser-var*:

assumes $\vdash g \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)$

shows $\vdash (\text{schopstar } (f \vee g)) = (\text{schopstar } f) \frown (\text{schopstar } g)$

using *assms*

using *SChopstar-church-rosser SChopstar-confluence* by blast

lemma *SChopstarMore-church-rosser-var*:

assumes $\vdash (g \wedge \text{more}) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)$

shows $\vdash (\text{schopstar } (f \vee g)) = (\text{schopstar } f) \frown (\text{schopstar } g)$
using *assms* *SChopstar-church-rosser-var*[of *LIFT* $g \wedge \text{more}$ *LIFT* $f \wedge \text{more}$]
SChopstar-and-more[of g] *SChopstar-and-more*[of f]
SChopstar-and-more[of *LIFT* $(f \vee g)$]
SChopstarMore-or-and[of f g]
by (*metis* *inteq-reflection*)

lemma *WPowerstar-church-rosser-to-confluence*:
assumes $\vdash (\text{wpowerstar } (f \vee g)) = (\text{wpowerstar } f) ; (\text{wpowerstar } g)$
shows $\vdash (\text{wpowerstar } g) ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f) ; (\text{wpowerstar } g)$
using *assms*
by (*metis* *WPowerstar-subdist-var-3* *WPowerstar-swap* *inteq-reflection*)

lemma *SChopstar-church-rosser-to-confluence*:
assumes $\vdash (\text{schopstar } (f \vee g)) = (\text{schopstar } f) \frown (\text{schopstar } g)$
shows $\vdash (\text{schopstar } g) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)$
using *assms*
by (*metis* *SChopstar-subdist-var-3* *SChopstar-swap* *inteq-reflection*)

lemma *WPowerstar-church-rosser-equiv*:
 $(\vdash (\text{wpowerstar } g) ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f) ; (\text{wpowerstar } g)) =$
 $(\vdash (\text{wpowerstar } (f \vee g)) = (\text{wpowerstar } f) ; (\text{wpowerstar } g))$
using *WPowerstar-church-rosser* *WPowerstar-church-rosser-to-confluence* **by** *blast*

lemma *SChopstar-church-rosser-equiv*:
 $(\vdash (\text{schopstar } g) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)) =$
 $(\vdash (\text{schopstar } (f \vee g)) = (\text{schopstar } f) \frown (\text{schopstar } g))$
using *SChopstar-church-rosser* *SChopstar-church-rosser-to-confluence* **by** *blast*

lemma *WPowerstar-confluence-to-local-confluence*:
assumes $\vdash (\text{wpowerstar } g) ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f) ; (\text{wpowerstar } g)$
shows $\vdash g ; f \longrightarrow (\text{wpowerstar } f) ; (\text{wpowerstar } g)$
using *assms*
WPowerstar-church-rosser[of g f]
by (*metis* *WPowerstar-denest* *WPowerstar-denest-var-4* *WPowerstar-subdist-var-2* *inteq-reflection*)

lemma *SChopstar-confluence-to-local-confluence*:
assumes $\vdash (\text{schopstar } g) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)$
shows $\vdash (g) \frown (f \wedge \text{finite}) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)$
using *assms*
SChopstar-church-rosser[of g f]
by (*metis* *SChopstar-denest* *SChopstar-denest-var-4* *SChopstar-subdist-var-2* *inteq-reflection*)

lemma *SChopstarMore-confluence-to-local-confluence*:
assumes $\vdash (\text{schopstar } g) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)$
shows $\vdash (g \wedge \text{more}) \frown (f \wedge \text{finite}) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)$
using *assms*

SChopstar-confluence-to-local-confluence[of *LIFT g ∧ more f*]
SChopstar-and-more[of *g*]
by (*metis integ-reflection*)

lemma *WPowerstar-sup-id-star1*:

assumes $\vdash \text{empty} \longrightarrow f$

shows $\vdash f;(\text{wpowerstar } f) = (\text{wpowerstar } f)$

using *assms*

by (*metis (no-types, lifting) AndEmptyChopAndEmptyEqvAndEmpty ChopImpChop ChopOrEqv ChopOrImp*)

FiniteAndEmptyEqvEmpty Prop02 WPowerstarEqv WPowerstar-1L WPowerstar-imp-empty int-iffI integ-reflection)

lemma *SChopstar-sup-id-star1*:

assumes $\vdash \text{empty} \longrightarrow f$

shows $\vdash f \frown (\text{schopstar } f) = (\text{schopstar } f)$

using *assms*

by (*metis EmptyImpFinite Prop12 SChopstar-WPowerstar WPowerstar-sup-id-star1 integ-reflection schop-d-def*)

lemma *WPowerstar-sup-id-star2*:

assumes $\vdash \text{empty} \longrightarrow f$

shows $\vdash (\text{wpowerstar } f);f = (\text{wpowerstar } f)$

using *assms*

by (*metis ChopEmpty ChopImpChop WPowerstar-1R int-eq int-iffD2 int-iffI*)

lemma *SChopstar-sup-id-star2*:

assumes $\vdash \text{empty} \longrightarrow f$

shows $\vdash (\text{schopstar } f) \frown (f \wedge \text{finite}) = (\text{schopstar } f)$

using *assms*

WPowerstar-sup-id-star2[of *LIFT (f ∧ finite)*] *SChopstar-WPowerstar*[of *f*]

by (*metis EmptyImpFinite Prop10 Prop12 SChopstar-finite integ-reflection schop-d-def*)

lemma *WPowerstar-unfoldr-eq*:

$\vdash (\text{empty} \vee (\text{wpowerstar } f);f) = (\text{wpowerstar } f)$

proof –

have 1: $\vdash (\text{empty} \vee (\text{wpowerstar } f);f) \longrightarrow (\text{wpowerstar } f)$

using *WPowerstar-unfoldR* **by** *auto*

have 2: $\vdash (\text{empty} \vee f;(\text{empty} \vee (\text{wpowerstar } f);f)) =$
 $(\text{empty} \vee (\text{empty} \vee f;(\text{wpowerstar } f));f)$

by (*metis (no-types, lifting) ChopAssoc ChopEmpty ChopOrEqv EmptyOrChopEqv integ-reflection*)

have 3: $\vdash (\text{empty} \vee (\text{empty} \vee f;(\text{wpowerstar } f));f) =$
 $(\text{empty} \vee (\text{wpowerstar } f);f)$

by (*metis 2 integ-reflection wpowerstar-unfoldl-eq*)

have 4: $\vdash (\text{wpowerstar } f) \longrightarrow (\text{empty} \vee (\text{wpowerstar } f);f)$

by (*metis 2 3 WPowerstar-induct-lvar-empty int-eq int-iffD1*)

show *?thesis*

using 1 4 *int-iffI* **by** *blast*

qed

lemma *SChopstar-unfoldr-eq*:

$\vdash (\text{empty} \vee (\text{schopstar } f) \frown (f \wedge \text{finite})) = (\text{schopstar } f)$
proof –
have 1: $\vdash (\text{empty} \vee (\text{schopstar } f) \frown (f \wedge \text{finite})) \longrightarrow (\text{schopstar } f)$
using *SChopstar-unfoldR* **by** *auto*
have 2: $\vdash (\text{empty} \vee f \frown (\text{empty} \vee (\text{schopstar } f) \frown (f \wedge \text{finite}))) =$
 $(\text{empty} \vee (\text{empty} \vee f \frown (\text{schopstar } f) \frown (f \wedge \text{finite})))$
by (*metis* (*no-types*, *lifting*) *ChopEmpty EmptyOrSChopEqv SChopAssoc SChopOrEqv inteq-reflection*
itl-def(9))
have 3: $\vdash (\text{empty} \vee (\text{empty} \vee f \frown (\text{schopstar } f)) \frown (f \wedge \text{finite})) =$
 $(\text{empty} \vee (\text{schopstar } f) \frown (f \wedge \text{finite}))$
by (*metis* 2 *SChopstar-unfoldl-eq inteq-reflection*)
have 4: $\vdash (\text{schopstar } f) \longrightarrow (\text{empty} \vee (\text{schopstar } f) \frown (f \wedge \text{finite}))$
by (*metis* 2 3 *SChopstar-induct-lvar-empty int-eq int-iffD1*)
show ?thesis
using 1 4 *int-iffI* **by** *blast*
qed

lemma *SChopstarMore-unfoldr-eq*:

$\vdash (\text{empty} \vee (\text{schopstar } f) \frown ((f \wedge \text{more}) \wedge \text{finite})) = (\text{schopstar } f)$
using *SChopstar-unfoldr-eq*[of *LIFT* $f \wedge \text{more}$]
SChopstar-and-more[of f]
by (*metis* *inteq-reflection*)

lemma *WPowerstar-star-prod-unfold*:

$\vdash (\text{empty} \vee f; ((\text{wpowerstar } (g;f));g)) = (\text{wpowerstar } (f ; g))$
proof –
have 1: $\vdash (\text{wpowerstar } (f ; g)) = (\text{empty} \vee (\text{wpowerstar } (f;g));(f;g))$
by (*meson* *WPowerstar-unfoldR WPowerstar-unfoldr-eq int-iffD2 int-iffI*)
have 2: $\vdash (\text{wpowerstar } (f;g));(f;g) \longrightarrow f;((\text{wpowerstar } (g;f));g)$
using *WPowerstar-slide1*[of $f g$]
by (*metis* *AndChopB ChopAssoc Prop10 int-eq*)
have 3: $\vdash (\text{wpowerstar } (f ; g)) \longrightarrow (\text{empty} \vee f;((\text{wpowerstar } (g;f));g))$
using 1 2 **by** *fastforce*
have 4: $\vdash f;((\text{wpowerstar } (g;f));g) \longrightarrow (f;g);(\text{wpowerstar } (f ; g))$
using *WPowerstar-slide1*[of $g f$]
by (*metis* *ChopAssoc RightChopImpChop inteq-reflection*)
have 5: $\vdash (\text{empty} \vee f;((\text{wpowerstar } (g;f));g)) \longrightarrow$
 $\text{empty} \vee (f;g);(\text{wpowerstar } (f ; g))$
using 4 **by** *fastforce*
have 6: $\vdash (\text{empty} \vee (f;g);(\text{wpowerstar } (f ; g))) \longrightarrow (\text{wpowerstar } (f ; g))$
by (*meson* *WPowerstarEqv int-iffD2*)
show ?thesis
by (*meson* 3 5 6 *int-iffI lift-imp-trans*)
qed

lemma *FiniteImportSChopRight*:

$\vdash (\text{finite} \wedge (f \frown g)) = f \frown (g \wedge \text{finite})$
by (*metis* *ChopEmpty SChopAssoc inteq-reflection lift-and-com schop-d-def*)

lemma *SChopstar-star-prod-unfold*:

$\vdash (\text{empty} \vee (f) \frown ((\text{schopstar } (g \frown f)) \frown (g \wedge \text{finite}))) = (\text{schopstar } (f \frown g))$

proof –

have 1: $\vdash (\text{schopstar } (f \frown g)) = (\text{empty} \vee (\text{schopstar } (f \frown g)) \frown (f \frown (g \wedge \text{finite})))$

by (*metis FiniteImportSChopRight SChopAndCommute SChopstar-unfoldr-eq inteq-reflection*)

have 2: $\vdash (\text{schopstar } (f \frown g)) \frown (f \frown (g \wedge \text{finite})) \longrightarrow (f) \frown ((\text{schopstar } (g \frown f)) \frown (g \wedge \text{finite}))$

using *SChopstar-slide1*[of *f g*]

by (*metis (no-types, opaque-lifting) ChopAndFiniteDist ChopEmpty LeftSChopImpSChop SChopAssoc inteq-reflection chop-d-def*)

have 3: $\vdash (\text{schopstar } (f \frown g)) \longrightarrow (\text{empty} \vee (f) \frown ((\text{schopstar } (g \frown f)) \frown (g \wedge \text{finite})))$

using 1 2 **by** *fastforce*

have 4: $\vdash (f) \frown ((\text{schopstar } (g \frown f)) \frown (g \wedge \text{finite})) \longrightarrow (f \frown g) \frown (\text{schopstar } (f \frown g))$

using *SChopstar-slide1*[of *g f*]

by (*metis RightSChopImpSChop SChopAssoc inteq-reflection*)

have 5: $\vdash (\text{empty} \vee (f) \frown ((\text{schopstar } (g \frown f)) \frown (g \wedge \text{finite}))) \longrightarrow \text{empty} \vee (f \frown g) \frown (\text{schopstar } (f \frown g))$

using 4 **by** *fastforce*

have 6: $\vdash (\text{empty} \vee (f \frown g) \frown (\text{schopstar } (f \frown g))) \longrightarrow (\text{schopstar } (f \frown g))$

by (*meson Prop02 SChopstar-1L SChopstar-imp-empty*)

show *?thesis*

by (*meson 3 5 6 int-iffI lift-imp-trans*)

qed

lemma *WPowerstar-slide* :

$\vdash (\text{wpowerstar } (f;g));f = f;(\text{wpowerstar } (g;f))$

proof –

have 1: $\vdash f;(\text{wpowerstar } (g;f)) = f;(\text{empty} \vee g;((\text{wpowerstar } (f;g));f))$

by (*metis RightChopEqvChop WPowerstar-star-prod-unfold inteq-reflection*)

have 2: $\vdash f;(\text{empty} \vee g;((\text{wpowerstar } (f;g));f)) = (f; \text{empty} \vee f;(g;((\text{wpowerstar } (f;g));f)))$

by (*simp add: ChopOrEqv*)

have 3: $\vdash f; \text{empty} = \text{empty};f$

by (*metis ChopEmpty EmptyChop int-eq*)

have 4: $\vdash f;(g;((\text{wpowerstar } (f;g));f)) = ((f;g);(\text{wpowerstar } (f;g)));f$

by (*metis ChopAssoc int-eq*)

have 5: $\vdash (f; \text{empty} \vee f;(g;((\text{wpowerstar } (f;g));f))) = (\text{empty};f \vee ((f;g);(\text{wpowerstar } (f;g)));f)$

using 3 4 **by** *fastforce*

have 6: $\vdash (\text{empty};f \vee ((f;g);(\text{wpowerstar } (f;g)));f) = (\text{empty} \vee ((f;g);(\text{wpowerstar } (f;g))));f$

by (*meson OrChopEqv Prop11*)

have 7: $\vdash f;(\text{empty} \vee g;((\text{wpowerstar } (f;g));f)) = (\text{empty} \vee (f;g);(\text{wpowerstar } (f;g)));f$

by (*metis 2 3 4 6 inteq-reflection*)

have 8: $\vdash (\text{empty} \vee (f;g);(\text{wpowerstar } (f;g))) = (\text{wpowerstar } (f;g))$

by (*simp add: wpowerstar-unfoldl-eq*)

show *?thesis* **by** (*metis 1 7 8 int-eq*)

qed

lemma *SChopstar-slide* :

$\vdash (\text{schopstar } (f \frown g)) \frown (f \wedge \text{finite}) = f \frown (\text{schopstar } (g \frown f))$

proof –

have 1: $\vdash f \frown (\text{schopstar } (g \frown f)) = f \frown (\text{empty} \vee g \frown ((\text{schopstar } (f \frown g)) \frown (f \wedge \text{finite})))$

by (*metis RightSChopEqvSChop SChopstar-star-prod-unfold inteq-reflection*)

have 2: $\vdash f \frown (\text{empty} \vee g \frown ((\text{schopstar } (f \frown g)) \frown (f \wedge \text{finite}))) =$

$(f \frown \text{empty} \vee f \frown (g \frown ((\text{schopstar } (f \frown g)) \frown (f \wedge \text{finite}))))$

by (*simp add: SChopOrEqv*)

have 3: $\vdash f \frown \text{empty} = \text{empty} \frown (f \wedge \text{finite})$

by (*metis DiamondEmptyEqvFinite EmptySChop FiniteAndEmptyEqvEmpty SChopAndCommute SChopAndEmptyEqvSChopAndEmpty TrueSChopEqvDiamond inteq-reflection*)

have 4: $\vdash f \frown (g \frown ((\text{schopstar } (f \frown g)) \frown (f \wedge \text{finite}))) =$

$((f \frown g) \frown (\text{schopstar } (f \frown g))) \frown (f \wedge \text{finite})$

by (*metis SChopAssoc inteq-reflection*)

have 5: $\vdash (f \frown \text{empty} \vee f \frown (g \frown ((\text{schopstar } (f \frown g)) \frown (f \wedge \text{finite})))) =$

$(\text{empty} \frown (f \wedge \text{finite}) \vee ((f \frown g) \frown (\text{schopstar } (f \frown g))) \frown (f \wedge \text{finite}))$

using 3 4 **by** *fastforce*

have 6: $\vdash (\text{empty} \frown (f \wedge \text{finite}) \vee ((f \frown g) \frown (\text{schopstar } (f \frown g))) \frown (f \wedge \text{finite})) =$

$(\text{empty} \vee ((f \frown g) \frown (\text{schopstar } (f \frown g)))) \frown (f \wedge \text{finite})$

by (*meson OrSChopEqv Prop11*)

have 7: $\vdash f \frown (\text{empty} \vee g \frown ((\text{schopstar } (f \frown g)) \frown (f \wedge \text{finite}))) =$

$(\text{empty} \vee (f \frown g) \frown (\text{schopstar } (f \frown g))) \frown (f \wedge \text{finite})$

by (*metis 2 3 4 6 inteq-reflection*)

have 8: $\vdash (\text{empty} \vee (f \frown g) \frown (\text{schopstar } (f \frown g))) = (\text{schopstar } (f \frown g))$

by (*simp add: SChopstar-unfoldl-eq*)

show *?thesis* **by** (*metis 1 7 8 int-eq*)

qed

lemma *WPowerstar-slide-var* :

$\vdash (\text{wpowerstar } f) ; f = f; (\text{wpowerstar } f)$

by (*metis EmptyOrChopEqv Prop11 WPowerstarInductR WPowerstar-slide1-var1 WPowerstar-unfoldr-eq int-eq*)

lemma *SChopstar-slide-var* :

$\vdash (\text{schopstar } f) \frown (f \wedge \text{finite}) = f \frown (\text{schopstar } f)$

using *WPowerstar-slide-var*

by (*metis (no-types, lifting) FPowerstar-WPowerstar Prop10 SChopstar-finite SChopstar-WPowerstar int-eq schop-d-def*)

lemma *SChopstarMore-slide-var* :

$\vdash (\text{schopstar } f) \frown ((f \wedge \text{more}) \wedge \text{finite}) = (f \wedge \text{more}) \frown (\text{schopstar } f)$

using *SChopstar-slide-var* [of *LIFT f* \wedge *more*]

SChopstar-and-more [of *f*]

by (*metis inteq-reflection*)

lemma *WPowerstar-or-unfold-var*:

$\vdash (\text{empty} \vee (\text{wpowerstar } f); ((\text{wpowerstar } (f \vee g)); (\text{wpowerstar } g))) = (\text{wpowerstar } (f \vee g))$

by (*metis ChopAssoc WPowerstar-denest WPowerstar-denest-var-4 WPowerstar-slide inteq-reflection wpowerstar-unfoldl-eq*)

lemma *SChopstar-or-unfold-var*:

$\vdash (\text{empty} \vee (\text{schopstar } f) \frown ((\text{schopstar } (f \vee g)) \frown (\text{schopstar } g))) = (\text{schopstar } (f \vee g))$

proof –

have 1: $\vdash (\text{schopstar } f) \frown ((\text{schopstar } (f \vee g)) \frown (\text{schopstar } g)) =$

$(\text{schopstar } f) \frown ((\text{schopstar } (f \vee g)) \frown ((\text{schopstar } g) \wedge \text{finite}))$

by (*simp add: Prop10 RightSChopEqvSChop SChopstar-finite*)

have 2: $\vdash (\text{schopstar } f) \frown ((\text{schopstar } (f \vee g)) \frown ((\text{schopstar } g) \wedge \text{finite})) =$
 $(\text{schopstar } (f \vee g))$

by (*metis (no-types, lifting) SChopstar-denest-var SChopstar-denest-var-2 SChopstar-denest-var-3 SChopstar-slide SChopstar-swap int-eq*)

show *?thesis*

using 1 2 *SChopstar-imp-empty* **by** *fastforce*

qed

lemma *WPowerstar-or-unfold*:

$\vdash ((\text{wpowerstar } f) \vee (\text{wpowerstar } f);(g;(\text{wpowerstar } (f \vee g)))) = (\text{wpowerstar } (f \vee g))$

proof –

have 1: $\vdash (\text{wpowerstar } (f \vee g)) = (\text{wpowerstar } f);(\text{wpowerstar } (g;(\text{wpowerstar } f)))$

by (*meson Prop11 WPowerstar-denest-var*)

have 2: $\vdash (\text{wpowerstar } f);(\text{wpowerstar } (g;(\text{wpowerstar } f))) =$

$(\text{wpowerstar } f);(\text{empty} \vee (g;(\text{wpowerstar } f));(\text{wpowerstar } (g;(\text{wpowerstar } f))))$

using *RightChopEqvChop WPowerstarEqv* **by** *blast*

have 3: $\vdash (\text{wpowerstar } f);(\text{empty} \vee (g;(\text{wpowerstar } f));(\text{wpowerstar } (g;(\text{wpowerstar } f)))) =$
 $(\text{wpowerstar } f);(\text{empty} \vee g;(\text{wpowerstar } (f \vee g)))$

by (*metis 1 2 ChopAssoc int-eq*)

have 4: $\vdash (\text{wpowerstar } f);(\text{empty} \vee g;(\text{wpowerstar } (f \vee g))) =$

$((\text{wpowerstar } f); \text{empty} \vee (\text{wpowerstar } f);(g;(\text{wpowerstar } (f \vee g))))$

using *ChopOrEqv* **by** *blast*

have 5: $\vdash (\text{wpowerstar } f); \text{empty} = (\text{wpowerstar } f)$

by (*simp add: ChopEmpty*)

have 6: $\vdash ((\text{wpowerstar } f); \text{empty} \vee (\text{wpowerstar } f);(g;(\text{wpowerstar } (f \vee g)))) =$
 $((\text{wpowerstar } f) \vee (\text{wpowerstar } f);(g;(\text{wpowerstar } (f \vee g))))$

using 5 **by** *auto*

show *?thesis*

by (*metis 1 2 3 4 6 int-eq*)

qed

lemma *SChopstar-or-unfold*:

$\vdash ((\text{schopstar } f) \vee (\text{schopstar } f) \frown (g \frown (\text{schopstar } (f \vee g)))) = (\text{schopstar } (f \vee g))$

proof –

have 1: $\vdash (\text{schopstar } (f \vee g)) = (\text{schopstar } f) \frown (\text{schopstar } (g \frown (\text{schopstar } f)))$

by (*meson Prop11 SChopstar-denest-var*)

have 2: $\vdash (\text{schopstar } f) \frown (\text{schopstar } (g \frown (\text{schopstar } f))) =$

$(\text{schopstar } f) \frown (\text{empty} \vee (g \frown (\text{schopstar } f)) \frown (\text{schopstar } (g \frown (\text{schopstar } f))))$

by (*meson Prop11 RightSChopEqvSChop SChopstar-unfoldl-eq*)

have 3: $\vdash (\text{schopstar } f) \frown (\text{empty} \vee (g \frown (\text{schopstar } f)) \frown (\text{schopstar } (g \frown (\text{schopstar } f)))) =$
 $(\text{schopstar } f) \frown (\text{empty} \vee g \frown (\text{schopstar } (f \vee g)))$

by (*metis 1 2 SChopAssoc inteq-reflection*)

have 4: $\vdash (\text{schopstar } f) \frown (\text{empty} \vee g \frown (\text{schopstar } (f \vee g))) =$

$((\text{schopstar } f) \frown \text{empty} \vee (\text{schopstar } f) \frown (g \frown (\text{schopstar } (f \vee g))))$
using *SChopOrEqv* **by** *blast*
have 5: $\vdash (\text{schopstar } f) \frown \text{empty} = (\text{schopstar } f)$
by (*metis ChopEmpty Prop10 SChopstar-finite inteq-reflection chop-d-def*)
have 6: $\vdash ((\text{schopstar } f) \frown \text{empty} \vee (\text{schopstar } f) \frown (g \frown (\text{schopstar } (f \vee g)))) =$
 $((\text{schopstar } f) \vee (\text{schopstar } f) \frown (g \frown (\text{schopstar } (f \vee g))))$
using 5 **by** *auto*
show *?thesis*
by (*metis 1 2 3 4 6 int-eq*)
qed

lemma *SChopstarMore-or-unfold*:

$\vdash ((\text{schopstar } f) \vee (\text{schopstar } f) \frown ((g \wedge \text{more}) \frown (\text{schopstar } (f \vee g)))) = (\text{schopstar } (f \vee g))$
using *SChopstar-or-unfold*[of *LIFT f* \wedge *more LIFT g* \wedge *more*]
SChopstar-and-more[of *f*] *SChopstar-and-more*[of *g*]
SChopstar-and-more[of *LIFT (f* \vee *g)*]
SChopstarMore-or-and[of *f g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-troeger*:

$\vdash (\text{wpowerstar } (f \vee g)); h =$
 $(\text{wpowerstar } f); (g; ((\text{wpowerstar } (f \vee g)); h) \vee h)$
proof –
have 1: $\vdash (\text{wpowerstar } (f \vee g)); h =$
 $((\text{wpowerstar } f) \vee (\text{wpowerstar } f); (g; (\text{wpowerstar } (f \vee g)))); h$
using *WPowerstar-or-unfold*[of *f g*] **by** (*metis LeftChopEqvChop int-eq*)
have 2: $\vdash ((\text{wpowerstar } f) \vee (\text{wpowerstar } f); (g; (\text{wpowerstar } (f \vee g)))); h =$
 $((\text{wpowerstar } f); h \vee ((\text{wpowerstar } f); (g; (\text{wpowerstar } (f \vee g))))); h$
by (*simp add: OrChopEqv*)
have 3: $\vdash ((\text{wpowerstar } f); (g; (\text{wpowerstar } (f \vee g)))); h =$
 $(\text{wpowerstar } f); (g; ((\text{wpowerstar } (f \vee g)); h))$
by (*metis ChopAssoc inteq-reflection*)
have 3: $\vdash ((\text{wpowerstar } f); h \vee ((\text{wpowerstar } f); (g; (\text{wpowerstar } (f \vee g))))); h =$
 $(\text{wpowerstar } f); (h \vee g; ((\text{wpowerstar } (f \vee g)); h))$
by (*metis 3 ChopOrEqv inteq-reflection*)
have 4: $\vdash (h \vee g; ((\text{wpowerstar } (f \vee g)); h)) =$
 $(g; ((\text{wpowerstar } (f \vee g)); h) \vee h)$
by *fastforce*
show *?thesis*
by (*metis 1 2 3 4 int-eq*)
qed

lemma *SChopstar-troeger*:

$\vdash (\text{schopstar } (f \vee g)) \frown (h) =$
 $(\text{schopstar } f) \frown (g \frown ((\text{schopstar } (f \vee g)) \frown (h)) \vee (h))$
proof –
have 1: $\vdash (\text{schopstar } (f \vee g)) \frown (h) =$
 $((\text{schopstar } f) \vee (\text{schopstar } f) \frown (g \frown (\text{schopstar } (f \vee g)))) \frown (h)$
using *SChopstar-or-unfold*[of *f g*] **by** (*metis LeftSChopEqvSChop int-eq*)

have 2: $\vdash ((\text{schopstar } f) \vee (\text{schopstar } f) \frown (g \frown (\text{schopstar } (f \vee g)))) \frown (h) =$
 $((\text{schopstar } f) \frown h \vee ((\text{schopstar } f) \frown (g \frown (\text{schopstar } (f \vee g)))) \frown h)$
by (*simp add: OrSCHopEqv*)
have 3: $\vdash ((\text{schopstar } f) \frown (g \frown (\text{schopstar } (f \vee g)))) \frown h =$
 $(\text{schopstar } f) \frown (g \frown ((\text{schopstar } (f \vee g)) \frown (h)))$
by (*metis SCHopAssoc inteq-reflection*)
have 3: $\vdash ((\text{schopstar } f) \frown h \vee ((\text{schopstar } f) \frown (g \frown (\text{schopstar } (f \vee g)))) \frown h) =$
 $(\text{schopstar } f) \frown ((h) \vee g \frown ((\text{schopstar } (f \vee g)) \frown (h)))$
by (*metis 3 SCHopOrEqv inteq-reflection*)
have 4: $\vdash ((h) \vee g \frown ((\text{schopstar } (f \vee g)) \frown (h))) =$
 $(g \frown ((\text{schopstar } (f \vee g)) \frown (h)) \vee (h))$
by *fastforce*
show *?thesis*
by (*metis 1 2 3 4 int-eq*)
qed

lemma *SChopstarMore-troeger*:

$\vdash (\text{schopstar } (f \vee g)) \frown (h) =$
 $(\text{schopstar } f) \frown ((g \wedge \text{more}) \frown ((\text{schopstar } (f \vee g)) \frown (h)) \vee (h))$
using *SChopstar-troeger*[*of LIFT f \wedge more LIFT g \wedge more h*]
SChopstar-and-more[*of f*] *SChopstar-and-more*[*of g*]
SChopstar-and-more[*of LIFT (f \vee g)*]
SChopstarMore-or-and[*of f g*]
by (*metis inteq-reflection*)

lemma *WPowerstar-square*:

$\vdash (\text{wpowerstar } (f;f)) \longrightarrow (\text{wpowerstar } f)$

proof –

have 1: $\vdash (f;f);(\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f)$
by (*simp add: Chop-WPowerstar-Closure WPowerstar-ext*)
show *?thesis*
by (*simp add: 1 WPowerstar-induct-lvar-star*)

qed

lemma *SChopstar-square*:

$\vdash (\text{schopstar } (f \frown f)) \longrightarrow (\text{schopstar } f)$

proof –

have 1: $\vdash (f \frown f) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f)$
by (*metis Prop05 RightSCHopImpSCHop SCHopAssoc SCHopstar-1L SCHopstar-unfoldl-eq inteq-reflection*)
show *?thesis*
by (*simp add: 1 SCHopstar-induct-lvar-star*)

qed

lemma *WPowerstar-meyer-1*:

$\vdash (\text{empty} \vee f);(\text{wpowerstar } (f ; f)) = (\text{wpowerstar } f)$

proof –

have 1: $\vdash f;((\text{empty} \vee f);(\text{wpowerstar } (f ; f))) =$
 $f;(\text{empty};(\text{wpowerstar } (f ; f)) \vee f;(\text{wpowerstar } (f ; f)))$
by (*simp add: OrChopEqv RightChopEqvChop*)
have 2: $\vdash (\text{empty};(\text{wpowerstar } (f ; f)) \vee f;(\text{wpowerstar } (f ; f))) =$

$((\text{wpowerstar } (f ; f)) \vee f;(\text{wpowerstar } (f ; f)))$
by (*meson EmptyOrChopEqv OrChopEqv Prop04*)
have 3: $\vdash f;(\text{empty};(\text{wpowerstar } (f ; f)) \vee f;(\text{wpowerstar } (f ; f))) =$
 $f;((\text{wpowerstar } (f ; f)) \vee f;(\text{wpowerstar } (f ; f)))$
using 2 *RightChopEqvChop* **by** *blast*
have 4: $\vdash f;((\text{wpowerstar } (f ; f)) \vee f;(\text{wpowerstar } (f ; f))) \longrightarrow$
 $f;(\text{wpowerstar } (f ; f)) \vee (\text{wpowerstar } (f ; f))$
by (*metis ChopAssoc ChopOrImp Prop05 Prop08 int-eq int-iffD2 wpowerstar-unfoldl-eq*)
have 41: $\vdash f;((\text{empty} \vee f);(\text{wpowerstar } (f ; f))) \longrightarrow f;(\text{wpowerstar } (f;f) \vee f;\text{wpowerstar } (f;f))$
using *EmptyOrChopEqv*[of *f LIFT (wpowerstar (f ; f))*]
by (*metis 1 Prop11 inteq-reflection*)
have 42: $\vdash f;(\text{wpowerstar } (f;f) \vee f;\text{wpowerstar } (f;f)) = (f;\text{wpowerstar } (f;f) \vee f;(f;\text{wpowerstar } (f;f)))$
using *ChopOrEqv*[of *f LIFT wpowerstar (f;f) LIFT f;wpowerstar (f;f)*]
by *auto*
have 43: $\vdash (f;\text{wpowerstar } (f;f) \vee f;(f;\text{wpowerstar } (f;f))) =$
 $(f;\text{wpowerstar } (f;f) \vee (f;f);\text{wpowerstar } (f;f))$
using *ChopAssoc*[of *f f LIFT wpowerstar (f;f)*] **by** *auto*
have 44: $\vdash (f;\text{wpowerstar } (f;f) \vee (f;f);\text{wpowerstar } (f;f)) \longrightarrow ((\text{empty} \vee f);(\text{wpowerstar } (f ; f)))$
using *wpowerstar-unfoldl-eq*[of *LIFT (f;f)*]
using *EmptyOrChopEqv* **by** *fastforce*
have 5: $\vdash f;((\text{empty} \vee f);(\text{wpowerstar } (f ; f))) \longrightarrow$
 $((\text{empty} \vee f);(\text{wpowerstar } (f ; f)))$
by (*metis 1 3 42 43 44 int-eq*)
have 6: $\vdash \text{empty} \longrightarrow ((\text{empty} \vee f);(\text{wpowerstar } (f ; f)))$
using *EmptyOrChopEqv wpowerstar-unfoldl-eq* **by** *fastforce*
have 7: $\vdash \text{empty} \vee f;((\text{empty} \vee f);(\text{wpowerstar } (f ; f))) \longrightarrow$
 $((\text{empty} \vee f);(\text{wpowerstar } (f ; f)))$
using 5 6 *Prop02* **by** *blast*
have 8: $\vdash (\text{wpowerstar } f) \longrightarrow ((\text{empty} \vee f);(\text{wpowerstar } (f ; f)))$
using 7 *WPowerstar-induct-lvar-empty* **by** *blast*
have 9: $\vdash ((\text{empty} \vee f);(\text{wpowerstar } (f ; f))) \longrightarrow (\text{empty} \vee f);(\text{wpowerstar } f)$
using *WPowerstar-square*[of *f*]
using *RightChopImpChop* **by** *blast*
have 10: $\vdash (\text{empty} \vee f);(\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f)$
by (*meson Prop02 WPowerstarInductR WPowerstar-1R WPowerstar-ext WPowerstar-imp-empty*)
show *?thesis*
by (*meson 10 8 9 int-iffI lift-imp-trans*)
qed

lemma *SChopstar-meyer-1:*

$\vdash (\text{empty} \vee f) \frown (\text{schopstar } (f \frown f)) = (\text{schopstar } f)$

proof –

have 1: $\vdash f \frown ((\text{empty} \vee f) \frown (\text{schopstar } (f \frown f))) =$
 $f \frown (\text{empty} \frown (\text{schopstar } (f \frown f)) \vee f \frown (\text{schopstar } (f \frown f)))$
by (*simp add: OrSChopEqv RightSChopEqvSChop*)
have 2: $\vdash (\text{empty} \frown (\text{schopstar } (f \frown f)) \vee f \frown (\text{schopstar } (f \frown f))) =$
 $((\text{schopstar } (f \frown f)) \vee f \frown (\text{schopstar } (f \frown f)))$
by (*meson EmptyOrSChopEqv OrSChopEqv Prop04*)
have 3: $\vdash f \frown (\text{empty} \frown (\text{schopstar } (f \frown f)) \vee f \frown (\text{schopstar } (f \frown f))) =$
 $f \frown ((\text{schopstar } (f \frown f)) \vee f \frown (\text{schopstar } (f \frown f)))$

using 2 *RightSChopEqvSChop* **by** *blast*
have 4: $\vdash f \neg ((\text{schopstar } (f \neg f)) \vee f \neg (\text{schopstar } (f \neg f))) \longrightarrow$
 $(\text{schopstar } (f \neg f)) \vee f \neg (\text{schopstar } (f \neg f))$
using *SChopAssoc SChopOrEqv SChopstar-unfoldl-eq* **by** *fastforce*
have 41: $\vdash ((\text{schopstar } (f \neg f)) \vee f \neg (\text{schopstar } (f \neg f))) =$
 $((\text{empty} \vee f) \neg (\text{schopstar } (f \neg f)))$
by (*metis* 2 *OrSChopEqv inteq-reflection*)
have 5: $\vdash f \neg ((\text{empty} \vee f) \neg (\text{schopstar } (f \neg f))) \longrightarrow$
 $((\text{empty} \vee f) \neg (\text{schopstar } (f \neg f)))$
by (*metis* 4 41 *int-eq*)
have 6: $\vdash \text{empty} \longrightarrow ((\text{empty} \vee f) \neg (\text{schopstar } (f \neg f)))$
using *EmptyOrSChopEqv SChopstar-imp-empty* **by** *fastforce*
have 7: $\vdash \text{empty} \vee f \neg ((\text{empty} \vee f) \neg (\text{schopstar } (f \neg f))) \longrightarrow$
 $((\text{empty} \vee f) \neg (\text{schopstar } (f \neg f)))$
using 5 6 *Prop02* **by** *blast*
have 8: $\vdash (\text{schopstar } f) \longrightarrow ((\text{empty} \vee f) \neg (\text{schopstar } (f \neg f)))$
using 7 *SChopstar-induct-lvar-empty* **by** *blast*
have 9: $\vdash ((\text{empty} \vee f) \neg (\text{schopstar } (f \neg f))) \longrightarrow (\text{empty} \vee f) \neg (\text{schopstar } f)$
using *SChopstar-square[of f]*
using *RightSChopImpSChop* **by** *blast*
have 10: $\vdash (\text{empty} \vee f) \neg (\text{schopstar } f) \longrightarrow (\text{schopstar } f)$
by (*metis* *SChopstar-1L SChopstar-star2 inteq-reflection*)
show ?thesis
by (*meson* 10 8 9 *int-iffI lift-imp-trans*)
qed

lemma *SChopstarMore-meyer-1*:

$\vdash (\text{empty} \vee (f \wedge \text{more})) \neg (\text{schopstar } (f \neg f)) = (\text{schopstar } f)$

proof –

have 1: $\vdash (f \wedge \text{more}) \neg ((\text{empty} \vee (f \wedge \text{more})) \neg (\text{schopstar } (f \neg f))) =$
 $(f \wedge \text{more}) \neg (\text{empty} \neg (\text{schopstar } (f \neg f)) \vee (f \wedge \text{more}) \neg (\text{schopstar } (f \neg f)))$
by (*simp add: OrSChopEqv RightSChopEqvSChop*)
have 2: $\vdash (\text{empty} \neg (\text{schopstar } (f \neg f)) \vee (f \wedge \text{more}) \neg (\text{schopstar } (f \neg f))) =$
 $((\text{schopstar } (f \neg f)) \vee (f \wedge \text{more}) \neg (\text{schopstar } (f \neg f)))$
by (*meson EmptyOrSChopEqv OrSChopEqv Prop04*)
have 3: $\vdash (f \wedge \text{more}) \neg (\text{empty} \neg (\text{schopstar } (f \neg f)) \vee (f \wedge \text{more}) \neg (\text{schopstar } (f \neg f))) =$
 $(f \wedge \text{more}) \neg ((\text{schopstar } (f \neg f)) \vee (f \wedge \text{more}) \neg (\text{schopstar } (f \neg f)))$
using 2 *RightSChopEqvSChop* **by** *blast*
have 30: $\vdash ((f \wedge \text{more}) \neg (f \wedge \text{more})) \longrightarrow f \neg f$
by (*metis Prop11 Prop12 SChopImpSChop lift-and-com*)
have 31: $\vdash ((f \wedge \text{more}) \neg (f \wedge \text{more})) \neg (\text{schopstar } (f \neg f)) \longrightarrow (\text{schopstar } (f \neg f))$
by (*metis* 30 *AndSChopA Prop05 Prop10 SChopstar-unfoldl-eq int-eq lift-and-com*)
have 4: $\vdash (f \wedge \text{more}) \neg ((\text{schopstar } (f \neg f)) \vee (f \wedge \text{more}) \neg (\text{schopstar } (f \neg f))) \longrightarrow$
 $(f \wedge \text{more}) \neg (\text{schopstar } (f \neg f)) \vee (\text{schopstar } (f \neg f))$
using 31 *SChopAssoc SChopOrImp* **by** *fastforce*
have 5: $\vdash (f \wedge \text{more}) \neg ((\text{empty} \vee (f \wedge \text{more})) \neg (\text{schopstar } (f \neg f))) \longrightarrow$
 $((\text{empty} \vee (f \wedge \text{more})) \neg (\text{schopstar } (f \neg f)))$
by (*metis* (*no-types, opaque-lifting*) 4 *EmptyOrSChopEqv Prop11 int-simps(33)*
inteq-reflection lift-and-com lift-imp-trans)
have 6: $\vdash \text{empty} \longrightarrow ((\text{empty} \vee (f \wedge \text{more})) \neg (\text{schopstar } (f \neg f)))$

```

using EmptyOrSChopEqv SChopstar-unfoldl-eq by fastforce
have 7:  $\vdash \text{empty} \vee (f \wedge \text{more}) \frown ((\text{empty} \vee (f \wedge \text{more})) \frown (\text{schopstar } (f \frown f))) \longrightarrow$ 
 $((\text{empty} \vee (f \wedge \text{more})) \frown (\text{schopstar } (f \frown f)))$ 
using 5 6 Prop02 by blast
have 8:  $\vdash (\text{schopstar } f) \longrightarrow ((\text{empty} \vee (f \wedge \text{more})) \frown (\text{schopstar } (f \frown f)))$ 
using 7 SChopstarMore-induct-lvar-empty by blast
have 9:  $\vdash ((\text{empty} \vee (f \wedge \text{more})) \frown (\text{schopstar } (f \frown f))) \longrightarrow (\text{empty} \vee (f \wedge \text{more})) \frown (\text{schopstar } f)$ 
using SChopstar-square[of f]
using RightSChopImpSChop by blast
have 10:  $\vdash (\text{empty} \vee (f \wedge \text{more})) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f)$ 
by (metis SChopstar-1L SChopstar-and-more SChopstar-star2 inteq-reflection)
show ?thesis
by (meson 10 8 9 int-iffI lift-imp-trans)
qed

```

```

lemma WPowerstar-tc:
assumes  $\vdash f;f \longrightarrow f$ 
shows  $\vdash (\text{wpowerstar } f);f = f$ 
proof –
have 1:  $\vdash f \vee f;f \longrightarrow f$ 
using assms by fastforce
have 2:  $\vdash (\text{wpowerstar } f);f \longrightarrow f$ 
by (simp add: WPowerstar-inductL-var-equiv assms)
have 3:  $\vdash f \longrightarrow (\text{wpowerstar } f);f$ 
by (metis AndChopB EmptyChop Prop10 WPowerstar-imp-empty inteq-reflection)
show ?thesis
by (simp add: 2 3 Prop11)
qed

```

```

lemma SChopstar-tc:
assumes  $\vdash f \frown f \longrightarrow f$ 
shows  $\vdash (\text{schopstar } f) \frown (f \wedge \text{finite}) = (f \wedge \text{finite})$ 
proof –
have 1:  $\vdash f \vee f \frown f \longrightarrow f$ 
using assms by fastforce
have 2:  $\vdash (\text{schopstar } f) \frown (f \wedge \text{finite}) \longrightarrow f \wedge \text{finite}$ 
by (metis Prop12 SChopAndA SChopstar-1R SChopstar-finite SChopstar-induct-lvar assms lift-imp-trans)
have 3:  $\vdash f \wedge \text{finite} \longrightarrow (\text{schopstar } f) \frown (f \wedge \text{finite})$ 
by (metis EmptySChop LeftSChopImpSChop SChopstar-imp-empty inteq-reflection)
show ?thesis
by (simp add: 2 3 Prop11)
qed

```

```

lemma WPowerstar-tc-eq:
assumes  $\vdash f;f = f$ 
shows  $\vdash (\text{wpowerstar } f);f = f$ 
using assms
by (simp add: WPowerstar-induct-lvar-eq2)

```

```

lemma SChopstar-tc-eq:

```


assumes $\vdash f \frown f = f$
shows $\vdash (\text{schopstar } f) \frown (f \wedge \text{finite}) = (f \wedge \text{finite})$
using *assms*
by (*simp add: SChopstar-tc int-iffD1*)

lemma *WPowerstar-boffa-var*:
assumes $\vdash f;f \longrightarrow f$
shows $\vdash (\text{wpowerstar } f) = (\text{empty} \vee f)$
proof –
have 1: $\vdash (\text{wpowerstar } f) = (\text{empty} \vee (\text{wpowerstar } f);f)$
by (*metis WPowerstar-slide-var WPowerstarEqv int-eq*)
show ?thesis
using 1 *Prop06 WPowerstar-tc assms* **by** *blast*
qed

lemma *SChopstar-boffa-var*:
assumes $\vdash f \frown f \longrightarrow f$
shows $\vdash (\text{schopstar } f) = (\text{empty} \vee (f \wedge \text{finite}))$
proof –
have 1: $\vdash (\text{schopstar } f) = (\text{empty} \vee (\text{schopstar } f) \frown (f \wedge \text{finite}))$
by (*meson SChopstar-unfoldR SChopstar-unfoldr-eq int-iffD2 int-iffI*)
show ?thesis
using 1 *Prop06 SChopstar-tc assms* **by** *blast*
qed

lemma *WPowerstar-boffa*:
assumes $\vdash f;f = f$
shows $\vdash (\text{wpowerstar } f) = (\text{empty} \vee f)$
using *assms*
by (*simp add: WPowerstar-boffa-var int-iffD1*)

lemma *SChopstar-boffa*:
assumes $\vdash f \frown f = f$
shows $\vdash (\text{schopstar } f) = (\text{empty} \vee (f \wedge \text{finite}))$
using *assms*
by (*simp add: SChopstar-boffa-var int-iffD1*)

lemma *WPowerstar-sim2*:
assumes $\vdash h ; f \longrightarrow g ; h$
shows $\vdash h ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } g) ; h$
proof –
have 1: $\vdash (\text{wpowerstar } g) ; (h ; f) \longrightarrow (\text{wpowerstar } g) ; (g ; h)$
by (*simp add: RightChopImpChop assms*)
have 2: $\vdash (\text{wpowerstar } g) ; (g ; h) \longrightarrow (\text{wpowerstar } g) ; h$
by (*metis ChopAssoc LeftChopImpChop WPowerstar-1L WPowerstar-slide-var inteq-reflection*)
have 3: $\vdash (\text{wpowerstar } g) ; (h ; f) \longrightarrow (\text{wpowerstar } g) ; h$
using 1 2 *lift-imp-trans* **by** *blast*
have 4: $\vdash h \longrightarrow (\text{wpowerstar } g) ; h$
by (*metis AndChopB EmptyChop Prop10 WPowerstar-imp-empty inteq-reflection*)

have 5: $\vdash h \vee (\text{wpowerstar } g) ; (h ; f) \longrightarrow (\text{wpowerstar } g) ; h$
using 3 4 Prop02 **by** blast
show ?thesis
by (metis 5 ChopAssoc WPowerstarInductR inteq-reflection)
qed

lemma *SChopstarInductR*:
assumes $\vdash g \vee h \frown f \longrightarrow h$
shows $\vdash g \frown \text{schopstar } f \longrightarrow h$
proof –
have 1: $\vdash g \wedge \text{finite} \longrightarrow h$
using assms **by** fastforce
have 2: $\vdash h \frown f \longrightarrow h$
using assms **by** fastforce
have 3: $\vdash g \wedge \text{finite} \vee h \frown f \longrightarrow h$
using 1 2 **by** fastforce
have 4: $\vdash (g \wedge \text{finite}); \text{fpowerstar } f \longrightarrow h$
by (metis 1 2 ChopSChopdef FPowerstarInductR OrFiniteInf Prop02 Prop03 inteq-reflection)
show ?thesis
using FPowerstar-more-absorb[of f] 4
by (metis int-eq schop-d-def schopstar-d-def)
qed

lemma *SChopstar-sim2*:
assumes $\vdash h \frown f \longrightarrow g \frown h$
shows $\vdash h \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } g) \frown h$
proof –
have 1: $\vdash (\text{schopstar } g) \frown (h \frown (f)) \longrightarrow (\text{schopstar } g) \frown (g \frown h)$
by (simp add: RightSChopImpSChop assms)
have 2: $\vdash (\text{schopstar } g) \frown (g \frown h) \longrightarrow (\text{schopstar } g) \frown h$
by (metis (no-types, lifting) ChopAssoc LeftChopImpChop Prop10 SChopstar-1L SChopstar-finite
SChopstar-WPowerstar WPowerstar-slide-var inteq-reflection schop-d-def)
have 3: $\vdash (\text{schopstar } g) \frown (h \frown (f)) \longrightarrow (\text{schopstar } g) \frown h$
using 1 2 lift-imp-trans **by** blast
have 4: $\vdash h \longrightarrow (\text{schopstar } g) \frown h$
by (meson EmptySChop LeftSChopImpSChop Prop11 SChopstar-imp-empty lift-imp-trans)
have 5: $\vdash h \vee (\text{schopstar } g) \frown (h \frown (f)) \longrightarrow (\text{schopstar } g) \frown h$
using 3 4 Prop02 **by** blast
show ?thesis
by (metis 5 SChopAssoc SChopstarInductR inteq-reflection)
qed

lemma *SChopImpFinite*:
 $\vdash f \longrightarrow \text{finite} \implies \vdash g \frown f \longrightarrow \text{finite}$
by (metis DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty Prop10 SChopSFinExportA
SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection)

lemma *SChopstar-sim2-finite*:
assumes $\vdash h \frown f \longrightarrow g \frown h$
shows $\vdash h \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } g) \frown (h \wedge \text{finite})$

using *assms SChopstar-sim2*
by (*metis FiniteImportSChopRight Prop12 SChopImpFinite SChopstar-finite inteq-reflection*)

lemma *WPowerstar-inductr-var*:
assumes $\vdash g ; f \longrightarrow g$
shows $\vdash g ; (\text{wpowerstar } f) \longrightarrow g$
using *assms*
by (*simp add: WPowerstarInductR*)

lemma *SChopstar-inductr-var*:
assumes $\vdash g \frown f \longrightarrow g$
shows $\vdash g \frown (\text{schopstar } f) \longrightarrow g$
using *assms*
by (*simp add: SChopstarInductR*)

lemma *SChopstarMore-inductr-var*:
assumes $\vdash g \frown (f \wedge \text{more}) \longrightarrow g$
shows $\vdash g \frown (\text{schopstar } f) \longrightarrow g$
using *assms*
by (*metis SChopstar-and-more SChopstar-inductr-var inteq-reflection*)

lemma *SChopstar-finite-absorb*:
 $\vdash \text{schopstar } (f \wedge \text{finite}) = \text{schopstar } f$
by (*metis Prop10 SChopstar-ext SChopstar-finite SChopstar-WPowerstar int-eq lift-imp-trans*)

lemma *SChopstar-inductr-finite-var*:
assumes $\vdash g \frown (f \wedge \text{finite}) \longrightarrow g$
shows $\vdash g \frown (\text{schopstar } f) \longrightarrow g$
using *assms*
by (*metis SChopstar-finite-absorb SChopstar-inductr-var inteq-reflection*)

lemma *SChopstarMore-inductr-finite-var*:
assumes $\vdash g \frown ((f \wedge \text{more}) \wedge \text{finite}) \longrightarrow g$
shows $\vdash g \frown (\text{schopstar } f) \longrightarrow g$
using *assms*
by (*metis SChopstar-and-more SChopstar-inductr-finite-var inteq-reflection*)

lemma *WPowerstar-inductr-var-equiv*:
 $(\vdash g ; f \longrightarrow g) = (\vdash g ; (\text{wpowerstar } f) \longrightarrow g)$
proof –
have 1: $(\vdash g ; f \longrightarrow g) \implies (\vdash g ; (\text{wpowerstar } f) \longrightarrow g)$
by (*simp add: WPowerstar-inductr-var*)
have 2: $(\vdash g ; (\text{wpowerstar } f) \longrightarrow g) \implies (\vdash g ; f \longrightarrow g)$
by (*metis ChopAndB Prop10 WPowerstar-ext int-eq lift-imp-trans*)
show ?thesis **using** 1 2 **by** blast
qed

lemma *SChopstar-inductr-var-equiv*:
 $(\vdash g \frown (f \wedge \text{finite}) \longrightarrow g) = (\vdash g \frown (\text{schopstar } f) \longrightarrow g)$

proof –
have 1: $(\vdash g \frown (f \wedge \text{finite}) \longrightarrow g) \implies (\vdash g \frown (\text{schopstar } f) \longrightarrow g)$
by (*simp add: SChopstar-inductr-finite-var*)
have 2: $(\vdash g \frown (\text{schopstar } f) \longrightarrow g) \implies (\vdash g \frown (f \wedge \text{finite}) \longrightarrow g)$
by (*metis Prop10 Prop12 SChopAndB SChopstar-ext inteq-reflection*)
show ?thesis **using** 1 2 **by** blast
qed

lemma *SChopstarMore-inductr-var-equiv*:
 $(\vdash g \frown ((f \wedge \text{more}) \wedge \text{finite}) \longrightarrow g) = (\vdash g \frown (\text{schopstar } f) \longrightarrow g)$
using *SChopstar-inductr-var-equiv[of g LIFT f \wedge more]*
SChopstar-and-more[of f]
by (*metis inteq-reflection*)

lemma *WPowerstar-sim3*:
assumes $\vdash h ; f = g ; h$
shows $\vdash h ; (\text{wpowerstar } f) = (\text{wpowerstar } g) ; h$
using *assms*
by (*simp add: Prop11 WPowerstar-sim1 WPowerstar-sim2*)

lemma *SChopstar-sim3*:
assumes $\vdash h \frown f = g \frown h$
shows $\vdash h \frown (\text{schopstar } f) = (\text{schopstar } g) \frown (h \wedge \text{finite})$
using *SChopstar-sim1 SChopstar-sim2-finite*
by (*metis Prop11 assms*)

lemma *SChopstarMore-sim3*:
assumes $\vdash h \frown (f \wedge \text{more}) = (g \wedge \text{more}) \frown h$
shows $\vdash h \frown (\text{schopstar } f) = (\text{schopstar } g) \frown (h \wedge \text{finite})$
by (*metis SChopstar-and-more SChopstar-sim3 assms inteq-reflection*)

lemma *WPowerstar-sim4*:
assumes $\vdash f ; g \longrightarrow g ; f$
shows $\vdash (\text{wpowerstar } f);(\text{wpowerstar } g) \longrightarrow (\text{wpowerstar } g) ; (\text{wpowerstar } f)$
using *assms*
by (*simp add: WPowerstar-sim1 WPowerstar-sim2*)

lemma *SChopstar-sim4*:
assumes $\vdash f \frown g \longrightarrow g \frown f$
shows $\vdash (\text{schopstar } f) \frown (\text{schopstar } g) \longrightarrow (\text{schopstar } g) \frown (\text{schopstar } f)$
using *assms*
by (*metis Prop10 SChopstar-finite SChopstar-sim1 SChopstar-sim2 inteq-reflection*)

lemma *WPowerstar-inductr-eq*:
assumes $\vdash (h \vee g ; f) = g$
shows $\vdash h;(\text{wpowerstar } f) \longrightarrow g$
using *assms*
using *WPowerstarInductR int-iffD1* **by** blast

lemma *SChopstar-inductr-eq*:
assumes $\vdash (h \vee g \frown f) = g$
shows $\vdash h \frown (\text{schopstar } f) \longrightarrow g$
using *assms* **using** *SChopstarInductR int-iffD1* **by** *blast*

lemma *SChopstarMore-inductr-eq*:
assumes $\vdash (h \vee g \frown (f \wedge \text{more})) = g$
shows $\vdash h \frown (\text{schopstar } f) \longrightarrow g$
using *assms*
by (*metis SChopstar-and-more SChopstar-inductr-eq inteq-reflection*)

lemma *WPowerstar-inductr-var-eq*:
assumes $\vdash (g ; f) = g$
shows $\vdash g ; (\text{wpowerstar } f) \longrightarrow g$
using *assms*
using *WPowerstar-inductr-var int-iffD1* **by** *blast*

lemma *SChopstar-inductr-var-eq*:
assumes $\vdash (g \frown f) = g$
shows $\vdash g \frown (\text{schopstar } f) \longrightarrow g$
using *assms*
using *SChopstar-inductr-var int-iffD1* **by** *blast*

lemma *SChopstarMore-inductr-var-eq*:
assumes $\vdash (g \frown (f \wedge \text{more})) = g$
shows $\vdash g \frown (\text{schopstar } f) \longrightarrow g$
using *assms*
by (*simp add: SChopstarMore-inductr-var int-iffD1*)

lemma *WPowerstar-inductr-var-eq2*:
assumes $\vdash (g ; f) = g$
shows $\vdash g ; (\text{wpowerstar } f) = g$
using *assms*
by (*metis Prop11 RightChopImpChop WPowerstar-ext WPowerstar-inductr-var-eq inteq-reflection*)

lemma *SChopstar-inductr-var-eq2*:
assumes $\vdash (g \frown (f \wedge \text{finite})) = g$
shows $\vdash g \frown (\text{schopstar } f) = g$
using *assms*
by (*metis Prop11 RightSChopImpSChop SChopstar-ext SChopstar-inductr-finite-var inteq-reflection*)

lemma *SChopstarMore-inductr-var-eq2*:
assumes $\vdash (g \frown ((f \wedge \text{more}) \wedge \text{finite})) = g$
shows $\vdash g \frown (\text{schopstar } f) = g$
using *assms*
SChopstar-inductr-var-eq2[$\text{of } g \text{ LIFT } f \wedge \text{more}$]
SChopstar-and-more[$\text{of } f$]
by (*metis inteq-reflection*)

lemma *WPowerstar-bubble-sort*:

assumes $\vdash g ; f \longrightarrow f ; g$

shows $\vdash (\text{wpowerstar } (f \vee g)) = (\text{wpowerstar } f) ; (\text{wpowerstar } g)$

using *assms*

using *WPowerstar-church-rosser WPowerstar-sim4* **by** *blast*

lemma *SChopstar-bubble-sort*:

assumes $\vdash g \frown f \longrightarrow f \frown g$

shows $\vdash (\text{schopstar } (f \vee g)) = (\text{schopstar } f) \frown (\text{schopstar } g)$

using *assms* **by** (*simp add: SChopstar-church-rosser SChopstar-sim4*)

lemma *WPowerstar-indepence1*:

assumes $\vdash f ; g = \#False$

shows $\vdash (\text{wpowerstar } f);g = g$

using *assms*

by (*metis EmptyOrChopEqv WPowerstar-induct-lvar-eq2 WPowerstar-star2 assms int-eq int-simps(25)*)

lemma *SChopRightFalse*:

$\vdash f \frown \#False = \#False$

by (*simp add: intI itl-defs*)

lemma *SChopstar-indepence1*:

assumes $\vdash f \frown g = \#False$

shows $\vdash (\text{schopstar } f) \frown g = g$

proof –

have $0 : \vdash (\text{schopstar } f) = (\text{empty} \vee (\text{schopstar } f) \frown (f \wedge \text{finite}))$

by (*meson Prop11 SChopstar-unfoldr-eq*)

have $01 : \vdash (\text{schopstar } f) \frown (f \wedge \text{finite}) = ((\text{schopstar } f) \frown f) \wedge \text{finite}$

by (*meson FiniteImportSChopRight Prop04 lift-and-com*)

have $1 : \vdash (\text{empty} \vee (\text{schopstar } f) \frown (f \wedge \text{finite})) \frown g = (\text{empty} \frown g \vee ((\text{schopstar } f) \frown f) \frown (g))$

using 01 **unfolding** *schop-d-def*

by (*metis 0 EmptyImpFinite OrChopEqv Prop10 SChopstar-finite inteq-reflection chop-d-def*)

have $11 : \vdash ((\text{schopstar } f) \frown f) \frown (g) = ((\text{schopstar } f)) \frown (f \frown g)$

by (*meson Prop11 SChopAssoc*)

have $12 : \vdash \text{empty} \frown g = g$

by (*simp add: EmptySChop*)

have $2 : \vdash (\text{schopstar } f) \frown (f \frown g) = (\text{schopstar } f) \frown (\#False)$

by (*simp add: RightSChopEqvSChop assms*)

have $3 : \vdash (\text{schopstar } f) \frown (\#False) = \#False$

by (*simp add: SChopRightFalse*)

show *?thesis*

by (*metis 0 1 11 12 3 assms int-simps(25) inteq-reflection*)

qed

lemma *WPowerstar-indepence2*:

assumes $\vdash f ; g = \#False$

shows $\vdash f ; (\text{wpowerstar } g) = f$
using *assms WPowerstar-inductr-var-eq2[of f g] WPowerstar-star2[of g]*
by (*metis ChopEmpty RightChopImpChop WPowerstar-imp-empty WPowerstar-inductr-var-equiv int-eq int-iffI int-simps(11)*)

lemma *SChopstar-indepence2:*

assumes $\vdash f \frown g = \#False$

shows $\vdash f \frown (\text{schopstar } g) = (f \wedge \text{finite})$

proof –

have 1: $\vdash f \frown (\text{schopstar } g) = ((f \wedge \text{finite}) \vee (f \frown g) \frown (\text{schopstar } g))$

by (*metis ChopEmpty FPowerstarEqv FPowerstar-more-absorb SChopAssoc SChopOrEqv int-eq chop-d-def chopstar-d-def*)

have 2: $\vdash (f \frown g) \frown (\text{schopstar } g) = \#False$

by (*metis AndInfChopEqvAndInf assms int-eq int-simps(19) chop-d-def*)

show ?thesis

by (*metis 1 2 int-simps(25) inteq-reflection*)

qed

lemma *WPowerstar-lazy-comm:*

$(\vdash g;f \longrightarrow f;(\text{wpowerstar } (f \vee g)) \vee g) =$

$(\vdash g;(\text{wpowerstar } f) \longrightarrow f;(\text{wpowerstar } (f \vee g)) \vee g)$

proof

assume 1: $\vdash g;f \longrightarrow f;(\text{wpowerstar } (f \vee g)) \vee g$

show $\vdash g;(\text{wpowerstar } f) \longrightarrow f;(\text{wpowerstar } (f \vee g)) \vee g$

proof –

have 2: $\vdash (f;(\text{wpowerstar } (f \vee g)) \vee g);f =$
 $((f;(\text{wpowerstar } (f \vee g)));f \vee g;f)$

by (*simp add: OrChopEqv*)

have 3: $\vdash ((f;(\text{wpowerstar } (f \vee g)));f \vee g;f) \longrightarrow$
 $((f;(\text{wpowerstar } (f \vee g)));f \vee (f;(\text{wpowerstar } (f \vee g))) \vee g)$

using 1 **by** *fastforce*

have 4: $\vdash (f;(\text{wpowerstar } (f \vee g)));f \longrightarrow (f;(\text{wpowerstar } (f \vee g)))$

by (*metis ChopAssoc RightChopImpChop WPowerstar-subdist-var-1 WPowerstar-trans-eq inteq-reflection*)

have 5: $\vdash ((f;(\text{wpowerstar } (f \vee g)));f \vee (f;(\text{wpowerstar } (f \vee g))) \vee g) \longrightarrow$
 $((f;(\text{wpowerstar } (f \vee g))) \vee g)$

using 4 **by** *fastforce*

have 6: $\vdash g \vee ((f;(\text{wpowerstar } (f \vee g))) \vee g) \longrightarrow ((f;(\text{wpowerstar } (f \vee g))) \vee g)$

by *fastforce*

show ?thesis **using** *WPowerstarInductR[of g LIFT (f;wpowerstar (f ∨ g) ∨ g) f]*

using 2 3 5 **by** *fastforce*

qed

next

assume 7: $\vdash g;(\text{wpowerstar } f) \longrightarrow f;(\text{wpowerstar } (f \vee g)) \vee g$

show $\vdash g;f \longrightarrow f;(\text{wpowerstar } (f \vee g)) \vee g$

proof –

have 80: $\vdash f;(\text{wpowerstar } f) = (f \vee f;(\text{wpowerstar } f))$

by (*meson ChopEmpty Prop02 Prop05 Prop11 RightChopImpChop WPowerstar-imp-empty lift-imp-trans*)

have 8: $\vdash (\text{wpowerstar } f) = (\text{empty} \vee f \vee f;(\text{wpowerstar } f))$

```

  by (meson 80 Prop06 WPowerstarEqv)
have 9:  $\vdash g ; (wpowerstar\ f) = (g ; empty \vee g ; f \vee g ; (f ; (wpowerstar\ f)))$ 
  by (metis 8 ChopOrEqv int-eq)
show ?thesis
  using 7 9 by fastforce
qed
qed

lemma SChopstar-lazy-comm:
( $\vdash g \frown (f \wedge finite) \longrightarrow f \frown (schopstar\ (f \vee g)) \vee g =$ 
 $(\vdash g \frown (schopstar\ f) \longrightarrow f \frown (schopstar\ (f \vee g)) \vee g)$ 
proof
  assume 1:  $\vdash g \frown (f \wedge finite) \longrightarrow f \frown (schopstar\ (f \vee g)) \vee g$ 
  show  $\vdash g \frown (schopstar\ f) \longrightarrow f \frown (schopstar\ (f \vee g)) \vee g$ 
  proof -
    have 2:  $\vdash (f \frown (schopstar\ (f \vee g)) \vee g) \frown (f \wedge finite) =$ 
       $((f \frown (schopstar\ (f \vee g))) \frown (f \wedge finite) \vee g \frown (f \wedge finite))$ 
    by (simp add: OrSChopEqv)
    have 3:  $\vdash ((f \frown (schopstar\ (f \vee g))) \frown (f \wedge finite) \vee g \frown (f \wedge finite)) \longrightarrow$ 
       $((f \frown (schopstar\ (f \vee g))) \frown (f \wedge finite) \vee (f \frown (schopstar\ (f \vee g))) \vee g)$ 
    using 1
    using SChopAndA by fastforce
    have 4:  $\vdash (f \frown (schopstar\ (f \vee g))) \frown (f \wedge finite) \longrightarrow (f \frown (schopstar\ (f \vee g)))$ 
    using SChopstar-subdist-var-1
    by (metis Prop11 RightSChopImpSChop SChopAssoc SChop-SChopstar-Closure SChopstardef lift-imp-trans)
    have 5:  $\vdash ((f \frown (schopstar\ (f \vee g))) \frown (f \wedge finite) \vee (f \frown (schopstar\ (f \vee g))) \vee g) \longrightarrow$ 
       $((f \frown (schopstar\ (f \vee g))) \vee g)$ 
    using 4 by fastforce
    have 6:  $\vdash g \vee ((f \frown (schopstar\ (f \vee g))) \vee g) \longrightarrow ((f \frown (schopstar\ (f \vee g))) \vee g)$ 
    by fastforce
    show ?thesis
    by (metis (mono-tags, lifting) 3 5 OrSChopEqv Prop03 SChopstar-inductr-var-equiv int-eq lift-imp-trans)

  qed
next
  assume 7:  $\vdash g \frown (schopstar\ f) \longrightarrow f \frown (schopstar\ (f \vee g)) \vee g$ 
  show  $\vdash g \frown (f \wedge finite) \longrightarrow f \frown (schopstar\ (f \vee g)) \vee g$ 
  proof -
    have 8:  $\vdash (schopstar\ f) = (empty \vee (f \wedge finite) \vee f \frown (schopstar\ f))$ 
    by (meson AndSChopA Prop02 Prop05 Prop08 SChopstarMore-unfoldl-eq SChopstar-1L SChopstar-ext
      SChopstar-imp-empty int-iffD2 int-iffI)
    have 9:  $\vdash g \frown (schopstar\ f) = (g \frown empty \vee g \frown (f \wedge finite) \vee g \frown (f \frown (schopstar\ f)))$ 
    by (metis 8 SChopOrEqv inteq-reflection)
    show ?thesis
    using 7 9 by fastforce
  qed
qed

```


2.5.6 WPowerstar

lemma *WPowerstar-and-inf*:

$\vdash (wpowerstar (f \wedge inf)) = (empty \vee (f \wedge inf))$

by (*simp add: AndInfChopEqvAndInf WPowerstar-boffa*)

lemma *WPowerstar-chop-and-finite-inf*:

$\vdash (wpowerstar f) = (wpowerstar (f \wedge finite));(wpowerstar (f \wedge inf))$

by (*metis AndInfChopEqvAndInf OrFiniteInf WPowerstar-denest-var inteq-reflection*)

lemma *WPowerstar-SChopstar*:

$\vdash (wpowerstar f) = (schopstar f);(empty \vee (f \wedge inf))$

by (*metis SChopstar-WPowerstar WPowerstar-and-inf WPowerstar-chop-and-finite-inf int-eq*)

lemma *WPowerstar-empty*:

$\vdash (wpowerstar empty) = empty$

by (*simp add: WPowerstar-subid*)

lemma *WPowerstar-more*:

$\vdash (wpowerstar more)$

by (*metis ChopImpDi DiMoreEqvMore TrueW WPowerstar-boffa-var empty-d-def int-simps(29) inteq-reflection*)

lemma *WPowerstar-false*:

$\vdash (wpowerstar \#False) = empty$

by (*simp add: WPowerstar-subid*)

lemma *WPowerstar-true*:

$\vdash (wpowerstar \#True)$

by (*metis MP TrueW WPowerstar-ext*)

lemma *WPowerstar-inf*:

$\vdash (wpowerstar inf) = (empty \vee inf)$

by (*simp add: InfChopInfEqvInf WPowerstar-boffa*)

lemma *WPowerstar-finite*:

$\vdash (wpowerstar finite) = finite$

by (*meson EmptyImpFinite FiniteChopFiniteEqvFinite Prop02 Prop11 WPowerstar-ext WPowerstar-induct-lvar-emp*)

lemma *WPowerstar-imp-finite*:

assumes $\vdash f \longrightarrow finite$

shows $\vdash wpowerstar f \longrightarrow finite$

using *assms*

by (*metis WPowerstar-finite WPowerstar-iso inteq-reflection*)

lemma *WPowerstar-and-empty*:

$\vdash (wpowerstar (f \wedge empty)) = empty$

by (*metis AndEmptyChopAndEmptyEqvAndEmpty Prop11 Prop12 WPowerstar-subid inteq-reflection*)

lemma *WPowerstarIntro*:
assumes $\vdash f \longrightarrow (g \wedge \text{more}); f$
shows $\vdash f \wedge \text{finite} \longrightarrow \text{wpowerstar } g$
proof –
have 14: $\vdash \text{wpowerstar } g = (\text{empty} \vee (g \wedge \text{more}); (\text{wpowerstar } g))$
using *WPowerstar-more-absorb*[of g] *WPowerstarEqv*[of *LIFT* ($g \wedge \text{more}$)]
by (*metis int-eq*)
have 15: $\vdash (\neg(\text{wpowerstar } g)) = (\text{more} \wedge \neg((g \wedge \text{more}); \text{wpowerstar } g))$
using 14 **unfolding** *empty-d-def* **by** *fastforce*
have 20: $\vdash f \wedge \neg(\text{wpowerstar } g) \longrightarrow (g \wedge \text{more}); f \wedge \neg((g \wedge \text{more}); \text{wpowerstar } g)$
using *assms 15* **by** *fastforce*
have 21: $\vdash g \wedge \text{more} \longrightarrow \text{more}$
by *auto*
from 20 21 **show** ?thesis **using** *ChopContraB*[of f *LIFT* $\text{wpowerstar } g$ *LIFT* ($g \wedge \text{more}$)]
by *blast*
qed

lemma *WPowerstarIntroMore*:
assumes $\vdash f \wedge \text{more} \longrightarrow (g \wedge \text{more}); f$
shows $\vdash f \wedge \text{finite} \longrightarrow \text{wpowerstar } g$
proof –
have 14: $\vdash \text{wpowerstar } g = (\text{empty} \vee (g \wedge \text{more}); (\text{wpowerstar } g))$
using *WPowerstar-more-absorb*[of g] *WPowerstarEqv*[of *LIFT* ($g \wedge \text{more}$)]
by (*metis int-eq*)
have 15: $\vdash (\neg(\text{wpowerstar } g)) = (\text{more} \wedge \neg((g \wedge \text{more}); \text{wpowerstar } g))$
using 14 **unfolding** *empty-d-def* **by** *fastforce*
have 20: $\vdash f \wedge \neg(\text{wpowerstar } g) \longrightarrow (g \wedge \text{more}); f \wedge \neg((g \wedge \text{more}); \text{wpowerstar } g)$
using *assms 15* **by** *fastforce*
have 21: $\vdash g \wedge \text{more} \longrightarrow \text{more}$
by *auto*
from 20 21 **show** ?thesis **using** *ChopContraB*[of f *LIFT* $\text{wpowerstar } g$ *LIFT* ($g \wedge \text{more}$)]
by *blast*
qed

lemma *Chopstar-WPowerstar*:
 $\vdash \text{chopstar } f = \text{wpowerstar } (f \wedge \text{more})$
by (*metis FPowerstar-WPowerstar WPowerstar-and-inf WPowerstar-chop-and-finite-inf chopstar-d-def*
inteq-reflection powerstar-d-def)

lemma *Chopstar-FPowerstar*:
 $\vdash \text{chopstar } f = (\text{fpowerstar } f); (\text{empty} \vee (f \wedge \text{inf}))$
by (*metis Chopstar-WPowerstar FPowerstar-WPowerstar WPowerstar-and-inf WPowerstar-chop-and-finite-inf*
WPowerstar-more-absorb inteq-reflection)

lemma *SChopstar-WPowerstar-more*:
 $\vdash \text{schopstar } f = \text{wpowerstar } ((f \wedge \text{more}) \wedge \text{finite})$
by (*simp add: FPowerstar-WPowerstar schopstar-d-def*)

lemma *SChopstar-FPowerstar*:

$\vdash \text{schopstar } f = (\text{fpowerstar } f)$

by (*simp add: FPowerstar-more-absorb chopstar-d-def*)

lemma *WPowerstar-skip-finite*:

$\vdash (\text{wpowerstar skip}) = \text{finite}$

by (*metis EmptyImpFinite EmptyNextInducta Prop02 SkipChopFiniteImpFinite WPowerstar-1L WPowerstar-imp-empty WPowerstar-induct-lvar-empty int-iffI next-d-def*)

lemma *SPSEqvEmptyOrSChopSPS*:

$\vdash \text{fpowerstar } f = (\text{empty} \vee f \frown \text{fpowerstar } f)$

by (*simp add: FPowerstarEqv chop-d-def*)

lemma *ChopstarEqvPowerstar*:

$\vdash f^* = \text{powerstar } f$

by (*metis Chopstar-FPowerstar powerstar-d-def*)

lemma *ChopplusCommute*:

$\vdash f;f^* = f^*;f$

by (*metis Chopstar-WPowerstar WPowerstar-more-absorb WPowerstar-slide-var int-eq*)

lemma *SChopplusCommute*:

$\vdash f \frown \text{schopstar } f = \text{schopstar } f \frown (f \wedge \text{finite})$

by (*meson Prop04 SChopstar-sim3 int-simps(27)*)

lemma *CSEqvOrChopCSB*:

$\vdash f^* = (\text{empty} \vee (f^*;f))$

by (*metis ChopplusCommute ChopstarEqvPowerstar PowerstarEqvSem intI inteq-reflection*)

lemma *SCSEqvOrChopSCSB*:

$\vdash \text{schopstar } f = (\text{empty} \vee (\text{schopstar } f \frown (f \wedge \text{finite})))$

by (*metis SChopplusCommute SChopstar-FPowerstar SPSEqvEmptyOrSChopSPS inteq-reflection*)

lemma *CSChopCS*:

$\vdash f^* ; f^* = f^*$

by (*metis Chopstar-WPowerstar WPowerstar-trans-eq inteq-reflection*)

lemma *SCSSChopSCS*:

$\vdash \text{schopstar } f \frown \text{schopstar } f = \text{schopstar } f$

by (*metis SChopstar-imp-empty SChopstar-invol SChopstar-sup-id-star1 inteq-reflection*)

lemma *CSCSEqvCS*:

$\vdash (f^*)^* = f^*$

by (*metis (no-types, lifting) Chopstar-WPowerstar WPowerstar-invol WPowerstar-more-absorb int-eq*)

lemma *ChopPlusEqvOrChopChopPlus*:

$\vdash (f;f^*) = (f \vee f; (f;f^*))$

by (*metis CSeqvOrChopCSB ChopEmpty ChopOrEqv ChopplusCommute inteq-reflection*)

lemma *SChopPlusEqvOrSChopSChopPlus*:

$\vdash (f \frown \text{schopstar } f) = ((f \wedge \text{finite}) \vee f \frown (f \frown \text{schopstar } f))$

by (*metis ChopEmpty SChopOrEqvRule SChopstar-WPowerstar WPowerstarEqv inteq-reflection schop-d-def*)

lemma *ChopPlusEqv*:

$\vdash (f;f^*) = (f \vee (f \wedge \text{more}); (f;f^*))$

using *ChopplusCommute*[of *f*]

by (*metis ChopAssoc ChopstarEqv EmptyOrChopEqv int-eq*)

lemma *SChopPlusEqv*:

$\vdash (f \frown \text{schopstar } f) = ((f \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f))$

by (*metis (no-types, opaque-lifting) ChopAssoc EmptyOrChopEqv Prop10 SChopstarEqv SChopplusCommute SChopstar-finite inteq-reflection schop-d-def*)

lemma *CSIntro*:

assumes $\vdash f \longrightarrow (g \wedge \text{more}); f$

shows $\vdash f \wedge \text{finite} \longrightarrow g^*$

by (*metis Chopstar-WPowerstar WPowerstarIntro WPowerstar-more-absorb assms inteq-reflection*)

lemma *CSIntroMore*:

assumes $\vdash f \wedge \text{more} \longrightarrow (g \wedge \text{more}); f$

shows $\vdash f \wedge \text{finite} \longrightarrow g^*$

by (*metis Chopstar-WPowerstar WPowerstarIntroMore WPowerstar-more-absorb assms inteq-reflection*)

lemma *SCSIntro*:

assumes $\vdash f \longrightarrow (g \wedge \text{more}) \frown f$

shows $\vdash f \wedge \text{finite} \longrightarrow \text{schopstar } g$

proof –

have 1: $\vdash ((g \wedge \text{more}) \wedge \text{finite}) = ((g \wedge \text{finite}) \wedge \text{more})$

by *auto*

show *?thesis*

using *assms SChopstar-WPowerstar*[of *g*] *WPowerstarIntro*[of *f LIFT (g \wedge finite)*] 1

by (*metis inteq-reflection schop-d-def*)

qed

lemma *SCSIntroMore*:

assumes $\vdash f \wedge \text{more} \longrightarrow (g \wedge \text{more}) \frown f$

shows $\vdash f \wedge \text{finite} \longrightarrow \text{schopstar } g$

proof –

have 1: $\vdash ((g \wedge \text{more}) \wedge \text{finite}) = ((g \wedge \text{finite}) \wedge \text{more})$

by *auto*

show *?thesis*

using *assms SChopstar-WPowerstar*[of *g*] *WPowerstarIntroMore*[of *f LIFT (g \wedge finite)*] 1

by (metis inteq-reflection chop-d-def)
qed

lemma CSElim:

assumes $\vdash \text{empty} \longrightarrow g$
 $\vdash (f \wedge \text{more}); g \longrightarrow g$
shows $\vdash f^* \longrightarrow g$

using assms

by (metis Chopstar-WPowerstar Prop02 WPowerstar-induct-lvar-empty inteq-reflection)

lemma SCSElim:

assumes $\vdash \text{empty} \longrightarrow g$
 $\vdash (f \wedge \text{more}) \frown g \longrightarrow g$
shows $\vdash \text{schopstar } f \longrightarrow g$

using assms

by (metis Prop02 SChopstar-WPowerstar-more WPowerstar-induct-lvar-empty inteq-reflection chop-d-def)

lemma CSElimWithoutMore:

assumes $\vdash \text{empty} \longrightarrow g$
 $\vdash f; g \longrightarrow g$
shows $\vdash f^* \longrightarrow g$

using assms

by (metis AndChopA CSElim Prop10 Prop12 int-eq)

lemma SCSElimWithoutMore:

assumes $\vdash \text{empty} \longrightarrow g$
 $\vdash f \frown g \longrightarrow g$
shows $\vdash \text{schopstar } f \longrightarrow g$

using assms

by (metis Prop02 SChopstar-WPowerstar WPowerstar-induct-lvar-empty inteq-reflection chop-d-def)

lemma WPowerstarChopEqvChopOrRule:

assumes $\vdash f = ((\text{wpowerstar } g); h)$
shows $\vdash f = ((g; f) \vee h)$

proof -

have 1: $\vdash f = ((\text{wpowerstar } g); h)$ using assms by auto
have 2: $\vdash (\text{wpowerstar } g) = (\text{empty} \vee (g; (\text{wpowerstar } g)))$ by (simp add: WPowerstarEqv)
hence 3: $\vdash (\text{wpowerstar } g); h = (h \vee ((g; (\text{wpowerstar } g)); h))$ by (rule EmptyOrChopEqvRule)
have 4: $\vdash (g; (\text{wpowerstar } g)); h = g; ((\text{wpowerstar } g); h)$ by (meson ChopAssoc Prop11)
hence 41: $\vdash (\text{wpowerstar } g); h = (h \vee g; ((\text{wpowerstar } g); h))$ using 3 by fastforce
have 5: $\vdash g; f = g; ((\text{wpowerstar } g); h)$ using 1 by (rule RightChopEqvChop)
hence 6: $\vdash ((\text{wpowerstar } g); h) = (h \vee g; f)$ using 41 by fastforce
hence 61: $\vdash ((\text{wpowerstar } g); h) = ((g; f) \vee h)$ by auto
from 1 61 show ?thesis by fastforce

qed

lemma CSChopEqvChopOrRule:

assumes $\vdash f = (g^*; h)$
shows $\vdash f = ((g; f) \vee h)$

using *assms*

by (*metis Chopstar-WPowerstar WPowerstarChopEqvChopOrRule WPowerstar-more-absorb int-eq*)

lemma *SCSChopEqvSChopOrRule*:

assumes $\vdash f = (\text{schopstar } g \frown h)$

shows $\vdash f = ((g \frown f) \vee h)$

using *assms*

by (*metis (no-types, lifting) FPowerstar-WPowerstar FPowerstar-more-absorb Prop10 SChopstar-finite WPowerstarChopEqvChopOrRule int-eq schop-d-def schopstar-d-def*)

lemma *WPowerstarChopIntroRule*:

assumes $\vdash f \wedge \neg h \longrightarrow g; f$

$\vdash g \longrightarrow \text{more}$

shows $\vdash f \wedge \text{finite} \longrightarrow (\text{wpowerstar } g); h$

proof –

have 1: $\vdash f \wedge \neg h \longrightarrow g; f$

using *assms* **by** *blast*

have 2: $\vdash g \longrightarrow \text{more}$

using *assms* **by** *blast*

hence 3: $\vdash g \longrightarrow g \wedge \text{more}$

by *auto*

hence 4: $\vdash g; f \longrightarrow (g); f$

by *auto*

have 5: $\vdash f \longrightarrow (g); f \vee h$

using 1 4 **by** *fastforce*

have 6: $\vdash \text{wpowerstar } g = (\text{empty} \vee g; \text{wpowerstar } g)$

by (*simp add: WPowerstarEqv*)

hence 7: $\vdash (\text{wpowerstar } g); h = (h \vee (g; \text{wpowerstar } g); h)$

by (*rule EmptyOrChopEqvRule*)

have 8: $\vdash (g; \text{wpowerstar } g); h = g; (\text{wpowerstar } g; h)$

by (*meson ChopAssoc Prop11*)

have 9: $\vdash (\text{wpowerstar } g); h = (h \vee g; (\text{wpowerstar } g; h))$

using 7 8 **by** *fastforce*

have 10: $\vdash f \wedge \neg (\text{wpowerstar } g; h) \longrightarrow (g); f \wedge \neg ((g); (\text{wpowerstar } g; h))$

using 5 9 **by** *fastforce*

have 11: $\vdash g \wedge \text{more} \longrightarrow \text{more}$

by *fastforce*

from 10 11 **show** *?thesis* **using** *ChopContraB* **using** 2 **by** *blast*

qed

lemma *CSChopIntroRule*:

assumes $\vdash f \wedge \neg h \longrightarrow g; f$

$\vdash g \longrightarrow \text{more}$

shows $\vdash f \wedge \text{finite} \longrightarrow g^*; h$

using *assms*

by (*metis Chopstar-WPowerstar Prop10 WPowerstarChopIntroRule inteq-reflection*)

lemma *SCSChopIntroRule*:
assumes $\vdash f \wedge \neg h \longrightarrow g \smallfrown f$
 $\vdash g \longrightarrow \text{more}$
shows $\vdash f \wedge \text{finite} \longrightarrow \text{schopstar } g \smallfrown h$
using *assms SChopstar-WPowerstar*[of *g*] *WPowerstarChopIntroRule*[of *f h LIFT (g \wedge finite)*]
unfolding *schop-d-def*
by (*metis Prop05 Prop07 Prop10 SChopstar-finite finite-d-def inteq-reflection*)

lemma *DiamondAndEmptyEqvAndEmpty*:
 $\vdash (\Diamond f \wedge \text{empty}) = (f \wedge \text{empty})$
by (*metis ChopAndEmptyEqvEmptyChopEmpty EmptyChop FiniteAndEmptyEqvEmpty int-eq sometimes-d-def*)

lemma *InitAndEmptyEqvAndEmpty*:
 $\vdash ((\text{init } w) \wedge \text{empty}) = (w \wedge \text{empty})$
proof –
have 1: $\vdash ((\text{init } w) \wedge \text{empty}) = ((w \wedge \text{empty}); \# \text{True} \wedge \text{empty})$
by (*metis init-d-def int-eq lift-and-com*)
have 2: $\vdash ((w \wedge \text{empty}); \# \text{True} \wedge \text{empty}) = (w \wedge \text{empty}); (\# \text{True} \wedge \text{empty})$
by (*meson AndChopA ChopAndA ChopAndEmptyEqvEmptyChopEmpty lift-imp-trans Prop11 Prop12*)
have 3: $\vdash (w \wedge \text{empty}); (\# \text{True} \wedge \text{empty}) = (w \wedge \text{empty}); \text{empty}$
using *RightChopEqvChop* **by** *fastforce*
have 4: $\vdash (w \wedge \text{empty}); \text{empty} = (w \wedge \text{empty})$
using *ChopEmpty* **by** *blast*
from 1 2 3 4 **show** *?thesis* **by** *fastforce*
qed

lemma *InitAndNotBoxInitImpNotEmpty*:
 $\vdash \text{init } w \wedge \neg(\Box(\text{init } w)) \longrightarrow \neg \text{empty}$
proof –
have 1: $\vdash ((\text{init } w) \wedge \text{empty}) = (w \wedge \text{empty})$
by (*rule InitAndEmptyEqvAndEmpty*)
have 2: $\vdash (\neg(\Box(\text{init } w)) \wedge \text{empty}) = (\Diamond(\neg(\text{init } w)) \wedge \text{empty})$
by (*auto simp: always-d-def*)
have 3: $\vdash (\Diamond(\neg(\text{init } w)) \wedge \text{empty}) = (\neg(\text{init } w) \wedge \text{empty})$
by (*simp add: DiamondAndEmptyEqvAndEmpty*)
have 4: $\vdash (\neg(\text{init } w)) = (\text{init }(\neg w))$
using *Initprop(2)* **by** *blast*
have 5: $\vdash (\neg(\text{init } w) \wedge \text{empty}) = (\neg w \wedge \text{empty})$
using 4 *InitAndEmptyEqvAndEmpty* **by** (*metis inteq-reflection*)
have 6: $\vdash (\neg(\Box(\text{init } w)) \wedge \text{empty}) = (\neg w \wedge \text{empty})$
using 2 3 5 **by** *fastforce*
have 7: $\vdash \neg(\text{init } w \wedge \neg(\Box(\text{init } w)) \wedge \text{empty})$
using 1 6 **by** *fastforce*
from 7 **show** *?thesis* **by** *auto*
qed

lemma *BoxImpTrueChopAndEmpty*:
 $\vdash \Box f \longrightarrow \# \text{True}; (f \wedge \text{empty})$
using *BoxAndChopImport Finprop(3)* **by** *fastforce*

lemma *BoxInitAndMoreImpBoxInitAndMoreAndFinInit:*

$\vdash \Box (init\ w) \wedge more \longrightarrow (\Box (init\ w) \wedge more) \wedge fin\ (init\ w)$

proof –

have 1: $\vdash fin\ (init\ w) = \#True ; (init\ w \wedge empty)$ **using** *FinEqvTrueChopAndEmpty* **by** *blast*

have 2: $\vdash \Box (init\ w) \longrightarrow \#True ; (init\ w \wedge empty)$ **by** (rule *BoxImpTrueChopAndEmpty*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *BoxImpTrueSChopAndEmpty:*

$\vdash \Box f \wedge finite \longrightarrow \#True \frown (f \wedge empty)$

by (metis *BoxAndSChopImport DiamondEmptyEqvFinite TrueSChopEqvDiamond inteq-reflection*)

lemma *BoxInitAndMoreImpBoxInitAndMoreAndSFinInit:*

$\vdash \Box (init\ w) \wedge more \wedge finite \longrightarrow (\Box (init\ w) \wedge more) \wedge sfin\ (init\ w)$

proof –

have 1: $\vdash sfin\ (init\ w) = \#True \frown (init\ w \wedge empty)$ **using** *SFinEqvTrueSChopAndEmpty* **by** *blast*

have 2: $\vdash \Box (init\ w) \wedge finite \longrightarrow \#True \frown (init\ w \wedge empty)$ **by** (rule *BoxImpTrueSChopAndEmpty*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *CSImpBox:*

assumes $\vdash f \longrightarrow empty \vee ((\Box (init\ w) \wedge more) \wedge finite) ; f$

shows $\vdash (init\ w \wedge f) \wedge finite \longrightarrow \Box (init\ w)$

proof –

have 1: $\vdash f \longrightarrow empty \vee ((\Box (init\ w) \wedge more) \wedge finite) ; f$
using *assms* **by** *auto*

have 2: $\vdash init\ w \wedge \neg(\Box (init\ w)) \longrightarrow \neg empty$
by (rule *InitAndNotBoxInitImpNotEmpty*)

have 3: $\vdash init\ w \wedge f \wedge \neg(\Box (init\ w)) \longrightarrow ((\Box (init\ w) \wedge more) \wedge finite) ; f$
using 1 2 **by** *fastforce*

have 4: $\vdash \Box (init\ w) \wedge more \longrightarrow (\Box (init\ w) \wedge more) \wedge fin\ (init\ w)$
by (rule *BoxInitAndMoreImpBoxInitAndMoreAndFinInit*)

have 41: $\vdash (\Box (init\ w) \wedge more) \wedge finite \longrightarrow$
 $((\Box (init\ w) \wedge more) \wedge finite) \wedge fin\ (init\ w)$
using 4 **by** *auto*

hence 5: $\vdash ((\Box (init\ w) \wedge more) \wedge finite) ; f \longrightarrow$
 $((\Box (init\ w) \wedge more) \wedge finite) \wedge fin\ (init\ w) ; f$
by (rule *LeftChopImpChop*)

have 6: $\vdash (((\Box (init\ w) \wedge more) \wedge finite) \wedge fin\ (init\ w) ; f =$
 $((\Box (init\ w) \wedge more) \wedge finite) ; (init\ w \wedge f)$
using *AndFinChopEqvStateAndChop* **by** *blast*

have 7: $\vdash \neg(\Box (init\ w)) \longrightarrow (\Box (init\ w))\ yields\ (\neg(\Box (init\ w)))$
by (rule *NotBoxStateImpBoxYieldsNotBox*)

have 8: $\vdash (\Box (init\ w))\ yields\ (\neg(\Box (init\ w))) \longrightarrow$
 $((\Box (init\ w) \wedge more) \wedge finite)\ yields\ (\neg(\Box (init\ w)))$
using *AndYieldsA*

by (metis *AndMoreAndFiniteEqvAndFmore inteq-reflection*)

have 9: $\vdash ((\Box (init\ w) \wedge more) \wedge finite) ; (init\ w \wedge f) \wedge$
 $((\Box (init\ w) \wedge more) \wedge finite)\ yields\ (\neg(\Box (init\ w)))$

\longrightarrow
 $((\Box (init\ w) \wedge more) \wedge finite); ((init\ w \wedge f) \wedge \neg (\Box (init\ w))))$
by (*rule ChopAndYieldsImp*)
have 10: $\vdash (init\ w \wedge f) \wedge \neg (\Box (init\ w)) \longrightarrow$
 $((\Box (init\ w) \wedge more) \wedge finite); ((init\ w \wedge f) \wedge \neg (\Box (init\ w))))$
using 3 5 6 7 8 9 by fastforce
have 11: $\vdash ((\Box (init\ w) \wedge more) \wedge finite); ((init\ w \wedge f) \wedge \neg (\Box (init\ w))) \longrightarrow$
 $more; ((init\ w \wedge f) \wedge \neg (\Box (init\ w)))$
by (*metis 41 LeftChopImpChop Prop12*)
have 12: $\vdash (init\ w \wedge f) \wedge \neg (\Box (init\ w)) \longrightarrow$
 $more; ((init\ w \wedge f) \wedge \neg (\Box (init\ w)))$
using 10 11 by fastforce
from 12 show ?thesis using MoreChopContraFiniteB by blast
qed

lemma SCSImpBox:

assumes $\vdash f \longrightarrow empty \vee ((\Box (init\ w) \wedge more) \frown f)$
shows $\vdash (init\ w \wedge f) \wedge finite \longrightarrow \Box (init\ w)$
using assms by (simp add: CSImpBox schop-d-def)

lemma ChopstarInductR:

assumes $\vdash g \vee h; f \longrightarrow h$
shows $\vdash g; (chopstar\ f) \longrightarrow h$
by (*metis Chopstar-WPowerstar WPowerstarInductR WPowerstar-more-absorb assms int-eq*)

lemma BoxWPowerstarEqvBox:

$\vdash (init\ w \wedge wpowerstar (\Box (init\ w))) = \Box (init\ w)$
proof –
have 1: $\vdash \Box (init\ w); \Box (init\ w) \longrightarrow \Box (init\ w)$
by (*simp add: BoxStateChopBoxEqvBox int-iffD1*)
have 2: $\vdash (init\ w \wedge empty) \longrightarrow \Box (init\ w)$
by (*simp add: StateAndEmptyImpBoxState*)
have 3: $\vdash (init\ w \wedge empty) \vee \Box (init\ w); \Box (init\ w) \longrightarrow \Box (init\ w)$
using 1 2 by fastforce
have 4: $\vdash (init\ w \wedge empty); wpowerstar (\Box (init\ w)) \longrightarrow \Box (init\ w)$
using 3 WPowerstarInductR by blast
have 5: $\vdash init\ w \wedge wpowerstar (\Box (init\ w)) \longrightarrow \Box (init\ w)$
using 4 StateAndEmptyChop by fastforce
have 11: $\vdash \Box (init\ w) \longrightarrow (init\ w)$
using BoxElim by blast
have 12: $\vdash \Box (init\ w) \longrightarrow wpowerstar (\Box (init\ w))$
by (*simp add: WPowerstar-ext*)
have 13: $\vdash \Box (init\ w) \longrightarrow init\ w \wedge wpowerstar (\Box (init\ w))$
using 11 12 by fastforce
from 5 13 show ?thesis by fastforce
qed

lemma BoxCSEqvBox:

$\vdash (init\ w \wedge (\Box (init\ w))^*) = \Box (init\ w)$
by (*metis BoxWPowerstarEqvBox Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)

lemma *BoxSCSEqvBox*:

$\vdash (init\ w \wedge schopstar (\Box (init\ w))) = (\Box (init\ w) \wedge finite)$
proof –
have 1: $\vdash \Box (init\ w) \frown (\Box (init\ w) \wedge finite) \longrightarrow \Box (init\ w) \wedge finite$
by (*metis BoxStateAndChopEqvChop FiniteChopFiniteEqvFinite int-iffD2 inteq-reflection schop-d-def*)
have 2: $\vdash (init\ w \wedge empty) \longrightarrow \Box (init\ w) \wedge finite$
using *EmptyImpFinite StateAndEmptyImpBoxState* **by** *fastforce*
have 3: $\vdash (init\ w \wedge empty) \vee \Box (init\ w) \frown (\Box (init\ w) \wedge finite) \longrightarrow \Box (init\ w) \wedge finite$
using 1 2 **by** *fastforce*
have 4: $\vdash (init\ w \wedge empty) \frown schopstar (\Box (init\ w)) \longrightarrow \Box (init\ w) \wedge finite$
using *SChopstarInductR* 3
by (*metis Prop12 SChopImpFinite SChopstar-finite SChopstar-finite-absorb inteq-reflection*)
have 5: $\vdash init\ w \wedge schopstar (\Box (init\ w)) \longrightarrow \Box (init\ w) \wedge finite$
using 4 *StateAndEmptySChop* **by** *fastforce*
have 11: $\vdash \Box (init\ w) \longrightarrow (init\ w)$
using *BoxElim* **by** *blast*
have 12: $\vdash \Box (init\ w) \wedge finite \longrightarrow schopstar (\Box (init\ w))$
by (*metis SChopstar-WPowerstar WPowerstar-ext inteq-reflection*)
have 13: $\vdash \Box (init\ w) \wedge finite \longrightarrow init\ w \wedge schopstar (\Box (init\ w))$
using 11 12 **by** *fastforce*
from 5 13 **show** *?thesis* **by** *fastforce*
qed

lemma *WpowerstarAndMoreEqvAndMoreChop*:

$\vdash (wpowerstar\ f \wedge more) = (f \wedge more); wpowerstar\ f$
proof –
have 1: $\vdash (empty \vee (f \wedge more); wpowerstar\ f) \wedge more \longrightarrow (f \wedge more); wpowerstar\ f$
by (*auto simp: empty-d-def*)
have 2: $\vdash wpowerstar\ f = (empty \vee (f \wedge more); wpowerstar\ f)$
by (*metis ChopstarEqv Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)
have 3: $\vdash wpowerstar\ f \wedge more \longrightarrow (f \wedge more); wpowerstar\ f$
using 1 2 **by** *fastforce*
have 4: $\vdash (f \wedge more); wpowerstar\ f \longrightarrow wpowerstar\ f$
using 2 **by** *fastforce*
have 5: $\vdash (f \wedge more) \longrightarrow more$
by *auto*
hence 6: $\vdash (f \wedge more); wpowerstar\ f \longrightarrow more$
by (*rule LeftChopImpMoreRule*)
have 7: $\vdash (f \wedge more); wpowerstar\ f \longrightarrow wpowerstar\ f \wedge more$
using 4 6 **by** *fastforce*
from 3 7 **show** *?thesis* **by** *fastforce*
qed

lemma *CSAndMoreEqvAndMoreChop*:

$\vdash (f^* \wedge more) = (f \wedge more); f^*$

by (metis Chopstar-WPowerstar WPowerstar-more-absorb WpowerstarAndMoreEqvAndMoreChop int-eq)

lemma *SCSAndMoreEqvAndMoreSChop*:

$\vdash (\text{schopstar } f \wedge \text{more}) = (f \wedge \text{more}) \frown \text{schopstar } f$

proof –

have 1: $\vdash ((f \wedge \text{more}) \wedge \text{finite}) = ((f \wedge \text{finite}) \wedge \text{more})$

by *auto*

show *?thesis*

using *SChopstar-WPowerstar*[of *f*] *WPowerstar-more-absorb*[of *LIFT (f ∧ finite)*]

WpowerstarAndMoreEqvAndMoreChop[of *LIFT (f ∧ finite)*]

by (metis 1 int-eq schop-d-def)

qed

lemma *SCSAndMoreEqvAndFMoreSChop*:

$\vdash (\text{schopstar } f \wedge \text{fmore}) = (f \wedge \text{more}) \frown \text{schopstar } f$

by (metis *AndMoreAndFiniteEqvAndFmore Prop10 SCSAndMoreEqvAndMoreSChop SChopImpFinite SChop-star-finite*

inteq-reflection)

lemma *BoxStateAndWPowerstarEqvWPowerstar*:

$\vdash (\Box(\text{init } w) \wedge \text{wpowerstar } f \wedge \text{finite}) = (\text{init } w \wedge \text{wpowerstar } (\Box(\text{init } w) \wedge f) \wedge \text{finite})$

proof –

have 1: $\vdash \Box(\text{init } w) \longrightarrow \text{init } w$

using *BoxElim* by *blast*

have 2: $\vdash (\text{wpowerstar } f \wedge \text{more}) = (f \wedge \text{more}); \text{wpowerstar } f$

by (simp add: *WpowerstarAndMoreEqvAndMoreChop*)

have 3: $\vdash (\Box(\text{init } w) \wedge ((f \wedge \text{more}); \text{wpowerstar } f)) =$

$((\Box(\text{init } w) \wedge f \wedge \text{more}); (\Box(\text{init } w) \wedge \text{wpowerstar } f))$

by (rule *BoxStateAndChopEqvChop*)

have 4: $\vdash \Box(\text{init } w) \wedge f \wedge \text{more} \longrightarrow (\Box(\text{init } w) \wedge f) \wedge \text{more}$

by *auto*

hence 5: $\vdash (\Box(\text{init } w) \wedge f \wedge \text{more}); (\Box(\text{init } w) \wedge \text{wpowerstar } f) \longrightarrow$

$((\Box(\text{init } w) \wedge f) \wedge \text{more}); (\Box(\text{init } w) \wedge \text{wpowerstar } f)$

by (rule *LeftChopImpChop*)

have 6: $\vdash (\Box(\text{init } w) \wedge \text{wpowerstar } f) \wedge \text{more} \longrightarrow$

$((\Box(\text{init } w) \wedge f) \wedge \text{more}); (\Box(\text{init } w) \wedge \text{wpowerstar } f)$

using 2 3 5 by *fastforce*

hence 7: $\vdash (\Box(\text{init } w) \wedge \text{wpowerstar } f) \wedge \text{finite} \longrightarrow \text{wpowerstar } (\Box(\text{init } w) \wedge f)$

using *WPowerstarIntroMore*[of *LIFT (Box (init w) ∧ wpowerstar f)* *LIFT (Box (init w) ∧ f)*]

by *blast*

have 71: $\vdash \text{init } w \wedge \Box(\text{init } w) \wedge \text{wpowerstar } f \wedge \text{finite} \longrightarrow \text{init } w \wedge \text{wpowerstar } (\Box(\text{init } w) \wedge f) \wedge \text{finite}$

using 7 by *fastforce*

have 8: $\vdash \Box(\text{init } w) \wedge \text{wpowerstar } f \wedge \text{finite} \longrightarrow \text{init } w \wedge \text{wpowerstar } (\Box(\text{init } w) \wedge f) \wedge \text{finite}$

using 1 71 by *fastforce*

have 11: $\vdash \text{wpowerstar } (\Box(\text{init } w) \wedge f) \longrightarrow \text{wpowerstar } (\Box(\text{init } w))$

by (meson *Prop12 WPowerstar-iso int-iffD2 lift-and-com*)

have 12: $\vdash (\text{init } w \wedge \text{wpowerstar } (\Box(\text{init } w))) = \Box(\text{init } w)$

by (simp add: *BoxWPowerstarEqvBox*)

have 13: $\vdash \text{wpowerstar } (\Box(\text{init } w) \wedge f) \longrightarrow \text{wpowerstar } f$

by (meson Prop12 WPowerstar-iso int-iffD2 lift-and-com)
 have 14: $\vdash \text{init } w \wedge \text{wpowerstar } (\Box (\text{init } w) \wedge f) \longrightarrow \text{init } w \wedge \text{wpowerstar } (\Box (\text{init } w)) \wedge \text{wpowerstar } f$
 using 11 13 by fastforce
 have 15: $\vdash \text{init } w \wedge \text{wpowerstar } (\Box (\text{init } w)) \wedge \text{wpowerstar } f \longrightarrow \Box (\text{init } w) \wedge \text{wpowerstar } f$
 using 12 by auto
 have 16: $\vdash \text{init } w \wedge \text{wpowerstar } (\Box (\text{init } w) \wedge f) \longrightarrow \Box (\text{init } w) \wedge \text{wpowerstar } f$
 using 14 15 lift-imp-trans by blast
 from 8 16 show ?thesis by fastforce
 qed

lemma *BoxStateAndCSEqvCS:*

$\vdash (\Box (\text{init } w) \wedge f^* \wedge \text{finite}) = (\text{init } w \wedge (\Box (\text{init } w) \wedge f)^* \wedge \text{finite})$
 by (metis BoxStateAndWPowerstarEqvWPowerstar Chopstar-WPowerstar WPowerstar-more-absorb int-eq)

lemma *BoxStateAndSCSEqvSCS:*

$\vdash (\Box (\text{init } w) \wedge \text{schopstar } f) = (\text{init } w \wedge \text{schopstar } (\Box (\text{init } w) \wedge f))$

proof –

have 1: $\vdash (\Box (\text{init } w) \wedge f \wedge \text{finite}) = ((\Box (\text{init } w) \wedge f) \wedge \text{finite})$

by auto

show ?thesis

using BoxStateAndWPowerstarEqvWPowerstar[of w LIFT (f \wedge finite)]

SChopstar-WPowerstar[of f] SChopstar-WPowerstar[of LIFT ($\Box (\text{init } w) \wedge f$)]

SChopstar-finite[of f] SChopstar-finite[of LIFT ($\Box (\text{init } w) \wedge f$)]

by (metis 1 Prop10 inteq-reflection)

qed

lemma *BaWPowerstarImpWPowerstar:*

$\vdash \text{ba } (f \longrightarrow g) \wedge \text{finite} \longrightarrow \text{wpowerstar } f \longrightarrow \text{wpowerstar } g$

proof –

have 1: $\vdash \text{wpowerstar } f = (\text{empty} \vee (f \wedge \text{more}); \text{wpowerstar } f)$

by (metis ChopstarEqv Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection)

have 2: $\vdash \text{wpowerstar } g = (\text{empty} \vee (g \wedge \text{more}); \text{wpowerstar } g)$

by (metis ChopstarEqv Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection)

have 21: $\vdash \neg(\text{wpowerstar } g) = (\neg \text{empty} \wedge \neg((g \wedge \text{more}); \text{wpowerstar } g))$

using 2 by fastforce

hence 22: $\vdash \neg(\text{wpowerstar } g) = (\text{more} \wedge \neg((g \wedge \text{more}); \text{wpowerstar } g))$

by (simp add: empty-d-def)

have 3: $\vdash \text{wpowerstar } f \wedge \neg(\text{wpowerstar } g) \longrightarrow$

$(\text{empty} \vee (f \wedge \text{more}); \text{wpowerstar } f) \wedge \text{more} \wedge \neg((g \wedge \text{more}); \text{wpowerstar } g)$

using 1 22 by fastforce

have 31: $\vdash ((\text{empty} \vee (f \wedge \text{more}); \text{wpowerstar } f) \wedge \text{more}) = ((f \wedge \text{more}); \text{wpowerstar } f \wedge \text{more})$

by (auto simp: empty-d-def)

have 32: $\vdash \text{wpowerstar } f \wedge \neg(\text{wpowerstar } g) \longrightarrow (f \wedge \text{more}); \text{wpowerstar } f \wedge \neg((g \wedge \text{more}); \text{wpowerstar } g)$

using 3 31 by fastforce

have 4: $\vdash (f \longrightarrow g) \longrightarrow (f \wedge \text{more} \longrightarrow g \wedge \text{more})$

by auto

hence 5: $\vdash \text{ba } (f \longrightarrow g) \longrightarrow \text{ba } (f \wedge \text{more} \longrightarrow g \wedge \text{more})$

by (rule BaImpBa)

have 6: $\vdash ba (f \wedge more \longrightarrow g \wedge more) \longrightarrow$
 $(f \wedge more); wpowerstar f \longrightarrow (g \wedge more); wpowerstar f$
by (rule BaLeftChopImpChop)
have 7: $\vdash ba (f \longrightarrow g) \wedge (f \wedge more); wpowerstar f \longrightarrow (g \wedge more); wpowerstar f$
using 5 6 **by** fastforce
have 8: $\vdash (g \wedge more); wpowerstar f \wedge \neg ((g \wedge more); wpowerstar g)$
 $\longrightarrow (g \wedge more); (wpowerstar f \wedge \neg (wpowerstar g))$
by (rule ChopAndNotChopImp)
have 9: $\vdash (g \wedge more); (wpowerstar f \wedge \neg (wpowerstar g)) \longrightarrow more; (wpowerstar f \wedge \neg (wpowerstar$
 $g))$
by (rule AndChopB)
have 10: $\vdash ba (f \longrightarrow g) \longrightarrow more; (wpowerstar f \wedge \neg (wpowerstar g)) \longrightarrow$
 $more; (ba (f \longrightarrow g) \wedge wpowerstar f \wedge \neg (wpowerstar g))$
by (rule BaChopImpChopBa)
have 11: $\vdash ba (f \longrightarrow g) \wedge wpowerstar f \wedge \neg (wpowerstar g) \longrightarrow$
 $more; (ba (f \longrightarrow g) \wedge wpowerstar f \wedge \neg (wpowerstar g))$
using 32 7 8 9 10 **by** fastforce
hence 12: $\vdash finite \longrightarrow \neg ((ba (f \longrightarrow g)) \wedge (wpowerstar f) \wedge (\neg (wpowerstar g)))$
using MoreChopLoopFiniteB **by** blast
from 12 **show** ?thesis **by** (simp add: Valid-def)
qed

lemma BaCSImpCS:

$\vdash ba (f \longrightarrow g) \wedge finite \longrightarrow f^* \longrightarrow g^*$
by (metis BaWPowerstarImpWPowerstar Chopstar-WPowerstar WPowerstar-more-absorb int-eq)

lemma SDa-Da:

$\vdash sda f = da (f \wedge finite)$
unfolding sda-d-def da-d-def schop-d-def
by simp

lemma SBa-Ba:

$\vdash sba f = ba (finite \longrightarrow f)$
proof –
have 1: $\vdash (\neg (finite \longrightarrow f)) = (finite \wedge \neg f)$
by auto
show ?thesis
unfolding sba-d-def ba-d-def sda-d-def da-d-def schop-d-def
by (metis 1 AndChopCommute int-eq int-simps(17))
qed

lemma SBaSCSImpSCS:

$\vdash sba (f \longrightarrow g) \longrightarrow schopstar f \longrightarrow schopstar g$
proof –
have 1: $\vdash ba (finite \longrightarrow f \longrightarrow g) \longrightarrow ba (f \wedge finite \longrightarrow g \wedge finite)$
by (simp add: BaImpBa intI)
have 2: $\vdash schopstar f \longrightarrow finite$

by (simp add: SChopstar-finite)
 show ?thesis
 using BaWPowerstarImpWPowerstar[of LIFT (f ∧ finite) LIFT (g ∧ finite)]
 SChopstar-WPowerstar[of f] SChopstar-WPowerstar[of g]
 SBa-Ba[of LIFT (f → g)] 1 2 by fastforce
 qed

lemma BaWPowerstarEqvWPowerstar:

⊢ ba (f = g) ∧ finite → (wpowerstar f = wpowerstar g)
proof –
 have 0: ⊢ (f = g) = ((f → g) ∧ (g → f))
 by fastforce
 have 1: ⊢ ba (f = g) = (ba (f → g) ∧ ba (g → f))
 by (metis 0 BaAndEqv int-eq)
 have 2: ⊢ ba (f → g) ∧ finite → (wpowerstar f → wpowerstar g)
 by (rule BaWPowerstarImpWPowerstar)
 have 3: ⊢ ba (g → f) ∧ finite → (wpowerstar g → wpowerstar f)
 by (rule BaWPowerstarImpWPowerstar)
 have 4: ⊢ ba (f = g) ∧ finite → (wpowerstar f → wpowerstar g) ∧ (wpowerstar g → wpowerstar f)
 using 1 2 3 by fastforce
 have 5: ⊢ ((wpowerstar f → wpowerstar g) ∧ (wpowerstar g → wpowerstar f)) = (wpowerstar f = wpowerstar g)
 by auto
 from 4 5 show ?thesis by auto
 qed

lemma BaCSEqvCS:

⊢ ba (f = g) ∧ finite → (f[★] = g[★])
 by (metis BaWPowerstarEqvWPowerstar Chopstar-WPowerstar WPowerstar-more-absorb int-eq)

lemma SBaSCSEqvSCS:

⊢ sba (f = g) → (schopstar f = chopstar g)
proof –
 have 0: ⊢ (f = g) = ((f → g) ∧ (g → f))
 by fastforce
 have 1: ⊢ sba (f = g) = (sba (f → g) ∧ sba (g → f))
 by (metis 0 SBaAndEqv int-eq)
 show ?thesis using SBaSCSImpSCS[of f g] SBaSCSImpSCS[of g f]
 1 by fastforce
 qed

lemma BaAndWPowerstarImport:

⊢ ba f ∧ wpowerstar g ∧ finite → wpowerstar (f ∧ g)
proof –
 have 1: ⊢ f → (g → f ∧ g) by auto
 hence 2: ⊢ ba f → ba (g → f ∧ g) by (rule BaImpBa)
 have 3: ⊢ ba (g → f ∧ g) ∧ finite → wpowerstar g → wpowerstar (f ∧ g)
 by (rule BaWPowerstarImpWPowerstar)
 from 2 3 show ?thesis by fastforce

qed

lemma *BaAndCSImport*:

$\vdash \text{ba } f \wedge g^* \wedge \text{finite} \longrightarrow (f \wedge g)^*$

by (*metis BaAndWPowerstarImport Chopstar-WPowerstar WPowerstar-more-absorb int-eq*)

lemma *SBaAndSCSImport*:

$\vdash \text{sba } f \wedge \text{schopstar } g \longrightarrow \text{schopstar } (f \wedge g)$

proof –

have 1: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ **by** *auto*

hence 2: $\vdash \text{sba } f \longrightarrow \text{sba } (g \longrightarrow f \wedge g)$ **by** (*rule SBaImpSBa*)

have 3: $\vdash \text{sba } (g \longrightarrow f \wedge g) \longrightarrow \text{schopstar } g \longrightarrow \text{schopstar } (f \wedge g)$ **by** (*rule SBaSCSImpSCS*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

2.5.7 Len

lemma *wpower-len*:

$\vdash \text{wpower skip } k = \text{len } k$

by (*simp add: len-d-def*)

lemma *wpowerstar-skip-finite*:

$\vdash \text{finite} = \text{wpowerstar skip}$

using *WPowerstar-skip-finite* **by** *fastforce*

lemma *schopstar-skip-finite*:

$\vdash \text{finite} = \text{schopstar skip}$

by (*metis Prop10 SChopstar-WPowerstar WPowerstar-ext WPowerstar-skip-finite inteq-reflection*)

lemma *wpowerstar-skip-fmore*:

$\vdash \text{fmore} = \text{skip}; \text{wpowerstar skip}$

by (*metis FmoreEqvSkipChopFinite inteq-reflection wpowerstar-skip-finite*)

lemma *schopstar-skip-fmore*:

$\vdash \text{fmore} = \text{skip} \frown \text{schopstar skip}$

by (*metis NextSChopdef WPowerstar-skip-finite inteq-reflection next-d-def chopstar-skip-finite wpowerstar-skip-fmore*)

lemma *len-k-finite*:

$\vdash \text{len } k \longrightarrow \text{finite}$

proof (*induct k*)

case 0

then show *?case*

by (*metis EmptyImpFinite len-d-def wpow-0*)

next

case (*Suc k*)

then show *?case*

proof –

have 1: $\vdash \text{len } (\text{Suc } k) = \text{skip}; \text{len } k$

by (*simp add: len-d-def*)

have 2: $\vdash \text{skip} \longrightarrow \text{finite}$
by (*metis WPowerstar-ext WPowerstar-skip-finite inteq-reflection*)
show ?thesis
by (*metis 1 2 ChopImpChop FiniteChopFiniteEqvFinite Suc inteq-reflection*)
qed
qed

lemma *len-k-schop*:
 $\vdash \text{len } \text{Suc } k = \text{len } k \frown \text{skip}$
unfolding *len-d-def* **unfolding** *schop-d-def*
by (*metis Prop10 WPowerCommute int-eq len-k-finite wpow-Suc wpower-len*)

lemma *SkipChopAnd*:
 $\vdash ((\text{skip};f) \wedge (\text{skip};g)) = \text{skip};(f \wedge g)$
proof –
have 1: $\vdash \text{skip};(f \wedge g) \longrightarrow ((\text{skip};f) \wedge (\text{skip};g))$
by (*simp add: ChopAndA ChopAndB Prop12*)
have 2: $\vdash ((\text{skip};f) \wedge (\text{skip};g)) \longrightarrow \text{skip};(f \wedge g)$
by (*metis NextAndEqvNextAndNext SkipChopEqvNext int-eq int-iffD2*)
show ?thesis
using 1 2 *int-iffI* **by** *blast*
qed

lemma *SkipSChopAnd*:
 $\vdash ((\text{skip} \frown f) \wedge (\text{skip} \frown g)) = \text{skip} \frown (f \wedge g)$
by (*metis NextAndNextEqvNextRule NextSChopdef inteq-reflection lift-and-com*)

lemma *LenChopAnd*:
 $\vdash (\text{len } k;f \wedge \text{len } k;g) = \text{len } k;(f \wedge g)$
proof –
have 2: $\vdash \text{len } k;(f \wedge g) \longrightarrow (\text{len } k;f \wedge \text{len } k;g)$
by (*simp add: ChopAndA ChopAndB Prop12*)
have 1: $\vdash (\text{len } k;f \wedge \text{len } k;g) \longrightarrow \text{len } k;(f \wedge g)$
proof (*induct k arbitrary: f g*)
case 0
then show ?case
by (*metis EmptyChop int-iffD2 inteq-reflection wpow-0 wpower-len*)
next
case (*Suc k*)
then show ?case
proof –
have 1: $\vdash \text{len } \text{Suc } k;f = \text{len } k;(\text{skip};f)$
using *WPowerCommute[of LIFT skip k] ChopAssoc[of LIFT len k LIFT skip f]*
by (*metis inteq-reflection wpow-Suc wpower-len*)
have 2: $\vdash \text{len } \text{Suc } k;g = \text{len } k;(\text{skip};g)$
using *WPowerCommute[of LIFT skip k] ChopAssoc[of LIFT len k LIFT skip g]*
by (*metis inteq-reflection wpow-Suc wpower-len*)
have 3: $\vdash (\text{len } k;(\text{skip};f) \wedge \text{len } k;(\text{skip};g)) \longrightarrow \text{len } k;((\text{skip};f) \wedge (\text{skip};g))$
by (*simp add: Suc*)


```

have 4:  $\vdash ((\text{skip};f) \wedge (\text{skip};g)) = \text{skip};(f \wedge g)$ 
  by (simp add: SkipChopAnd)
have 5:  $\vdash \text{len } k;((\text{skip};f) \wedge (\text{skip};g)) = \text{len } (\text{Suc } k);(f \wedge g)$ 
  using WPowerCommute[of LIFT skip k] ChopAssoc[of LIFT len k LIFT skip LIFT (f \wedge g)]
  by (metis 4 inteq-reflection wpow-Suc wpower-len)
show ?thesis
by (metis 1 2 3 5 int-eq)
qed
qed
show ?thesis
using 1 2 int-iffI by blast
qed

```

```

lemma LenSChopAnd:
 $\vdash (\text{len } k \frown f \wedge \text{len } k \frown g) = \text{len } k \frown (f \wedge g)$ 
proof –
have 1:  $\vdash \text{len } k \longrightarrow \text{finite}$ 
  using len-k-finite by blast
show ?thesis unfolding schop-d-def
by (metis 1 LenChopAnd Prop10 int-eq)
qed

```

```

lemma LenEqvLenChopLen:
 $\vdash \text{len}(i+j) = \text{len}(i);\text{len}(j)$ 
proof
  (induct i)
  case 0
  then show ?case
  by (metis EmptyChop Prop11 add-cancel-left-left len-d-def wpow-0)
  next
  case (Suc i)
  then show ?case
  by (metis ChopAssoc add-Suc int-eq len-d-def wpow-Suc)
qed

```

```

lemma LenChopFalse:
 $\vdash \text{len } k ;\#False \longrightarrow \#False$ 
by (metis AndInfEqvChopFalse ChopImpChop InfEqvNotFinite int-iffD1 int-simps(21) inteq-reflection len-k-finite)

```

```

lemma LenSChopFalse:
 $\vdash \text{len } k \frown \#False \longrightarrow \#False$ 
by (metis AndChopB InfEqvNotFinite TrueChopAndFiniteEqvAndFiniteChopFinite infinite-d-def int-eq int-simps(19) int-simps(22) schop-d-def)

```

```

lemma len-len-suc-not:
 $\vdash \neg(\text{len } k \wedge \text{len } (\text{Suc } k);f)$ 
proof –
have 1:  $\vdash \text{len } k = \text{len } k;\text{empty}$ 
  by (meson ChopEmpty Prop11)

```

have 2: $\vdash \text{len } (\text{Suc } k);f = \text{len } k;(\text{skip};f)$
using *ChopAssoc[of LIFT len k LIFT skip f] WPowerCommute[of LIFT skip k]*
by (*metis inteq-reflection wpow-Suc wpower-len*)
have 3: $\vdash (\text{len } k;\text{empty} \wedge \text{len } k;(\text{skip};f)) = \text{len } k;(\text{empty} \wedge \text{skip};f)$
by (*simp add: LenChopAnd*)
have 4: $\vdash (\text{empty} \wedge \text{skip};f) \longrightarrow \#False$
unfolding *empty-d-def more-d-def next-d-def*
by (*metis ChopAndB Prop01 int-simps(16) int-simps(25) int-simps(4) inteq-reflection*)
have 5: $\vdash (\text{len } k \wedge \text{len } (\text{Suc } k);f) \longrightarrow \text{len } k;\#False$
by (*metis 1 2 3 4 RightChopImpChop inteq-reflection*)
have 6: $\vdash \text{len } k ;\#False \longrightarrow \#False$
using *LenChopFalse* **by** *blast*
show *?thesis*
by (*metis 5 6 int-simps(14) inteq-reflection lift-imp-trans*)
qed

lemma *len-len-suc-not-schop*:
 $\vdash \neg(\text{len } k \wedge \text{len } (\text{Suc } k) \frown f)$
unfolding *schop-d-def*
by (*metis AndInfEqvChopFalse LenChopFalse Prop09 Prop10 int-eq int-simps(14) itl-def(8) len-len-suc-not*)

lemma *Finite-exist-len*:
 $\vdash \text{finite} = (\exists k. \text{len } k)$
by (*metis ExEqvRule WPowerstar-skip-finite int-eq wpower-len wpowerstar-d-def*)

lemma *LenNPlusOneB*:
 $\vdash \text{len}(n+1) = \text{len}(n);\text{skip}$
proof –
have 1: $\vdash \text{len}(n+1) = \text{len}(n);\text{len}(1)$ **by** (*rule LenEqvLenChopLen*)
have 2: $\vdash \text{len}(1) = \text{skip}$ **by** (*simp add: ChopEmpty len-d-def*)
hence 3: $\vdash \text{len}(n);\text{len}(1) = \text{len}(n);\text{skip}$ **using** *RightChopEqvChop* **by** *blast*
from 1 3 **show** *?thesis* **by** *fastforce*
qed

lemma *LenCommute*:
 $\vdash (\text{skip};(\text{len } n)) = (\text{len } n);\text{skip}$
by (*simp add: WPowerCommute len-d-def*)

lemma *NotFixedChop*:
 $\vdash (\neg((g \wedge \text{len}(k));f)) = (\neg(\text{di}(g \wedge \text{len}(k))) \vee ((g \wedge \text{len}(k));(\neg f)))$
by (*auto simp add: itl-defs len-defs Valid-def min-def nlength-eq-enat-nfiniteD*)

2.5.8 Properties of While

lemma *SChopstar-Chopstar*:
 $\vdash \text{schopstar } f = \text{chopstar } (f \wedge \text{finite})$
by (*metis Chopstar-WPowerstar SChopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)

lemma *SWhile-While*:
 $\vdash \text{swhile } g \text{ do } f = \text{while } g \text{ do } (f \wedge \text{finite})$

proof –

have 1: $\vdash ((g \wedge f) \wedge \text{finite}) = (g \wedge f \wedge \text{finite})$

by *auto*

have 2: $\vdash \text{chopstar } ((g \wedge f) \wedge \text{finite}) = \text{chopstar } (g \wedge f \wedge \text{finite})$

using 1 **by** (*metis Chopstardef int-eq*)

show *?thesis*

unfolding *swhile-d-def while-d-def*

using *SFinEqvFinAndFinite*[of *LIFT* $\neg g$] *SChopstar-Chopstar*[of *LIFT* $(g \wedge f)$]

2 *SChopstar-finite*[of *LIFT* $(g \wedge f)$] **by** *fastforce*

qed

lemma *IfAndFiniteDist*:

$\vdash (\text{if}_i (\text{init } w) \text{ then } (f;g) \text{ else } \text{empty} \wedge \text{finite}) =$

$(\text{if}_i (\text{init } w) \text{ then } ((f \wedge \text{finite});(g \wedge \text{finite})) \text{ else } \text{empty})$

proof –

have 1: $\vdash (\text{if}_i (\text{init } w) \text{ then } (f;g) \text{ else } \text{empty} \wedge \text{finite}) =$

$(((\text{init } w \wedge (f;g)) \vee (\neg(\text{init } w) \wedge \text{empty})) \wedge \text{finite})$

by (*auto simp: ifthenelse-d-def*)

have 2: $\vdash (((\text{init } w \wedge (f;g)) \vee (\neg(\text{init } w) \wedge \text{empty})) \wedge \text{finite}) =$

$(((\text{init } w \wedge (f;g) \wedge \text{finite}) \vee (\neg(\text{init } w) \wedge \text{empty} \wedge \text{finite})))$

by *auto*

have 3: $\vdash (\text{init } w \wedge (f;g) \wedge \text{finite}) = (\text{init } w \wedge (f \wedge \text{finite});(g \wedge \text{finite}))$

using *ChopAndFiniteDist* **by** *fastforce*

have 4: $\vdash (\neg(\text{init } w) \wedge \text{empty} \wedge \text{finite}) = (\neg(\text{init } w) \wedge \text{empty})$

using *FiniteAndEmptyEqvEmpty* **by** *auto*

have 5: $\vdash (((\text{init } w \wedge (f;g) \wedge \text{finite}) \vee (\neg(\text{init } w) \wedge \text{empty} \wedge \text{finite}))) =$

$(((\text{init } w \wedge (f \wedge \text{finite});(g \wedge \text{finite})) \vee (\neg(\text{init } w) \wedge \text{empty}))$

by (*metis 2 3 4 inteq-reflection*)

have 6: $\vdash (((\text{init } w \wedge (f \wedge \text{finite});(g \wedge \text{finite})) \vee (\neg(\text{init } w) \wedge \text{empty})) =$

$(\text{if}_i (\text{init } w) \text{ then } ((f \wedge \text{finite});(g \wedge \text{finite})) \text{ else } \text{empty})$

by (*auto simp: ifthenelse-d-def*)

from 1 2 5 6 **show** *?thesis* **by** (*metis inteq-reflection*)

qed

lemma *WhileEqvIf*:

$\vdash ((\text{while } (\text{init } w) \text{ do } f) \wedge \text{finite}) =$

$(\text{if}_i (\text{init } w) \text{ then } (f; (\text{while } (\text{init } w) \text{ do } f)) \text{ else } \text{empty} \wedge \text{finite})$

proof –

have 1: $\vdash (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) =$

$(((((\text{init } w) \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite})$

by (*simp add: while-d-def*)

have 2: $\vdash (\text{init } w \wedge f)^* = (\text{empty} \vee ((\text{init } w \wedge f); (\text{init } w \wedge f)^*))$

by (*metis CSEqvOrChopCSB ChopplusCommutate int-eq*)

have 21: $\vdash (((\text{init } w) \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite} =$

$((\text{empty} \vee ((\text{init } w \wedge f); (\text{init } w \wedge f)^*)) \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite}$

using 2 **by** *fastforce*

have 22: $\vdash ((\text{empty} \vee ((\text{init } w \wedge f); (\text{init } w \wedge f)^*)) \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite} =$

$((\text{empty} \wedge \text{fin } (\neg(\text{init } w))) \wedge \text{finite}) \vee$

$(((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite}$

by *auto*
have 23: $\vdash (((init\ w) \wedge f)^* \wedge fin\ (\neg\ (init\ w))) \wedge finite =$
 $((empty \wedge fin\ (\neg\ (init\ w))) \wedge finite) \vee$
 $((init\ w \wedge f); (init\ w \wedge f)^* \wedge fin\ (\neg\ (init\ w))) \wedge finite))$
 using 21 22 by *auto*
have 3: $\vdash (empty \wedge fin\ (\neg\ (init\ w))) = (\neg\ (init\ w) \wedge empty)$
 by (*metis* *FinAndEmpty* *Prop04* *lift-and-com*)
hence 31: $\vdash (empty \wedge fin\ (\neg\ (init\ w))) \wedge finite = (\neg\ (init\ w) \wedge empty \wedge finite)$
 by *auto*
have 32: $\vdash (\neg\ (init\ w) \wedge empty \wedge finite) = (\neg\ (init\ w) \wedge empty)$
 using *FiniteAndEmptyEqvEmpty* by *auto*
have 33: $\vdash (empty \wedge fin\ (\neg\ (init\ w))) \wedge finite = (\neg\ (init\ w) \wedge empty)$
 using 31 32 by *fastforce*
have 34: $\vdash ((empty \wedge fin\ (\neg\ (init\ w))) \wedge finite) \vee$
 $((init\ w \wedge f); (init\ w \wedge f)^* \wedge fin\ (\neg\ (init\ w))) \wedge finite) =$
 $((\neg\ (init\ w) \wedge empty) \vee$
 $((init\ w \wedge f); (init\ w \wedge f)^* \wedge fin\ (\neg\ (init\ w))) \wedge finite))$
 using 23 33 by *auto*
have 4: $\vdash (init\ w \wedge f); (init\ w \wedge f)^* = (init\ w \wedge (f; (init\ w \wedge f)^*))$
 by (*rule* *StateAndChop*)
have 41: $\vdash (((init\ w \wedge f); (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w))) \wedge finite =$
 $(init\ w \wedge (f; (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w))) \wedge finite)$
 using 4 by *auto*
have 42: $\vdash (init\ w \wedge ((f; (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w))) \wedge finite) =$
 $(init\ w \wedge ((f; (init\ w \wedge f)^*) \wedge fin\ (init\ (\neg\ w))) \wedge finite))$
 using *Initprop(2)* by (*metis* *StateAndEmptyChop* *int-eq*)
have 5: $\vdash ((f; ((init\ w \wedge f)^*)) \wedge (fin\ (init\ (\neg\ w)))) \wedge finite =$
 $((f \wedge finite); ((init\ w \wedge f)^* \wedge (fin\ (init\ (\neg\ w)))) \wedge finite))$
 using *ChopAndFin* by *fastforce*
hence 49: $\vdash (init\ w \wedge (f; (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w))) \wedge finite =$
 $(init\ w \wedge (f \wedge finite); ((init\ w \wedge f)^* \wedge (fin\ (init\ (\neg\ w)))) \wedge finite))$
 using 42 by *fastforce*
have 50: $\vdash (((init\ w \wedge f); (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w))) \wedge finite =$
 $(init\ w \wedge (f \wedge finite); ((init\ w \wedge f)^* \wedge (fin\ (init\ (\neg\ w)))) \wedge finite))$
 by (*meson* 41 49 *Prop04* *lift-and-com*)
have 51: $\vdash (init\ w \wedge (f \wedge finite); ((init\ w \wedge f)^* \wedge (fin\ (init\ (\neg\ w)))) \wedge finite =$
 $(init\ w \wedge (f \wedge finite); ((init\ w \wedge f)^* \wedge (fin\ (\neg\ (init\ w)))) \wedge finite))$
 by (*metis* (*no-types*) *EmptyChop* *Initprop(2)* *inteq-reflection*)
have 52: $\vdash (((init\ w \wedge f); (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w))) \wedge finite =$
 $(init\ w \wedge ((f \wedge finite); ((init\ w \wedge f)^* \wedge fin\ (\neg\ (init\ w)))) \wedge finite))$
 using 50 51 by *fastforce*
have 53: $\vdash ((\neg\ (init\ w) \wedge empty) \vee$
 $((init\ w \wedge f); (init\ w \wedge f)^* \wedge fin\ (\neg\ (init\ w))) \wedge finite) =$
 $((\neg\ (init\ w) \wedge empty) \vee$
 $(init\ w \wedge ((f \wedge finite); ((init\ w \wedge f)^* \wedge fin\ (\neg\ (init\ w)))) \wedge finite))$
 using 52 34 by *auto*
have 6: $\vdash ((f \wedge finite); (((init\ w \wedge f)^* \wedge fin\ (\neg\ (init\ w))) \wedge finite)) =$
 $(f \wedge finite); (while\ (init\ w)\ do\ f \wedge finite)$
 by (*simp* *add*: *while-d-def*)
have 61: $\vdash (init\ w \wedge ((f \wedge finite); (((init\ w \wedge f)^* \wedge fin\ (\neg\ (init\ w))) \wedge finite))) =$

```

      (init w ∧ ((f ∧ finite); (while (init w) do f ∧ finite)))
using 6 by auto
have 62: ⊢ ((¬ (init w) ∧ empty) ∨
  (init w ∧ ((f ∧ finite); (((init w ∧ f)* ∧ fin (¬ (init w))) ∧ finite)) ))
  =((¬ (init w) ∧ empty) ∨
  (init w ∧ ((f ∧ finite);(while (init w) do f ∧ finite)) )) )
using 61 by fastforce
have 7: ⊢ (while (init w) do f ∧ finite)
  = (((¬ (init w) ∧ empty) ∨
  (init w ∧ (f ; while (init w) do f) ∧ finite)))
proof –
  have ⊢ (while (init w) do f ∧ finite) =
    (¬ init w ∧ empty ∨ (init w ∧ f);(init w ∧ f)* ∧ fin (¬ init w) ∧ finite)
  by (metis 1 23 34 inteq-reflection)
  then show ?thesis
    by (metis 21 22 34 52 ChopAndFiniteDist int-eq)
qed
have 71: ⊢ ((ifi (init w) then (f; (while (init w) do f)) else empty) ∧ finite) =
  (((¬ (init w) ∧ empty) ∨ (init w ∧ (f; while (init w) do f)) ∧ finite))
  using FiniteAndEmptyEqvEmpty by (auto simp: ifthenelse-d-def)
from 7 71 show ?thesis by fastforce
qed

```

```

lemma WPower-import-finite:
  ⊢ (wpower f k ∧ finite) = wpower (f ∧ finite) k
proof (induct k)
case 0
then show ?case
using FiniteAndEmptyEqvEmpty by fastforce
next
case (Suc k)
then show ?case
by (metis ChopAndFiniteDist inteq-reflection wpow-Suc)
qed

```

```

lemma WPowerstar-import-finite:
  ⊢ (wpowerstar f ∧ finite) = (wpowerstar (f ∧ finite))
proof –
  have 1: ⊢ (wpowerstar (f ∧ finite)) → (wpowerstar f ∧ finite)
    by (metis OrFiniteInf Prop12 SChopstar-finite SChopstar-WPowerstar WPowerstar-subdist inteq-reflection)
  have 2: ⊢ wpowerstar f ∧ finite → (wpowerstar (f ∧ finite))
    unfolding wpowerstar-d-def using WPower-import-finite[of f] by fastforce
  show ?thesis
    using 1 2 int-iffI by blast
qed

```

```

lemma Chopstar-import-finite:
  ⊢ (chopstar f ∧ finite) = (chopstar (f ∧ finite))
using Chopstar-WPowerstar[of f] Chopstar-WPowerstar[of LIFT (f ∧ finite) ]

```

by (metis WPowerstar-import-finite WPowerstar-more-absorb int-eq)

lemma AndMoreSChopAndMoreEqvAndMoreSChop:

$\vdash ((f \wedge \text{more}) \frown g \wedge \text{more}) = (f \wedge \text{more}) \frown g$

by (meson AndSChopB MoreSChopImpMore Prop10 Prop11 lift-imp-trans)

lemma WPowerstar-chopstar:

$\vdash (\text{wpowerstar } (f \wedge \text{more})) = (\text{chopstar } f)$

by (meson Chopstar-WPowerstar Prop11)

lemma While-import-finite:

$\vdash (\text{while } g \text{ do } f \wedge \text{finite}) = (\text{while } g \text{ do } (f \wedge \text{finite}))$

proof –

have 1: $\vdash (g \wedge f \wedge \text{finite}) = ((g \wedge f) \wedge \text{finite})$

by auto

have 2: $\vdash \text{chopstar } (g \wedge f \wedge \text{finite}) = \text{chopstar } ((g \wedge f) \wedge \text{finite})$

by (metis 1 Chopstardef int-eq)

show ?thesis

unfolding while-d-def

using Chopstar-import-finite[of LIFT (g ∧ f)] 2 **by** fastforce

qed

lemma SWhileEqvIf:

$\vdash \text{swhile } (\text{init } w) \text{ do } f = \text{if}_i (\text{init } w) \text{ then } (f \frown (\text{swhile } (\text{init } w) \text{ do } f)) \text{ else empty}$

proof –

have 1: $\vdash ((\text{init } w \wedge f) \wedge \text{finite}) = (\text{init } w \wedge f \wedge \text{finite})$

by auto

have 2: $\vdash \text{chopstar } ((\text{init } w \wedge f) \wedge \text{finite}) = \text{chopstar } (\text{init } w \wedge f \wedge \text{finite})$

using 1 **by** (metis Chopstardef inteq-reflection)

have 3: $\vdash ((\text{while } (\text{init } w) \text{ do } f) \wedge \text{finite}) = \text{while } (\text{init } w) \text{ do } (f \wedge \text{finite})$

unfolding while-d-def

using Chopstar-import-finite[of LIFT (init w ∧ f)] 2 **by** fastforce

have 4: $\vdash \text{while } (\text{init } w) \text{ do } (f \wedge \text{finite}) = \text{swhile } (\text{init } w) \text{ do } f$

by (metis Prop04 SWhile-While lift-and-com swhile-d-def)

have 5: $\vdash ((\text{init } w \wedge (f \wedge \text{finite}); \text{while } (\text{init } w) \text{ do } (f \wedge \text{finite}) \vee \neg \text{init } w \wedge \text{empty}) \wedge \text{finite}) =$
 $((\text{init } w \wedge (f \wedge \text{finite}); \text{while } (\text{init } w) \text{ do } (f \wedge \text{finite}) \wedge \text{finite} \vee \neg \text{init } w \wedge \text{empty} \wedge \text{finite}))$

by fastforce

have 6: $\vdash ((f \wedge \text{finite}); \text{while } (\text{init } w) \text{ do } (f \wedge \text{finite}) \wedge \text{finite}) =$
 $(f \wedge \text{finite}); \text{while } (\text{init } w) \text{ do } (f \wedge \text{finite})$

by (metis 3 ChopAndFiniteDist Prop10 Prop12 int-eq int-iffD2)

have 7: $\vdash (\neg \text{init } w \wedge \text{empty} \wedge \text{finite}) = (\neg \text{init } w \wedge \text{empty})$

using FiniteAndEmptyEqvEmpty **by** auto

have 8: $\vdash (\text{if}_i (\text{init } w) \text{ then } ((f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } (f \wedge \text{finite}))) \text{ else empty} \wedge \text{finite}) =$
 $(\text{if}_i (\text{init } w) \text{ then } (f \frown (\text{swhile } (\text{init } w) \text{ do } f)) \text{ else empty})$

unfolding ifthenelse-d-def schop-d-def

by (metis 4 5 6 7 inteq-reflection)

have 9: $\vdash ((\text{while } (\text{init } w) \text{ do } (f \wedge \text{finite})) \wedge \text{finite}) =$
 $(\text{if}_i (\text{init } w) \text{ then } ((f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } (f \wedge \text{finite}))) \text{ else } \text{empty} \wedge \text{finite})$
by (*simp add: WhileEqvIf*)
show ?thesis
by (*metis 3 4 8 9 Prop10 Prop12 int-iffD2 inteq-reflection*)
qed

lemma *WhileChopEqvIf*:

$\vdash ((\text{while } (\text{init } w) \text{ do } f) \wedge \text{finite}); g =$
 $\text{if}_i (\text{init } w) \text{ then } ((f \wedge \text{finite}); ((\text{while } (\text{init } w) \text{ do } f) \wedge \text{finite}); g)) \text{ else } g$

proof –

have 1: $\vdash (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) =$
 $(\text{if}_i (\text{init } w) \text{ then } (f; (\text{while } (\text{init } w) \text{ do } f)) \text{ else } \text{empty} \wedge \text{finite})$

by (*rule WhileEqvIf*)

have 11: $\vdash (\text{if}_i (\text{init } w) \text{ then } (f; (\text{while } (\text{init } w) \text{ do } f)) \text{ else } \text{empty} \wedge \text{finite}) =$
 $(\text{if}_i (\text{init } w) \text{ then } ((f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})) \text{ else } \text{empty})$

using *IfAndFiniteDist* **by** *fastforce*

have 12: $\vdash (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) =$
 $(\text{if}_i (\text{init } w) \text{ then } ((f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})) \text{ else } \text{empty})$

using 1 11 **by** *fastforce*

hence 2: $\vdash ((\text{while } (\text{init } w) \text{ do } f) \wedge \text{finite}); g =$
 $(\text{if}_i (\text{init } w)$
 $\text{then } (((f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})); g)$
 $\text{else } (\text{empty}; g))$

by (*rule IfChopEqvRule*)

have 3: $\vdash \text{empty}; g = g$

by (*rule EmptyChop*)

have 4: $\vdash (\text{if}_i (\text{init } w)$
 $\text{then } (((f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})); g)$
 $\text{else } (\text{empty}; g)) =$
 $(\text{if}_i (\text{init } w)$
 $\text{then } (((f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})); g)$
 $\text{else } g)$

using 3 **using** *inteq-reflection* **by** *fastforce*

have 5: $\vdash (((f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})); g) =$
 $((f \wedge \text{finite}); ((\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}); g))$

by (*meson ChopAssoc Prop11*)

have 6: $\vdash (\text{if}_i (\text{init } w)$
 $\text{then } (((f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})); g)$
 $\text{else } g) =$
 $(\text{if}_i (\text{init } w)$
 $\text{then } ((f \wedge \text{finite}); ((\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}); g))$
 $\text{else } g)$

using 5 **using** *inteq-reflection* **by** *fastforce*

show ?thesis

using 2 4 6 **by** *fastforce*

qed

lemma *SWhileSChopEqvIf*:
 $\vdash (\text{ swhile } (\text{ init } w) \text{ do } f) \frown g = \text{ if}_i (\text{ init } w) \text{ then } (f \frown ((\text{ swhile } (\text{ init } w) \text{ do } f) \frown g)) \text{ else } g$
unfolding *schop-d-def*
by (*metis* (*no-types*, *opaque-lifting*) *ChopEmpty EmptySChop SChopAssoc SWhile-While WhileChopEqvIf* *inteq-reflection chop-d-def*)

lemma *WhileChopEqvIfRule*:
assumes $\vdash f = (\text{ while } (\text{ init } w) \text{ do } g \wedge \text{ finite}); h$
shows $\vdash f = \text{ if}_i (\text{ init } w) \text{ then } ((g \wedge \text{ finite}); f) \text{ else } h$
proof –
have 1: $\vdash f = (\text{ while } (\text{ init } w) \text{ do } g \wedge \text{ finite}); h$
using *assms* **by** *auto*
have 2: $\vdash (\text{ while } (\text{ init } w) \text{ do } g \wedge \text{ finite}); h =$
 $\text{ if}_i (\text{ init } w) \text{ then } ((g \wedge \text{ finite}); ((\text{ while } (\text{ init } w) \text{ do } g \wedge \text{ finite}); h)) \text{ else } h$
by (*rule* *WhileChopEqvIf*)
have 3: $\vdash ((g \wedge \text{ finite}); f) = ((g \wedge \text{ finite}); ((\text{ while } (\text{ init } w) \text{ do } g \wedge \text{ finite}); h))$
using 1 **by** (*rule* *RightChopEqvChop*)
have 4: $\vdash ((g \wedge \text{ finite}); ((\text{ while } (\text{ init } w) \text{ do } g \wedge \text{ finite}); h)) = ((g \wedge \text{ finite}); f)$
using 3 **by** *auto*
have 5: $\vdash \text{ if}_i (\text{ init } w) \text{ then } ((g \wedge \text{ finite}); ((\text{ while } (\text{ init } w) \text{ do } g \wedge \text{ finite}); h)) \text{ else } h =$
 $\text{ if}_i (\text{ init } w) \text{ then } ((g \wedge \text{ finite}); f) \text{ else } h$
using 4 **using** *inteq-reflection* **by** *fastforce*
from 1 2 5 **show** *?thesis* **by** *fastforce*
qed

lemma *SWhileSChopEqvIfRule*:
assumes $\vdash f = (\text{ swhile } (\text{ init } w) \text{ do } g) \frown h$
shows $\vdash f = \text{ if}_i (\text{ init } w) \text{ then } (g \frown f) \text{ else } h$
using *assms*
by (*metis* *SWhileSChopEqvIf* *inteq-reflection*)

lemma *WhileImpFin*:
 $\vdash \text{ while } (\text{ init } w) \text{ do } f \longrightarrow \text{ fin } (\neg (\text{ init } w))$
proof –
have 1: $\vdash (\text{ init } w \wedge f)^* \wedge \text{ fin } (\neg (\text{ init } w)) \longrightarrow \text{ fin } (\neg (\text{ init } w))$ **by** *auto*
from 1 **show** *?thesis* **by** (*simp* *add*: *while-d-def*)
qed

lemma *SWhileImpFin*:
 $\vdash \text{ swhile } (\text{ init } w) \text{ do } f \longrightarrow \text{ sfin } (\neg (\text{ init } w))$
by (*simp* *add*: *Prop01 Prop05 swhile-d-def*)

lemma *WhileEqvEmptyOrChopWhile*:
 $\vdash (\text{ while } (\text{ init } w) \text{ do } f \wedge \text{ finite}) =$
 $((\neg (\text{ init } w) \wedge \text{ empty}) \vee (\text{ init } w \wedge ((f \wedge \text{ more}) \wedge \text{ finite}); (\text{ while } (\text{ init } w) \text{ do } f \wedge \text{ finite})))$
proof –
have 1: $\vdash (\text{ init } w \wedge f)^* = (\text{ empty } \vee ((\text{ init } w \wedge f) \wedge \text{ more}); (\text{ init } w \wedge f)^*)$

by (rule ChopstarEqv)
 have 2: $\vdash ((\text{init } w \wedge f) \wedge \text{more}) = (\text{init } w \wedge (f \wedge \text{more}))$
 by auto
 hence 3: $\vdash ((\text{init } w \wedge f) \wedge \text{more}); (\text{init } w \wedge f)^* = (\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^*$
 by (rule LeftChopEqvChop)
 have 4: $\vdash (\text{init } w \wedge f)^* = (\text{empty} \vee (\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^*)$
 using 1 3 by fastforce
 have 40: $\vdash \text{fin } (\neg (\text{init } w)) = \text{fin } (\neg w)$
 by (metis FinEqvTrueChopAndEmpty InitAndEmptyEqvAndEmpty Initprop(2) inteq-reflection)
 have 5: $\vdash ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$
 $((\text{empty} \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) \vee$
 $((\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))$
 using 1 4 by fastforce
 have 51: $\vdash ((\text{empty} \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite}) = ((\neg (\text{init } w) \wedge \text{empty}) \wedge \text{finite})$
 by (metis FinAndEmpty inteq-reflection lift-and-com)
 have 52: $\vdash ((\neg (\text{init } w) \wedge \text{empty}) \wedge \text{finite}) = (\neg (\text{init } w) \wedge \text{empty})$
 using EmptyImpFinite by auto
 have 6: $\vdash ((\text{empty} \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite}) = (\neg (\text{init } w) \wedge \text{empty})$
 using 51 52 by fastforce
 have 60: $\vdash ((\text{empty} \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) \vee$
 $((\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $((\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))$
 using 6 by fastforce
 have 61: $\vdash ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $((\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))$
 by (metis 5 60 inteq-reflection)
 have 70: $\vdash (\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^* = (\text{init } w \wedge (f \wedge \text{more}); (\text{init } w \wedge f)^*)$
 by (rule StateAndChop)
 have 7: $\vdash ((\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\text{init } (\neg w)) \wedge \text{finite}) =$
 $(\text{init } w \wedge (f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\text{init } (\neg w)) \wedge \text{finite})$
 using 70 by auto
 have 8: $\vdash ((f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\text{init } (\neg w)) \wedge \text{finite}) =$
 $((f \wedge \text{more}) \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge \text{fin } (\text{init } (\neg w)) \wedge \text{finite}))$
 using ChopAndFin by fastforce
 have 81: $\vdash \text{fin } (\text{init } (\neg w)) = \text{fin } (\neg (\text{init } w))$
 by (meson FinEqvFin Initprop(2) Prop11)
 have 82: $\vdash ((f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$
 $((f \wedge \text{more}) \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))$
 using 8 81 by (metis inteq-reflection)
 have 83: $\vdash (\text{init } w \wedge (f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))$
 using 82 by fastforce
 have 84: $\vdash ((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge (f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})))$
 using 82 by auto
 have 9: $\vdash ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$

$((\neg (init\ w) \wedge empty) \vee$
 $(init\ w \wedge ((f \wedge more) \wedge finite);$
 $((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w)) \wedge finite)))$
by (*metis 61 7 81 84 inteq-reflection*)
have 10: $\vdash ((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w)) \wedge finite) =$
 $((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w))) \wedge finite)$
by auto
hence 11: $\vdash ((\neg (init\ w) \wedge empty) \vee$
 $(init\ w \wedge ((f \wedge more) \wedge finite);$
 $((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w)) \wedge finite))) =$
 $((\neg (init\ w) \wedge empty) \vee$
 $(init\ w \wedge ((f \wedge more) \wedge finite);$
 $((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w))) \wedge finite)))$
by (*metis EmptyChop int-eq*)
have 12: $\vdash ((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w)) \wedge finite) =$
 $((\neg (init\ w) \wedge empty) \vee$
 $(init\ w \wedge ((f \wedge more) \wedge finite);$
 $((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w))) \wedge finite)))$
by (*metis 11 9 inteq-reflection*)
from 12 show ?thesis
by (*metis 10 inteq-reflection while-d-def*)
qed

lemma *SWhileEqvEmptyOrSChopSWhile*:

$\vdash\ swhile\ (init\ w)\ do\ f = ((\neg (init\ w) \wedge empty) \vee (init\ w \wedge (f \wedge more) \wedge swhile\ (init\ w)\ do\ f))$
proof –
have 1: $\vdash (((f \wedge finite) \wedge more) \wedge finite) = ((f \wedge more) \wedge finite)$
by auto
show ?thesis
unfolding *schop-d-def*
using *WhileEqvEmptyOrChopWhile*[*of w LIFT (f) SWhile-While*][*of LIFT (init w) f*]
by (*metis While-import-finite inteq-reflection*)
qed

lemma *WhileIntro*:

assumes $\vdash \neg (init\ w) \wedge f \longrightarrow empty$
 $\vdash init\ w \wedge f \longrightarrow ((g \wedge more) \wedge finite); f$
shows $\vdash f \wedge finite \longrightarrow while\ (init\ w)\ do\ g$
proof –
have 1: $\vdash \neg (init\ w) \wedge f \longrightarrow empty$
using *assms* **by blast**
have 2: $\vdash init\ w \wedge f \longrightarrow ((g \wedge more) \wedge finite); f$
using *assms* **by blast**
have 3: $\vdash (while\ (init\ w)\ do\ g \wedge finite) =$
 $((\neg (init\ w) \wedge empty) \vee$
 $(init\ w \wedge ((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite)))$
by (*rule WhileEqvEmptyOrChopWhile*)
hence 31: $\vdash \neg (while\ (init\ w)\ do\ g \wedge finite) =$
 $(\neg(\neg (init\ w) \wedge empty) \vee$
 $(init\ w \wedge ((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite))))$

by *fastforce*
hence 32: $\vdash (f \wedge \neg (\text{while } (init\ w) \text{ do } g \wedge finite)) =$
 $(f \wedge \neg (\neg (init\ w) \wedge empty) \vee$
 $(init\ w \wedge ((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite))))$
 by *fastforce*
have 33: $\vdash (f \wedge \neg (\neg (init\ w) \wedge empty) \vee$
 $(init\ w \wedge ((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite)))) =$
 $(f \wedge \neg (\neg (init\ w) \wedge empty) \wedge$
 $\neg (init\ w \wedge ((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite))))$
 by *auto*
have 34: $\vdash (f \wedge \neg (\neg (init\ w) \wedge empty) \wedge$
 $\neg ((init\ w) \wedge (((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite)))) =$
 $(f \wedge ((init\ w) \vee more) \wedge$
 $(\neg (init\ w) \vee \neg (((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite))))$
 by (*auto simp: empty-d-def*)
have 35: $\vdash (f \wedge ((init\ w) \vee more) \wedge$
 $(\neg (init\ w) \vee \neg (((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite)))) =$
 $((f \wedge (init\ w) \wedge \neg (((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite))) \vee$
 $(f \wedge (init\ w) \wedge \neg (init\ w)) \vee$
 $(f \wedge more \wedge \neg (((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite))) \vee$
 $(f \wedge more \wedge \neg (init\ w)))$
 by *auto*
have 36: $\vdash (f \wedge \neg (\text{while } (init\ w) \text{ do } g \wedge finite)) =$
 $((f \wedge (init\ w) \wedge \neg (((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite))) \vee$
 $(f \wedge (init\ w) \wedge \neg (init\ w)) \vee$
 $(f \wedge more \wedge \neg (((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite))) \vee$
 $(f \wedge more \wedge \neg (init\ w)))$
 by (*metis 32 33 34 35 int-eq*)
have 37: $\vdash \neg (f \wedge more \wedge \neg (init\ w))$
 using 1 by (*auto simp: empty-d-def*)
have 38: $\vdash (f \wedge more \wedge \neg (((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite))) \longrightarrow$
 $((g \wedge more) \wedge finite); f \wedge$
 $\neg (((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite)))$
 using 1 2 by (*auto simp: empty-d-def Valid-def*)
have 39: $\vdash (f \wedge (init\ w) \wedge \neg (((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite))) \longrightarrow$
 $((g \wedge more) \wedge finite); f \wedge$
 $\neg (((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite)))$
 using 2 by *auto*
have 40: $\vdash ((f \wedge (init\ w) \wedge \neg (((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite))) \vee$
 $(f \wedge (init\ w) \wedge \neg (init\ w)) \vee$
 $(f \wedge more \wedge \neg (((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite))) \vee$
 $(f \wedge more \wedge \neg (init\ w))) \longrightarrow$
 $((g \wedge more) \wedge finite); f \wedge$
 $\neg (((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite)))$
 using 39 38 37 38 by *fastforce*
have 4: $\vdash f \wedge \neg (\text{while } (init\ w) \text{ do } g \wedge finite) \longrightarrow$
 $((g \wedge more) \wedge finite); f \wedge$
 $\neg (((g \wedge more) \wedge finite); (\text{while } (init\ w) \text{ do } g \wedge finite))$
 by (*meson 36 40 Prop11 lift-imp-trans*)
have 50: $\vdash g \wedge more \longrightarrow more$

by *auto*
have 5: $\vdash (g \wedge \text{more}) \wedge \text{finite} \longrightarrow \text{more}$
 by (*simp add: 50 Prop05 Prop07 finite-d-def*)
have 6: $\vdash f \wedge \text{finite} \longrightarrow (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite})$
 using 4 5 *ChopContraB* by *blast*
from 6 **show** *?thesis* by (*simp add: Prop12*)
qed

lemma *SWhileIntro*:

assumes $\vdash \neg (\text{init } w) \wedge f \longrightarrow \text{empty}$
 $\vdash \text{init } w \wedge f \longrightarrow (g \wedge \text{more}) \frown f$
shows $\vdash f \wedge \text{finite} \longrightarrow \text{swhile } (\text{init } w) \text{ do } g$
proof –
have 1: $\vdash f \wedge \text{finite} \longrightarrow \text{while } (\text{init } w) \text{ do } g$
 using *assms*
 using *assms*
unfolding *schop-d-def*
 using *WhileIntro*[*of w f g*]
 by *blast*
have 2: $\vdash f \wedge \text{finite} \longrightarrow \text{while } (\text{init } w) \text{ do } g \wedge \text{finite}$
 using 1 by *auto*
show *?thesis*
 using 2
SWhile-While[*of LIFT (init w) g*]
While-import-finite[*of LIFT (init w) g*]
 by *fastforce*
qed

lemma *WhileElim*:

assumes $\vdash \neg (\text{init } w) \wedge \text{empty} \longrightarrow g$
 $\vdash \text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite}); g \longrightarrow g$
shows $\vdash \text{while } (\text{init } w) \text{ do } f \wedge \text{finite} \longrightarrow g$
proof –
have 1: $\vdash (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})))$
 by (*rule WhileEqvEmptyOrChopWhile*)
hence 11: $\vdash ((\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) \wedge \neg g) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}))) \wedge \neg g$
 by (*metis inteq-reflection lift-and-com*)
have 2: $\vdash \neg (\text{init } w) \wedge \text{empty} \longrightarrow g$
 using *assms* by *blast*
hence 21: $\vdash \neg g \longrightarrow \neg (\neg (\text{init } w) \wedge \text{empty})$
 by *auto*
have 22: $\vdash ((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}))) \wedge \neg g \longrightarrow$
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}))$
 using 21 by *auto*
have 23: $\vdash (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) \wedge \neg g \longrightarrow$

$(init\ w \wedge ((f \wedge more) \wedge finite); (while\ (init\ w)\ do\ f \wedge finite)) \wedge \neg\ g$
using 11 21 **by** fastforce
have 3: $\vdash (init\ w) \wedge (((f \wedge more) \wedge finite); g) \longrightarrow g$
using assms **by** blast
hence 31: $\vdash \neg\ g \longrightarrow \neg((init\ w) \wedge (((f \wedge more) \wedge finite); g))$
by fastforce
have 32: $\vdash (init\ w \wedge ((f \wedge more) \wedge finite); (while\ (init\ w)\ do\ f \wedge finite)) \wedge \neg\ g \longrightarrow$
 $((((f \wedge more) \wedge finite); (while\ (init\ w)\ do\ f \wedge finite)) \wedge$
 $\neg\ (((f \wedge more) \wedge finite); g)) \wedge \neg\ g$
using 31 **by** fastforce
have 4: $\vdash (while\ (init\ w)\ do\ f \wedge finite) \wedge \neg\ g \longrightarrow$
 $((f \wedge more) \wedge finite); (while\ (init\ w)\ do\ f \wedge finite)) \wedge$
 $\neg\ (((f \wedge more) \wedge finite); g)$
by (meson 23 32 Prop12 lift-imp-trans)
have 5: $\vdash (f \wedge more) \wedge finite \longrightarrow more$
by auto
from 4 5 **show** ?thesis **using**
ChopContraB[of LIFT(while (init w) do f ∧ finite) LIFT(g) LIFT(((f ∧ more) ∧ finite))]
by auto
qed

lemma SWhileElim:

assumes $\vdash \neg\ (init\ w) \wedge empty \longrightarrow g$
 $\vdash init\ w \wedge (f \wedge more) \frown g \longrightarrow g$
shows $\vdash swhile\ (init\ w)\ do\ f \longrightarrow g$
using assms
unfolding schop-d-def
using WhileElim[of w g f] SWhile-While[of LIFT (init w) f]
While-import-finite[of LIFT (init w) f]
by fastforce

lemma BaWhileImpWhile:

$\vdash ba\ (f \longrightarrow g) \wedge finite \longrightarrow (while\ (init\ w)\ do\ f) \longrightarrow (while\ (init\ w)\ do\ g)$
proof –
have 1: $\vdash (f \longrightarrow g) \longrightarrow ((init\ w \wedge f) \longrightarrow (init\ w \wedge g))$
by auto
hence 2: $\vdash ba\ (f \longrightarrow g) \longrightarrow ba\ ((init\ w \wedge f) \longrightarrow (init\ w \wedge g))$
using BaImpBa **by** blast
have 3: $\vdash ba\ ((init\ w \wedge f) \longrightarrow (init\ w \wedge g)) \wedge finite \longrightarrow ((init\ w \wedge f)^* \longrightarrow (init\ w \wedge g)^*)$
by (rule BaCSImpCS)
have 4: $\vdash ba\ (f \longrightarrow g) \wedge finite \longrightarrow ((init\ w \wedge f)^* \wedge fin\ (\neg\ (init\ w))$
 $\longrightarrow (init\ w \wedge g)^* \wedge fin\ (\neg\ (init\ w)))$
using 2 3 **by** fastforce
from 4 **show** ?thesis **by** (simp add: while-d-def)
qed

lemma SBaSWhileImpSWhile:

$\vdash sba\ (f \longrightarrow g) \longrightarrow (swhile\ (init\ w)\ do\ f) \longrightarrow (swhile\ (init\ w)\ do\ g)$
proof –

have 1: $\vdash (f \longrightarrow g) \longrightarrow ((\text{init } w \wedge f) \longrightarrow (\text{init } w \wedge g))$
by *auto*
hence 2: $\vdash \text{ sba } (f \longrightarrow g) \longrightarrow \text{ sba } ((\text{init } w \wedge f) \longrightarrow (\text{init } w \wedge g))$
by (rule *SBaImpSBa*)
have 3: $\vdash \text{ sba } ((\text{init } w \wedge f) \longrightarrow (\text{init } w \wedge g)) \longrightarrow$
 $(\text{schopstar } (\text{init } w \wedge f) \longrightarrow \text{schopstar } (\text{init } w \wedge g))$
by (rule *SBaSCSImpSCS*)
have 4: $\vdash \text{ sba } (f \longrightarrow g) \longrightarrow (\text{schopstar } (\text{init } w \wedge f) \wedge \text{ sfin } (\neg (\text{init } w)) \longrightarrow$
 $\text{schopstar } (\text{init } w \wedge g) \wedge \text{ sfin } (\neg (\text{init } w)))$
using 2 3 **by** *fastforce*
from 4 **show** ?thesis **by** (simp add: *swhile-d-def*)
qed

lemma *WhileImpWhile*:

assumes $\vdash f \longrightarrow g$
shows $\vdash (\text{while } (\text{init } w) \text{ do } f) \wedge \text{finite} \longrightarrow (\text{while } (\text{init } w) \text{ do } g)$
proof –
have 1: $\vdash f \longrightarrow g$
using *assms* **by** *auto*
hence 2: $\vdash \text{ ba } (f \longrightarrow g)$
by (rule *BaGen*)
have 3: $\vdash \text{ ba } (f \longrightarrow g) \wedge \text{finite} \longrightarrow (\text{while } (\text{init } w) \text{ do } f) \longrightarrow (\text{while } (\text{init } w) \text{ do } g)$
by (rule *BaWhileImpWhile*)
have 4: $\vdash \text{ ba } (f \longrightarrow g) \longrightarrow (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) \longrightarrow (\text{while } (\text{init } w) \text{ do } g)$
using 3 **by** (auto simp: *Valid-def*)
from 2 4 **show** ?thesis **using** *MP* **by** *blast*
qed

lemma *SWhileImpSWhile*:

assumes $\vdash f \longrightarrow g$
shows $\vdash (\text{swhile } (\text{init } w) \text{ do } f) \longrightarrow (\text{swhile } (\text{init } w) \text{ do } g)$
proof –
have 1: $\vdash f \longrightarrow g$
using *assms* **by** *auto*
hence 2: $\vdash \text{ sba } (f \longrightarrow g)$
by (rule *SBaGen*)
have 3: $\vdash \text{ sba } (f \longrightarrow g) \longrightarrow (\text{swhile } (\text{init } w) \text{ do } f) \longrightarrow (\text{swhile } (\text{init } w) \text{ do } g)$
by (rule *SBaSWhileImpSWhile*)
from 2 3 **show** ?thesis **using** *MP* **by** *blast*
qed

lemma *NotSkipEqvEmptyOrSkipChopMore*:

$\vdash (\neg \text{skip}) = (\text{empty} \vee \text{skip}; \text{more})$
using *ChopEmpty WnextEqvEmptyOrNext* **unfolding** *next-d-def wnext-d-def empty-d-def*
by (metis *inteq-reflection*)

lemma *FiniteInfSplitAnd*:

$\vdash f = ((f \vee \text{inf}) \wedge (f \vee \text{finite}))$
unfolding *finite-d-def* **by** *fastforce*

lemma *ChopAndInf-alt-1:*

$\vdash ((f;g) \wedge \text{inf}) = (f;(g \wedge \text{inf}) \vee (f \wedge \text{inf});(g \wedge \text{finite}))$

proof –

have 1: $\vdash ((f;g) \wedge \text{inf}) = (((f \wedge \text{finite});g \wedge \text{inf}) \vee ((f \wedge \text{inf});g \wedge \text{inf}))$

by (*metis ChopAndInf OrChopEqv OrFiniteInf inteq-reflection*)

have 2: $\vdash ((f \wedge \text{finite});g \wedge \text{inf}) = (f \wedge \text{finite});(g \wedge \text{inf})$

by (*simp add: ChopAndInf*)

have 3: $\vdash ((f \wedge \text{inf});g \wedge \text{inf}) = ((f \wedge \text{inf});(g \wedge \text{inf}) \vee (f \wedge \text{inf});(g \wedge \text{finite}))$

by (*metis AndInfChopEqvAndInf ChopAndInf int-simps(27) inteq-reflection*)

have 4: $\vdash ((f \wedge \text{finite});(g \wedge \text{inf}) \vee (f \wedge \text{inf});(g \wedge \text{inf})) = (f;(g \wedge \text{inf}))$

by (*metis 1 ChopAndInf inteq-reflection*)

have 5: $\vdash (((f \wedge \text{finite});g \wedge \text{inf}) \vee ((f \wedge \text{inf});g \wedge \text{inf})) = (f;(g \wedge \text{inf}) \vee (f \wedge \text{inf});(g \wedge \text{finite}))$

using 2 3 4 **by** *fastforce*

show *?thesis*

by (*meson 1 5 Prop04 Prop11*)

qed

lemma *ChopAndInf-alt-2:*

$\vdash ((f;g) \wedge \text{inf}) = ((f \wedge \text{finite});(g \wedge \text{inf}) \vee (f \wedge \text{inf});(g \wedge \text{inf}) \vee ((f \wedge \text{inf});(g \wedge \text{finite})))$

proof –

have 1: $\vdash (f;(g \wedge \text{inf})) = ((f \wedge \text{finite});(g \wedge \text{inf}) \vee (f \wedge \text{inf});(g \wedge \text{inf}))$

using *OrChopEqvRule OrFiniteInf* **by** *blast*

show *?thesis* **using** *ChopAndInf-alt-1 1* **by** *fastforce*

qed

lemma *AndEmptyChopAndMore:*

$\vdash ((f \wedge \text{empty});g \wedge \text{more}) = (f \wedge \text{empty});(g \wedge \text{more})$

proof –

have 1: $\vdash (f \wedge \text{empty});g = (\text{init } f \wedge g)$

by (*meson InitAndEmptyEqvAndEmpty LeftChopEqvChop Prop04 StateAndEmptyChop*)

have 2: $\vdash ((f \wedge \text{empty});g \wedge \text{more}) = (\text{init } f \wedge g \wedge \text{more})$

using 1 **by** *fastforce*

have 3: $\vdash (\text{init } f \wedge g \wedge \text{more}) = (f \wedge \text{empty});(g \wedge \text{more})$

by (*meson InitAndEmptyEqvAndEmpty LeftChopEqvChop Prop04 StateAndEmptyChop*)

show *?thesis* **using** 2 3

by (*meson Prop04 Prop11*)

qed

lemma *OrChopOr:*

$\vdash (f1 \vee f2);(g1 \vee g2) = (f1;g1 \vee f1;g2 \vee f2;g1 \vee f2;g2)$

using *ChopOrEqv[of LIFT (f1 \vee f2) g1 g2] OrChopEqv[of f1 f2 g1] OrChopEqv[of f1 f2 g2]*

by *fastforce*

lemma *ChopAndMore-alt-1:*

$\vdash (f;g \wedge \text{more}) = ((f \wedge \text{empty});(g \wedge \text{more}) \vee (f \wedge \text{more});g)$

proof –

have 1: $\vdash (f;g) = ((f \wedge \text{empty});g \vee (f \wedge \text{more});g)$

using *EmptyOrMoreSplit OrChopEqvRule* **by** *blast*

have 2: $\vdash (f;g \wedge \text{more}) = (((f \wedge \text{empty});g \wedge \text{more}) \vee ((f \wedge \text{more});g \wedge \text{more}))$
using 1 **by** *fastforce*
have 3: $\vdash ((f \wedge \text{empty});g \wedge \text{more}) = (f \wedge \text{empty});(g \wedge \text{more})$
using *AndEmptyChopAndMore* **by** *blast*
have 4: $\vdash ((f \wedge \text{more});g \wedge \text{more}) = ((f \wedge \text{more});g)$
by (*meson AndChopB MoreChopImpMore Prop04 Prop10 int-simps(4) lift-imp-trans*)
show *?thesis*
by (*metis 2 3 4 int-eq*)
qed

lemma *ChopAndMore-alt-2:*

$\vdash (f;g \wedge \text{more}) = ((f \wedge \text{empty});(g \wedge \text{more}) \vee (f \wedge \text{more});(g \wedge \text{empty}) \vee (f \wedge \text{more});(g \wedge \text{more}))$
using *ChopAndMore-alt-1[of f g]*
by (*meson ChopOrEqvRule EmptyOrMoreSplit Prop06*)

lemma *WPowerstar-And-Inf-Eqv-WPowerplus-And-Inf:*

$\vdash (\text{wpowerstar } f \wedge \text{inf}) = (\text{wpowerstar } f;f \wedge \text{inf})$
proof –
have 1: $\vdash \text{wpowerstar } f = (\text{empty} \vee \text{wpowerstar } f;f)$
by (*meson ChopEmpty Prop04 WPowerstar-unfoldr-eq*)
have 2: $\vdash ((\text{empty} \vee \text{wpowerstar } f;f) \wedge \text{inf}) = (\text{wpowerstar } f;f \wedge \text{inf})$
using *EmptyImpFinite unfolding finite-d-def* **by** *fastforce*
show *?thesis* **using** 1 2
by *fastforce*
qed

lemma *WPowerstar-Inf-Import:*

$\vdash ((\text{wpowerstar } f) \wedge \text{inf}) = ((\text{wpowerstar } f) \wedge \text{finite});(f \wedge \text{inf})$
proof –
have 2: $\vdash ((\text{istar } f) \wedge \text{inf}) = ((\text{wpowerstar } f) \wedge \text{inf})$
by (*metis IStarWPowerstar int-eq lift-and-com*)
have 3: $\vdash (\text{wpowerstar } f) = (\text{empty} \vee (\text{wpowerstar } f;f))$
by (*meson WPowerstar-unfoldR WPowerstar-unfoldr-eq int-iffD2 int-iffI*)
have 4: $\vdash ((\text{empty} \vee (\text{wpowerstar } f;f)) \wedge \text{inf}) = ((\text{wpowerstar } f;f) \wedge \text{inf})$
using *EmptyImpFinite unfolding finite-d-def* **by** *fastforce*
have 6: $\vdash \text{empty} \longrightarrow \text{inf} \longrightarrow (\text{wpowerstar } f \wedge \text{finite});(f \wedge \text{inf})$
using *EmptyImpFinite unfolding finite-d-def* **by** *fastforce*
have 61: $\vdash (f;((\text{wpowerstar } f \wedge \text{finite});(f \wedge \text{inf})) \wedge \text{inf}) =$
 $((f \wedge \text{inf}) \vee ((f \wedge \text{finite});((\text{wpowerstar } f \wedge \text{finite});(f \wedge \text{inf})) \wedge \text{inf})))$
by (*simp add: ChopAndInfB*)
have 62: $\vdash ((f \wedge \text{finite});((\text{wpowerstar } f \wedge \text{finite});(f \wedge \text{inf})) \wedge \text{inf}) =$
 $((f \wedge \text{finite});((\text{wpowerstar } f \wedge \text{finite});(f \wedge \text{inf}))))$
by (*metis (no-types, lifting) ChopAndInfB Prop03 Prop10 Prop12 inteq-reflection*)
have 63: $\vdash (f \wedge \text{inf}) \wedge \text{inf} \longrightarrow (\text{wpowerstar } f \wedge \text{finite});(f \wedge \text{inf})$
by (*meson ChopEqvChop Prop03 Prop10 Prop11 Prop12 WPowerstarChopEqvChopOrRule WPowerstar-import-finite lift-and-com*)
have 64: $\vdash ((f \wedge \text{finite});((\text{wpowerstar } f \wedge \text{finite});(f \wedge \text{inf})))) \wedge \text{inf} \longrightarrow (\text{wpowerstar } f \wedge \text{finite});(f \wedge \text{inf})$
by (*metis (no-types, lifting) ChopAndInf ChopAssoc EmptyOrChopEqv Prop03 Prop12 WPowerstarEqv*)

WPowerstar-import-finite inteq-reflection)
have 7: $\vdash f; (wpowerstar\ f \wedge finite); (f \wedge inf) \longrightarrow (wpowerstar\ f \wedge finite); (f \wedge inf)$
using 61 63 64 *ChopAndInf* **by** *fastforce*
have 8: $\vdash (f; finite \wedge inf) = (f \wedge inf)$
using *ChopAndInfB*[of *f LIFT finite*] **unfolding** *finite-d-def*
by (*metis AndInfEqvChopFalse ChopAndInf FiniteInfSplitAnd finite-d-def int-simps(26) inteq-reflection lift-and-com*)
have 9: $\vdash f; finite \wedge inf \longrightarrow (wpowerstar\ f \wedge finite); (f \wedge inf)$
by (*metis 7 8 ChopAndInf-alt-1 PowerstarEqvSemhelp2 Prop03 inteq-reflection lift-imp-trans*)
have 10: $\vdash (inf \longrightarrow (wpowerstar\ f \wedge finite); (f \wedge inf)) =$
 $(finite \vee (wpowerstar\ f \wedge finite); (f \wedge inf))$
unfolding *finite-d-def* **by** *fastforce*
have 11: $\vdash f; (inf \longrightarrow (wpowerstar\ f \wedge finite); (f \wedge inf)) = ((f; finite) \vee f; ((wpowerstar\ f \wedge finite); (f \wedge inf)))$
by (*metis 10 ChopOrEqv inteq-reflection*)
have 12: $\vdash empty \vee f; (inf \longrightarrow (wpowerstar\ f \wedge finite); (f \wedge inf)) \longrightarrow inf \longrightarrow (wpowerstar\ f \wedge finite); (f \wedge inf)$
by (*metis 11 6 7 9 Prop02 Prop09 int-eq*)
have 13: $\vdash ((istar\ f) \longrightarrow (inf \longrightarrow ((wpowerstar\ f) \wedge finite); (f \wedge inf)))$
using *IStarWeakInduct*[of *f LIFT (inf \longrightarrow ((wpowerstar\ f) \wedge finite); (f \wedge inf))*]]
using 12 **by** *blast*
have 14: $\vdash ((wpowerstar\ f) \wedge finite); (f \wedge inf) \longrightarrow ((wpowerstar\ f); f \wedge inf)$
by (*meson AndChopA ChopAndInf int-iffD2 lift-imp-trans*)
show *?thesis*
by (*metis 10 13 14 2 3 4 InfEqvNotFinite Prop07 int-iffI inteq-reflection*)
qed

lemma *Prop-Var-Empty-Finite-Import:*

$\vdash ((\$p \wedge empty) \wedge finite) = (\$p \wedge empty)$

using *EmptyImpFinite* **by** *auto*

lemma *Not-Prop-Var-And-Empty-Inf:*

$\vdash \neg((\$p \wedge empty) \wedge inf)$

using *FiniteAndEmptyEqvEmpty InfEqvNotFinite* **by** *fastforce*

lemma *Init-And-Finite-Absorb:*

$\vdash init\ (f \wedge finite) = init\ f$

by (*metis EmptySChop Initprop(1) init-d-def int-simps(17) inteq-reflection lift-and-com schop-d-def*)

lemma *Chop-And-More:*

$\vdash (f; g \wedge more) = ((f \wedge more); g \vee f; (g \wedge more))$

proof –

have 1: $\vdash f; g =$

$((f \wedge empty); (g \wedge empty) \vee (f \wedge empty); (g \wedge more) \vee$
 $(f \wedge more); (g \wedge empty) \vee (f \wedge more); (g \wedge more))$

by (*meson ChopEqvChop EmptyOrMoreSplit OrChopOr Prop04 Prop11*)

have 2: $\vdash (f; g \wedge more) = ($

$((f \wedge empty); (g \wedge empty) \wedge more) \vee$
 $((f \wedge empty); (g \wedge more) \wedge more) \vee$
 $((f \wedge more); (g \wedge empty) \wedge more) \vee$

```

      ( (f ∧ more);(g ∧ more) ∧ more) )
    using 1 by fastforce
  have 3: ⊢ ( ( (f ∧ empty);(g ∧ empty) ∧ more) ) =
    ( (f;g ∧ empty) ∧ more)
    by (metis ChopAndEmptyEqvEmptyChopEmpty int-simps(1) inteq-reflection)
  have 4: ⊢ ¬ ( (f;g ∧ empty) ∧ more)
    unfolding empty-d-def by fastforce
  have 5: ⊢ ( (f ∧ empty);(g ∧ more) ∧ more) = ( (f ∧ empty);(g ∧ more) )
    by (meson ChopAndB ChopEmpty ChopMoreImpMore Prop04 Prop10 lift-imp-trans)
  have 6: ⊢ ( (f ∧ more);(g ∧ empty) ∧ more) = ( (f ∧ more);(g ∧ empty) )
    using AndChopB MoreChopImpMore by fastforce
  have 7: ⊢ ( (f ∧ more);(g ∧ more) ∧ more) = ( (f ∧ more);(g ∧ more) )
    by (meson ChopAndB ChopMoreImpMore Prop10 Prop11 lift-imp-trans)
  have 8: ⊢ (f ∧ more);g = ( (f ∧ more);(g ∧ empty) ∨ (f ∧ more);(g ∧ more) )
    using ChopOrEqvRule EmptyOrMoreSplit by blast
  have 9: ⊢ f;(g ∧ more) = ( (f ∧ empty);(g ∧ more) ∨ (f ∧ more);(g ∧ more) )
    by (simp add: EmptyOrMoreSplit OrChopEqvRule)
  have 10: ⊢ ( (f ∧ more);g ∨ f;(g ∧ more) ) =
    ( (f ∧ more);(g ∧ empty) ∨ (f ∧ more);(g ∧ more) ∨
      (f ∧ empty);(g ∧ more) ∨ (f ∧ more);(g ∧ more) )
    using 8 9 by fastforce
  have 11: ⊢ ( ( (f ∧ empty);(g ∧ more) ) ∨ ( (f ∧ more);(g ∧ empty) ) ∨ ( (f ∧ more);(g ∧ more) ) ) =
    ( (f ∧ more);g ∨ f;(g ∧ more) )
    using 10 8 9 by fastforce
  show ?thesis
  by (meson 11 ChopAndMore-alt-2 Prop11 lift-imp-trans)
qed

end

```

```

theory Omega
imports Chopstar
begin

```

This theory defines the omega operator for infinite ITL and provides a library of lemmas. We also define a weak version womega that corresponds to the omega operator from Omega Algebra [1]. We also provide a semantic version aomega and provide lemmas that express various relationships between them. We also ported the numerous omega algebra lemmas from [1] to ITL.

2.6 Omega and variants

2.6.1 Definitions

Weak Omega

lemma *MoreChopSemMono* [mono]:

$$\begin{aligned}
 (w \models (f \wedge \text{more}) ; g) = \\
 ((\exists n. 0 < n \wedge \text{enat } n \leq \text{nlength } w \wedge f \ (\text{ntaken } n \ w)) \wedge g \ (\text{ndropn } n \ w)) \vee
 \end{aligned}$$

$(\neg \text{finite } w \wedge f w)$
by (*simp add: itl-defs*)
(metis bot-nat-0.not-eq-extremum le-zero-eq nlength-eq-enat-nfiniteD the-enat.simps zero-enat-def)

coinductive *womega-d* :: ('a::world) formula \Rightarrow 'a formula
for *F* **where**
 $(s \models (F \wedge \text{more}); (\text{womega-d } F)) \Longrightarrow (s \models (\text{womega-d } F))$

syntax
 $\text{-womega-d} :: \text{lift} \Rightarrow \text{lift} \quad ((\text{-}\mathcal{W}) [85] 85)$

syntax (*ASCII*)
 $\text{-womega-d} :: \text{lift} \Rightarrow \text{lift} \quad ((\text{womega } -) [85] 85)$

translations
 $\text{-womega-d} \quad \quad \quad \Longleftrightarrow \text{CONST womega-d}$

lemma *WOmegaIntros*:
 $\vdash (f \wedge \text{more}); (\text{womega } f) \longrightarrow (\text{womega } f)$
using *womega-d.intros* **using** *unl-lift2* **by** *blast*

lemma *WOmegaCases*:
 $(w \models (\text{womega } F)) \Longrightarrow$
 $(w \models (F \wedge \text{more}); (\text{womega } F) \Longrightarrow P) \Longrightarrow P$
using *womega-d.cases[of F w P]* **by** *auto*

lemma *WOmegaUnrollSem*:
 $(s \models (\text{womega } f)) = (s \models (f \wedge \text{more}); (\text{womega } f))$
using *womega-d.cases[of f]*
by (*metis womega-d.simps*)

lemma *WOmegaUnroll*:
 $\vdash (\text{womega } f) = (f \wedge \text{more}); (\text{womega } f)$
using *WOmegaUnrollSem*
using *unl-lift2* **by** *blast*

lemma *WOmegaCoinductSem*:
assumes $\bigwedge x. X x \Longrightarrow x \models (F \wedge \text{more}); (X \vee \text{womega } F)$
shows $x \models X \longrightarrow \text{womega } F$
using *assms womega-d.coinduct[of X x F]*
by (*auto simp add: chop-defs*)

lemma *WOmegaCoinduct*:
assumes $\vdash X \longrightarrow (F \wedge \text{more}); (X \vee (\text{womega } F))$
shows $\vdash X \longrightarrow (\text{womega } F)$
using *assms WOmegaCoinductSem[of X F]*
by (*simp add: Valid-def*)

lemma *WOmegaWeakCoinductSem*:
assumes $\bigwedge x. x \models X \implies x \models (F \wedge \text{more}); X$
shows $x \models X \longrightarrow \text{womega } F$
using *assms womega-d.coinduct[of X x F]*
by (*auto simp add: chop-defs*)

lemma *WOmegaWeakCoinduct*:
assumes $\vdash X \longrightarrow (F \wedge \text{more}); X$
shows $\vdash X \longrightarrow (\text{womega } F)$
using *assms WOmegaWeakCoinductSem[of X F]*
by (*simp add: Valid-def*)

Omega

lemma *FMoreSem-var* [*mono*]:
 $(w \models ((f \wedge \text{more}) \wedge \text{finite}); g) =$
 $((0 < \text{nlength } w \wedge (\exists n. f ((\text{ntaken } n \ w)) \wedge 0 < n \wedge g (\text{ndropn } n \ w)))))$
by (*simp add: itl-defs*)
(metis enat-0-iff(1) linorder-not-less ndropn-all neq0-conv
nfinite-nlength-enat nfinite-ntaken ntaken-all order-le-less)

definition *omega-d* :: (*'a::world*) *formula* \Rightarrow *'a formula*
where *omega-d* *F* $\equiv \text{LIFT}(\text{womega}(F \wedge \text{finite}))$

syntax
 $\text{-omega-d} :: \text{lift} \Rightarrow \text{lift} \quad ((-\omega) \text{ [85] 85})$

syntax (*ASCII*)
 $\text{-omega-d} :: \text{lift} \Rightarrow \text{lift} \quad ((\text{omega } -) \text{ [85] 85})$

translations
 $\text{-omega-d} \quad \Rightarrow \text{CONST } \text{omega-d}$

lemma *Chop-more-finite-swap*:
 $\vdash ((f \wedge \text{more}) \wedge \text{finite}) = ((f \wedge \text{finite}) \wedge \text{more})$
by *fastforce*

lemma *OmegaIntros*:
 $\vdash (f \wedge \text{more}) \frown (\text{omega } f) \longrightarrow (\text{omega } f)$
unfolding *omega-d-def schop-d-def*
by (*meson Chop-more-finite-swap LeftChopImpChop Prop11 WOmegaIntros lift-imp-trans*)

lemma *OmegaCases*:
 $(w \models (\text{omega } F)) \implies$
 $(w \models (F \wedge \text{more}) \frown (\text{omega } F) \implies P) \implies P$
unfolding *omega-d-def schop-d-def* **using** *womega-d.cases[of LIFT (F \wedge finite) w]*
by (*metis Chop-more-finite-swap inteq-reflection*)

lemma *OmegaUnrollSem*:

$(s \models (\text{omega } f)) = (s \models (f \wedge \text{more}) \frown (\text{omega } f))$

unfolding *omega-d-def schop-d-def*

by (*metis Chop-more-finite-swap WOmegaUnroll inteq-reflection*)

lemma *OmegaCoinductSem*:

assumes $\bigwedge x. X x \implies x \models (F \wedge \text{more}) \frown (X \vee \text{omega } F)$

shows $x \models X \longrightarrow \text{omega } F$

using *assms unfolding omega-d-def schop-d-def*

by (*metis Chop-more-finite-swap WOmegaCoinductSem inteq-reflection*)

lemma *OmegaWeakCoinductSem*:

assumes $\bigwedge x. X x \implies x \models (F \wedge \text{more}) \frown X$

shows $x \models X \longrightarrow \text{omega } F$

using *assms unfolding omega-d-def schop-d-def*

by (*metis Chop-more-finite-swap WOmegaWeakCoinductSem inteq-reflection*)

lemma *OmegaUnroll*:

$\vdash f^\omega = (f \wedge \text{more}) \frown f^\omega$

using *OmegaUnrollSem unl-lift2 by blast*

lemma *OmegaCoinduct*:

assumes $\vdash X \longrightarrow (F \wedge \text{more}) \frown (X \vee (\text{omega } F))$

shows $\vdash X \longrightarrow (\text{omega } F)$

using *assms OmegaCoinductSem[of X F]*

by (*simp add: Valid-def*)

lemma *OmegaWeakCoinduct*:

assumes $\vdash X \longrightarrow (F \wedge \text{more}) \frown X$

shows $\vdash X \longrightarrow (\text{omega } F)$

using *assms OmegaWeakCoinductSem[of X F]*

by (*simp add: Valid-def*)

Alternative definition for Omega

lemma *infinite-nidx-imp-infinite-interval*:

assumes $\neg \text{nfinite } l$

$\text{nidx } l$

$(\text{nnth } l \ 0) = 0$

$\forall i. (\text{nnth } l \ i) \leq \text{nlength } s$

shows $\neg \text{nfinite } s$

proof

assume $\text{nfinite } s$

thus *False*

using *assms*

proof (*induct s rule: nfinite-induct*)

case (*NNil y*)

then show *?case*

by (*metis dual-order.antisym enat-ord-simps(1) linorder-linear nfinite-ntaken nidx-gr-first nlength-NNil not-gr-zero ntaken-all zero-le zero-less-Suc*)

```

next
case (NCons x nell)
then show ?case
proof -
have 1:  $\bigwedge j. (nnth\ l\ j) < (nnth\ l\ (Suc\ j))$ 
by (metis assms(1) assms(2) linorder-le-cases nfinite-ntaken nidx-expand ntaken-all)
have 2:  $\forall i. enat\ (nnth\ l\ i) \leq nlength\ (NCons\ x\ nell) \implies False$ 
by (metis 1 NCons.hyps(2) assms(1) assms(2) assms(3) dual-order.strict-iff-order
    enat-ord-simps(1) iless-Suc-eq linorder-not-le nlength-NCons)
show ?thesis
using 2 NCons.prem(4) by auto
qed
qed
qed

```

definition *aomega-d* :: ('a::world) formula \Rightarrow 'a formula
where *aomega-d* *F* \equiv
 $(\lambda s.$
 $(\exists (l:: nat\ nellist).$
 $\neg nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l \wedge$
 $(\forall i. (nnth\ l\ i) \leq nlength\ s) \wedge$
 $(\forall i. ((nsubn\ s\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models F))$
 $)$
 $)$

syntax
-aomega-d :: lift \Rightarrow lift $((aomega\ -)\ [85]\ 85)$

syntax (ASCII)
-aomega-d :: lift \Rightarrow lift $((aomega\ -)\ [85]\ 85)$

translations
-aomega-d $\rightleftharpoons CONST\ aomega-d$

lemma *aomega-unroll-chain-a*:
assumes $(\exists l. \neg nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l \wedge$
 $(\forall i. (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
 $(\forall i. f\ ((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))))$
shows $(\exists n. enat\ n \leq nlength\ \sigma \wedge$
 $f\ ((ntaken\ n\ \sigma)) \wedge$
 $0 < n \wedge$
 $enat\ 0 < nlength\ \sigma \wedge$
 $(\exists l.$
 $\neg nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l \wedge$
 $(\forall i. (nnth\ l\ i) \leq nlength\ \sigma - enat\ n) \wedge$
 $(\forall i. f\ ((nsubn\ (ndropn\ n\ \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))))$
 $)$
 $)$

proof –

obtain l **where** $1: \neg \text{nfinite } l \wedge (\text{nnth } l \ 0) = 0 \wedge \text{nidx } l \wedge$
 $(\forall i. (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge (\forall i. f \ (\ (\text{nsubn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))))))$

using *assms* **by** *auto*

have $2: l = \text{NCons } (\text{nnth } l \ 0) \ (\text{ndropn } 1 \ l)$
by (*metis* $1 \ \text{One-nat-def } \text{gr-zeroI } \text{ndropn-0 } \text{ndropn-Suc-conv-ndropn } \text{nlength-eq-enat-nfiniteD}$
 zero-enat-def)

have $3: (\text{nnth } l \ 0) = 0$
using 1 **by** *blast*

have $4: (\text{nnth } l \ 0) < (\text{nnth } l \ 1)$
using 1
by (*metis* $\text{One-nat-def } \text{Suc-ile-eq } \text{enat-defs}(1) \ \text{nidx-gr-first } \text{nlength-eq-enat-nfiniteD}$
 $\text{not-gr-zero } \text{zero-less-one}$)

have $5: \text{nidx } (\text{ndropn } 1 \ l)$
using $1 \ \text{nidx-expand}[of \ l] \ \text{nidx-expand}[of \ (\text{ndropn } 1 \ l)]$
by (*metis* $\text{dual-order.order-iff-strict } \text{enat-defs}(1) \ \text{ndropn-all } \text{ndropn-nnth } \text{nfinite-ndropn-b}$
 $\text{nlength-NNil } \text{nlength-eq-enat-nfiniteD } \text{not-le-imp-less } \text{plus-1-eq-Suc}$)

have $6: f \ (\ (\text{nsubn } \sigma \ (\text{nnth } l \ 0) \ (\text{nnth } l \ 1)))$
by (*metis* $1 \ \text{One-nat-def}$)

have $7: (\forall i. f \ (\ (\text{nsubn } \sigma \ (\text{nnth } (\text{ndropn } 1 \ l) \ i) \ (\text{nnth } (\text{ndropn } 1 \ l) \ (\text{Suc } i))))$
by (*simp* *add: 1*)

have $8: f \ (\ (\text{ntaken } (\text{nnth } l \ 1) \ \sigma))$
by (*metis* $1 \ 4 \ \text{One-nat-def } \text{Suc-diff-1 } \text{Suc-diff-Suc } \text{ndropn-0 } \text{nsubn-def1}$)

have $81: \neg \text{nfinite } (\text{nmap } (\lambda e. e - (\text{nnth } l \ 1)) \ (\text{ndropn } 1 \ l))$
by (*simp* *add: 1*)

have $82: \bigwedge j. 0 < j \longrightarrow (\text{nnth } l \ 1) < \text{nnth } (\text{ndropn } 1 \ l) \ j$
by (*metis* $1 \ \text{Suc-diff-1 } \text{enat-defs}(1) \ \text{linorder-le-cases } \text{ndropn-all } \text{ndropn-nnth } \text{nfinite-ndropn-b}$
 $\text{nidx-less } \text{nlength-NNil } \text{nlength-eq-enat-nfiniteD } \text{plus-1-eq-Suc}$)

have $83: \bigwedge j. \text{nnth } (\text{nmap } (\lambda e. e - \text{nnth } l \ 1) \ (\text{ndropn } 1 \ l)) \ j =$
 $(\text{nnth } l \ (\text{Suc } j)) - (\text{nnth } l \ 1)$
by (*metis* $81 \ \text{le-cases } \text{ndropn-nnth } \text{nfinite-ntaken } \text{nlength-nmap } \text{nnth-nmap } \text{ntaken-all } \text{plus-1-eq-Suc}$)

have $84: \bigwedge j. \text{nnth } (\text{nmap } (\lambda e. e - \text{nnth } l \ 1) \ (\text{ndropn } 1 \ l)) \ (\text{Suc } j) =$
 $(\text{nnth } l \ (\text{Suc } (\text{Suc } j))) - (\text{nnth } l \ 1)$
using 83 **by** *blast*

have $840: \bigwedge j. (\text{nnth } l \ 1) \leq (\text{nnth } l \ (\text{Suc } j))$
by (*metis* $1 \ \text{diff-add-zero } \text{diff-is-0-eq } \text{linorder-le-cases } \text{nfinite-ntaken } \text{nidx-less-eq } \text{ntaken-all}$
 plus-1-eq-Suc)

have $85: \bigwedge j. (\text{nnth } l \ (\text{Suc } j)) - (\text{nnth } l \ 1) < (\text{nnth } l \ (\text{Suc } (\text{Suc } j))) - (\text{nnth } l \ 1)$
by (*metis* $1 \ 840 \ \text{diff-less-mono } \text{linorder-le-cases } \text{nfinite-ntaken } \text{nidx-expand } \text{ntaken-all}$)

have $86: (\forall i. \text{enat } (\text{Suc } i) \leq \text{nlength } (\text{nmap } (\lambda e. e - \text{nnth } l \ 1) \ (\text{ndropn } 1 \ l)) \longrightarrow$
 $\text{nnth } (\text{nmap } (\lambda e. e - \text{nnth } l \ 1) \ (\text{ndropn } 1 \ l)) \ i <$
 $\text{nnth } (\text{nmap } (\lambda e. e - \text{nnth } l \ 1) \ (\text{ndropn } 1 \ l)) \ (\text{Suc } i))$
using $83 \ 85$ **by** *presburger*

have $9: \text{nidx } (\text{nmap } (\lambda e. e - (\text{nnth } l \ 1)) \ (\text{ndropn } 1 \ l))$
using $\text{nidx-expand}[of \ (\text{nmap } (\lambda e. e - (\text{nnth } l \ 1)) \ (\text{ndropn } 1 \ l))]$
using $83 \ 85$ **by** *presburger*

have $91: \bigwedge j. (\text{nnth } (\text{nmap } (\lambda e. e + (\text{nnth } l \ 1)) \ (\text{nmap } (\lambda e. e - (\text{nnth } l \ 1)) \ (\text{ndropn } 1 \ l))) \ j) =$
 $(\text{nnth } l \ (\text{Suc } j))$
by (*metis* $81 \ 82 \ 83 \ \text{One-nat-def } \text{add-Suc } \text{diff-Suc-1 } \text{diff-Suc-eq-diff-pred}$
 $\text{diff-add } \text{diff-is-0-eq } \text{le-refl } \text{linorder-le-cases } \text{ndropn-nnth } \text{nfinite-ntaken } \text{nlength-nmap}$)

$nnth\text{-}nmap\ not\text{-}le\text{-}imp\text{-}less\ ntaken\text{-}all\ order\text{-}less\text{-}imp\text{-}le\ plus\text{-}1\text{-}eq\text{-}Suc$
have 10: $(\forall i. f\ ((nsubn\ \sigma$
 $\quad (nnth\ (nmap\ (\lambda e. e + (nnth\ l\ 1))\ (nmap\ (\lambda e. e - (nnth\ l\ 1))\ (ndropn\ 1\ l)))\ i)$
 $\quad (nnth\ (nmap\ (\lambda e. e + (nnth\ l\ 1))\ (nmap\ (\lambda e. e - (nnth\ l\ 1))\ (ndropn\ 1\ l)))\ (Suc\ i))\)))$
using 1 91 by presburger
have 11: $(\forall i. f\ ((nsubn\ \sigma$
 $\quad ((nnth\ (nmap\ (\lambda e. e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ i) + (nnth\ l\ 1))$
 $\quad ((nnth\ (nmap\ (\lambda e. e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ (Suc\ i)) + (nnth\ l\ 1))$
 $\quad)))$
using 7 83 840 by fastforce
have 12: $(\forall i. f\ ((nsubn\ (ndropn\ (nnth\ l\ 1)\ \sigma)$
 $\quad ((nnth\ (nmap\ (\lambda e. e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ i))$
 $\quad ((nnth\ (nmap\ (\lambda e. e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ (Suc\ i)))$
 $\quad)))$
by (metis 11 83 85 nsubn-ndropn)
have 121: $nnth\ (nmap\ (\lambda e. e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ 0 = 0$
using 83 One-nat-def by presburger
have 122: $\neg nfinite\ (nmap\ (\lambda e. e - (nnth\ l\ 1))\ (ndropn\ 1\ l)) \wedge$
 $nnth\ (nmap\ (\lambda e. e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ 0 = 0 \wedge$
 $nidx\ (nmap\ (\lambda e. e - (nnth\ l\ 1))\ (ndropn\ 1\ l)) \wedge$
 $(\forall i. f\ ((nsubn\ (ndropn\ (nnth\ l\ 1)\ \sigma)$
 $\quad ((nnth\ (nmap\ (\lambda e. e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ i))$
 $\quad ((nnth\ (nmap\ (\lambda e. e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ (Suc\ i)))$
 $\quad)))$
using 12 121 81 9 by blast
have 123: $(\forall i. (nnth\ (nmap\ (\lambda e. e - (nnth\ l\ 1))\ (ndropn\ 1\ l))\ i) \leq nlength\ (ndropn\ (nnth\ l\ 1)\ \sigma))$
by (meson 1 enat-ile infinite-nidx-imp-infinite-interval le-cases nfinite-ndropn-b
 $nlength\text{-}eq\text{-}enat\text{-}nfiniteD)$
have 13: $(\exists ls.$
 $\quad \neg nfinite\ ls \wedge (nnth\ ls\ 0) = 0 \wedge nidx\ ls \wedge$
 $\quad (\forall i. (nnth\ ls\ i) \leq nlength\ (ndropn\ (nnth\ l\ 1)\ \sigma)) \wedge$
 $\quad (\forall i. f\ ((nsubn\ (ndropn\ (nnth\ l\ 1)\ \sigma)\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))$
 $)$
using 122 123 by blast
from 13 show ?thesis using 4 8
by (metis 1 enat-ord-simps(1) linorder-not-less ndropn-nlength not-gr-zero zero-enat-def)
qed

lemma aomega-unroll-chain-b:

assumes $(\exists n. enat\ n \leq nlength\ \sigma \wedge$
 $\quad f\ ((ntaken\ n\ \sigma)) \wedge$
 $\quad 0 < n \wedge$
 $\quad enat\ 0 < nlength\ \sigma \wedge$
 $(\exists l.$
 $\quad \neg nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l \wedge$
 $\quad (\forall i. (nnth\ l\ i) \leq nlength\ \sigma - enat\ n) \wedge$
 $\quad (\forall i. f\ ((nsubn\ (ndropn\ n\ \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))))$
 $)$
 $)$

shows $(\exists l. \neg \text{nfinite } l \wedge (\text{nnth } l \ 0) = 0 \wedge \text{nidx } l \wedge$
 $(\forall i. (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. f \ (\ (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))))))$
proof –
obtain n **where** $1: \text{enat } n \leq \text{nlength } \sigma \wedge f \ (\ (\text{ntaken } n \ \sigma)) \wedge$
 $0 < n \wedge \text{enat } 0 < \text{nlength } \sigma \wedge$
 $(\exists l.$
 $\neg \text{nfinite } l \wedge (\text{nnth } l \ 0) = 0 \wedge \text{nidx } l \wedge$
 $(\forall i. (\text{nnth } l \ i) \leq \text{nlength } \sigma - \text{enat } n) \wedge$
 $(\forall i. f \ (\ (\text{nsbn } (\text{ndropn } n \ \sigma) \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))))))$
 $)$
using *assms* **by** *auto*
have $2: (\exists l.$
 $\neg \text{nfinite } l \wedge (\text{nnth } l \ 0) = 0 \wedge \text{nidx } l \wedge$
 $(\forall i. (\text{nnth } l \ i) \leq \text{nlength } (\text{ndropn } n \ \sigma)) \wedge$
 $(\forall i. f \ (\ (\text{nsbn } (\text{ndropn } n \ \sigma) \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))))))$
 $)$
using 1 **by** *auto*
obtain l **where** $3: \neg \text{nfinite } l \wedge (\text{nnth } l \ 0) = 0 \wedge \text{nidx } l \wedge$
 $(\forall i. (\text{nnth } l \ i) \leq \text{nlength } (\text{ndropn } n \ \sigma)) \wedge$
 $(\forall i. f \ (\ (\text{nsbn } (\text{ndropn } n \ \sigma) \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))))))$
using 2 **by** *auto*
have $4: \text{nidx } (\text{nmap } (\lambda e. e + n) \ l)$
using 3
by (*simp add: Suc-ile-eq nidx-expand*)
have $42: \bigwedge j. \text{nnth } (\text{NCons } 0 \ (\text{nmap } (\lambda e. e + n) \ l)) \ j =$
 $(\text{case } j \text{ of } 0 \Rightarrow 0 \mid$
 $(\text{Suc } k) \Rightarrow \text{nnth } (\text{nmap } (\lambda e. e + n) \ l) \ k)$
by (*simp add: nnth-NCons*)
have $43: \bigwedge j. \text{nnth } (\text{NCons } 0 \ (\text{nmap } (\lambda e. e + n) \ l)) \ (\text{Suc } j) =$
 $\text{nnth } (\text{nmap } (\lambda e. e + n) \ l) \ j$
by *simp*
have $44: 0 < \text{nnth } (\text{nmap } (\lambda e. e + n) \ l) \ 0$
using 1 *enat-defs(1)* **by** *auto*
have $45: \bigwedge j. \text{nnth } (\text{nmap } (\lambda e. e + n) \ l) \ j < \text{nnth } (\text{nmap } (\lambda e. e + n) \ l) \ (\text{Suc } j)$
by (*metis 3 4 add.right-neutral le-cases nfinite-ntaken nidx-less nlength-nmap ntaken-all*)
have $46: (\forall i. \text{enat } (\text{Suc } i) \leq \text{nlength } (\text{NCons } 0 \ (\text{nmap } (\lambda e. e + n) \ l))) \longrightarrow$
 $\text{nnth } (\text{NCons } 0 \ (\text{nmap } (\lambda e. e + n) \ l)) \ i <$
 $\text{nnth } (\text{NCons } 0 \ (\text{nmap } (\lambda e. e + n) \ l)) \ (\text{Suc } i)$
by (*metis 43 44 45 lessI less-Suc-eq-0-disj nnth-0*)
have $5: \text{nidx } (\text{NCons } 0 \ (\text{nmap } (\lambda e. e + n) \ l))$
using 46 *nidx-expand* **by** *blast*
have $6: (\ (\text{ntaken } n \ \sigma)) =$
 $(\ (\text{nsbn } \sigma \ (\text{nnth } (\text{NCons } 0 \ (\text{nmap } (\lambda e. e + n) \ l)) \ 0)$
 $(\text{nnth } (\text{NCons } 0 \ (\text{nmap } (\lambda e. e + n) \ l)) \ (\text{Suc } 0)) \))$
by (*metis 3 43 44 One-nat-def Suc-diff-1 Suc-diff-Suc add-0 ndropn-0 nnth-0 nnth-nmap*
nsbn-def1 zero-enat-def zero-le)
have $7: (\forall i. f \ (\ (\text{nsbn } (\text{ndropn } n \ \sigma) \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))))))$
using 3 **by** *auto*
have $8: (\forall i. f \ (\ (\text{nsbn } \sigma \ ((\text{nnth } l \ i) + n) \ ((\text{nnth } l \ (\text{Suc } i)) + n) \)))$

```

using 7
by (simp add: add.commute ndropn-ndropn nsubn-def1)
have 9: ( $\forall i. f \ ( \ (nsubn \ \sigma \ ( \ (nnth \ (nmap \ (\lambda e. e + n) \ l) \ i))$ 
      ( $\ ( \ (nnth \ (nmap \ (\lambda e. e + n) \ l) \ (Suc \ i))) \ ) \ )$ )

using 8
by (metis 45 Suc-ile-eq dual-order.order-iff-strict linorder-le-cases nat-neq-iff nlength-nmap
      nnth-beyond nnth-nmap)
have 10:  $f \ ( \ (nsubn \ \sigma \ (nnth \ (NCons \ 0 \ (nmap \ (\lambda e. e + n) \ l)) \ 0)$ 
      ( $\ (nnth \ (NCons \ 0 \ (nmap \ (\lambda e. e + n) \ l)) \ (Suc \ 0))))$ )

using 1 6 by auto
have 11: ( $\forall i. i > 0 \longrightarrow$ 
       $f \ ( \ (nsubn \ \sigma \ (nnth \ (NCons \ 0 \ (nmap \ (\lambda e. e + n) \ l)) \ i)$ 
      ( $\ (nnth \ (NCons \ 0 \ (nmap \ (\lambda e. e + n) \ l)) \ (Suc \ i))))$ )
by (metis 9 Suc-diff-1 nnth-Suc-NCons)
have 12: ( $\forall i.$ 
       $f \ ( \ (nsubn \ \sigma \ (nnth \ (NCons \ 0 \ (nmap \ (\lambda e. e + n) \ l)) \ i)$ 
      ( $\ (nnth \ (NCons \ 0 \ (nmap \ (\lambda e. e + n) \ l)) \ (Suc \ i))))$ )

using 10 11 neg0-conv by blast
have 13:  $\forall i. (nnth \ (NCons \ 0 \ (nmap \ (\lambda e. e + n) \ l)) \ i) \leq nlength \ \sigma$ 
by (metis 3 add.commute enat-ile infinite-nidx-imp-infinite-interval linorder-le-cases ndropn-all
      ndropn-ndropn nfinite-ndropn-b nlength-NNil nlength-eq-enat-nfiniteD zero-le)
show ?thesis using 12 5
by (metis 13 3 nfinite-NCons nfinite-nmap nnth-0)
qed

```

lemma *aomega-unroll-chain:*

```

( $\exists l. \neg nfinite \ l \wedge (nnth \ l \ 0) = 0 \wedge nidx \ l \wedge$ 
  ( $\forall i. (nnth \ l \ i) \leq nlength \ \sigma$ )  $\wedge$ 
  ( $\forall i. f \ ( \ (nsubn \ \sigma \ (nnth \ l \ i) \ (nnth \ l \ (Suc \ i))))$ )
=
( $\exists n. enat \ n \leq nlength \ \sigma \wedge$ 
   $f \ ((ntaken \ n \ \sigma)) \wedge$ 
   $0 < n \wedge$ 
   $enat \ 0 < nlength \ \sigma \wedge$ 
  ( $\exists l.$ 
    ( $\neg nfinite \ l \wedge (nnth \ l \ 0) = 0 \wedge nidx \ l \wedge$ 
      ( $\forall i. (nnth \ l \ i) \leq nlength \ \sigma - enat \ n$ )  $\wedge$ 
      ( $\forall i. f \ ( \ (nsubn \ (ndropn \ n \ \sigma) \ (nnth \ l \ i) \ (nnth \ l \ (Suc \ i))))$ )
    )
  )
)

```

```

using aomega-unroll-chain-a[of  $\sigma \ f$ ] aomega-unroll-chain-b[of  $\sigma \ f$ ]
by blast

```

lemma *aomega-unroll-sem:*

```

( $\sigma \models ((f \wedge more) \wedge finite); (aomega \ f) = (aomega \ f))$ 

```

proof

```

(simp add: itl-defs zero-enat-def aomega-d-def)

```

```

show ( $\exists n. enat \ n \leq nlength \ \sigma \wedge$ 
   $f \ (ntaken \ n \ \sigma) \wedge$ 

```

$0 < n \wedge$
 $enat\ 0 < nlength\ \sigma \wedge$
 $(\exists l. \neg nfinite\ l \wedge nnth\ l\ 0 = 0 \wedge nidx\ l \wedge$
 $(\forall i. enat\ (nnth\ l\ i) \leq nlength\ \sigma - enat\ n) \wedge$
 $(\forall i. f\ (nsubn\ (ndropn\ n\ \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))))) \vee$
 $\neg nfinite\ \sigma \wedge f\ \sigma \wedge enat\ 0 < nlength\ \sigma \wedge nfinite\ \sigma) =$
 $(\exists l. \neg nfinite\ l \wedge nnth\ l\ 0 = 0 \wedge nidx\ l \wedge (\forall i. enat\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
 $(\forall i. f\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))))))$
using *aomega-unroll-chain*[of $\sigma\ f$] **by** *blast*
qed

lemma *AOmegaUnroll*:

$\vdash (aomega\ f) = ((f \wedge more) \wedge finite);(aomega\ f)$

unfolding *Valid-def*

using *aomega-unroll-sem* **by** *auto*

lemma *AOmegaInductSem-help*:

$(\sigma \models inf \wedge g \wedge \Box(g \longrightarrow ((f \wedge more) \wedge finite);g)) =$
 $(\neg nfinite\ \sigma \wedge g\ \sigma \wedge$
 $(\forall n. g\ (ndropn\ n\ \sigma)) \longrightarrow$
 $(\exists na. f\ (nsubn\ \sigma\ n\ (na+n))) \wedge$
 $0 < na \wedge g\ (ndropn\ (n + na)\ \sigma)))$
 $)$

by (*simp add: itl-defs zero-enat-def min-def ndropn-ndropn nsubn-def1*)

(*metis linorder-le-cases ndropn-all ndropn-nlength nfinite-NNil nfinite-ndropn-b*)

primrec *cpoint* :: ('a::world) formula \Rightarrow 'a formula \Rightarrow nat \Rightarrow 'a intervals \Rightarrow nat

where *cpoint* $f\ g\ 0\ \sigma = 0$

| *cpoint* $f\ g\ (Suc\ n)\ \sigma =$

$(\epsilon\ x. (\exists\ m. (nsubn\ \sigma\ (cpoint\ f\ g\ n\ \sigma)\ (m+(cpoint\ f\ g\ n\ \sigma))) \models f)$
 $\wedge m > 0 \wedge ((ndropn\ (m+(cpoint\ f\ g\ n\ \sigma))\ \sigma) \models g) \wedge$
 $x = m + (cpoint\ f\ g\ n\ \sigma))$
 $)$

lemma *cpoint-expand-0*:

$(cpoint\ f\ g\ 0\ \sigma) = 0$

by *simp*

lemma *cpoint-expand-1*:

$(cpoint\ f\ g\ 1\ \sigma) =$

$(SOME\ x. (\exists\ m. f\ (nsubn\ \sigma\ 0\ (m)))$
 $\wedge m > 0 \wedge g\ (ndropn\ (m)\ \sigma))$
 $\wedge x = m))$

by (*simp add: itl-defs*)

lemma *cpoint-expand-n*:
 $(\text{cpoint } f \ g \ (\text{Suc } n) \ \sigma) =$
 $(\text{SOME } x. (\exists \ m. f \ (\ (\text{nsubn } \sigma \ (\text{cpoint } f \ g \ n \ \sigma) \ (m+(\text{cpoint } f \ g \ n \ \sigma))))$
 $\quad \wedge \ m > 0 \wedge g \ ((\text{ndropn } (m+(\text{cpoint } f \ g \ n \ \sigma)) \ \sigma))$
 $\quad \wedge \ x = m + (\text{cpoint } f \ g \ n \ \sigma))$
 $)$
by (*simp add: itl-defs*)

lemma *cpoint-0*:
assumes $\neg \text{nfinite } \sigma \wedge g \ \sigma \wedge$
 $(\forall k. g \ (\ (\text{ndropn } k \ \sigma)) \longrightarrow$
 $\quad (\exists m. f \ (\ (\text{nsubn } \sigma \ k \ (m+k))) \wedge$
 $\quad \quad 0 < m \wedge g \ (\ (\text{ndropn } (m+k) \ \sigma))))$

shows $g \ ((\text{ndropn } (\text{cpoint } f \ g \ i \ \sigma) \ \sigma))$
proof
(induct i)
case 0
then show ?case **by** (*simp add: assms*)
next
case (*Suc i*)
then show ?case
proof –
have 1: $g \ ((\text{ndropn } (\text{cpoint } f \ g \ i \ \sigma) \ \sigma))$
by (*simp add: Suc.hyps*)
have 2: $g \ ((\text{ndropn } (\text{cpoint } f \ g \ i \ \sigma) \ \sigma)) \longrightarrow$
 $(\exists m. f \ (\ (\text{nsubn } \sigma \ (\text{cpoint } f \ g \ i \ \sigma) \ (m+(\text{cpoint } f \ g \ i \ \sigma)))) \wedge$
 $\quad 0 < m \wedge g \ ((\text{ndropn } (m+(\text{cpoint } f \ g \ i \ \sigma)) \ \sigma))$
using *assms* **by** *blast*
have 3: $(\exists m. f \ (\ (\text{nsubn } \sigma \ (\text{cpoint } f \ g \ i \ \sigma) \ (m+(\text{cpoint } f \ g \ i \ \sigma)))) \wedge$
 $\quad 0 < m \wedge g \ (\ (\text{ndropn } (m+(\text{cpoint } f \ g \ i \ \sigma)) \ \sigma))$
using 1 2 **by** *auto*
have 4: $(\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma) =$
 $(\text{SOME } x. (\exists \ m. f \ (\ (\text{nsubn } \sigma \ (\text{cpoint } f \ g \ i \ \sigma) \ (m+(\text{cpoint } f \ g \ i \ \sigma))))$
 $\quad \wedge \ m > 0 \wedge g \ ((\text{ndropn } (m+(\text{cpoint } f \ g \ i \ \sigma)) \ \sigma))$
 $\quad \wedge \ x = m + (\text{cpoint } f \ g \ i \ \sigma))$
by *simp*
have 5: $g \ ((\text{ndropn } ((\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma)) \ \sigma))$
using 3 4 *someI-ex*[*of* $\lambda x. (\exists \ m. f \ (\ (\text{nsubn } \sigma \ (\text{cpoint } f \ g \ i \ \sigma) \ (m+(\text{cpoint } f \ g \ i \ \sigma))))$
 $\quad \wedge \ m > 0 \wedge g \ ((\text{ndropn } (m+(\text{cpoint } f \ g \ i \ \sigma)) \ \sigma))$
 $\quad \wedge \ x = m + (\text{cpoint } f \ g \ i \ \sigma))$] **by** *auto*
from 5 **show** ?thesis **by** *auto*
qed
qed

lemma *cpoint-1*:

assumes $\neg \text{nfinites } \sigma \wedge g \ \sigma \wedge$
 $(\forall k. g \ (\text{ndropn } k \ \sigma)) \longrightarrow$
 $(\exists m. f \ (\text{nsubn } \sigma \ k \ (m+k))) \wedge$
 $0 < m \wedge g \ (\text{ndropn } (m+k) \ \sigma))$

shows $(g \ ((\text{ndropn } (\text{cpoint } f \ g \ i \ \sigma) \ \sigma))$
 $\implies g \ ((\text{ndropn } (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma) \ \sigma)))$

using *assms cpoint-0* **by** *blast*

lemma *cpoint-2*:

assumes $\neg \text{nfinites } \sigma \wedge g \ \sigma \wedge$
 $(\forall k. g \ (\text{ndropn } k \ \sigma)) \longrightarrow$
 $(\exists m. f \ (\text{nsubn } \sigma \ k \ (m+k))) \wedge$
 $0 < m \wedge g \ (\text{ndropn } (m+k) \ \sigma))$

shows $f \ (\text{nsubn } \sigma \ (\text{cpoint } f \ g \ i \ \sigma) \ (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma))$

proof

(*induct i*)

case *0*

then show ?*case*

proof –

have *1*: $g \ ((\text{ndropn } 0 \ \sigma))$

using *assms cpoint-0 cpoint-expand-0* **by** *force*

have *2*: $(\exists m. f \ (\text{nsubn } \sigma \ (\text{cpoint } f \ g \ 0 \ \sigma) \ (m+(\text{cpoint } f \ g \ 0 \ \sigma)))) \wedge$
 $0 < m \wedge g \ (\text{ndropn } (m+(\text{cpoint } f \ g \ 0 \ \sigma)) \ \sigma))$

using *assms 1* **by** (*metis cpoint-expand-0*)

have *3*: $(\text{cpoint } f \ g \ 1 \ \sigma) =$
 $(\text{SOME } x. (\exists m. f \ ((\text{nsubn } \sigma \ (\text{cpoint } f \ g \ 0 \ \sigma) \ (m+(\text{cpoint } f \ g \ 0 \ \sigma)) \))$
 $\wedge m > 0 \wedge g \ (\text{ndropn } (m+(\text{cpoint } f \ g \ 0 \ \sigma)) \ \sigma))$
 $\wedge x = m+(\text{cpoint } f \ g \ 0 \ \sigma))$
 $)$

by *simp*

have *4*: $f \ (\text{nsubn } \sigma \ (\text{cpoint } f \ g \ 0 \ \sigma) \ ((\text{cpoint } f \ g \ 1 \ \sigma)) \))$

using *2 3 someI-ex[of $\lambda x. (\exists m. f \ (\text{nsubn } \sigma \ (\text{cpoint } f \ g \ 0 \ \sigma)$*
 $(m+(\text{cpoint } f \ g \ 0 \ \sigma)) \))$
 $\wedge m > 0 \wedge g \ (\text{ndropn } (m+(\text{cpoint } f \ g \ 0 \ \sigma)) \ \sigma))$
 $\wedge x = m+(\text{cpoint } f \ g \ 0 \ \sigma)]$ **by** *auto*

from *4* **show** ?*thesis* **by** *auto*

qed

next

case (*Suc i*)

then show ?*case*

proof –

have *n1*: $g \ (\text{ndropn } (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma) \ \sigma))$

using *assms cpoint-0* **by** *blast*

have *n2*: $(\exists m. f \ (\text{nsubn } \sigma \ (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma) \ (m+(\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma)))) \wedge$
 $0 < m \wedge g \ (\text{ndropn } (m+(\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma)) \ \sigma))$

using *assms n1* **by** *auto*

have $n3$: ($cpoint\ f\ g\ (Suc\ (Suc\ i))\ \sigma$) =
 $(SOME\ x. (\exists\ m. f\ (\ (nsubn\ \sigma\ (cpoint\ f\ g\ (Suc\ i)\ \sigma)\ (m+(cpoint\ f\ g\ (Suc\ i)\ \sigma))))$
 $\wedge\ m>0 \wedge g\ ((ndropn\ (m+(cpoint\ f\ g\ (Suc\ i)\ \sigma))\ \sigma))$
 $\wedge\ x=m+(cpoint\ f\ g\ (Suc\ i)\ \sigma))$
 $)$
using $cpoint\text{-}expand\text{-}n$ **by** $blast$
have $n4$: $f\ (\ (nsubn\ \sigma\ (cpoint\ f\ g\ (Suc\ i)\ \sigma)\ ((cpoint\ f\ g\ (Suc\ (Suc\ i))\ \sigma))))$
using $n2\ n3\ someI\text{-}ex$ [$of\ \lambda x. (\exists\ m. f\ (\ (nsubn\ \sigma\ (cpoint\ f\ g\ (Suc\ i)\ \sigma)$
 $(m+(cpoint\ f\ g\ (Suc\ i)\ \sigma))))$
 $\wedge\ m>0 \wedge g\ ((ndropn\ (m+(cpoint\ f\ g\ (Suc\ i)\ \sigma))\ \sigma))$
 $\wedge\ x=m+(cpoint\ f\ g\ (Suc\ i)\ \sigma))$] **by** $auto$
from $n4$ **show** $?thesis$ **by** $auto$
qed
qed

lemma $cpoint\text{-}3a$:
 $m>0 \wedge x=m+(cpoint\ f\ g\ (Suc\ i)\ \sigma) \implies (cpoint\ f\ g\ (Suc\ i)\ \sigma) < x$
by $auto$

lemma $cpoint\text{-}3$:
assumes $\neg nfinite\ \sigma \wedge g\ \sigma \wedge$
 $(\forall k. g\ (\ (ndropn\ k\ \sigma)) \longrightarrow$
 $(\exists m. f\ (\ (nsubn\ \sigma\ k\ (m+k))) \wedge$
 $0 < m \wedge g\ (\ (ndropn\ (m+k)\ \sigma))))$
shows $(cpoint\ f\ g\ i\ \sigma) < (cpoint\ f\ g\ (Suc\ i)\ \sigma)$

proof
 $(induct\ i)$
case 0
then show $?case$
proof –
have 1 : $g\ ((ndropn\ 0\ \sigma))$
using $assms\ cpoint\text{-}0\ cpoint\text{-}expand\text{-}0$ **by** $force$
have 2 : $(\exists m. f\ (\ (nsubn\ \sigma\ (cpoint\ f\ g\ 0\ \sigma)\ (m+(cpoint\ f\ g\ 0\ \sigma)))) \wedge$
 $0 < m \wedge g\ (\ (ndropn\ (m+(cpoint\ f\ g\ 0\ \sigma))\ \sigma))$
using $assms\ 1$ **by** $(metis\ cpoint\text{-}expand\text{-}0)$
have 3 : $(cpoint\ f\ g\ 1\ \sigma) =$
 $(SOME\ x. (\exists\ m. f\ (\ (nsubn\ \sigma\ (cpoint\ f\ g\ 0\ \sigma)\ (m+(cpoint\ f\ g\ 0\ \sigma))))$
 $\wedge\ m>0 \wedge g\ (\ (ndropn\ (m+(cpoint\ f\ g\ 0\ \sigma))\ \sigma))$
 $\wedge\ x=m+(cpoint\ f\ g\ 0\ \sigma))$
 $)$
by $simp$
have 4 : $(cpoint\ f\ g\ 0\ \sigma) < (cpoint\ f\ g\ 1\ \sigma)$
using $2\ 3\ someI\text{-}ex$ [$of\ \lambda x. (\exists\ m. f\ (\ (nsubn\ \sigma\ (cpoint\ f\ g\ 0\ \sigma)\ (m+(cpoint\ f\ g\ 0\ \sigma))))$
 $\wedge\ m>0 \wedge g\ (\ (ndropn\ (m+(cpoint\ f\ g\ 0\ \sigma))\ \sigma))$
 $\wedge\ x=m+(cpoint\ f\ g\ 0\ \sigma))$] **by** $auto$
from 4 **show** $?thesis$ **by** $auto$

```

qed
next
case (Suc i)
then show ?case
proof -
  have n1: g ( (ndropn (cpoint f g (Suc i) σ) σ))
    using assms cpoint-0 by blast
  have n2: (∃ m. f ( (nsubn σ (cpoint f g (Suc i) σ) (m+(cpoint f g (Suc i) σ)))) ∧
    0 < m ∧ g ( (ndropn (m+(cpoint f g (Suc i) σ)) σ)))
    using assms n1 by auto
  have n3: (cpoint f g (Suc (Suc i)) σ) =
    (SOME x. (∃ m. f ( (nsubn σ (cpoint f g (Suc i) σ) (m+(cpoint f g (Suc i) σ))))
      ∧ m > 0 ∧ g ( (ndropn (m+(cpoint f g (Suc i) σ)) σ))
      ∧ x = m+(cpoint f g (Suc i) σ))
    )
    using cpoint-expand-n by blast
  have n4: (∃ m. f ( (nsubn σ (cpoint f g (Suc i) σ) (m+(cpoint f g (Suc i) σ))))
    ∧ m > 0 ∧ g ( (ndropn (m+(cpoint f g (Suc i) σ)) σ))
    ∧ (cpoint f g (Suc (Suc i)) σ) = m+(cpoint f g (Suc i) σ))
    using n2 n3 someI-ex[of λx. (∃ m. f ( (nsubn σ (cpoint f g (Suc i) σ) (m+(cpoint f g (Suc i) σ))))
      ∧ m > 0 ∧ g ( (ndropn (m+(cpoint f g (Suc i) σ)) σ))
      ∧ x = m+(cpoint f g (Suc i) σ)) ] by auto
  have n5: (cpoint f g (Suc i) σ) < (cpoint f g (Suc (Suc i)) σ)
    using n4 using cpoint-3a by blast
  from n5 show ?thesis by auto
qed
qed

```

primcorec *cpl* :: ('a::world) formula ⇒ 'a formula ⇒ nat ⇒ 'a intervals ⇒ nat nellist
where *cpl* f g x σ = NCons (cpoint f g x σ) (cpl f g (Suc x) σ)

lemma

nnth (cpl f g 0 σ) 0 = 0

by (metis cpl.disc-iff cpl.simps(2) cpoint-expand-0 nhd-conv-nnth)

lemma

nnth (cpl f g 0 σ) 1 = cpoint f g 1 σ

by (metis One-nat-def cpl.code cpl.disc-iff cpl.simps(2) nhd-conv-nnth nnth-Suc-NCons)

lemma *nnth-cpl*:

nnth (cpl f g x σ) i = cpoint f g (x+i) σ

proof (induct i arbitrary: x)

case 0

then show ?case **by** (simp add: nnth-0-conv-nhd)

next

case (Suc i)

then show ?case **by** (metis add-Suc-shift cpl.simps(3) nnth-ntl)

qed

lemma *cpl-infinite*:

$\neg nfinite (cpl\ f\ g\ x\ \sigma)$

proof

assume $nfinite (cpl\ f\ g\ x\ \sigma)$

thus *False*

proof (*induct zs* $\equiv (cpl\ f\ g\ x\ \sigma)$ *arbitrary: x rule: nfinite-induct*)

case (*NNil y*)

then show ?*case* **by** (*metis cpl.disc-iff nellist.disc(1)*)

next

case (*NCons x nell*)

then show ?*case* **by** (*metis cpl.simps(3) nellist.sel(5)*)

qed

qed

lemma *AOmegaInductSem*:

$(w \models (inf \wedge g \wedge \Box(g \longrightarrow ((f \wedge more) \wedge finite);g)) \longrightarrow aomega\ f)$

proof –

have 1: $(w \models (inf \wedge g \wedge \Box(g \longrightarrow ((f \wedge more) \wedge finite);g))) =$
 $(\neg nfinite\ w \wedge g\ w \wedge$
 $(\forall k. g\ (ndropn\ k\ w) \longrightarrow$
 $(\exists m. f\ (nsubn\ w\ k\ (m+k))) \wedge$
 $0 < m \wedge g\ (ndropn\ (m+k)\ w))))$

using *AOmegaInductSem-help[of g f w]*

by (*simp add: add commute*)

have 2: $(\neg nfinite\ w \wedge g\ w \wedge$
 $(\forall k. g\ (ndropn\ k\ w) \longrightarrow$
 $(\exists m. f\ (nsubn\ w\ k\ (m+k))) \wedge$
 $0 < m \wedge g\ (ndropn\ (m+k)\ w)))) \longrightarrow$
 $nidx\ (cpl\ f\ g\ 0\ w)$

using *nidx-expand[of (cpl f g 0 w) nnth-cpl[of f g 0 w]*

by (*metis add-0 cpoint-3*)

have 3: $(\neg nfinite\ w \wedge g\ w \wedge$
 $(\forall k. g\ (ndropn\ k\ w) \longrightarrow$
 $(\exists m. f\ (nsubn\ w\ k\ (m+k))) \wedge$
 $0 < m \wedge g\ (ndropn\ (m+k)\ w)))) \longrightarrow$
 $(\forall i. f\ (nsubn\ w\ (nnth\ (cpl\ f\ g\ 0\ w)\ i)$
 $(nnth\ (cpl\ f\ g\ 0\ w)\ (Suc\ i))))$

using 1 *cpoint-2* **by** (*metis add-0 nnth-cpl*)

have 31: $(\neg nfinite\ w \wedge g\ w \wedge$
 $(\forall k. g\ (ndropn\ k\ w) \longrightarrow$
 $(\exists m. f\ (nsubn\ w\ k\ (m+k))) \wedge$
 $0 < m \wedge g\ (ndropn\ (m+k)\ w)))) \longrightarrow$
 $(\forall i. (nnth\ (cpl\ f\ g\ 0\ w)\ i) \leq nlength\ w)$

by (*simp add: nfinite-conv-nlength-enat*)

have 4: $(\neg nfinite\ w \wedge g\ w \wedge$
 $(\forall k. g\ (ndropn\ k\ w) \longrightarrow$
 $(\exists m. f\ (nsubn\ w\ k\ (m+k))) \wedge$
 $0 < m \wedge g\ (ndropn\ (m+k)\ w)))) \longrightarrow$

$$\neg \text{finite } w \wedge \neg \text{finite } (\text{cpl } f \text{ } g \text{ } 0 \text{ } w) \wedge \text{nnth } (\text{cpl } f \text{ } g \text{ } 0 \text{ } w) \text{ } 0 = 0 \wedge$$

$$\text{nidx } (\text{cpl } f \text{ } g \text{ } 0 \text{ } w) \wedge$$

$$(\forall i. (\text{nnth } (\text{cpl } f \text{ } g \text{ } 0 \text{ } w) \text{ } i) \leq \text{nlength } w) \wedge$$

$$(\forall i. f \text{ } ((\text{nsbn } w \text{ } (\text{nnth } (\text{cpl } f \text{ } g \text{ } 0 \text{ } w) \text{ } i) \text{ } (\text{nnth } (\text{cpl } f \text{ } g \text{ } 0 \text{ } w) \text{ } (\text{Suc } i))))))$$

using 2 3 31

by (simp add: cpl-infinite nnth-0-conv-nhd)

have 5: $(\neg \text{finite } w \wedge g \text{ } w \wedge$
 $(\forall k. g \text{ } (\text{ndropn } k \text{ } w)) \longrightarrow$
 $(\exists m. f \text{ } ((\text{nsbn } w \text{ } k \text{ } (m+k))) \wedge$
 $0 < m \wedge g \text{ } (\text{ndropn } (m+k) \text{ } w)))) \longrightarrow$
 $\neg \text{finite } w \wedge$
 $(\exists (ls). \neg \text{finite } ls \wedge \text{nnth } ls \text{ } 0 = 0 \wedge \text{nidx } ls \wedge$
 $(\forall i. (\text{nnth } ls \text{ } i) \leq \text{nlength } w) \wedge$
 $(\forall i. f \text{ } ((\text{nsbn } w \text{ } (\text{nnth } ls \text{ } i) \text{ } (\text{nnth } ls \text{ } (\text{Suc } i)))))) \wedge$
 $ls = (\text{cpl } f \text{ } g \text{ } 0 \text{ } w))$

using 4 **by** auto

have 6: $(\neg \text{finite } w \wedge g \text{ } w \wedge$
 $(\forall k. g \text{ } (\text{ndropn } k \text{ } w)) \longrightarrow$
 $(\exists m. f \text{ } ((\text{nsbn } w \text{ } k \text{ } (m+k))) \wedge$
 $0 < m \wedge g \text{ } (\text{ndropn } (m+k) \text{ } w)))) \longrightarrow (w \models \text{aomega } f)$

using 3 5 **unfolding** aomega-d-def **by** blast

have 7: $(w \models (\text{inf } \wedge g \wedge \Box(g \longrightarrow ((f \wedge \text{more}) \wedge \text{finite});g)) \longrightarrow (\text{aomega } f))$

using 6 1 **by** auto

from 7 **show** ?thesis **by** blast

qed

lemma AOmegaInduct:

$\vdash (\text{inf } \wedge g \wedge \Box(g \longrightarrow ((f \wedge \text{more}) \wedge \text{finite});g)) \longrightarrow \text{aomega } f$
unfolding Valid-def **using** AOmegaInductSem[of g f] **by** blast

lemma AOmegaWeakCoInduct:

assumes $\vdash g \longrightarrow ((f \wedge \text{more}) \wedge \text{finite});g$

shows $\vdash \text{inf } \wedge g \longrightarrow \text{aomega } f$

proof –

have 1: $\vdash \Box(g \longrightarrow ((f \wedge \text{more}) \wedge \text{finite});g)$

using assms **by** (simp add: BoxGen)

show ?thesis **using** assms AOmegaInduct[of g f]

using 1 **by** fastforce

qed

lemma Omega-WOmega:

$\vdash (\text{omega } f) = (\text{womega } (f \wedge \text{finite}))$

unfolding *omega-d-def* **by** *simp*

lemma *WOmegaInducthelp*:

$\vdash (g \wedge \Box(g \longrightarrow f;g))$
 \longrightarrow
 $f; (g \wedge \Box(g \longrightarrow f;g))$

proof –

have 1: $\vdash (g \wedge \Box(g \longrightarrow f;g)) = (g \wedge (g \longrightarrow f;g) \wedge \Box(g \longrightarrow f;g))$
using *BoxEqvAndBox[of LIFT g \longrightarrow f;g]* **by** *fastforce*
have 2: $\vdash (g \wedge (g \longrightarrow f;g) \wedge \Box(g \longrightarrow f;g)) = (g \wedge (f;g) \wedge \Box(g \longrightarrow f;g))$
by *fastforce*
have 4: $\vdash \Box(g \longrightarrow f;g) = \Box(\Box(g \longrightarrow f;g))$
by (*simp add: BoxEqvBoxBox*)
have 5: $\vdash (g \wedge (f;g) \wedge \Box(g \longrightarrow f;g)) = (g \wedge (f;g) \wedge \Box(\Box(g \longrightarrow f;g)))$
using 4 **by** *fastforce*
have 6: $\vdash (g \wedge (f;g) \wedge \Box(\Box(g \longrightarrow f;g))) \longrightarrow f; (g \wedge \Box(g \longrightarrow f;g))$
using *ChopAndBoxImport[of f g LIFT $\Box(g \longrightarrow f;g)$]*
by (*meson Prop01 Prop05*)
show *?thesis*
by (*metis 1 2 5 6 int-eq*)
qed

lemma *OmegaInducthelp*:

$\vdash (g \wedge \Box(g \longrightarrow ((f \wedge \text{more})) \frown g))$
 \longrightarrow
 $((f \wedge \text{more})) \frown (g \wedge \Box(g \longrightarrow ((f \wedge \text{more})) \frown g))$

using *WOmegaInducthelp* **unfolding** *schop-d-def* **by** *blast*

lemma *WOmegaInduct*:

$\vdash (g \wedge \Box(g \longrightarrow (f \wedge \text{more});g)) \longrightarrow \text{womega } f$
by (*simp add: WOmegaInducthelp WOmegaWeakCoinduct*)

lemma *OmegaInduct*:

$\vdash (g \wedge \Box(g \longrightarrow ((f \wedge \text{more})) \frown g)) \longrightarrow \text{omega } f$
by (*simp add: OmegaInducthelp OmegaWeakCoinduct*)

lemma *WOmega-coinduct*:

assumes $\vdash g \longrightarrow h \vee (f \wedge \text{more});g$
shows $\vdash g \longrightarrow (\text{womega } f) \vee (\text{wpowerstar } (f \wedge \text{more});h)$

proof –

have 1: $\vdash (\text{wpowerstar } (f \wedge \text{more});h) = (\text{empty} \vee (f \wedge \text{more});(\text{wpowerstar } (f \wedge \text{more})));h$
by (*simp add: LeftChopEqvChop WPowerstarEqv*)
have 2: $\vdash (\text{empty} \vee (f \wedge \text{more});(\text{wpowerstar } (f \wedge \text{more})));h = (h \vee ((f \wedge \text{more});(\text{wpowerstar } (f \wedge \text{more})));h)$

by (*simp add: EmptyOrChopEqv*)
have 3: $\vdash (\neg(h \vee ((f \wedge \text{more});(\text{wpowerstar } (f \wedge \text{more})));h)) = (\neg h \wedge \neg((f \wedge \text{more});(\text{wpowerstar } (f \wedge \text{more})));h))$
using *ChopAssoc* **by** *fastforce*

have 31: $\vdash (\neg (wpowerstar (f \wedge more); h)) = (\neg h \wedge \neg ((f \wedge more); (wpowerstar (f \wedge more); h)))$
by (metis 1 2 3 int-eq)
have 32: $\vdash ((\neg h \wedge \neg ((f \wedge more); (wpowerstar (f \wedge more); h))) \wedge (h \vee (f \wedge more); g)) \longrightarrow$
 $(\neg ((f \wedge more); (wpowerstar (f \wedge more); h))) \wedge (f \wedge more); g$
by force
have 33: $\vdash g \wedge \neg (wpowerstar (f \wedge more); h) \longrightarrow ((\neg h \wedge \neg ((f \wedge more); (wpowerstar (f \wedge more); h)))$
 $\wedge (h \vee (f \wedge more); g))$
using assms 31 **by** fastforce
have 4: $\vdash g \wedge \neg (wpowerstar (f \wedge more); h) \longrightarrow \neg ((f \wedge more); (wpowerstar (f \wedge more); h)) \wedge (f \wedge$
 $more); g$
using assms 31 32 33
using lift-imp-trans **by** blast
have 5: $\vdash \neg ((f \wedge more); (wpowerstar (f \wedge more); h)) \wedge (f \wedge more); g \longrightarrow (f \wedge more); (g \wedge \neg wpowerstar$
 $(f \wedge more); h)$
by (metis ChopAndNotChopImp int-eq lift-and-com)
have 6: $\vdash g \wedge \neg (wpowerstar (f \wedge more); h) \longrightarrow (f \wedge more); (g \wedge \neg wpowerstar (f \wedge more); h)$
using 4 5 lift-imp-trans **by** blast
have 7: $\vdash g \wedge \neg ((wpowerstar (f \wedge more); h) \longrightarrow (womega f)$
using WOmegaWeakCoinduct[of LIFT $g \wedge \neg ((wpowerstar (f \wedge more); h) f]$ 6
by blast
show ?thesis **using** 7 **by** fastforce
qed

lemma WOmega-coinduct-var:
assumes $\vdash g \longrightarrow h \vee (f \wedge more); g$
shows $\vdash g \longrightarrow (womega f) \vee (wpowerstar f); h$
using assms
by (metis WOmega-coinduct WPowerstar-more-absorb int-eq)

lemma WOmega-coinduct-no-more:
assumes $\vdash g \longrightarrow f; g$
shows $\vdash g \longrightarrow (womega f) \vee (wpowerstar f); init f$
proof –
have 1: $\vdash (f) = ((f \wedge empty) \vee (f \wedge more))$
using EmptyOrMoreSplit **by** auto
have 2: $\vdash (f; g) = ((f \wedge empty); g \vee (f \wedge more); g)$
by (simp add: 1 OrChopEqvRule)
have 3: $\vdash ((f \wedge empty); g) = (g \wedge init f)$
by (meson InitAndEmptyEqvAndEmpty LeftChopEqvChop Prop04 StateAndEmptyChop lift-and-com)
have 4: $\vdash g \longrightarrow init f \vee (f \wedge more); g$
using 2 3 assms **by** fastforce
show ?thesis **using** 4 WOmega-coinduct-var **by** blast
qed

lemma WOmega-coinduct-no-more-var:
assumes $\vdash g \longrightarrow h \vee f; g$
shows $\vdash g \longrightarrow (womega f) \vee (wpowerstar f); (h \vee init f)$
proof –
have 1: $\vdash (f) = ((f \wedge empty) \vee (f \wedge more))$
using EmptyOrMoreSplit **by** auto

have 2: $\vdash (f;g) = (f \wedge \text{empty});g \vee (f \wedge \text{more});g$
by (*simp add: 1 OrChopEqvRule*)
have 3: $\vdash ((f \wedge \text{empty});g) = (g \wedge \text{init } f)$
by (*meson InitAndEmptyEqvAndEmpty LeftChopEqvChop Prop04 StateAndEmptyChop lift-and-com*)
have 4: $\vdash g \longrightarrow (h \vee \text{init } f) \vee (f \wedge \text{more});g$
using *assms 2 3 by fastforce*
show ?thesis **using** 4 *WOmega-coinduct-var by blast*
qed

lemma *Omega-coinduct:*

assumes $\vdash g \longrightarrow h \vee (f \wedge \text{more}) \frown g$
shows $\vdash g \longrightarrow (\text{omega } f) \vee (\text{schopstar } f) \frown h$
using *assms WOmega-coinduct[of g h LIFT (f \wedge finite)]*
Omega-WOmega[of f] SChopstar-WPowerstar[of LIFT (f \wedge more)]
by (*metis Chop-more-finite-swap Prop10 SChopstar-WPowerstar SChopstar-finite WPowerstar-more-absorb*
inteq-reflection chop-d-def)

2.6.2 Omega algebra

lemma *WOmega-coinduct-var1:*

assumes $\vdash f \longrightarrow \text{empty} \vee (g \wedge \text{more});f$
shows $\vdash f \longrightarrow (\text{womega } g) \vee (\text{wpowerstar } g)$
using *assms WOmega-coinduct-var[of f LIFT empty g]*
by (*metis ChopEmpty inteq-reflection*)

lemma *WOmega-coinduct-no-more-var1:*

assumes $\vdash f \longrightarrow \text{empty} \vee g;f$
shows $\vdash f \longrightarrow (\text{womega } g) \vee (\text{wpowerstar } g);(\text{empty} \vee \text{init } g)$
using *assms WOmega-coinduct-no-more-var[of f LIFT empty g]* **by** *blast*

lemma *Omega-coinduct-var1:*

assumes $\vdash f \longrightarrow \text{empty} \vee (g \wedge \text{more}) \frown f$
shows $\vdash f \longrightarrow (\text{omega } g) \vee (\text{schopstar } g)$
using *assms*
by (*metis Omega-coinduct Prop08 SChop-SChopstar-Closure SChopstar-imp-empty SChopstardef int-iffD1*
schopstar-d-def)

lemma *WOmega-coinduct-eq:*

assumes $\vdash f = (h \vee (g \wedge \text{more});f)$
shows $\vdash f \longrightarrow (\text{womega } g) \vee (\text{wpowerstar } g);h$
using *assms WOmega-coinduct-var int-iffD1 by blast*

lemma *WOmega-coinduct-no-more-eq:*

assumes $\vdash f = (h \vee g;f)$
shows $\vdash f \longrightarrow (\text{womega } g) \vee (\text{wpowerstar } g);(h \vee \text{init } g)$
using *assms by (simp add: Prop11 WOmega-coinduct-no-more-var)*

lemma *Omega-coinduct-eq:*

assumes $\vdash f = (h \vee (g \wedge \text{more}) \frown f)$
shows $\vdash f \longrightarrow (\text{omega } g) \vee (\text{schopstar } g) \frown h$
using *assms* **by** (*simp add: Omega-coinduct Prop11*)

lemma *WOmega-coinduct-eq-var1*:
assumes $\vdash f = (\text{empty} \vee (g \wedge \text{more}); f)$
shows $\vdash f \longrightarrow (\text{womega } g) \vee (\text{wpowerstar } g)$
using *assms* *WOmega-coinduct-var1 int-iffD1* **by** *blast*

lemma *WOmega-coinduct-no-more-eq-var1*:
assumes $\vdash f = (\text{empty} \vee g; f)$
shows $\vdash f \longrightarrow (\text{womega } g) \vee (\text{wpowerstar } g); (\text{empty} \vee \text{init } g)$
using *assms* *WOmega-coinduct-no-more-eq* **by** *blast*

lemma *Omega-coinduct-eq-var1*:
assumes $\vdash f = (\text{empty} \vee (g \wedge \text{more}) \frown f)$
shows $\vdash f \longrightarrow (\text{omega } g) \vee (\text{schopstar } g)$
using *assms* **by** (*simp add: Omega-coinduct-var1 Prop11*)

lemma *WOmega-coinduct-eq-var2*:
assumes $\vdash f = (g \wedge \text{more}); f$
shows $\vdash f \longrightarrow (\text{womega } g)$
using *assms* **by** (*simp add: Prop11 WOmegaWeakCoinduct*)

lemma *WOmega-coinduct-no-more-eq-var2*:
assumes $\vdash f = g; f$
shows $\vdash f \longrightarrow (\text{womega } g) \vee (\text{wpowerstar } g); \text{init } g$
using *assms* *WOmega-coinduct-no-more int-iffD1* **by** *blast*

lemma *Omega-coinduct-eq-var2*:
assumes $\vdash f = (g \wedge \text{more}) \frown f$
shows $\vdash f \longrightarrow (\text{omega } g)$
using *assms* *OmegaWeakCoinduct int-iffD1* **by** *blast*

lemma *WOmega-subdist*:
 $\vdash (\text{womega } f) \longrightarrow (\text{womega } (f \vee g))$
proof –
have 0: $\vdash ((f \vee g) \wedge \text{more}) = ((f \wedge \text{more} \vee g \wedge \text{more}))$
by *fastforce*
have 1: $\vdash (\text{womega } f) \longrightarrow ((f \wedge \text{more} \vee g \wedge \text{more}); (\text{womega } f))$
using *OrChopEqv WOmegaCases* **by** *fastforce*
have 15: $\vdash ((f \wedge \text{more} \vee g \wedge \text{more}); (\text{womega } f)) = ((f \vee g) \wedge \text{more}); (\text{womega } f)$
using 0 **by** (*metis LeftChopEqvChop int-eq*)
have 2: $\vdash (\text{womega } f) \longrightarrow ((f \vee g) \wedge \text{more}); (\text{womega } f)$
by (*metis 0 1 int-eq*)
show ?thesis **by** (*simp add: 2 WOmegaWeakCoinduct*)
qed

lemma *Omega-subdist*:
 $\vdash (\text{omega } f) \longrightarrow (\text{omega } (f \vee g))$

by (*metis FiniteOr WOmega-subdist int-eq omega-d-def*)

lemma *WOmega-iso*:

assumes $\vdash f \longrightarrow g$

shows $\vdash (\text{womega } f) \longrightarrow (\text{womega } g)$

using *assms WOmega-subdist[of f g]*

by (*metis Prop02 Prop03 Prop11 int-simps(20) inteq-reflection*)

lemma *Omega-iso*:

assumes $\vdash f \longrightarrow g$

shows $\vdash (\text{omega } f) \longrightarrow (\text{omega } g)$

using *assms WOmega-iso[of LIFT (f \wedge finite) LIFT (g \wedge finite)]*

unfolding *omega-d-def*

by (*metis (no-types, opaque-lifting) AndChopA AndChopB ChopEmpty Prop12 inteq-reflection lift-imp-trans omega-d-def*)

lemma *WOmega-subdist-var*:

$\vdash (\text{womega } f) \vee (\text{womega } g) \longrightarrow (\text{womega } (f \vee g))$

by (*meson EmptyChop Prop02 Prop04 Prop05 Prop11 WOmega-iso WOmega-subdist*)

lemma *Omega-subdist-var*:

$\vdash (\text{omega } f) \vee (\text{omega } g) \longrightarrow (\text{omega } (f \vee g))$

by (*meson Omega-iso Omega-subdist Prop02 Prop04 Prop05 int-iffD2 int-simps(26)*)

lemma *WOmega-zero*:

$\vdash \neg(\text{womega } \#False)$

by (*metis AndInfChopEqvAndInf TrueW WOmegaUnroll int-simps(19) int-simps(3) inteq-reflection*)

lemma *Omega-zero*:

$\vdash \neg(\text{omega } \#False)$

using *Omega-WOmega[of LIFT #False] WOmega-zero* **by** *fastforce*

lemma *WOmega-star-1*:

$\vdash (\text{wpowerstar } f);(\text{womega } f) = (\text{womega } f)$

proof –

have 1: $\vdash (f \wedge \text{more}); (\text{womega } f) \longrightarrow (\text{womega } f)$

by (*simp add: WOmegaUnroll int-iffD2*)

have 2: $\vdash (\text{wpowerstar } f);(\text{womega } f) \longrightarrow (\text{womega } f)$

by (*meson 1 LeftChopImpChop WPowerstar-inductL-var-equiv WPowerstar-more-absorb int-iffD2 lift-imp-trans*)

have 3: $\vdash (\text{womega } f) \longrightarrow (\text{wpowerstar } f);(\text{womega } f)$

by (*metis 2 Prop03 Prop10 WPowerstarChopEqvChopOrRule inteq-reflection*)

show *?thesis*

by (*simp add: 2 3 int-iffI*)

qed

lemma *Omega-star-1:*

$\vdash (schopstar\ f) \frown (\omega\ f) = (\omega\ f)$

by (*meson OmegaUnroll Prop11 SChopstarMore-induct-lvar-eq2*)

lemma *WOmega-max-element:*

$\vdash f \longrightarrow (w\omega\ empty) \vee wpowerstar\ empty; init\ empty$

by (*metis EmptyChop WOmega-coinduct-no-more-eq-var2 inteq-reflection*)

lemma *WOmega-empty-zero:*

$\vdash \neg(w\omega\ empty)$

by (*metis Prop10 Prop12 WOmegaUnroll WOmegaWeakCoinduct WOmega-zero empty-d-def int-iffD2 int-simps(21) inteq-reflection lift-and-com*)

lemma *Omega-empty-zero:*

$\vdash \neg(\omega\ empty)$

by (*metis FiniteAndEmptyEqvEmpty WOmega-empty-zero int-eq lift-and-com omega-d-def*)

lemma *WOmega-1:*

$\vdash (w\omega\ f); g \longrightarrow (w\omega\ f)$

by (*meson ChopAssoc LeftChopEqvChop Prop11 WOmegaUnroll WOmegaWeakCoinduct lift-imp-trans*)

lemma *Omega-1:*

$\vdash (\omega\ f); g \longrightarrow (\omega\ f)$

by (*simp add: WOmega-1 omega-d-def*)

lemma *WOmega-sup-id:*

assumes $\vdash empty \longrightarrow g$

shows $\vdash (w\omega\ f); g = (w\omega\ f)$

by (*meson ChopEmpty Prop11 RightChopImpChop WOmega-1 assms lift-imp-trans*)

lemma *Omega-sup-id:*

assumes $\vdash empty \longrightarrow g$

shows $\vdash (\omega\ f); g = (\omega\ f)$

using *assms*

by (*metis Omega-WOmega WOmega-sup-id int-eq*)

lemma *WOmega-simulation:*

assumes $\vdash h; (f \wedge more) \longrightarrow (g \wedge more); h$

shows $\vdash h; (w\omega\ f) \longrightarrow (w\omega\ g)$

proof –

have 1: $\vdash h; (w\omega\ f) = h; ((f \wedge more); (w\omega\ f))$

by (*simp add: RightChopEqvChop WOmegaUnroll*)

have 2: $\vdash h; ((f \wedge more); (w\omega\ f)) \longrightarrow (g \wedge more); (h; (w\omega\ f))$

by (*metis AndChopB ChopAssoc Prop10 assms inteq-reflection*)

have 3: $\vdash h; (w\omega\ f) \longrightarrow (g \wedge more); (h; (w\omega\ f))$

by (*metis 1 2 int-eq*)

show *?thesis* **by** (*simp add: 3 WOmegaWeakCoinduct*)
qed

lemma *Omega-simulation*:

assumes $\vdash (h) \smallfrown (f \wedge \text{more}) \longrightarrow (g \wedge \text{more}) \smallfrown (h)$

shows $\vdash (h) \smallfrown (\text{omega } f) \longrightarrow (\text{omega } g)$

using *assms*

by (*metis AndSChopA OmegaUnroll OmegaWeakCoinduct Prop10 SChopAssoc inteq-reflection lift-and-com*)

lemma *WOmega-WOmega*:

$\vdash (\text{womega } (\text{womega } f)) \longrightarrow (\text{womega } f)$

by (*meson AndChopA Prop11 WOmegaUnroll WOmega-1 lift-imp-trans*)

lemma *WOmega-equiv*:

assumes $\vdash f = g$

shows $\vdash \text{womega } f = \text{womega } g$

by (*meson Prop11 WOmega-iso assms*)

lemma *WOmega-More-absorb*:

$\vdash \text{womega } f = \text{womega } (f \wedge \text{more})$

proof –

have 1: $\vdash f^{\mathcal{W}} \longrightarrow (f \wedge \text{more})^{\mathcal{W}}$

using *WOmega-coinduct-eq-var2[of LIFT womega f LIFT (f \wedge more)]*

by (*metis LeftChopImpChop Prop12 TrueW WOmegaUnroll WOmega-simulation int-simps(13) inteq-reflection*)

have 2: $\vdash (f \wedge \text{more})^{\mathcal{W}} \longrightarrow f^{\mathcal{W}}$

using *WOmega-coinduct-eq-var2[of LIFT womega (f \wedge more) f]*

by (*metis AndChopB WOmegaUnroll WOmega-simulation inteq-reflection lift-and-com*)

show *?thesis* **using** 1 2 **by** *fastforce*

qed

lemma *WOmega-more-equiv*:

assumes $\vdash f = g$

shows $\vdash \text{womega } (f \wedge \text{more}) = \text{womega } g$

by (*meson Prop04 WOmega-More-absorb WOmega-equiv assms*)

lemma *WOmega-Wagner-1*:

$\vdash (\text{womega } (\text{wpowerstar } f)) = (\text{womega } f)$

proof –

have 1: $\vdash (\text{womega } ((f \wedge \text{more}); \text{wpowerstar } f)) =$

$((f \wedge \text{more}); (\text{wpowerstar } f)); ((f \wedge \text{more}); (\text{wpowerstar } f)); (\text{womega } ((f \wedge \text{more}); \text{wpowerstar } f))$

by (*metis (no-types, lifting) ChopAssoc WOmegaUnroll WOmega-star-1 WPowerstar-more-absorb WpowerstarAndMoreEqvAndMoreChop inteq-reflection*)

have 2: $\vdash (((f \wedge \text{more});(\text{wpowerstar } f));((f \wedge \text{more});(\text{wpowerstar } f)));(\text{womega } ((f \wedge \text{more});\text{wpowerstar } f))$
 $=$
 $(f \wedge \text{more}); (\text{womega } ((f \wedge \text{more});\text{wpowerstar } f))$
by (*metis* (*no-types*, *lifting*) *AndChopB ChopAssoc Prop11 WOmegaUnroll WPowerstar-inductL-var-equiv WPowerstar-invol WPowerstar-more-absorb WpowerstarAndMoreEqvAndMoreChop inteq-reflection lift-and-com*)

have 3: $\vdash (\text{womega } ((f \wedge \text{more});\text{wpowerstar } f)) \longrightarrow (\text{womega } f)$
by (*metis* 1 2 *WOmega-coinduct-eq-var2 inteq-reflection*)
have 4: $\vdash (\text{womega } f) \longrightarrow (\text{womega } ((f \wedge \text{more});\text{wpowerstar } f))$
by (*metis* *WOmega-More-absorb WOmega-iso WPowerstar-ext WpowerstarAndMoreEqvAndMoreChop inteq-reflection*)
have 5: $\vdash (\text{wpowerstar } f \wedge \text{more}) = (f \wedge \text{more});\text{powerstar } f$
by (*metis* *ChopstarEqvPowerstar Chopstar-WPowerstar WPowerstar-more-absorb WpowerstarAndMoreEqvAndMoreChop inteq-reflection*)
have 6: $\vdash (\text{womega } (\text{wpowerstar } f)) = (\text{womega } ((f \wedge \text{more});\text{wpowerstar } f))$
by (*metis* 5 *ChopstarEqvPowerstar Chopstar-WPowerstar WOmega-More-absorb WPowerstar-more-absorb inteq-reflection*)
show ?thesis **using** 3 4 6 **by** *fastforce*
qed

lemma *Omega-Wagner-1:*

$\vdash (\text{omega } (\text{s chopstar } f)) = (\text{omega } f)$
by (*metis* *Prop10 SChopstar-WPowerstar SChopstar-finite WOmega-Wagner-1 inteq-reflection omega-d-def*)

lemma *Omega-Eqv-rule:*

assumes $\vdash f = g$
shows $\vdash \text{omega } f = \text{omega } g$
using *assms*
by (*simp add: Omega-iso Prop11*)

lemma *Omega-Wagner-1-var2:*

$\vdash (\text{omega } ((f \wedge \text{more}) \frown \text{s chopstar } f)) = (\text{omega } f)$

proof –

have 1: $\vdash ((f \wedge \text{more}) \frown \text{s chopstar } f \wedge \text{finite} \wedge \text{more}) = ((f \wedge \text{more}) \frown \text{s chopstar } f)$
by (*metis* *Prop10 Prop11 Prop12 SCSAndMoreEqvAndMoreSChop SChopImpFinite SChopstar-finite*)
have 2: $\vdash ((\text{s chopstar } f \wedge \text{finite}) \wedge \text{more}) = (f \wedge \text{more}) \frown (\text{s chopstar } f)$
by (*metis* *Prop10 SCSAndMoreEqvAndMoreSChop SChopstar-finite inteq-reflection*)
have 3: $\vdash (\text{omega } ((f \wedge \text{more}) \frown \text{s chopstar } f)) = (\text{omega } (\text{s chopstar } f))$
by (*metis* 2 *Omega-Wagner-1 SChopstar-and-more SChopstar-finite-absorb inteq-reflection*)
show ?thesis
by (*metis* 3 *Omega-Wagner-1 inteq-reflection*)
qed

lemma *WOmega-Wagner-2-var:*

$\vdash (f \wedge \text{more});(\text{womega } ((g \wedge \text{more});(f \wedge \text{more}))) \longrightarrow (\text{womega } ((f \wedge \text{more});(g \wedge \text{more})))$

proof –

have 1: $\vdash (f \wedge \text{more});((g \wedge \text{more});(f \wedge \text{more}) \wedge \text{more}) \longrightarrow ((f \wedge \text{more});(g \wedge \text{more}) \wedge \text{more});(f \wedge \text{more})$
by (*metis* (*no-types*, *lifting*) *ChopAssoc Prop10 Prop12 RightChopImpMoreRule int-iffD1 inteq-reflection*)

lift-and-com)

show *?thesis* **using** *WOmega-simulation*[*of LIFT* ($f \wedge \text{more}$) *LIFT* ($(g \wedge \text{more}); (f \wedge \text{more})$) *LIFT* ($(f \wedge \text{more}); (g \wedge \text{more})$)]
using 1 **by** *blast*
qed

lemma *Omega-Wagner-2-var*:

$\vdash (f \wedge \text{more}) \frown (\text{omega } ((g \wedge \text{more}) \frown (f \wedge \text{more}))) \longrightarrow (\text{omega } ((f \wedge \text{more}) \frown (g \wedge \text{more})))$
using *WOmega-Wagner-2-var*[*of LIFT* ($(f \wedge \text{more}) \wedge \text{finite}$) *LIFT* ($(g \wedge \text{more}) \wedge \text{finite}$)]
Omega-WOmega[*of LIFT* ($((g \wedge \text{more}) \wedge \text{finite}); ((f \wedge \text{more}) \wedge \text{finite})$)]
Omega-WOmega[*of LIFT* ($((f \wedge \text{more}) \wedge \text{finite}); ((g \wedge \text{more}) \wedge \text{finite})$)]
by (*metis* (*no-types*, *lifting*) *AndMoreSChopAndMoreEqvAndMoreSChop Omega-simulation SChopAssoc int-eq int-iffD1*)

lemma *WOmega-Wagner-2*:

$\vdash (\text{womega } ((f \wedge \text{more}); (g \wedge \text{more}))) = (f \wedge \text{more}) ; (\text{womega } ((g \wedge \text{more}); (f \wedge \text{more})))$
proof –
have 1: $\vdash (f \wedge \text{more}) ; (\text{womega } ((g \wedge \text{more}); (f \wedge \text{more}))) \longrightarrow (\text{womega } ((f \wedge \text{more}); (g \wedge \text{more})))$
by (*simp add: WOmega-Wagner-2-var*)
have 2: $\vdash (\text{womega } ((f \wedge \text{more}); (g \wedge \text{more}))) = ((f \wedge \text{more}); (g \wedge \text{more})); (\text{womega } ((f \wedge \text{more}); (g \wedge \text{more})))$
by (*meson AndChopA LeftChopImpChop Prop11 WOmegaUnroll WOmega-star-1 WPowerstar-ext lift-imp-trans*)
have 3: $\vdash (\text{womega } ((f \wedge \text{more}); (g \wedge \text{more}))) \longrightarrow (f \wedge \text{more}) ; (\text{womega } ((g \wedge \text{more}); (f \wedge \text{more})))$
by (*metis* (*no-types*, *lifting*) 2 *ChopAndB ChopAssoc Prop10 WOmega-Wagner-2-var inteq-reflection*)
show *?thesis*
using 1 3 *int-iffI* **by** *blast*
qed

lemma *Omega-Wagner-2*:

$\vdash (\text{omega } ((f \wedge \text{more}) \frown (g \wedge \text{more}))) = (f \wedge \text{more}) \frown (\text{omega } ((g \wedge \text{more}) \frown (f \wedge \text{more})))$
proof –
have 1: $\vdash (f \wedge \text{more}) \frown (\text{omega } ((g \wedge \text{more}) \frown (f \wedge \text{more}))) \longrightarrow (\text{omega } ((f \wedge \text{more}) \frown (g \wedge \text{more})))$
by (*simp add: Omega-Wagner-2-var*)
have 2: $\vdash (\text{omega } ((f \wedge \text{more}) \frown (g \wedge \text{more}))) = ((f \wedge \text{more}) \frown (g \wedge \text{more})) \frown (\text{omega } ((f \wedge \text{more}) \frown (g \wedge \text{more})))$
by (*metis AndMoreSChopAndMoreEqvAndMoreSChop OmegaUnroll int-eq*)
have 3: $\vdash (\text{omega } ((f \wedge \text{more}) \frown (g \wedge \text{more}))) \longrightarrow (f \wedge \text{more}) \frown (\text{omega } ((g \wedge \text{more}) \frown (f \wedge \text{more})))$
by (*metis* (*no-types*, *lifting*) 2 *Omega-Wagner-2-var RightSChopImpSChop SChopAssoc inteq-reflection*)
show *?thesis*
using 1 3 *int-iffI* **by** *blast*
qed

lemma *WOmega-Wagner-2-var1*:

$\vdash (\text{womega } ((f); (g \wedge \text{more}))) = (f) ; (\text{womega } ((g \wedge \text{more}); (f)))$
proof –
have 1: $\vdash (f) ; (\text{womega } ((g \wedge \text{more}); (f))) \longrightarrow (\text{womega } ((f); (g \wedge \text{more})))$
using *WOmega-simulation*[*of* *f LIFT* ($g \wedge \text{more}$) ; *f LIFT* f ; ($g \wedge \text{more}$)]

by (*metis* (*no-types*, *lifting*) *AndChopB ChopAndB ChopAssoc ChopMoreImpMore MoreChopImpMore Prop10 int-iffD1 inteq-reflection lift-imp-trans*)
have 2: $\vdash (\text{womega } ((f);(g \wedge \text{more}))) = ((f);(g \wedge \text{more})); (\text{womega } ((f);(g \wedge \text{more})))$
by (*metis* *AndChopA Prop11 WOmegaUnroll WOmega-star-1 WPowerstar-inductL-var-equiv inteq-reflection*)
have 25: $\vdash (g \wedge \text{more}); \text{womega } (f; (g \wedge \text{more})) = ((g \wedge \text{more}); f \wedge \text{more}); ((g \wedge \text{more}); \text{womega } (f; (g \wedge \text{more})))$
by (*metis* (*no-types*, *lifting*) 2 *AndChopB ChopAssoc MoreChopImpMore Prop10 int-eq lift-imp-trans*)
have 3: $\vdash (\text{womega } ((f);(g \wedge \text{more}))) \longrightarrow (f) ; (\text{womega } ((g \wedge \text{more});(f)))$
using 2 25 *WOmega-coinduct-eq-var2*[of *LIFT* ($g \wedge \text{more}$) ; ($\text{womega } ((f);(g \wedge \text{more}))$) *LIFT* ($((g \wedge \text{more});(f))$)]
by (*metis* (*no-types*, *opaque-lifting*) *ChopAssoc RightChopImpChop inteq-reflection*)
show ?thesis
using 1 3 *int-iffI* **by** *blast*
qed

lemma *Omega-Wagner-2-var1*:

$\vdash (\text{omega } ((f) \frown (g \wedge \text{more}))) = (f) \frown (\text{omega } ((g \wedge \text{more}) \frown (f)))$
proof –
have 1: $\vdash (f) \frown (\text{omega } ((g \wedge \text{more}) \frown (f))) \longrightarrow (\text{omega } ((f) \frown (g \wedge \text{more})))$
using *Omega-simulation*[of *f LIFT* ($g \wedge \text{more}$) \frown *f LIFT* $f \frown (g \wedge \text{more})$]
by (*metis* (*no-types*, *lifting*) *AndMoreSChopAndMoreEqvAndMoreSChop Prop10 SChopAndB SChopAssoc SChopMoreImpMore int-iffD1 inteq-reflection lift-imp-trans*)
have 2: $\vdash (\text{omega } ((f) \frown (g \wedge \text{more}))) = ((f) \frown (g \wedge \text{more})) \frown (\text{omega } ((f) \frown (g \wedge \text{more})))$
by (*metis* *AndSChopA OmegaUnroll SChopstarMore-induct-lvar-eq SChopstar-inductL-var-equiv int-eq int-iffI*)
have 3: $\vdash (\text{omega } ((f) \frown (g \wedge \text{more}))) \longrightarrow (f) \frown (\text{omega } ((g \wedge \text{more}) \frown (f)))$
using 2 *Omega-coinduct-eq-var2*[of *LIFT* ($g \wedge \text{more}$) \frown ($\text{omega } ((f) \frown (g \wedge \text{more}))$) *LIFT* ($((g \wedge \text{more}) \frown (f))$)]
by (*metis* (*no-types*, *lifting*) *AndMoreSChopAndMoreEqvAndMoreSChop RightSChopImpSChop SChopAssoc inteq-reflection*)
show ?thesis
using 1 3 *int-iffI* **by** *blast*
qed

lemma *WOmega-Wagner-2-var2*:

$\vdash (\text{womega } ((f \wedge \text{more});(g))) = (f \wedge \text{more}) ; (\text{womega } ((g);(f \wedge \text{more})))$
proof –
have 1: $\vdash (f \wedge \text{more}) ; (\text{womega } ((g);(f \wedge \text{more}))) \longrightarrow (\text{womega } ((f \wedge \text{more});(g)))$
by (*metis* (*no-types*, *lifting*) *ChopAssoc WOmega-Wagner-2-var1 WOmega-star-1 WPowerstar-inductL-var-equiv int-eq int-iffD1*)
have 2: $\vdash (\text{womega } ((f \wedge \text{more});(g))) = ((f \wedge \text{more});(g)); (\text{womega } ((f \wedge \text{more});(g)))$
by (*metis* (*no-types*, *lifting*) 1 *AndChopA ChopAssoc WOmegaUnroll WOmega-Wagner-2-var1 int-iffI inteq-reflection*)
have 3: $\vdash (\text{womega } ((f \wedge \text{more});(g))) \longrightarrow (f \wedge \text{more}) ; (\text{womega } ((g);(f \wedge \text{more})))$
using 1 2 *WOmega-Wagner-2-var1*[of - *f*]
by (*metis* 1 2 *ChopAssoc inteq-reflection*)
show ?thesis
using 1 3 *int-iffI* **by** *blast*
qed

lemma *Omega-Wagner-2-var2*:

$\vdash (\text{omega } ((f \wedge \text{more}) \neg (g))) = (f \wedge \text{more}) \neg (\text{omega } ((g) \neg (f \wedge \text{more})))$

proof –

have 1: $\vdash (f \wedge \text{more}) \neg (\text{omega } ((g) \neg (f \wedge \text{more}))) \longrightarrow (\text{omega } ((f \wedge \text{more}) \neg (g)))$

by (*metis* (*no-types*, *lifting*) *AndMoreSChopAndMoreEqvAndMoreSChop Omega-simulation SChopAndASChopAssoc int-eq*)

have 2: $\vdash (\text{omega } ((f \wedge \text{more}) \neg (g))) = ((f \wedge \text{more}) \neg (g)) \neg (\text{omega } ((f \wedge \text{more}) \neg (g)))$

by (*metis* *AndMoreSChopAndMoreEqvAndMoreSChop OmegaUnroll inteq-reflection*)

have 3: $\vdash (\text{omega } ((f \wedge \text{more}) \neg (g))) \longrightarrow (f \wedge \text{more}) \neg (\text{omega } ((g) \neg (f \wedge \text{more})))$

by (*meson* 2 *Omega-Wagner-2-var1 Prop04 RightSChopEqvSChop SChopAssoc int-iffD1 int-iffD2 int-iffI*)

show *?thesis*

using 1 3 *int-iffI* **by** *blast*

qed

lemma *WOmega-Wagner-3*:

assumes $\vdash ((f \wedge \text{more}) ; (\text{womega } (f \vee g)) \vee h) = (\text{womega } (f \vee g))$

shows $\vdash (\text{womega } (f \vee g)) = ((\text{womega } f) \vee (\text{wpowerstar } f); h)$

proof –

have 1: $\vdash (\text{womega } (f \vee g)) \longrightarrow ((\text{womega } f) \vee (\text{wpowerstar } (f \wedge \text{more})); h)$

using *WOmega-coinduct[of LIFT (womega (f ∨ g)) h f] assms*

by *fastforce*

have 11: $\vdash (\text{wpowerstar } (f \wedge \text{more})); h = (\text{wpowerstar } f); h$

by (*simp add: LeftChopEqvChop WPowerstar-more-absorb*)

have 2: $\vdash (\text{wpowerstar } f); h \longrightarrow (\text{womega } (f \vee g))$

by (*metis ChopImpChop Prop03 WOmega-star-1 WPowerstar-subdist assms inteq-reflection*)

have 3: $\vdash ((\text{womega } f) \vee (\text{wpowerstar } f); h) \longrightarrow (\text{womega } (f \vee g))$

by (*meson* 2 *Prop02 WOmega-subdist*)

show *?thesis*

using 1 3 *int-iffI*

by (*metis* 11 *inteq-reflection*)

qed

lemma *Omega-Wagner-3*:

assumes $\vdash ((f \wedge \text{more}) \neg (\text{omega } (f \vee g)) \vee (h)) = (\text{omega } (f \vee g))$

shows $\vdash (\text{omega } (f \vee g)) = ((\text{omega } f) \vee (\text{schopstar } f) \neg (h))$

proof –

have 1: $\vdash (\text{omega } (f \vee g)) \longrightarrow ((\text{omega } f) \vee (\text{schopstar } f) \neg h)$

using *assms Omega-coinduct[of LIFT (omega (f ∨ g)) h f]*

by *fastforce*

have 2: $\vdash (\text{schopstar } f) \neg h \longrightarrow (\text{omega } (f \vee g))$

by (*meson Omega-star-1 Prop03 Prop11 SChopImpSChop SChopstar-subdist assms lift-imp-trans*)

have 3: $\vdash ((\text{omega } f) \vee (\text{schopstar } f) \neg h) \longrightarrow (\text{omega } (f \vee g))$

by (*meson* 2 *Prop02 Omega-subdist*)

show *?thesis* **using** 1 3 **by** *fastforce*

qed

lemma *WOmega-Wagner-1-var*:

$\vdash (\text{womega } ((\text{wpowerstar } f); (f \wedge \text{more}))) = (\text{womega } f)$

by (metis WOmega-More-absorb WOmega-Wagner-1 WOmega-Wagner-2-var1 WOmega-star-1
WpowerstarAndMoreEqvAndMoreChop inteq-reflection)

lemma Omega-Wagner-1-var3:

$\vdash (\text{omega } ((\text{schopstar } f) \frown (f \wedge \text{more}))) = (\text{omega } f)$

by (metis Omega-Wagner-1-var2 Omega-Wagner-2-var1 Omega-star-1 inteq-reflection)

lemma WOmega-star-4:

$\vdash (\text{wpowerstar } (\text{womega } f)) = (\text{empty} \vee (\text{womega } f))$

by (simp add: WOmega-1 WPowerstar-boffa-var)

lemma WOmega-star-5:

$\vdash (\text{womega } f);(\text{wpowerstar } (\text{womega } f)) = (\text{womega } f)$

proof –

have 1: $\vdash (\text{womega } f);(\text{wpowerstar } (\text{womega } f)) \longrightarrow (\text{womega } f)$

by (simp add: WOmega-1)

have 2: $\vdash (\text{womega } f) \longrightarrow (\text{womega } f);(\text{wpowerstar } (\text{womega } f))$

by (simp add: WOmega-sup-id WPowerstar-imp-empty int-iffD2)

show ?thesis

by (simp add: 1 2 int-iffI)

qed

lemma WOmega-or-unfold:

$\vdash ((\text{womega } f) \vee ((\text{wpowerstar } f);(g \wedge \text{more})); (\text{womega } (f \vee g))) = (\text{womega } (f \vee g))$

proof –

have 10: $\vdash (\text{womega } (f \vee g)) = ((f \vee g) \wedge \text{more});(\text{womega } (f \vee g))$

by (simp add: WOmegaUnroll)

have 11: $\vdash ((f \vee g) \wedge \text{more}) = ((f \wedge \text{more}) \vee (g \wedge \text{more}))$

by fastforce

have 1: $\vdash (\text{womega } (f \vee g)) = ((f \wedge \text{more});(\text{womega } (f \vee g)) \vee (g \wedge \text{more});(\text{womega } (f \vee g)))$

using 10 11 by (metis OrChopEqv inteq-reflection)

have 2: $\vdash ((\text{wpowerstar } f);(g \wedge \text{more})); (\text{womega } (f \vee g)) = (\text{wpowerstar } f);((g \wedge \text{more}); (\text{womega } (f \vee g)))$

by (meson ChopAssoc int-iffD1 int-iffD2 int-iffI)

show ?thesis using 1 2 WOmega-Wagner-3[of f g LIFT (g \wedge more);(womega (f \vee g))]

by (metis int-eq)

qed

lemma Omega-or-unfold:

$\vdash ((\text{omega } f) \vee ((\text{schopstar } f) \frown (g \wedge \text{more})) \frown (\text{omega } (f \vee g))) = (\text{omega } (f \vee g))$

proof –

have 10: $\vdash (\text{omega } (f \vee g)) = ((f \vee g) \wedge \text{more}) \frown (\text{omega } (f \vee g))$

by (simp add: OmegaUnroll)

have 11: $\vdash ((f \vee g) \wedge \text{more}) = ((f \wedge \text{more}) \vee (g \wedge \text{more}))$

by fastforce

have 1: $\vdash (\text{omega } (f \vee g)) = ((f \wedge \text{more}) \frown (\text{omega } (f \vee g)) \vee (g \wedge \text{more}) \frown (\text{omega } (f \vee g)))$

using 10 11 by (metis OrSChopEqv inteq-reflection)

have 2: $\vdash ((\text{schopstar } f) \frown (g \wedge \text{more})) \frown (\text{omega } (f \vee g)) = (\text{schopstar } f) \frown ((g \wedge \text{more}) \frown (\text{omega } (f \vee g)))$

by (meson SChopAssoc int-iffD1 int-iffD2 int-iffI)

show *?thesis* **using** 1 2 *Omega-Wagner-3*[*of f g LIFT (g ∧ more)⊃(omega (f ∨ g))*]]
by (*metis int-eq*)
qed

lemma *WOmega-or-unfold-coinduct*:

$\vdash (\text{womega } (f \vee g)) \longrightarrow (\text{womega } (((\text{wpowerstar } f); (g \wedge \text{more})) \)) \vee (\text{wpowerstar } ((\text{wpowerstar } f); (g \wedge \text{more}) \)) ; (\text{womega } f))$

using *WOmega-or-unfold*[*of f g*]
WOmega-coinduct-eq[*of LIFT (womega (f ∨ g)) LIFT (womega f)*
LIFT (wpowerstar f);(g ∧ more)]
by (*metis Prop10 Prop11 Prop12 RightChopImpMoreRule inteq-reflection lift-and-com*)

lemma *Omega-or-unfold-coinduct*:

$\vdash (\text{omega } (f \vee g)) \longrightarrow (\text{omega } (((\text{schopstar } f) \frown (g \wedge \text{more})) \)) \vee (\text{schopstar } ((\text{schopstar } f) \frown (g \wedge \text{more}) \)) \frown (\text{omega } f))$

using *Omega-or-unfold*[*of f g*]
Omega-coinduct-eq[*of LIFT (omega (f ∨ g)) LIFT (omega f)*
LIFT (schopstar f)⊃(g ∧ more)]
by (*metis Prop10 Prop12 RightSChopImpMoreRule int-eq int-iffD2 lift-and-com*)

lemma *WOmega-or-unfold-induct*:

$\vdash (\text{wpowerstar } ((\text{wpowerstar } f); (g \wedge \text{more}))) ; (\text{womega } f) \longrightarrow (\text{womega } (f \vee g))$

using *WOmega-or-unfold*[*of f g*]
using *WPowerstar-induct-leq* **by** *blast*

lemma *Omega-or-unfold-induct*:

$\vdash (\text{schopstar } ((\text{schopstar } f) \frown (g \wedge \text{more}))) \frown (\text{omega } f) \longrightarrow (\text{omega } (f \vee g))$

using *Omega-or-unfold*[*of f g*]
by (*metis AndChopA SChopstar-WPowerstar WPowerstar-induct-leq inteq-reflection lift-imp-trans schop-d-def*)

lemma *WOmega-Wagner-1-gen*:

$\vdash (\text{womega } ((f \wedge \text{more}); (\text{wpowerstar } g) \)) \longrightarrow (\text{womega } (f \vee g))$

proof –

have 01: $\vdash f \longrightarrow (f \vee g)$

by *auto*

have 02: $\vdash (\text{wpowerstar } g) \longrightarrow (\text{wpowerstar } (f \vee g))$

by (*metis WPowerstar-subdist WPowerstar-swap inteq-reflection*)

have 03: $\vdash ((f \wedge \text{more}); (\text{wpowerstar } g) \)) \longrightarrow ((f \vee g) \wedge \text{more}); (\text{wpowerstar } (f \vee g))$

using 01 02 *ChopImpChop*

by (*metis Prop10 Prop12 int-iffD1 inteq-reflection lift-and-com*)

have 1: $\vdash (\text{womega } ((f \wedge \text{more}); (\text{wpowerstar } g) \)) \longrightarrow (\text{womega } (((f \vee g) \wedge \text{more}); (\text{wpowerstar } (f \vee g))))$

by (*simp add: 03 WOmega-iso*)

show *?thesis* **using** *WOmega-Wagner-1*

by (*metis 03 Prop12 WOmega-iso WpowerstarAndMoreEqvAndMoreChop inteq-reflection*)

qed

lemma *Omega-Wagner-1-gen*:

$\vdash (\text{omega } ((f \wedge \text{more}) \frown (\text{schopstar } g) \)) \longrightarrow (\text{omega } (f \vee g))$

proof –

have 1: $\vdash ((f \wedge \text{more}) \wedge \text{finite} \vee (g \wedge \text{more}) \wedge \text{finite}) = (((f \vee g) \wedge \text{more}) \wedge \text{finite})$

by *fastforce*

show *?thesis*

using *Omega-WOmega[of LIFT ((f \wedge more) \frown (schopstar g))] Omega-WOmega[of LIFT (f \vee g)]*

WOmega-Wagner-1-gen[of LIFT ((f \wedge more) \wedge finite) LIFT ((g \wedge more) \wedge finite)]

SChopstar-WPowerstar[of LIFT (g \wedge more)] SChopstar-WPowerstar-more[of g] 1

by (*metis* (*no-types*, *lifting*) *Omega-Wagner-1 Prop10 Prop12 SChopImpFinite SChopstar-WPowerstar-more SChopstar-finite int-iffD1 inteq-reflection itl-def(9) lift-and-com omega-d-def*)

qed

lemma *WOmega-swap*:

$\vdash (\text{womega } (f \vee g)) = (\text{womega } (g \vee f))$

proof –

have 1: $\vdash (f \vee g) = (g \vee f)$

by *fastforce*

show *?thesis*

by (*metis* 1 *WOmega-star-5 int-eq*)

qed

lemma *Omega-swap*:

$\vdash (\text{omega } (f \vee g)) = (\text{omega } (g \vee f))$

proof –

have 1: $\vdash (f \vee g) = (g \vee f)$

by *fastforce*

show *?thesis*

by (*metis* 1 *Omega-star-1 inteq-reflection*)

qed

lemma *WOmega-Wagner-1-var-gen*:

$\vdash (\text{womega } ((\text{wpowerstar } f);(g \wedge \text{more}))) \longrightarrow (\text{womega } (f \vee g))$

proof –

have 1: $\vdash (\text{womega } ((\text{wpowerstar } f);(g \wedge \text{more}))) =$

$(\text{wpowerstar } f);(\text{womega } ((g \wedge \text{more});(\text{wpowerstar } f)))$

by (*simp add: WOmega-Wagner-2-var1*)

have 2: $\vdash (\text{womega } ((g \wedge \text{more});(\text{wpowerstar } f))) \longrightarrow (\text{womega } (g \vee f))$

using *WOmega-Wagner-1-gen[of g f]* **by** *blast*

have 5: $\vdash (\text{wpowerstar } f);(\text{womega } ((g \wedge \text{more});(\text{wpowerstar } f)))$

$\longrightarrow (\text{wpowerstar } f);(\text{womega } (f \vee g))$

by (*metis* 2 *RightChopImpChop WOmega-swap int-eq*)

have 6: $\vdash (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } (f \vee g))$

by (*simp add: WPowerstar-subdist*)

have 7: $\vdash (\text{wpowerstar } f);(\text{womega } (f \vee g)) \longrightarrow (\text{wpowerstar } (f \vee g));(\text{womega } (f \vee g))$

by (*simp add: 6 LeftChopImpChop*)

have 8: $\vdash (\text{wpowerstar } (f \vee g));(\text{womega } (f \vee g)) \longrightarrow (\text{womega } (f \vee g))$

by (*simp add: WOmega-star-1 int-iffD1*)

show *?thesis*

using 1 5 7 8

by (*metis int-eq lift-imp-trans*)

qed

lemma *Omega-Wagner-1-var-gen*:

$\vdash (\text{omega } ((\text{schopstar } f) \neg (g \wedge \text{more}))) \longrightarrow (\text{omega } (f \vee g))$

proof –

have 1: $\vdash (\text{omega } ((\text{schopstar } f) \neg (g \wedge \text{more}))) =$
 $(\text{schopstar } f) \neg (\text{omega } ((g \wedge \text{more}) \neg (\text{schopstar } f)))$

by (*simp add: Omega-Wagner-2-var1*)

have 2: $\vdash (\text{omega } ((g \wedge \text{more}) \neg (\text{schopstar } f))) \longrightarrow (\text{omega } (g \vee f))$

using *Omega-Wagner-1-gen*[of *g f*] **by** *blast*

have 3: $\vdash (g \vee f) = (f \vee g)$

by *fastforce*

have 4: $\vdash (\text{omega } (g \vee f)) = (\text{omega } (f \vee g))$

by (*simp add: Omega-swap*)

have 5: $\vdash (\text{schopstar } f) \neg (\text{omega } ((g \wedge \text{more}) \neg (\text{schopstar } f)))$
 $\longrightarrow (\text{schopstar } f) \neg (\text{omega } (f \vee g))$

by (*metis 2 3 RightSChopImpSChop inteq-reflection*)

have 6: $\vdash (\text{schopstar } f) \longrightarrow (\text{schopstar } (f \vee g))$

by (*metis 3 MP Prop03 SBaGen SBaSCSImpSCS int-eq*)

have 7: $\vdash (\text{schopstar } f) \neg (\text{omega } (f \vee g)) \longrightarrow (\text{schopstar } (f \vee g)) \neg (\text{omega } (f \vee g))$

by (*simp add: 6 LeftSChopImpSChop*)

have 8: $\vdash (\text{schopstar } (f \vee g)) \neg (\text{omega } (f \vee g)) \longrightarrow (\text{omega } (f \vee g))$

by (*simp add: Omega-star-1 int-iffD1*)

show *?thesis*

using 1 5 7 8 **by** *fastforce*

qed

lemma *WOmega-Denest*:

$\vdash (\text{womega } (f \vee g)) =$

$((\text{womega } ((\text{wpowerstar } f); (g \wedge \text{more}))) \vee$
 $(\text{wpowerstar } ((\text{wpowerstar } f); (g \wedge \text{more}))); (\text{womega } f))$

proof –

have 1: $\vdash (\text{womega } (f \vee g)) \longrightarrow (\text{womega } (((\text{wpowerstar } f); (g \wedge \text{more})) \vee$
 $(\text{wpowerstar } ((\text{wpowerstar } f); (g \wedge \text{more}))); (\text{womega } f))$

by (*simp add: WOmega-or-unfold-coinduct*)

have 2: $\vdash (\text{womega } ((\text{wpowerstar } f); (g \wedge \text{more}))) \longrightarrow (\text{womega } (f \vee g))$

by (*simp add: WOmega-Wagner-1-var-gen*)

have 3: $\vdash (\text{wpowerstar } ((\text{wpowerstar } f); (g \wedge \text{more}))); (\text{womega } f) \longrightarrow (\text{womega } (f \vee g))$

by (*simp add: WOmega-or-unfold-induct*)

show *?thesis*

by (*meson 1 2 3 Prop02 int-iffI*)

qed

lemma *Omega-Denest*:

$\vdash (\text{omega } (f \vee g)) =$

$((\text{omega } ((\text{schopstar } f) \neg (g \wedge \text{more}))) \vee$
 $(\text{schopstar } ((\text{schopstar } f) \neg (g \wedge \text{more}))) \neg (\text{omega } f))$

proof –

have 1: $\vdash (\text{omega } (f \vee g)) \longrightarrow (\text{omega } (((\text{schopstar } f) \neg (g \wedge \text{more})) \vee$
 $(\text{schopstar } ((\text{schopstar } f) \neg (g \wedge \text{more}))) \neg (\text{omega } f))$

by (simp add: Omega-or-unfold-coinduct)
 have 2: $\vdash (\text{omega } ((\text{schopstar } f) \frown (g \wedge \text{more}))) \longrightarrow (\text{omega } (f \vee g))$
 by (simp add: Omega-Wagner-1-var-gen)
 have 3: $\vdash (\text{schopstar } ((\text{schopstar } f) \frown (g \wedge \text{more}))) \frown (\text{omega } f) \longrightarrow (\text{omega } (f \vee g))$
 by (simp add: Omega-or-unfold-induct)
 show ?thesis
 by (meson 1 2 3 Prop02 int-iffI)
 qed

lemma *WOmega-or-refine:*

assumes $\vdash (g \wedge \text{more}); (f \wedge \text{more}) \longrightarrow (f \wedge \text{more}) ; (\text{wpowerstar } (f \vee g))$

shows $\vdash (\text{womega } (f \vee g)) = ((\text{womega } f) \vee (\text{wpowerstar } f); (\text{womega } g))$

proof –

have 1: $\vdash (\text{wpowerstar } g); (f \wedge \text{more}) \longrightarrow (f \wedge \text{more}); (\text{wpowerstar } (f \vee g))$
 using assms WPowerstar-invol WPowerstar-sim1[of LIFT (g \wedge more) LIFT (f \wedge more) LIFT powerstar (f \vee g)]

by (metis (no-types, opaque-lifting) ChopstarEqPowerstar Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection)
 have 2: $\vdash (\text{womega } (f \vee g)) = ((\text{womega } g) \vee ((\text{wpowerstar } g); (f \wedge \text{more})); (\text{womega } (f \vee g)))$
 by (metis WOmega-swap WOmega-or-unfold inteq-reflection)
 have 3: $\vdash ((\text{womega } g) \vee ((\text{wpowerstar } g); (f \wedge \text{more})); (\text{womega } (f \vee g))) \longrightarrow (\text{womega } g) \vee ((f \wedge \text{more}); (\text{wpowerstar } (f \vee g))); (\text{womega } (f \vee g))$
 by (metis 1 2 AndChopB Prop08 Prop10 int-iffD2 inteq-reflection)
 have 40: $\vdash (\text{womega } g) \longrightarrow (f \wedge \text{more}); (\text{womega } (f \vee g)) \vee (\text{womega } g)$
 by (simp add: Prop05)
 have 41: $\vdash ((f \wedge \text{more}); (\text{wpowerstar } (f \vee g))); (\text{womega } (f \vee g)) \longrightarrow (f \wedge \text{more}); (\text{womega } (f \vee g)) \vee (\text{womega } g)$
 by (metis (mono-tags, lifting) ChopAssoc WOmega-star-1 intI inteq-reflection unl-lift2)
 have 4: $\vdash (\text{womega } g) \vee ((f \wedge \text{more}); (\text{wpowerstar } (f \vee g))); (\text{womega } (f \vee g)) \longrightarrow (f \wedge \text{more}); (\text{womega } (f \vee g)) \vee (\text{womega } g)$
 using 40 41 Prop02 by blast
 have 5: $\vdash (\text{womega } (f \vee g)) \longrightarrow ((\text{womega } f) \vee (\text{wpowerstar } f); (\text{womega } g))$
 by (metis 2 3 ChopAssoc WOmega-coinduct-var WOmega-star-1 inteq-reflection)
 have 6: $\vdash ((\text{womega } f) \vee (\text{wpowerstar } f); (\text{womega } g)) \longrightarrow (\text{womega } (f \vee g)) \vee (\text{wpowerstar } (f \vee g)); (\text{womega } (f \vee g))$
 by (metis ChopImpChop Prop02 Prop05 WOmega-star-1 WOmega-subdist WOmega-swap WPowerstar-subdist int-eq)
 have 7: $\vdash ((\text{womega } f) \vee (\text{wpowerstar } f); (\text{womega } g)) \longrightarrow (\text{womega } (f \vee g))$
 using 6 WOmega-star-1
 by (metis Prop02 int-iffD1 inteq-reflection lift-imp-trans)
 show ?thesis
 using 5 7 int-iffI by blast
 qed

lemma *Omega-or-refine:*

assumes $\vdash (g \wedge \text{more}) \frown ((f \wedge \text{more}) \wedge \text{finite}) \longrightarrow (f \wedge \text{more}) \frown (\text{schopstar } (f \vee g))$

shows $\vdash (\text{omega } (f \vee g)) = ((\text{omega } f) \vee (\text{schopstar } f) \frown (\text{omega } g))$

proof –

have 1: $\vdash (\text{schopstar } (g)) \frown ((f \wedge \text{more}) \wedge \text{finite}) \longrightarrow (f \wedge \text{more}) \frown (\text{schopstar } (f \vee g))$
using *assms*
using *SChopstar-QuasicommMore-var* **by** *blast*
have 2: $\vdash (\text{omega } (f \vee g)) = ((\text{omega } g) \vee ((\text{schopstar } g) \frown (f \wedge \text{more})) \frown (\text{omega } (f \vee g)))$
by (*metis Omega-swap Omega-or-unfold inteq-reflection*)
have 20: $\vdash ((\text{schopstar } g) \frown (f \wedge \text{more})) \frown (\text{omega } (f \vee g)) =$
 $((\text{schopstar } g) \frown ((f \wedge \text{more}) \wedge \text{finite})) \frown (\text{omega } (f \vee g))$
by (*metis (no-types, lifting) AndMoreAndFiniteEqvAndFmore Prop10 Prop12 SChopAssoc*
int-eq int-iffD2 itl-def(9))
have 25: $\vdash ((\text{schopstar } g) \frown (f \wedge \text{more})) \frown (\text{omega } (f \vee g)) \longrightarrow$
 $((f \wedge \text{more}) \frown (\text{schopstar } (f \vee g))) \frown (\text{omega } (f \vee g))$
using 1 20 **by** (*metis LeftSChopImpSChop inteq-reflection*)
have 3: $\vdash ((\text{omega } g) \vee ((\text{schopstar } g) \frown (f \wedge \text{more})) \frown (\text{omega } (f \vee g))) \longrightarrow$
 $(\text{omega } g) \vee ((f \wedge \text{more}) \frown (\text{schopstar } (f \vee g))) \frown (\text{omega } (f \vee g))$
using 25 **by** *auto*
have 4: $\vdash (\text{omega } g) \vee ((f \wedge \text{more}) \frown (\text{schopstar } (f \vee g))) \frown (\text{omega } (f \vee g)) \longrightarrow$
 $(\text{omega } g) \vee (f \wedge \text{more}) \frown (\text{omega } (f \vee g))$
by (*metis (mono-tags, lifting) Omega-star-1 SChopAssoc intI int-eq intensional-rews(3)*)
have 5: $\vdash (\text{omega } (f \vee g)) \longrightarrow ((\text{omega } f) \vee (\text{schopstar } f) \frown (\text{omega } g))$
by (*metis 2 3 Omega-coinduct Omega-star-1 SChopAssoc inteq-reflection*)
have 50: $\vdash (\text{omega } f) \longrightarrow (\text{omega } (f \vee g))$
by (*simp add: Omega-subdist*)
have 51: $\vdash (\text{omega } g) \longrightarrow (\text{omega } (f \vee g))$
by (*metis Omega-swap Omega-subdist inteq-reflection*)
have 52: $\vdash (\text{schopstar } f) \longrightarrow (\text{schopstar } (f \vee g))$
by (*simp add: SChopstar-subdist*)
have 6: $\vdash ((\text{omega } f) \vee (\text{schopstar } f) \frown (\text{omega } g)) \longrightarrow$
 $(\text{omega } (f \vee g)) \vee (\text{schopstar } (f \vee g)) \frown (\text{omega } (f \vee g))$
by (*metis 50 51 52 Omega-star-1 Prop02 Prop05 SChopImpSChop inteq-reflection*)
have 7: $\vdash ((\text{omega } f) \vee (\text{schopstar } f) \frown (\text{omega } g)) \longrightarrow (\text{omega } (f \vee g))$
using 6 *Omega-star-1* **by** *fastforce*
show *?thesis*
using 5 7 *int-iffI* **by** *blast*
qed

lemma *WOmega-bachmair-dershowitz*:

assumes $\vdash (g \wedge \text{more}); (f \wedge \text{more}) \longrightarrow (f \wedge \text{more}); (\text{wpowerstar } (f \vee g))$

shows $\vdash (\text{womega } (f \vee g)) = \#False = \vdash ((\text{womega } f) \vee (\text{womega } g)) = \#False$

proof –

have 1: $\vdash (\text{womega } (f \vee g)) = \#False \implies \vdash ((\text{womega } f) \vee (\text{womega } g)) = \#False$
using *assms*
by (*metis Prop10 WOmega-subdist-var int-eq int-simps(18)*)
have 2: $\vdash ((\text{womega } f) \vee (\text{womega } g)) = \#False \implies \vdash (\text{womega } (f \vee g)) = \#False$
using *assms WOmega-or-refine[of g f]*
by (*metis Prop03 Prop10 WOmega-star-1 int-simps(18) int-simps(25) int-simps(26) inteq-reflection*)
show *?thesis*
using 1 2 **by** *blast*
qed

lemma *Omega-bachmair-dershowitz:*

assumes $\vdash (g \wedge \text{more}) \neg ((f \wedge \text{more}) \wedge \text{finite}) \longrightarrow (f \wedge \text{more}) \neg (\text{schopstar } (f \vee g))$

shows $\vdash (\text{omega } (f \vee g)) = \#False = \vdash ((\text{omega } f) \vee (\text{omega } g)) = \#False$

proof –

have 1: $\vdash (\text{omega } (f \vee g)) = \#False \implies \vdash ((\text{omega } f) \vee (\text{omega } g)) = \#False$

using *assms*

by (*metis Prop10 Omega-subdist-var int-eq int-simps(18)*)

have 2: $\vdash ((\text{omega } f) \vee (\text{omega } g)) = \#False \implies \vdash (\text{omega } (f \vee g)) = \#False$

using *assms Omega-or-refine[of g f]*

by (*metis Prop03 SChopRightFalse int-simps(14) int-simps(16) int-simps(21) int-simps(9) inteq-reflection*)

show *?thesis*

using 1 2 **by** *blast*

qed

lemma *Omega-and-more-imp-more:*

$\vdash (\text{omega } (f \wedge \text{more})) \longrightarrow \text{more}$

by (*metis AndSChopB MoreSChopImpMore OmegaUnroll inteq-reflection lift-imp-trans*)

lemma *WOmega-and-more-imp-more:*

$\vdash (\text{womega } (f \wedge \text{more})) \longrightarrow \text{more}$

by (*metis ChopImpDi DiAndB DiMoreEqvMore WOmegaUnroll int-eq lift-imp-trans*)

lemma *Omega-and-fmore-imp-more:*

$\vdash (\text{omega } ((f \wedge \text{more}) \wedge \text{finite})) \longrightarrow \text{more}$

by (*metis Omega-and-more-imp-more Omega-subdist OrFiniteInf inteq-reflection lift-imp-trans*)

lemma *WOmega-and-fmore-imp-more:*

$\vdash (\text{womega } ((f \wedge \text{more}) \wedge \text{finite})) \longrightarrow \text{more}$

by (*metis LeftChopImpMoreRule Prop12 WOmegaUnroll int-iffD1 inteq-reflection lift-and-com*)

lemma *womega-len:*

$\vdash \text{womega skip} = (\text{len } k); \text{womega skip}$

proof (*induct k*)

case 0

then show *?case*

by (*metis EmptyChop inteq-reflection wpow-0 wpower-len*)

next

case (*Suc k*)

then show *?case*

proof –

have 1: $\vdash \text{len } \text{Suc } k = \text{len } k ; \text{skip}$

by (*metis LenNPlusOneB Suc-eq-plus1*)

have 2: $\vdash \text{len } \text{Suc } k ; \text{womega skip} = \text{len } k ; (\text{skip}; \text{womega skip})$

by (*metis 1 ChopAssoc inteq-reflection*)

have 3: $\vdash \text{skip}; \text{womega skip} = \text{womega skip}$

by (*metis AndChopA Prop11 WOmegaUnroll WPowerstar-inductL-var-equiv WPowerstar-more-absorb*)

```

inteq-reflection)
  show ?thesis
  by (metis 2 3 Suc inteq-reflection)
qed
qed

```

lemma *omega-len*:

```

  ⊢ omega skip = (len k) ⋈ omega skip
proof (induct k)
case 0
then show ?case
by (metis EmptyChop FiniteAndEmptyEqvEmpty inteq-reflection lift-and-com schop-d-def wpow-0 wpower-len)
next
case (Suc k)
then show ?case
proof –
  have 1: ⊢ len Suc k = len k ⋈ skip
    by (simp add: len-k-schop)
  have 2: ⊢ len Suc k ⋈ omega skip = len k ⋈ (skip ⋈ omega skip)
    by (metis 1 SChopAssoc inteq-reflection)
  have 3: ⊢ skip ⋈ omega skip = omega skip
    by (metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite FmoreEqvSkipChopFinite
      OmegaUnroll Omega-Wagner-1 Omega-star-1 SChopAssoc inteq-reflection schopstar-skip-finite schopstar-skip-fmore)
  show ?thesis
  by (metis 2 3 Suc inteq-reflection)
qed
qed

```

lemma *womega-exist-len*:

```

  ⊢ (womega skip) = (∃ k. len k; (womega skip))
proof –
  have 1: ⊢ (womega skip) = (wpowerstar skip);(womega skip)
    by (meson Prop11 WOmega-star-1)
  have 2: ⊢ (wpowerstar skip) = (∃ k. len k)
    unfolding wpowerstar-d-def
    by (simp add: ExEqvRule wpower-len)
  have 3: ⊢ (∃ k. len k);(womega skip) = (∃ k. len k; (womega skip))
    by (metis ExistChop int-eq)
  show ?thesis using 1 2 3
  by (metis int-eq)
qed

```

lemma *omega-exist-len*:

```

  ⊢ (omega skip) = (∃ k. len k ⋈ (omega skip))
proof –
  have 1: ⊢ (omega skip) = (s chopstar skip) ⋈ (omega skip)
    by (meson Omega-star-1 Prop11)
  have 2: ⊢ (s chopstar skip) = (∃ k. len k)

```

```

  by (metis ExEqvRule int-eq schopstar-skip-finite wpower-len wpowerstar-d-def wpowerstar-skip-finite)
have 3:  $\vdash (\exists k. \text{len } k) \frown (\text{omega skip}) = (\exists k. \text{len } k \frown (\text{omega skip}))$ 
  by (metis ExistSChop inteq-reflection)
show ?thesis using 1 2 3
  by (metis int-eq)
qed

```

```

lemma not-len-and-womega:
 $\vdash \neg (\text{len } k \wedge (\text{womega skip}))$ 
using womega-len[of Suc k ]
using len-len-suc-not by fastforce

```

```

lemma not-len-and-omega:
 $\vdash \neg (\text{len } k \wedge (\text{omega skip}))$ 
using omega-len[of Suc k ]
len-len-suc-not-schop by fastforce

```

```

lemma not-len-and-womega-var:
 $\vdash (\text{womega skip}) \longrightarrow \neg(\text{len } k)$ 
using not-len-and-womega by fastforce

```

```

lemma not-len-and-omega-var:
 $\vdash (\text{omega skip}) \longrightarrow \neg(\text{len } k)$ 
using not-len-and-omega by fastforce

```

```

lemma womega-skip-inf:
 $\vdash (\text{womega skip}) \longrightarrow \text{inf}$ 
proof -
  have 1:  $\vdash \text{finite} = (\exists k. \text{len } k)$ 
    by (simp add: intI itl-defs len-defs nfinite-conv-nlength-enat)
  have 2:  $\vdash \text{inf} = (\forall k. \neg \text{len } k)$ 
    using 1 unfolding finite-d-def by fastforce
  have 3:  $\vdash (\text{womega skip}) \longrightarrow (\forall k. \neg \text{len } k)$ 
    using not-len-and-womega-var by fastforce
  show ?thesis using 2 3
    by (metis int-eq)
qed

```

```

lemma omega-skip-inf:
 $\vdash (\text{omega skip}) \longrightarrow \text{inf}$ 
proof -
  have 1:  $\vdash \text{finite} = (\exists k. \text{len } k)$ 
    by (simp add: intI itl-defs len-defs nfinite-conv-nlength-enat)
  have 2:  $\vdash \text{inf} = (\forall k. \neg \text{len } k)$ 
    using 1 unfolding finite-d-def by fastforce
  have 3:  $\vdash (\text{omega skip}) \longrightarrow (\forall k. \neg \text{len } k)$ 
    using not-len-and-omega-var by fastforce
  show ?thesis using 2 3

```

by *fastforce*
qed

lemma *womega-fmore-inf*:

$\vdash (womega\ fmore) \longrightarrow inf$

using *wpowerstar-skip-fmore*

by (*metis SkipChopFiniteImpFinite WOmega-Wagner-1 WOmega-iso inteq-reflection lift-imp-trans womega-skip-inf wpowerstar-skip-finite*)

lemma *omega-fmore-inf*:

$\vdash (omega\ fmore) \longrightarrow inf$

by (*metis AndChopA ChopEmpty WOmega-iso inteq-reflection lift-imp-trans omega-d-def womega-fmore-inf*)

lemma *womega-and-fmore-inf*:

$\vdash (womega\ (f \wedge fmore)) \longrightarrow inf$

by (*meson Prop12 WOmega-iso int-iffD1 lift-and-com lift-imp-trans womega-fmore-inf*)

lemma *omega-and-fmore-inf*:

$\vdash (omega\ (f \wedge fmore)) \longrightarrow inf$

by (*meson Prop12 Omega-iso int-iffD1 lift-and-com lift-imp-trans omega-fmore-inf*)

lemma *womega-and-empty*:

$\vdash \neg(womega\ (f \wedge empty))$

by (*metis WOmega-Wagner-1 WOmega-empty-zero WPowerstar-and-empty inteq-reflection*)

lemma *omega-and-empty*:

$\vdash \neg(omega\ (f \wedge empty))$

by (*meson MP Omega-empty-zero Omega-iso Prop11 Prop12 lift-and-com lift-imp-neg*)

lemma *womega-split-empty-more*:

$\vdash (womega\ f) = (womega\ (f \wedge more))$

by (*metis WOmega-Wagner-1 WPowerstar-more-absorb inteq-reflection*)

lemma *omega-split-empty-more*:

$\vdash (omega\ f) = (omega\ (f \wedge more))$

by (*metis Omega-Wagner-1 SChopstar-and-more inteq-reflection*)

lemma *omega-split-empty-more-var*:

$\vdash (omega\ (f \wedge more)) = (omega\ (f \wedge fmore))$

by (*metis AndMoreSChopEqvAndFmoreChop ChopEmpty EmptySChop Omega-Wagner-2-var1 inteq-reflection*)

lemma *omega-imp-inf*:

$\vdash (omega\ f) \longrightarrow inf$

by (*metis int-eq omega-and-fmore-inf omega-split-empty-more omega-split-empty-more-var*)

lemma *inf-imp-omega-skip*:
 $\vdash \text{inf} \longrightarrow (\text{omega skip})$
proof –
have 1: $\vdash ((\text{skip} \wedge \text{more}) \wedge \text{finite}) = \text{skip}$
by (*metis DiIntro DiSkipEqvMore Prop10 WPowerstar-ext WPowerstar-skip-finite inteq-reflection*)
have 2: $\vdash \text{inf} \longrightarrow \text{skip};\text{inf}$
by (*metis AndInfEqvChopFalse ChopAndInf MoreAndInfEqvInf MoreEqvSkipChopTrue infinite-d-def int-eq int-iffD1*)
have 3: $\vdash \text{inf} \longrightarrow ((\text{skip} \wedge \text{more}) \wedge \text{finite});\text{inf}$
using 1 2 **by** (*metis int-eq*)
show ?thesis
by (*simp add: 3 OmegaWeakCoinduct schop-d-def*)
qed

lemma *Omega-schopstar-max-element*:
 $\vdash f \longrightarrow (\text{omega skip}) \vee (\text{schopstar skip})$
proof –
have 1: $\vdash (\text{schopstar skip}) = \text{finite}$
by (*meson Prop11 schopstar-skip-finite*)
have 2: $\vdash (\text{omega skip}) = \text{inf}$
by (*simp add: inf-imp-omega-skip int-iffI omega-skip-inf*)
show ?thesis **using** 1 2 **unfolding** *finite-d-def* **by** *fastforce*
qed

lemma *Omega-fmore-absorb*:
 $\vdash (\text{omega } f) = (\text{omega } (f \wedge \text{fmore}))$
by (*metis int-eq omega-split-empty-more omega-split-empty-more-var*)

lemma *Omega-top*:
 $\vdash (\text{omega } f);(\text{omega empty}) = (\text{omega } f)$
proof –
have 1: $\vdash (\text{omega } f) \longrightarrow \text{inf}$
by (*simp add: omega-imp-inf*)
have 2: $\vdash (\text{omega } f);(\text{omega empty}) \longrightarrow (\text{omega } f)$
by (*simp add: Omega-1*)
have 3: $\vdash (\text{omega } f) \longrightarrow (\text{omega } f);(\text{omega empty})$
by (*metis 1 AndInfChopEqvAndInf Prop10 int-iffD2 inteq-reflection*)
show ?thesis
by (*simp add: 2 3 int-iffI*)
qed

lemma *womega-and-inf*:
 $\vdash (\text{womega } (f \wedge \text{inf})) = (f \wedge \text{inf})$
by (*metis (no-types, lifting) AndInfChopEqvAndInf AndMoreAndInfEqvAndInf Prop10 Prop12 WOmegaUn-*)

roll int-iffD1 inteq-reflection)

lemma *womega-split-finite-inf*:

$\vdash (\text{womega } f) =$
 $(\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf}) \vee (\text{womega } (f \wedge \text{finite})))$

proof –

have 0: $\vdash (\text{womega } f) = (\text{womega } ((f \wedge \text{finite}) \vee (f \wedge \text{inf})))$

by (*metis OrFiniteInf int-eq*)

have 1: $\vdash (\text{womega } ((f \wedge \text{finite}) \vee (f \wedge \text{inf}))) =$

$(\text{womega } (\text{wpowerstar } (f \wedge \text{finite}); ((f \wedge \text{inf}) \wedge \text{more})) \vee$
 $\text{wpowerstar } (\text{wpowerstar } (f \wedge \text{finite}); ((f \wedge \text{inf}) \wedge \text{more})); \text{womega } (f \wedge \text{finite}))$

using *WOmega-Denest*[of *LIFT* $f \wedge \text{finite}$ *LIFT* $(f \wedge \text{inf})$]

by *blast*

have 15: $\vdash ((f \wedge \text{inf}) \wedge \text{more}) = (f \wedge \text{inf})$

using *MoreAndInfEqvInf* **by** *auto*

have 2: $\vdash (\text{womega } (\text{wpowerstar } (f \wedge \text{finite}); ((f \wedge \text{inf}) \wedge \text{more}))) = \text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf})$

by (*metis 15 AndInfChopEqvAndInf WOmega-Wagner-2-var1 inteq-reflection*)

have 3: $\vdash \text{wpowerstar } (\text{wpowerstar } (f \wedge \text{finite}); ((f \wedge \text{inf}) \wedge \text{more})) =$

$(\text{empty} \vee (\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf})))$

by (*metis 15 2 WOmega-star-4 inteq-reflection*)

have 4: $\vdash (\text{empty} \vee (\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf}))); \text{womega } (f \wedge \text{finite}) =$

$(\text{womega } (f \wedge \text{finite}) \vee (\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf})))$

by (*meson AndInfChopEqvAndInf ChopAssoc EmptyOrChopEqv Prop04 Prop06 RightChopEqvChop*)

have 5: $\vdash (\text{womega } f) =$

$(\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf}) \vee$

$(\text{empty} \vee (\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf}))); \text{womega } (f \wedge \text{finite}))$

by (*metis 0 1 2 3 int-eq*)

have 6: $\vdash (\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf}) \vee$

$(\text{empty} \vee (\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf}))); \text{womega } (f \wedge \text{finite})) =$

$((\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf})) \vee$

$(\text{womega } (f \wedge \text{finite})) \vee (\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf})))$

using 4

by (*metis 5 int-eq*)

have 7: $\vdash ((\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf})) \vee$

$(\text{womega } (f \wedge \text{finite})) \vee (\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf}))) =$

$((\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf})) \vee$

$(\text{womega } (f \wedge \text{finite})))$

by *auto*

show *?thesis* **using** 5 6 7

by (*metis int-eq*)

qed

lemma *womega-imp-inf*:

$\vdash (\text{womega } f) \longrightarrow \text{inf}$

proof –

have 1: $\vdash (\text{womega } f) = (\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf}) \vee (\text{womega } (f \wedge \text{finite})))$

using *womega-split-finite-inf* **by** *blast*

have 2: $\vdash (\text{womega } (f \wedge \text{finite})) \longrightarrow \text{inf}$

by (*metis omega-d-def omega-imp-inf*)

have 3: $\vdash \text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf}) \longrightarrow \text{inf}$

by (*meson AndChopB ChopAndB FiniteChopInfEqvInf Prop11 WPowerstar-inductL-var-equiv lift-imp-trans*)
show *?thesis using 1 2 3 by fastforce*
qed

lemma *WOmega-Omega-skip:*

$\vdash \text{womega skip} = \text{omega skip}$

by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite Omega-Wagner-1 Prop10 SkipChopFiniteImpFinite WOmega-Wagner-1 WPowerstar-skip-finite inteq-reflection omega-d-def omega-split-empty-schopstar-skip-finite womega-split-empty-more*)

lemma *WOmega-Omega:*

$\vdash (\text{womega } f) = ((\text{schopstar } f); (f \wedge \text{inf})) \vee (\text{omega } f)$

by (*metis SChopstar-WPowerstar inteq-reflection omega-d-def womega-split-finite-inf*)

lemma *womega-and-more-inf:*

$\vdash (\text{womega } (f \wedge \text{more})) \longrightarrow \text{inf}$

by (*simp add: womega-imp-inf*)

lemma *womega-fmore-inf-eq:*

$\vdash \text{womega fmore} = \text{inf}$

by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite Prop11 WOmega-Omega-skip*

WOmega-Wagner-1 WPowerstar-more-absorb inf-imp-omega-skip inteq-reflection womega-fmore-inf wpowerstar-skip-finite wpowerstar-skip-fmore)

lemma *womega-skip-inf-eq:*

$\vdash \text{womega skip} = \text{inf}$

by (*metis Prop11 WOmega-Omega-skip inf-imp-omega-skip inteq-reflection womega-skip-inf*)

lemma *womega-more-inf:*

$\vdash \text{womega more} \longrightarrow \text{inf}$

by (*simp add: womega-imp-inf*)

lemma *womega-more-inf-eq:*

$\vdash \text{womega more} = \text{inf}$

by (*metis OrFiniteInf WOmega-subdist fmore-d-def int-eq int-iffI womega-fmore-inf-eq womega-more-inf*)

lemma *womega-true:*

$\vdash \text{womega } \# \text{True} = \text{inf}$

by (*metis AndInfEqvChopFalse MoreAndInfEqvInf Prop11 RightChopImpChop WOmegaWeakCoinduct int-simps(11) int-simps(17) inteq-reflection womega-imp-inf*)

lemma *OmegaChopOmega:*

$\vdash (\text{omega } f);g = (\text{omega } f)$

by (*metis AndInfChopEqvAndInf Prop10 inteq-reflection omega-imp-inf*)

lemma *WOmegaChopWOmega*:

$\vdash (\text{womega } f); g = (\text{womega } f)$

by (*metis ChopSCHopdef Prop04 Prop05 Prop12 WOmega-1 WPowerstar-tc int-iffD2 int-iffI lift-imp-trans womega-imp-inf*)

lemma *WOmegaYieldsEvNotWOmega*:

$\vdash (\text{womega } f) \text{ yields } g = (\neg(\text{womega } f))$

by (*simp add: WOmegaChopWOmega WOmega-1 int-iffD2 int-iffI yields-d-def*)

2.6.3 Properties of Omega

lemma *NotOmegaInf*:

$\vdash \neg(\text{omega } (\text{inf}))$

proof –

have 1: $\vdash \text{omega } (\text{inf}) = ((\text{inf} \wedge \text{more}) \wedge \text{finite}); \text{omega } \text{inf}$

by (*metis OmegaUnroll schop-d-def*)

have 2: $\vdash ((\text{inf} \wedge \text{more}) \wedge \text{finite}) = \#False$

unfolding *finite-d-def* **by** *auto*

have 3: $\vdash \#False; \text{omega } \text{inf} = \#False$

by (*metis AndInfChopEqvAndInf int-eq int-simps(22)*)

from 1 2 3 **show** *?thesis*

by (*metis TrueW int-simps(3) inteq-reflection*)

qed

lemma *InfImpOmegaLenPlusOne*:

$\vdash \text{inf} \longrightarrow \text{omega } (\text{len}(\text{Suc } n))$

proof –

have 1: $\vdash \text{inf} \longrightarrow ((\text{len } \text{Suc } n \wedge \text{more}) \wedge \text{finite}); \text{inf}$

by (*auto simp add: Valid-def itl-defs len-defs nfinite-conv-nlength-enat zero-enat-def*)

show *?thesis*

using *OmegaWeakCoinduct[of LIFT inf LIFT len (Suc n)] 1*

by (*metis schop-d-def*)

qed

lemma *OmegaLenPlusOneEqvInf*:

$\vdash \text{omega } (\text{len}(\text{Suc } n)) = \text{inf}$

by (*simp add: InfImpOmegaLenPlusOne int-iffI omega-imp-inf*)

lemma *OmegaSkipEqvInf*:

$\vdash \text{omega } \text{skip} = \text{inf}$

proof –

have 1: $\vdash \text{skip} = (\text{len } 1)$

by (*simp add: len-d-def wpower-d-def*)

(*metis ChopEmpty inteq-reflection*)

have 2: $\vdash \text{omega } \text{skip} = \text{omega } (\text{len } 1)$

using 1 **by** (*metis OmegaUnroll inteq-reflection*)

from 2 **show** *?thesis* **using** *OmegaLenPlusOneEqvInf* **by** *fastforce*

qed

lemma *InfImpOmegaTrue*:

$\vdash \text{inf} \longrightarrow \text{omega} \# \text{True}$

proof –

have *1*: $\vdash \text{inf} \longrightarrow ((\# \text{True} \wedge \text{more}) \wedge \text{finite}); \text{inf}$

by (*auto simp add: Valid-def itl-defs nfinite-conv-nlength-enat zero-enat-def*)

show *?thesis*

using *OmegaWeakCoinduct[of LIFT inf LIFT #True] 1*

by (*metis schop-d-def*)

qed

lemma *OmegaTrueEqvInf*:

$\vdash \text{omega} \# \text{True} = \text{inf}$

by (*simp add: InfImpOmegaTrue int-iffI omega-imp-inf*)

lemma *InfImpOmegaMore*:

$\vdash \text{inf} \longrightarrow \text{omega} \text{ more}$

using *OmegaWeakCoinduct[of LIFT inf LIFT more]*

proof –

have *1*: $\vdash \text{inf} \longrightarrow ((\text{more} \wedge \text{more}) \wedge \text{finite}); \text{inf}$

by (*auto simp add: Valid-def itl-defs nfinite-conv-nlength-enat zero-enat-def*)

show *?thesis* **using** *OmegaWeakCoinduct[of LIFT inf LIFT more] 1* **by** (*metis schop-d-def*)

qed

lemma *OmegaMoreEqvInf*:

$\vdash \text{omega} \text{ more} = \text{inf}$

by (*simp add: InfImpOmegaMore int-iffI omega-imp-inf*)

lemma *InfImpOmegaFinite*:

$\vdash \text{inf} \longrightarrow \text{omega} \text{ finite}$

proof –

have *1*: $\vdash \text{inf} \longrightarrow ((\text{finite} \wedge \text{more}) \wedge \text{finite}); \text{inf}$

by (*auto simp add: Valid-def itl-defs nfinite-conv-nlength-enat zero-enat-def*)

show *?thesis* **using** *OmegaWeakCoinduct[of LIFT inf LIFT finite] 1* **by** (*metis schop-d-def*)

qed

lemma *OmegaFiniteEqvInf*:

$\vdash \text{omega} \text{ finite} = \text{inf}$

by (*simp add: InfImpOmegaFinite int-iffI omega-imp-inf*)

lemma *BoxStateAndInfImpWOmegaBoxState*:

$\vdash \Box(\text{init } w) \wedge \text{inf} \longrightarrow \text{womega } (\Box(\text{init } w))$

using *WOmegaWeakCoinduct[of LIFT ($\Box(\text{init } w)$) LIFT $\Box(\text{init } w)$]*

by (*metis BoxStateChopBoxAndInfImpBox BoxStateChopBoxEqvBox WOmega-iso inteq-reflection wom-ega-and-inf*)

lemma *BoxStateAndInfImpOmegaBoxState:*

$\vdash \Box(\text{init } w) \wedge \text{inf} \longrightarrow \text{omega } (\Box(\text{init } w))$

proof –

have 1: $\vdash ((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}); (\Box(\text{init } w) \wedge \text{inf}) =$
 $(\Box(\text{init } w) \wedge (\text{more} \wedge \text{finite}); \text{inf})$

using *BoxStateAndChopEqvChop*[of *w LIFT (more \wedge finite) LIFT inf*]

by (*metis AndMoreAndFiniteEqvAndFmore fmore-d-def int-eq*)

have 2: $\vdash (\text{more} \wedge \text{finite}); \text{inf} = \text{inf}$

by (*metis AndInfEqvChopFalse ChopAssoc FiniteChopMoreEqvMore FmoreEqvSkipChopFinite MoreAnd-InfEqvInf MoreEqvSkipChopTrue fmore-d-def infinite-d-def inteq-reflection*)

have 3: $\vdash \Box(\text{init } w) \wedge \text{inf} \longrightarrow ((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}); (\Box(\text{init } w) \wedge \text{inf})$

by (*metis 1 2 Prop11 int-eq*)

show ?thesis **using** *OmegaWeakCoinduct*[of *LIFT ($\Box(\text{init } w) \wedge \text{inf}$) LIFT $\Box(\text{init } w)$] 3*

by (*metis schop-d-def*)

qed

lemma *WOmegaBoxStateImpBoxState:*

$\vdash \text{womega } (\Box(\text{init } w)) \longrightarrow \Box(\text{init } w)$

proof –

have 1: $\vdash \text{womega } (\Box(\text{init } w)) \longrightarrow \text{init } w$

by (*meson BoxElim Prop11 StateChopExportA WOmegaUnroll WOmega-iso lift-imp-trans*)

have 2: $\vdash \text{womega } (\Box(\text{init } w)) \longrightarrow (\Box(\text{init } w) \wedge \text{more}); \text{womega } (\Box(\text{init } w))$

by (*simp add: WOmegaUnroll int-iffD1*)

have 21: $\vdash ((\Box(\text{init } w) \wedge \text{more})) \longrightarrow \bigcirc(\Box(\text{init } w))$

by (*metis BoxImpNowAndWeakNext Prop01 Prop12 WnextEqvEmptyOrNext empty-d-def inteq-reflection*

lift-and-com)

have 22: $\vdash \text{womega } (\Box(\text{init } w)) \longrightarrow \text{more}$

by (*meson Prop11 WOmega-and-more-imp-more lift-imp-trans womega-split-empty-more*)

have 23: $\vdash \bigcirc(\Box(\text{init } w)) \longrightarrow \bigcirc(\text{wpowerstar } (\Box(\text{init } w)))$

by (*simp add: NextImpNext WPowerstar-ext*)

have 24: $\vdash \text{womega } (\Box(\text{init } w)) \longrightarrow (\bigcirc(\text{wpowerstar } (\Box(\text{init } w)))); (\text{womega } (\Box(\text{init } w)))$

by (*metis 21 23 LeftChopImpChop WOmegaUnroll int-eq lift-imp-trans*)

have 10: $\vdash \text{womega } (\Box(\text{init } w)) \longrightarrow \bigcirc(\text{womega } (\Box(\text{init } w)))$

using *WOmega-star-1*[of *LIFT ($\Box(\text{init } w)$)*]

by (*metis 24 NextChop inteq-reflection*)

show ?thesis **using** *BoxIntro*[of *LIFT womega ($\Box(\text{init } w)$) LIFT ($\text{init } w$)*]

by (*simp add: 1 10 Prop01 Prop05 Prop12 womega-imp-inf*)

qed

lemma *OmegaBoxStateImpBoxState:*

$\vdash \text{omega } (\Box(\text{init } w)) \longrightarrow \Box(\text{init } w)$

by (*meson Prop03 WOmegaBoxStateImpBoxState WOmega-Omega lift-imp-trans*)

lemma *OmegaIntro:*

assumes $\vdash h \longrightarrow (f \wedge \text{fmore}); h$

shows $\vdash h \longrightarrow \text{omega } f$

using *assms*

by (*metis AndMoreSChopEqvAndFmoreChop OmegaWeakCoinduct inteq-reflection*)

lemma *WOmegaEqvRule*:

assumes $\vdash f = g$

shows $\vdash \text{womega } f = \text{womega } g$

using *assms using int-eq by force*

lemma *OmegaEqvRule*:

assumes $\vdash f = g$

shows $\vdash \text{omega } f = \text{omega } g$

using *assms using int-eq by force*

lemma *AndWOmegaA*:

$\vdash \text{womega } (f \wedge g) \longrightarrow \text{womega } f$

by (*meson WOmega-iso Prop12 int-iffD2 lift-and-com*)

lemma *AndOmegaA*:

$\vdash \text{omega } (f \wedge g) \longrightarrow \text{omega } f$

by (*meson Omega-iso Prop12 int-iffD2 lift-and-com*)

lemma *AndWOmegaB*:

$\vdash \text{womega } (f \wedge g) \longrightarrow \text{womega } g$

by (*meson WOmega-iso Prop12 int-iffD2 lift-and-com*)

lemma *AndOmegaB*:

$\vdash \text{omega } (f \wedge g) \longrightarrow \text{omega } g$

by (*meson Omega-iso Prop12 int-iffD2 lift-and-com*)

lemma *BaWOmegaImpWOmega*:

$\vdash \text{ba } (f \longrightarrow g) \longrightarrow \text{womega } f \longrightarrow \text{womega } g$

proof –

have 1: $\vdash \text{ba } (f \longrightarrow g) \wedge (f \wedge \text{more}); \text{womega } f \longrightarrow ((f \longrightarrow g) \wedge (f \wedge \text{more})); ((f \longrightarrow g) \wedge \text{womega } f)$

using *BaAndChopImport*[of *LIFT* ($f \longrightarrow g$) *LIFT* ($f \wedge \text{more}$) *LIFT* *womega f*]

by *blast*

have 2: $\vdash (f \longrightarrow g) \wedge (f \wedge \text{more}) \longrightarrow (g \wedge \text{more})$

by *auto*

have 3: $\vdash (f \longrightarrow g) \wedge \text{womega } f \longrightarrow \text{womega } f$

by *auto*

have 4: $\vdash \text{ba } (f \longrightarrow g) \wedge (f \wedge \text{more}); \text{womega } f \longrightarrow (g \wedge \text{more}); \text{womega } f$

using 1 2 3

by (*metis* (*no-types, lifting*) *AndChopB ChopAndB Prop10 int-eq lift-imp-trans*)

have 5: $\vdash \text{ba } (f \longrightarrow g) \wedge (f \wedge \text{more}); \text{womega } f \longrightarrow ((f \longrightarrow g) \wedge (f \wedge \text{more})); (\text{ba}(f \longrightarrow g) \wedge \text{womega } f)$

using *BaAndChopImportB by blast*

have 6: $\vdash ((f \longrightarrow g) \wedge (f \wedge \text{more})); (\text{ba}(f \longrightarrow g) \wedge \text{womega } f) \longrightarrow$

$(g \wedge \text{more}); (\text{ba}(f \longrightarrow g) \wedge \text{womega } f)$

using 2 *LeftChopImpChop by blast*

have 61: $\vdash \text{womega } f = (f \wedge \text{more}); \text{womega } f$

by (*simp add: WOmegaUnroll*)

have 62: $\vdash (\text{ba } (f \longrightarrow g) \wedge \text{womega } f) \longrightarrow (\text{ba } (f \longrightarrow g) \wedge (f \wedge \text{more}); \text{womega } f)$

by (metis 61 ChopEmpty Prop11 int-eq)
 have 7: $\vdash (ba\ (f \longrightarrow g) \wedge womega\ f) \longrightarrow (g \wedge more); (ba\ (f \longrightarrow g) \wedge womega\ f)$
 using 62 5 6
 by (meson lift-imp-trans)
 have 8: $\vdash (ba\ (f \longrightarrow g) \wedge womega\ f) \longrightarrow womega\ g$
 using 7
 using WOmegaWeakCoinduct by blast
 from 8 show ?thesis by fastforce
 qed

lemma BaOmegaImpOmega:

$\vdash ba\ (f \longrightarrow g) \longrightarrow omega\ f \longrightarrow omega\ g$
proof –
 have 1: $\vdash ba\ (f \longrightarrow g) \wedge (f \wedge fmore); omega\ f \longrightarrow ((f \longrightarrow g) \wedge (f \wedge fmore)); ((f \longrightarrow g) \wedge omega\ f)$
 using BaAndChopImport[of LIFT (f \longrightarrow g) LIFT (f \wedge fmore) LIFT omega f]
 by blast
 have 2: $\vdash (f \longrightarrow g) \wedge (f \wedge fmore) \longrightarrow (g \wedge fmore)$
 by auto
 have 3: $\vdash (f \longrightarrow g) \wedge omega\ f \longrightarrow omega\ f$
 by auto
 have 4: $\vdash ba\ (f \longrightarrow g) \wedge (f \wedge fmore); omega\ f \longrightarrow (g \wedge fmore); omega\ f$
 using 1 2 3
 by (metis (no-types, lifting) AndChopB ChopAndB Prop10 int-eq lift-imp-trans)
 have 5: $\vdash ba\ (f \longrightarrow g) \wedge (f \wedge fmore); omega\ f \longrightarrow ((f \longrightarrow g) \wedge (f \wedge fmore)); (ba\ (f \longrightarrow g) \wedge omega\ f)$
 using BaAndChopImportB by blast
 have 6: $\vdash ((f \longrightarrow g) \wedge (f \wedge fmore)); (ba\ (f \longrightarrow g) \wedge omega\ f) \longrightarrow$
 $(g \wedge fmore); (ba\ (f \longrightarrow g) \wedge omega\ f)$
 using 2 LeftChopImpChop by blast
 have 61: $\vdash omega\ f = (f \wedge fmore); omega\ f$
 by (metis AndMoreAndFiniteEqvAndFmore OmegaUnroll inteq-reflection schop-d-def)
 have 62: $\vdash (ba\ (f \longrightarrow g) \wedge omega\ f) \longrightarrow (ba\ (f \longrightarrow g) \wedge (f \wedge fmore); omega\ f)$
 by (metis 61 ChopEmpty Prop11 int-eq)
 have 7: $\vdash (ba\ (f \longrightarrow g) \wedge omega\ f) \longrightarrow (g \wedge fmore); (ba\ (f \longrightarrow g) \wedge omega\ f)$
 using 62 5 6
 by (meson lift-imp-trans)
 have 8: $\vdash (ba\ (f \longrightarrow g) \wedge omega\ f) \longrightarrow omega\ g$
 using 7 OmegaIntro by blast
 from 8 show ?thesis by fastforce
 qed

lemma BaWOmegaEqvWOmega:

$\vdash ba\ (f = g) \longrightarrow (womega\ f = womega\ g)$
proof –
 have 0: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$ by fastforce
 have 1: $\vdash ba\ (f = g) = (ba\ (f \longrightarrow g) \wedge ba\ (g \longrightarrow f))$ by (metis 0 BaAndEqv int-eq)
 have 2: $\vdash ba\ (f \longrightarrow g) \longrightarrow (womega\ f \longrightarrow womega\ g)$ using BaWOmegaImpWOmega by blast
 have 3: $\vdash ba\ (g \longrightarrow f) \longrightarrow (womega\ g \longrightarrow womega\ f)$ using BaWOmegaImpWOmega by blast
 have 4: $\vdash ba\ (f = g) \longrightarrow (womega\ f \longrightarrow womega\ g) \wedge (womega\ g \longrightarrow womega\ f)$ using 1 2 3 by fastforce

have 5: $\vdash ((w\omega f \longrightarrow w\omega g) \wedge (w\omega g \longrightarrow w\omega f)) = (w\omega f = w\omega g)$ **by** *auto*
from 4 5 **show** *?thesis* **by** *auto*
qed

lemma *BaOmegaEqvOmega*:

$\vdash ba (f = g) \longrightarrow (\omega f = \omega g)$

proof –

have 0: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** *fastforce*

have 1: $\vdash ba (f = g) = (ba (f \longrightarrow g) \wedge ba (g \longrightarrow f))$ **by** (*metis* 0 *BaAndEqv int-eq*)

have 2: $\vdash ba (f \longrightarrow g) \longrightarrow (\omega f \longrightarrow \omega g)$ **using** *BaOmegaImpOmega* **by** *blast*

have 3: $\vdash ba (g \longrightarrow f) \longrightarrow (\omega g \longrightarrow \omega f)$ **using** *BaOmegaImpOmega* **by** *blast*

have 4: $\vdash ba (f = g) \longrightarrow (\omega f \longrightarrow \omega g) \wedge (\omega g \longrightarrow \omega f)$ **using** 1 2 3 **by** *fastforce*

have 5: $\vdash ((\omega f \longrightarrow \omega g) \wedge (\omega g \longrightarrow \omega f)) = (\omega f = \omega g)$ **by** *auto*

from 4 5 **show** *?thesis* **by** *auto*

qed

lemma *BaAndWOmegaImport*:

$\vdash ba f \wedge w\omega g \longrightarrow w\omega (f \wedge g)$

proof –

have 1: $\vdash f \longrightarrow (g \longrightarrow (f \wedge g))$ **by** *auto*

hence 2: $\vdash ba f \longrightarrow ba (g \longrightarrow f \wedge g)$ **by** (*rule* *BaImpBa*)

have 3: $\vdash ba (g \longrightarrow f \wedge g) \longrightarrow w\omega g \longrightarrow w\omega (f \wedge g)$ **by** (*rule* *BaWOmegaImpWOmega*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *BaAndOmegaImport*:

$\vdash ba f \wedge \omega g \longrightarrow \omega (f \wedge g)$

proof –

have 1: $\vdash f \longrightarrow (g \longrightarrow (f \wedge g))$ **by** *auto*

hence 2: $\vdash ba f \longrightarrow ba (g \longrightarrow f \wedge g)$ **by** (*rule* *BaImpBa*)

have 3: $\vdash ba (g \longrightarrow f \wedge g) \longrightarrow \omega g \longrightarrow \omega (f \wedge g)$ **by** (*rule* *BaOmegaImpOmega*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *InfAndChop*:

$\vdash (inf \wedge (f \wedge fmore);g) = (f \wedge fmore);(g \wedge inf)$

by (*meson* *ChopAndInf Prop04 lift-and-com*)

lemma *InfImportBox*:

$\vdash (inf \wedge \Box f) = \Box (inf \wedge f)$

by (*metis* *BoxAndBoxEqvBoxRule FiniteChopEqvDiamond FiniteChopFiniteEqvFinite InfEqvNotFinite NotDiamondNotEqvBox OrFiniteInf finite-d-def int-eq*)

lemma *InfImportBoxImp*:

$\vdash (inf \wedge \Box (f \longrightarrow g)) = \Box ((inf \longrightarrow f) \longrightarrow (inf \wedge g))$

proof –

have 1: $\vdash (inf \wedge (f \longrightarrow g)) = ((inf \longrightarrow f) \longrightarrow (inf \wedge g))$

by *auto*

show *?thesis*

by (*metis* 1 *InfImportBox inteq-reflection*)

qed

lemma *OmegaImpAOmega*:

$\vdash \text{omega } f \longrightarrow \text{aomega } f$

using *AOmegaWeakCoInduct*[of *LIFT omega f f*]

by (*metis OmegaUnroll Prop10 Prop11 inteq-reflection lift-and-com omega-imp-inf schop-d-def*)

lemma *AOmegaImpOmega*:

$\vdash \text{aomega } f \longrightarrow \text{omega } f$

using *OmegaWeakCoinduct*[of *LIFT aomega f f*]

by (*simp add: AOmegaUnroll int-iffD1 schop-d-def*)

lemma *OmegaEqvAOmega*:

$\vdash \text{omega } f = \text{aomega } f$

using *OmegaImpAOmega AOmegaImpOmega*

by (*metis int-iffI*)

lemma *WOmega-Finite-Import*:

$\vdash \neg((\text{womega } f) \wedge \text{finite})$

by (*metis MP NotEmptyAndInf Prop05 Prop13 finite-d-def int-eq int-simps(32) womega-imp-inf*)

lemma *WOmega-Inf-Import*:

$\vdash ((\text{womega } f) \wedge \text{inf}) = ((f; (\text{wpowerstar } f) \wedge \text{inf}) \vee ((\text{womega } (f \wedge \text{finite})) \wedge \text{inf}))$

proof –

have 1: $\vdash (\text{womega } f) = (\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf}) \vee (\text{womega } (f \wedge \text{finite})))$

using *womega-split-finite-inf* **by** *blast*

have 2: $\vdash (f; (\text{wpowerstar } f) \wedge \text{inf}) = ((\text{wpowerstar } f) \wedge \text{finite}); (f \wedge \text{inf})$

by (*metis WPowerstar-And-Inf-Eqv-WPowerplus-And-Inf WPowerstar-Inf-Import WPowerstar-slide-var inteq-reflection*)

have 4: $\vdash ((\text{womega } f) \wedge \text{inf}) = ((\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf}) \wedge \text{inf}) \vee ((\text{womega } (f \wedge \text{finite})) \wedge \text{inf}))$

using 1 **by** *fastforce*

have 5: $\vdash (\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf}) \wedge \text{inf}) = \text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf})$

by (*metis ChopAndInf Prop10 inteq-reflection womega-and-inf womega-imp-inf*)

show *?thesis*

by (*metis 2 4 5 WPowerstar-import-finite inteq-reflection*)

qed

lemma *WOmega-And-Finite-Inf-Import*:

$\vdash (\text{womega } (f \wedge \text{finite}) \wedge \text{inf}) = (\text{womega } ((f \wedge \text{finite})))$

by (*meson Prop10 Prop11 womega-imp-inf*)

lemma *WOmega-Inf-Import-var-1*:

$\vdash ((\text{womega } f) \wedge \text{inf}) = (\text{wpowerstar } (f \wedge \text{finite}); ((f \wedge \text{inf})) \vee \text{womega } ((f \wedge \text{finite})))$

by (*meson Prop04 Prop10 womega-imp-inf womega-split-finite-inf*)

lemma *WOmega-Inf-Import-var-2:*

$\vdash ((womega(f)) \wedge inf) = (womega(f))$
by (*meson Prop10 Prop11 womega-imp-inf*)

lemma *womega-4:*

$\vdash (womega ((g \vee (womega(f))))) =$
 $((womega(g)) \vee wpowerstar g; (womega(f)))$

using *WOmega-or-unfold[of g LIFT (womega(f))]*

by (*metis (no-types, lifting) ChopAssoc Prop10 WOmegaChopWOmega WOmega-and-more-imp-more inteq-reflection womega-split-empty-more*)

lemma *womega-5:*

$\vdash (womega ((g \vee h; (womega(f))))) =$
 $((womega(g)) \vee wpowerstar g; (h; (womega(f))))$

proof –

have 1: $\vdash (g^W \vee$
 $(wpowerstar g; (h; f^W \wedge more)); (g \vee h; f^W)^W) =$
 $(g \vee h; f^W)^W$

using *WOmega-or-unfold[of g LIFT (h; (womega(f)))]* **by** *blast*

have 2: $\vdash (wpowerstar g; (h; f^W \wedge more)); (g \vee h; f^W)^W = (wpowerstar g); ((h; f^W \wedge more); ((g \vee h; f^W)^W))$

by (*meson ChopAssoc Prop11*)

have 3: $\vdash ((h; f^W \wedge more)) = (h; f^W)$

by (*metis MoreAndInfEqvInf Prop10 Prop12 RightChopImpMoreRule inteq-reflection womega-imp-inf*)

have 4: $\vdash (h; f^W); ((g \vee h; f^W)^W) = (h; f^W)$

by (*metis ChopAssoc WOmegaChopWOmega inteq-reflection*)

show *?thesis*

by (*metis 1 2 3 4 inteq-reflection*)

qed

lemma *womega-6:*

$\vdash womega ((womega(f))) = womega(f)$

by (*meson Prop04 WOmegaEqvRule WOmega-Inf-Import-var-2 womega-and-inf*)

lemma *womega-7:*

$\vdash (womega ((g; (womega(f))))) = g; (womega(f))$

proof –

have 1: $\vdash (womega ((g; (womega(f))))) = (g; (womega(f))) ; (womega ((g; (womega(f)))))$

by (*meson AndChopA LeftChopImpChop Prop11 WOmegaUnroll WOmega-star-1 WPowerstar-ext lift-imp-trans*)

have 2: $\vdash (g; (womega(f))) ; (womega ((g; (womega(f))))) = (g; (womega(f)))$

by (*metis ChopAssoc WOmegaChopWOmega inteq-reflection*)

show *?thesis* **using** 1 2 **by** *fastforce*

qed

lemma *womega-8:*

$\vdash (womega(f)); (wpowerstar (womega(f))) = womega(f)$

using *WOmega-star-5* **by** *auto*

lemma *womega-9:*

$\vdash (g; (womega(f))); (wpowerstar (g; (womega(f)))) = g; womega(f)$

by (metis inteq-reflection womega-7 womega-8)

lemma womega-10:

⊢ ((g ∨ (womega (f))) ; (wpowerstar ((g ∨ (womega (f))))) =
 (g ; wpowerstar g ∨ wpowerstar g ; (womega (f)))

proof –

have 1: ⊢ (wpowerstar ((g ∨ (womega (f)))) =
 wpowerstar (wpowerstar g ; wpowerstar ((f)^W))

using WPowerstar-denest[of g LIFT (womega (f))] **by** auto

have 2: ⊢ wpowerstar ((f)^W) = (empty ∨ ((f)^W))

by (simp add: WOmega-star-4)

have 3: ⊢ (wpowerstar g ; (empty ∨ ((f)^W))) =
 (wpowerstar g ∨ (wpowerstar g) ; ((f)^W))

by (metis ChopEmpty ChopOrEqv inteq-reflection)

have 4: ⊢ wpowerstar (wpowerstar g ∨ (wpowerstar g) ; ((f)^W)) =
 wpowerstar (wpowerstar g ; wpowerstar ((wpowerstar g) ; ((f)^W)))

using WPowerstar-denest **by** (metis WPowerstar-invol inteq-reflection)

have 5: ⊢ wpowerstar (wpowerstar g ; wpowerstar ((wpowerstar g) ; ((f)^W))) =
 wpowerstar g ; wpowerstar ((wpowerstar g) ; ((f)^W))

by (metis (no-types, opaque-lifting) ChopAssoc WOmegaChopWOmega WPowerstar-denest WPowerstar-denest-var inteq-reflection)

have 6: ⊢ wpowerstar ((wpowerstar g) ; ((f)^W)) =
 (empty ∨ wpowerstar g ; (wpowerstar ((f)^W ; wpowerstar g) ; (f)^W))

using WPowerstar-star-prod-unfold[of LIFT (wpowerstar g) LIFT ((f)^W)]

by auto

have 7: ⊢ (wpowerstar g ; (wpowerstar ((f)^W ; wpowerstar g) ; (f)^W)) =
 (wpowerstar g ; (empty ∨ ((f)^W)) ; ((f)^W))

by (metis WOmegaChopWOmega WOmega-star-4 int-simps(1) inteq-reflection)

have 8: ⊢ (wpowerstar g ; (empty ∨ ((f)^W)) ; ((f)^W)) =
 (wpowerstar g ; ((f)^W))

by (metis 2 RightChopEqvChop WOmegaChopWOmega WPowerstar-induct-lvar-eq2 inteq-reflection)

have 9: ⊢ wpowerstar ((wpowerstar g) ; ((f)^W)) = (empty ∨ (wpowerstar g ; ((f)^W)))

by (meson Prop06 WPowerstarEqv womega-9)

have 10: ⊢ (wpowerstar ((g ∨ (womega (f))))) =
 wpowerstar g ; (empty ∨ (wpowerstar g ; ((f)^W)))

by (metis 1 2 3 4 5 6 7 8 inteq-reflection)

have 11: ⊢ wpowerstar g ; (empty ∨ (wpowerstar g ; ((f)^W))) =
 (wpowerstar g ∨ (wpowerstar g ; ((f)^W)))

by (metis 10 2 3 WOmegaChopWOmega WPowerstar-denest-var inteq-reflection)

have 12: ⊢ ((g ∨ (womega (f))) ; (wpowerstar g ∨ (wpowerstar g ; ((f)^W))) =
 (g ; wpowerstar g ∨ g ; (wpowerstar g ; ((f)^W)) ∨ (womega (f)))

by (meson ChopOrEqv OrChopOr Prop04 Prop06 WOmegaChopWOmega int-simps(20))

have 13: ⊢ (g ; (wpowerstar g ; ((f)^W)) ∨ (womega (f))) =
 (wpowerstar g ; ((f)^W))

by (metis LeftChopEqvChop Prop04 WPowerstarChopEqvChopOrRule WPowerstar-tc WPowerstar-trans)

have 14: ⊢ (g ; wpowerstar g ∨ g ; (wpowerstar g ; ((f)^W)) ∨ (womega (f))) =
 (g ; wpowerstar g ∨ (wpowerstar g ; ((f)^W)))

using 13 **by** auto

show ?thesis **by** (metis 10 11 12 14 inteq-reflection)

qed

lemma *womega-11*:

$\vdash ((g \vee h;(\text{womega } (f)))); (\text{wpowerstar } ((g \vee h;(\text{womega } (f))))) =$
 $(g; \text{wpowerstar } g \vee \text{wpowerstar } g; (h;(\text{womega } (f))))$

proof –

have 1: $\vdash (\text{wpowerstar } ((g \vee h;(\text{womega } (f)))) =$
 $\text{wpowerstar } (\text{wpowerstar } g; \text{wpowerstar } (h;(f)^{\mathcal{W}}))$

using *WPowerstar-denest*[of *g LIFT h;(\text{womega } (f))*] **by** *auto*

have 2: $\vdash \text{wpowerstar } (h;(f)^{\mathcal{W}}) = (\text{empty} \vee h;((f)^{\mathcal{W}}))$
by (*meson Prop06 WPowerstarEqv womega-9*)

have 3: $\vdash (\text{wpowerstar } g; (\text{empty} \vee h;((f)^{\mathcal{W}}))) =$
 $(\text{wpowerstar } g \vee (\text{wpowerstar } g); (h;(f)^{\mathcal{W}}))$
by (*metis ChopEmpty ChopOrEqv inteq-reflection*)

have 4: $\vdash \text{wpowerstar } (\text{wpowerstar } g \vee (\text{wpowerstar } g); (h;(f)^{\mathcal{W}})) =$
 $\text{wpowerstar } (\text{wpowerstar } g; \text{wpowerstar } ((\text{wpowerstar } g); (h;(f)^{\mathcal{W}})))$
using *WPowerstar-denest* **by** (*metis WPowerstar-invol inteq-reflection*)

have 5: $\vdash \text{wpowerstar } (\text{wpowerstar } g; \text{wpowerstar } ((\text{wpowerstar } g); (h;(f)^{\mathcal{W}}))) =$
 $\text{wpowerstar } g; \text{wpowerstar } ((\text{wpowerstar } g); (h;(f)^{\mathcal{W}}))$
by (*metis (no-types, opaque-lifting) ChopAssoc WOmegaChopWOmega WPowerstar-denest WPowerstar-denest-var inteq-reflection*)

have 6: $\vdash \text{wpowerstar } ((\text{wpowerstar } g); (h;(f)^{\mathcal{W}})) =$
 $(\text{empty} \vee \text{wpowerstar } g; (\text{wpowerstar } ((h;(f)^{\mathcal{W}}); \text{wpowerstar } g); (h;(f)^{\mathcal{W}})))$
using *WPowerstar-star-prod-unfold*[of *LIFT (wpowerstar g) LIFT (h;(f)^{\mathcal{W}})*]
by *auto*

have 7: $\vdash (\text{wpowerstar } g; (\text{wpowerstar } ((h;(f)^{\mathcal{W}}); \text{wpowerstar } g); (h;(f)^{\mathcal{W}}))) =$
 $(\text{wpowerstar } g; ((\text{empty} \vee (h;(f)^{\mathcal{W}})); (h;(f)^{\mathcal{W}})))$
by (*metis 2 ChopAssoc WOmegaChopWOmega inteq-reflection*)

have 8: $\vdash (\text{wpowerstar } g; ((\text{empty} \vee (h;(f)^{\mathcal{W}})); (h;(f)^{\mathcal{W}}))) =$
 $(\text{wpowerstar } g; (h;(f)^{\mathcal{W}}))$
by (*metis 2 RightChopEqvChop WPowerstar-slide-var inteq-reflection womega-9*)

have 9: $\vdash \text{wpowerstar } ((\text{wpowerstar } g); (h;(f)^{\mathcal{W}})) =$
 $(\text{empty} \vee (\text{wpowerstar } g; (h;(f)^{\mathcal{W}})))$
by (*meson 6 7 8 Prop06*)

have 10: $\vdash (\text{wpowerstar } ((g \vee h;(\text{womega } (f))))) =$
 $\text{wpowerstar } g; (\text{empty} \vee (\text{wpowerstar } g; (h;(f)^{\mathcal{W}})))$
by (*metis 1 2 3 4 5 6 7 8 inteq-reflection*)

have 11: $\vdash \text{wpowerstar } g; (\text{empty} \vee (\text{wpowerstar } g; (h;(f)^{\mathcal{W}}))) =$
 $(\text{wpowerstar } g \vee (\text{wpowerstar } g; (h;(f)^{\mathcal{W}})))$
by (*metis (no-types, lifting) 10 3 WOmegaChopWOmega WPowerstarEqv WPowerstar-denest-var inteq-reflection womega-7*)

have 12: $\vdash ((g \vee h;(\text{womega } (f)))); (\text{wpowerstar } g \vee (\text{wpowerstar } g; (h;(f)^{\mathcal{W}}))) =$
 $(g; \text{wpowerstar } g \vee g; (\text{wpowerstar } g; (h;(f)^{\mathcal{W}})) \vee h;(\text{womega } (f)))$
by (*metis (no-types, lifting) 10 11 WPowerstarChopEqvChopOrRule inteq-reflection womega-10 womega-7*)

have 13: $\vdash (g; (\text{wpowerstar } g; (h;(f)^{\mathcal{W}})) \vee h;(\text{womega } (f))) =$
 $(\text{wpowerstar } g; (h;(f)^{\mathcal{W}}))$
by (*metis LeftChopEqvChop Prop04 WPowerstarChopEqvChopOrRule WPowerstar-tc WPowerstar-trans*)

have 14: $\vdash (g; \text{wpowerstar } g \vee g; (\text{wpowerstar } g; (h;(f)^{\mathcal{W}})) \vee h;(\text{womega } (f))) =$
 $(g; \text{wpowerstar } g \vee (\text{wpowerstar } g; (h;(f)^{\mathcal{W}})))$

using 13 by auto
 show ?thesis by (metis 10 11 12 14 inteq-reflection)
 qed

lemma *WPowerstar-imp-finite-or-finite-chop-and-inf*:

$\vdash \text{wpowerstar } f \longrightarrow \text{finite} \vee \text{finite};(f \wedge \text{inf})$

proof –

have 1: $\vdash \text{wpowerstar } f \longrightarrow \text{finite};(\text{empty} \vee (f \wedge \text{inf}))$

by (meson LeftChopImpChop Prop11 SChopstar-finite WPowerstar-SChopstar lift-imp-trans)

have 2: $\vdash \text{finite};(\text{empty} \vee (f \wedge \text{inf})) = (\text{finite} \vee \text{finite};(f \wedge \text{inf}))$

by (metis ChopEmpty ChopOrEqv int-eq)

show ?thesis using 1 2 by fastforce

qed

lemma *Inf-and-always-imp-not-wpowerstar*:

$\vdash \text{inf} \wedge \square (f \longrightarrow \text{finite}) \longrightarrow \neg(\text{wpowerstar } f)$

proof –

have 1: $\vdash (\neg (f \longrightarrow \neg \text{inf})) = (f \wedge \text{inf})$

by fastforce

have 2: $\vdash (\neg (\neg \text{inf});(\neg (f \longrightarrow \neg \text{inf}))) = (\neg((\neg \text{inf});(f \wedge \text{inf})))$

by (metis 1 int-simps(15) inteq-reflection)

show ?thesis

using WPowerstar-imp-finite-or-finite-chop-and-inf **unfolding** always-d-def sometimes-d-def finite-d-def

using 2 by fastforce

qed

lemma *WOmega-And-empty*:

$\vdash \neg(\text{womega } f \wedge \text{empty})$

proof –

have 1: $\vdash (\text{womega } f \wedge \text{empty}) =$

$(((\text{schopstar } f);(f \wedge \text{inf})) \vee (\text{omega } f)) \wedge \text{empty}$

using WOmega-Omega **by** (metis int-simps(1) inteq-reflection)

have 2: $\vdash (((\text{schopstar } f);(f \wedge \text{inf})) \vee (\text{omega } f)) \wedge \text{empty} =$

$(((\text{schopstar } f);(f \wedge \text{inf})) \wedge \text{empty}) \vee$

$(\text{omega } f \wedge \text{empty})$

by fastforce

have 3: $\vdash \neg(\text{omega } f \wedge \text{empty})$

using NotEmptyAndInf omega-imp-inf **by** fastforce

have 5: $\vdash \neg((\text{schopstar } f);(f \wedge \text{inf}) \wedge \text{empty})$

by (metis (no-types, opaque-lifting) AndInfChopEqvAndInf ChopAndEmptyEqvEmptyChopEmpty ChopAnd-Inf

InitAndEmptyEqvAndEmpty NotEmptyAndInf StateAndEmptyChop inteq-reflection lift-and-com)

show ?thesis using 1 2 3 5 by fastforce

qed

lemma *WOmega-And-More*:

$\vdash (\text{womega } f \wedge \text{more}) = (\text{womega } f)$

by (metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite FmoreEqvSkipChopFinite

Prop10 WOmega-and-more-imp-more inteq-reflection lift-imp-trans womega-fmore-inf-eq womega-imp-inf)

end

2.7 Projection operator

theory *Projection*
imports *Omega*
begin

This theory introduces the projection operator `fproj` [7]. The projection operator is defined and we prove the soundness of the rules and axiom system. We also provide the infinite projection operator `oproj` which was defined in [10].

2.7.1 Definitions

primcorec *pfilt* :: 'a intervals \Rightarrow nat nellist \Rightarrow 'a intervals

where

pfilt σ *ls* = (case *ls* of (NNil *n*) \Rightarrow (NNil (nnth σ *n*)) |
 (NCons *n* *ls1*) \Rightarrow (NCons (nnth σ *n*) (pfilt σ *ls1*)))

simps-of-case *pfilt-code* [*code*, *simp*, *nitpick-simp*]: *pfilt.code*

primcorec *lsum* :: 'a intervals intervals \Rightarrow nat \Rightarrow nat nellist

where

lsum *nells* *a* = (case *nells* of (NNil *nell*) \Rightarrow (NNil (a+(the-enat (nlength *nell*)))) |
 (NCons *nell* *nells1*) \Rightarrow (NCons (a+(the-enat (nlength *nell*)))
 (lsum *nells1* (a+(the-enat (nlength *nell*))))))

simps-of-case *lsum-code* [*code*, *simp*, *nitpick-simp*]: *lsum.code*

definition *addzero* :: nat nellist \Rightarrow nat nellist

where *addzero* *ls* = (if nlength *ls* = 0 then

(if nfirst *ls* = 0 then *ls* else (NCons 0 *ls*)) else (NCons 0 *ls*))

definition *powerinterval* :: ('a::world) formula \Rightarrow 'a intervals \Rightarrow nat nellist \Rightarrow bool

where *powerinterval* *F* σ *l* =

(\forall (*i*::nat). (enat *i*) < nlength *l* \longrightarrow ((nsubn σ (nnth *l* *i*) (nnth *l* (Suc *i*))) \models *F*))

definition *cppl* :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a intervals \Rightarrow nat nellist

where *cppl* *f* *g* σ = (ϵ *ls*. nidx *ls* \wedge (nnth *ls* 0) = 0 \wedge *powerinterval* *f* σ *ls* \wedge
 ((nfinite *ls* \wedge nfinite σ \wedge (nlast *ls*) = the-enat(nlength σ)) \vee
 (\neg nfinite *ls* \wedge \neg nfinite σ)) \wedge
 ((pfilt σ *ls*) \models *g*))

primcorec *lcpl* :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a intervals \Rightarrow nat nellist \Rightarrow nat intervals intervals

where *lcpl* *f* *g* σ *ls* =

(case *ls* of (NNil *x*) \Rightarrow (NNil (nmap (λ l. l+x) (cppl *f* *g* (nsubn σ *x*)))) |
 (NCons *x* *ls1*) \Rightarrow

$$(case\ ls1\ of\ (NNil\ y) \Rightarrow (NNil\ (nmap\ (\lambda l. l+x)\ (cppl\ f\ g\ (nsubn\ \sigma\ x\ y)))) \mid \\ (NCons\ y\ ls2) \Rightarrow (NCons\ (nmap\ (\lambda l. l+x)\ (cppl\ f\ g\ (nsubn\ \sigma\ x\ y)))\ (lcppl\ f\ g\ \sigma\ ls1))))$$

simps-of-case *lcppl-code* [*code*, *simp*, *nitpick-simp*]: *lcppl.code*

definition *fprojection-d* :: ('a:: world) formula \Rightarrow 'a formula \Rightarrow 'a formula

where *fprojection-d* *F* *G* $\equiv \lambda \sigma. (\exists\ ls. \text{nid}x\ ls \wedge (\text{nnth}\ ls\ 0) = 0 \wedge \text{nfinite}\ ls \wedge \text{nfinite}\ \sigma \wedge$
 $(\text{nlast}\ ls) = \text{the-enat}(\text{nlength}\ \sigma) \wedge$
 $\text{powerinterval}\ F\ \sigma\ ls \wedge ((\text{pfilt}\ \sigma\ ls) \models G)$
 $)$

definition *oprojection-d* :: ('a:: world) formula \Rightarrow 'a formula \Rightarrow 'a formula

where *oprojection-d* *F* *G* $\equiv \lambda \sigma. (\exists\ ls. \text{nid}x\ ls \wedge (\text{nnth}\ ls\ 0) = 0 \wedge \neg \text{nfinite}\ ls \wedge \neg \text{nfinite}\ \sigma \wedge$
 $\text{powerinterval}\ F\ \sigma\ ls \wedge ((\text{pfilt}\ \sigma\ ls) \models G)$
 $)$

syntax

-fprojection-d :: [*lift*,*lift*] \Rightarrow *lift* ((- *fproj* -) [84,84] 83)
-oprojection-d :: [*lift*,*lift*] \Rightarrow *lift* ((- *oproj* -) [84,84] 83)

syntax (*ASCII*)

-fprojection-d :: [*lift*,*lift*] \Rightarrow *lift* ((- *fproj* -) [84,84] 83)
-oprojection-d :: [*lift*,*lift*] \Rightarrow *lift* ((- *oproj* -) [84,84] 83)

translations

-fprojection-d \equiv *CONST* *fprojection-d*
-oprojection-d \equiv *CONST* *oprojection-d*

definition *ufprojection-d* :: ('a:: world) formula \Rightarrow 'a formula \Rightarrow 'a formula

where *ufprojection-d* *F* *G* $\equiv \text{LIFT}(\neg(F\ \text{fproj}\ (\neg\ G)))$

definition *uoprojection-d* :: ('a:: world) formula \Rightarrow 'a formula \Rightarrow 'a formula

where *uoprojection-d* *F* *G* $\equiv \text{LIFT}(\neg(F\ \text{oproj}\ (\neg\ G)))$

syntax

-ufprojection-d :: [*lift*,*lift*] \Rightarrow *lift* ((- *ufproj* -) [84,84] 83)

syntax

-uoprojection-d :: [*lift*,*lift*] \Rightarrow *lift* ((- *uoproj* -) [84,84] 83)

syntax (*ASCII*)

-ufprojection-d :: [*lift*,*lift*] \Rightarrow *lift* ((- *ufproj* -) [84,84] 83)
-uoprojection-d :: [*lift*,*lift*] \Rightarrow *lift* ((- *uoproj* -) [84,84] 83)

translations

-ufprojection-d \equiv *CONST* *ufprojection-d*
-uoprojection-d \equiv *CONST* *uoprojection-d*

definition $fdp-d :: ('a :: world) formula \Rightarrow 'a formula$
where $fdp-d F \equiv LIFT(\#True \ fproj \ F)$

definition $fbp-d :: ('a :: world) formula \Rightarrow 'a formula$
where $fbp-d F \equiv LIFT(\#True \ ufproj \ F)$

definition $odp-d :: ('a :: world) formula \Rightarrow 'a formula$
where $odp-d F \equiv LIFT(\#True \ oproj \ F)$

definition $obp-d :: ('a :: world) formula \Rightarrow 'a formula$
where $obp-d F \equiv LIFT(\#True \ uoproj \ F)$

syntax

$-fdp-d \quad :: lift \Rightarrow lift \quad ((fdp \ -) \ [88] \ 87)$
 $-fbp-d \quad :: lift \Rightarrow lift \quad ((fbp \ -) \ [88] \ 87)$
 $-odp-d \quad :: lift \Rightarrow lift \quad ((odp \ -) \ [88] \ 87)$
 $-obp-d \quad :: lift \Rightarrow lift \quad ((obp \ -) \ [88] \ 87)$

syntax (*ASCII*)

$-fdp-d \quad :: lift \Rightarrow lift \quad ((fdp \ -) \ [88] \ 87)$
 $-fbp-d \quad :: lift \Rightarrow lift \quad ((fbp \ -) \ [88] \ 87)$
 $-odp-d \quad :: lift \Rightarrow lift \quad ((odp \ -) \ [88] \ 87)$
 $-obp-d \quad :: lift \Rightarrow lift \quad ((obp \ -) \ [88] \ 87)$

translations

$-fdp-d \quad \Rightarrow CONST \ fdp-d$
 $-fbp-d \quad \Rightarrow CONST \ fbp-d$
 $-odp-d \quad \Rightarrow CONST \ odp-d$
 $-obp-d \quad \Rightarrow CONST \ obp-d$

abbreviation $all-nfinite \ nells \equiv (\forall \ nell \in \ nset \ nells. \ nfinite \ nell)$

abbreviation $all-gr-zero \ nells \equiv (\forall \ nell \in \ nset \ nells. \ 0 < nlength \ nell)$

2.7.2 Lemmas

Misc

lemma $all-nfinite-nnth-a:$

assumes $\forall i \leq nlength \ nells. \ nfinite \ (nnth \ nells \ i)$

shows $all-nfinite \ nells$

using $assms$

by $(metis \ in-nset-conv-nnth)$

lemma $all-nfinite-nnth-b:$

assumes $all-nfinite \ nells$

shows $\forall i \leq nlength \ nells. \ nfinite \ (nnth \ nells \ i)$

using $assms$

by $(meson \ in-nset-conv-nnth)$

lemma *all-gr-zero-nnth-a*:
assumes $\forall i \leq \text{nlength } \text{nells}. 0 < \text{nlength } (\text{nnth } \text{nells } i)$
shows *all-gr-zero nells*
using *assms*
by (*metis in-nset-conv-nnth*)

lemma *all-gr-zero-nnth-b*:
assumes *all-gr-zero nells*
shows $\forall i \leq \text{nlength } \text{nells}. 0 < \text{nlength } (\text{nnth } \text{nells } i)$
using *assms*
by (*meson in-nset-conv-nnth*)

lemma *all-nfinite-nnth-ntaken*:
assumes *all-nfinite nells*
 $n \leq \text{nlength } \text{nells}$
shows *all-nfinite (ntaken n nells)*
using *assms*
by (*meson nset-ntaken subsetD*)

lemma *all-nfinite-nnth-ndropn*:
assumes *all-nfinite nells*
 $n \leq \text{nlength } \text{nells}$
shows *all-nfinite (ndropn n nells)*
using *assms*
by (*metis nset-ndropn subsetD*)

lemma *all-nfinite-nnth-nsubn*:
assumes *all-nfinite nells*
 $n \leq \text{nlength } \text{nells}$
 $k \leq n$
shows *all-nfinite (nsubn nells k n)*
using *assms*
by (*metis in-mono nset-ndropn nset-ntaken nsubn-def1*)

lemma *all-gr-zero-nnth-ntaken*:
assumes *all-gr-zero nells*
 $n \leq \text{nlength } \text{nells}$
shows *all-gr-zero (ntaken n nells)*
using *assms*
by (*meson nset-ntaken subsetD*)

lemma *all-gr-zero-nnth-ndropn*:
assumes *all-gr-zero nells*
 $n \leq \text{nlength } \text{nells}$
shows *all-gr-zero (ndropn n nells)*
using *assms*
by (*metis nset-ndropn subsetD*)

lemma *all-gr-zero-nnth-nsubn*:

assumes *all-gr-zero nells*
 $n \leq \text{nlength } nells$
 $k \leq n$
shows *all-gr-zero (nsubn nells k n)*
using *assms*
by (*metis in-mono nset-ndropn nset-ntaken nsubn-def1*)

pfilt Lemmas

lemma *pfilt-nmap*:
 $\text{pfilt } nell \text{ } ls = \text{nmap } (\lambda x. (\text{nnth } nell \text{ } x)) \text{ } ls$
proof (*coinduction arbitrary: nell ls*)
case (*Eq-nellist nell1*)
then show *?case by simp (metis nellist.case-eq-if)*
qed

lemma *pfilt-nlength*:
 $\text{nlength}(\text{pfilt } nell \text{ } ls) = \text{nlength } ls$
by (*simp add: pfilt-nmap*)

lemma *pfilt-nnth*:
assumes $i \leq \text{nlength } (\text{pfilt } nell \text{ } ls)$
shows $(\text{nnth } (\text{pfilt } nell \text{ } ls) \text{ } i) = (\text{nnth } nell \text{ } (\text{nnth } ls \text{ } i))$
using *assms*
by (*simp add: pfilt-nmap*)

lemma *pfilt-expand*:
 $(nell1 = (\text{pfilt } nell \text{ } ls)) =$
 $(\text{nlength } nell1 = \text{nlength } ls \wedge$
 $(\forall (i::nat). i \leq \text{nlength } nell1 \longrightarrow (\text{nnth } nell1 \text{ } i) = (\text{nnth } nell \text{ } (\text{nnth } ls \text{ } i))))$
by (*metis nellist-eq-nnth-eq pfilt-nlength pfilt-nnth*)

lemma *pfilt-nfuse*:
 $\text{pfilt } nell \text{ } (\text{nfuse } ls1 \text{ } ls2) = (\text{nfuse } (\text{pfilt } nell \text{ } ls1) \text{ } (\text{pfilt } nell \text{ } ls2))$
using *pfilt-nmap[of nell (nfuse ls1 ls2)] pfilt-nmap[of nell ls1] pfilt-nmap[of nell ls2]*
by (*simp add: nmap-nfuse*)

lemma *nfuse-pfilt-nlength*:
shows $\text{nlength } (\text{pfilt } (\text{nfuse } (\text{ntaken } (\text{nnth } ls \text{ } n) \text{ } nell) \text{ } (\text{ndropn } (\text{nnth } ls \text{ } n) \text{ } nell)) \text{ } ls) =$
 $\text{nlength } (\text{nfuse } (\text{pfilt } (\text{ntaken } (\text{nnth } ls \text{ } n) \text{ } nell) \text{ } (\text{ntaken } n \text{ } ls))$
 $(\text{pfilt } (\text{ndropn } (\text{nnth } ls \text{ } n) \text{ } nell) \text{ } (\text{nmap } (\lambda x. x - (\text{nnth } ls \text{ } n)) \text{ } (\text{ndropn } n \text{ } ls)) \text{ }))$
by (*metis add.right-neutral linorder-le-cases ndropn-all nfuse-nlength nfuse-ntaken-ndropn-nlength nlength-NNil nlength-nmap ntaken-all pfilt-nlength*)

lemma *nfuse-pfilt-nnth-a*:
assumes *nidx ls*
 $(\text{nnth } ls \text{ } 0) = 0$

$(nlast\ ls) = the-enat(nlength\ nell)$
 $nfinite\ nell$
 $nfinite\ ls$
 $i \leq nlength\ ls$
 $n \leq nlength\ ls$
shows $nnth\ (pfilt\ (nfuse\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ndropn\ (nnth\ ls\ n)\ nell))\ ls)\ i =$
 $nnth\ nell\ (nnth\ ls\ i)$
using *assms*
by (*metis linorder-le-cases ndropn-all nfinite-conv-nlength-enat nfuse-ntaken-ndropn ntaken-all*
pfilt-nlength pfilt-nnth)

lemma *nfuse-pfilt-nnth-a-alt:*

assumes $nidx\ ls$
 $(nnth\ ls\ 0) = 0$
 $\neg nfinite\ nell$
 $\neg nfinite\ ls$
 $i \leq nlength\ ls$
 $n \leq nlength\ ls$
shows $nnth\ (pfilt\ (nfuse\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ndropn\ (nnth\ ls\ n)\ nell))\ ls)\ i =$
 $nnth\ nell\ (nnth\ ls\ i)$

using *assms*

by (*metis linorder-le-cases nfinite-ntaken nfuse-ntaken-ndropn ntaken-all pfilt-nlength pfilt-nnth*)

lemma *nfuse-pfilt-nnth-b:*

assumes $nidx\ ls$
 $nnth\ ls\ 0 = 0$
 $(nlast\ ls) = the-enat(nlength\ nell)$
 $nfinite\ nell$
 $nfinite\ ls$
 $i \leq nlength\ ls$
 $n \leq nlength\ ls$
shows $nnth\ (nfuse\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))$
 $(pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\)))\ i =$
 $nnth\ nell\ (nnth\ ls\ i)$

proof –

have 1: $i \leq nlength(nfuse\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))$
 $(pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\)))$

using *assms*

by (*metis nfuse-pfilt-nlength pfilt-nlength*)

have 2: $nlast(pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls)) = nnth\ nell\ (nnth\ ls\ n)$

using *assms pfilt-nnth[of - (ntaken (nnth ls n) nell) (ntaken n ls)]*

by (*simp add: nlength-eq-enat-nfiniteD nnth-nlast ntaken-nnth pfilt-nlength*)

have 3: $nfirst(pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\)) =$
 $nnth\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\))\ 0$

by (*metis ndropn-0 ndropn-nfirst*)

have 4: $nnth\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\))\ 0 =$
 $nnth\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nnth\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\)\ 0$

using *pfilt-nnth*

by (metis i0-lb zero-enat-def)
have 5: $\text{nnth } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \)) \ 0) =$
 $\text{nnth } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nnth } ls \ (n+0) - (\text{nnth } ls \ n))$
 using zero-enat-def by force
have 6: $\text{nnth } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nnth } ls \ (n+0) - (\text{nnth } ls \ n)) =$
 $\text{nnth } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ 0$
 by simp
have 7: $\text{nnth } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ 0 = \text{nnth } \text{nell} \ (\text{nnth } ls \ n)$
 using assms by simp
have 8: $\text{nlast}(\text{pfilt } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ntaken } n \ ls)) =$
 $\text{nfist}(\text{pfilt } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \))$
 using 2 4 5 7 3 6 by presburger
have 10: $\text{nnth } (\text{nfuse } (\text{pfilt } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ntaken } n \ ls))$
 $(\text{pfilt } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \))) \ i =$
 $(\text{if } i \leq \text{nlength } (\text{pfilt } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ntaken } n \ ls))$
 $\text{then } \text{nnth } (\text{pfilt } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ntaken } n \ ls)) \ i$
 $\text{else } \text{nnth } (\text{pfilt } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \))$
 $(i - (\text{the-enat } (\text{nlength } (\text{pfilt } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ntaken } n \ ls)))))$
 using 1 8 by (meson nfuse-nnth not-le-imp-less)
have 11: $\text{nlength } (\text{pfilt } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ntaken } n \ ls)) = n$
 using assms by (simp add: pfilt-nlength)
have 12: $i \leq n \longrightarrow \text{nnth } (\text{pfilt } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ntaken } n \ ls)) \ i =$
 $\text{nnth } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nnth } (\text{ntaken } n \ ls) \ i)$
 by (simp add: 11 pfilt-nnth)
have 13: $i \leq n \longrightarrow \text{nnth } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nnth } (\text{ntaken } n \ ls) \ i) =$
 $\text{nnth } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nnth } ls \ i)$
 by (simp add: ntaken-nnth)
have 15: $i \leq n \longrightarrow \text{nnth } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nnth } ls \ i) = \text{nnth } \text{nell} \ (\text{nnth } ls \ i)$
 using assms by (simp add: nidx-less-eq ntaken-nnth)
have 16: $\text{nnth } (\text{pfilt } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \))$
 $(i - (\text{the-enat } (\text{nlength } (\text{pfilt } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ntaken } n \ ls))))) =$
 $\text{nnth } (\text{pfilt } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \))$
 $(i - n)$
 by (simp add: 11)
have 17: $\text{nnth } (\text{pfilt } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \)) \ (i - n) =$
 $\text{nnth } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \)) \ (i - n))$
 using assms pfilt-nnth[of $i - n$ ($\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}$)
 $(\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \)]$
 by (metis enat-minus-mono1 idiff-enat-enat ndropn-nlength nlength-nmap pfilt-nlength)
have 18: $i > n \longrightarrow \text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \) \ (i - n) =$
 $\text{nnth } ls \ (n + (i - n)) - (\text{nnth } ls \ n)$
 using assms
 by (metis linorder-le-cases ndropn-nnth nfinite-ntaken nlast-nmap nlength-nmap nnth-nmap ntaken-all
 ntaken-nlast)
have 19: $i > n \longrightarrow \text{nnth } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \)) \ (i$
 $- n))$
 $= \text{nnth } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ ((\text{nnth } ls \ i) - (\text{nnth } ls \ n))$
 by (simp add: 18)
have 20 : $i > n \longrightarrow \text{nnth } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ ((\text{nnth } ls \ i) - (\text{nnth } ls \ n)) =$
 $\text{nnth } \text{nell} \ ((\text{nnth } ls \ n) + ((\text{nnth } ls \ i) - (\text{nnth } ls \ n)))$

using *assms ndropn-nnth by blast*
have 22: $i > n \longrightarrow (nnth\ ls\ n) + ((nnth\ ls\ i) - (nnth\ ls\ n)) = (nnth\ ls\ i)$
using *assms by (simp add: nidx-less-eq)*
have 23: $i > n \longrightarrow nnth\ nell\ ((nnth\ ls\ n) + ((nnth\ ls\ i) - (nnth\ ls\ n))) = nnth\ nell\ (nnth\ ls\ i)$
by *(simp add: 22)*
from 10 **show** *?thesis*
by *(metis 11 12 13 15 16 17 19 20 23 enat-ord-simps(1) not-le-imp-less)*
qed

lemma *nfuse-pfilt-nnth-b-alt:*

assumes *nidx ls*
 $nnth\ ls\ 0 = 0$
 $\neg nfinite\ nell$
 $\neg nfinite\ ls$
 $i \leq nlength\ ls$
 $n \leq nlength\ ls$
shows $nnth\ (nfuse\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))$
 $\quad (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\))\ i =$
 $nnth\ nell\ (nnth\ ls\ i)$
proof –
have 1: $i \leq nlength\ (nfuse\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))$
 $\quad (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\))$
using *assms*
by *(metis nfuse-pfilt-nlength pfilt-nlength)*
have 2: $nlast\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls)) = nnth\ nell\ (nnth\ ls\ n)$
using *assms pfilt-nnth[of - (ntaken (nnth ls n) nell) (ntaken n ls)]*
by *(simp add: nlength-eq-enat-nfiniteD nnth-nlast ntaken-nnth pfilt-nlength)*
have 3: $nfirst\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\)) =$
 $nnth\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\))\ 0$
by *(metis ndropn-0 ndropn-nfirst)*
have 4: $nnth\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\))\ 0 =$
 $nnth\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nnth\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\)\ 0)$
using *pfilt-nnth*
by *(metis i0-lb zero-enat-def)*
have 5: $nnth\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nnth\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\)\ 0 =$
 $nnth\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nnth\ ls\ (n+0) - (nnth\ ls\ n))$
using *zero-enat-def by force*
have 6: $nnth\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nnth\ ls\ (n+0) - (nnth\ ls\ n)) =$
 $nnth\ (ndropn\ (nnth\ ls\ n)\ nell)\ 0$
by *simp*
have 7: $nnth\ (ndropn\ (nnth\ ls\ n)\ nell)\ 0 = nnth\ nell\ (nnth\ ls\ n)$
using *assms by simp*
have 8: $nlast\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls)) =$
 $nfirst\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\))$
using 2 4 5 7 3 6 **by** *presburger*
have 10: $nnth\ (nfuse\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))$
 $\quad (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\))\ i =$
 $(if\ i \leq nlength\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))$
 $\quad then\ nnth\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))\ i$

else $\text{nnth } (\text{pfilt } (\text{ndropn } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } n)) (\text{ndropn } n \text{ ls})))$
 $(i - (\text{the-enat } (\text{nlength } (\text{pfilt } (\text{ntaken } (\text{nnth } \text{ls } n) \text{ nell}) (\text{ntaken } n \text{ ls}))))))$
 using 1 8 by (meson nfuse-nnth not-le-imp-less)
 have 11: $\text{nlength } (\text{pfilt } (\text{ntaken } (\text{nnth } \text{ls } n) \text{ nell}) (\text{ntaken } n \text{ ls})) = n$
 using assms by (simp add: pfilt-nlength)
 have 12: $i \leq n \longrightarrow \text{nnth } (\text{pfilt } (\text{ntaken } (\text{nnth } \text{ls } n) \text{ nell}) (\text{ntaken } n \text{ ls})) i =$
 $\text{nnth } (\text{ntaken } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nnth } (\text{ntaken } n \text{ ls}) i)$
 by (simp add: 11 pfilt-nnth)
 have 13: $i \leq n \longrightarrow \text{nnth } (\text{ntaken } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nnth } (\text{ntaken } n \text{ ls}) i) =$
 $\text{nnth } (\text{ntaken } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nnth } \text{ls } i)$
 by (simp add: ntaken-nnth)
 have 15: $i \leq n \longrightarrow \text{nnth } (\text{ntaken } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nnth } \text{ls } i) = \text{nnth } \text{nell } (\text{nnth } \text{ls } i)$
 using assms by (simp add: nidx-less-eq ntaken-nnth)
 have 16: $\text{nnth } (\text{pfilt } (\text{ndropn } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } n)) (\text{ndropn } n \text{ ls})))$
 $(i - (\text{the-enat } (\text{nlength } (\text{pfilt } (\text{ntaken } (\text{nnth } \text{ls } n) \text{ nell}) (\text{ntaken } n \text{ ls})))) =$
 $\text{nnth } (\text{pfilt } (\text{ndropn } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } n)) (\text{ndropn } n \text{ ls})))$
 $(i - n)$
 by (simp add: 11)
 have 17: $\text{nnth } (\text{pfilt } (\text{ndropn } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } n)) (\text{ndropn } n \text{ ls}))) (i - n) =$
 $\text{nnth } (\text{ndropn } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } n)) (\text{ndropn } n \text{ ls})) (i - n))$
 using assms pfilt-nnth[of $i - n$ ($\text{ndropn } (\text{nnth } \text{ls } n) \text{ nell}$)
 $(\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } n)) (\text{ndropn } n \text{ ls}))]$
 by (metis enat-minus-mono1 idiff-enat-enat ndropn-nlength nlength-nmap pfilt-nlength)
 have 18: $i > n \longrightarrow \text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } n)) (\text{ndropn } n \text{ ls})) (i - n) =$
 $\text{nnth } \text{ls } (n + (i - n)) - (\text{nnth } \text{ls } n)$
 using assms
 by (metis linorder-le-cases ndropn-nnth nfinite-ntaken nlast-nmap nlength-nmap nnth-nmap ntaken-all
 ntaken-nlast)
 have 19: $i > n \longrightarrow \text{nnth } (\text{ndropn } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } n)) (\text{ndropn } n \text{ ls})) (i - n))$
 $= \text{nnth } (\text{ndropn } (\text{nnth } \text{ls } n) \text{ nell}) ((\text{nnth } \text{ls } i) - (\text{nnth } \text{ls } n))$
 by (simp add: 18)
 have 20: $i > n \longrightarrow \text{nnth } (\text{ndropn } (\text{nnth } \text{ls } n) \text{ nell}) ((\text{nnth } \text{ls } i) - (\text{nnth } \text{ls } n)) =$
 $\text{nnth } \text{nell } ((\text{nnth } \text{ls } n) + ((\text{nnth } \text{ls } i) - (\text{nnth } \text{ls } n)))$
 using assms ndropn-nnth by blast
 have 22: $i > n \longrightarrow (\text{nnth } \text{ls } n) + ((\text{nnth } \text{ls } i) - (\text{nnth } \text{ls } n)) = (\text{nnth } \text{ls } i)$
 using assms by (simp add: nidx-less-eq)
 have 23: $i > n \longrightarrow \text{nnth } \text{nell } ((\text{nnth } \text{ls } n) + ((\text{nnth } \text{ls } i) - (\text{nnth } \text{ls } n))) = \text{nnth } \text{nell } (\text{nnth } \text{ls } i)$
 by (simp add: 22)
 from 10 show ?thesis
 by (metis 11 12 13 15 16 17 19 20 23 enat-ord-simps(1) not-le-imp-less)
 qed

lemma nfuse-pfilt-nnth:

assumes nidx ls

$\text{nnth } \text{ls } 0 = 0$

$(\text{nlast } \text{ls}) = \text{the-enat}(\text{nlength } \text{nell})$

nfinite nell

nfinite ls

$i \leq \text{nlength } ls$
 $n \leq \text{nlength } ls$
shows $\text{nnth } (\text{pfilt } (\text{nfuse } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell})) \ ls) \ i =$
 $\text{nnth } (\text{nfuse } (\text{pfilt } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ntaken } n \ ls)))$
 $(\text{pfilt } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \))) \ i$
using $\text{assms } \text{nfuse-pfilt-nnth-a}[\text{of } ls \ \text{nell } i \ n]$
 $\text{nfuse-pfilt-nnth-b}[\text{of } ls \ \text{nell } i \ n]$
by *simp*

lemma *nfuse-pfilt-nnth-alt:*

assumes $\text{nidx } ls$
 $\text{nnth } ls \ 0 = 0$
 $\neg \text{nfinite } nell$
 $\neg \text{nfinite } ls$
 $i \leq \text{nlength } ls$
 $n \leq \text{nlength } ls$
shows $\text{nnth } (\text{pfilt } (\text{nfuse } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell})) \ ls) \ i =$
 $\text{nnth } (\text{nfuse } (\text{pfilt } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ntaken } n \ ls)))$
 $(\text{pfilt } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \))) \ i$
using $\text{assms } \text{nfuse-pfilt-nnth-a-alt}[\text{of } ls \ \text{nell } i \ n]$
 $\text{nfuse-pfilt-nnth-b-alt}[\text{of } ls \ \text{nell } i \ n]$
by *simp*

lemma *nfuse-pfilt:*

assumes $\text{nidx } ls$
 $\text{nnth } ls \ 0 = 0$
 $(\text{nlast } ls) = \text{the-enat}(\text{nlength } nell)$
 $\text{nfinite } nell$
 $\text{nfinite } ls$
 $n \leq \text{nlength } ls$
shows $\text{pfilt } (\text{nfuse } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell})) \ ls =$
 $\text{nfuse } (\text{pfilt } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ntaken } n \ ls))$
 $(\text{pfilt } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \)))$
using $\text{assms } \text{nfuse-pfilt-nlength } \text{nfuse-pfilt-nnth}$
 $\text{nellist-eq-nnth-eq}$ **by** (*metis pfilt-nlength*)

lemma *nfuse-pfilt-alt:*

assumes $\text{nidx } ls$
 $\text{nnth } ls \ 0 = 0$
 $\neg \text{nfinite } nell$
 $\neg \text{nfinite } ls$
 $n \leq \text{nlength } ls$
shows $\text{pfilt } (\text{nfuse } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell})) \ ls =$
 $\text{nfuse } (\text{pfilt } (\text{ntaken } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{ntaken } n \ ls))$
 $(\text{pfilt } (\text{ndropn } (\text{nnth } ls \ n) \ \text{nell}) \ (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \)))$
using $\text{assms } \text{nfuse-pfilt-nlength } \text{nfuse-pfilt-nnth-alt}$
 $\text{nellist-eq-nnth-eq}$ **by** (*metis pfilt-nlength*)

lemma *nfuse-pfilt-a:*

```

assumes nidx ls1
  nfinite ls1
  nnth ls1 0 = 0
  nidx ls2
  nfinite ls2
  nnth ls2 0 = nlast ls1
  (nlast ls2) = the-enat(nlength nell)
  nfinite nell
shows pfilt (nfuse (ntaken (nlast ls1) nell) (ndropn (nfirst ls2) nell)) (nfuse ls1 ls2) =
  nfuse (pfilt (ntaken (nlast ls1) nell) ls1)
  (pfilt (ndropn (nfirst ls2) nell) (nmap (λx. x - (nfirst ls2)) ls2))
proof -
  have 0: ∃ ll. (enat ll) = nlength ls1
    using assms(2) nfinite-nlength-enat by fastforce
  obtain ll where 01: (enat ll) = nlength ls1
    using 0 by auto
  have 1: nlast ls1 = nfirst ls2
    using assms by (metis nlast-NNil ntaken-0 ntaken-nlast)
  have 2: nidx (nfuse ls1 ls2)
    using assms nidx-nfuse by blast
  have 20: nnth (nfuse ls1 ls2) 0 = 0
    by (metis 1 assms(3) nfuse-nnth zero-enat-def zero-le)
  have 3: nlast(nfuse ls1 ls2) = the-enat(nlength nell)
    using assms
    by (simp add: 1 nlast-nfuse)
  have 4: ll ≤ nlength (nfuse ls1 ls2)
    by (simp add: 01 nfuse-nlength)
  have 5: pfilt (nfuse (ntaken (nnth (nfuse ls1 ls2) ll) nell)
    (ndropn (nnth (nfuse ls1 ls2) ll) nell))
    (nfuse ls1 ls2)
    =
    nfuse (pfilt (ntaken (nnth (nfuse ls1 ls2) ll) nell) (ntaken ll (nfuse ls1 ls2)))
    (pfilt (ndropn (nnth (nfuse ls1 ls2) ll) nell)
    (nmap (λ x. x - (nnth (nfuse ls1 ls2) ll)) (ndropn ll (nfuse ls1 ls2))))
    using 2 3 4 nfuse-pfilt[of (nfuse ls1 ls2) nell ll]
    using 20 assms nfuse-nfinite by blast
  have 6: pfilt (nfuse (ntaken (nlast ls1) nell) (ndropn (nfirst ls2) nell)) (nfuse ls1 ls2) =
    pfilt (nfuse (ntaken (nnth (nfuse ls1 ls2) ll) nell)
    (ndropn (nnth (nfuse ls1 ls2) ll) nell))
    (nfuse ls1 ls2)
    using 1 4
    by (metis 01 assms(2) ntaken-nfuse ntaken-nlast the-enat.simps)
  have 7: (pfilt (ntaken (nlast ls1) nell) ls1) =
    (pfilt (ntaken (nnth (nfuse ls1 ls2) ll) nell) (ntaken ll (nfuse ls1 ls2)))
    using 1 4
    by (metis 01 assms(2) ntaken-nfuse ntaken-nlast the-enat.simps)
  have 8: (pfilt (ndropn (nfirst ls2) nell) (nmap (λx. x - (nfirst ls2)) ls2)) =
    (pfilt (ndropn (nnth (nfuse ls1 ls2) ll) nell)
    (nmap (λx. x - (nnth (nfuse ls1 ls2) ll)) (ndropn ll (nfuse ls1 ls2)) ))
    using 1 4

```

```

  by (metis 01 ndropn-nfirst ndropn-nfuse nfinite-conv-nlength-enat the-enat.simps)
show ?thesis using 5 6 7 8 by auto
qed

```

lemma *nfuse-pfilt-a-alt*:

assumes *nidx ls1*

nfinite ls1

nnth ls1 0 = 0

nidx ls2

\neg *nfinite ls2*

nnth ls2 0 = nlast ls1

\neg *nfinite nell*

shows $\text{pfilt } (\text{nfuse } (\text{ntaken } (\text{nlast } \text{ls1}) \text{ nell}) (\text{ndropn } (\text{nfirst } \text{ls2}) \text{ nell})) (\text{nfuse } \text{ls1 } \text{ls2}) =$
 $\text{nfuse } (\text{pfilt } (\text{ntaken } (\text{nlast } \text{ls1}) \text{ nell}) \text{ls1})$
 $(\text{pfilt } (\text{ndropn } (\text{nfirst } \text{ls2}) \text{ nell}) (\text{nmap } (\lambda x. x - (\text{nfirst } \text{ls2})) \text{ls2}))$

proof –

have 0: $\exists ll. (\text{enat } ll) = \text{nlength } \text{ls1}$

using *assms(2) nfinite-nlength-enat* **by** *fastforce*

obtain *ll* **where** 01: $(\text{enat } ll) = \text{nlength } \text{ls1}$

using 0 **by** *auto*

have 1: $\text{nlast } \text{ls1} = \text{nfirst } \text{ls2}$

using *assms* **by** (*metis nlast-NNil ntaken-0 ntaken-nlast*)

have 2: *nidx* (*nfuse* *ls1* *ls2*)

using *assms nidx-nfuse* **by** *blast*

have 20: *nnth* (*nfuse* *ls1* *ls2*) 0 = 0

by (*metis 1 assms(3) nfuse-nnth zero-enat-def zero-le*)

have 4: $ll \leq \text{nlength } (\text{nfuse } \text{ls1 } \text{ls2})$

by (*simp add: 01 nfuse-nlength*)

have 5: $\text{pfilt } (\text{nfuse } (\text{ntaken } (\text{nnth } (\text{nfuse } \text{ls1 } \text{ls2}) ll) \text{ nell})$
 $(\text{ndropn } (\text{nnth } (\text{nfuse } \text{ls1 } \text{ls2}) ll) \text{ nell}))$
 $(\text{nfuse } \text{ls1 } \text{ls2})$

=

$\text{nfuse } (\text{pfilt } (\text{ntaken } (\text{nnth } (\text{nfuse } \text{ls1 } \text{ls2}) ll) \text{ nell}) (\text{ntaken } ll (\text{nfuse } \text{ls1 } \text{ls2})))$
 $(\text{pfilt } (\text{ndropn } (\text{nnth } (\text{nfuse } \text{ls1 } \text{ls2}) ll) \text{ nell})$
 $(\text{nmap } (\lambda x. x - (\text{nnth } (\text{nfuse } \text{ls1 } \text{ls2}) ll)) (\text{ndropn } ll (\text{nfuse } \text{ls1 } \text{ls2}))))$

using 2 4 *nfuse-pfilt-alt[of (nfuse ls1 ls2) nell ll]* 20 *assms nfuse-nfinite* **by** *blast*

have 6: $\text{pfilt } (\text{nfuse } (\text{ntaken } (\text{nlast } \text{ls1}) \text{ nell}) (\text{ndropn } (\text{nfirst } \text{ls2}) \text{ nell})) (\text{nfuse } \text{ls1 } \text{ls2}) =$
 $\text{pfilt } (\text{nfuse } (\text{ntaken } (\text{nnth } (\text{nfuse } \text{ls1 } \text{ls2}) ll) \text{ nell})$
 $(\text{ndropn } (\text{nnth } (\text{nfuse } \text{ls1 } \text{ls2}) ll) \text{ nell}))$
 $(\text{nfuse } \text{ls1 } \text{ls2})$

using 1 4

by (*metis 01 assms(2) ntaken-nfuse ntaken-nlast the-enat.simps*)

have 7: $(\text{pfilt } (\text{ntaken } (\text{nlast } \text{ls1}) \text{ nell}) \text{ls1}) =$

$(\text{pfilt } (\text{ntaken } (\text{nnth } (\text{nfuse } \text{ls1 } \text{ls2}) ll) \text{ nell}) (\text{ntaken } ll (\text{nfuse } \text{ls1 } \text{ls2})))$

using 1 4

by (*metis 01 assms(2) ntaken-nfuse ntaken-nlast the-enat.simps*)

have 8: $(\text{pfilt } (\text{ndropn } (\text{nfirst } \text{ls2}) \text{ nell}) (\text{nmap } (\lambda x. x - (\text{nfirst } \text{ls2})) \text{ls2})) =$

$(\text{pfilt } (\text{ndropn } (\text{nnth } (\text{nfuse } \text{ls1 } \text{ls2}) ll) \text{ nell})$

$(\text{nmap } (\lambda x. x - (\text{nnth } (\text{nfuse } \text{ls1 } \text{ls2}) ll)) (\text{ndropn } ll (\text{nfuse } \text{ls1 } \text{ls2})))$


```

using 1 4
by (metis 01 ndropn-nfirst ndropn-nfuse nfinite-conv-nlength-enat the-enat.simps)
show ?thesis using 5 6 7 8 by auto
qed

```

```

lemma pfilt-ntaken:
assumes  $n \leq \text{nlength } ls$ 
shows  $\text{ntaken } n (\text{pfilt } \sigma \text{ } ls) = \text{pfilt } \sigma (\text{ntaken } n \text{ } ls)$ 
proof -
have 1:  $\text{nlength } (\text{ntaken } n (\text{pfilt } \sigma \text{ } ls)) = \text{nlength } (\text{pfilt } \sigma (\text{ntaken } n \text{ } ls))$ 
by (simp add: assms pfilt-nlength)
have 2:  $\forall (i::\text{nat}). i \leq \text{nlength}(\text{ntaken } n (\text{pfilt } \sigma \text{ } ls)) \longrightarrow$ 
 $(\text{nnth } (\text{ntaken } n (\text{pfilt } \sigma \text{ } ls)) \text{ } i) = (\text{nnth } (\text{pfilt } \sigma (\text{ntaken } n \text{ } ls)) \text{ } i)$ 
by (simp add: pfilt-nmap)
show ?thesis using 1 2 nellist-eq-nnth-eq by blast
qed

```

```

lemma pfilt-ntaken-nidx:
assumes  $n \leq \text{nlength } nell$ 
 $\text{nidx } ls$ 
 $\text{nnth } ls \text{ } 0 = 0$ 
 $\text{nfinite } ls$ 
 $\text{nlast } ls = n$ 
shows  $\text{pfilt } nell \text{ } ls = \text{pfilt } (\text{ntaken } n \text{ } nell) \text{ } ls$ 
proof -
have 1:  $\text{nlength}(\text{pfilt } nell \text{ } ls) = \text{nlength } (\text{pfilt } (\text{ntaken } n \text{ } nell) \text{ } ls)$ 
by (simp add: pfilt-nlength)
have 2:  $\forall (i::\text{nat}). i \leq \text{nlength}(\text{pfilt } nell \text{ } ls) \longrightarrow$ 
 $(\text{nnth } (\text{pfilt } nell \text{ } ls) \text{ } i) = (\text{nnth } (\text{pfilt } (\text{ntaken } n \text{ } nell) \text{ } ls) \text{ } i)$ 
using assms
by (auto simp add: nidx-all-le-nlast ntaken-nnth pfilt-nlength pfilt-nnth)
show ?thesis
by (simp add: 1 2 nellist-eq-nnth-eq)
qed

```

```

lemma pfilt-ndropn:
assumes  $n \leq \text{nlength } ls$ 
shows  $\text{ndropn } n (\text{pfilt } nell \text{ } ls) = \text{pfilt } nell (\text{ndropn } n \text{ } ls)$ 
proof -
have 1:  $\text{nlength } (\text{ndropn } n (\text{pfilt } nell \text{ } ls)) = \text{nlength } (\text{pfilt } nell (\text{ndropn } n \text{ } ls))$ 
by (simp add: assms pfilt-nlength)
have 2:  $\forall (i::\text{nat}). i \leq \text{nlength } (\text{ndropn } n (\text{pfilt } nell \text{ } ls)) \longrightarrow$ 
 $(\text{nnth } (\text{ndropn } n (\text{pfilt } nell \text{ } ls)) \text{ } i) = (\text{nnth } (\text{pfilt } nell (\text{ndropn } n \text{ } ls)) \text{ } i)$ 
by (simp add: ndropn-nmap pfilt-nmap)
show ?thesis using 1 2 nellist-eq-nnth-eq by blast
qed

```

lemma *pfilt-ndropn-nidx-nlength*:
assumes $(\text{enat } n) \leq \text{nlength } \text{nell}$
 $\text{nidx } \text{ls}$
 $\text{nnth } \text{ls } 0 = 0$
shows $\text{nlength } (\text{pfilt } \text{nell } (\text{nmap } (\lambda x. x + n) \text{ls})) = \text{nlength } (\text{pfilt } (\text{ndropn } n \text{nell}) \text{ls})$
using *assms* **by** (*simp add: pfilt-nlength*)

lemma *pfilt-ndropn-nidx-nnth*:
assumes $(\text{enat } n) \leq \text{nlength } \text{nell}$
 $\text{nidx } \text{ls}$
 $\text{nnth } \text{ls } 0 = 0$
 $i \leq \text{nlength } \text{ls}$
shows $\text{nnth } (\text{pfilt } \text{nell } (\text{nmap } (\lambda x. x + n) \text{ls})) i = \text{nnth } (\text{pfilt } (\text{ndropn } n \text{nell}) \text{ls}) i$
proof –
have 1: $\text{nnth } (\text{pfilt } \text{nell } (\text{nmap } (\lambda x. x + n) \text{ls})) i = \text{nnth } \text{nell } (\text{nnth } (\text{nmap } (\lambda x. x + n) \text{ls}) i)$
by (*simp add: assms pfilt-nnth pfilt-nlength*)
have 2: $\text{nnth } \text{nell } (\text{nnth } (\text{nmap } (\lambda x. x + n) \text{ls}) i) = (\text{nnth } \text{nell } ((\text{nnth } \text{ls } i) + n))$
by (*simp add: assms*)
have 3: $\text{nnth } (\text{pfilt } (\text{ndropn } n \text{nell}) \text{ls}) i = \text{nnth } (\text{ndropn } n \text{nell}) (\text{nnth } \text{ls } i)$
by (*simp add: assms pfilt-nnth pfilt-nlength*)
have 4: $\text{nnth } (\text{ndropn } n \text{nell}) (\text{nnth } \text{ls } i) = (\text{nnth } \text{nell } ((\text{nnth } \text{ls } i) + n))$
by (*metis ntaken-ndropn-nlast ntaken-nlast*)
show ?thesis
using 1 2 3 4 **by** *presburger*
qed

lemma *pfilt-ndropn-nidx*:
assumes $(\text{enat } n) \leq \text{nlength } \text{nell}$
 $\text{nidx } \text{ls}$
 $\text{nnth } \text{ls } 0 = 0$
shows $\text{pfilt } \text{nell } (\text{nmap } (\lambda x. x + n) \text{ls}) = \text{pfilt } (\text{ndropn } n \text{nell}) \text{ls}$
using *assms pfilt-ndropn-nidx-nlength pfilt-ndropn-nidx-nnth nellist-eq-nnth-eq*
by (*simp add: pfilt-ndropn-nidx-nnth nellist-eq-nnth-eq pfilt-nlength*)

lemma *pfilt-pfilt*:
 $(\text{nnth } (\text{pfilt } (\text{pfilt } \text{nell } \text{ls1}) \text{ls2}) k) = (\text{nnth } \text{nell } (\text{nnth } \text{ls1 } (\text{nnth } \text{ls2 } k)))$
by (*metis enat-le-plus-same(1) gen-nlength-def ndropn-nmap nlength-code nnth-nmap nnth-zero-ndropn pfilt-nmap*)

lemma *pfilt-pfilt-nmap*:
 $(\text{pfilt } (\text{pfilt } \text{nell } \text{ls1}) \text{ls2}) = (\text{pfilt } \text{nell } (\text{nmap } (\lambda x. (\text{nnth } \text{ls1 } x)) \text{ls2}))$
by (*simp add: pfilt-expand pfilt-nlength pfilt-pfilt*)

lemma *pfilt-nmap-pfilt*:
 $(\text{pfilt } (\text{pfilt } \text{nell } \text{ls1}) \text{ls2}) = \text{pfilt } \text{nell } (\text{pfilt } \text{ls1 } \text{ls2})$
by (*metis pfilt-nmap pfilt-pfilt-nmap*)

lemma *pfilt-nsubn*:

assumes $k \leq n$
 $n \leq \text{nlength } ls$
shows $(\text{nsubn } (\text{pfilt } nell \text{ } ls)) \text{ } k \text{ } n = (\text{pfilt } nell (\text{nsubn } ls \text{ } k \text{ } n))$
using *assms*
by (*simp add: ndropn-nmap nsubn-def1 pfilt-nmap*)

lemma *pfilt-nfusecat-nmap*:
 $(\text{pfilt } nell (\text{nfusecat } lls)) = (\text{nfusecat } (\text{nmap } (\lambda \text{ } ls . (\text{pfilt } nell \text{ } ls)) \text{ } lls))$
proof –
have 1: $\text{pfilt } nell (\text{nfusecat } lls) = \text{nmap } (\text{nnth } nell) (\text{nfusecat } lls)$
using *pfilt-nmap[of nell (nfusecat lls)] by blast*
have 2: $(\text{nfusecat } (\text{nmap } (\lambda \text{ } ls . (\text{pfilt } nell \text{ } ls)) \text{ } lls)) =$
 $(\text{nfusecat } (\text{nmap } (\text{nmap } (\text{nnth } nell) \text{ }) \text{ } lls))$
using *pfilt-nmap by metis*
have 3: $\text{nmap } (\text{nnth } nell) (\text{nfusecat } lls) = (\text{nfusecat } (\text{nmap } (\text{nmap } (\text{nnth } nell) \text{ }) \text{ } lls))$
by (*simp add: nmap-nfusecat*)
show ?thesis
by (*simp add: 1 2 3*)
qed

powerinterval lemmas

lemma *powerinterval-split0*:
assumes *nidx ls*
 $\text{nnth } ls \text{ } 0 = 0$
 $n \leq \text{nlength } ls$
nfinite ls
nfinite σ
 $\text{nlast } ls = \text{the-enat}(\text{nlength } \sigma)$
 $i < \text{nlength}(\text{ntaken } n \text{ } ls)$
shows $(\text{nsubn } (\text{ntaken } (\text{nnth } ls \text{ } n) \text{ } \sigma) \text{ } (\text{nnth } (\text{ntaken } n \text{ } ls) \text{ } i) \text{ } (\text{nnth } (\text{ntaken } n \text{ } ls) \text{ } (\text{Suc } i))) =$
 $(\text{nsubn } \sigma \text{ } (\text{nnth } ls \text{ } i) \text{ } (\text{nnth } ls \text{ } (\text{Suc } i)))$
proof –
have 01: $(\text{nnth } ls \text{ } n) \leq \text{nlength } \sigma$
using *assms*
by (*metis enat-ord-simps(1) nfinite-nlength-enat nidx-less-eq nnth-nlast the-enat.simps*
verit-comp-simplify1(2))
have 02: $(\text{nnth } (\text{ntaken } n \text{ } ls) \text{ } i) \leq (\text{nnth } (\text{ntaken } n \text{ } ls) \text{ } (\text{Suc } i))$
using *assms*
by (*metis dual-order.trans eSuc-enat enat-ord-simps(1) ileI1 le-add2 min-def nidx-less-eq*
ntaken-nlength ntaken-nnth plus-1-eq-Suc)
have 03: $(\text{nnth } (\text{ntaken } n \text{ } ls) \text{ } (\text{Suc } i)) \leq (\text{nnth } ls \text{ } n)$
using *assms*
by (*metis eSuc-enat enat-ord-simps(1) ileI1 min-def nidx-less-eq ntaken-nlength ntaken-nnth*)
show ?thesis **unfolding** *nsubn-def1* **using** 01 02 03 *assms*
by (*simp add: ntaken-nnth ntaken-ndropn-swap order-subst2*)
qed

lemma *powerinterval-split0-alt*:

assumes *nidx ls*

$nnth\ ls\ 0 = 0$
 $n \leq nlength\ ls$
 $\neg nfinite\ ls$
 $\neg nfinite\ \sigma$
 $i < nlength(ntaken\ n\ ls)$
shows $(nsubn\ (ntaken\ (nnth\ ls\ n)\ \sigma)\ (nnth\ (ntaken\ n\ ls)\ i)\ (nnth\ (ntaken\ n\ ls)\ (Suc\ i))) =$
 $(nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
proof –
have 01: $(nnth\ ls\ n) \leq nlength\ \sigma$
by (meson assms(5) enat-less-imp-le less-enatE nfinite-conv-nlength-enat)
have 02: $(nnth\ (ntaken\ n\ ls)\ i) \leq (nnth\ (ntaken\ n\ ls)\ (Suc\ i))$
using assms
by (metis dual-order.trans eSuc-enat enat-ord-simps(1) ileI1 le-add2 min-def nidx-less-eq
ntaken-nlength ntaken-nnth plus-1-eq-Suc)
have 03: $(nnth\ (ntaken\ n\ ls)\ (Suc\ i)) \leq (nnth\ ls\ n)$
using assms
by (metis eSuc-enat enat-ord-simps(1) ileI1 min-def nidx-less-eq ntaken-nlength ntaken-nnth)
show ?thesis **unfolding** nsubn-def1 **using** 01 02 03 assms
by (simp add: ntaken-nnth ntaken-ndropn-swap order-subst2)
qed

lemma powerinterval-splita:

assumes $nidx\ ls$
 $nnth\ ls\ 0 = 0$
 $nfinite\ ls$
 $n \leq nlength\ ls$
 $nfinite\ \sigma$
 $nlast\ ls = the-enat(nlength\ \sigma)$
 $powerinterval\ f\ \sigma\ ls$
shows $powerinterval\ f\ (ntaken\ (nnth\ ls\ n)\ \sigma)\ (ntaken\ n\ ls)$
proof –
have 0: $(\forall i. i < nlength\ ls \longrightarrow ((nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f))$
using assms **by** (simp add: powerinterval-def)
have 1: $powerinterval\ f\ (ntaken\ (nnth\ ls\ n)\ \sigma)\ (ntaken\ n\ ls) =$
 $(\forall i. i < nlength(ntaken\ n\ ls) \longrightarrow$
 $((nsubn\ (ntaken\ (nnth\ ls\ n)\ \sigma)\ (nnth\ (ntaken\ n\ ls)\ i)\ (nnth\ (ntaken\ n\ ls)\ (Suc\ i))) \models f))$
using powerinterval-def **by** blast
have 2: $(\forall i. i < nlength(ntaken\ n\ ls) \longrightarrow$
 $((nsubn\ (ntaken\ (nnth\ ls\ n)\ \sigma)\ (nnth\ (ntaken\ n\ ls)\ i)\ (nnth\ (ntaken\ n\ ls)\ (Suc\ i))) \models f))$
proof
fix i
show $i < nlength(ntaken\ n\ ls) \longrightarrow$
 $((nsubn\ (ntaken\ (nnth\ ls\ n)\ \sigma)\ (nnth\ (ntaken\ n\ ls)\ i)\ (nnth\ (ntaken\ n\ ls)\ (Suc\ i))) \models f)$
proof –
have 21: $i < n \longrightarrow ((nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f)$
using 0 assms less-le-trans **by** (metis enat-ord-simps(2))
have 22: $i < nlength(ntaken\ n\ ls) \longrightarrow$
 $((nsubn\ \sigma\ (nnth\ (ntaken\ n\ ls)\ i)\ (nnth\ (ntaken\ n\ ls)\ (Suc\ i))) \models f)$
by (simp add: 21 assms(4) ntaken-nnth)
have 23: $i < nlength(ntaken\ n\ ls) \longrightarrow$

```

      ((nsubn (ntaken (nnth ls n)  $\sigma$ ) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) )  $\models$  f)
    using 22 assms powerinterval-splita0
    using 21 by fastforce
    show ?thesis using 23 by blast
  qed
  qed
  show ?thesis using 1 2 by blast
qed

lemma powerinterval-splita-alt:
  assumes nidx ls
    nnth ls 0 = 0
     $\neg$ nfinite ls
     $n \leq$  nlength ls
     $\neg$ nfinite  $\sigma$ 
    powerinterval f  $\sigma$  ls
  shows powerinterval f (ntaken (nnth ls n)  $\sigma$ ) (ntaken n ls)
proof -
  have 0: ( $\forall i. i < \text{nlength } ls \longrightarrow ( (nsubn \sigma (nnth ls i) (nnth ls (Suc i)) ) \models f )$ )
    using assms by (simp add: powerinterval-def)
  have 1: powerinterval f (ntaken (nnth ls n)  $\sigma$ ) (ntaken n ls) =
    ( $\forall i. i < \text{nlength}(ntaken n ls) \longrightarrow$ 
      ((nsubn (ntaken (nnth ls n)  $\sigma$ ) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) )  $\models$  f))
    using powerinterval-def by blast
  have 2: ( $\forall i. i < \text{nlength}(ntaken n ls) \longrightarrow$ 
    ((nsubn (ntaken (nnth ls n)  $\sigma$ ) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) )  $\models$  f))
  proof
    fix i
    show  $i < \text{nlength}(ntaken n ls) \longrightarrow$ 
      ((nsubn (ntaken (nnth ls n)  $\sigma$ ) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) )  $\models$  f)
  proof -
    have 21:  $i < n \longrightarrow ((nsubn \sigma (nnth ls i) (nnth ls (Suc i)) ) \models f)$ 
      using 0 assms less-le-trans by (metis enat-ord-simps(2))
    have 22:  $i < \text{nlength}(ntaken n ls) \longrightarrow$ 
      ((nsubn  $\sigma$  (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) )  $\models$  f)
      by (simp add: 21 assms(4) ntaken-nnth)
    have 23:  $i < \text{nlength}(ntaken n ls) \longrightarrow$ 
      ((nsubn (ntaken (nnth ls n)  $\sigma$ ) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) )  $\models$  f)
      using 22 assms powerinterval-splita0-alt
      using 21 by fastforce
    show ?thesis using 23 by blast
  qed
  qed
  show ?thesis using 1 2 by blast
qed

lemma powerinterval-splitb0:
  assumes nidx ls
    nnth ls 0 = 0
    nfinite ls

```

```

    n ≤ nlength ls
  nfinite σ
  nlast ls = the-enat(nlength σ)
  i < (nlength ls - n)
shows (nsubn (ndropn (nnth ls n) σ)
        (nnth (((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) i)
        (nnth (((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) (Suc i))
        ) =
        (nsubn σ (nnth ls (i+n)) (nnth ls ((Suc i)+n)) )
proof -
  have 1: nlength (((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) = (nlength ls) - n
    by (simp add: assms)
  have 2: i < (nlength ls - n) ⟶
    (nnth (((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) i) =
    (nnth ls (n + i)) - (nnth ls n)
    by simp
  have 3: i < (nlength ls - n) ⟶
    (nnth (((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) (Suc i)) =
    (nnth ls (n + (Suc i))) - (nnth ls n)
    using assms by (metis eSuc-enat ileI1 ndropn-nlength ndropn-nnth nnth-nmap)
  have 4: i < (nlength ls - n) ⟶
    (nsubn (ndropn (nnth ls n) σ)
      (nnth (((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) i)
      (nnth (((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) (Suc i))
      ) =
    (nsubn (ndropn (nnth ls n) σ)
      ( (nnth ls (n + i)) - (nnth ls n) )
      ( (nnth ls (n + (Suc i))) - (nnth ls n) )
      )
    by (simp add: 2 3)
  have 5: i < (nlength ls - n) ⟶ (nnth ls (n + i)) < (nnth ls (n + (Suc i)))
    using assms
    by (metis Suc-ile-eq add.commute add-Suc-right enat-min nidx-expand plus-enat-simps(1))
  have 6: i < (nlength ls - n) ⟶ (nnth ls n) ≤ (nnth ls (n + i))
    using assms
    by (metis add.commute enat-min le-add1 nidx-less-eq order-less-imp-le plus-enat-simps(1))
  have 7: i < (nlength ls - n) ⟶ (nnth ls n) ≤ (nnth ls (n + (Suc i)))
    using 5 6 by linarith
  have 8: i < (nlength ls - n) ⟶ (nnth ls (n + i)) - (nnth ls n) < (nnth ls (n + (Suc i))) - (nnth ls n)
    using 5 6 diff-less-mono by blast
  have 10: i < (nlength ls - n) ⟶
    (nsubn (ndropn (nnth ls n) σ)
      ( (nnth ls (n + i)) - (nnth ls n) )
      ( (nnth ls (n + (Suc i))) - (nnth ls n) )
      ) =
    (nsubn σ (nnth ls (i+n)) (nnth ls ((Suc i)+n)) )
    by (metis 6 7 8 add.commute diff-add nsubn-ndropn)
  show ?thesis
  using 2 3 10 assms by auto
qed

```

lemma *powerinterval-splitb0-alt:*

assumes *nidx ls*

nnth ls 0 = 0

\neg *nfinite ls*

n ≤ nlength ls

\neg *nfinite σ*

i < (nlength ls - n)

shows $(nsubn (ndropn (nnth ls n) \sigma)$

$(nnth (((nmap (\lambda x. x - (nnth ls n)) (ndropn n ls)))) i)$

$(nnth (((nmap (\lambda x. x - (nnth ls n)) (ndropn n ls)))) (Suc i))$

$) =$

$(nsubn \sigma (nnth ls (i+n)) (nnth ls ((Suc i)+n)))$

proof –

have 1: $nlength (((nmap (\lambda x. x - (nnth ls n)) (ndropn n ls)))) = (nlength ls) - n$

by (*simp add: assms*)

have 2: $i < (nlength ls - n) \longrightarrow$

$(nnth (((nmap (\lambda x. x - (nnth ls n)) (ndropn n ls)))) i) =$

$(nnth ls (n + i)) - (nnth ls n)$

by *simp*

have 3: $i < (nlength ls - n) \longrightarrow$

$(nnth (((nmap (\lambda x. x - (nnth ls n)) (ndropn n ls)))) (Suc i)) =$

$(nnth ls (n + (Suc i))) - (nnth ls n)$

using *assms* **by** (*metis eSuc-enat ileI1 ndropn-nlength ndropn-nnth nnth-nmap*)

have 4: $i < (nlength ls - n) \longrightarrow$

$(nsubn (ndropn (nnth ls n) \sigma)$

$(nnth (((nmap (\lambda x. x - (nnth ls n)) (ndropn n ls)))) i)$

$(nnth (((nmap (\lambda x. x - (nnth ls n)) (ndropn n ls)))) (Suc i))$

$) =$

$(nsubn (ndropn (nnth ls n) \sigma)$

$((nnth ls (n + i)) - (nnth ls n))$

$((nnth ls (n + (Suc i))) - (nnth ls n))$

$)$

by (*simp add: 2 3*)

have 5: $i < (nlength ls - n) \longrightarrow (nnth ls (n + i)) < (nnth ls (n + (Suc i)))$

using *assms*

by (*metis Suc-ile-eq add.commute add-Suc-right enat-min nidx-expand plus-enat-simps(1)*)

have 6: $i < (nlength ls - n) \longrightarrow (nnth ls n) \leq (nnth ls (n + i))$

using *assms*

by (*metis add.commute enat-min le-add1 nidx-less-eq order-less-imp-le plus-enat-simps(1)*)

have 7: $i < (nlength ls - n) \longrightarrow (nnth ls n) \leq (nnth ls (n + (Suc i)))$

using 5 6 **by** *linarith*

have 8: $i < (nlength ls - n) \longrightarrow (nnth ls (n + i)) - (nnth ls n) < (nnth ls (n + (Suc i))) - (nnth ls n)$

using 5 6 *diff-less-mono* **by** *blast*

have 9: $i < (nlength ls - n) \longrightarrow (nnth ls (n + (Suc i))) - (nnth ls n) \leq nlength \sigma - (nnth ls n)$

using *assms*

by (*simp add: nfinite-conv-nlength-enat*)

have 10: $i < (nlength ls - n) \longrightarrow$

$(nsubn (ndropn (nnth ls n) \sigma)$

$((nnth ls (n + i)) - (nnth ls n))$

```

      ( (nnth ls (n + (Suc i))) - (nnth ls n) )
    ) =
      (nsubn σ (nnth ls (i+n)) (nnth ls ((Suc i)+n)) )
  by (metis 6 7 8 add commute diff-add nsubn-ndropn)
show ?thesis
using 2 3 10 assms by auto
qed

lemma powerinterval-splitb:
  assumes nidx ls
    nnth ls 0 = 0
    nfinite ls
    n ≤ nlength ls
    nfinite σ
    nlast ls = the-enat(nlength σ)
    powerinterval f σ ls
  shows powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x - (nnth ls n)) (ndropn n ls)))
proof -
  have 0: (∀ i. i < nlength ls → ( (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f))
    using assms by (simp add: powerinterval-def)
  have 1: powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x - (nnth ls n)) (ndropn n ls))) =
    (∀ i. i < nlength (((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) →
      ( (nsubn (ndropn (nnth ls n) σ)
        (nnth (((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) i)
        (nnth (((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) (Suc i))
        ) ⊨ f))
    by (simp add: powerinterval-def)
  have 2: (∀ i. i < nlength(((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) →
    ( (nsubn (ndropn (nnth ls n) σ)
      (nnth (((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) i)
      (nnth (((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) (Suc i))
      ) ⊨ f))
    proof
      fix i
      show i < nlength(((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) →
        ( (nsubn (ndropn (nnth ls n) σ)
          (nnth (((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) i)
          (nnth (((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) (Suc i))
          ) ⊨ f)
    proof -
      have 21: i < (nlength ls) - n →
        ((nsubn σ (nnth ls (i+n)) (nnth ls ((Suc i)+n)) ) ⊨ f)
        using 0 assms(4) enat-min by auto
      show ?thesis
        using 21 assms powerinterval-splitb0 by fastforce
    qed
  qed
show ?thesis using 1 2 by blast
qed

```


lemma *powerinterval-splitb-alt*:

assumes *nidx ls*

nnth ls 0 = 0

\neg *nfinite ls*

n \leq *nlength ls*

\neg *nfinite* σ

powerinterval f σ *ls*

shows *powerinterval f* (*ndropn* (*nnth ls n*) σ) ((*nmap* ($\lambda x. x - (\text{nnth } ls \ n)$) (*ndropn n ls*)))

proof –

have 0: ($\forall i. i < \text{nlength } ls \longrightarrow ((\text{nsubn } \sigma (\text{nnth } ls \ i) (\text{nnth } ls \ (\text{Suc } i))) \models f)$)

using *assms* **by** (*simp add: powerinterval-def*)

have 1: *powerinterval f* (*ndropn* (*nnth ls n*) σ) ((*nmap* ($\lambda x. x - (\text{nnth } ls \ n)$) (*ndropn n ls*))) =
 $(\forall i. i < \text{nlength } ((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) (\text{ndropn } n \ ls)))) \longrightarrow$
 $((\text{nsubn } (\text{ndropn } (\text{nnth } ls \ n) \ \sigma)$
 $(\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) (\text{ndropn } n \ ls)))) i$
 $(\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) (\text{ndropn } n \ ls)))) (\text{Suc } i))$
 $) \models f)$)

by (*simp add: powerinterval-def*)

have 2: ($\forall i. i < \text{nlength } ((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) (\text{ndropn } n \ ls)))) \longrightarrow$
 $((\text{nsubn } (\text{ndropn } (\text{nnth } ls \ n) \ \sigma)$
 $(\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) (\text{ndropn } n \ ls)))) i$
 $(\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) (\text{ndropn } n \ ls)))) (\text{Suc } i))$
 $) \models f)$)

proof

fix *i*

show *i* $< \text{nlength } ((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) (\text{ndropn } n \ ls)))) \longrightarrow$
 $((\text{nsubn } (\text{ndropn } (\text{nnth } ls \ n) \ \sigma)$
 $(\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) (\text{ndropn } n \ ls)))) i$
 $(\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) (\text{ndropn } n \ ls)))) (\text{Suc } i))$
 $) \models f)$

proof –

have 21: *i* $< (\text{nlength } ls) - n \longrightarrow$
 $((\text{nsubn } \sigma (\text{nnth } ls \ (i+n)) (\text{nnth } ls \ ((\text{Suc } i)+n))) \models f)$

using 0 *assms*(4) *enat-min* **by** *auto*

show *?thesis*

using 21 *assms* *powerinterval-splitb0-alt* **by** *fastforce*

qed

qed

show *?thesis* **using** 1 2 **by** *blast*

qed

lemma *powerinterval-split*:

assumes *nidx ls*

nnth ls 0 = 0

nfinite ls

n \leq *nlength ls*

nfinite σ

nlast ls = *the-enat*(*nlength* σ)

shows *powerinterval f* σ *ls* =

(*powerinterval f* (*ntaken* (*nnth ls n*) σ) (*ntaken n ls*) \wedge

$\text{powerinterval } f \text{ (ndropn (nnth ls n) } \sigma \text{) ((nmap } (\lambda x. x - (\text{nnth ls } n)) \text{ (ndropn } n \text{ ls)))}$
proof –
have 1: $\text{powerinterval } f \text{ } \sigma \text{ ls} \implies$
 $\text{powerinterval } f \text{ (ntaken (nnth ls n) } \sigma \text{) (ntaken } n \text{ ls)}$
by (simp add: assms powerinterval-splita)
have 2: $\text{powerinterval } f \text{ } \sigma \text{ ls} \implies$
 $\text{powerinterval } f \text{ (ndropn (nnth ls n) } \sigma \text{) ((nmap } (\lambda x. x - (\text{nnth ls } n)) \text{ (ndropn } n \text{ ls)))}$
by (simp add: assms powerinterval-splitb)
have 3: $\text{powerinterval } f \text{ (ntaken (nnth ls n) } \sigma \text{) (ntaken } n \text{ ls)} =$
 $(\forall i. i < n \longrightarrow ((\text{nsbn } \sigma \text{ (nnth ls } i) \text{ (nnth ls (Suc } i))}) \models f))$
using assms **by** (simp add: powerinterval-def powerinterval-splita0)
have 4: $\text{powerinterval } f \text{ (ndropn (nnth ls n) } \sigma \text{) ((nmap } (\lambda x. x - (\text{nnth ls } n)) \text{ (ndropn } n \text{ ls)))} =$
 $(\forall i. i < (\text{nlength ls}) - n \longrightarrow ((\text{nsbn } \sigma \text{ (nnth ls (i+n)) (nnth ls ((Suc } i) + n))}) \models f))$
unfolding powerinterval-def **using** assms powerinterval-splitb0[of ls n σ] **by** (simp)
have 5: $(\forall i. i < (\text{nlength ls}) - n \longrightarrow ((\text{nsbn } \sigma \text{ (nnth ls (i+n)) (nnth ls ((Suc } i) + n))}) \models f)) =$
 $(\forall i. n \leq i \wedge i < (\text{nlength ls}) \longrightarrow ((\text{nsbn } \sigma \text{ (nnth ls } i) \text{ (nnth ls (Suc } i))}) \models f))$
using assms **enat-min** **by** auto
 $(\text{metis diff-add diff-less-mono enat-ord-simps(2) idiff-enat-enat nfinite-nlength-enat})$
have 6: $(\text{powerinterval } f \text{ (ntaken (nnth ls n) } \sigma \text{) (ntaken } n \text{ ls)} \wedge$
 $\text{powerinterval } f \text{ (ndropn (nnth ls n) } \sigma \text{) ((nmap } (\lambda x. x - (\text{nnth ls } n)) \text{ (ndropn } n \text{ ls)))}) \implies$
 $\text{powerinterval } f \text{ } \sigma \text{ ls}$
using 3 4 5 assms powerinterval-def
by (metis not-less-eq not-less-less-Suc-eq order.order-iff-strict)
show ?thesis
using 1 2 6 **by** blast
qed

lemma powerinterval-split-alt:

assumes nidx ls
 $\text{nnth ls } 0 = 0$
 $\neg \text{nfinite ls}$
 $n \leq \text{nlength ls}$
 $\neg \text{nfinite } \sigma$
shows $\text{powerinterval } f \text{ } \sigma \text{ ls} =$
 $(\text{powerinterval } f \text{ (ntaken (nnth ls n) } \sigma \text{) (ntaken } n \text{ ls)} \wedge$
 $\text{powerinterval } f \text{ (ndropn (nnth ls n) } \sigma \text{) ((nmap } (\lambda x. x - (\text{nnth ls } n)) \text{ (ndropn } n \text{ ls)))})$

proof –

have 1: $\text{powerinterval } f \text{ } \sigma \text{ ls} \implies$
 $\text{powerinterval } f \text{ (ntaken (nnth ls n) } \sigma \text{) (ntaken } n \text{ ls)}$
by (simp add: assms powerinterval-splita-alt)
have 2: $\text{powerinterval } f \text{ } \sigma \text{ ls} \implies$
 $\text{powerinterval } f \text{ (ndropn (nnth ls n) } \sigma \text{) ((nmap } (\lambda x. x - (\text{nnth ls } n)) \text{ (ndropn } n \text{ ls)))}$
by (simp add: assms powerinterval-splitb-alt)
have 3: $\text{powerinterval } f \text{ (ntaken (nnth ls n) } \sigma \text{) (ntaken } n \text{ ls)} =$
 $(\forall i. i < n \longrightarrow ((\text{nsbn } \sigma \text{ (nnth ls } i) \text{ (nnth ls (Suc } i))}) \models f))$
using assms **by** (simp add: powerinterval-def powerinterval-splita0-alt)
have 4: $\text{powerinterval } f \text{ (ndropn (nnth ls n) } \sigma \text{) ((nmap } (\lambda x. x - (\text{nnth ls } n)) \text{ (ndropn } n \text{ ls)))} =$
 $(\forall i. i < (\text{nlength ls}) - n \longrightarrow ((\text{nsbn } \sigma \text{ (nnth ls (i+n)) (nnth ls ((Suc } i) + n))}) \models f))$
unfolding powerinterval-def **using** powerinterval-splitb0-alt[of ls n σ] **assms**
by simp

have 5: $(\forall i. i < (\text{nlength } ls) - n \longrightarrow ((\text{nsubn } \sigma (\text{nnth } ls (i+n)) (\text{nnth } ls ((\text{Suc } i)+n))) \models f)) =$
 $(\forall i. n \leq i \wedge i < (\text{nlength } ls) \longrightarrow ((\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) \models f))$
using *assms enat-min*
by *auto*
 $(\text{metis diff-add ndropn-nlength nfinite-ndropn-b nfinite-ntaken not-le-imp-less ntaken-all})$
have 6: $(\text{powerinterval } f (\text{ntaken } (\text{nnth } ls n) \sigma) (\text{ntaken } n ls) \wedge$
 $\text{powerinterval } f (\text{ndropn } (\text{nnth } ls n) \sigma) ((\text{nmap } (\lambda x. x - (\text{nnth } ls n)) (\text{ndropn } n ls)))) \implies$
 $\text{powerinterval } f \sigma ls$
using 3 4 5 *assms powerinterval-def*
by $(\text{metis not-less-eq not-less-less-Suc-eq order.order-iff-strict})$
show *?thesis*
using 1 2 6 **by** *blast*
qed

lemma *powerinterval-nfuse:*

assumes *nidx ls1*
 $\text{nnth } ls1 0 = 0$
nidx ls2
 $\text{nnth } ls2 0 = 0$
nfinite ls1
 $\text{nlast } ls1 = cp$
 $cp \leq \text{nlength } \sigma$
nfinite ls2
nfinite σ
 $\text{nlast } ls2 = \text{the-enat}(\text{nlength } \sigma) - cp$
shows $\text{powerinterval } f \sigma (\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2)) =$
 $(\text{powerinterval } f (\text{ntaken } cp \sigma) ls1 \wedge$
 $\text{powerinterval } f (\text{ndropn } cp \sigma) ls2)$

proof –

have 1: *nidx* $(\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2))$
using *assms nidx-nfuse[of ls1 (nmap (λx. x + cp) ls2)]*
using *Suc-ile-eq nidx-expand zero-enat-def* **by** *force*
have 2: $cp = (\text{nnth } ls1 (\text{the-enat } (\text{nlength } ls1)))$
using *assms using nnth-nlast* **by** *blast*
have 3: $\text{nlength } ls1 \leq \text{nlength } (\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2))$
by $(\text{simp add: nfuse-nlength})$
have 30: *nfinite* $(\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2))$
using *assms*
by $(\text{metis nfinite-conv-nlength-enat nfuse-nlength nlength-nmap plus-enat-simps}(1))$
have 31: $\text{nfirst } (\text{nmap } (\lambda x. x + cp) ls2) = \text{nnth } (\text{nmap } (\lambda x. x + cp) ls2) 0$
by $(\text{metis ndropn-0 ndropn-nfirst})$
have 32: $\text{nnth } (\text{nmap } (\lambda x. x + cp) ls2) 0 = cp$
using *assms nnth-nmap[of 0 ls2 (λx. x + cp)]*
using *zero-enat-def* **by** *auto*
have 33: $\text{nlast } ls1 = \text{nfirst } (\text{nmap } (\lambda x. x + cp) ls2)$
by $(\text{simp add: 31 32 assms}(6))$
have 34: $\text{nlast } (\text{nmap } (\lambda x. x + cp) ls2) = \text{the-enat}(\text{nlength } \sigma)$
using *assms*
by $(\text{metis diff-add enat.simps}(3) \text{ enat-ord-simps}(1) \text{ enat-the-enat nfinite-conv-nlength-enat nlast-nmap})$
have 4: $\text{nlast } (\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2)) = \text{the-enat}(\text{nlength } \sigma)$

```

using assms using 30 assms nlast-nfuse[of ls1 (nmap ( $\lambda x. x + cp$ ) ls2)] 33 34 by presburger
have 5: cp = nnth (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)) (the-enat (nlength ls1))
using assms 2
by (metis enat.simps(3) enat-the-enat nfinite-conv-nlength-enat nfuse-def1 nnth-nappend
    not-le-imp-less order-less-imp-le)
have 50: nlength (nmap ( $\lambda x. x - cp$ ) (nmap ( $\lambda x. x + cp$ ) ls2)) = nlength ls2
by simp
have 51:  $\bigwedge j. j \leq \text{nlength } ls2 \longrightarrow$ 
    nnth (nmap ( $\lambda x. x - cp$ ) (nmap ( $\lambda x. x + cp$ ) ls2)) j = nnth ls2 j
by simp
have 6: (nmap ( $\lambda x. x - cp$ ) (nmap ( $\lambda x. x + cp$ ) ls2)) = ls2
using 50 51 by (simp add: nellist-eq-nnth-eq)
have 70: ntaken (the-enat (nlength ls1)) (nfuse ls1 (nmap ( $\lambda x::nat. x + cp$ ) ls2)) = ls1
by (simp add: 33 assms(5) ntaken-nfuse)
have 71: ndropn (the-enat (nlength ls1)) (nfuse ls1 (nmap ( $\lambda x::nat. x + cp$ ) ls2)) =
    nmap ( $\lambda x::nat. x + cp$ ) ls2
by (simp add: 33 assms(5) ndropn-nfuse)
have 72: nidx (nfuse (ls1::nat nellist) (nmap ( $\lambda x::nat. x + (cp::nat)$ ) (ls2::nat nellist)))
by (simp add: 1)
have 73: nnth (nfuse ls1 (nmap ( $\lambda x::nat. x + cp$ ) ls2)) (0::nat) = (0::nat)
by (metis 33 assms(2) nfuse-nnth zero-enat-def zero-le)
have 74: enat (the-enat (nlength ls1))  $\leq$  nlength (nfuse ls1 (nmap ( $\lambda x::nat. x + cp$ ) ls2))
by (metis 70 assms(5) enat.simps(3) enat-the-enat min-def nfinite-nlength-enat ntaken-nlength
    order-eq-refl)
have 75: enat (nlast (nfuse ls1 (nmap ( $\lambda x::nat. x + cp$ ) ls2))) = nlength  $\sigma$ 
using 4 assms enat-the-enat nfinite-conv-nlength-enat by auto
have 76: nfinite  $\sigma$ 
by (simp add: assms(9))
have 80: powerinterval f  $\sigma$  (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)) =
    (powerinterval f (ntaken (nnth (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)) (the-enat (nlength ls1)))  $\sigma$ )
    (ntaken (the-enat (nlength ls1)) (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)))  $\wedge$ 
    powerinterval f (ndropn (nnth (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)) (the-enat (nlength ls1)))  $\sigma$ )
    (nmap ( $\lambda x. x - \text{nnth} (nfuse\ ls1\ (nmap\ (\lambda x.\ x + cp)\ ls2))\ (the-enat\ (nlength\ ls1))$ )
    (ndropn (the-enat (nlength ls1)) (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2))))))
using 72 73 30 74 76 75 4 using powerinterval-split[of (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2))
    (the-enat (nlength ls1))  $\sigma$  f]
by fastforce
have 81: powerinterval f (ntaken (nnth (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)) (the-enat (nlength ls1)))  $\sigma$ )
    (ntaken (the-enat (nlength ls1)) (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2))) =
    powerinterval f (ntaken cp  $\sigma$ ) ls1
using 5 70 by auto
have 82: powerinterval f (ndropn (nnth (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)) (the-enat (nlength ls1)))  $\sigma$ )
    (nmap ( $\lambda x. x - \text{nnth} (nfuse\ ls1\ (nmap\ (\lambda x.\ x + cp)\ ls2))\ (the-enat\ (nlength\ ls1))$ )
    (ndropn (the-enat (nlength ls1)) (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)))) =
    powerinterval f (ndropn cp  $\sigma$ ) ls2
using 5 6 71 by presburger
show ?thesis
using 80 81 82 by blast
qed

```

lemma *powerinterval-nfuse-alt:*

assumes *nidx ls1*

nnth ls1 0 = 0

nidx ls2

nnth ls2 0 = 0

nfinite ls1

nlast ls1 = cp

cp ≤ nlength σ

\neg *nfinite ls2*

\neg *nfinite σ*

shows *powerinterval f σ (nfuse ls1 (nmap (λx. x+ cp) ls2)) =*

(powerinterval f (ntaken cp σ) ls1 ∧

powerinterval f (ndropn cp σ) ls2)

proof –

have 1: *nidx (nfuse ls1 (nmap (λx. x+ cp) ls2))*

using *assms nidx-nfuse[of ls1 (nmap (λx. x+ cp) ls2)]*

using *Suc-ile-eq nidx-expand zero-enat-def* **by** *force*

have 2: *cp = (nnth ls1 (the-enat (nlength ls1)))*

using *assms using nnth-nlast* **by** *blast*

have 3: *nlength ls1 ≤ nlength (nfuse ls1 (nmap (λx. x+ cp) ls2))*

by *(simp add: nfuse-nlength)*

have 30: \neg *nfinite (nfuse ls1 (nmap (λx. x+ cp) ls2))*

by *(metis assms(8) enat-ile enat-le-plus-same(2) nfinite-conv-nlength-enat nfuse-nlength nlength-nmap)*

have 5: *cp = nnth (nfuse ls1 (nmap (λx. x+ cp) ls2)) (the-enat (nlength ls1))*

by *(metis 2 assms(5) enat.simps(3) enat-the-enat nfinite-conv-nlength-enat nfuse-def1 nnth-nappend not-le-imp-less order-less-imp-le)*

have 50: *nlength (nmap (λx. x- cp) (nmap (λx. x+ cp) ls2)) = nlength ls2*

by *simp*

have 51: $\bigwedge j. j \leq nlength ls2 \longrightarrow$

nnth (nmap (λx. x- cp) (nmap (λx. x+ cp) ls2)) j = nnth ls2 j

by *simp*

have 6: *(nmap (λx. x- cp) (nmap (λx. x+ cp) ls2)) = ls2*

using 50 51 **by** *(simp add: nellist-eq-nnth-eq)*

have 7: *nlast ls1 = nfirst (nmap (λx. x+ cp) ls2)*

by *(metis add-0 assms(4) assms(6) nfinite-conv-nlength-enat nfinite-ntaken nlast-NNil nlength-NNil nnth-nmap ntaken-0 ntaken-nlast the-enat.simps the-enat-0 zero-le)*

have 70: *ntaken (the-enat (nlength ls1)) (nfuse ls1 (nmap (λx::nat. x + cp) ls2)) = ls1*

by *(simp add: 7 assms(5) ntaken-nfuse)*

have 71: *ndropn (the-enat (nlength ls1)) (nfuse ls1 (nmap (λx::nat. x + cp) ls2)) =*

nmap (λx::nat. x + cp) ls2

by *(simp add: 7 assms(5) ndropn-nfuse)*

have 72: *nidx (nfuse (ls1::nat nellist) (nmap (λx::nat. x + (cp::nat)) (ls2::nat nellist)))*

by *(simp add: 1)*

have 73: *nnth (nfuse ls1 (nmap (λx::nat. x + cp) ls2)) (0::nat) = (0::nat)*

by *(metis 7 assms(2) nfuse-nnth zero-enat-def zero-le)*

have 74: *enat (the-enat (nlength ls1)) ≤ nlength (nfuse ls1 (nmap (λx. x + cp) ls2))*

by *(metis 70 assms(5) enat.simps(3) enat-the-enat min-def nfinite-nlength-enat ntaken-nlength order-eq-refl)*

have 76: \neg *nfinite σ*

```

by (simp add: assms(9))
have 80: powerinterval f  $\sigma$  (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)) =
  (powerinterval f (ntaken (nnth (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)) (the-enat (nlength ls1)))  $\sigma$ )
    (ntaken (the-enat (nlength ls1)) (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)))  $\wedge$ 
  powerinterval f (ndropn (nnth (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)) (the-enat (nlength ls1)))  $\sigma$ )
    (nmap ( $\lambda x::nat. x - nnth$  (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)) (the-enat (nlength ls1)))
      (ndropn (the-enat (nlength ls1)) (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)))))
using 72 73 30 74 76
using powerinterval-split-alt[of (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2))
  (the-enat (nlength ls1))  $\sigma$  f]
by blast
have 81: powerinterval f (ntaken (nnth (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)) (the-enat (nlength ls1)))  $\sigma$ )
  (ntaken (the-enat (nlength ls1)) (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2))) =
  powerinterval f (ntaken cp  $\sigma$ ) ls1
using 5 70 by auto
have 82: powerinterval f (ndropn (nnth (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)) (the-enat (nlength ls1)))  $\sigma$ )
  (nmap ( $\lambda x. x - nnth$  (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)) (the-enat (nlength ls1)))
    (ndropn (the-enat (nlength ls1)) (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2))))) =
  powerinterval f (ndropn cp  $\sigma$ ) ls2
using 5 6 71 by presburger
show ?thesis
using 80 81 82 by blast
qed

```

cppl lemmas

lemma *cppl-expand*:

```

assumes ( $\exists$  ls. nidx ls  $\wedge$  nnth ls 0 = 0  $\wedge$  powerinterval f  $\sigma$  ls  $\wedge$ 
  ((nfinite ls  $\wedge$  nfinite  $\sigma$   $\wedge$  nlast ls = the-enat(nlength  $\sigma$ ))  $\vee$ 
  ( $\neg$ nfinite ls  $\wedge$   $\neg$ nfinite  $\sigma$ ))  $\wedge$ 
  ((pfilt  $\sigma$  ls)  $\models$  g ))
shows nidx (cppl f g  $\sigma$ )  $\wedge$  nnth (cppl f g  $\sigma$ ) 0 = 0  $\wedge$  powerinterval f  $\sigma$  (cppl f g  $\sigma$ )  $\wedge$ 
  ((nfinite (cppl f g  $\sigma$ )  $\wedge$  nfinite  $\sigma$   $\wedge$  nlast (cppl f g  $\sigma$ ) = the-enat(nlength  $\sigma$ ))  $\vee$ 
  ( $\neg$ nfinite (cppl f g  $\sigma$ )  $\wedge$   $\neg$ nfinite  $\sigma$ ))  $\wedge$  ((pfilt  $\sigma$  (cppl f g  $\sigma$ ))  $\models$  g)
proof -
  have 0: cppl f g  $\sigma$  = ( $\epsilon$  ls. nidx ls  $\wedge$  nnth ls 0 = 0  $\wedge$  powerinterval f  $\sigma$  ls  $\wedge$ 
    ((nfinite ls  $\wedge$  nfinite  $\sigma$   $\wedge$  nlast ls = the-enat(nlength  $\sigma$ ))  $\vee$ 
    ( $\neg$ nfinite ls  $\wedge$   $\neg$ nfinite  $\sigma$ ))  $\wedge$  ((pfilt  $\sigma$  ls)  $\models$  g))
  using cppl-def by blast
  have 1: ( $\exists$  ls. nidx ls  $\wedge$  nnth ls 0 = 0  $\wedge$  powerinterval f  $\sigma$  ls  $\wedge$ 
    ((nfinite ls  $\wedge$  nfinite  $\sigma$   $\wedge$  nlast ls = the-enat(nlength  $\sigma$ ))  $\vee$ 
    ( $\neg$ nfinite ls  $\wedge$   $\neg$ nfinite  $\sigma$ ))  $\wedge$  ((pfilt  $\sigma$  ls)  $\models$  g))
  using assms by auto
  have 2: nidx (cppl f g  $\sigma$ )  $\wedge$  nnth (cppl f g  $\sigma$ ) 0 = 0  $\wedge$  powerinterval f  $\sigma$  (cppl f g  $\sigma$ )  $\wedge$ 
    ((nfinite (cppl f g  $\sigma$ )  $\wedge$  nfinite  $\sigma$   $\wedge$  nlast (cppl f g  $\sigma$ ) = the-enat(nlength  $\sigma$ ))  $\vee$ 
    ( $\neg$ nfinite (cppl f g  $\sigma$ )  $\wedge$   $\neg$ nfinite  $\sigma$ ))  $\wedge$  ((pfilt  $\sigma$  (cppl f g  $\sigma$ ))  $\models$  g)
  using 0 1
  someI-ex[of  $\lambda$ ls. nidx ls  $\wedge$  nnth ls 0 = 0  $\wedge$  powerinterval f  $\sigma$  ls  $\wedge$ 
    ((nfinite ls  $\wedge$  nfinite  $\sigma$   $\wedge$  nlast ls = the-enat(nlength  $\sigma$ ))  $\vee$ 
    ( $\neg$ nfinite ls  $\wedge$   $\neg$ nfinite  $\sigma$ ))  $\wedge$ 

```

$((pfilt\ \sigma\ ls) \models g)]$ **by** *simp*
show *?thesis*
using 2 **by** *blast*
qed

lemma *cppl-fprojection*:

$(\sigma \models f\ fproj\ g) =$
 $(\text{nid}x\ (cppl\ f\ g\ \sigma) \wedge \text{nnth}\ (cppl\ f\ g\ \sigma)\ 0 = 0 \wedge \text{powerinterval}\ f\ \sigma\ (cppl\ f\ g\ \sigma) \wedge$
 $\text{nfinite}\ (cppl\ f\ g\ \sigma) \wedge \text{nfinite}\ \sigma \wedge$
 $\text{nlast}\ (cppl\ f\ g\ \sigma) = \text{the-enat}(\text{nlength}\ \sigma) \wedge g\ (pfilt\ \sigma\ (cppl\ f\ g\ \sigma)))$
using *cppl-expand*[*of f σ g*]
by (*simp add: fprojection-d-def*)
 $(metis\ \text{nfinite-conv-nlength-enat}\ \text{the-enat.simps})$

lemma *cppl-oprojection*:

$(\sigma \models f\ oproj\ g) =$
 $(\text{nid}x\ (cppl\ f\ g\ \sigma) \wedge \text{nnth}\ (cppl\ f\ g\ \sigma)\ 0 = 0 \wedge \text{powerinterval}\ f\ \sigma\ (cppl\ f\ g\ \sigma) \wedge$
 $\neg \text{nfinite}\ (cppl\ f\ g\ \sigma) \wedge \neg \text{nfinite}\ \sigma \wedge g\ (pfilt\ \sigma\ (cppl\ f\ g\ \sigma)))$
using *cppl-expand* **by** (*simp add: oprojection-d-def, blast*)

lemma *cppl-empty*:

assumes $\text{nlength}\ \sigma = 0$
 $(\sigma \models f\ fproj\ g)$
shows $(cppl\ f\ g\ \sigma) = (NNil\ 0)$
using *assms cppl-fprojection*[*of f g σ*]
by (*metis gr-zeroI less-numeral-extra(3) ndropn-0 ndropn-nlast nfinite-conv-nlength-enat*
 $\text{nid}x\text{-less-last-1}\ \text{nnth-nlast}\ \text{the-enat.simps}\ \text{the-enat-0}$)

lemma *cppl-empty-a*:

assumes $\text{nlength}\ \sigma = 0$
 $(cppl\ f\ g\ \sigma) = NNil\ 0$
 $g(pfilt\ \sigma\ (NNil\ 0))$
shows $(\sigma \models f\ fproj\ g)$
proof –
have 1: $\text{nid}x\ (NNil\ 0)$
by (*simp add: nidx-def*)
have 10: $\text{nnth}\ (NNil\ 0)\ 0 = 0$
by (*simp add: nnth-NNil*)
have 11: $\text{nfinite}\ (NNil\ 0)$
by *simp*
have 12: $\text{nfinite}\ \sigma$
by (*simp add: assms(1) nlength-eq-enat-nfiniteD zero-enat-def*)
have 2: $\text{powerinterval}\ f\ \sigma\ (NNil\ 0)$
by (*simp add: powerinterval-def*)
have 3: $\text{nlast}\ (NNil\ 0) = \text{nlength}\ \sigma$
by (*simp add: assms*)
have 4: $g(pfilt\ \sigma\ (NNil\ 0))$
using *assms* **by** *blast*

```

from 1 10 11 12 2 3 4 show ?thesis using assms
by (simp add: cppl-fprojection zero-enat-def)
qed

```

```

lemma cppl-more:
  assumes nlength  $\sigma > 0$ 
    ( $\sigma \models f \text{ fproj } g$ )
  shows nlength(cppl f g  $\sigma$ )  $> 0$ 
using assms
by (metis cppl-fprojection enat-add-sub-same gen-nlength-def gr-zeroI i0-ne-infinity ndropn-nlast
    ndropn-nlength nlength-NNil nlength-code nnth-nlast the-enat-0 zero-enat-def)

```

```

lemma cppl-more-infinite:
  assumes nlength  $\sigma > 0$ 
    ( $\sigma \models f \text{ oproj } g$ )
  shows nlength(cppl f g  $\sigma$ )  $> 0$ 
using assms
by (simp add: cppl-oprojection nfinite-conv-nlength-enat)

```

```

lemma cppl-more-than-first:
  assumes nlength  $\sigma > 0$ 
    ( $\sigma \models f \text{ fproj } g$ )
  shows (nnth (cppl f g  $\sigma$ ) 0) = 0
using assms
using cppl-fprojection by blast

```

```

lemma cppl-more-than-first-alt:
  assumes nlength  $\sigma > 0$ 
    ( $\sigma \models f \text{ oproj } g$ )
  shows (nnth (cppl f g  $\sigma$ ) 0) = 0
using assms
using cppl-oprojection by blast

```

```

lemma cppl-more-than-last:
  assumes nlength  $\sigma > 0$ 
    ( $\sigma \models f \text{ fproj } g$ )
  shows nlast (cppl f g  $\sigma$ ) = the-enat(nlength  $\sigma$ )
using assms cppl-fprojection by blast

```

```

lemma cppl-sub-more:
  assumes  $n < k$ 
     $k \leq \text{nlength } \sigma$ 
    ( $(\text{nsubn } \sigma \text{ } n \text{ } k) \models f \text{ fproj } g$ )
  shows nlength(cppl f g (nsubn  $\sigma$  n k))  $> 0$ 
using assms cppl-fprojection[of f g (nsubn  $\sigma$  n k)]
using cppl-more nsubn-nlength-gr-one by blast

```

```

lemma cppl-bounds:

```


assumes $n < k$
 $k \leq \text{nlength } \sigma$
 $((\text{nsubn } \sigma \ n \ k) \models f \text{ fproj } g)$
 $i < \text{nlength } (\text{cppl } f \ g \ (\text{nsubn } \sigma \ n \ k))$
shows $0 \leq (\text{nnth } (\text{cppl } f \ g \ (\text{nsubn } \sigma \ n \ k)) \ i) \wedge (\text{nnth } (\text{cppl } f \ g \ (\text{nsubn } \sigma \ n \ k)) \ i) \leq k - n$
using *assms*
using *cppl-fprojection*[of $f \ g \ (\text{nsubn } \sigma \ n \ k)$] **using** *nsubn-nlength*[of $\sigma \ n \ k$]
by *simp*
(metis enat-minus-mono1 idiff-enat-enat min-def nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast order-less-imp-le the-enat.simps)

lemma *cppl-nmap-bounds*:

assumes $n < k$
 $k \leq \text{nlength } \sigma$
 $((\text{nsubn } \sigma \ n \ k) \models f \text{ fproj } g)$
 $i < \text{nlength } (\text{nmap } (\lambda x. x + n) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ n \ k)))$
shows $n \leq (\text{nnth } (\text{nmap } (\lambda x. x + n) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ n \ k))) \ i) \wedge$
 $(\text{nnth } (\text{nmap } (\lambda x. x + n) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ n \ k))) \ i) \leq k$
using *assms*
using *cppl-bounds* **by** *fastforce*

lemma *cppl-nfirst*:

assumes $(\text{nsubn } \sigma \ x1a \ (\text{nfirst } ls)) \models f \text{ fproj } g$
shows $\text{nfirst}((\text{nmap } (\lambda x. x + x1a) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ x1a \ (\text{nfirst } ls)))) = x1a$

proof –

have 1: $(\text{nidx } (\text{cppl } f \ g \ (\text{nsubn } \sigma \ x1a \ (\text{nfirst } ls))) \wedge$
 $\text{nnth } (\text{cppl } f \ g \ (\text{nsubn } \sigma \ x1a \ (\text{nfirst } ls))) \ 0 = 0 \wedge$
 $\text{powerinterval } f \ (\text{nsubn } \sigma \ x1a \ (\text{nfirst } ls)) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ x1a \ (\text{nfirst } ls))) \wedge$
 $\text{nfinite } (\text{cppl } f \ g \ (\text{nsubn } \sigma \ x1a \ (\text{nfirst } ls))) \wedge \text{nfinite } (\text{nsubn } \sigma \ x1a \ (\text{nfirst } ls)) \wedge$
 $\text{nlast } (\text{cppl } f \ g \ (\text{nsubn } \sigma \ x1a \ (\text{nfirst } ls))) =$
 $\text{the-enat}(\text{nlength } (\text{nsubn } \sigma \ x1a \ (\text{nfirst } ls))) \wedge$
 $g \ (\text{pfilt } (\text{nsubn } \sigma \ x1a \ (\text{nfirst } ls)) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ x1a \ (\text{nfirst } ls))))$
using *cppl-fprojection* *assms* **by** *auto*

have 3: $(\text{nnth } (\text{cppl } f \ g \ (\text{nsubn } \sigma \ x1a \ (\text{nfirst } ls))) \ 0) = 0$

by *(simp add: 1)*

show *?thesis*

by *(metis 3 add-cancel-right-left nfirst-eq-nnth-zero nnth-nmap zero-enat-def zero-le)*

qed

lemma *cppl-nfirst-same*:

assumes $(\text{nsubn } \sigma \ x1a \ x1a) \models f \text{ fproj } g$
shows $\text{nfirst}((\text{nmap } (\lambda x. x + x1a) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ x1a \ x1a)))) = x1a$

proof –

have 1: $\text{nfirst } (\text{NNil } x1a) = x1a$

by *(metis NNil-eq-ntake-iff nellist.inject(1))*

from 1 *cppl-nfirst* **show** *?thesis*

by *(metis assms)*

qed

lemma *cppl-nlast*:

assumes $((\text{nsbn } \sigma \ x \ (\text{nfirst } ls)) \models f \ \text{fproj } g)$
 $x < \text{nfirst } ls$
 $\text{nfirst } ls \leq \text{nlength } \sigma$
shows $\text{nlast}((\text{nmap } (\lambda y. y + x) (\text{cppl } f \ g \ (\text{nsbn } \sigma \ x \ (\text{nfirst } ls)))) = \text{nfirst } ls$
proof –
have 01: $(\text{nidx } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ x \ (\text{nfirst } ls))) \wedge$
 $\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ x \ (\text{nfirst } ls))) \ 0 = 0 \wedge$
 $\text{powerinterval } f \ (\text{nsbn } \sigma \ x \ (\text{nfirst } ls)) \ (\text{cppl } f \ g \ (\text{nsbn } \sigma \ x \ (\text{nfirst } ls))) \wedge$
 $\text{nfinite } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ x \ (\text{nfirst } ls))) \wedge \text{nfinite } (\text{nsbn } \sigma \ x \ (\text{nfirst } ls)) \wedge$
 $\text{nlast } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ x \ (\text{nfirst } ls))) = \text{the-enat}(\text{nlength } (\text{nsbn } \sigma \ x \ (\text{nfirst } ls))) \wedge$
 $g \ (\text{pfilt } (\text{nsbn } \sigma \ x \ (\text{nfirst } ls)) (\text{cppl } f \ g \ (\text{nsbn } \sigma \ x \ (\text{nfirst } ls))))$
using *cppl-fprojection* **assms** **by** (*simp add: cppl-fprojection*)
have 02: $\text{nlast } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ x \ (\text{nfirst } ls))) =$
 $\text{the-enat}(\text{nlength } (\text{nsbn } \sigma \ x \ (\text{nfirst } ls)))$
using 01 **by** *auto*
have 04: $x < \text{nfirst } ls$
using *assms* **by** *blast*
have 05: $\text{nlength } (\text{nsbn } \sigma \ x \ (\text{nfirst } ls)) = (\text{nfirst } ls) - x$
using *assms* **by** (*metis enat-minus-mono1 idiff-enat-enat min.orderE nsbn-nlength*)
have 06: $\text{nlast}((\text{nmap } (\lambda y. y + x) (\text{cppl } f \ g \ (\text{nsbn } \sigma \ x \ (\text{nfirst } ls)))) =$
 $((\text{nfirst } ls) - x) + x$
using 01 05 **by** *auto*
show ?thesis **using** 04 06 **by** *auto*
qed

lemma *cppl-nlast-i*:

assumes $((\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) \models f \ \text{fproj } g)$
 $(\text{nnth } ls \ i) < (\text{nnth } ls \ (\text{Suc } i))$
 $(\text{nnth } ls \ (\text{Suc } i)) \leq \text{nlength } \sigma$
shows $\text{nlast}((\text{nmap } (\lambda x. x + (\text{nnth } ls \ i)) (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) =$
 $(\text{nnth } ls \ (\text{Suc } i))$
proof –
have 01: $(\text{nidx } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \wedge$
 $\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \ 0 = 0 \wedge$
 $\text{powerinterval } f \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))$
 $(\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \wedge$
 $\text{nfinite } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \wedge$
 $\text{nfinite } (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) \wedge$
 $\text{nlast } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) =$
 $\text{the-enat}(\text{nlength } (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \wedge$
 $g \ (\text{pfilt } (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))$
 $(\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))))$
using *assms* **by** (*simp add: cppl-fprojection*)
have 02: $\text{nlast } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) =$
 $\text{the-enat}(\text{nlength } (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))))$
using 01 **by** *auto*
have 04: $(\text{nnth } ls \ i) < (\text{nnth } ls \ (\text{Suc } i))$
by (*simp add: assms*)
have 05: $\text{nlength } (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) = (\text{nnth } ls \ (\text{Suc } i)) - (\text{nnth } ls \ i)$
by (*metis assms(3) enat-minus-mono1 idiff-enat-enat min.orderE nsbn-nlength*)

```

have 06: nlast((nmap (λx. x+ (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))) =
  ((nnth ls (Suc i)) - (nnth ls i)) + (nnth ls i)
  using 01 05 by auto
show ?thesis using 04 06 by auto
qed

```

lcppl lemmas

lemma lcppl-nnth:

```

assumes nidx ls
        i < nlength ls
shows   (nnth (lcppl f g σ ls) i) =
        (nmap (λx .x+ (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )))
using assms
proof (induct i arbitrary: ls)
case 0
then show ?case
  proof (cases ls)
  case (NNil x1)
  then show ?thesis
  using 0.premis(2) by auto
  next
  case (NCons x21 x22)
  then show ?thesis using NCons
  proof (cases is-NNil x22)
  case True
  then show ?thesis using NCons by simp
  (metis is-NNil-def lcppl-code(2) nnth-NNil)
  next
  case False
  then show ?thesis using NCons by simp
  (metis lcppl-code(3) nellist.collapse(2) nnth-0)
  qed
qed
next
case (Suc i)
then show ?case
  proof (cases ls)
  case (NNil x1)
  then show ?thesis
  using Suc.premis(2) by auto
  next
  case (NCons x21 x22)
  then show ?thesis
  proof (cases is-NNil x22)
  case True
  then show ?thesis
  by (metis NCons Suc.premis(2) enat-0-iff(1) ile0-eq iless-Suc-eq nat.simps(3) nellist.collapse(1)
      nlength-NCons nlength-NNil)
  next

```

```

case False
then show ?thesis using NCons by simp
  (metis (no-types, lifting) Suc(1) Suc.prems(1) Suc.prems(2) Suc-ile-eq iless-Suc-eq
    lcppl-code(3) nellist.collapse(2) nellist.sel(5) nidx-expand nlength-NCons nnth-ntl)
qed
qed
qed

lemma lcppl-nlength:
assumes nfinite ls
  nidx ls
  nlength ls > 0
shows nlength(lcppl f g σ ls) = (epred (nlength ls))
using assms
proof (induct ls rule: nfinite-induct)
case (NNil y)
then show ?case by simp
next
case (NCons x ls)
then show ?case
  proof (cases is-NNil ls)
  case True
  then show ?thesis using NCons
  by (metis co.enat.sel(2) lcppl-code(2) nellist.collapse(1) nlength-NCons nlength-NNil)
  next
  case False
  then show ?thesis using NCons unfolding nidx-expand by simp
  (metis NCons.prems(1) co.enat.exhaust-sel lcppl-code(3) ndropn-0 ndropn-nlast nellist.collapse(2)
    nellist.disc(1) nidx-LCons-1 nidx-expand nlength-NCons the-enat-0)
  qed
qed

lemma lcppl-nfinite:
assumes nidx ls
shows nfinite ls ⟷ nfinite(lcppl f g σ ls) (is ?lhs ⟷ ?rhs)
proof
assume a: ?lhs
show ?rhs
  using a assms
  proof (induct ls rule: nfinite-induct)
  case (NNil y)
  then show ?case by simp
  next
  case (NCons x ls)
  then show ?case
  proof (cases is-NNil ls)
  case True
  then show ?thesis using NCons
  by simp

```

```

    next
    case False
    then show ?thesis using NCons by simp
      (metis lcppl-code(3) nellist.collapse(2) nfinite-NConsI)
    qed
  qed
next
assume b: ?rhs
show ?lhs
using b assms
proof (induct zs≡(lcppl f g  $\sigma$  ls) arbitrary: ls rule: nfinite-induct)
case (NNil y)
then show ?case
by (metis is-NNil-imp-nfinite lcppl.disc(2) nellist.disc(1) nfinite-ntl)
next
case (NCons x ls1)
then show ?case
proof (cases is-NNil ls1)
case True
then show ?thesis using NCons
by (metis is-NNil-imp-nfinite lcppl-code(3) nellist.collapse(2) nellist.disc(2) nellist.sel(5)
    nfinite-ntl)
next
case False
then show ?thesis using NCons unfolding nidx-expand
by (metis NCons.hyps(2) NCons.premis is-NNil-imp-nfinite lcppl-code(3) nellist.collapse(2)
    nellist.inject(2) nfinite-ntl nidx-LCons-1)
qed
qed
qed

```

```

lemma lcppl-nlength-alt:
  assumes  $\neg$ nfinite ls
    nidx ls
  shows nlength(lcppl f g  $\sigma$  ls) = (epred (nlength ls))
proof -
  have 1:  $\neg$  nfinite (lcppl f g  $\sigma$  ls)
    using assms(1) assms(2) lcppl-nfinite by blast
  have 2: epred (nlength ls) =  $\infty$ 
    using assms
    by (metis enat2-cases epred-Infty nlength-eq-enat-nfiniteD)
  have 3: nlength (lcppl f g  $\sigma$  ls) =  $\infty$ 
    by (meson 1 enat2-cases nlength-eq-enat-nfiniteD)
  show ?thesis
    by (simp add: 2 3)
qed

```

```

lemma lcppl-nlength-zero:

```

assumes $nidx\ ls$
 $nlength\ ls = 0$
shows $nlength(lcppl\ f\ g\ \sigma\ ls) = 0$
using $assms$
by ($metis\ (no-types,\ lifting)\ is-NNil-def\ lcppl.ctr(1)\ le-numeral-extra(3)\ nlength-NNil$
 $ntaken-0\ ntaken-all\ zero-enat-def$)

lemma $lcppl-nlast$:

assumes $nidx\ ls$
 $nfinite\ ls$
 $nfinite\ \sigma$
 $nlast\ ls = the-enat(nlength\ \sigma)$
 $nlength\ ls > 0$
shows $nlast\ (lcppl\ f\ g\ \sigma\ ls) =$
 $(nmap\ (\lambda x. x + (nnth\ ls\ (the-enat\ (epred(nlength\ ls)))))$
 $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ (the-enat\ (epred\ (nlength\ ls)))))$
 $(nnth\ ls\ (the-enat\ (nlength\ ls)))\)))$

proof –

have 1: $nlast\ (lcppl\ f\ g\ \sigma\ ls) = (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ ((the-enat\ (nlength\ (lcppl\ f\ g\ \sigma\ ls)))))$
using $assms\ lcppl-nfinite\ nnth-nlast\ by\ blast$
have 2: $nlast\ (lcppl\ f\ g\ \sigma\ ls) =$
 $(nmap\ (\lambda x. x + (nnth\ ls\ ((the-enat\ (nlength\ (lcppl\ f\ g\ \sigma\ ls)))))$
 $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ ((the-enat\ (nlength\ (lcppl\ f\ g\ \sigma\ ls)))))$
 $(nnth\ ls\ (Suc\ (the-enat\ (nlength\ (lcppl\ f\ g\ \sigma\ ls)))))$
 $)))$
using $assms\ lcppl-nnth[of\ ls\ ((the-enat\ (nlength\ (lcppl\ f\ g\ \sigma\ ls)))]\ f\ g\ \sigma]$
 $lcppl-nlength[of\ ls\ f\ g\ \sigma]$
by ($metis\ 1\ co.enat.exhaust-sel\ eSuc-enat\ enat-ord-simps(2)\ lcppl-nfinite\ less-add-same-cancel2$
 $nfinite-nlength-enat\ plus-1-eq-Suc\ the-enat.simps\ zero-less-Suc\ zero-less-iff-neq-zero$)
have 3: $(nmap\ (\lambda x. x + (nnth\ ls\ (the-enat\ (nlength\ (lcppl\ f\ g\ \sigma\ ls)))))$
 $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ ((the-enat\ (nlength\ (lcppl\ f\ g\ \sigma\ ls)))))$
 $(nnth\ ls\ (Suc\ (the-enat\ (nlength\ (lcppl\ f\ g\ \sigma\ ls)))))$
 $))) =$
 $(nmap\ (\lambda x. x + (nnth\ ls\ ((the-enat\ (epred(nlength\ ls)))))$
 $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ ((the-enat\ (epred\ (nlength\ ls)))))$
 $(nnth\ ls\ (Suc\ (the-enat\ (epred(nlength\ ls)))))\)))$
using $assms\ lcppl-nlength$
by ($metis\ (no-types,\ lifting)\ nellist.map-cong0$)
have 4: $(Suc\ (the-enat\ (epred(nlength\ ls)))) = (the-enat\ (nlength\ ls))$
by ($metis\ assms(2)\ assms(5)\ co.enat.exhaust-sel\ enat.distinct(2)\ epred-Infty\ infinity-ne-i0$
 $nfinite-conv-nlength-enat\ not-gr-zero\ the-enat-eSuc$)
show $?thesis$
using 1 2 3 4 **by** $presburger$
qed

lemma $lcppl-nlast-nlast$:

assumes $nidx\ ls$
 $nfinite\ ls$
 $nfinite\ \sigma$
 $nlast\ ls = the-enat(nlength\ \sigma)$

$((\text{nsbn } \sigma \text{ (nnth } ls \text{ (the-enat (epred(nlength } ls)))) \text{ (nnth } ls \text{ (the-enat(nlength } ls)))) \models f \text{ fproj } g)$
 $nlength \text{ } ls > 0$
shows $nlast \text{ (lcppl } f \text{ } g \text{ } \sigma \text{ } ls) = the-enat(nlength \text{ } \sigma)$
proof –
have 2: $(nlast \text{ (lcppl } f \text{ } g \text{ } \sigma \text{ } ls) =$
 $(nmap \text{ (}\lambda x. x + \text{ (nnth } ls \text{ (the-enat (epred(nlength } ls))))$
 $(cppl \text{ } f \text{ } g \text{ (nsbn } \sigma \text{ (nnth } ls \text{ (the-enat (epred (nlength } ls))))$
 $(nnth \text{ } ls \text{ (the-enat (nlength } ls))))$)))
using *assms lcppl-nlast[of ls σ f g]* **by** *blast*
have 3: $nlast \text{ (nmap (}\lambda x. x + \text{ (nnth } ls \text{ (the-enat (epred(nlength } ls))))$
 $(cppl \text{ } f \text{ } g \text{ (nsbn } \sigma \text{ (nnth } ls \text{ (the-enat (epred (nlength } ls))))$
 $(nnth \text{ } ls \text{ (the-enat (nlength } ls)))) =$
 $(\lambda x. x + \text{ (nnth } ls \text{ (the-enat (epred(nlength } ls))))$
 $(nlast \text{ (cppl } f \text{ } g \text{ (nsbn } \sigma \text{ (nnth } ls \text{ (the-enat (epred (nlength } ls))))$
 $(nnth \text{ } ls \text{ (the-enat (nlength } ls))))$)))
using *nnth-nmap assms* **by** (*simp add: cppl-fprojection*)
have 4: $nnth \text{ } ls \text{ (the-enat (epred (nlength } ls))) < nnth \text{ } ls \text{ (Suc (the-enat (epred (nlength } ls)))}$
using *assms unfolding nidx-expand*
by (*metis Suc-ile-eq co.enat.exhaust-sel eSuc-enat enat.simps(3) enat-ord-simps(2) enat-the-enat*
i0-less i0-ne-infinity lessI nfinite-conv-nlength-enat)
have 5: $enat \text{ (nnth } ls \text{ (Suc (the-enat (epred (nlength } ls))))} \leq nlength \text{ } \sigma$
using *assms*
by (*metis Orderings.order-eq-iff co.enat.exhaust-sel enat.simps(3) enat-the-enat epred-simps(5)*
i0-less i0-ne-infinity nfinite-conv-nlength-enat nnth-nlast the-enat-eSuc)
have 6: $nlast \text{ (nmap (}\lambda x. x + \text{ (nnth } ls \text{ (the-enat (epred(nlength } ls))))$
 $(cppl \text{ } f \text{ } g \text{ (nsbn } \sigma \text{ (nnth } ls \text{ (the-enat (epred (nlength } ls))))$
 $(nnth \text{ } ls \text{ (the-enat (nlength } ls)))) =$
 $nnth \text{ } ls \text{ (Suc (the-enat (epred (nlength } ls)))}$
using *cppl-nlast-i[of f g σ ls (the-enat (epred (nlength } ls))]* *assms 4 5*
by (*metis co.enat.exhaust-sel enat.simps(3) epred-simps(5) i0-less i0-ne-infinity*
nfinite-conv-nlength-enat the-enat-eSuc)
have 7: $nnth \text{ } ls \text{ (Suc (the-enat (epred (nlength } ls)))} = the-enat(nlength \text{ } \sigma)$
using *assms*
by (*metis co.enat.collapse enat-eSuc-iff nfinite-nlength-enat nnth-nlast the-enat.simps*
zero-less-iff-neq-zero)
show *?thesis*
by (*simp add: 2 6 7*)
qed

lemma *lcppl-zero-zero*:
assumes *nidx ls*
nfinite ls
nfinite σ
 $nlast \text{ } ls = the-enat(nlength \text{ } \sigma)$
 $nlength \text{ } ls = 0$
shows $(nnth \text{ (lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } 0) =$
 $(nmap \text{ (}\lambda x. x + \text{ (nnth } ls \text{ } 0)) \text{ (cppl } f \text{ } g \text{ (nsbn } \sigma \text{ (nnth } ls \text{ } 0) \text{ (nnth } ls \text{ } 0))))$
proof (*cases ls*)
case (*NNil x1*)
then show *?thesis*

```

by (metis lcppl-code(1) nnth-NNil)
next
case (NCons x21 ls)
then show ?thesis
using assms(5) by auto
qed

```

lemma *lcppl-nfirst*:

```

assumes nid $x$  ls
        nfinite ls
        nfinite  $\sigma$ 
        nlast ls = the-enat(nlength  $\sigma$ )
        ( $\forall i. i < \text{nlength } ls \longrightarrow ((\text{nsubn } \sigma (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) \models f \text{ fproj } g))$ 
        nlength ls > 0
shows   nfirst(nfirst((lcppl f g  $\sigma$  ls))) = nfirst ls
proof -
have 01: (nfirst((lcppl f g  $\sigma$  ls))) =
        (nmap ( $\lambda x. x + (\text{nnth } ls \ 0)$ ) (cppl f g (nsubn  $\sigma$  (nnth ls 0) (nnth ls (Suc 0)))))
using assms
by (metis lcppl-nnth ndropn-0 ndropn-nfirst zero-enat-def)
have 02: nlength ls > 0  $\longrightarrow$ 
        nfirst((nmap ( $\lambda x. x + (\text{nnth } ls \ 0)$ ) (cppl f g (nsubn  $\sigma$  (nnth ls 0) (nnth ls (Suc 0))))) =
        (nnth ls 0)
        using assms 01 cppl-fprojection[of f g ]
        by (metis cppl-nfirst enat-0-iff(1) ndropn-nfirst)
show ?thesis
using 01 02 assms
by (metis ndropn-0 ndropn-nfirst)
qed

```

lemma *lcppl-nfirst-alt*:

```

assumes nid $x$  ls
         $\neg$ nfinite ls
         $\neg$ nfinite  $\sigma$ 
        ( $\forall i. i < \text{nlength } ls \longrightarrow ((\text{nsubn } \sigma (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) \models f \text{ fproj } g))$ 
shows   nfirst(nfirst((lcppl f g  $\sigma$  ls))) = nfirst ls
proof -
have 01: (nfirst((lcppl f g  $\sigma$  ls))) =
        (nmap ( $\lambda x. x + (\text{nnth } ls \ 0)$ ) (cppl f g (nsubn  $\sigma$  (nnth ls 0) (nnth ls (Suc 0)))))
using assms
by (metis i0-less lcppl-nnth nfinite-ntaken nlength-eq-enat-nfiniteD nnth-NNil nnth-nlast
        ntaken-0 ntaken-nlast zero-enat-def)
have 02: nfirst((nmap ( $\lambda x. x + (\text{nnth } ls \ 0)$ ) (cppl f g (nsubn  $\sigma$  (nnth ls 0) (nnth ls (Suc 0))))) =
        (nnth ls 0)
        using assms 01 cppl-fprojection[of f g ]
        by (metis (no-types, lifting) add-0 gr-zeroI ndropn-0 ndropn-nfirst nlength-eq-enat-nfiniteD
        nnth-nmap zero-enat-def zero-le)
show ?thesis
using 01 02 assms
by (metis ndropn-0 ndropn-nfirst)

```


qed

lemma *lcppl-nfusecat-nlastnfirst*:

assumes *nfinite ls*

nidx ls

nfinite σ

nlast ls = the-enat(nlength σ)

$((nlength\ ls = 0 \wedge ((nsubn\ \sigma\ (nnth\ ls\ 0)\ (nnth\ ls\ 0)) \models f\ fproj\ g)) \vee$

$(nlength\ ls > 0 \wedge (\forall i. i < nlength\ ls \longrightarrow ((nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g))))$

shows *nlastnfirst (lcppl f g σ ls)*

proof (*cases is-NNil ls*)

case *True*

then show *?thesis*

by (*metis is-NNil-def lcppl-code(1) nlastnfirst-NNil*)

next

case *False*

then show *?thesis*

proof (*auto simp add: nlastnfirst-def1*)

fix *i*

assume *a: enat (Suc i) ≤ nlength (lcppl f g σ ls)*

assume *b: ¬ is-NNil ls*

show *nlast (nnth (lcppl f g σ ls) i) = nfirst (nnth (lcppl f g σ ls) (Suc i))*

proof —

have *1: nlength ls > 0*

by (*metis a assms(2) assms(5) enat.inject ile0-eq lcppl-nlength-zero old.nat.distinct(1) zero-enat-def*)

have *2: (∀ i. i < nlength ls → ((nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f fproj g))*

using *assms(5)* **by** *auto*

have *3: nidx ls*

by (*simp add: assms(2)*)

have *4: i < nlength ls*

by (*metis 1 3 Suc-ile-eq a assms(1) co.enat.exhaust-sel dual-order.order-iff-strict illess-Suc-eq lcppl-nlength less-numeral-extra(3)*)

have *5: (nnth (lcppl f g σ ls) i) =*

nmap (λx::nat. x + nnth ls i) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))

using *lcppl-nnth[of ls i f g σ]*

using *3 4* **by** *blast*

have *6: (Suc i) < nlength ls*

by (*metis 1 3 a assms(1) co.enat.exhaust-sel illess-Suc-eq lcppl-nlength less-numeral-extra(3)*)

have *7: (nnth (lcppl f g σ ls) (Suc i)) =*

nmap (λx::nat. x + nnth ls (Suc i)) (cppl f g (nsubn σ (nnth ls (Suc i)) (nnth ls (Suc (Suc i)))))

using *lcppl-nnth[of ls Suc i f g σ]*

using *3 6* **by** *blast*

have *71: nnth ls i < nnth ls (Suc i)*

using *assms nidx-expand[of ls] 6 order-less-imp-le* **by** *blast*

have *72: enat (nnth ls (Suc i)) ≤ nlength σ*

using *assms*

by (*metis 3 6 dual-order.order-iff-strict enat-ord-simps(1) nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast the-enat.simps*)

```

have 8:  $nlast (nmap (\lambda x::nat. x + nnth\ ls\ i) (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) =$ 
   $(nnth\ ls\ (Suc\ i))$ 
  using assms 2 3 4 5 6 cppl-nlast-i[of f g  $\sigma$  ls i]
  using 71 72 by blast
have 9:  $nfirst (nmap (\lambda x::nat. x + nnth\ ls\ (Suc\ i))$ 
   $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ (Suc\ i))\ (nnth\ ls\ (Suc\ (Suc\ i)))) =$ 
   $(nnth\ ls\ (Suc\ i))$ 
  using 2 6 cppl-fprojection[of f g]
  by (metis add-0 nlast-NNil nnth-nmap ntaken-0 ntaken-nlast zero-enat-def zero-order(1))
show ?thesis using 8 9 5 7 by presburger
qed
qed
qed

```

lemma *lcppl-nfusecat-nlastnfirst-alt:*

```

assumes  $\neg nfinite\ ls$ 
   $nidx\ ls$ 
   $\neg nfinite\ \sigma$ 
   $(\forall i. i < nlength\ ls \longrightarrow ((nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g))$ 
shows  $nlastnfirst\ (lcppl\ f\ g\ \sigma\ ls)$ 
proof (auto simp add: nlastnfirst-def1)
fix  $i$ 
assume  $a: enat\ (Suc\ i) \leq nlength\ (lcppl\ f\ g\ \sigma\ ls)$ 
show  $nlast\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i) = nfirst\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ (Suc\ i))$ 
proof -
  have 2:  $(\forall i. i < nlength\ ls \longrightarrow ((nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g))$ 
    using assms by auto
  have 3:  $nidx\ ls$ 
    by (simp add: assms)
  have 4:  $i < nlength\ ls$ 
    by (meson assms(1) enat-iless linorder-less-linear nfinite-conv-nlength-enat)
  have 5:  $(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i) =$ 
     $nmap (\lambda x::nat. x + nnth\ ls\ i) (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$ 
    using lcppl-nnth[of ls i f g  $\sigma$ ]
    using 3 4 by blast
  have 6:  $(Suc\ i) < nlength\ ls$ 
    by (metis 4 assms(1) eSuc-enat ileI1 nlength-eq-enat-nfiniteD order-neq-le-trans)
  have 7:  $(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ (Suc\ i)) =$ 
     $nmap (\lambda x::nat. x + nnth\ ls\ (Suc\ i))$ 
     $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ (Suc\ i))\ (nnth\ ls\ (Suc\ (Suc\ i))))$ 
    using lcppl-nnth[of ls Suc i f g  $\sigma$ ]
    using 3 6 by blast
  have 8:  $nlast (nmap (\lambda x::nat. x + nnth\ ls\ i) (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) =$ 
     $(nnth\ ls\ (Suc\ i))$ 
    using assms 2 3 4 5 6
    by (simp add: cppl-nlast-i nfinite-conv-nlength-enat nidx-expand)
  have 9:  $nfirst (nmap (\lambda x::nat. x + nnth\ ls\ (Suc\ i))$ 
     $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ (Suc\ i))\ (nnth\ ls\ (Suc\ (Suc\ i)))) =$ 
     $(nnth\ ls\ (Suc\ i))$ 
    using 2 6 cppl-fprojection[of f g]

```

```

    by (metis add-0 nlast-NNil nnth-nmap ntaken-0 ntaken-nlast zero-enat-def zero-order(1))
  show ?thesis using 8 9 5 7 by presburger
qed
qed

lemma lcppl-nfusecat-nlength:
  assumes nidx ls
    nfinite ls
    nfinite σ
    nlast ls = the-enat(nlength σ)
    ((nlength ls = 0 ∧ ((nsubn σ (nnth ls 0) (nnth ls 0) ⊨ f fproj g))) ∨
     (nlength ls > 0 ∧ (∀ i. i < nlength ls ⟶ ((nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f fproj g))))
  shows (nlength ls = 0 ⟶ nlength(nfusecat (lcppl f g σ ls)) = 0) ∧
    (nlength ls > 0 ⟶
     nlength(nfusecat (lcppl f g σ ls)) =
     ( ∑ k=0..(the-enat (epred(nlength ls))). ( nlength (nnth (lcppl f g σ ls) k) ) ) )
proof -
  have 1: nlastnfirst (lcppl f g σ ls)
    using assms lcppl-nfusecat-nlastnfirst by blast
  have 20: nlength ls = 0 ⟶
    (lcppl f g σ ls) =
    (NNil ((nmap (λx. x+ (nnth ls 0)) (cppl f g (nsubn σ (nnth ls 0) (nnth ls 0) ) ) ) ) )
  by (metis ile0-eq lcppl-code(1) nellist-eq-nnth-eq nlength-NNil nnth-NNil the-enat.simps
    the-enat-0)
  have 2: nlength ls = 0 ⟶
    nfusecat (lcppl f g σ ls) =
    ((nmap (λx. x+ (nnth ls 0)) (cppl f g (nsubn σ (nnth ls 0) (nnth ls 0) ) ) ) )
  using 20 nfusecat-NNil by auto
  have 3: nlength ls = 0 ⟶
    nlength ((nmap (λx. x+ (nnth ls 0)) (cppl f g (nsubn σ (nnth ls 0) (nnth ls 0) ) ) ) ) = 0
  using assms
  by (metis cppl-empty diff-self-eq-0 idiff-enat-enat less-numeral-extra(3) min.orderE ndropn-nlast
    ndropn-nlength nlength-NNil nlength-nmap nnth-nlast not-le-imp-less nsubn-nlength the-enat-0
    zero-enat-def)
  have 4: nlength ls = 0 ⟶ nlength(nfusecat (lcppl f g σ ls)) = 0
    using 2 3 by auto
  have 5: nfinite (lcppl f g σ ls)
    using assms(1) assms(2) lcppl-nfinite by blast
  have 6: all-nfinite (lcppl f g σ ls)
  by (metis (no-types, lifting) 20 all-nfinite-nnth-a assms(1) assms(2) assms(5) co-enat.exhaust-sel
    cppl-fprojection iless-Suc-eq lcppl-nlength lcppl-nnth nfinite-nmap nnth-NNil
    not-less-iff-gr-or-eq)
  have 7: nlength (lcppl f g σ ls) = (epred(nlength ls))
    by (metis assms(1) assms(2) assms(5) epred-0 lcppl-nlength lcppl-nlength-zero)
  have 8: nlength ls > 0 ⟶
    nlength(nfusecat (lcppl f g σ ls)) =
    ( ∑ k=0..(the-enat (epred(nlength ls))). ( nlength (nnth (lcppl f g σ ls) k) ) )
  using nfusecat-nlength-nfinite[of (lcppl f g σ ls)] 5 6 7 1 by presburger
  show ?thesis
  using 4 8 by blast

```

qed

lemma *lcppl-nfusecat-nidx*:

assumes *nidx ls*

nnth ls 0 = 0

nfinite ls

nfinite σ

nlast ls = the-enat(nlength σ)

$(\forall i < \text{nlength } ls. (\text{nsubn } \sigma (\text{nnth } ls \ i) (\text{nnth } ls \ (\text{Suc } i))) \models f \text{ fproj } g)$

nlength ls > 0

shows *nidx (nfusecat ((lcppl f g σ ls)))*

proof –

have 0: *nlength σ > 0 → nlength ls > 0*

using *assms by auto*

have 2: *nlength σ > 0 → nlastnfirst (lcppl f g σ ls)*

using *assms lcppl-nfusecat-nlastnfirst by blast*

have 3: $(\forall i < \text{nlength } ls. \text{nidx } (\text{cppl } f \ g \ (\text{nsubn } \sigma (\text{nnth } ls \ i) (\text{nnth } ls \ (\text{Suc } i))))$

using *assms cppl-fprojection by auto*

have 4: $(\forall i < \text{nlength } ls.$

nidx (nmap (λx. x+(nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))))

using 3 **by** (*simp add: Suc-ile-eq nidx-expand*)

have 5: *nlength σ > 0 →*

$(\forall i < \text{nlength } ls. \text{nidx } (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i))$

using *assms by (simp add: 4 lcppl-nnth)*

have 6: *nlength σ > 0 → nlength (lcppl f g σ ls) = epred(nlength ls)*

using *assms lcppl-nlength by blast*

have 7: *nlength σ > 0 →*

$(\forall i \leq \text{nlength } ls.$

nidx (nnth (lcppl f g σ ls) i))

using 5 *assms*

by (*metis 6 Extended-Nat.eSuc-mono antisym-conv2 co.enat.exhaust-sel enat-the-enat epred-simps(5)*

i0-ne-infinity iless-Suc-eq lcppl-nfinite nnth-beyond nnth-nlast)

have 68: $\bigwedge i. i < \text{nlength } ls \implies 0 < \text{nlength } (\text{nsubn } \sigma (\text{nnth } ls \ i) (\text{nnth } ls \ (\text{Suc } i))) \implies$

$(\text{nsubn } \sigma (\text{nnth } ls \ i) (\text{nnth } ls \ (\text{Suc } i))) \models (f \text{ fproj } g) \implies$

$0 < \text{nlength } (\text{cppl } f \ g \ (\text{nsubn } \sigma (\text{nnth } ls \ i) (\text{nnth } ls \ (\text{Suc } i))))$

using *cppl-more by blast*

have 69: $\bigwedge i. i < \text{nlength } ls \implies 0 < \text{nlength } (\text{nsubn } \sigma (\text{nnth } ls \ i) (\text{nnth } ls \ (\text{Suc } i)))$

using *assms unfolding nidx-expand by simp*

(*metis assms(1) dual-order.order-iff-strict eSuc-enat enat-ord-simps(2) ileI1 less-numeral-extra(3)*

nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast nsubn-nlength-gr-one the-enat.simps)

have 70: *nlength σ > 0 → (∀ i < nlength ls. (0::enat) < nlength (nnth (lcppl f g σ ls) i))*

using *lcppl-nnth[of ls - f g σ] 68 69 assms(1) assms(6) by auto*

have 700: *nlength σ > 0 → (∀ lx ∈ nset (lcppl f g σ ls). (0::enat) < nlength lx)*

using 70 *assms*

by (*metis co.enat.exhaust-sel iless-Suc-eq in-nset-conv-nnth lcppl-nlength less-numeral-extra(3)*)

have 71: *nlength σ > 0 → (∀ lx ∈ nset (lcppl f g σ ls). nfinite lx)*

using *assms unfolding cppl-fprojection*

by (*metis (no-types, lifting) co.enat.exhaust-sel i0-less iless-Suc-eq in-nset-conv-nnth*

lcppl-nlength lcppl-nnth nfinite-nmap)

have 8: *nlength σ > 0 →*

```

      nidx (nfusecat ( (lcppl f g σ ls)))
    using assms nidx-nfusecat[of ( (lcppl f g σ ls)) ] 700 71
  by (metis 2 5 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength zero-less-iff-neq-zero)
from 8 show ?thesis using assms
by (metis gr-zeroI less-numeral-extra(3) nfinite-nlength-enat nidx-less-last-1 nnth-nlast
the-enat.simps zero-enat-def)

```

qed

lemma *lcppl-nfusecat-nidx-alt:*

```

assumes nidx ls
      nnth ls 0 = 0
      ¬nfinite ls
      ¬nfinite σ
      (∀ i < nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f fproj g)
shows nidx (nfusecat ( (lcppl f g σ ls)))
proof -
  have 2: nlastnfirst (lcppl f g σ ls)
    using assms lcppl-nfusecat-nlastnfirst-alt by blast
  have 3: (∀ i < nlength ls. nidx (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )))
    using assms cppl-fprojection by auto
  have 4: (∀ i < nlength ls.
      nidx (nmap (λx. x+(nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ))))
    using 3 by (simp add: Suc-ile-eq nidx-expand)
  have 5: (∀ i < nlength ls. nidx (nnth (lcppl f g σ ls) i) )
    using assms by (simp add: 4 lcppl-nnth)
  have 7: (∀ i < nlength ls.
      nidx (nnth (lcppl f g σ ls) i) )
    using 5 by simp
  have 60: ∧j. (nsubn σ (nnth ls j) (nnth ls (Suc j))) ⊨ (f fproj g)
    by (metis assms(3) assms(5) leI nfinite-ntaken ntaken-all)
  have 61: ∧j. nnth ls j < nnth ls (Suc j)
    by (metis Suc-ile-eq assms(1) assms(3) nfinite-ntaken nidx-expand not-le-imp-less ntaken-all)
  have 62: ∧j. enat (nnth ls (Suc j)) ≤ nlength σ
    by (meson assms(4) enat-iless enat-less-imp-le nfinite-conv-nlength-enat)
  have 63: ∧j. nlast (nmap (λx::nat. x + nnth ls j) (cppl f g (nsubn σ (nnth ls j) (nnth ls (Suc j))))) =
      nnth ls (Suc j)
    using 60 61 62 cppl-nlast-i by blast
  have 68: ∧i. i < nlength ls ⇒ 0 < nlength (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⇒
      (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ (f fproj g) ⇒
      0 < nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))
    using cppl-more by blast
  have 69: ∧i. i < nlength ls ⇒ 0 < nlength (nsubn σ (nnth ls i) (nnth ls (Suc i)))
    using assms unfolding nidx-expand using 61 62 nsubn-nlength-gr-one by blast
  have 700: nlength σ > 0 ⇒ (∀ i < nlength ls. (0::enat) < nlength (nnth (lcppl f g σ ls) i))
    using lcppl-nnth[of ls - f g σ] 68 69 assms by (simp )
  have 701: nlength σ > 0 ⇒ (∀ lx ∈ nset (lcppl f g σ ls). (0::enat) < nlength lx)
    using 700 assms
    by (metis co.enat.exhaust-sel iless-Suc-eq in-nset-conv-nnth lcppl-nlength-alt
      nlength-eq-enat-nfiniteD zero-enat-def)
  have 70: (∀ lx ∈ nset (lcppl f g σ ls). (0::enat) < nlength lx)

```

```

using 701 assms nfinite-conv-nlength-enat zero-enat-def by fastforce
have 71:  $(\forall lx \in nset (lcppl\ f\ g\ \sigma\ ls). \text{ nfinite } lx)$ 
using assms
by (metis (no-types, lifting) co.enat.exhaust-sel cppl-fprojection iless-Suc-eq in-nset-conv-nnth
lcppl-nlength-alt lcppl-nnth nfinite-nmap nlength-eq-enat-nfiniteD zero-enat-def)
have 8: nidx (nfusecat ( (lcppl f g  $\sigma$  ls)))
using assms nidx-nfusecat[of ( (lcppl f g  $\sigma$  ls)) ] 70 71
by (metis 2 7 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength-alt
nlength-eq-enat-nfiniteD zero-enat-def)
from 8 show ?thesis using assms by blast
qed

```

lemma *lcppl-nlength-all-nfinite*:

```

assumes nidx ls
  nnth ls 0 = 0
  nfinite ls
  nfinite  $\sigma$ 
  nlast ls = the-enat(nlength  $\sigma$ )
   $(\forall i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$ 
  nlength  $\sigma > 0$ 
shows  $(\forall j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls). \text{ nfinite}(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j))$ 
proof
  fix j
  show  $j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls) \longrightarrow \text{ nfinite } (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j)$ 
  proof –
  have 1: nlength  $\sigma > 0 \longrightarrow nlastnfirst\ (lcppl\ f\ g\ \sigma\ ls)$ 
    using assms
    by (metis enat.distinct(2) enat-the-enat i0-less lcppl-nfusecat-nlastnfirst
nfinite-conv-nlength-enat nnth-nlast the-enat-0)
  have 2: nlength  $\sigma > 0 \longrightarrow nlength\ ls > 0$ 
    using assms
    by (metis enat.distinct(2) enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0)
  have 3: nlength  $\sigma > 0 \longrightarrow j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls) \longrightarrow$ 
     $(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j) =$ 
     $(nmap\ (\lambda x. x + (nnth\ ls\ j))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ j)\ (nnth\ ls\ (Suc\ j)))))$ 
    using assms lcppl-nnth[of ls j f g  $\sigma$  ]
    by (metis 2 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength not-gr-zero)
  have 4: nlength  $\sigma > 0 \longrightarrow j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls) \longrightarrow$ 
    nfinite  $(nmap\ (\lambda x. x + (nnth\ ls\ j))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ j)\ (nnth\ ls\ (Suc\ j)))))$ 
    using assms 2
    by (metis co.enat.exhaust-sel cppl-fprojection iless-Suc-eq lcppl-nlength nfinite-nmap not-gr-zero)
  show ?thesis
  using 3 4 assms(7) by presburger
qed
qed

```

lemma *lcppl-nlength-all-gr-zero*:

```

assumes nidx ls
  nnth ls 0 = 0
  nfinite ls

```

```

  nfinite  $\sigma$ 
  nlast  $ls = the-enat(nlength\ \sigma)$ 
   $(\forall\ i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$ 
   $nlength\ \sigma > 0$ 
shows  $(\forall\ j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls). nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j) > 0)$ 
proof
  fix  $j$ 
  show  $j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls) \longrightarrow 0 < nlength\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j)$ 
  proof -
  have 1:  $nlength\ \sigma > 0 \longrightarrow nlastnfirst\ (lcppl\ f\ g\ \sigma\ ls)$ 
    using assms
    by (metis enat.distinct(2) enat-the-enat i0-less lcppl-nfusecat-nlastnfirst
      nfinite-conv-nlength-enat nnth-nlast the-enat-0)
  have 2:  $nlength\ \sigma > 0 \longrightarrow nlength\ ls > 0$ 
    using assms
    by (metis enat.distinct(2) enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0)
  have 3:  $nlength\ \sigma > 0 \longrightarrow j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls) \longrightarrow$ 
     $(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j) =$ 
     $(nmap\ (\lambda x. x + (nnth\ ls\ j))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ j)\ (nnth\ ls\ (Suc\ j)))))$ 
    using assms lcppl-nnth[of ls j f g]
    by (metis 2 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength not-gr-zero)
  have 4:  $nlength\ \sigma > 0 \longrightarrow j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls) \longrightarrow$ 
     $nlength\ (nmap\ (\lambda x. x + (nnth\ ls\ j))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ j)\ (nnth\ ls\ (Suc\ j))))) > 0$ 
    using assms 2 cppl-more[of (nsubn  $\sigma$  (nnth  $ls\ j$ ) (nnth  $ls\ (Suc\ j)$ ) f g]
    by simp
    (metis co.enat.exhaust-sel eSuc-enat enat-le-plus-same(2) enat-ord-simps(1) gen-nlength-def
      i0-less ileI1 iless-Suc-eq lcppl-nlength nfinite-nlength-enat nidx-expand nidx-less-eq
      nlength-code nnth-nlast nsubn-nlength-gr-one the-enat.simps)
  show ?thesis
  using 3 4 assms(7) by presburger
qed
qed

```

lemma *lcppl-nlength-all-nfinite-alt:*

```

assumes nidx ls
   $nnth\ ls\ 0 = 0$ 
   $\neg nfinite\ ls$ 
   $\neg nfinite\ \sigma$ 
   $(\forall\ i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$ 
shows  $(\forall\ j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls). nfinite(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j))$ 
proof
  fix  $j$ 
  show  $j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls) \longrightarrow nfinite\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j)$ 
  proof -
  have 1:  $nlastnfirst\ (lcppl\ f\ g\ \sigma\ ls)$ 
    using assms
    using lcppl-nfusecat-nlastnfirst-alt by blast
  have 3:  $j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls) \longrightarrow$ 
     $(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j) =$ 
     $(nmap\ (\lambda x. x + (nnth\ ls\ j))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ j)\ (nnth\ ls\ (Suc\ j)))))$ 

```

```

using assms lcppl-nnth[of ls j f g σ ] by (metis leI nfinite-ntaken ntaken-all)
have 4:  $j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls) \longrightarrow$ 
 $\text{nfinite } (\text{nmap } (\lambda x. x + (\text{nnth } ls \ j)) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } ls \ j) (\text{nnth } ls \ (\text{Suc } j))))$ 
using assms
by (metis Suc-ile-eq cppl-fprojection linorder-le-cases nfinite-nmap nfinite-ntaken ntaken-all)
show ?thesis
using 3 4 assms by presburger
qed
qed

```

lemma *lcppl-nlength-all-gr-zero-alt:*

```

assumes nidx ls
 $\text{nnth } ls \ 0 = 0$ 
 $\neg \text{nfinite } ls$ 
 $\neg \text{nfinite } \sigma$ 
 $(\forall i < \text{nlength } ls. (\text{nsubn } \sigma \ (\text{nnth } ls \ i) (\text{nnth } ls \ (\text{Suc } i))) \models f \ \text{fproj } g)$ 
shows  $(\forall j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ j) > 0)$ 
proof
fix j
show  $j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls) \longrightarrow 0 < \text{nlength } (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ j)$ 
proof –
have 1:  $\text{nlastnfirst } (\text{lcppl } f \ g \ \sigma \ ls)$ 
using assms
using lcppl-nfusecat-nlastnfirst-alt by blast
have 3:  $j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls) \longrightarrow$ 
 $(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ j =$ 
 $(\text{nmap } (\lambda x. x + (\text{nnth } ls \ j)) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } ls \ j) (\text{nnth } ls \ (\text{Suc } j))))$ 
using assms lcppl-nnth[of ls j f g σ ] by (metis leI nfinite-ntaken ntaken-all)
have 4:  $j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls) \longrightarrow$ 
 $\text{nlength } (\text{nmap } (\lambda x. x + (\text{nnth } ls \ j)) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } ls \ j) (\text{nnth } ls \ (\text{Suc } j)))) > 0$ 
using assms
by (metis Suc-ile-eq cppl-more enat-ile linorder-le-cases nfinite-conv-nlength-enat
 $\text{nidx-expand nlength-nmap nsubn-nlength-gr-one}$ )
show ?thesis
using 3 4 assms by presburger
qed
qed

```

lemma *lcppl-nlast-nnth:*

```

assumes nidx ls
 $\text{nnth } ls \ 0 = 0$ 
 $\text{nfinite } ls$ 
 $\text{nfinite } \sigma$ 
 $\text{nlast } ls = \text{the-enat}(\text{nlength } \sigma)$ 
 $(\forall i < \text{nlength } ls. (\text{nsubn } \sigma \ (\text{nnth } ls \ i) (\text{nnth } ls \ (\text{Suc } i))) \models f \ \text{fproj } g)$ 
 $\text{nlength } \sigma > 0$ 
 $j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls)$ 
shows  $\text{nlast}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ j) = (\text{nnth } ls \ (\text{Suc } j))$ 
proof –
have 0:  $\text{nlength } \sigma > 0 \longrightarrow \text{nlength } ls > 0$ 

```



```

using assms
by (metis enat.distinct(2) enat-the-enat nfinite-nlength-enat nnth-nlast the-enat-0
    zero-less-iff-neq-zero)
have 1: nlength  $\sigma > 0 \longrightarrow$ 
     $j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls) \longrightarrow$ 
     $\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ j =$ 
     $(\text{nmap } (\lambda x. x + (\text{nnth } ls \ j)) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } ls \ j) \ (\text{nnth } ls \ (\text{Suc } j)) \ )))$ 
using assms lcppl-nnth[of ls j f g  $\sigma$ ]
by (metis 0 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength not-gr-zero)
have 2: nlength  $\sigma > 0 \longrightarrow$ 
     $j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls) \longrightarrow (\text{nnth } ls \ (\text{Suc } j)) \leq \text{nlength } \sigma$ 
using assms
by (metis dual-order.eq-iff enat.simps(3) enat-ord-simps(1) enat-the-enat
    nfinite-conv-nlength-enat nid $\bar{x}$ -all-le-nlast nnth-beyond not-le-imp-less)
have 3: nlength  $\sigma > 0 \longrightarrow$ 
     $j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls) \longrightarrow$ 
     $\text{nlast } (\text{nmap } (\lambda x. x + (\text{nnth } ls \ j)) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } ls \ j) \ (\text{nnth } ls \ (\text{Suc } j)) \ )))) =$ 
     $(\text{nnth } ls \ (\text{Suc } j))$ 
using assms cppl-nlast-i[of f g  $\sigma$  ls j]
by (metis 0 2 Suc-ile-eq co.enat.exhaust-sel i0-less iless-Suc-eq lcppl-nlength nid $\bar{x}$ -expand)
show ?thesis using 1 3 assms
by presburger
qed

```

lemma *lcppl-nlast-nnth-alt:*

```

assumes nid $\bar{x}$  ls
     $\text{nnth } ls \ 0 = 0$ 
     $\neg \text{nfinite } ls$ 
     $\neg \text{nfinite } \sigma$ 
     $(\forall \ i < \text{nlength } ls. (\text{nsubn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)) \ ) \models f \ fproj \ g)$ 
     $j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls)$ 
shows  $\text{nlast}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ j) = (\text{nnth } ls \ (\text{Suc } j))$ 
proof –
have 1:
     $j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls) \longrightarrow$ 
     $\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ j =$ 
     $(\text{nmap } (\lambda x. x + (\text{nnth } ls \ j)) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } ls \ j) \ (\text{nnth } ls \ (\text{Suc } j)) \ )))$ 

    using assms lcppl-nnth[of ls j f g  $\sigma$ ] by (metis leI nfinite-ntaken ntaken-all)
have 2:
     $j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls) \longrightarrow (\text{nnth } ls \ (\text{Suc } j)) \leq \text{nlength } \sigma$ 
using assms
by (simp add: nfinite-conv-nlength-enat)
have 3:  $j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls) \longrightarrow$ 
     $\text{nlast } (\text{nmap } (\lambda x. x + (\text{nnth } ls \ j)) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } ls \ j) \ (\text{nnth } ls \ (\text{Suc } j)) \ )))) =$ 
     $(\text{nnth } ls \ (\text{Suc } j))$ 
using assms cppl-nlast-i[of f g  $\sigma$  ls j]
by (metis 2 co.enat.exhaust-sel eSuc-enat ileI1 iless-Suc-eq lcppl-nlength-alt
    nid $\bar{x}$ -expand nlength-eq-enat-nfiniteD zero-enat-def)
show ?thesis using 1 3 assms

```

by presburger
qed

lemma *lcppl-nfusecat-pfilt-fpower-help:*

assumes *nidx ls*

nnth ls 0 = 0

nfinite ls

nfinite σ

nlst ls = the-enat(nlength σ)

$(\forall i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$

nlength σ > 0

shows $(\forall i < nlength\ ls. g\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)))$

proof –

have 1: $(\forall i < nlength\ ls. g\ (pfilt\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
 $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))$

using *assms cppl-fprojection* **by** *blast*

have 2: *nlength σ > 0* \longrightarrow

$(\forall i < nlength\ ls.$
 $(pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)) =$
 $(pfilt\ \sigma\ (nmap\ (\lambda x. x + (nnth\ ls\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))))))$

using *assms* **by** (*simp add: lcppl-nnth*)

have 3: *nlength σ > 0* \longrightarrow

$(\forall i < nlength\ ls.$
 $nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)) =$
 $nlength\ (pfilt\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
 $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$
 $)$

by (*simp add: 2 pfilt-nlength*)

have 4: *nlength σ > 0* \longrightarrow

$(\forall i < nlength\ ls.$
 $(\forall j \leq nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$
 $(nnth\ (pfilt\ \sigma\ (nmap\ (\lambda x. x + (nnth\ ls\ i))$
 $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ j)$
 $=$
 $(nnth\ \sigma\ ((nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ j) + (nnth\ ls\ i)))$
 $)$
 $)$

using *nnth-nmap pfilt-nmap* **by** (*metis 2 nlength-nmap*)

have 5: *nlength σ > 0* \longrightarrow

$(\forall i < nlength\ ls.$
 $(\forall j \leq nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$
 $(nnth\ (pfilt\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
 $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ j) =$
 $(nnth\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
 $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ j))$
 $)$

by (*simp add: 3 pfilt-nnth*)

have 6: *nlength σ > 0* \longrightarrow

```

    (∀ i < nlength ls.
      (nnth ls (Suc i)) ≤ nlength σ
    )
  using assms
  by (metis eSuc-enat enat-le-plus-same(2) enat-ord-simps(1) gen-nlength-def ileI1
    nfinite-conv-nlength-enat nidx-less-eq nlength-code nnth-nlast the-enat.simps)
have 7: nlength σ > 0 ⟶
  (∀ i < nlength ls.
    (nnth ls i) ≤ (nnth ls (Suc i))
  )
  using assms
  by (metis eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc)
have 70: ⋀ i j . i < nlength ls ⟹ (nnth ls i) < (nnth ls (Suc i)) ⟹
  enat (nnth ls (Suc i)) ≤ nlength σ ⟹
  (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ (f fproj g) ⟹
  enat j < nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ⟹
  0 ≤ nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) j ∧
  nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) j ≤ (nnth ls (Suc i)) - (nnth ls i)
  using cppl-bounds by blast
have 71: ⋀ i . i < nlength ls ⟹ (nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ (f fproj g) ⟹
  nnth ls i < nnth ls (Suc i) ⟹
  enat (nnth ls (Suc i)) ≤ nlength σ ⟹
  nlast (nmap (λx. x + nnth ls i) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))) =
  nnth ls (Suc i)
  using cppl-nlast-i by blast
have 8: nlength σ > 0 ⟶
  (∀ i < nlength ls.
    (∀ j ≤ nlength (pfilt σ (nnth (lcppl f g σ ls) i)).
      (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))) j) ≤
        (nnth ls (Suc i)) - (nnth ls i)
    ))
  )
  using 6 assms 70 71 unfolding cppl-fprojection nidx-expand by simp
  (metis add-diff-cancel-right' dual-order.eq-iff eSuc-enat ileI1 not-le-imp-less ntaken-all
    ntaken-nlast)
have 9: nlength σ > 0 ⟶
  (∀ i < nlength ls.
    (∀ j ≤ nlength (pfilt σ (nnth (lcppl f g σ ls) i)).
      (nnth (nsubn σ (nnth ls i) (nnth ls (Suc i)))
        (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))) j)) =
        (nnth σ ((nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))) j) + (nnth ls i)))
      ))
    )
  )
  using 6 7 8
  by (simp add: add.commute nsubn-def1 ntaken-nnth)
have 10: nlength σ > 0 ⟶
  (∀ i < nlength ls.
    (∀ j ≤ nlength (pfilt σ (nnth (lcppl f g σ ls) i)).
      (pfilt (nsubn σ (nnth ls i) (nnth ls (Suc i)))
        (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))) j) =
        (pfilt σ (nnth (lcppl f g σ ls) i)
      ))
    )
  )

```

using 2 3 4 9 **by** (*simp add: pfilt-expand pfilt-nlength pfilt-nnth*)
show ?thesis **using** 1 10 *assms* **by** *fastforce*
qed

lemma *lcppl-nfusecat-pfilt-fpower-help-alt:*

assumes *nidx ls*
 $nnth\ ls\ 0 = 0$
 $\neg nfinite\ ls$
 $\neg nfinite\ \sigma$
 $(\forall\ i < nlength\ ls.\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$
shows $(\forall\ i < nlength\ ls.\ g\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)))$
proof –
have 1: $(\forall\ i < nlength\ ls.\ g\ (pfilt\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
 $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))$
using *assms cppl-fprojection* **by** *blast*
have 2: $(\forall\ i < nlength\ ls.$
 $(pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)) =$
 $(pfilt\ \sigma\ (nmap\ (\lambda x. x + (nnth\ ls\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))))))$
using *assms* **by** (*simp add: lcppl-nnth*)
have 3: $(\forall\ i < nlength\ ls.$
 $nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)) =$
 $nlength\ (pfilt\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
 $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$
 $)$
by (*simp add: 2 pfilt-nlength*)
have 4: $(\forall\ i < nlength\ ls.$
 $(\forall\ j \leq nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$
 $(nnth\ (pfilt\ \sigma\ (nmap\ (\lambda x. x + (nnth\ ls\ i))$
 $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))\ j)$
 $=$
 $(nnth\ \sigma\ ((nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ j) + (nnth\ ls\ i)))$
 $)$
using *nnth-nmap pfilt-nmap* **by** (*metis 2 nlength-nmap*)
have 5: $(\forall\ i < nlength\ ls.$
 $(\forall\ j \leq nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$
 $(nnth\ (pfilt\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
 $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ j) =$
 $(nnth\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
 $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ j))$
 $)$
by (*simp add: 3 pfilt-nnth*)
have 6: $(\forall\ i < nlength\ ls.$
 $(nnth\ ls\ (Suc\ i)) \leq nlength\ \sigma$
 $)$
using *assms*
by (*simp add: nfinite-conv-nlength-enat*)
have 7: $(\forall\ i < nlength\ ls.$
 $(nnth\ ls\ i) \leq (nnth\ ls\ (Suc\ i))$
 $)$

using *assms*
by (*metis eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc*)
have 70: $\bigwedge i j . i < \text{nlength } ls \implies (\text{nnth } ls \ i) < (\text{nnth } ls \ (\text{Suc } i)) \implies$
 $\text{enat } (\text{nnth } ls \ (\text{Suc } i)) \leq \text{nlength } \sigma \implies$
 $(\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) \models (f \ \text{fproj } \ g) \implies$
 $\text{enat } j < \text{nlength } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \implies$
 $0 \leq \text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \ j \wedge$
 $\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \ j \leq (\text{nnth } ls \ (\text{Suc } i)) - (\text{nnth } ls \ i)$
using *cppl-bounds* **by** *blast*
have 71: $\bigwedge i . i < \text{nlength } ls \implies (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) \models (f \ \text{fproj } \ g) \implies$
 $\text{nnth } ls \ i < \text{nnth } ls \ (\text{Suc } i) \implies$
 $\text{enat } (\text{nnth } ls \ (\text{Suc } i)) \leq \text{nlength } \sigma \implies$
 $\text{nlast } (\text{nmap } (\lambda x . x + \text{nnth } ls \ i) \ (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) =$
 $\text{nnth } ls \ (\text{Suc } i)$
using *cppl-nlast-i* **by** *blast*
have 8: $(\forall i < \text{nlength } ls .$
 $(\forall j \leq \text{nlength } (\text{pfilt } \sigma \ (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ i)).$
 $(\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \ j) \leq$
 $(\text{nnth } ls \ (\text{Suc } i)) - (\text{nnth } ls \ i)$
 $))$
using *assms* **using** 6 *assms* 70 71 **unfolding** *cppl-fprojection nidx-expand* **by** *simp*
 $(\text{metis add-diff-cancel-right' dual-order.eq-iff eSuc-enat ileI1 not-le-imp-less ntaken-all}$
 $\text{ntaken-nlast})$
have 9: $(\forall i < \text{nlength } ls .$
 $(\forall j \leq \text{nlength } (\text{pfilt } \sigma \ (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ i)).$
 $(\text{nnth } (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))))$
 $(\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \ j) =$
 $(\text{nnth } \sigma \ ((\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \ j) + (\text{nnth } ls \ i)))$
 $))$
using 6 7 8 **by** (*simp add: add.commute nsbn-def1 ntaken-nnth*)
have 10: $(\forall i < \text{nlength } ls .$
 $(\forall j \leq \text{nlength } (\text{pfilt } \sigma \ (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ i)).$
 $(\text{pfilt } (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))))$
 $(\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) =$
 $(\text{pfilt } \sigma \ (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ i))$
 $))$
using 2 3 4 9 **by** (*simp add: pfilt-expand pfilt-nlength pfilt-nnth*)
show *?thesis* **using** 1 10 *assms* **by** *fastforce*
qed

lsum lemmas

lemma *lsum-NNil*:

lsum (*NNil nell*) *a* = *NNil* (*a* + (*the-enat* (*nlength nell*)))

by *simp*

lemma *lsum-addzero-NNil*:

assumes *nfinite nell*

shows *addzero* (*lsum* (*NNil nell*) 0) =

(if nlength nell = 0 then (NNil 0) else (NCons 0 (NNil (the-enat (nlength nell))))))
using *assms* **unfolding** *addzero-def*
by *simp*
 (metis less-nat-zero-code ndropn-0 ndropn-nfirst ndropn-nlast nlength-NNil nnth-NNil)

lemma *lsum-eq-NNil-conv*:
 (lsum nells a) = (NNil b) \longleftrightarrow is-NNil nells \wedge a+ (the-enat (nlength(nfirst nells))) = b
by (metis NNil-eq-ntake-iff lsum.disc-iff(1) lsum-code(1) nellist.collapse(1) nellist.disc(1)
 nellist.inject(1))

lemma *lsum-eq-NCons-conv*:
 lsum nells a = (NCons b nells1) \longleftrightarrow
 (\exists nell nells'. nells = (NCons nell nells') \wedge b = a+(the-enat (nlength nell)) \wedge
 nells1 = lsum nells' (a+(the-enat (nlength nell))))
by (cases nells) (simp-all, blast)

lemma *lsum-addzero-NCons*:
 addzero (lsum (NCons nell nells) 0) = (NCons 0 (lsum (NCons nell nells) 0))
by (simp add: addzero-def)

lemma *lsum-nfirst*:
 nfirst (lsum nells a) = a+(the-enat (nlength(nfirst nells)))
proof (cases nells)
case (NNil x1)
then show ?thesis **by** simp (metis NNil-eq-ntake-iff nellist.inject(1))
next
case (NCons x21 nells1)
then show ?thesis **by** simp (metis NNil-eq-ntake-iff nnth-0 nnth-NNil ntaken-0 ntaken-nlast)
qed

lemma *nfinite-lsum-conv-a*:
assumes *nfinite nells*
shows *nfinite (lsum nells a)*
using *assms*
proof (induction nells arbitrary: a rule:nfinite-induct)
case (NNil y)
then show ?case **by** simp
next
case (NCons x nells)
then show ?case **by** simp
qed

lemma *nfinite-lsum-conv-b*:
assumes *nfinite (lsum nells a)*
shows *nfinite nells*

```

using assms
proof (induct zs≡lsum nells a arbitrary: a nells rule:nfinite-induct)
case (NNil y)
then show ?case
by (metis is-NNil-imp-nfinite lsum-eq-NNil-conv)
next
case (NCons x nell)
then show ?case
by (metis is-NNil-imp-nfinite lsum-code(2) nellist.collapse(2) nellist.sel(5) nfinite-NConsI)
qed

```

```

lemma nfinite-lsum-conv:
  nfinite (lsum nells a) ⟷ nfinite nells
using nfinite-lsum-conv-a nfinite-lsum-conv-b by blast

```

```

lemma lsum-nlength:
  nlength (lsum nells a) = nlength nells
proof (cases nfinite nells)
case True
then show ?thesis
  proof (induction nells arbitrary: a rule: nfinite-induct)
  case (NNil y)
  then show ?case by simp
  next
  case (NCons x nell)
  then show ?case by simp
  qed
next
case False
then show ?thesis
  proof (coinduction arbitrary: nells a rule: enat-coinduct)
  case (Eq-enat nells1)
  then show ?case
    proof –
      have 1: (nlength (lsum nells1 a) = (0::enat)) = (nlength nells1 = (0::enat))
        by (metis Eq-enat nfinite-lsum-conv-b nlength-eq-enat-nfiniteD zero-enat-def)
      show ?thesis
      by (metis 1 Eq-enat nbutlast-not-nfinite nfinite-lsum-conv-b nlength-nbutlast)
    qed
  qed
qed

```

```

lemma lsum-addzero-nfirst:
  nfirst (addzero (lsum nells 0)) = 0
by (metis addzero-def ndropn-nfirst ndropn-nfuse nellist.disc(1) nellist.disc(2) nfuse-leftneutral
  nfuse-nappend nlast-NNil nlength-NNil nnth-0 the-enat-0)

```

```

lemma lsum-addzero-nlength:
assumes nfinite(nfirst nells)
shows (nlength nells = 0  $\wedge$  nlength(nfirst nells) = 0  $\longrightarrow$ 
  nlength (addzero (lsum nells 0)) = 0)
 $\wedge$ 
(nlength nells = 0  $\wedge$  nlength(nfirst nells) > 0  $\longrightarrow$ 
  nlength (addzero (lsum nells 0)) = 1)
 $\wedge$ 
(nlength nells > 0  $\longrightarrow$ 
  nlength (addzero (lsum nells 0)) = (nlength nells) + 1)

using assms
by (simp add: addzero-def eSuc-plus-1 lsum-nfirst lsum-nlength)
  (metis enat.distinct(2) enat-the-enat nfinite-nlength-enat zero-enat-def)

```

```

lemma sum-nnth-help:
assumes i > 0
  i  $\leq$  nlength nells + 1
shows ( $\sum k = 0..(i-1). \textit{nlength} (\textit{nnth} (\textit{nells}) k) =$ 
  ( $\sum k = 1..i. \textit{nlength} (\textit{nnth} (\textit{nells}) (k-1))$ ))
using assms
proof
  (induct i)
  case 0
  then show ?case by blast
  next
  case (Suc i)
  then show ?case
  proof (cases i)
  case 0
  then show ?thesis by simp
  next
  case (Suc nat)
  then show ?thesis
    proof –
    have 1: (0::nat) < (i::nat)
      by (simp add: Suc)
    have 2: enat i  $\leq$  nlength (nells::'a nelist nelist) + (1::enat)
      using Suc.prems(2) Suc-ile-eq by auto
    have 3: ( $\sum k::nat = 0::nat..i - (1::nat). \textit{nlength} (\textit{nnth} \textit{nells} k) =$ 
      ( $\sum k::nat = 1::nat..i. \textit{nlength} (\textit{nnth} \textit{nells} (k - (1::nat)))$ ))
      using 1 2 Suc.hyps by blast
    have 4: ( $\sum k::nat = 0::nat..Suc i - (1::nat). \textit{nlength} (\textit{nnth} \textit{nells} k) =$ 
      ( $\sum k::nat = 0::nat..i. \textit{nlength} (\textit{nnth} \textit{nells} k)$ ))
      by simp
    have 5: ( $\sum k::nat = 0::nat..i. \textit{nlength} (\textit{nnth} \textit{nells} k) =$ 
      nlength (nnth nells i) + ( $\sum k::nat = 0::nat..(i-1). \textit{nlength} (\textit{nnth} \textit{nells} k)$ ))
      by (simp add: Suc)
    have 6: nlength (nnth nells i) + ( $\sum k::nat = 1::nat..i. \textit{nlength} (\textit{nnth} \textit{nells} (k - (1::nat)))$ ) =

```



```

      (∑ k::nat = 1::nat..Suc i. nlength (nnth nells (k - (1::nat))))
    by simp
  show ?thesis
  using 3 4 5 6 by presburger
qed
qed
qed

```

```

lemma lsum-nnth:
  assumes i ≤ nlength nells
    all-nfinite nells
  shows nnth (lsum nells a) i = a + (∑ k::nat = 0..i. (the-enat (nlength (nnth nells k))))
using assms
proof
  (induct i arbitrary: a nells)
  case 0
  then show ?case
    proof (cases nells)
    case (NNil x1)
    then show ?thesis by (simp add: nnth-NNil)
    next
    case (NCons x21 x22)
    then show ?thesis by simp
    qed
  next
  case (Suc i)
  then show ?case
    proof (cases nells)
    case (NNil x1)
    then show ?thesis using Suc.prem1 enat-0-iff(1) by simp
    next
    case (NCons x21 x22)
    then show ?thesis
      proof -
        have 1: nnth (lsum nells a) (Suc i) = nnth (lsum x22 (a + (the-enat (nlength x21)))) i
          by (simp add: NCons)
        have 2: enat (i::nat) ≤ nlength x22
          using NCons Suc.prem2 Suc-ile-eq by auto
        have 20: all-nfinite x22
          by (simp add: NCons Suc.prem2)
        have 3: nnth (lsum x22 (a + (the-enat (nlength x21)))) i =
          (a + (the-enat (nlength x21))) +
          (∑ k::nat = 0::nat..i. (the-enat (nlength (nnth x22 k))))
          using Suc.hyps[of x22 (a + (the-enat (nlength x21)))] 2 20 by blast
        have 4: a + (∑ k::nat = 0::nat..Suc i. (the-enat (nlength (nnth nells k)))) =
          a + (the-enat ((nlength (nnth nells 0)))) +
          (∑ k::nat = 1::nat..Suc i. (the-enat (nlength (nnth nells k))))
          by (simp add: sum.atLeast-Suc-atMost)
        have 5: (nlength (nnth nells 0)) = nlength x21

```

```

    by (simp add: NCons)
  have 6: ( $\sum k::nat = 1::nat..Suc\ i. (the-enat\ (nlength\ (nnth\ nells\ k)))) =$ 
    ( $\sum k::nat = 0::nat..i. (the-enat\ (nlength\ (nnth\ nells\ (k+1))))$ )
    using sum.shift-bounds-cl-nat-ivl[of  $\lambda k. (the-enat\ (nlength\ (nnth\ nells\ k)))\ 0\ 1\ i$ ]
    by simp
  have 7: ( $\sum k::nat = 0::nat..i. (the-enat\ (nlength\ (nnth\ nells\ (k+1)))) =$ 
    ( $\sum k::nat = 0::nat..i. (the-enat\ (nlength\ (nnth\ x22\ (k))))$ )
    using NCons by auto
  show ?thesis
  using 1 3 4 5 6 7 by presburger
qed
qed
qed

```

lemma *lsum-addzero-nnth*:

```

assumes  $i \leq nlength\ (addzero\ (lsum\ nells\ 0))$ 
          $nfinite\ (nfirst\ nells)$ 
shows ( $nlength\ nells = 0 \wedge nlength\ (nfirst\ nells) = 0 \longrightarrow$ 
          $nnth\ (addzero\ (lsum\ nells\ 0))\ i = (nnth\ (lsum\ nells\ 0)\ i)$ )
 $\wedge$ 
         ( $nlength\ nells = 0 \wedge nlength\ (nfirst\ nells) > 0 \longrightarrow$ 
          $nnth\ (addzero\ (lsum\ nells\ 0))\ i = (nnth\ (NCons\ 0\ (lsum\ nells\ 0))\ i)$ )
 $\wedge$ 
         ( $nlength\ nells > 0 \longrightarrow$ 
          $nnth\ (addzero\ (lsum\ nells\ 0))\ i = (nnth\ (NCons\ 0\ (lsum\ nells\ 0))\ i)$ )

```

```

using assms using lsum-addzero-nlength[of nells] lsum-nlength[of nells 0]
lsum-nfirst[of nells 0] using addzero-def by auto

```

lemma *lsum-nlast*:

```

assumes  $nfinite\ nells$ 
          $all-nfinite\ nells$ 
shows  $nlast\ (lsum\ nells\ a) = a + (\sum k::nat = 0..(the-enat\ (nlength\ nells)). (the-enat\ (nlength\ (nnth\ nells\ k))))$ 
using assms
by (metis (no-types, lifting) enat.simps(3) enat-le-plus-same(2) enat-the-enat gen-nlength-def
         lsum-nlength lsum-nnth nfinite-lsum-conv-a nfinite-nlength-enat nlength-code nnth-nlast sum.cong)

```

lemma *lsum-addzero-nlast*:

```

assumes  $nfinite\ nells$ 
shows  $nlast\ (addzero\ (lsum\ nells\ 0)) = nlast\ (lsum\ nells\ 0)$ 
by (simp add: addzero-def)

```

lemma *lsum-nnth-nfinite*:

```

assumes  $i \leq nlength\ nells$ 
          $all-gr-zero\ nells$ 
          $all-nfinite\ nells$ 
shows ( $\sum k::nat = 0..i. (the-enat\ (nlength\ (nnth\ nells\ k)))) < \infty$ 

```

```

using assms
using enat-ord-code(4) by blast

lemma sum-finite:
  assumes all-nfinite nells
     $i \leq \text{nlength } nells$ 
  shows  $(\sum k = 0..i. (\text{nlength}(\text{nnth } nells \ k))) < \infty$ 
using assms
proof (induction i arbitrary: nells)
case 0
then show ?case
  proof (cases nells)
  case (NNil x1)
  then show ?thesis using 0 by (simp add: nfinite-nlength-enat nnth-NNil)
  next
  case (NCons x21 x22)
  then show ?thesis using 0 using nfinite-nlength-enat by simp blast
  qed
next
case (Suc i)
then show ?case
  proof (cases nells)
  case (NNil x1)
  then show ?thesis using Suc by simp (metis enat.inject old.nat.distinct(2) zero-enat-def)
  next
  case (NCons x21 x22)
  then show ?thesis
    proof –
    have 1:  $(\sum k = 0..Suc \ i. \text{nlength } (\text{nnth } nells \ k))) =$ 
       $\text{nlength } (\text{nnth } nells \ 0) + (\sum k = 1..Suc \ i. \text{nlength } (\text{nnth } nells \ k))$ 
    by (metis One-nat-def sum.atLeast0-atMost-Suc-shift sum.atLeast-Suc-atMost-Suc-shift)
    have 2:  $(\sum k = 1..Suc \ i. \text{nlength } (\text{nnth } nells \ k))) =$ 
       $(\sum k = 0.. \ i. \text{nlength } (\text{nnth } nells \ (k+1)))$ 
    using sum.shift-bounds-cl-nat-ivl[of  $\lambda k. (\text{nlength } (\text{nnth } nells \ k)) \ 0 \ 1 \ i$ ]
    by (metis One-nat-def Suc-eq-plus1)
    have 3:  $(\sum k = 0.. \ i. \text{nlength } (\text{nnth } nells \ (k+1))) = (\sum k = 0.. \ i. \text{nlength } (\text{nnth } x22 \ k))$ 
    using NCons by auto
    have 4: all-nfinite x22
    by (simp add: NCons Suc.prem1)
    have 5:  $i \leq \text{nlength } x22$ 
    using NCons Suc.prem2 Suc-ile-eq by auto
    have 6:  $(\sum k = 0.. \ i. \text{nlength } (\text{nnth } x22 \ k)) < \infty$ 
    using Suc.IH[of x22] 4 5 by blast
    have 7:  $\text{nlength } (\text{nnth } nells \ 0) < \infty$ 
    by (metis Suc.prem1 all-nfinite-nnth-b enat.distinct(2) enat-ord-simps(4)
      nfinite-nlength-enat zero-enat-def zero-le)
    have 8:  $\text{nlength } (\text{nnth } nells \ 0) + (\sum k = 0.. \ i. \text{nlength } (\text{nnth } x22 \ k)) < \infty$ 
    using 6 7 by force
    show ?thesis using 1 2 3 8 by presburger

```

qed
 qed
 qed

lemma *sum-the-enat*:

assumes *all-nfinite nells*

i ≤ *nlength nells*

shows $(\sum k = 0..i. \text{the-enat}(\text{nlength}(\text{nnth nells } k))) = \text{the-enat}(\sum k = 0..i. (\text{nlength}(\text{nnth nells } k)))$

using *assms*

proof (*induction i arbitrary: nells*)

case 0

then show ?*case*

proof (*cases nells*)

case (*NNil x1*)

then show ?*thesis* **using** 0 **by** *simp*

next

case (*NCons x21 x22*)

then show ?*thesis* **using** 0 **by** *simp*

qed

next

case (*Suc i*)

then show ?*case*

proof (*cases nells*)

case (*NNil x1*)

then show ?*thesis* **using** *Suc* **by** *simp* (*metis enat-0-iff(2) old.nat.distinct(2)*)

next

case (*NCons x21 x22*)

then show ?*thesis*

proof –

have 1: $(\sum k = 0..Suc\ i. \text{the-enat}(\text{nlength}(\text{nnth nells } k))) =$
 $\text{the-enat}(\text{nlength}(\text{nnth nells } 0)) + (\sum k = 1..Suc\ i. \text{the-enat}(\text{nlength}(\text{nnth nells } k)))$

by (*simp add: sum.atLeast-Suc-atMost*)

have 2: $\text{the-enat}(\text{nlength}(\text{nnth nells } 0)) = \text{the-enat}(\text{nlength } x21)$

using *NCons* **by** *simp*

have 3: $(\sum k = 1..Suc\ i. \text{the-enat}(\text{nlength}(\text{nnth nells } k))) =$
 $(\sum k = 0..i. \text{the-enat}(\text{nlength}(\text{nnth nells } (k+1))))$

using *sum.shift-bounds-cl-nat-ivl*[*of* $\lambda k. \text{the-enat}(\text{nlength}(\text{nnth nells } k))$ 0 1 *i*]

by (*metis One-nat-def Suc-eq-plus1*)

have 4: $(\sum k = 0..i. \text{the-enat}(\text{nlength}(\text{nnth nells } (k+1)))) =$
 $(\sum k = 0..i. \text{the-enat}(\text{nlength}(\text{nnth } x22\ k))))$

using *NCons* **by** *auto*

have 5: *all-nfinite x22*

by (*simp add: NCons Suc.prem1*)

have 6: *i* ≤ *nlength x22*

using *NCons Suc.prem2 Suc-ile-eq* **by** *auto*

have 7: $(\sum k = 0..i. \text{the-enat}(\text{nlength}(\text{nnth } x22\ k))) =$
 $\text{the-enat}(\sum k = 0..i. (\text{nlength}(\text{nnth } x22\ k))))$

using *Suc.IH*[*of* *x22*] 5 6 **by** *blast*

have 8: $\text{the-enat}(\sum k = 0..i. (\text{nlength}(\text{nnth } x22\ k))) =$
 $\text{the-enat}(\sum k = 0..i. (\text{nlength}(\text{nnth nells } (k+1))))$

```

    using NCons by auto
  have 9: the-enat( $\sum k = 0..i. (nlength (nnth nells (k+1)))$ ) =
    the-enat( $\sum k = 1..Suc i. (nlength (nnth nells k))$ )
    using sum.shift-bounds-cl-nat-ivl[of  $\lambda k. (nlength (nnth nells k))$  0 1 i]
    by (metis One-nat-def Suc-eq-plus1)
  have 10: ( $\sum k = 1..Suc i. (nlength (nnth nells k))$ ) <  $\infty$ 
  proof -
    have 100: ( $\sum k = 1..Suc i. (nlength (nnth nells k))$ ) =
      ( $\sum k = 0..i. (nlength (nnth nells (k+1)))$ )
      using sum.shift-bounds-cl-nat-ivl[of  $\lambda k. (nlength (nnth nells k))$  0 1 i]
      by (metis One-nat-def Suc-eq-plus1)
    have 101: ( $\sum k = 0..i. (nlength (nnth nells (k+1)))$ ) =
      ( $\sum k = 0..i. (nlength (nnth x22 k))$ )
      using NCons by auto
    have 102: ( $\sum k = 0..i. (nlength (nnth x22 k))$ ) <  $\infty$ 
      using sum-finite[of x22 i] 5 6 by blast
    show ?thesis
      using 100 101 102 by presburger
  qed
  have 11:  $\exists m. (enat m) = (\sum k = 1..Suc i. (nlength (nnth nells k)))$ 
    by (metis 10 less-infinityE)
  obtain m where 12:  $(enat m) = (\sum k = 1..Suc i. (nlength (nnth nells k)))$ 
    using 11 by blast
  have 13:  $\exists n. (enat n) = (nlength (nnth nells 0))$ 
    by (metis Suc.premis(1) all-nfinite-nnth-b nfinite-nlength-enat zero-enat-def zero-le)
  obtain n where 14:  $(enat n) = (nlength (nnth nells 0))$ 
    using 13 by blast
  have 15: the-enat( $nlength (nnth nells 0)$ ) + the-enat( $\sum k = 1..Suc i. (nlength (nnth nells k))$ ) =
    the-enat( $nlength (nnth nells 0) + (\sum k = 1..Suc i. (nlength (nnth nells k)))$ )
    by (metis 12 14 plus-enat-simps(1) the-enat.simps)
  have 16: the-enat( $nlength (nnth nells 0)$ ) + the-enat( $\sum k = 1..Suc i. (nlength (nnth nells k))$ ) =
    the-enat( $\sum k = 0..Suc i. (nlength (nnth nells k))$ )
    using sum.atLeast-Suc-atMost[of 0 Suc i  $\lambda k. nlength (nnth nells k)$ ]
    by (metis 15 One-nat-def zero-le)
  show ?thesis
    using 1 16 3 4 7 8 9 by presburger
  qed
  qed
  qed

```

lemma *lsum-nnth-leq-Suc*:

assumes $i < nlength\ nells$

all-gr-zero nells

all-nfinite nells

$a < \infty$

shows $nnth (lsum\ nells\ a)\ i < nnth (lsum\ nells\ a)\ (Suc\ i)$

proof –

have 1: $nnth (lsum\ nells\ a)\ i = a + (\sum k::nat = 0..i. (the-enat (nlength (nnth nells k))))$

```

    using assms less-imp-le-nat lsum-nnth by (metis order-less-imp-le )
have 2: nnth (lsum nells a) (Suc i) = a + (∑ k::nat= 0..(Suc i). (the-enat (nlength(nnth nells k))))
    using assms Suc-ile-eq lsum-nnth by blast
have 3: (∑ k::nat= 0..(Suc i). (the-enat (nlength(nnth nells k)))) =
    (∑ k::nat= 0..i. (the-enat (nlength(nnth nells k)))) + (the-enat (nlength(nnth nells (Suc i))))
    using sum.atLeast0-atMost-Suc by blast
have 4: nlength(nnth nells (Suc i)) > 0
    using assms by (metis eSuc-enat ileI1 in-nset-conv-nnth)
have 5: (∑ k::nat= 0..i. (the-enat (nlength(nnth nells k)))) < ∞
    using lsum-nnth-nfinite[of i nells] assms order.order-iff-strict by blast
have 6: nlength(nnth nells (Suc i)) < ∞
    using assms
    by (metis all-nfinite-nnth-b eSuc-enat enat.distinct(2) enat-ord-simps(4) ileI1 nfinite-nlength-enat)
have 7: a + (∑ k::nat= 0..i. (the-enat (nlength(nnth nells k)))) <
    a + (∑ k::nat= 0..(Suc i). (the-enat (nlength(nnth nells k))))
    using 4 5 assms 6 enat-the-enat zero-enat-def by fastforce
show ?thesis
using 1 2 7 by presburger
qed

```

lemma *lsum-addzero-nnth-leq-Suc*:

```

assumes i < nlength(addzero (lsum nells 0))
    all-gr-zero nells
    all-nfinite nells
shows nnth (addzero (lsum nells 0)) i < nnth (addzero (lsum nells 0)) (Suc i)
using assms
proof (cases i)
case 0
then show ?thesis
    proof (cases nells)
    case (NNil x1)
    then show ?thesis using 0 assms unfolding addzero-def by simp
    (metis enat-0-iff(2) ndropn-nfirst ndropn-nlast nfinite-code(1) nfinite-nlength-enat nlast-NNil
    nlength-NNil nnth-nlast the-enat.simps)
    next
    case (NCons x21 x22)
    then show ?thesis using 0 assms unfolding addzero-def by simp
    (metis eSuc-infinity enat-the-enat gr-zeroI ndropn-eq-NNil ndropn-nlast not-eSuc-ilei0
    zero-enat-def)
    qed
    next
case (Suc nat)
then show ?thesis
    proof (cases nells)
    case (NNil x1)
    then show ?thesis
    using Suc addzero-def assms(1) enat-0-iff(1) by fastforce
    next
    case (NCons x21 x22)

```

```

then show ?thesis
proof -
  have 1:  $\text{nat} = 0 \implies ?thesis$ 
  using Suc assms unfolding addzero-def by auto
  (simp add: Suc-ile-eq lsum-nlength lsum-nnth-leq-Suc,
   metis Extended-Nat.eSuc-mono NCons One-nat-def assms(1) assms(2) enat-ord-code(4)
   lsum-addzero-NCons lsum-nlength lsum-nnth-leq-Suc nlength-NCons one-eSuc one-enat-def
   zero-enat-def)
  have 2:  $\text{nat} > 0 \implies ?thesis$ 
  using Suc assms unfolding addzero-def by auto
  (simp add: Suc-ile-eq lsum-nlength lsum-nnth-leq-Suc,
   metis NCons Suc-ile-eq assms(1) assms(2) enat-ord-code(4) iless-Suc-eq lsum-addzero-NCons
   lsum-nlength lsum-nnth-leq-Suc nlength-NCons)
  show ?thesis using 1 2 by blast
qed
qed
qed

```

```

lemma lsum-nidx:
assumes all-gr-zero nells
      all-nfinite nells
shows nidx (lsum nells a)
using assms unfolding nidx-expand
by (simp add: Suc-ile-eq lsum-nlength lsum-nnth-leq-Suc)

```

```

lemma lsum-addzero-nidx:
assumes all-gr-zero nells
      all-nfinite nells
shows nidx (addzero (lsum nells 0))
using assms unfolding nidx-expand
using Suc-ile-eq lsum-addzero-nnth-leq-Suc by blast

```

```

lemma pfilt-nfuse-lsum-a:
assumes nlast nell = nfirst nell1
      nlength nell > 0
      nlength nell1 > 0
      nfinite nell
      nfinite nell1
shows pfilt (nfuse nell nell1) (lsum (NNil nell1) (the-enat (nlength nell))) =
      pfilt nell1 (lsum (NNil nell1) 0)
proof -
  have 1: (pfilt (nfuse nell nell1) (lsum (NNil nell1) (the-enat (nlength nell)))) =
      nmap (nnth (nfuse nell nell1)) (lsum (NNil nell1) (the-enat (nlength nell)))
  unfolding pfilt-nmap by simp
  have 2: (pfilt (nell1) (lsum (NNil nell1) 0)) = nmap (nnth nell1) (lsum (NNil nell1) (0::nat))
  unfolding pfilt-nmap by simp
  have 3: nlength (nmap (nnth (nfuse nell nell1)) (lsum (NNil nell1) (the-enat (nlength nell)))) =
      nlength (nmap (nnth nell1) (lsum (NNil nell1) (0::nat)))

```

```

  by (simp add: lsum-nlength)
have 4:  $\bigwedge j. j \leq \text{nlength } (\text{nmap } (\text{nnth } \text{nell1}) (\text{lsum } (\text{NNil } \text{nell1}) (0::\text{nat}))) \longrightarrow$ 
   $\text{nnth } ((\text{nmap } (\text{nnth } (\text{nfuse } \text{nell } \text{nell1})) (\text{lsum } (\text{NNil } \text{nell1}) (\text{the-enat } (\text{nlength } \text{nell})))) j =$ 
   $\text{nnth } ((\text{nmap } (\text{nnth } \text{nell1}) (\text{lsum } (\text{NNil } \text{nell1}) (0::\text{nat})))) j$ 
  by (metis assms(1) assms(4) lsum-NNil ndropn-nfuse ndropn-nnth nellist.simps(14) plus-nat.add-0)
have 5:  $\text{nmap } (\text{nnth } (\text{nfuse } \text{nell } \text{nell1})) (\text{lsum } (\text{NNil } \text{nell1}) (\text{the-enat } (\text{nlength } \text{nell}))) =$ 
   $(\text{nmap } (\text{nnth } \text{nell1}) (\text{lsum } (\text{NNil } \text{nell1}) (0::\text{nat})))$ 
  using 3 4
  nellist-eq-nnth-eq[of nmap (nnth (nfuse nell nell1)) (lsum (NNil nell1) (the-enat (nlength nell))
    (nmap (nnth nell1) (lsum (NNil nell1) (0::nat))))]
  by presburger
show ?thesis using 5 1 2
by force
qed

```

lemma *pfilt-nfusecat-lsum-a:*

assumes *nlastnfirst* (*NCons* *nell* *nells*)

all-gr-zero *nells*

all-nfinite *nells*

nlength *nell* > 0

nfinite *nell*

shows $(\text{pfilt } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells})) (\text{lsum } \text{nells } (\text{the-enat } (\text{nlength } \text{nell})))) =$
 $(\text{pfilt } (\text{nfusecat } \text{nells}) (\text{lsum } \text{nells } 0))$

proof –

have 1: $(\text{pfilt } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells})) (\text{lsum } \text{nells } (\text{the-enat } (\text{nlength } \text{nell})))) =$
 $\text{nmap } (\text{nnth } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells}))) (\text{lsum } \text{nells } (\text{the-enat } (\text{nlength } \text{nell})))$

unfolding *pfilt-nmap* **by** *simp*

have 2: $(\text{pfilt } (\text{nfusecat } \text{nells}) (\text{lsum } \text{nells } 0)) = \text{nmap } (\text{nnth } (\text{nfusecat } \text{nells})) (\text{lsum } \text{nells } (0::\text{nat}))$

unfolding *pfilt-nmap* **by** *simp*

have 3: $\text{nlength } (\text{nmap } (\text{nnth } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells}))) (\text{lsum } \text{nells } (\text{the-enat } (\text{nlength } \text{nell})))) =$
 $\text{nlength } (\text{nmap } (\text{nnth } (\text{nfusecat } \text{nells})) (\text{lsum } \text{nells } (0::\text{nat})))$

by (*simp* *add: lsum-nlength*)

have 4: $\bigwedge j. (\text{enat } j) \leq \text{nlength } (\text{nmap } (\text{nnth } (\text{nfusecat } \text{nells})) (\text{lsum } \text{nells } (0::\text{nat}))) \implies$
 $\text{nnth } ((\text{nmap } (\text{nnth } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells}))) (\text{lsum } \text{nells } (\text{the-enat } (\text{nlength } \text{nell})))) j =$
 $\text{nnth } ((\text{nmap } (\text{nnth } (\text{nfusecat } \text{nells})) (\text{lsum } \text{nells } (0::\text{nat})))) j$

proof –

fix *j*

assume *a*: $(\text{enat } j) \leq \text{nlength } (\text{nmap } (\text{nnth } (\text{nfusecat } \text{nells})) (\text{lsum } \text{nells } (0::\text{nat})))$

show $\text{nnth } ((\text{nmap } (\text{nnth } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells}))) (\text{lsum } \text{nells } (\text{the-enat } (\text{nlength } \text{nell})))) j =$
 $\text{nnth } ((\text{nmap } (\text{nnth } (\text{nfusecat } \text{nells})) (\text{lsum } \text{nells } (0::\text{nat})))) j$

proof –

have 5: $\text{nnth } ((\text{nmap } (\text{nnth } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells}))) (\text{lsum } \text{nells } (\text{the-enat } (\text{nlength } \text{nell})))) j =$
 $\text{nnth } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells})) (\text{nnth } (\text{lsum } \text{nells } (\text{the-enat } (\text{nlength } \text{nell}))) j)$

using *nnth-nmap*[of *j* (*lsum* *nells* (*the-enat* (*nlength* *nell*))) (*nnth* (*nfuse* *nell* (*nfusecat* *nells*)))]

a 3 **by** *auto*

have 6: $(\text{nnth } (\text{lsum } \text{nells } (\text{the-enat } (\text{nlength } \text{nell}))) j) =$
 $\text{the-enat } (\text{nlength } \text{nell}) + (\sum k::\text{nat} = 0::\text{nat}..j. \text{the-enat } (\text{nlength } (\text{nnth } \text{nells } k)))$

using *lsum-nnth*[of *j* *nells* (*the-enat* (*nlength* *nell*))]

by (*metis* *a* *assms*(3) *lsum-nlength* *nlength-nmap*)

have 7: $\text{nnth } ((\text{nmap } (\text{nnth } (\text{nfusecat } \text{nells})) (\text{lsum } \text{nells } (0::\text{nat})))) j =$
 $\text{nnth } (\text{nfusecat } \text{nells}) (\text{nnth } (\text{lsum } \text{nells } (0::\text{nat})) j)$
using $\text{nnth-nmap}[\text{of } j (\text{lsum } \text{nells } (0::\text{nat})) (\text{nnth } (\text{nfusecat } \text{nells}))]$ **using** *a* **by** *auto*
have 8: $(\text{nnth } (\text{lsum } \text{nells } (0::\text{nat})) j) = (\sum k::\text{nat} = 0::\text{nat}..j. \text{the-enat } (\text{nlength } (\text{nnth } \text{nells } k)))$
by $(\text{metis } (\text{no-types, lifting}) 6 \text{ a add-0 add-diff-cancel-left' assms}(3) \text{ lsum-nlength}$
 $\text{lsum-nnth nlength-nmap})$
have 9: $\text{nlast } \text{nell} = \text{nfirst } (\text{nfusecat } \text{nells})$
by $(\text{metis } \text{assms}(1) \text{ nfirst-nfusecat-nfirst nlastnfirst-LCons})$
have 10: $(\text{the-enat } (\text{nlength } \text{nell}) + (\sum k::\text{nat} = 0::\text{nat}..j. \text{the-enat } (\text{nlength } (\text{nnth } \text{nells } k)))) \leq$
 $\text{nlength } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells}))$
proof $(\text{cases } \text{nfinite } \text{nells})$
case *True*
then show *?thesis*
proof –
have 101: $j \leq \text{nlength } \text{nells}$
by $(\text{metis } a \text{ lsum-nlength nlength-nmap})$
have 11: $\text{nlength } (\text{nfusecat } \text{nells}) =$
 $(\sum i::\text{nat} = 0::\text{nat}..\text{the-enat } (\text{nlength } \text{nells}). \text{nlength } (\text{nnth } \text{nells } i))$
using $\text{nfusecat-nlength-nfinite}[\text{of } \text{nells}]$
by $(\text{metis } \text{True } \text{assms}(1) \text{ assms}(3) \text{ nlastnfirst-LCons})$
have 12: $\text{nlength } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells})) =$
 $\text{nlength } \text{nell} + (\sum i::\text{nat} = 0::\text{nat}..\text{the-enat } (\text{nlength } \text{nells}). \text{nlength } (\text{nnth } \text{nells } i))$
using $\text{nfuse-nlength}[\text{of } \text{nell } \text{nfusecat } \text{nells}]$ 11 **by** *presburger*
have 13: $(\sum k::\text{nat} = 0::\text{nat}..j. \text{the-enat } (\text{nlength } (\text{nnth } \text{nells } k))) =$
 $(\text{the-enat } (\sum k::\text{nat} = 0::\text{nat}..j. (\text{nlength } (\text{nnth } \text{nells } k))))$
by $(\text{metis } 101 \text{ assms}(3) \text{ sum-the-enat})$
have 14: $j < \text{the-enat } (\text{nlength } \text{nells}) \implies$
 $(\sum k::\text{nat} = 0::\text{nat}..j. (\text{nlength } (\text{nnth } \text{nells } k))) \leq$
 $(\sum i::\text{nat} = 0::\text{nat}..\text{the-enat } (\text{nlength } \text{nells}). \text{nlength } (\text{nnth } \text{nells } i))$
by $(\text{metis } \text{bot-nat-0.extremum canonically-ordered-monoid-add-class.lessE}$
 $\text{enat-le-plus-same}(1) \text{ sum.ub-add-nat})$
have 141: $j = \text{the-enat } (\text{nlength } \text{nells}) \implies$
 $(\sum k::\text{nat} = 0::\text{nat}..j. (\text{nlength } (\text{nnth } \text{nells } k))) \leq$
 $(\sum i::\text{nat} = 0::\text{nat}..\text{the-enat } (\text{nlength } \text{nells}). \text{nlength } (\text{nnth } \text{nells } i))$
by *blast*
have 142: $(\sum k::\text{nat} = 0::\text{nat}..j. (\text{nlength } (\text{nnth } \text{nells } k))) \leq$
 $(\sum i::\text{nat} = 0::\text{nat}..\text{the-enat } (\text{nlength } \text{nells}). \text{nlength } (\text{nnth } \text{nells } i))$
using 14 141 101 *True* *nfinite-nlength-enat* **by** *fastforce*
have 143: $\text{enat } (\text{the-enat } (\text{nlength } \text{nell})) = \text{nlength } \text{nell}$
by $(\text{simp } \text{add: } \text{assms}(5) \text{ enat-the-enat nfinite-nlength-enat})$
have 144: $\text{enat } (\text{the-enat } (\sum k = 0..j. \text{nlength } (\text{nnth } \text{nells } k))) =$
 $(\sum k = 0..j. \text{nlength } (\text{nnth } \text{nells } k))$
by $(\text{metis } 11 142 \text{ True } \text{assms}(1) \text{ assms}(3) \text{ dual-order.refl enat-ord-code}(4)$
 $\text{enat-the-enat leD nfinite-nlength-enat nfusecat-nlength-nfinite}$
 $\text{nlastnfirst-LCons sum-finite})$
have 143: $\text{enat } (\text{the-enat } (\text{nlength } \text{nell}) + (\sum k = 0..j. \text{the-enat } (\text{nlength } (\text{nnth } \text{nells } k)))) =$
 $\text{nlength } \text{nell} + (\sum k = 0..j. \text{nlength } (\text{nnth } \text{nells } k))$
using 13
by $(\text{metis } 143 144 \text{ plus-enat-simps}(1))$
have 15: $(\text{the-enat } (\text{nlength } \text{nell}) + (\sum k::\text{nat} = 0::\text{nat}..j. \text{the-enat } (\text{nlength } (\text{nnth } \text{nells } k)))) \leq$

```

      nlength nell + (∑ i::nat = 0::nat..the-enat (nlength nells). nlength (nnth nells i))
    using 142 6 8 13 by (metis 143 add-left-mono)
  show ?thesis
    using 12 15 by presburger
  qed
next
case False
then show ?thesis
  proof -
    have 200: ¬nfinite (nfuse nell (nfusecat nells))
      using assms by (metis False nfuse-nfinite nfusecat-nfinite nlastnfirst-LCons)
    show ?thesis
      by (metis 200 linorder-le-cases nfinite-ntaken ntaken-all)
    qed
  qed
have 30: nnth (nfuse nell (nfusecat nells)) (the-enat (nlength nell) +
  (∑ k::nat = 0::nat..j. the-enat (nlength (nnth nells k)))) =
  nnth (nfusecat nells) (∑ k::nat = 0::nat..j. the-enat (nlength (nnth nells k)))
  by (metis 9 assms(5) ndropn-nfuse ndropn-nnth)
show ?thesis
  using 30 5 6 7 8 by presburger
qed
qed
show ?thesis
  by (metis 3 4 nellist-eq-nnth-eq pfilt-nmap)
qed

```

lemma *pfilt-nfusecat-lsum*:

assumes *nlastnfirst* (*NCons* *nell* *nells*)

all-gr-zero *nells*

all-nfinite *nells*

nlength *nell* > 0

nfinite *nell*

shows (*pfilt* (*nfusecat* (*NCons* *nell* *nells*)) (*addzero* (*lsum* (*NCons* *nell* *nells*) 0))) =
 (*NCons* (*nfist* *nell*) (*NCons* (*nlast* *nell*) (*pfilt* (*nfusecat* *nells*) (*lsum* *nells* 0))))

proof -

have 1: *nfusecat* (*NCons* *nell* *nells*) = *nfuse* *nell* (*nfusecat* *nells*)

by *simp*

have 2: (*pfilt* (*nfusecat* (*NCons* *nell* *nells*)) (*addzero* (*lsum* (*NCons* *nell* *nells*) 0))) =
 (*pfilt* (*nfuse* *nell* (*nfusecat* *nells*)) (*addzero* (*lsum* (*NCons* *nell* *nells*) 0)))

using 1 **by** *simp*

have 3: *addzero* (*lsum* (*NCons* *nell* *nells*) 0) =
 (*NCons* 0 (*NCons* (*the-enat* (*nlength* *nell*)) (*lsum* *nells* (*the-enat* (*nlength* *nell*))))))

using *lsum-addzero-NCons* **by** *auto*

have 4: (*pfilt* (*nfuse* *nell* (*nfusecat* *nells*)) (*addzero* (*lsum* (*NCons* *nell* *nells*) 0))) =
 (*pfilt* (*nfuse* *nell* (*nfusecat* *nells*)) (*NCons* 0 (*NCons* (*the-enat* (*nlength* *nell*))
 (*lsum* *nells* (*the-enat* (*nlength* *nell*)))))))

using 3 **by** *auto*

have 5: (*pfilt* (*nfuse* *nell* (*nfusecat* *nells*)) (*NCons* 0 (*NCons* (*the-enat* (*nlength* *nell*))
 (*lsum* *nells* (*the-enat* (*nlength* *nell*)))))) =

```

      (NCons (nnth (nfuse nell (nfusecat nells)) 0)
        (NCons (nnth (nfuse nell (nfusecat nells)) (the-enat (nlength nell)))
          (pfilt (nfuse nell (nfusecat nells)) (lsum nells (the-enat (nlength nell))))))
    by simp
  have 6: (nnth (nfuse nell (nfusecat nells)) 0) = (nnth nell 0)
    using assms
    by (metis nfirst-nfusecat-nfirst nfuse-nnth nlastnfirst-LCons zero-enat-def zero-le)
  have 7: (nnth (nfuse nell (nfusecat nells)) (the-enat (nlength nell))) =
    (nnth nell (the-enat (nlength nell)))
    using assms
    by (metis ndropn-nfirst ndropn-nfuse nfirst-nfusecat-nfirst nlastnfirst-LCons nnth-nlast)
  have 8: nidx (addzero (lsum nells 0))
    using assms lsum-addzero-nidx by blast
  have 9: (pfilt (nfuse nell (nfusecat nells)) (lsum nells (the-enat (nlength nell)))) =
    (pfilt (nfusecat nells) (lsum nells 0))
    using assms pfilt-nfusecat-lsum-a by blast
  show ?thesis
    using 3 6 7 9
    by (metis 2 5 assms(5) nlast-NNil nnth-nlast ntaken-0 ntaken-nlast)
qed

```

lemma *pfilt-nfusecat-lsum-1:*

```

assumes nlastnfirst (NCons nell nells)
  all-gr-zero nells
  all-nfinite nells
  nlength nell > 0
  nfinite nell
shows (pfilt (nfusecat (NCons nell nells)) ((lsum (NCons nell nells) 0))) =
  (NCons (nlast nell) (pfilt (nfusecat nells) (lsum nells 0)))
using assms
pfilt-nfusecat-lsum[of nell nells]
by (metis lsum-addzero-NCons nelist.inject(2) pfilt-code(2))

```

lemma *pfilt-nfusecat-lsum-2:*

```

assumes nlastnfirst (nells)
  all-gr-zero nells
  all-nfinite nells
  j ≤ nlength nells
shows (nnth (pfilt (nfusecat (nells)) ((lsum (nells) 0))) j) = nlast(nnth nells j)
proof —
  have 1: (pfilt (nfusecat (nells)) ((lsum (nells) 0))) =
    nmap (nnth (nfusecat nells)) (lsum nells (0::nat))
    using pfilt-nmap[of (nfusecat (nells)) (lsum (nells) 0)] by simp
  have 2: (nnth (pfilt (nfusecat (nells)) ((lsum (nells) 0))) j) =
    nnth (nfusecat nells) (nnth (lsum nells (0::nat)) j)
    using 1 nnth-nmap[of j ((lsum (nells) 0)) (nnth (nfusecat nells)) ]
    by (simp add: assms(4) lsum-nlength)
  have 3: (nnth (lsum nells (0::nat)) j) = (∑ k::nat = 0::nat..j. the-enat (nlength (nnth nells k)))
    by (metis add-0 assms(3) assms(4) lsum-nnth)
  have 4: (∑ k::nat = 0::nat..j. the-enat (nlength (nnth nells k))) =

```

```

      the-enat (( $\sum k::nat = 0::nat..j. (nlength (nnth nells k))$ )))
    by (metis assms(3) assms(4) sum-the-enat)
  have 40: nfinite (ntaken j nells)
    by simp
  have 41: nlastnfirst (ntaken j nells)
    by (simp add: assms(1) assms(4) nlastnfirst-ntaken)
  have 42: all-nfinite (ntaken j nells)
    by (metis assms(3) nset-ntaken subset-iff)
  have 5: nnth (nfusecat nells) (the-enat (( $\sum k::nat = 0::nat..j. (nlength (nnth nells k))$ ))) =
    nlast(nnth nells j)
  using nlastfirst-nfusecat-nlast[of ntaken j nells ] nfusecat-ntake[of j nells] assms
    nlastnfirst-ntaken[of j nells] ntake-eq-ntaken ntaken-nlast[of j nells]
  by (metis 3 4 40 42 enat-ord-simps(4) enat-the-enat ntaken-nlast sum-finite)
show ?thesis
using 2 3 4 5 by presburger
qed

```

lemma pfilt-nfusecat-lsum-3:

```

assumes nlastnfirst (nells)
  all-gr-zero nells
  all-nfinite nells
   $j \leq nlength (addzero (lsum nells 0))$ 
shows (  $j=0 \longrightarrow (nnth (pfilt (nfusecat (nells)) (addzero(lsum (nells) 0))) j) = nfirst(nfirst nells)$  )
   $\wedge$ 
  (  $j>0 \longrightarrow (nnth (pfilt (nfusecat (nells)) (addzero(lsum (nells) 0))) j) = nlast(nnth nells (j-1))$  )

```

using assms

proof –

```

  have 1:  $j \leq nlength (addzero (lsum nells 0)) \wedge j=0 \longrightarrow$ 
    (nnth (pfilt (nfusecat (nells)) (addzero(lsum (nells) 0))) j) = nfirst(nfirst nells)
  using assms
  by (metis lsum-addzero-nfirst nfinite-ntaken nfirst-nfusecat-nfirst nnth-NNil nnth-nlast
    ntaken-0 ntaken-nlast pfilt-nlength pfilt-nnth)
  have 2:  $j \leq nlength (addzero (lsum nells 0)) \wedge j>0 \longrightarrow$ 
    (nnth (pfilt (nfusecat nells) (addzero(lsum (nells) 0))) j) =
    (nnth (nfusecat nells) (nnth (addzero(lsum (nells) 0)) j))
  by (simp add: pfilt-nlength pfilt-nnth)
  have 3:  $j \leq nlength (addzero (lsum nells 0)) \wedge j>0 \longrightarrow$ 
    nnth (addzero(lsum (nells) 0)) j = nnth (lsum nells 0) (j-1)
  by (metis Suc-diff-1 addzero-def enat-0-iff(1) le-zero-eq nat-less-le nnth-Suc-NCons)
  have 20:  $j \leq nlength (addzero (lsum nells 0)) \wedge j>0 \longrightarrow enat ((j::nat) - (1::nat)) \leq nlength nells$ 
  by (metis Suc-diff-1 Suc-ile-eq addzero-def dual-order.strict-trans1 enat-0-iff(1) iless-Suc-eq
    lsum-nlength nlength-NCons not-gr-zero)
  have 4:  $j \leq nlength (addzero (lsum nells 0)) \wedge j>0 \longrightarrow$ 
    (nnth (nfusecat nells) (nnth (lsum nells 0) (j-1))) = nlast(nnth nells (j-1))
  using assms 20 pfilt-nfusecat-lsum-2[of nells j-1]
  by (metis lsum-nlength pfilt-expand)
show ?thesis

```

```

using 1 2 3 4
by (simp add: assms(4))
qed

```

lemma *pfilt-nfusecat-lsum-4:*

```

assumes nlastnfirst nells
        all-gr-zero nells
        all-nfinite nells
        nfinite nells

```

shows $(\text{nnth } (\text{addzero } (\text{lsum } \text{nells } 0)) (\text{the-enat } (\text{nlength } (\text{addzero } (\text{lsum } \text{nells } 0))))) \leq \text{nlength } (\text{nfusecat } \text{nells})$

proof –

have 2: $\text{nlength } (\text{nfusecat } \text{nells}) = (\sum i = 0.. \text{the-enat } (\text{nlength } \text{nells}). \text{nlength } (\text{nnth } \text{nells } i))$

using *assms nfusecat-nlength-nfinite* **by** *blast*

have 3: $\text{nlast } (\text{lsum } \text{nells } 0) =$

$(\sum k = 0.. \text{the-enat } (\text{nlength } \text{nells}). \text{the-enat } (\text{nlength } (\text{nnth } \text{nells } k)))$

by *(simp add: assms(3) assms(4) lsum-nlast)*

have 4: $(\sum k = 0.. \text{the-enat } (\text{nlength } \text{nells}). \text{the-enat } (\text{nlength } (\text{nnth } \text{nells } k))) =$
 $\text{the-enat } (\sum k = 0.. \text{the-enat } (\text{nlength } \text{nells}). (\text{nlength } (\text{nnth } \text{nells } k)))$

by *(simp add: assms(3) assms(4) enat-the-enat nfinite-nlength-enat sum-the-enat)*

have 5: $\text{enat } (\sum k = 0.. \text{the-enat } (\text{nlength } \text{nells}). \text{the-enat } (\text{nlength } (\text{nnth } \text{nells } k))) \leq$
 $(\sum i = 0.. \text{the-enat } (\text{nlength } \text{nells}). \text{nlength } (\text{nnth } \text{nells } i))$

using *assms* **by** *(metis 4 dual-order.refl enat-ord-code(3) enat-the-enat)*

have 6: $\text{nlast } (\text{addzero } (\text{lsum } \text{nells } 0)) \leq \text{nlength } (\text{nfusecat } \text{nells})$

unfolding *addzero-def* **using** 3 2 5

by *simp*

have 7: $\text{nlast } (\text{addzero } (\text{lsum } \text{nells } 0)) =$

$(\text{nnth } (\text{addzero } (\text{lsum } \text{nells } 0)) (\text{the-enat } (\text{nlength } (\text{addzero } (\text{lsum } \text{nells } 0)))))$

by *(simp add: addzero-def assms(4) nfinite-lsum-conv-a nnth-nlast)*

show *?thesis* **using** *assms* 6 7 **by** *auto*

qed

lemma *nfusecat-nlength-b:*

assumes *nlastnfirst nells*

all-nfinite nells

$1 \leq i$

$i \leq \text{nlength } \text{nells}$

$j \leq \text{nlength}(\text{nnth } \text{nells } i)$

nfinite nells

all-gr-zero nells

shows $(\text{nnth } ((\text{lsum } \text{nells } 0)) (i-1) + j \leq \text{nlength } (\text{nfusecat } \text{nells}))$

proof –

have 0: $i-1 \leq \text{nlength } \text{nells}$

by *(metis Suc-ile-eq assms(3) assms(4) le-add-diff-inverse order-less-imp-le plus-1-eq-Suc)*

have 1: $(\text{nnth } ((\text{lsum } \text{nells } 0)) (i-1) = (\sum k::\text{nat} = 0..(i-1). \text{the-enat}(\text{nlength}(\text{nnth } \text{nells } k))))$

using 0 *lsum-nnth[of i-1 nells 0]* *assms* **by** *presburger*

have 2: $\text{nlength } (\text{nfusecat } \text{nells}) = (\sum k::\text{nat} = 0..(\text{the-enat}(\text{nlength } \text{nells})). \text{nlength}(\text{nnth } \text{nells } k))$

using *assms nfusecat-nlength-nfinite* **by** *metis*

have 20: $(\sum k::\text{nat} = 0..(i). \text{the-enat}(\text{nlength}(\text{nnth } \text{nells } k))) =$

$(\sum k = 0..i - 1. \text{the-enat} (\text{nlength} (\text{nnth nells } k))) +$
 $\text{the-enat} (\text{nlength} (\text{nnth nells} (\text{Suc } (i - 1))))$
using *assms sum.atLeast0-atMost-Suc*[of $\lambda k. \text{the-enat}(\text{nlength}(\text{nnth nells } k)) \ i - 1$] **by** *simp*
have 21: $(\sum k = 0..i - 1. \text{the-enat} (\text{nlength} (\text{nnth nells } k))) < \infty$
using *enat-ord-code(4)* **by** *blast*
have 22: $\text{the-enat} (\text{nlength} (\text{nnth nells} (\text{Suc } (i - 1)))) < \infty$
by *auto*
have 23: $(\sum k::\text{nat} = 0..(i). \text{the-enat}(\text{nlength}(\text{nnth nells } k))) < \infty$
using *enat-ord-simps(4)* **by** *blast*
have 3: $(\sum k::\text{nat} = 0..(i-1). \text{the-enat}(\text{nlength}(\text{nnth nells } k))) + j \leq$
 $(\sum k::\text{nat} = 0..(i). \text{the-enat}(\text{nlength}(\text{nnth nells } k)))$
using 20 *assms* **by** *simp*
 $(\text{metis } (\text{no-types, lifting}) \ \text{all-nfinite-nnth-b} \ \text{enat-ord-simps}(1) \ \text{enat-the-enat infinity-ileE}$
 $\ \text{ndropn-eq-NNil ndropn-nlast})$
have 4: $(\sum k::\text{nat} = 0..(i). \text{the-enat}(\text{nlength}(\text{nnth nells } k))) \leq$
 $(\sum k::\text{nat} = 0..(\text{the-enat}(\text{nlength nells})). \text{the-enat}(\text{nlength}(\text{nnth nells } k)))$
using *sum.ub-add-nat*[of 0 $i \ \lambda k. \text{the-enat}(\text{nlength}(\text{nnth nells } k)) \ (\text{the-enat}(\text{nlength nells})) - i$]
using *assms(4) assms(6) nfinite-nlength-enat* **by** *fastforce*
have 5: $(\sum k::\text{nat} = 0..(\text{the-enat}(\text{nlength nells})). \text{the-enat}(\text{nlength}(\text{nnth nells } k))) \leq$
 $(\sum k::\text{nat} = 0..(\text{the-enat}(\text{nlength nells})). \text{nlength}(\text{nnth nells } k))$
using *sum-the-enat*[of *nells* $(\text{the-enat}(\text{nlength nells}))$] *assms*
by $(\text{metis } \text{leI less-enatE order-less-imp-not-eq the-enat.simps})$
show ?thesis
using 1 2 3 4 5 **by** *simp*
 $(\text{meson dual-order.trans enat-ord-simps}(1))$
qed

lemma *nfusecat-nlength-b-alt:*

assumes *nlastnfirst nells*
all-nfinite nells
 $1 \leq i$
 $i \leq \text{nlength nells}$
 $j \leq \text{nlength}(\text{nnth nells } i)$
 $\neg \text{nfinite nells}$
all-gr-zero nells
shows $\text{nnth } ((\text{lsum nells } 0)) \ (i - 1) + j \leq \text{nlength } (\text{nfusecat nells})$

proof –

have 1: $\neg \text{nfinite } (\text{nfusecat nells})$
using *assms* **by** $(\text{metis } \text{nfusecat-nfinite})$
show ?thesis
by $(\text{meson } 1 \ \text{enat-ile linorder-le-cases nfinite-conv-nlength-enat})$
qed

lemma *pfilt-nfusecat-lsum-5:*

assumes *nlastnfirst nells*
all-gr-zero nells
all-nfinite nells
 $(\text{enat } i) \leq \text{nlength } (\text{addzero } (\text{lsum nells } 0))$
nfinite nells
shows $(\text{nnth } (\text{addzero } (\text{lsum nells } 0)) \ i) \leq \text{nlength } (\text{nfusecat nells})$

```

using assms
proof -
  have 2:  $\text{nlength} (\text{nfusecat } \text{nells}) = (\sum i = 0.. \text{the-enat } (\text{nlength } \text{nells}). \text{nlength} (\text{nnth } \text{nells } i))$ 
    using assms  $\text{nfusecat-nlength-nfinite}$  by blast
  have 3:  $\text{nlength } \text{nells} > 0 \implies \text{nlength} (\text{lsum } \text{nells } 0) > 0$ 
    by (simp add:  $\text{lsum-nlength}$ )
  have 4:  $\text{nlength } \text{nells} > 0 \implies (\text{nnth} (\text{addzero} (\text{lsum } \text{nells } 0)) i) = \text{nnth} (\text{NCons } 0 (\text{lsum } \text{nells } 0)) i$ 
    using assms unfolding  $\text{addzero-def}$  using 3 by simp
  have 5:  $\text{nlength } \text{nells} > 0 \wedge i=0 \implies (\text{nnth} (\text{addzero} (\text{lsum } \text{nells } 0)) i) = 0$ 
    using 4 by auto
  have 6:  $\text{nlength } \text{nells} > 0 \wedge i>0 \implies (\text{nnth} (\text{addzero} (\text{lsum } \text{nells } 0)) i) = \text{nnth} (\text{lsum } \text{nells } 0) (i - 1)$ 
    by (metis 4  $\text{Suc-diff-1}$   $\text{nnth-Suc-NCons}$ )
  have 7:  $\text{nlength } \text{nells} > 0 \wedge i>0 \implies \text{nnth} (\text{lsum } \text{nells } 0) (i - 1) =$ 
     $(\sum k = 0..(i-1). \text{the-enat} (\text{nlength} (\text{nnth } \text{nells } k)))$ 
    using assms  $\text{lsum-nnth}[of\ i-1\ \text{nells } 0]$ 
    by (metis  $\text{Suc-diff-1}$   $\text{Suc-ile-eq}$   $\text{add-0}$   $\text{addzero-def}$   $i0-less$   $i\text{less-Suc-eq}$   $\text{lsum-nlength}$   $\text{nlength-NCons}$ )
  have 8:  $\text{nlength } \text{nells} > 0 \wedge i>0 \implies$ 
     $(\sum k = 0..(i-1). \text{the-enat} (\text{nlength} (\text{nnth } \text{nells } k))) =$ 
     $\text{the-enat}(\sum k = 0..(i-1). (\text{nlength} (\text{nnth } \text{nells } k)))$ 
    using assms  $\text{sum-the-enat}[of\ \text{nells } i-1\ ]$ 
    by (metis  $\text{Suc-diff-1}$   $\text{Suc-ile-eq}$   $\text{addzero-def}$   $i0-less$   $i\text{less-Suc-eq}$   $\text{lsum-nlength}$   $\text{nlength-NCons}$ )
  have 9:  $\text{nlength } \text{nells} > 0 \wedge i>0 \implies \text{nnth} (\text{lsum } \text{nells } 0) (i - 1) \leq \text{nlength} (\text{nfusecat } \text{nells})$ 
    using assms  $\text{nfusecat-nlength-b}[of\ \text{nells } i\ 0]$   $\text{pfilt-nfusecat-lsum-4}[of\ \text{nells}]$ 
    3 4 6  $\text{lsum-nlength}[of\ \text{nells}]$  unfolding  $\text{addzero-def}$  by simp
    (metis  $i\text{less-Suc-eq}$   $\text{order.order-iff-strict}$   $\text{the-enat.simps}$   $\text{zero-enat-def}$   $\text{zero-le}$ )
  have 10:  $\text{nlength } \text{nells} = 0 \wedge i = 0 \implies \text{nnth} (\text{NCons } 0 (\text{lsum } \text{nells } 0)) i \leq \text{nlength} (\text{nfusecat } \text{nells})$ 
    using assms  $\text{zero-enat-def}$  by auto
  have 11:  $\text{nlength } \text{nells} = 0 \wedge i = 1 \implies \text{nnth} (\text{NCons } 0 (\text{lsum } \text{nells } 0)) i \leq \text{nlength} (\text{nfusecat } \text{nells})$ 
    using assms
    by (metis  $\text{addzero-def}$   $\text{lsum-nlength}$   $\text{nlength-NCons}$   $\text{not-one-le-zero}$   $\text{one-eSuc}$   $\text{one-enat-def}$   $\text{pfilt-nfusecat-lsum-4}$   $\text{the-enat.simps}$ )
  have 12:  $\text{nfinite} (\text{nfirst } \text{nells})$ 
    by (metis assms(3) assms(5)  $\text{nconcat-expand}$   $\text{nfinite-nappend}$   $\text{nfinite-nconcat}$ )
  have 13:  $\text{nlength} (\text{nfirst } \text{nells}) > 0$ 
    using assms
    by (metis  $\text{in-nset-conv-nnth}$   $\text{nlast-NNil}$   $\text{ntaken-0}$   $\text{ntaken-nlast}$   $\text{zero-enat-def}$   $\text{zero-le}$ )
  have 14:  $\text{nlength } \text{nells} = 0 \implies i \leq 1$ 
    using assms  $\text{lsum-addzero-nlength}[of\ \text{nells}]$ 
    by (metis 12 13  $\text{enat-ord-simps}(1)$   $\text{one-enat-def}$ )
  have 15:  $\text{nnth} (\text{NCons } 0 (\text{lsum } \text{nells } 0)) i \leq \text{nlength} (\text{nfusecat } \text{nells})$ 
    by (metis 10 11 14 4 5 6 9  $\text{One-nat-def}$   $\text{Suc-leI}$   $\text{dual-order.antisym}$   $\text{not-gr-zero}$   $\text{zero-enat-def}$   $\text{zero-le}$ )
  show ?thesis
  by (metis 12 13 15 assms(4)  $\text{gr-zeroI}$   $\text{lsum-addzero-nnth}$ )
qed

lemma  $\text{pfilt-nfusecat-lsum-5-alt}$ :
  assumes  $\text{nlastnfirst} (\text{nells})$ 
     $\text{all-gr-zero } \text{nells}$ 

```

```

    all-nfinite nells
    (enat i) ≤ nlength (addzero (lsum nells 0))
  ¬nfinite nells
shows (nnth (addzero (lsum nells 0)) i) ≤ nlength (nfusecat nells)
using assms
by (metis enat-ile linorder-le-cases nfinite-conv-nlength-enat nfusecat-nfinite)

```

```

lemma lsum-shift:
assumes nlastnfirst nells
    all-gr-zero nells
    all-nfinite nells
     $i \leq nlength\ nells$ 
shows  $nnth\ (lsum\ nells\ a)\ i = a + nnth\ (lsum\ nells\ 0)\ i$ 
using assms by (simp add: lsum-nnth)

```

```

lemma lsum-nfusecat-nnth-lsum-nnth:
assumes nlastnfirst nells
    all-gr-zero nells
    all-nfinite nells
     $i \leq nlength\ nells$ 
     $j \leq nlength\ (nnth\ nells\ i)$ 
shows  $(nnth\ (nfusecat\ nells)\ ((nnth\ (addzero\ (lsum\ nells\ 0))\ i) + j)) = (nnth\ (nnth\ nells\ i)\ j)$ 
using assms
proof (induction i arbitrary: nells j)
case 0
then show ?case
  proof (cases nells)
  case (NNil x1)
    then show ?thesis using 0 by simp
    (metis add-cancel-right-left addzero-def ndropn-nfirst ndropn-nlast nfinite-NNil nlast-NNil
      nnth-0 nnth-NNil)
  next
  case (NCons x21 x22)
    then show ?thesis using 0 by simp
    (metis 0.premis(1) 0.premis(2) 0.premis(3) add-cancel-right-left addzero-def eSuc-ne-0
      nfirst-nfusecat-nfirst nfuse-nnth nfusecat-NCons nfusecat-nlength-a nlength-NCons nnth-0)
  qed
next
case (Suc i)
then show ?case
  proof (cases nells)
  case (NNil x1)
    then show ?thesis using Suc by (simp add: zero-enat-def)
  next
  case (NCons x21 x22)
    then show ?thesis
    proof –
      have 0:  $(0::enat) < nlength\ nells$ 

```



```

    by (simp add: NCons)
have 1: nlastnfirst x22
    using NCons Suc.premis(1) by auto
have 2: all-gr-zero x22
    using NCons Suc.premis(2) by auto
have 3: all-nfinite x22
    using NCons Suc.premis(3) by auto
have 4: enat (i::nat) ≤ nlength x22
    using NCons Suc.premis(4) Suc-ile-eq by auto
have 5: nnth (nfusecat nells) (nnth (addzero (lsum nells (0::nat))) (Suc i) + j) =
    nnth (nfuse x21 (nfusecat x22)) (nnth (addzero (lsum nells (0::nat))) (Suc i) + j)
    by (simp add: NCons)
have 6: nlength (addzero (lsum nells (0::nat))) = nlength nells + (1::enat)
    using lsum-addzero-nlength[of nells ]
    by (metis NCons lsum-addzero-NCons lsum-nlength nlength-NCons plus-1-eSuc(2))
have 7: enat (Suc (i::nat) ) ≤ nlength (addzero (lsum (nells::'a nelist) (0::nat)))
    by (simp add: 6 Suc.premis(4) order-less-imp-le plus-1-eSuc(2))
have 8: (nnth (addzero (lsum nells (0::nat))) (Suc i)) =
    nnth (NCons (0::nat) (lsum nells (0::nat))) (Suc i)
    using lsum-addzero-nnth[of (Suc i) nells ] by (metis NCons lsum-addzero-NCons)
have 9: nnth (NCons (0::nat) (lsum nells (0::nat))) (Suc i) = nnth (lsum nells 0) i
    by simp
have 10: nnth (lsum nells 0) i = (∑ k::nat = 0::nat..i. the-enat (nlength (nnth nells k)))
    using lsum-nnth[of i nells 0]
    by (metis Suc.premis(3) Suc.premis(4) Suc-ile-eq add-0 order-less-imp-le)
have 11: nlength x21 ≤ (∑ k::nat = 0::nat..i. the-enat (nlength (nnth nells k))) + j
    proof (cases i)
    case 0
    then show ?thesis using NCons by simp
    (metis Suc.premis(3) enat-ord-simps(1) enat-the-enat infinity-ileE le-add1 ndropn-eq-NNil
    ndropn-nlast nelist.set-intros(2))
    next
    case (Suc nat)
    then show ?thesis
    proof -
    have 12: (∑ k::nat = 0::nat..i. the-enat (nlength (nnth nells k))) =
        the-enat (nlength (nnth nells 0)) +
        (∑ k::nat = 1::nat..i. the-enat (nlength (nnth nells k)))
        by (simp add: sum.atLeast-Suc-atMost)
    have 13: (nlength (nnth nells 0)) = nlength x21
        by (simp add: NCons)
    have 14: nlength x21 ≤
        the-enat (nlength x21) +
        (∑ k::nat = 1::nat..i. the-enat (nlength (nnth nells k))) + j
        using NCons Suc.premis(3) nfinite-nlength-enat by fastforce
    show ?thesis
        using 12 13 14 by presburger
    qed
    qed
have 15: nlast x21 = nfirst (nfusecat x22)

```

```

  by (metis NCons Suc.prem1 nfirst-nfusecat-nfirst nlastnfirst-LCons)
have 16: enat (nnth (addzero (lsum nells 0)) (Suc i) + j) ≤ nlength (nfusecat x21 (nfusecat x22))
  by (metis 8 9 NCons Suc.prem1 Suc.prem2 Suc.prem3 Suc.prem4 Suc.prem5
    add-diff-cancel-left' le-add1 nfusecat-NCons nfusecat-nlength-b nfusecat-nlength-b-alt
    plus-1-eq-Suc)
have 17: nlength x21 ≤ enat (nnth (addzero (lsum nells (0::nat))) (Suc i) + j)
  using 10 11 8 9 by presburger
have 18: nnth (nfusecat x21 (nfusecat x22)) (nnth (addzero (lsum nells (0::nat))) (Suc i) + j) =
  nnth (nfusecat x22) (nnth (addzero (lsum nells (0::nat))) (Suc i) + j - (the-enat(nlength x21)))
  using nfuse-nnth-var[of (nnth (addzero (lsum nells (0::nat))) (Suc i) + j) x21 nfusecat x22]
  using 15 16 17 by blast
have 19: i=0 ⇒ nnth (addzero (lsum nells (0::nat))) (Suc i) = (the-enat(nlength x21))
  using 8 NCons by auto
have 20: i=0 ⇒ nnth (nfusecat x22) 0 = nnth (nnth x22 i) 0
  using Suc.IH[of x22 0]
  by (metis 1 2 3 4 add.right-neutral all-gr-zero-nnth-b lsum-addzero-NCons
    lsum-addzero-nfirst-0 ntaken-0 ntaken-nlast order-less-imp-le zero-enat-def)
have 21: i=0 ⇒ ?thesis
  by (metis 1 18 19 2 3 4 5 NCons Suc.IH Suc.prem5 add commute add.right-neutral
    add-diff-cancel-left' lsum-addzero-NCons lsum-addzero-nfirst nnth-0 nnth-Suc-NCons ntaken-0
    ntaken-nlast)
have 22: 0 < i ⇒ (∑ k::nat = 0::nat..i. the-enat (nlength (nnth nells k))) =
  (the-enat(nlength x21)) + (∑ k::nat = 0::nat..(i-1). the-enat (nlength (nnth x22 k)))
proof -
  assume a1: 0 < i
  have 23: (∑ k::nat = 0::nat..i. the-enat (nlength (nnth nells k))) =
    the-enat(nlength x21) + (∑ k::nat = 1::nat..i. the-enat (nlength (nnth nells k)))
  by (simp add: NCons sum.atLeast-Suc-atMost)
  have 24: (∑ k::nat = 1::nat..i. the-enat (nlength (nnth nells k))) =
    (∑ k::nat = 0::nat..(i-1). the-enat (nlength (nnth nells (k+1))))
  using sum.shift-bounds-cl-nat-ivl[of λk .the-enat (nlength (nnth nells k)) 0 1 i-1]
  by (simp add: a1)
  have 25: (∑ k::nat = 0::nat..(i-1). the-enat (nlength (nnth nells (k+1)))) =
    (∑ k::nat = 0::nat..(i-1). the-enat (nlength (nnth x22 (k))))
  using NCons by auto
  show ?thesis
  using 23 24 25 by presburger
qed
have 26: 0 < i ⇒ nnth (addzero (lsum nells (0::nat))) (Suc i) - (the-enat(nlength x21)) =
  (∑ k::nat = 0::nat..(i-1). the-enat (nlength (nnth x22 (k))))
  by (simp add: 10 22 8)
have 260: 0 < nfirst (lsum x22 0)
proof (cases x22)
case (NNil x1)
then show ?thesis by simp
(metis 2 3 NNil-eq-ntake-iff enat-the-enat grOI infinity-ileE less-numeral-extra(3)
  ndropn-eq-NNil ndropn-nlast nelist.set-intros(1) nlast-NNil zero-enat-def)
next
case (NCons x21 x22)
then show ?thesis by simp

```

```

    (metis 2 3 grOI less-numeral-extra(3) ndropn-nfirst ndropn-nfuse ndropn-nlast
    nellist.set-intros(2) nfinite-NNil nfuse-leftneutral nlast-NNil nlength-NNil nnth-0 the-enat-0)
  qed
  have 27:  $0 < i \implies (\sum k::nat = 0::nat..(i-1). the-enat (nlength (nnth x22 (k)))) =$ 
     $nnth (addzero (lsum x22 0)) (i)$ 
    unfolding addzero-def using NCons 26 3 4 8 260 apply simp
    using lsum-nnth
    by (metis Suc-ile-eq Suc-pred add-0 nnth-Suc-NCons order-less-imp-le)
  have 28:  $0 < i \implies$ 
     $nnth (nfusecat x22) (nnth (addzero (lsum nells (0::nat))) (Suc i) + j - (the-enat(nlength x21)))$ 
  =
     $nnth (nnth x22 i) j$ 
    using Suc.IH[of x22 j]
    by (metis 1 10 2 22 26 27 3 4 8 NCons Nat.add-diff-assoc2 Suc.prem5)
    le-add1 nnth-Suc-NCons)
  have 29:  $0 < i \implies ?thesis$ 
    using 18 28 NCons by fastforce
  show ?thesis
    using 21 29 by blast
  qed
  qed
  qed

```

lemma *lcppl-lsum-less-th-equal:*

```

  assumes nidx ls
    nnth ls 0 = 0
    nfinite ls
    nfinite  $\sigma$ 
    nlast ls = (the-enat (nlength  $\sigma$ ))
     $(\forall i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) \models f\ fproj\ g)$ 
    nlength  $\sigma > 0$ 
     $i < nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))$ 
  shows  $(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i)) \leq$ 
     $nlength\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls))$ 
  proof -
    have 1: nlastnfirst (lcppl f g  $\sigma$  ls)
      using assms
    by (metis enat.distinct(2) enat-the-enat i0-less lcppl-nfusecat-nlastnfirst
    nfinite-conv-nlength-enat nnth-nlast the-enat-0)
  have 2: all-gr-zero (lcppl f g  $\sigma$  ls)
    using assms all-gr-zero-nnth-a lcppl-nlength-all-gr-zero by blast
  have 3: all-nfinite (lcppl f g  $\sigma$  ls)
    using assms all-nfinite-nnth-a lcppl-nlength-all-nfinite by blast
  have 4:  $enat\ (Suc\ (i::nat)) \leq nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ (0::nat)))$ 
    using assms using Suc-ile-eq by blast
  have 5: nfinite (lcppl f g  $\sigma$  ls)
    using assms using lcppl-nfinite by blast
  show ?thesis using
    pfilt-nfusecat-lsum-5[of (lcppl f g  $\sigma$  ls) Suc i ]

```

using 1 2 3 4 5 by blast
qed

lemma *lcppl-lsum-less-th-equal-alt*:

assumes *nidx ls*
 $nnth\ ls\ 0 = 0$
 $\neg nfinite\ ls$
 $\neg nfinite\ \sigma$
 $(\forall\ i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$
 $i < nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))$
shows $(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i)) \leq$
 $nlength\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls))$

proof –

have 1: *nlastnfirst* (*lcppl f g σ ls*)
 using *assms*
 using *lcppl-nfusecat-nlastnfirst-alt* by blast
have 2: *all-gr-zero* (*lcppl f g σ ls*)
 using *assms all-gr-zero-nnth-a lcppl-nlength-all-gr-zero-alt* by blast
have 3: *all-nfinite* (*lcppl f g σ ls*)
 using *assms all-nfinite-nnth-a lcppl-nlength-all-nfinite-alt* by blast
have 4: $enat\ (Suc\ (i::nat)) \leq nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ (0::nat)))$
 using *assms using Suc-ile-eq* by blast
show ?thesis
 using 1 2 3 4 *pfilt-nfusecat-lsum-5-alt* using *assms(1) assms(3) lcppl-nfinite* by blast
 qed

lemma *lcppl-lsum-nlength*:

assumes *nidx ls*
 $nnth\ ls\ 0 = 0$
 $nfinite\ ls$
 $nfinite\ \sigma$
 $nlast\ ls = (the-enat\ (nlength\ \sigma))$
 $(\forall\ i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$
 $nlength\ \sigma > 0$
shows $nlength\ (addzero\ (lsum\ ((lcppl\ f\ g\ \sigma\ ls)\ 0)) = nlength\ ls$
proof –
have 1: $nlength\ \sigma > 0 \longrightarrow nlength\ (lcppl\ f\ g\ \sigma\ ls) = nlength\ ls - 1$
 using *assms*
 by (*metis epred-0 epred-conv-minus i0-less lcppl-nlength lcppl-nlength-zero*)
have 2: $nlength\ \sigma > 0 \longrightarrow nlength\ ls > 0$
 using *assms*
 by (*metis enat.distinct(2) enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have 3: $nlength\ \sigma > 0 \longrightarrow nlength\ (nfirst\ (lcppl\ f\ g\ \sigma\ ls)) > 0$
 using *assms*
 by (*metis lcppl-nlength-all-gr-zero ndropn-0 ndropn-nfirst nfinite-nlength-enat the-enat.simps zero-enat-def zero-le*)
have 30: *nlastnfirst* (*lcppl f g σ ls*)
 using 2 *assms lcppl-nfusecat-nlastnfirst* by blast
have 31: *nfinite* (*nfirst* (*lcppl f g σ ls*))

```

using assms
by (metis all-nfinite-nnth-a lcppl-nfinite lcppl-nlength-all-nfinite nconcat-expand
      nfinite-nappend nfinite-nconcat)
have 4:  $nlength\ \sigma > 0 \longrightarrow nlength\ (lcppl\ f\ g\ \sigma\ ls) = 0 \longrightarrow$ 
       $nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) = nlength\ ls$ 
using 1 2 3 assms lsum-addzero-nlength[of (lcppl f g  $\sigma$  ls) ] 30 31
by (metis co.enat.exhaust-sel i0-less lcppl-nlength one-eSuc)
have 5:  $nlength\ \sigma > 0 \longrightarrow nlength\ (lcppl\ f\ g\ \sigma\ ls) > 0 \longrightarrow$ 
       $nlength\ (addzero\ (lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0)) = nlength\ ls$ 
using assms
by (metis 2 4 addzero-def co.enat.exhaust-sel lcppl-nlength lcppl-nlength-zero lsum-nlength
      nlength-NCons)
show ?thesis using 4 5
using assms(7) gr-zeroI by blast
qed

```

lemma *lcppl-lsum-nlength-alt*:

```

assumes nidx ls
       $nnth\ ls\ 0 = 0$ 
       $\neg nfinite\ ls$ 
       $\neg nfinite\ \sigma$ 
       $(\forall\ i < nlength\ ls.\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$ 
shows  $nlength\ (addzero\ (lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0)) = nlength\ ls$ 
proof –
have 1:  $nlength\ (lcppl\ f\ g\ \sigma\ ls) = nlength\ ls - 1$ 
      using assms
      by (simp add: epred-conv-minus lcppl-nlength-alt)
have 2:  $nlength\ ls > 0$ 
      using assms
      by (simp add: nfinite-conv-nlength-enat)
have 3:  $nlength\ (nfirst\ (lcppl\ f\ g\ \sigma\ ls)) > 0$ 
      using assms lcppl-nlength-all-gr-zero-alt[of ls  $\sigma$  f g]
      by (metis ndropn-0 ndropn-nfirst zero-enat-def zero-le)
have 4:  $nlength\ (lcppl\ f\ g\ \sigma\ ls) = 0 \longrightarrow$ 
       $nlength\ (addzero\ (lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0)) = nlength\ ls$ 
      using 1 2 3 assms
      by (metis lcppl-nfinite nlength-eq-enat-nfiniteD zero-enat-def)
have 5:  $nlength\ (lcppl\ f\ g\ \sigma\ ls) > 0 \longrightarrow$ 
       $nlength\ (addzero\ (lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0)) = nlength\ ls$ 
      using assms
      by (metis 4 addzero-def co.enat.exhaust-sel lcppl-nlength-alt lcppl-nlength-zero lsum-nlength
      nlength-NCons)
show ?thesis using 4 5
using assms gr-zeroI by blast
qed

```

lemma *lcppl-lsum-nnth*:

```

assumes nidx ls
       $nnth\ ls\ 0 = 0$ 
       $nfinite\ ls$ 

```

$nfinite\ \sigma$
 $nlast\ ls = the-enat(nlength\ \sigma)$
 $(\forall\ i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$
 $nlength\ \sigma > 0$
 $j \leq nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))$
shows $(j=0 \longrightarrow$
 $(nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0))))\ j) =$
 $nfirst(nfirst\ (lcppl\ f\ g\ \sigma\ ls))\)$
 \wedge
 $(j>0 \longrightarrow (nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0))))\ j) =$
 $nlast(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ (j-1)))$

proof –

have 0: $nlength\ \sigma > 0 \longrightarrow nlength\ ls > 0$

using *assms*

by (*metis enat.distinct(2) enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)

have 1: $nlength\ \sigma > 0 \longrightarrow nlastnfirst\ (lcppl\ f\ g\ \sigma\ ls)$

using 0 *assms*

by (*simp add: enat-the-enat lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat*)

have 2: $nlength\ \sigma > 0 \longrightarrow$
 $(\forall\ j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls). nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j) > 0)$

using *assms*

by (*metis enat.distinct(2) enat-the-enat lcppl-nlength-all-gr-zero nfinite-conv-nlength-enat*)

have 3: $nlength\ \sigma > 0 \longrightarrow$
 $nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) =$
 $nlength\ (lcppl\ f\ g\ \sigma\ ls) + 1$

using *assms*

by (*metis 0 co.enat.exhaust-sel i0-less lcppl-lsum-nlength lcppl-nlength plus-1-eSuc(2)*)

have 4: $nlength\ \sigma > 0 \longrightarrow$
 $(nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0))))\ 0) =$
 $nfirst(nfirst\ (lcppl\ f\ g\ \sigma\ ls))$

by (*metis lsum-addzero-nfirst nfirst-nfusecat-nfirst nlast-NNil ntaken-0 ntaken-nlast pfilt-nnth zero-enat-def zero-le*)

have 5: $nlength\ \sigma > 0 \longrightarrow$
 $j \leq nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) \wedge j > 0 \longrightarrow$
 $(nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0))))\ (j))$
 $= nlast(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ (j-1))$

using *assms pfilt-nfusecat-lsum-3[of (lcppl f g σ ls) j]*

by (*metis 1 2 all-gr-zero-nnth-a all-nfinite-nnth-a lcppl-nlength-all-nfinite*)

from 4 5 **show** *?thesis* **using** *assms*

by *blast*

qed

lemma *lcppl-lsum-nnth-alt:*

assumes *nidx ls*

$nnth\ ls\ 0 = 0$

$\neg nfinite\ ls$

$\neg nfinite\ \sigma$

$(\forall\ i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$

$j \leq nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))$

shows ($j=0 \longrightarrow$
 $(\text{nnth } (\text{pfilt } (\text{nfusecat } ((\text{lcpl } f \ g \ \sigma \ \text{ls})))) \ (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0))) \ j) =$
 $\text{nfist}(\text{nfist } (\text{lcpl } f \ g \ \sigma \ \text{ls})) \)$
 \wedge
 $(j>0 \longrightarrow (\text{nnth } (\text{pfilt } (\text{nfusecat } ((\text{lcpl } f \ g \ \sigma \ \text{ls})))) \ (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0))) \ j) =$
 $\text{nlast}(\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ (j-1)))$

proof –

have 1: $\text{nlastnfist } (\text{lcpl } f \ g \ \sigma \ \text{ls})$
using *assms*
using *lcpl-nfusecat-nlastnfist-alt* **by** *blast*
have 2: $(\forall j \leq \text{nlength } (\text{lcpl } f \ g \ \sigma \ \text{ls}). \text{nlength}(\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ j) > 0)$
using *assms lcpl-nlength-all-gr-zero-alt* **by** *blast*
have 3: $\text{nlength } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0)) = \text{nlength } (\text{lcpl } f \ g \ \sigma \ \text{ls}) + 1$
using *assms*
by (*simp add: lcpl-lsum-nlength-alt lcpl-nlength-alt nfinite-conv-nlength-enat*)
have 4: $(\text{nnth } (\text{pfilt } (\text{nfusecat } ((\text{lcpl } f \ g \ \sigma \ \text{ls})))) \ (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0))) \ 0) =$
 $\text{nfist}(\text{nfist } (\text{lcpl } f \ g \ \sigma \ \text{ls}))$
by (*metis lsum-addzero-nfirst nfirst-nfusecat-nfirst nlast-NNil ntaken-0 ntaken-nlast pfilt-nnth zero-enat-def zero-le*)
have 5: $j \leq \text{nlength } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0)) \wedge j>0 \longrightarrow$
 $(\text{nnth } (\text{pfilt } (\text{nfusecat } ((\text{lcpl } f \ g \ \sigma \ \text{ls})))) \ (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0))) \ (j))$
 $= \text{nlast}(\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ (j-1))$
using *assms pfilt-nfusecat-lsum-3[of (lcpl f g σ ls) j]*
using 1 2 *all-gr-zero-nnth-a all-nfinite-nnth-a lcpl-nlength-all-nfinite-alt* **by** *blast*
from 4 5 **show** *?thesis* **using** *assms*
by *blast*
qed

lemma *lcpl-lsum-nnth-a:*

assumes *nidx ls*
 $\text{nnth } \text{ls } 0 = 0$
 $\text{nfinite } \text{ls}$
 $\text{nfinite } \sigma$
 $\text{nlast } \text{ls} = \text{the-enat } (\text{nlength } \sigma)$
 $(\forall i < \text{nlength } \text{ls}. (\text{nsubn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i))) \models f \ \text{fproj } g)$
 $\text{nlength } \sigma > 0$
 $j \leq \text{nlength } \text{ls}$
shows $(\text{nnth } (\text{pfilt } (\text{nfusecat } ((\text{lcpl } f \ g \ \sigma \ \text{ls})))) \ (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0))) \ j) =$
 $(\text{nnth } \text{ls } j)$

proof –

have 1: $\text{nlength } \sigma > 0 \longrightarrow \text{nlength } \text{ls} > 0$
using *assms*
by (*metis enat.distinct(2) enat-the-enat nfinite-nlength-enat nnth-nlast the-enat-0 zero-less-iff-neq-zero*)
have 2: $\text{nlength } \sigma > 0 \longrightarrow \text{nlength } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0)) = \text{nlength } \text{ls}$
by (*simp add: assms lcpl-lsum-nlength*)
have 3: $\text{nlength } \sigma > 0 \longrightarrow$
 $j \leq \text{nlength } \text{ls} \longrightarrow$
 $(j=0 \longrightarrow$
 $(\text{nnth } (\text{pfilt } (\text{nfusecat } ((\text{lcpl } f \ g \ \sigma \ \text{ls})))) \ (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0))) \ j) =$

$$\begin{aligned} & \text{nfirst}(\text{nfirst}(\text{lcpl} f g \sigma \text{ ls}))) \\ \wedge \\ & (j > 0 \longrightarrow (\text{nnth}(\text{pfilt}(\text{nfusecat}((\text{lcpl} f g \sigma \text{ ls}))) (\text{addzero}(\text{lsum}((\text{lcpl} f g \sigma \text{ ls}) 0))) j) = \\ & \quad \text{nlast}(\text{nnth}(\text{lcpl} f g \sigma \text{ ls}) (j-1))) \\ \text{by } & (\text{metis } 2 \text{ assms lcpl-lsum-nnth}) \\ \text{have } & 4: \text{nlength } \sigma > 0 \longrightarrow \\ & \quad j \leq \text{nlength } \text{ls} \wedge j = 0 \longrightarrow \text{nfirst}(\text{nfirst}(\text{lcpl} f g \sigma \text{ ls})) = (\text{nnth } \text{ls } j) \\ \text{using } & \text{assms lcpl-nfirst[of } \text{ls } \sigma f g] \\ \text{by } & (\text{metis } 1 \text{ nlast-NNil ntaken-0 ntaken-nlast}) \\ \text{have } & 5: \text{nlength } \sigma > 0 \longrightarrow \\ & \quad j \leq \text{nlength } \text{ls} \wedge j > 0 \longrightarrow j-1 \leq \text{nlength}(\text{lcpl} f g \sigma \text{ ls}) \\ \text{using } & \text{assms} \\ \text{by } & (\text{metis } 1 \text{ Suc-diff-1 Suc-ile-eq co.enat.exhaust-sel i0-less iless-Suc-eq lcpl-nlength}) \\ \text{have } & 6: \text{nlength } \sigma > 0 \longrightarrow \\ & \quad j \leq \text{nlength } \text{ls} \wedge j > 0 \longrightarrow \text{nlast}(\text{nnth}(\text{lcpl} f g \sigma \text{ ls}) (j-1)) = (\text{nnth } \text{ls } j) \\ \text{using } & \text{lcpl-nlast-nnth assms} \\ \text{by } & (\text{metis } 5 \text{ Suc-diff-1 enat.distinct(2) enat-the-enat nfinite-conv-nlength-enat}) \\ \text{show } & ?thesis \\ \text{using } & 3 \ 4 \ 6 \text{ using assms} \\ \text{by } & (\text{metis not-gr-zero}) \\ \text{qed} &
\end{aligned}$$

lemma *lcpl-lsum-nnth-a-alt:*

assumes *nidx ls*
 $\text{nnth } \text{ls } 0 = 0$
 $\neg \text{nfinite } \text{ls}$
 $\neg \text{nfinite } \sigma$
 $(\forall i < \text{nlength } \text{ls}. (\text{nsbn } \sigma (\text{nnth } \text{ls } i) (\text{nnth } \text{ls } (\text{Suc } i))) \models f \text{ fproj } g)$
 $j \leq \text{nlength } \text{ls}$
shows $(\text{nnth}(\text{pfilt}(\text{nfusecat}((\text{lcpl} f g \sigma \text{ ls}))) (\text{addzero}(\text{lsum}((\text{lcpl} f g \sigma \text{ ls}) 0))) j) =$
 $(\text{nnth } \text{ls } j)$
proof –
have 2: $\text{nlength}(\text{addzero}(\text{lsum}(\text{lcpl} f g \sigma \text{ ls}) 0)) = \text{nlength } \text{ls}$
by (*simp add: assms lcpl-lsum-nlength-alt*)
have 3: $j \leq \text{nlength } \text{ls} \longrightarrow$
 $(j = 0 \longrightarrow$
 $(\text{nnth}(\text{pfilt}(\text{nfusecat}((\text{lcpl} f g \sigma \text{ ls}))) (\text{addzero}(\text{lsum}((\text{lcpl} f g \sigma \text{ ls}) 0))) j) =$
 $\text{nfirst}(\text{nfirst}(\text{lcpl} f g \sigma \text{ ls})))$
 \wedge
 $(j > 0 \longrightarrow (\text{nnth}(\text{pfilt}(\text{nfusecat}((\text{lcpl} f g \sigma \text{ ls}))) (\text{addzero}(\text{lsum}((\text{lcpl} f g \sigma \text{ ls}) 0))) j) =$
 $\text{nlast}(\text{nnth}(\text{lcpl} f g \sigma \text{ ls}) (j-1)))$
by (*metis 2 assms lcpl-lsum-nnth-alt*)
have 4: $j \leq \text{nlength } \text{ls} \wedge j = 0 \longrightarrow \text{nfirst}(\text{nfirst}(\text{lcpl} f g \sigma \text{ ls})) = (\text{nnth } \text{ls } j)$
using *assms lcpl-nfirst-alt[of } \text{ls } \sigma f g]* **by** (*metis ndropn-0 ndropn-nfirst*)
have 5: $j \leq \text{nlength } \text{ls} \wedge j > 0 \longrightarrow j-1 \leq \text{nlength}(\text{lcpl} f g \sigma \text{ ls})$
using *assms*
by (*meson enat-ile lcpl-nfinite linorder-le-cases nfinite-conv-nlength-enat*)
have 6: $j \leq \text{nlength } \text{ls} \wedge j > 0 \longrightarrow \text{nlast}(\text{nnth}(\text{lcpl} f g \sigma \text{ ls}) (j-1)) = (\text{nnth } \text{ls } j)$
using *lcpl-nlast-nnth-alt assms*
by (*metis 5 Suc-pred'*)

show *?thesis*
using 3 4 6 **using** *assms*
by (*metis not-gr-zero*)
qed

lemma *lcppl-pfilt-nfusecat-lsum:*

assumes *nidx ls*
 $nnth\ ls\ 0 = 0$
 $\neg nfinite\ ls$
 $\neg nfinite\ \sigma$
 $nlast\ ls = the-enat\ (nlength\ \sigma)$
 $(\forall\ i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$
 $nlength\ \sigma > 0$
shows $(pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0))) = ls$
using *assms*
proof –
have 0: $nlength\ \sigma > 0 \longrightarrow nlength\ ls > 0$
using *assms*
by (*metis enat.distinct(2) enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have 1: $nlength\ \sigma > 0 \longrightarrow$
 $nlength\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0))) = nlength\ ls$
using *assms*
by (*simp add: lcppl-lsum-nlength pfilt-nlength*)
have 2: $nlength\ \sigma > 0 \longrightarrow$
 $(\forall\ j. j \leq nlength\ ls \longrightarrow$
 $(nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0)))\ j) =$
 $(nnth\ ls\ j))$
by (*simp add: assms lcppl-lsum-nnth-a*)
from 1 2 **show** *?thesis* **using** *assms nellist-eq-nnth-eq*
by *metis*
qed

lemma *lcppl-pfilt-nfusecat-lsum-alt:*

assumes *nidx ls*
 $nnth\ ls\ 0 = 0$
 $\neg nfinite\ ls$
 $\neg nfinite\ \sigma$
 $(\forall\ i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$
shows $(pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0))) = ls$
using *assms*
proof –
have 1: $nlength\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0))) = nlength\ ls$
using *assms* **by** (*simp add: lcppl-lsum-nlength-alt pfilt-nlength*)
have 2: $(\forall\ j. j \leq nlength\ ls \longrightarrow$
 $(nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0)))\ j) =$
 $(nnth\ ls\ j))$
by (*simp add: assms lcppl-lsum-nnth-a-alt*)
from 1 2 **show** *?thesis* **using** *assms nellist-eq-nnth-eq*
by *metis*
qed

lemma *lcppl-nfusecat-pfilt-powerinterval:*

assumes *nidx ls*

nnth ls 0 = 0

nfinite ls

nfinite σ

nlast ls = the-enat (nlength σ)

$(\forall i < \text{nlength } ls. (\text{nsubn } \sigma (\text{nnth } ls \ i) (\text{nnth } ls \ (\text{Suc } i))) \models f \text{ fproj } g)$

nlength σ > 0

shows *powerinterval g (pfilt σ (nfusecat (lcppl f g σ ls))) (addzero (lsum (lcppl f g σ ls) 0))*

proof –

have *0: nlength σ > 0 ⟶ nlength ls > 0*

using *assms*

by (*metis enat.distinct(2) enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)

have *01: (∀ i < nlength ls.*

g (pfilt (nsubn σ (nnth ls i) (nnth ls (Suc i))) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))))

using *assms cppl-fprojection by auto*

have *02: nlength σ > 0 ⟶ (∀ i < nlength ls. g (pfilt σ (nnth (lcppl f g σ ls) i)))*

using *0 assms lcppl-nfusecat-pfilt-fpower-help*

by (*metis enat.distinct(2) enat-the-enat nfinite-conv-nlength-enat*)

have *03: powerinterval g (pfilt σ (nfusecat (lcppl f g σ ls))) (addzero (lsum (lcppl f g σ ls) 0)) =*
(∀ i < nlength (addzero (lsum (lcppl f g σ ls) 0)).

g (nsubn (pfilt σ (nfusecat (lcppl f g σ ls)))
(nnth (addzero (lsum (lcppl f g σ ls) 0)) i)
(nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i))
))

by (*simp add: powerinterval-def*)

have *04: nlength σ > 0 ⟶ nlength (addzero (lsum (lcppl f g σ ls) 0)) = nlength ls*

using *assms lcppl-lsum-nlength by blast*

have *05: nlength σ > 0 ⟶*

(∀ i < nlength ls.
(pfilt σ (nnth (lcppl f g σ ls) i)) =
(nsubn (pfilt σ (nfusecat (lcppl f g σ ls)))
(nnth (addzero (lsum (lcppl f g σ ls) 0)) i)
(nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i))
))

proof –

have *06: nlength σ > 0 ⟶*

(∀ i < nlength ls. nlength (pfilt σ (nnth (lcppl f g σ ls) i)) =
nlength (nnth (lcppl f g σ ls) i))

using *pfilt-nlength by blast*

have *07: nlength σ > 0 ⟶*

(∀ i < nlength ls. nlength (nnth (lcppl f g σ ls) i) =
nlength (nmap (λx. x + (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))))

using *assms by (simp add: lcppl-nnth)*

have *08: nlength σ > 0 ⟶*

(∀ i < nlength ls.
nlength (nmap (λx. x + (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))) =

$nlength\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\)))$

using *nlength-nmap by blast*

have 09: $nlength\ \sigma > 0 \longrightarrow$
 $(\forall i < nlength\ ls.$
 $nlength\ (nsubn\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))$
 $\quad (nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ i)$
 $\quad (nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i))$
 $\quad) =$
 $(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i)) -$
 $(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ i))$

using *assms*
by (*metis* 04 *enat-minus-mono1 idiff-enat-enat lcppl-lsum-less-th-equal min.orderE*
nsubn-nlength pfilt-nlength)

have 10: $nlength\ \sigma > 0 \longrightarrow$
 $(\forall i < nlength\ ls. (nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i)) =$
 $(nnth\ (NCons\ 0\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i)))$

using 04 *addzero-def by auto*

have 11: $nlength\ \sigma > 0 \longrightarrow$
 $(\forall i < nlength\ ls. (nnth\ (NCons\ 0\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i)) =$
 $(\sum k::nat = 0..i. nlength\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k)))$

using 04

proof *simp-all*

assume $nlength\ \sigma \neq (0::enat) \longrightarrow nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ (0::nat))) = nlength\ ls$

show $nlength\ \sigma \neq (0::enat) \longrightarrow$
 $(\forall i::nat. enat\ i < nlength\ ls \longrightarrow enat\ (nnth\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ (0::nat))\ i) =$
 $(\sum k::nat = 0::nat..i. nlength\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k)))$

proof

assume $nlength\ \sigma \neq (0::enat)$

show $(\forall i::nat. enat\ i < nlength\ ls \longrightarrow$
 $enat\ (nnth\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ (0::nat))\ i) =$
 $(\sum k::nat = 0::nat..i. nlength\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k)))$

proof

fix i

show $i < nlength\ ls \longrightarrow$
 $enat\ (nnth\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ (0::nat))\ i) =$
 $(\sum k::nat = 0::nat..i. nlength\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k))$

proof

assume $a: i < nlength\ ls$

show $enat\ (nnth\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)\ i) =$
 $(\sum k = 0..i. nlength\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k))$

proof –

have 110: $all\text{-}nfinite\ (lcppl\ f\ g\ \sigma\ ls)$

using *all-nfinite-nnth-a assms lcppl-nlength-all-nfinite by blast*

have 111: $nnth\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)\ i =$
 $(\sum k = 0..i. the\text{-}enat\ (nlength\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k)))$

using *lsum-nnth[of i (lcppl f g σ ls) 0] a assms*

by (*metis* 0 110 *add-cancel-right-left co.enat.exhaust-sel iless-Suc-eq*
lcppl-nlength not-gr-zero)

```

have 112: ( $\sum k = 0..i.$  the-enat (nlength (nnth (lcppl f g σ ls) k))) =
  the-enat( $\sum k = 0..i.$  (nlength (nnth (lcppl f g σ ls) k)))
using sum-the-enat[of (lcppl f g σ ls) i] assms
by (metis 0 110 a co.enat.exhaust-sel gr-implies-not-zero illess-Suc-eq
  lcppl-nlength)
have 113: ( $\sum k = 0..i.$  (nlength (nnth (lcppl f g σ ls) k))) < ∞
  using sum-finite[of (lcppl f g σ ls) i] 0 110 assms a
  by (metis co.enat.exhaust-sel gr-implies-not-zero illess-Suc-eq lcppl-nlength)
have 114: enat (the-enat( $\sum k = 0..i.$  (nlength (nnth (lcppl f g σ ls) k)))) =
  ( $\sum k = 0..i.$  (nlength (nnth (lcppl f g σ ls) k)))
  using 113 enat-ord-simps(4) enat-the-enat by blast
show ?thesis using 111 112 114 by presburger
qed
qed
qed
qed
qed
have 12: nlength σ > 0  $\longrightarrow$ 
  ( $\forall i < nlength\ ls.$  (nnth (addzero (lsum (lcppl f g σ ls) 0)) i) =
    (nnth (NCons 0 (lsum (lcppl f g σ ls) 0)) i) )
  using 04 addzero-def by auto
have 121: nlength σ > 0  $\longrightarrow$  (nnth (NCons 0 (lsum (lcppl f g σ ls) 0)) 0) = 0
  by simp
have 13: nlength σ > 0  $\longrightarrow$ 
  ( $\forall i < nlength\ ls.$  (nnth (NCons 0 (lsum (lcppl f g σ ls) 0)) i) =
    (if i = 0 then 0
      else ( $\sum k::nat = 0..(i-1).$  nlength(nnth (lcppl f g σ ls) k)) ))
  using 11
  by (metis 121 Suc-diff-1 Suc-ile-eq not-gr-zero order-less-imp-le zero-enat-def)
have 14: nlength σ > 0  $\longrightarrow$ 
  ( $\forall i < nlength\ ls.$ 
    (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i)) -
    (nnth (addzero (lsum (lcppl f g σ ls) 0)) i) =
    ( $\sum k::nat = 0..(i).$  nlength(nnth (lcppl f g σ ls) k)) -
    (if i = 0 then 0 else
      ( $\sum k::nat = 0..(i-1).$  nlength(nnth (lcppl f g σ ls) k)) ))
  using 10 11 12 13 by simp (metis One-nat-def idiff-enat-enat not-gr-zero)
have 15: nlength σ > 0  $\longrightarrow$ 
  ( $\forall i < nlength\ ls.$ 
    ( $\sum k::nat = 0..(i).$  nlength(nnth (lcppl f g σ ls) k)) -
    (if i = 0 then 0 else
      ( $\sum k::nat = 0..(i-1).$  nlength(nnth (lcppl f g σ ls) k)) )) =
    (if i = 0 then nlength(nnth (lcppl f g σ ls) 0) else
      ( $\sum k::nat = 0..(i).$  nlength(nnth (lcppl f g σ ls) k)) -
      ( $\sum k::nat = 0..(i-1).$  nlength(nnth (lcppl f g σ ls) k)) ))
  by (simp add: Nitpick.case-nat-unfold)
have 16: nlength σ > 0  $\longrightarrow$ 
  ( $\forall i < nlength\ ls.$ 
    (if i = 0 then nlength (nnth (lcppl f g σ ls) 0) else

```

$$\begin{aligned}
& (\sum k::nat= 0..(i). \text{ nlength}(\text{nnth}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } k)) - \\
& \quad (\sum k::nat= 0..(i-1). \text{ nlength}(\text{nnth}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } k))) = \\
& (\text{if } i = 0 \text{ then } \text{ nlength}(\text{nnth}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } 0) \text{ else } \\
& \quad \text{ nlength}(\text{nnth}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } i))) \\
& \text{using } 13 \text{ sum.cl-ivl-Suc[of } \lambda k. \text{ nlength}(\text{nnth}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } k) \text{ } 0] \\
& \text{by (metis Suc-diff-1 enat.distinct(2) enat-add-sub-same less-nat-zero-code not-gr-zero)} \\
& \text{have 17: } \text{ nlength } \sigma > 0 \longrightarrow \\
& \quad (\forall i < \text{ nlength } ls. \\
& \quad (\text{if } i = 0 \text{ then } \text{ nlength}(\text{nnth}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } 0) \text{ else } \\
& \quad \quad \text{ nlength}(\text{nnth}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } i)) = \\
& \quad \quad \text{ nlength}(\text{nnth}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } i)) \\
& \text{by (simp add: Nitpick.case-nat-unfold)} \\
& \text{have 18: } \text{ nlength } \sigma > 0 \longrightarrow \\
& \quad (\forall i < \text{ nlength } ls. \text{ nlength}(\text{pfilt } \sigma (\text{nnth}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } i)) = \\
& \quad \quad \text{ nlength}(\text{nsubn}(\text{pfilt } \sigma (\text{nfusecat}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls)))) \\
& \quad \quad (\text{nnth}(\text{addzero}(\text{lsum}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } 0)) \text{ } i) \\
& \quad \quad (\text{nnth}(\text{addzero}(\text{lsum}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } 0)) (\text{Suc } i)) \\
& \quad \quad)) \\
& \text{by (simp add: 09 14 15 16 pfilt-nlength)} \\
& \text{have 19: } \text{ nlength } \sigma > 0 \longrightarrow \\
& \quad (\forall i < \text{ nlength } ls. \\
& \quad (\forall j \leq \text{ nlength}(\text{nnth}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } i). \\
& \quad (\text{nnth}(\text{addzero}(\text{lsum}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } 0)) (\text{Suc } i)) \leq \\
& \quad \quad \text{ nlength}(\text{pfilt } \sigma (\text{nfusecat}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls)))) \\
& \text{by (simp add: 04 assms lcppl-lsum-less-th-equal pfilt-nlength)} \\
& \text{have 22: } \text{ nlength } \sigma > 0 \longrightarrow \text{ nlastnfirst}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \\
& \text{using } 0 \text{ assms} \\
& \text{by (simp add: enat-the-enat lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat)} \\
& \text{have 23: } \text{ nlength } \sigma > 0 \longrightarrow (\forall j \leq \text{ nlength}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls). \text{ nlength}(\text{nnth}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } j) > 0) \\
& \text{using } \text{assms} \\
& \text{by (metis enat.distinct(2) enat-the-enat lcppl-nlength-all-gr-zero nfinite-conv-nlength-enat)} \\
& \text{have 190: } \text{ nlength } \sigma > 0 \longrightarrow \\
& \quad (\forall i < \text{ nlength } ls. \\
& \quad (\forall j \leq \text{ nlength}(\text{nnth}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } i). \\
& \quad (\text{nnth}(\text{addzero}(\text{lsum}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } 0)) \text{ } i) \leq \\
& \quad (\text{nnth}(\text{addzero}(\text{lsum}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } 0)) (\text{Suc } i)) \\
& \quad)) \\
& \text{using } 0 \text{ 04 06 09 18 23 lsum-nlength[of } (\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } 0] \text{ lcppl-nlength[of } ls \text{ } f \text{ } g \text{ } \sigma] \text{ assms} \\
& \text{by simp} \\
& \text{(metis (no-types, lifting) co.enat.exhaust-sel diff-is-0-eq' iless-Suc-eq le-cases zero-enat-def)} \\
& \text{have 20: } \text{ nlength } \sigma > 0 \longrightarrow \\
& \quad (\forall i < \text{ nlength } ls. \\
& \quad (\forall j \leq \text{ nlength}(\text{nnth}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } i). \\
& \quad (\text{nnth}(\text{nsubn}(\text{pfilt } \sigma (\text{nfusecat}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls)))) \\
& \quad \quad (\text{nnth}(\text{addzero}(\text{lsum}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } 0)) \text{ } i) \\
& \quad \quad (\text{nnth}(\text{addzero}(\text{lsum}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } 0)) (\text{Suc } i)) \\
& \quad \quad) \text{ } j) = \\
& \quad (\text{nnth}(\text{pfilt } \sigma (\text{nfusecat}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls)))) \\
& \quad ((\text{nnth}(\text{addzero}(\text{lsum}(\text{lcppl } f \text{ } g \text{ } \sigma \text{ } ls) \text{ } 0)) \text{ } i) + j))))
\end{aligned}$$

```

    using nsubn-nnth[of (pfilt  $\sigma$  (nfusecat (lcppl f g  $\sigma$  ls))) ] by simp
    (metis 06 09 18 assms(7) enat-ord-simps(1) min.orderE)
have 21: nlength  $\sigma > 0 \longrightarrow$ 
  ( $\forall i < \text{nlength } ls.$ 
    ( $\forall j \leq \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i).$ 
      ( $\text{nnth } (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls)))$ 
        ( $((\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i) + j) \ ) =$ 
          ( $\text{nnth } \sigma \ (\text{nnth } (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls)))$ 
            ( $((\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i) + j) \ ) \ ) \ )$ 
      )
    using pfilt-nlength[of  $\sigma$  (nfusecat (lcppl f g  $\sigma$  ls)) ]
    pfilt-nnth[of -  $\sigma$  (nfusecat (lcppl f g  $\sigma$  ls)) ]
    by (metis nnth-0 pfilt-code(2) pfilt-pfilt)
have 24: nlength  $\sigma > 0 \longrightarrow$ 
  ( $\forall i \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls).$ 
    ( $\forall j \leq \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i).$ 
      ( $(\text{nnth } (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls)) ((\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i) + j) \ )) =$ 
        ( $(\text{nnth } (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i) \ j) \ ) \ )$ 
    )

  using assms lsum-nfusecat-nnth-lsum-nnth[of (lcppl f g  $\sigma$  ls)] lcppl-nlength[of ls f g  $\sigma$ ]
  0 06 09 18 22 23
  by (metis all-gr-zero-nnth-a all-nfinite-nnth-a lcppl-nlength-all-nfinite)
have 241: nlength  $\sigma > 0 \longrightarrow \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls) = \text{nlength } ls - 1$ 
  using 0 assms
  by (simp add: epred-conv-minus lcppl-nlength)
have 25: nlength  $\sigma > 0 \longrightarrow$ 
  ( $\forall i < \text{nlength } ls.$ 
    ( $\forall j \leq \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i).$ 
      ( $\text{nnth } \sigma \ (\text{nnth } (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls)))$ 
        ( $((\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i) + j) \ )) =$ 
          ( $\text{nnth } (\text{pfilt } \sigma \ (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i)) \ j) \ )$ 
      )
    using 0 04 24 lsum-nlength[of (lcppl f g  $\sigma$  ls) 0] pfilt-nlength[of  $\sigma$  ] pfilt-nnth
    by (metis addzero-def i0-less iless-Suc-eq nlength-NCons)
have 26: nlength  $\sigma > 0 \longrightarrow$ 
  ( $\forall i < \text{nlength } ls.$ 
    ( $\forall j \leq \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i).$ 
      ( $\text{nnth } (\text{pfilt } \sigma \ (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i)) \ j) =$ 
        ( $\text{nnth } (\text{nsubn } (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls)))$ 
          ( $\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i)$ 
            ( $\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ (\text{Suc } i))$ 
              ) j) \ ) \ )
    )

  by (simp add: 20 21 25)
from 18 26 show ?thesis using nellist-eq-nnth-eq
by (metis 06)
qed
show ?thesis
using 02 03 04 05 assms(7) by force
qed

```

lemma lcppl-nfusecat-pfilt-powerinterval-alt:

assumes $nidx\ ls$
 $nnth\ ls\ 0 = 0$
 $\neg nfinite\ ls$
 $\neg nfinite\ \sigma$
 $(\forall\ i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$
shows $powerinterval\ g\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))$
proof –
have 01: $(\forall\ i < nlength\ ls.$
 $g\ (pfilt\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))$
using *assms cppl-fprojection* **by** *auto*
have 02: $(\forall\ i < nlength\ ls. g\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)))$
using *assms lcppl-nfusecat-pfilt-fpower-help-alt*
by *(metis enat-the-enat nfinite-conv-nlength-enat)*
have 03 : $powerinterval\ g\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) =$
 $(\forall\ i < nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)).$
 $g\ (nsubn\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))$
 $(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ i)$
 $(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i))$
 $))$
by *(simp add: powerinterval-def)*
have 04: $nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) = nlength\ ls$
using *assms lcppl-lsum-nlength-alt* **by** *blast*
have 05: $(\forall\ i < nlength\ ls.$
 $(pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)) =$
 $(nsubn\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))$
 $(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ i)$
 $(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i))$
 $))$
proof –
have 06:
 $(\forall\ i < nlength\ ls. nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)) =$
 $nlength\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i))$
using *pfilt-nlength* **by** *blast*
have 07: $(\forall\ i < nlength\ ls. nlength\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i) =$
 $nlength\ (nmap\ (\lambda x. x + (nnth\ ls\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))$
using *assms* **by** *(simp add: lcppl-nnth)*
have 08: $(\forall\ i < nlength\ ls.$
 $nlength\ (nmap\ (\lambda x. x + (nnth\ ls\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) =$
 $nlength\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$
using *nlength-nmap* **by** *blast*
have 09: $(\forall\ i < nlength\ ls.$
 $nlength\ (nsubn\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))$
 $(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ i)$
 $(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i))$
 $) =$
 $(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i)) -$
 $(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ i)$
using *assms*
by *(metis 04 enat-minus-mono1 idiff-enat-enat lcppl-lsum-less-th-equal-alt min-def*
 $nsubn-nlength\ pfilt-nlength)$

```

have 10: (∀ i<nlength ls. (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i)) =
  (nnth (NCons 0 (lsum (lcppl f g σ ls) 0)) (Suc i)))
using 04 addzero-def by auto
have 11: (∀ i<nlength ls. (nnth (NCons 0 (lsum (lcppl f g σ ls) 0)) (Suc i)) =
  (∑ k::nat= 0..i. nlength(nnth (lcppl f g σ ls) k)) )
using 04
proof simp-all
assume nlength (addzero (lsum (lcppl f g σ ls) (0::nat))) = nlength ls
show (∀ i::nat. enat i < nlength ls ⟶ enat (nnth (lsum (lcppl f g σ ls) (0::nat)) i) =
  (∑ k::nat = 0::nat..i. nlength (nnth (lcppl f g σ ls) k)))
proof
fix i
show i < nlength ls ⟶
  enat (nnth (lsum (lcppl f g σ ls) (0::nat)) i) =
  (∑ k::nat = 0::nat..i. nlength (nnth (lcppl f g σ ls) k))
proof
assume a: i < nlength ls
show enat (nnth (lsum (lcppl f g σ ls) 0) i) =
  (∑ k = 0..i. nlength (nnth (lcppl f g σ ls) k))
proof -
have 110: all-nfinite (lcppl f g σ ls)
using all-nfinite-nnth-a assms lcppl-nlength-all-nfinite-alt by blast
have 111: nnth (lsum (lcppl f g σ ls) 0) i = (∑ k = 0..i. the-enat (nlength (nnth (lcppl f
g σ ls) k)))
using lsum-nnth[of i (lcppl f g σ ls) 0] a assms
by (metis 04 110 add-cancel-right-left addzero-def iless-Suc-eq lsum-nlength
  nlength-NCons nlength-eq-enat-nfiniteD zero-enat-def)
have 112: (∑ k = 0..i. the-enat (nlength (nnth (lcppl f g σ ls) k))) =
  the-enat(∑ k = 0..i. (nlength (nnth (lcppl f g σ ls) k)))
using sum-the-enat[of (lcppl f g σ ls) i] assms
by (metis 110 lcppl-nfinite linorder-le-cases nfinite-ntaken ntaken-all)
have 113: (∑ k = 0..i. (nlength (nnth (lcppl f g σ ls) k))) < ∞
using sum-finite[of (lcppl f g σ ls) i] 110 04 assms a
by (metis addzero-def iless-Suc-eq lsum-nlength nlength-NCons
  nlength-eq-enat-nfiniteD zero-enat-def)
have 114: enat (the-enat(∑ k = 0..i. (nlength (nnth (lcppl f g σ ls) k)))) =
  (∑ k = 0..i. (nlength (nnth (lcppl f g σ ls) k)))
using 113 enat-ord-simps(4) enat-the-enat by blast
show ?thesis using 111 112 114 by presburger
qed
qed
qed
qed
have 12: (∀ i<nlength ls. (nnth (addzero (lsum (lcppl f g σ ls) 0)) i) =
  (nnth (NCons 0 (lsum (lcppl f g σ ls) 0)) i) )
using 04 addzero-def by auto
have 121: (nnth (NCons 0 (lsum (lcppl f g σ ls) 0)) 0) = 0
by simp
have 13: (∀ i<nlength ls. (nnth (NCons 0 (lsum (lcppl f g σ ls) 0)) i) =
  (if i = 0 then 0

```


$\text{else } (\sum k::\text{nat} = 0..(i-1). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ k)) \)$
using 11
by (*metis* 121 *Suc-diff-1* *Suc-ile-eq* *not-gr-zero* *order-less-imp-le* *zero-enat-def*)
have 14: $(\forall i < \text{nlength } \text{ls}.$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ 0)) \ (\text{Suc } i)) -$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ 0)) \ i) =$
 $(\sum k::\text{nat} = 0..(i). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ k)) -$
 $(\text{if } i = 0 \text{ then } 0 \text{ else}$
 $(\sum k::\text{nat} = 0..(i-1). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ k)) \) \)$

using 10 11 12 13 **by** *simp* (*metis* *One-nat-def* *idiff-enat-enat* *not-gr-zero*)
have 15: $(\forall i < \text{nlength } \text{ls}.$
 $(\sum k::\text{nat} = 0..(i). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ k)) -$
 $(\text{if } i = 0 \text{ then } 0 \text{ else}$
 $(\sum k::\text{nat} = 0..(i-1). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ k)) \) =$
 $(\text{if } i = 0 \text{ then } \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ 0) \text{ else}$
 $(\sum k::\text{nat} = 0..(i). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ k)) -$
 $(\sum k::\text{nat} = 0..(i-1). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ k)) \) \)$

by (*simp* *add: Nitpick.case-nat-unfold*)
have 16: $(\forall i < \text{nlength } \text{ls}.$
 $(\text{if } i = 0 \text{ then } \text{nlength } (\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ 0) \text{ else}$
 $(\sum k::\text{nat} = 0..(i). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ k)) -$
 $(\sum k::\text{nat} = 0..(i-1). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ k)) \) =$
 $(\text{if } i = 0 \text{ then } \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ 0) \text{ else}$
 $\text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ i) \) \)$
using 13 *sum.cl-ivl-Suc*[*of* $\lambda k. \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ k) \ 0]$
by (*metis* *Suc-diff-1* *enat.distinct(2)* *enat-add-sub-same* *less-nat-zero-code* *not-gr-zero*)
have 17: $(\forall i < \text{nlength } \text{ls}.$
 $(\text{if } i = 0 \text{ then } \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ 0) \text{ else}$
 $\text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ i) \) =$
 $\text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ i))$
by (*simp* *add: Nitpick.case-nat-unfold*)
have 18: $(\forall i < \text{nlength } \text{ls}. \text{nlength } (\text{pfilt } \sigma \ (\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ i)) =$
 $\text{nlength } (\text{nsubn } (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ \text{ls})))$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ 0)) \ i)$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ 0)) \ (\text{Suc } i))$
 $) \)$
by (*simp* *add: 09 14 15 16 pfilt-nlength*)
have 19: $(\forall i < \text{nlength } \text{ls}.$
 $(\forall j \leq \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ i).$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ 0)) \ (\text{Suc } i)) \leq$
 $\text{nlength } (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ \text{ls}))))$
by (*simp* *add: 04 assms lcppl-lsum-less-th-equal-alt pfilt-nlength*)
have 22: $\text{nlastnfirst } (\text{lcppl } f \ g \ \sigma \ \text{ls})$
using *assms lcppl-nfusecat-nlastnfirst-alt* **by** *blast*
have 23: $(\forall j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ \text{ls}). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ j) > 0)$
using *assms lcppl-nlength-all-gr-zero-alt* **by** *blast*
have 190: $(\forall i < \text{nlength } \text{ls}.$
 $(\forall j \leq \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ \text{ls}) \ i).$

$$\begin{aligned} & (nnth (addzero (lsum (lcppl f g \sigma ls) 0)) i) \leq \\ & (nnth (addzero (lsum (lcppl f g \sigma ls) 0)) (Suc i)) \\ &)) \end{aligned}$$

using 04 06 09 18 23 *lsum-nlength[of (lcppl f g \sigma ls) 0] lcppl-nlength-alt[of ls f g \sigma] assms*
by *simp*
(metis (no-types, opaque-lifting) co.enat.exhaust-sel diff-is-0-eq' iless-Suc-eq nfinite-ntaken
not-le-imp-less ntaken-all order-less-imp-le zero-enat-def)

have 20: $(\forall i < nlength\ ls.$

$$\begin{aligned} & (\forall j \leq nlength(nnth (lcppl f g \sigma ls) i). \\ & (nnth (nsubn (pfilt \sigma (nfusecat (lcppl f g \sigma ls))) \\ & (nnth (addzero (lsum (lcppl f g \sigma ls) 0)) i) \\ & (nnth (addzero (lsum (lcppl f g \sigma ls) 0)) (Suc i)) \\ &) j) = \\ & (nnth (pfilt \sigma (nfusecat (lcppl f g \sigma ls))) \\ & ((nnth (addzero (lsum (lcppl f g \sigma ls) 0)) i) + j)))) \end{aligned}$$

using *nsubn-nnth[of (pfilt \sigma (nfusecat (lcppl f g \sigma ls)))] by simp*
(metis 06 09 18 enat-ord-simps(1) min.orderE)

have 21: $(\forall i < nlength\ ls.$

$$\begin{aligned} & (\forall j \leq nlength(nnth (lcppl f g \sigma ls) i). \\ & (nnth (pfilt \sigma (nfusecat (lcppl f g \sigma ls))) \\ & ((nnth (addzero (lsum (lcppl f g \sigma ls) 0)) i) + j)) = \\ & (nnth \sigma (nnth (nfusecat (lcppl f g \sigma ls)) \\ & ((nnth (addzero (lsum (lcppl f g \sigma ls) 0)) i) + j))))) \end{aligned}$$

using *pfilt-nlength[of \sigma (nfusecat (lcppl f g \sigma ls))]*
pfilt-nnth[of - \sigma (nfusecat (lcppl f g \sigma ls))]

by *(metis enat-ord-code(4) enat-the-enat not-le-imp-less ntaken-all ntaken-nlast order-less-imp-le)*

have 24: $(\forall i \leq nlength\ (lcppl f g \sigma ls).$

$$\begin{aligned} & (\forall j \leq nlength(nnth (lcppl f g \sigma ls) i). \\ & ((nnth (nfusecat (lcppl f g \sigma ls)) ((nnth (addzero (lsum (lcppl f g \sigma ls) 0)) i) + j)) = \\ & ((nnth (nnth (lcppl f g \sigma ls) i) j))) \end{aligned}$$

using *assms lsum-nfusecat-nnth-lsum-nnth[of (lcppl f g \sigma ls)] 04 22 23*
all-gr-zero-nnth-a all-nfinite-nnth-a lcppl-nlength-all-nfinite-alt by blast

have 241: $nlength\ (lcppl f g \sigma ls) = nlength\ ls - 1$

using *assms by (simp add: epred-conv-minus lcppl-nlength-alt)*

have 25: $(\forall i < nlength\ ls.$

$$\begin{aligned} & (\forall j \leq nlength(nnth (lcppl f g \sigma ls) i). \\ & (nnth \sigma (nnth (nfusecat (lcppl f g \sigma ls)) \\ & ((nnth (addzero (lsum (lcppl f g \sigma ls) 0)) i) + j))) = \\ & (nnth (pfilt \sigma (nnth (lcppl f g \sigma ls) i) j))) \end{aligned}$$

using 04 24 *lsum-nlength[of (lcppl f g \sigma ls) 0] pfilt-nlength[of \sigma] pfilt-nnth*
assms(3)

by *(metis 241 enat2-cases enat-ord-code(3) epred-Infty epred-conv-minus*
nfinite-conv-nlength-enat)

have 26: $(\forall i < nlength\ ls.$

$$\begin{aligned} & (\forall j \leq nlength(nnth (lcppl f g \sigma ls) i). \\ & (nnth (pfilt \sigma (nnth (lcppl f g \sigma ls) i) j) = \\ & (nnth (nsubn (pfilt \sigma (nfusecat (lcppl f g \sigma ls))) \\ & (nnth (addzero (lsum (lcppl f g \sigma ls) 0)) i) \\ & (nnth (addzero (lsum (lcppl f g \sigma ls) 0)) (Suc i)) \\ &) j))) \end{aligned}$$

```

    by (simp add: 20 21 25)
  from 18 26 show ?thesis using nellist-eq-nnth-eq
  by (metis 06)
qed
show ?thesis
using 02 03 04 05 assms by force
qed

lemma lcppl-nfusecat-pfilt-nlength:
  assumes nidx ls
    nnth ls 0 = 0
    nfinite ls
    nfinite σ
    nlast ls = the-enat (nlength σ)
    (∀ i < nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f fproj g)
    nlength σ > 0
  shows   nlast (addzero (lsum (lcppl f g σ ls) 0)) =
    the-enat(nlength (pfilt σ (nfusecat (lcppl f g σ ls))))
proof -
  have 0: nlength ls > 0
    using assms
  by (metis enat.distinct(2) enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0)
  have 1: nlength (pfilt σ (nfusecat (lcppl f g σ ls))) = nlength ( (nfusecat (lcppl f g σ ls)))
    using pfilt-nlength by blast
  have 10: nfinite (lcppl f g σ ls)
    using assms(1) assms(3) lcppl-nfinite by blast
  have 11: nlastnfirst (lcppl f g σ ls)
    using assms
  by (metis 0 lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat)
  have 12: ∀ lx ∈ nset (lcppl f g σ ls). 0 < nlength lx
    using assms
  by (metis enat.distinct(2) enat-the-enat in-nset-conv-nnth lcppl-nlength-all-gr-zero
    nfinite-conv-nlength-enat)
  have 13: ∀ lx ∈ nset (lcppl f g σ ls). nfinite lx
    by (metis (no-types, lifting) 0 assms(1) assms(3) assms(6) co.enat.exhaust-sel
      cppl-fprojection i0-less illess-Suc-eq in-nset-conv-nnth lcppl-nlength lcppl-nnth nfinite-nmap)
  have 141: ∀ i ≤ nlength (lcppl f g σ ls). 0 < nlength (nnth (lcppl f g σ ls) i)
    using 12 assms
  by (metis enat.distinct(2) enat-the-enat lcppl-nlength-all-gr-zero nfinite-conv-nlength-enat)
  have 142: ∀ i ≤ nlength (lcppl f g σ ls). nfinite (nnth (lcppl f g σ ls) i)
    using 13 assms
  by (meson in-nset-conv-nnth)
  have 15: nfinite (nfusecat (lcppl f g σ ls))
    using nfusecat-nfinite[of (lcppl f g σ ls)] 10 11 12 13 by blast
  have 2: nlength ( (nfusecat (lcppl f g σ ls))) =
    (the-enat (∑ k::nat= 0..(the-enat(nlength (lcppl f g σ ls))). nlength(nnth (lcppl f g σ ls) k)))
    using 0 assms
  by (metis 10 11 13 15 nfinite-conv-nlength-enat nfusecat-nlength-nfinite the-enat.simps)
  have 3:
    nlast( addzero (lsum (lcppl f g σ ls) 0)) = nlast( (lsum (lcppl f g σ ls) 0))

```

```

using lsum-addzero-nlast
using assms(1) assms(3) lcppl-nfinite by blast
have 4: nlast( (lsum (lcppl f g σ ls) 0)) =
  (∑ k::nat = 0..the-enat (nlength (lcppl f g σ ls)). the-enat (nlength (nnth (lcppl f g σ ls) k)))
using assms lsum-nlast[of (lcppl f g σ ls) 0] 10 13 plus-nat.add-0 by presburger
have 5: (∑ k::nat = 0..the-enat (nlength (lcppl f g σ ls)). the-enat (nlength (nnth (lcppl f g σ ls) k))) =
  (the-enat (∑ k::nat = 0..the-enat(nlength (lcppl f g σ ls)). nlength(nnth (lcppl f g σ ls) k)))
using assms lsum-nnth[of the-enat (nlength (lcppl f g σ ls)) (lcppl f g σ ls) 0 ]
  sum-the-enat[of (lcppl f g σ ls) the-enat(nlength (lcppl f g σ ls))]
by (metis 13 dual-order.refl enat-ord-code(3) enat-the-enat)
show ?thesis using 1 2 3 4 5 assms
by (simp add: lcppl-lsum-nlength nfinite-conv-nlength-enat)
qed

```

2.7.3 Soundness of Projection Axioms

lemma *PJ00sem*:

$\sigma \models \neg(f \text{ fproj } \text{ inf})$

by (auto simp add: fprojection-d-def infinite-defs nfinite-conv-nlength-enat pfilt-nlength)

lemma *OPJ00sem*:

$\sigma \models \neg(f \text{ oproj } \text{ finite})$

by (auto simp add: oprojection-d-def finite-defs nfinite-conv-nlength-enat pfilt-nlength)

lemma *PJ01sem*:

$\sigma \models (f \text{ fproj } g) \longrightarrow \text{ finite}$

by (auto simp add: fprojection-d-def finite-defs nfinite-conv-nlength-enat pfilt-nlength)

lemma *OPJ01sem*:

$\sigma \models (f \text{ oproj } g) \longrightarrow \text{ inf}$

by (auto simp add: oprojection-d-def infinite-defs nfinite-conv-nlength-enat pfilt-nlength)

lemma *PJ02sem*:

$\sigma \models (f \text{ fproj } g) = ((f \wedge \text{ finite}) \text{ fproj } g)$

by (auto simp add: fprojection-d-def finite-defs powerinterval-def nfinite-conv-nlength-enat pfilt-nlength)

(metis enat-ord-simps(2) nfinite-nlength-enat nsubn-nfinite)

lemma *OPJ02sem*:

$\sigma \models (f \text{ oproj } g) = ((f \wedge \text{ finite}) \text{ oproj } g)$

by (auto simp add: oprojection-d-def finite-defs powerinterval-def nfinite-conv-nlength-enat pfilt-nlength)

(metis nfinite-conv-nlength-enat nfinite-ntaken nsubn-def1)

lemma *PJ03sem*:

$\sigma \models (f \text{ fproj } g) = (f \text{ fproj } (g \wedge \text{ finite}))$

by (auto simp add: fprojection-d-def finite-defs powerinterval-def nfinite-conv-nlength-enat pfilt-nlength)

lemma *OPJ03sem*:

$\sigma \models (f \text{ oproj } g) = (f \text{ oproj } (g \wedge \text{inf}))$

by (*auto simp add: oprojection-d-def infinite-defs powerinterval-def nfinite-conv-nlength-enat pfilt-nlength*)

PJ1

lemma *PJ1sem*:

$(\sigma \models f \text{ fproj } (g \vee h) = (f \text{ fproj } g \vee f \text{ fproj } h))$

by (*simp add: fprojection-d-def*) *blast*

lemma *OPJ1sem*:

$(\sigma \models f \text{ oproj } (g \vee h) = (f \text{ oproj } g \vee f \text{ oproj } h))$

by (*simp add: oprojection-d-def*) *blast*

PJ2

lemma *PJ2sem*:

$(\sigma \models f \text{ fproj } \text{empty} = \text{empty})$

proof *auto*

show $(\sigma \models f \text{ fproj } \text{empty}) \implies \sigma \models \text{empty}$

unfolding *fprojection-d-def empty-defs*

by (*metis enat.distinct(2) enat-the-enat nfinite-nlength-enat nnth-nlast pfilt-nlength the-enat-0*)

show $\sigma \models \text{empty} \implies (\sigma \models f \text{ fproj } \text{empty})$

unfolding *fprojection-d-def empty-defs powerinterval-def nidx-expand*

by (*metis enat-ord-simps(2) leD nfinite-conv-nlength-enat nlast-NNil nlength-NNil nnth-NNil not-iless0 pfilt-nlength the-enat-0 zero-enat-def zero-less-Suc*)

qed

lemma *OPJ2sem*:

$(\sigma \models \neg(f \text{ oproj } \text{empty}))$

unfolding *oprojection-d-def empty-defs pfilt-nlength*

using *nlength-eq-enat-nfiniteD* **by** (*simp add: zero-enat-def*) *blast*

PJ3

lemma *PJ3help*:

assumes *nfinite* σ

shows $\text{nsubn } \sigma \ 0 \ (\text{the-enat } (\text{nlength } \sigma)) = \sigma$

using *assms*

by (*metis diff-zero dual-order.refl ndropn-0 nfinite-conv-nlength-enat nsubn-def1 ntaken-all the-enat.simps*)

lemma *PJ3help1*:

assumes $f \ \sigma \wedge 0 < \text{nlength } \sigma \wedge \text{nfinite } \sigma$

shows $(\exists l. \text{nidx } l \wedge \text{nnth } l \ 0 = 0 \wedge \text{nfinite } l \wedge \text{nfinite } \sigma \wedge$

$\text{nlast } l = (\text{the-enat } (\text{nlength } \sigma)) \wedge$

$(\forall i < \text{nlength } l. f \ (\text{nsubn } \sigma \ (\text{nnth } l \ i)) \ (\text{nnth } l \ (\text{Suc } i))) \wedge$

$(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } l \wedge$

$(\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 \ i = \text{nnth } \sigma \ (\text{nnth } l \ i)) \wedge \text{nlength } \sigma 1 = \text{Suc } 0))$

proof –

have 1: $\text{nidx } (NCons\ 0\ (NNil\ (the-enat(nlength\ \sigma))))$
using *assms unfolding nidx-expand by simp*
 $(metis\ Suc-ile-eq\ assms\ enat-0-iff(1)\ enat-ord-simps(2)\ iless-eSuc0\ nfinite-nlength-enat\ nnth-0\ nnth-NNil\ the-enat.simps)$
have 2: $\text{nnth } (NCons\ 0\ (NNil\ (the-enat(nlength\ \sigma))))\ ((the-enat(nlength\ (NCons\ 0\ (NNil\ (nlength\ \sigma))))))$
 $=$
 $(the-enat\ (nlength\ \sigma))$
by $(metis\ nfinite-NCons\ nfinite-NNil\ nlast-NCons\ nlast-NNil\ nlength-NCons\ nlength-NNil\ nnth-nlast)$
have 3: $(\forall i < nlength\ (NCons\ 0\ (NNil\ (the-enat(nlength\ \sigma))))).$
 $f\ (nsbn\ \sigma\ (\text{nnth } (NCons\ 0\ (NNil\ (the-enat(nlength\ \sigma))))\ i)$
 $(\text{nnth } (NCons\ 0\ (NNil\ (the-enat(nlength\ \sigma))))\ (Suc\ i))\)$
using *assms PJ3help*
by $(metis\ enat-0-iff(1)\ iless-eSuc0\ nlength-NCons\ nlength-NNil\ nnth-0\ nnth-NNil\ nnth-Suc-NCons)$
have 4: $nlength\ (NCons\ (\text{nnth } \sigma\ (0))\ (NNil\ (\text{nnth } \sigma\ (the-enat(nlength\ \sigma)))) =$
 $nlength\ (NCons\ 0\ (NNil\ (the-enat(nlength\ \sigma))))$
by *simp*
have 5: $(\forall i \leq nlength\ (NCons\ (\text{nnth } \sigma\ (0))\ (NNil\ (\text{nnth } \sigma\ (the-enat(nlength\ \sigma))))).$
 $\text{nnth } (NCons\ (\text{nnth } \sigma\ (0))\ (NNil\ (\text{nnth } \sigma\ (the-enat(nlength\ \sigma))))\ i =$
 $\text{nnth } \sigma\ (\text{nnth } (NCons\ 0\ (NNil\ (the-enat(nlength\ \sigma))))\ i)$
by $(metis\ iless-Suc-eq\ le-zero-eq\ ndropn-nlast\ nfinite-NCons\ nfinite-NNil\ nlast-NCons\ nlength-NCons\ nlength-NNil\ nnth-0\ nnth-NNil\ nnth-zero-ndropn\ order-neq-le-trans\ the-enat.simps\ the-enat-0)$
have 6: $nlength\ (NCons\ (\text{nnth } \sigma\ (0))\ (NNil\ (\text{nnth } \sigma\ (the-enat(nlength\ \sigma)))) = Suc\ 0$
using *one-eSuc one-enat-def by auto*
have 7: $nfinite\ (NCons\ 0\ (NNil\ (the-enat(nlength\ \sigma))))$
by *simp*
have 8: $nlast\ (NCons\ 0\ (NNil\ (the-enat(nlength\ \sigma)))) = (the-enat\ (nlength\ \sigma))$
by *simp*
have 9: $\text{nnth } (NCons\ 0\ (NNil\ (the-enat(nlength\ \sigma))))\ 0 = 0$
using *nnth-0 by simp*
show *?thesis*
using *1 2 3 4 5 6 7 8 9 assms by blast*
qed

lemma *PJ3sem:*

$(\sigma \models f\ fproj\ skip = (f \wedge more \wedge finite))$

proof –

have 1: $(\sigma \models f\ fproj\ skip) \implies (\sigma \models f \wedge more \wedge finite)$

unfolding *cppl-fprojection pfilt-expand nidx-expand skip-defs more-defs finite-defs powerinterval-def*

by $(metis\ (no-types,\ lifting)\ One-nat-def\ PJ3help\ eSuc-enat\ i0-less\ ileI1\ intensional-rews(3)$

$less-numeral-extra(1)\ less-numeral-extra(3)\ nnth-nlast\ pfilt-nlength\ the-enat.simps\ zero-enat-def)$

have 2: $(\sigma \models f \wedge more \wedge finite) \implies (\sigma \models f\ fproj\ skip)$

by $(simp\ add:\ fprojection-d-def\ finite-defs\ skip-defs\ more-defs\ powerinterval-def\ pfilt-nlength\)$

$(metis\ PJ3help1\ i0-less\ nfinite-conv-nlength-enat\ the-enat.simps)$

show *?thesis*

using *1 2 unl-lift2 by blast*

qed

lemma *OPJ3sem:*

$(\sigma \models \neg(f \text{ oproj } \text{skip}))$)
unfolding *oprojection-d-def skip-defs pfilt-nlength*
using *nlength-eq-enat-nfiniteD* **by** *auto*

PJ4

lemma *PJ4semchaina*:

assumes $(\sigma \models f \text{ fproj } (g;h))$

shows $(\sigma \models (f \text{ fproj } g) ; (f \text{ fproj } h))$

proof –

have 1: $(\sigma \models f \text{ fproj } (g;h))$

using *assms* **by** *auto*

have 2: $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge$

$\text{nlast } ls = (\text{the-enat } (\text{nlength } \sigma)) \wedge$

$\text{powerinterval } f \ \sigma \ ls \wedge$

$(\exists n \leq \text{nlength } (\text{pfilt } \sigma \ ls). \ g \ (\text{ntaken } n \ (\text{pfilt } \sigma \ ls)) \wedge h \ (\text{ndropn } n \ (\text{pfilt } \sigma \ ls))))$

using *assms* **unfolding** *chop-defs fprojection-d-def*

by *(metis nfinite-conv-nlength-enat pfilt-nlength the-enat.simps)*

obtain *ls* **where** 3: $\text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge$

$\text{nlast } ls = (\text{the-enat } (\text{nlength } \sigma)) \wedge$

$\text{powerinterval } f \ \sigma \ ls \wedge$

$(\exists n \leq \text{nlength } (\text{pfilt } \sigma \ ls). \ g \ (\text{ntaken } n \ (\text{pfilt } \sigma \ ls)) \wedge h \ (\text{ndropn } n \ (\text{pfilt } \sigma \ ls))))$

using 2 **by** *auto*

have 4: $\text{nidx } ls \wedge \text{nnth } ls \ 0 = 0$

using 3 **by** *auto*

have 5: $\text{powerinterval } f \ \sigma \ ls$

using 3 **by** *auto*

have 6: $\text{nlast } ls = \text{the-enat } (\text{nlength } \sigma)$

using 3 **by** *auto*

have 7: $(\exists n \leq \text{nlength } (\text{pfilt } \sigma \ ls). \ g \ (\text{ntaken } n \ (\text{pfilt } \sigma \ ls)) \wedge h \ (\text{ndropn } n \ (\text{pfilt } \sigma \ ls)))$

using 3 **by** *auto*

obtain *n* **where** 8: $n \leq \text{nlength } (\text{pfilt } \sigma \ ls) \wedge g \ (\text{ntaken } n \ (\text{pfilt } \sigma \ ls)) \wedge h \ (\text{ndropn } n \ (\text{pfilt } \sigma \ ls))$

using 7 **by** *auto*

have 9: $n \leq \text{nlength } (\text{pfilt } \sigma \ ls)$

using 8 **by** *auto*

have 10: $g \ (\text{ntaken } n \ (\text{pfilt } \sigma \ ls))$

using 8 **by** *auto*

have 11: $h \ (\text{ndropn } n \ (\text{pfilt } \sigma \ ls))$

using 8 **by** *auto*

have 12: $\text{nidx } (\text{ntaken } n \ ls) \wedge \text{nnth } (\text{ntaken } n \ ls) \ 0 = 0$

using 4 8 **unfolding** *nidx-expand pfilt-nlength* **by** *simp*

(metis Suc-leD bot-nat-0.extremum dual-order.trans enat-ord-simps(1) min.orderE ntaken-nnth)

have 13: $\text{nidx } (\text{ndropn } n \ ls) \wedge \text{nnth } (\text{ndropn } n \ ls) \ 0 = (\text{nnth } ls \ n)$

using 4 9 **unfolding** *pfilt-nlength nidx-expand*

using *nidx-expand nidx-ndropn* **by** *auto*

have 131: $((\text{nmap } (\lambda x. \ x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls))) = \text{ndropn } n \ (\text{nmap } (\lambda x. \ x - \text{nnth } ls \ n) \ ls)$

using *ndropn-nmap[of n ($\lambda x. \ x - (\text{nnth } ls \ n)) \ ls]$* **by** *presburger*

have 132: $\text{nnth } (\text{ndropn } n \ (\text{nmap } (\lambda x. \ x - \text{nnth } ls \ n) \ ls)) \ 0 = 0$

by *(metis 13 131 diff-self-eq-0 nnth-nmap zero-enat-def zero-le)*

have 133: $\bigwedge j. \text{enat } j \leq \text{nlength } ls \implies \text{nnth } (\text{nmap } (\lambda x. \ x - \text{nnth } ls \ n) \ ls) \ j = \text{nnth } ls \ j - \text{nnth } ls \ n$

```

using nnth-nmap by blast
have 134:  $nlength\ (ndropn\ n\ (nmap\ (\lambda x. x - nnth\ ls\ n)\ ls)) = nlength\ ls - enat\ n$ 
by simp
have 135:  $\bigwedge j. j \leq nlength\ ls - enat\ n \implies$ 
 $nnth\ (ndropn\ n\ (nmap\ (\lambda x. x - nnth\ ls\ n)\ ls))\ j = nnth\ ls\ (n + j) - nnth\ ls\ n$ 
by (metis 131 ndropn-nlength ndropn-nnth nnth-nmap)
have 136:  $\bigwedge j. (Suc\ j) \leq nlength\ ls - enat\ n \implies$ 
 $nnth\ (ndropn\ n\ (nmap\ (\lambda x. x - nnth\ ls\ n)\ ls))\ (Suc\ j) = nnth\ ls\ (n + (Suc\ j)) - nnth\ ls\ n$ 
using 135 by blast
have 137:  $\bigwedge j. (Suc\ j) \leq nlength\ ls - enat\ n \implies$ 
 $nnth\ ls\ (n + j) - nnth\ ls\ n < nnth\ ls\ (n + (Suc\ j)) - nnth\ ls\ n$ 
by (metis 13 Suc-ile-eq diff-less-mono le-add1 le-add-same-cancel1 ndropn-nlength ndropn-nnth
nidx-expand nidx-less-eq order-less-imp-le)
have 14:  $nidx\ (ndropn\ n\ (nmap\ (\lambda x. x - nnth\ ls\ n)\ ls))$ 
by (metis 134 135 137 Suc-ile-eq dual-order.strict-iff-order nidx-expand)
have 15:  $g\ (pfilt\ \sigma\ (ntaken\ n\ ls))$ 
by (metis 10 9 pfilt-nlength pfilt-ntaken)
have 16:  $h\ (pfilt\ \sigma\ (ndropn\ n\ ls))$ 
by (metis 8 pfilt-ndropn pfilt-nlength)
have 17:  $g\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ \sigma)\ (ntaken\ n\ ls))$ 
by (metis 12 15 nfinite-ntaken nle-le ntaken-all ntaken-nlast pfilt-ntaken-nidx)
have 170:  $nmap\ (\lambda x. nnth\ \sigma\ (nnth\ ls\ n + x))\ (nmap\ (\lambda x. x - nnth\ ls\ n)\ (ndropn\ n\ ls)) =$ 
 $nmap\ ((\lambda x. nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x. x - nnth\ ls\ n))\ (ndropn\ n\ ls)$ 
using nellist.map-comp by blast
have 171:  $nmap\ ((\lambda x. nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x. x - nnth\ ls\ n))\ (ndropn\ n\ ls) =$ 
 $nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls)$ 
proof –
have 172:  $nlength\ (nmap\ ((\lambda x. nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x. x - nnth\ ls\ n))\ (ndropn\ n\ ls)) =$ 
 $nlength\ (nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls))$ 
by auto
have 173:  $\bigwedge j. j \leq nlength\ (nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls)) \implies$ 
 $nnth\ (nmap\ ((\lambda x. nnth\ \sigma\ (nnth\ ls\ n + x)) \circ (\lambda x. x - nnth\ ls\ n))\ (ndropn\ n\ ls))\ j =$ 
 $nnth\ (nmap\ (nnth\ \sigma)\ (ndropn\ n\ ls))\ j$ 
by auto
(metis 13 131 134 le-add1 le-add-diff-inverse le-add-same-cancel1 ndropn-nnth
nidx-less-eq nlength-nmap)
show ?thesis
by (metis 172 173 nellist-eq-nnth-eq)
qed
have 171:  $(pfilt\ (ndropn\ (nnth\ ls\ n)\ \sigma)\ ((nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls)))) =$ 
 $(pfilt\ \sigma\ (ndropn\ n\ ls))$ 
using pfilt-nmap[of (ndropn (nnth ls n) σ) ((nmap (λx. x - (nnth ls n)) (ndropn n ls)))]
pfilt-nmap[of σ (ndropn n ls)] 170 171 by force

have 18:  $h\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ \sigma)\ ((nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))))$ 
using 16 171 by auto
have 19:  $powerinterval\ f\ (ntaken\ (nnth\ ls\ n)\ \sigma)\ (ntaken\ n\ ls)$ 
by (metis 3 9 nfinite-conv-nlength-enat pfilt-nlength powerinterval-splita)
have 20:  $powerinterval\ f\ (ndropn\ (nnth\ ls\ n)\ \sigma)\ ((nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls)))$ 
using powerinterval-split[of ls n σ f]

```



```

  by (metis 3 9 pfilt-nlength)
have 21: (nnth ls n) ≤ nlength σ
  by (metis 3 9 enat-ord-simps(2) nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast
      order.order-iff-strict pfilt-nlength the-enat.simps)
have 22: nnth (ntaken n ls) (the-enat(nlength (ntaken n ls))) = (nnth ls n)
  by (metis 9 min-def ntaken-nlength ntaken-nnth pfilt-nlength the-enat.simps)
have 222: nlast (ntaken n ls) = (nnth ls n)
  by (simp add: ntaken-nlast)
have 23: nnth ((nmap (λx. x - (nnth ls n)) (ndropn n ls)))
  (the-enat(nlength ((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) =
  (the-enat(nlength σ)) - (nnth ls n)
  by (metis 171 222 3 9 enat-le-plus-same(2) enat-ord-simps(3) enat-the-enat gen-nlength-def
      ndropn-nfirst nfinite-ndropn-a nfinite-ntaken nfuse-ntaken-ndropn nlast-nfuse nlength-code
      nnth-nlast nnth-nmap pfilt-nlength)
have 223: nlast ((nmap (λx. x - (nnth ls n)) (ndropn n ls))) = (the-enat(nlength σ)) - (nnth ls n)
  by (metis 23 3 nfinite-ndropn-a nfinite-nmap nnth-nlast)
have 224: (nmap (λx. x + (nnth ls n)) ((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) = ndropn n ls
  proof -
    have 2241: nlength (nmap (λx. x + (nnth ls n)) ((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) =
      nlength (ndropn n ls)
    by auto
    have 2242: ∧j. j ≤ nlength (ndropn n ls) →
      nnth (nmap (λx. x + (nnth ls n)) ((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) j =
      nnth (ndropn n ls) j
    by (metis 13 eq-imp-le le-add-diff-inverse2 nidx-gr-first nlength-nmap nnth-nmap
        not-gr-zero order-less-imp-le)
    show ?thesis using 2241 2242 nelist-eq-nnth-eq
    by metis
  qed
have 24: ls = nfuse (ntaken n ls)
  (nmap (λx. x + (nnth ls n)) ((nmap (λx. x - (nnth ls n)) (ndropn n ls))))
  using nfuse-ntaken-ndropn
  by (metis 224 9 pfilt-nlength)
have 241: nfinite (ntaken n ls)
  by simp
have 242: nfinite (ntaken (nnth ls n) σ)
  by simp
have 243: nlast (ntaken n ls) = (the-enat(nlength (ntaken (nnth ls n) σ)))
  by (simp add: 21 222)
have 25: (∃ ls1. nidx ls1 ∧ nnth ls1 0 = 0 ∧ nfinite ls1 ∧ nfinite (ntaken (nnth ls n) σ) ∧
  nlast ls1 = (the-enat(nlength (ntaken (nnth ls n) σ))) ∧
  powerinterval f (ntaken (nnth ls n) σ) ls1 ∧ g (pfilt (ntaken (nnth ls n) σ) ls1))
  using 12 17 19 241 242 243 by blast
have 251: nfinite (ndropn (nnth ls n) σ)
  by (simp add: 3)
have 252: nlast (nmap (λx. x - nnth ls n) (ndropn n ls)) = the-enat(nlength (ndropn (nnth ls n) σ))
  by (metis 223 3 idiff-enat-enat ndropn-nlength nfinite-nlength-enat the-enat.simps)
have 26: (∃ ls2. nidx ls2 ∧ nnth ls2 0 = 0 ∧ nfinite ls2 ∧ nfinite (ndropn (nnth ls n) σ) ∧
  nlast ls2 = the-enat(nlength (ndropn (nnth ls n) σ)) ∧
  powerinterval f (ndropn (nnth ls n) σ) ls2 ∧ h (pfilt (ndropn (nnth ls n) σ) ls2))

```

```

  by (metis 131 132 14 18 20 252 3 nfinite-ndropn-a nfinite-nmap)
have 27: (ntaken (nnth ls n)  $\sigma$ )  $\models$   $f$  fproj  $g$ 
  by (metis 25 fprojection-d-def)
have 28: (ndropn (nnth ls n)  $\sigma$ )  $\models$   $f$  fproj  $h$ 
  by (metis 26 fprojection-d-def nfinite-nlength-enat the-enat.simps)
show ?thesis
using 21 27 28 chop-defs by auto
qed

```

lemma *OPJ4semchaina*:

```

assumes ( $\sigma \models f$  oproj ( $(g \wedge \text{finite}); h$ ))
shows ( $\sigma \models (f$  fproj  $g$ ) ; ( $f$  oproj  $h$ ))
proof -
  have 1: ( $\sigma \models f$  oproj ( $(g \wedge \text{finite}); h$ ))
  using assms by auto
  have 2: ( $\exists$   $ls$ .  $nidx$   $ls \wedge nnth$   $ls$   $0 = 0 \wedge \neg nfinite$   $ls \wedge \neg nfinite$   $\sigma \wedge$ 
    powerinterval  $f$   $\sigma$   $ls \wedge$ 
    ( $\exists n \leq nlength$  ( $pfilt$   $\sigma$   $ls$ ).  $g$  ( $ntaken$   $n$  ( $pfilt$   $\sigma$   $ls$ ))  $\wedge h$  ( $ndropn$   $n$  ( $pfilt$   $\sigma$   $ls$ ))))
  using assms unfolding chop-defs oprojection-d-def finite-defs
  by auto
  obtain  $ls$  where 3:  $nidx$   $ls \wedge nnth$   $ls$   $0 = 0 \wedge \neg nfinite$   $ls \wedge \neg nfinite$   $\sigma \wedge$ 
    powerinterval  $f$   $\sigma$   $ls \wedge$ 
    ( $\exists n \leq nlength$  ( $pfilt$   $\sigma$   $ls$ ).  $g$  ( $ntaken$   $n$  ( $pfilt$   $\sigma$   $ls$ ))  $\wedge h$  ( $ndropn$   $n$  ( $pfilt$   $\sigma$   $ls$ )))
  using 2 by auto
  have 4:  $nidx$   $ls \wedge nnth$   $ls$   $0 = 0$ 
  using 3 by auto
  have 5: powerinterval  $f$   $\sigma$   $ls$ 
  using 3 by auto
  have 7: ( $\exists n \leq nlength$  ( $pfilt$   $\sigma$   $ls$ ).  $g$  ( $ntaken$   $n$  ( $pfilt$   $\sigma$   $ls$ ))  $\wedge h$  ( $ndropn$   $n$  ( $pfilt$   $\sigma$   $ls$ )))
  using 3 by auto
  obtain  $n$  where 8:  $n \leq nlength$  ( $pfilt$   $\sigma$   $ls$ )  $\wedge g$  ( $ntaken$   $n$  ( $pfilt$   $\sigma$   $ls$ ))  $\wedge h$  ( $ndropn$   $n$  ( $pfilt$   $\sigma$   $ls$ ))
  using 7 by auto
  have 9:  $n \leq nlength$  ( $pfilt$   $\sigma$   $ls$ )
  using 8 by auto
  have 10:  $g$  ( $ntaken$   $n$  ( $pfilt$   $\sigma$   $ls$ ))
  using 8 by auto
  have 11:  $h$  ( $ndropn$   $n$  ( $pfilt$   $\sigma$   $ls$ ))
  using 8 by auto
  have 12:  $nidx$  ( $ntaken$   $n$   $ls$ )  $\wedge nnth$  ( $ntaken$   $n$   $ls$ )  $0 = 0$ 
  using 4 8 unfolding nidx-expand pfilt-nlength by simp
  (metis 3 Suc-leD bot-nat-0.extremum min.orderE nfinite-ntaken nle-le ntaken-all ntaken-nnth)
  have 13:  $nidx$  ( $ndropn$   $n$   $ls$ )  $\wedge nnth$  ( $ndropn$   $n$   $ls$ )  $0 = (nnth$   $ls$   $n$ )
  using 4 9 unfolding pfilt-nlength nidx-expand
  using nidx-expand nidx-ndropn by auto
  have 131: ( $(nmap$  ( $\lambda x$ .  $x - (nnth$   $ls$   $n$ )) ( $ndropn$   $n$   $ls$ ))) =  $ndropn$   $n$  ( $nmap$  ( $\lambda x$ .  $x - nnth$   $ls$   $n$ )  $ls$ )
  using ndropn-nmap[of  $n$  ( $\lambda x$ .  $x - (nnth$   $ls$   $n$ ))  $ls$ ] by presburger
  have 132:  $nnth$  ( $ndropn$   $n$  ( $nmap$  ( $\lambda x$ .  $x - nnth$   $ls$   $n$ )  $ls$ ))  $0 = 0$ 
  by (metis 13 131 diff-self-eq-0 nnth-nmap zero-enat-def zero-le)
  have 133:  $\bigwedge j$ .  $enat$   $j \leq nlength$   $ls \implies nnth$  ( $nmap$  ( $\lambda x$ .  $x - nnth$   $ls$   $n$ )  $ls$ )  $j = nnth$   $ls$   $j - nnth$   $ls$   $n$ 
  using nnth-nmap by blast

```

have 134: $\text{nlength } (\text{ndropn } n \text{ (nmap } (\lambda x. x - \text{nnth } ls \ n) \ ls)) = \text{nlength } ls - \text{enat } n$
by simp
have 135: $\bigwedge j. j \leq \text{nlength } ls - \text{enat } n \implies$
 $\text{nnth } (\text{ndropn } n \text{ (nmap } (\lambda x. x - \text{nnth } ls \ n) \ ls)) \ j = \text{nnth } ls \ (n + j) - \text{nnth } ls \ n$
by (metis 131 ndropn-nlength ndropn-nnth nnth-nmap)
have 136: $\bigwedge j. (\text{Suc } j) \leq \text{nlength } ls - \text{enat } n \implies$
 $\text{nnth } (\text{ndropn } n \text{ (nmap } (\lambda x. x - \text{nnth } ls \ n) \ ls)) \ (\text{Suc } j) = \text{nnth } ls \ (n + (\text{Suc } j)) - \text{nnth } ls \ n$
using 135 by blast
have 137: $\bigwedge j. (\text{Suc } j) \leq \text{nlength } ls - \text{enat } n \implies$
 $\text{nnth } ls \ (n + j) - \text{nnth } ls \ n < \text{nnth } ls \ (n + (\text{Suc } j)) - \text{nnth } ls \ n$
by (metis 13 Suc-ile-eq diff-less-mono le-add1 le-add-same-cancel1 ndropn-nlength ndropn-nnth
 $\text{nidx-expand nidx-less-eq order-less-imp-le})$
have 14: $\text{nidx } (\text{ndropn } n \text{ (nmap } (\lambda x. x - \text{nnth } ls \ n) \ ls))$
by (metis 134 135 137 Suc-ile-eq dual-order.order-iff-strict nidx-expand)
have 15: $g \text{ (pfilt } \sigma \text{ (ntaken } n \text{ } ls))$
by (metis 10 9 pfilt-nlength pfilt-ntaken)
have 16: $h \text{ (pfilt } \sigma \text{ (ndropn } n \text{ } ls))$
by (metis 8 pfilt-ndropn pfilt-nlength)
have 17: $g \text{ (pfilt (ntaken (nnth } ls \ n) \ \sigma) (ntaken \ n \text{ } ls))$
by (metis 12 15 nfinite-ntaken nle-le ntaken-all ntaken-nlast pfilt-ntaken-nidx)
have 170: $\text{nmap } (\lambda x. \text{nnth } \sigma \text{ (nnth } ls \ n + x)) \text{ (nmap } (\lambda x. x - \text{nnth } ls \ n) \text{ (ndropn } n \text{ } ls)) =$
 $\text{nmap } ((\lambda x. \text{nnth } \sigma \text{ (nnth } ls \ n + x)) \circ (\lambda x. x - \text{nnth } ls \ n)) \text{ (ndropn } n \text{ } ls)$
using nellist.map-comp by blast
have 171: $\text{nmap } ((\lambda x. \text{nnth } \sigma \text{ (nnth } ls \ n + x)) \circ (\lambda x. x - \text{nnth } ls \ n)) \text{ (ndropn } n \text{ } ls) =$
 $\text{nmap } (\text{nnth } \sigma) \text{ (ndropn } n \text{ } ls)$
proof –
have 172: $\text{nlength } (\text{nmap } ((\lambda x. \text{nnth } \sigma \text{ (nnth } ls \ n + x)) \circ (\lambda x. x - \text{nnth } ls \ n)) \text{ (ndropn } n \text{ } ls)) =$
 $\text{nlength } (\text{nmap } (\text{nnth } \sigma) \text{ (ndropn } n \text{ } ls))$
by auto
have 173: $\bigwedge j. j \leq \text{nlength } (\text{nmap } (\text{nnth } \sigma) \text{ (ndropn } n \text{ } ls)) \implies$
 $\text{nnth } (\text{nmap } ((\lambda x. \text{nnth } \sigma \text{ (nnth } ls \ n + x)) \circ (\lambda x. x - \text{nnth } ls \ n)) \text{ (ndropn } n \text{ } ls)) \ j =$
 $\text{nnth } (\text{nmap } (\text{nnth } \sigma) \text{ (ndropn } n \text{ } ls)) \ j$
by auto
 $(\text{metis 13 131 134 le-add1 le-add-diff-inverse le-add-same-cancel1 ndropn-nnth}$
 $\text{nidx-less-eq nlength-nmap})$
show ?thesis
by (metis 172 173 nellist-eq-nnth-eq)
qed
have 171: $(\text{pfilt } (\text{ndropn } (\text{nnth } ls \ n) \ \sigma) ((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \text{ (ndropn } n \text{ } ls)))) =$
 $(\text{pfilt } \sigma \text{ (ndropn } n \text{ } ls))$
using pfilt-nmap[of (ndropn (nnth ls n) σ) ((nmap (λx. x - (nnth ls n)) (ndropn n ls)))]
 $\text{pfilt-nmap[of } \sigma \text{ (ndropn } n \text{ } ls)] \text{ 170 171 by force}$
have 18: $h \text{ (pfilt (ndropn (nnth } ls \ n) \ \sigma) ((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \text{ (ndropn } n \text{ } ls))))$
using 16 171 by auto
have 19: $\text{powerinterval } f \text{ (ntaken (nnth } ls \ n) \ \sigma) \text{ (ntaken } n \text{ } ls)$
by (metis 3 8 pfilt-nlength powerinterval-splita-alt)
have 20: $\text{powerinterval } f \text{ (ndropn (nnth } ls \ n) \ \sigma) ((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \text{ (ndropn } n \text{ } ls))))$
using powerinterval-split[of ls n σ f]
by (metis 3 8 pfilt-nlength powerinterval-splitb-alt)
have 21: $(\text{nnth } ls \ n) \leq \text{nlength } \sigma$

by (metis 3 linorder-le-cases nfinite-ntaken ntaken-all)
 have 22: $\text{nnth } (\text{ntaken } n \text{ } ls) (\text{the-enat}(\text{nlength } (\text{ntaken } n \text{ } ls))) = (\text{nnth } ls \text{ } n)$
 by (metis 9 min-def ntaken-nlength ntaken-nnth pfilt-nlength the-enat.simps)
 have 222: $\text{nlast } (\text{ntaken } n \text{ } ls) = (\text{nnth } ls \text{ } n)$
 by (simp add: ntaken-nlast)
 have 224: $(\text{nmap } (\lambda x. x + (\text{nnth } ls \text{ } n)) ((\text{nmap } (\lambda x. x - (\text{nnth } ls \text{ } n)) (\text{ndropn } n \text{ } ls)))) = \text{ndropn } n \text{ } ls$
 proof –
 have 2241: $\text{nlength } (\text{nmap } (\lambda x. x + (\text{nnth } ls \text{ } n)) ((\text{nmap } (\lambda x. x - (\text{nnth } ls \text{ } n)) (\text{ndropn } n \text{ } ls)))) = \text{nlength } (\text{ndropn } n \text{ } ls)$
 by auto
 have 2242: $\bigwedge j. j \leq \text{nlength } (\text{ndropn } n \text{ } ls) \longrightarrow \text{nnth } (\text{nmap } (\lambda x. x + (\text{nnth } ls \text{ } n)) ((\text{nmap } (\lambda x. x - (\text{nnth } ls \text{ } n)) (\text{ndropn } n \text{ } ls)))) j = \text{nnth } (\text{ndropn } n \text{ } ls) j$
 by (metis 13 132 9 diff-is-0-eq le-add-diff-inverse2 nidx-gr-first nlength-nmap nnth-nmap nnth-zero-ndropn not-gr-zero order-less-imp-le pfilt-nlength)
 show ?thesis using 2241 2242 nellist-eq-nnth-eq
 by metis
 qed
 have 24: $ls = \text{nfuse } (\text{ntaken } n \text{ } ls) (\text{nmap } (\lambda x. x + (\text{nnth } ls \text{ } n)) ((\text{nmap } (\lambda x. x - (\text{nnth } ls \text{ } n)) (\text{ndropn } n \text{ } ls))))$
 using nfuse-ntaken-ndropn
 by (metis 224 9 pfilt-nlength)
 have 25: $(\exists ls1. \text{nidx } ls1 \wedge \text{nnth } ls1 \text{ } 0 = 0 \wedge \text{nfinite } ls1 \wedge \text{nfinite } (\text{ntaken } (\text{nnth } ls \text{ } n) \text{ } \sigma) \wedge \text{nlast } ls1 = (\text{the-enat}(\text{nlength } (\text{ntaken } (\text{nnth } ls \text{ } n) \text{ } \sigma))) \wedge \text{powerinterval } f (\text{ntaken } (\text{nnth } ls \text{ } n) \text{ } \sigma) \text{ } ls1 \wedge g (\text{pfilt } (\text{ntaken } (\text{nnth } ls \text{ } n) \text{ } \sigma) \text{ } ls1))$
 using 12 17 19 21 222 3
 by (metis min.orderE nfinite-ntaken ntaken-nlength the-enat.simps)
 have 26: $(\exists ls2. \text{nidx } ls2 \wedge \text{nnth } ls2 \text{ } 0 = 0 \wedge \neg \text{nfinite } ls2 \wedge \neg \text{nfinite } (\text{ndropn } (\text{nnth } ls \text{ } n) \text{ } \sigma) \wedge \text{powerinterval } f (\text{ndropn } (\text{nnth } ls \text{ } n) \text{ } \sigma) \text{ } ls2 \wedge h (\text{pfilt } (\text{ndropn } (\text{nnth } ls \text{ } n) \text{ } \sigma) \text{ } ls2))$
 using assms 14 18 20 3 132 131 by (metis nfinite-ndropn nfinite-nmap)
 have 27: $(\text{ntaken } (\text{nnth } ls \text{ } n) \text{ } \sigma) \models f \text{ } fproj \text{ } g$
 by (metis 25 fprojection-d-def)
 have 28: $(\text{ndropn } (\text{nnth } ls \text{ } n) \text{ } \sigma) \models f \text{ } oproj \text{ } h$
 by (metis 26 oprojection-d-def)
 show ?thesis
 using 21 27 28 chop-defs by auto
 qed

lemma PJ4semchainb:

assumes $(\sigma \models (f \text{ } fproj \text{ } g) ; (f \text{ } fproj \text{ } h))$

shows $(\sigma \models f \text{ } fproj \text{ } (g;h))$

proof –

have 1: $(\exists n \leq \text{nlength } \sigma.$

$(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \text{ } 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } (\text{ntaken } n \text{ } \sigma) \wedge \text{nlast } ls = (\text{the-enat}(\text{nlength } (\text{ntaken } n \text{ } \sigma))) \wedge \text{powerinterval } f (\text{ntaken } n \text{ } \sigma) \text{ } ls \wedge g (\text{pfilt } (\text{ntaken } n \text{ } \sigma) \text{ } ls)) \wedge$

$(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \text{ } 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } (\text{ndropn } n \text{ } \sigma) \wedge \text{nlast } ls = (\text{the-enat}(\text{nlength } (\text{ndropn } n \text{ } \sigma))) \wedge \text{powerinterval } f (\text{ndropn } n \text{ } \sigma) \text{ } ls \wedge h (\text{pfilt } (\text{ndropn } n \text{ } \sigma) \text{ } ls)))$

using assms unfolding chop-defs fprojection-d-def by simp blast

obtain cp **where** 2: $cp \leq nlength\ \sigma \wedge$
 $(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ntaken\ cp\ \sigma) \wedge$
 $nlast\ ls = (the-enat(nlength\ (ntaken\ cp\ \sigma))) \wedge$
 $powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls \wedge g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls)) \wedge$
 $(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ndropn\ cp\ \sigma) \wedge$
 $nlast\ ls = (the-enat(nlength\ (ndropn\ cp\ \sigma))) \wedge$
 $powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls \wedge h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls))$
using 1 **by** *auto*
have 3: $cp \leq nlength\ \sigma$
using 2 **by** *auto*
have 4: $(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ntaken\ cp\ \sigma) \wedge$
 $nlast\ ls = (the-enat(nlength\ (ntaken\ cp\ \sigma))) \wedge$
 $powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls \wedge g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls))$
using 2 **by** *auto*
obtain $ls1$ **where** 5: $nidx\ ls1 \wedge nnth\ ls1\ 0 = 0 \wedge nfinite\ ls1 \wedge nfinite\ (ntaken\ cp\ \sigma) \wedge$
 $nlast\ ls1 = (the-enat(nlength\ (ntaken\ cp\ \sigma))) \wedge$
 $powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls1 \wedge g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls1)$
using 4 **by** *auto*
have 6: $nidx\ ls1 \wedge nnth\ ls1\ 0 = 0$
using 5 **by** *auto*
have 61: $nfinite\ ls1 \wedge nfinite\ (ntaken\ cp\ \sigma)$
using 5 **by** *auto*
have 7: $nlast\ ls1 = the-enat(nlength\ (ntaken\ cp\ \sigma))$
using 5 **by** *auto*
have 8: $powerinterval\ f\ (ntaken\ cp\ \sigma)\ ls1$
using 5 **by** *auto*
have 9: $g\ (pfilt\ (ntaken\ cp\ \sigma)\ ls1)$
using 5 **by** *auto*
have 10: $(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ (ndropn\ cp\ \sigma) \wedge$
 $nlast\ ls = (the-enat(nlength\ (ndropn\ cp\ \sigma))) \wedge$
 $powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls \wedge h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls))$
using 2 **by** *auto*
obtain $ls2$ **where** 11: $nidx\ ls2 \wedge nnth\ ls2\ 0 = 0 \wedge nfinite\ ls2 \wedge nfinite\ (ndropn\ cp\ \sigma) \wedge$
 $nlast\ ls2 = (the-enat(nlength\ (ndropn\ cp\ \sigma))) \wedge$
 $powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls2 \wedge h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls2)$
using 10 **by** *auto*
have 12: $nidx\ ls2 \wedge nnth\ ls2\ 0 = 0$
using 11 **by** *auto*
have 13: $nlast\ ls2 = the-enat(nlength\ (ndropn\ cp\ \sigma))$
using 11 **by** *auto*
have 14: $powerinterval\ f\ (ndropn\ cp\ \sigma)\ ls2$
using 11 **by** *auto*
have 15: $h\ (pfilt\ (ndropn\ cp\ \sigma)\ ls2)$
using 11 **by** *auto*
have 150: $nlast\ ls1 = cp$
by (*simp add: 2 7*)
have 151: $nfirst\ (nmap\ (\lambda x. x + cp)\ ls2) = cp$
by (*metis 11 NNil-eq-nmap-conv nlast-NNil ntake-eq-NNil-iff ntake-nmap ntaken-0 ntaken-nlast plus-nat.add-0*)
have 152: $nnth\ (nfuse\ ls1\ (nmap\ (\lambda x. x + cp)\ ls2))\ 0 = 0$

by (metis 150 151 5 nfuse-nnth zero-enat-def zero-le)
 have 153: $\text{nidx } ls1 \wedge \text{nnth } ls1 \ 0 = 0 \wedge \text{nfinite } ls1 \wedge \text{nlast } ls1 = cp$
 by (simp add: 150 5)
 have 154: $\text{nidx } ls2 \wedge \text{nnth } ls2 \ 0 = 0 \wedge \text{nfinite } ls2 \wedge \text{nfinite } \sigma \wedge \text{nlast } ls2 = \text{the-enat } (\text{nlength } \sigma) - cp$
 by (metis 11 idiff-enat-enat ndropn-nlength nfinite-conv-nlength-enat nfinite-ndropn the-enat.simps)
 have 16: $\text{nidx } (\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2)) \wedge \text{nnth } (\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2)) \ 0 = 0$
 using $\text{nidx-nfuse-nidx[of } ls1 \ ls2 \ cp \ \sigma \] \ 153 \ 154 \ 3$ by fastforce
 have 17: $\text{nlast } ls1 = \text{nfirst } (\text{nmap } (\lambda x. x + cp) ls2)$
 by (simp add: 150 151)
 have 18: $\text{nnth } (\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2)) (\text{the-enat}(\text{nlength } ls1)) = cp$
 by (metis 151 17 5 ntaken-nfuse ntaken-nlast)
 have 19: $\text{nlast } (\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2)) = (\text{the-enat } (\text{nlength } \sigma))$
 by (metis 154 17 3 5 diff-add enat.distinct(2) enat-ord-simps(1) enat-the-enat
 nfinite-conv-nlength-enat nlast-nfuse nlast-nmap)
 have 20: $\text{powerinterval } f \ \sigma (\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2))$
 using $\text{powerinterval-nfuse[of } ls1 \ (ls2) \ cp \ \sigma \ f] \ 11 \ 150 \ 154 \ 3 \ 5$ by fastforce
 have 21: $\sigma = \text{nfuse } (\text{ntaken } cp \ \sigma) (\text{ndropn } cp \ \sigma)$
 by (simp add: 2 nfuse-ntaken-ndropn)
 have 22: $\text{nnth } ((\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2))) (\text{the-enat}(\text{nlength } ls1)) = cp$
 using 18 by blast
 have 23: $(\text{ntaken } (\text{the-enat}(\text{nlength } ls1)) (\text{pfilt } \sigma (\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2)))) =$
 $(\text{pfilt } (\text{ntaken } cp \ \sigma) \ ls1)$
 by (metis 151 17 2 5 enat.distinct(2) enat-le-plus-same(1) enat-the-enat
 nfinite-conv-nlength-enat nfuse-nlength ntaken-nfuse pfilt-ntaken pfilt-ntaken-nidx)
 have 24: $g (\text{ntaken } (\text{the-enat}(\text{nlength } ls1)) (\text{pfilt } \sigma (\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2))))$
 by (simp add: 23 9)
 have 25: $(\text{ndropn } (\text{the-enat}(\text{nlength } ls1)) (\text{pfilt } \sigma (\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2)))) =$
 $(\text{pfilt } (\text{ndropn } cp \ \sigma) \ ls2)$
 using $\text{pfilt-ndropn[of } (\text{the-enat}(\text{nlength } ls1)) (\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2)) \ \sigma]$
 $\text{pfilt-ndropn-nidx[of } cp \ \sigma \ ls2 \]$
 by (metis 12 17 23 3 5 enat-ord-code(4) enat-the-enat min-def
 ndropn-nfuse nle-le ntaken-nlength order-less-imp-le pfilt-nlength)
 have 26: $\text{nlength } ls1 \leq \text{nlength } (\text{pfilt } \sigma (\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2)))$
 by (simp add: nfuse-nlength pfilt-nlength)
 have 27: $(\exists \text{ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls} \ 0 = 0 \wedge \text{nfinite } \text{ls} \wedge \text{nfinite } \sigma \wedge$
 $\text{nlast } \text{ls} = \text{the-enat}(\text{nlength } \sigma) \wedge$
 $\text{powerinterval } f \ \sigma \ \text{ls} \wedge$
 $(\exists n \leq \text{nlength } (\text{pfilt } \sigma \ \text{ls}). \ g (\text{ntaken } n (\text{pfilt } \sigma \ \text{ls})) \wedge h (\text{ndropn } n (\text{pfilt } \sigma \ \text{ls}))))$
 using 11 16 19 20 24 25 26
 by (metis 5 nfinite-ndropn nfinite-nlength-enat nfinite-nmap pfilt-nmap the-enat.simps)
 show ?thesis unfolding fprojection-d-def using chop-nfuse nfuse-ntaken-ndropn 27
 by (metis ndropn-nfirst nfinite-conv-nlength-enat nfinite-ntaken ntaken-nlast)
 qed

lemma *OPJ4semchainb*:

assumes $(\sigma \models (f \text{ fproj } g) ; (f \text{ oproj } h))$

shows $(\sigma \models f \text{ oproj } ((g \wedge \text{finite}); h))$

proof –

have 1: $(\exists n \leq \text{nlength } \sigma.$

$(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } (\text{ntaken } n \ \sigma) \wedge$
 $\text{nlast } ls = (\text{the-enat}(\text{nlength } (\text{ntaken } n \ \sigma))) \wedge$
 $\text{powerinterval } f \ (\text{ntaken } n \ \sigma) \ ls \wedge g \ (\text{pfilt } (\text{ntaken } n \ \sigma) \ ls)) \wedge$
 $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } (\text{ndropn } n \ \sigma) \wedge$
 $\text{powerinterval } f \ (\text{ndropn } n \ \sigma) \ ls \wedge h \ (\text{pfilt } (\text{ndropn } n \ \sigma) \ ls))$
using *assms unfolding chop-defs fprojection-d-def oprojection-d-def*
by *simp blast*
obtain *cp* **where** *2*: $\text{cp} \leq \text{nlength } \sigma \wedge$
 $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } (\text{ntaken } cp \ \sigma) \wedge$
 $\text{nlast } ls = (\text{the-enat}(\text{nlength } (\text{ntaken } cp \ \sigma))) \wedge$
 $\text{powerinterval } f \ (\text{ntaken } cp \ \sigma) \ ls \wedge g \ (\text{pfilt } (\text{ntaken } cp \ \sigma) \ ls)) \wedge$
 $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } (\text{ndropn } cp \ \sigma) \wedge$
 $\text{powerinterval } f \ (\text{ndropn } cp \ \sigma) \ ls \wedge h \ (\text{pfilt } (\text{ndropn } cp \ \sigma) \ ls))$
using *1* **by** *auto*
have *3*: $\text{cp} \leq \text{nlength } \sigma$
using *2* **by** *auto*
have *4*: $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } (\text{ntaken } cp \ \sigma) \wedge$
 $\text{nlast } ls = (\text{the-enat}(\text{nlength } (\text{ntaken } cp \ \sigma))) \wedge$
 $\text{powerinterval } f \ (\text{ntaken } cp \ \sigma) \ ls \wedge g \ (\text{pfilt } (\text{ntaken } cp \ \sigma) \ ls))$
using *2* **by** *auto*
obtain *ls1* **where** *5*: $\text{nidx } ls1 \wedge \text{nnth } ls1 \ 0 = 0 \wedge \text{nfinite } ls1 \wedge \text{nfinite } (\text{ntaken } cp \ \sigma) \wedge$
 $\text{nlast } ls1 = (\text{the-enat}(\text{nlength } (\text{ntaken } cp \ \sigma))) \wedge$
 $\text{powerinterval } f \ (\text{ntaken } cp \ \sigma) \ ls1 \wedge g \ (\text{pfilt } (\text{ntaken } cp \ \sigma) \ ls1)$
using *4* **by** *auto*
have *6*: $\text{nidx } ls1 \wedge \text{nnth } ls1 \ 0 = 0$
using *5* **by** *auto*
have *61*: $\text{nfinite } ls1 \wedge \text{nfinite } (\text{ntaken } cp \ \sigma)$
using *5* **by** *auto*
have *7*: $\text{nlast } ls1 = \text{the-enat}(\text{nlength } (\text{ntaken } cp \ \sigma))$
using *5* **by** *auto*
have *8*: $\text{powerinterval } f \ (\text{ntaken } cp \ \sigma) \ ls1$
using *5* **by** *auto*
have *9*: $g \ (\text{pfilt } (\text{ntaken } cp \ \sigma) \ ls1)$
using *5* **by** *auto*
have *10*: $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } (\text{ndropn } cp \ \sigma) \wedge$
 $\text{powerinterval } f \ (\text{ndropn } cp \ \sigma) \ ls \wedge h \ (\text{pfilt } (\text{ndropn } cp \ \sigma) \ ls))$
using *2* **by** *auto*
obtain *ls2* **where** *11*: $\text{nidx } ls2 \wedge \text{nnth } ls2 \ 0 = 0 \wedge \neg \text{nfinite } ls2 \wedge \neg \text{nfinite } (\text{ndropn } cp \ \sigma) \wedge$
 $\text{powerinterval } f \ (\text{ndropn } cp \ \sigma) \ ls2 \wedge h \ (\text{pfilt } (\text{ndropn } cp \ \sigma) \ ls2)$
using *10* **by** *auto*
have *12*: $\text{nidx } ls2 \wedge \text{nnth } ls2 \ 0 = 0$
using *11* **by** *auto*
have *14*: $\text{powerinterval } f \ (\text{ndropn } cp \ \sigma) \ ls2$
using *11* **by** *auto*
have *15*: $h \ (\text{pfilt } (\text{ndropn } cp \ \sigma) \ ls2)$
using *11* **by** *auto*
have *150*: $\text{nlast } ls1 = cp$
by *(simp add: 2 7)*
have *151*: $\text{nfirst } (\text{nmap } (\lambda x. x + cp) \ ls2) = cp$
by *(metis 11 NNil-eq-nmap-conv nlast-NNil ntake-eq-NNil-iff ntake-nmap ntaken-0 ntaken-nlast*

$\text{plus-nat.add-0})$
have 152: $\text{nnth (nfuse ls1 (nmap (\lambda x. x+ cp) ls2)) 0 = 0}$
by (metis 150 151 5 nfuse-nnth zero-enat-def zero-le)
have 153: $\text{nidx ls1} \wedge \text{nnth ls1 0 = 0} \wedge \text{nfinite ls1} \wedge \text{nlast ls1 = cp}$
by (simp add: 150 5)
have 154: $\text{nidx ls2} \wedge \text{nnth ls2 0 = 0} \wedge \neg \text{nfinite ls2} \wedge \neg \text{nfinite } \sigma$
using 11 nfinite-ndropn-a **by** blast
have 16: $\text{nidx (nfuse ls1 (nmap (\lambda x. x+ cp) ls2))} \wedge \text{nnth (nfuse ls1 (nmap (\lambda x. x+ cp) ls2)) 0 = 0}$
using nidx-nfuse-nidx-infinite[of ls1 ls2 cp σ] 150 154 5 **by** fastforce
have 17: $\text{nlast ls1 = nfirst (nmap (\lambda x. x+ cp) ls2)}$
by (simp add: 150 151)
have 18: $\text{nnth (nfuse ls1 (nmap (\lambda x. x+ cp) ls2)) (the-enat(nlength ls1)) = cp}$
by (metis 150 17 5 ntaken-nfuse ntaken-nlast)
have 20: $\text{powerinterval f } \sigma \text{ (nfuse ls1 (nmap (\lambda x. x+ cp) ls2))}$
using powerinterval-nfuse-alt[of ls1 (ls2) cp σ f]
using 14 154 3 5 **by** fastforce
have 21: $\sigma = \text{nfuse (ntaken cp } \sigma) \text{ (ndropn cp } \sigma)$
by (simp add: 2 nfuse-ntaken-ndropn)
have 22: $\text{nnth ((nfuse ls1 (nmap (\lambda x. x+ cp) ls2))) (the-enat(nlength ls1)) = cp}$
using 18 **by** blast
have 23: $(\text{ntaken (the-enat(nlength ls1)) (pfilt } \sigma \text{ (nfuse ls1 (nmap (\lambda x. x+ cp) ls2))})) =$
 $(\text{pfilt (ntaken cp } \sigma) \text{ ls1})$
by (metis 150 17 2 5 enat.distinct(2) enat-le-plus-same(1) enat-the-enat
nfinite-conv-nlength-enat nfuse-nlength ntaken-nfuse pfilt-ntaken pfilt-ntaken-nidx)
have 24: $g (\text{ntaken (the-enat(nlength ls1)) (pfilt } \sigma \text{ (nfuse ls1 (nmap (\lambda x. x+ cp) ls2))}))$
by (simp add: 23 9)
have 25: $(\text{ndropn (the-enat(nlength ls1)) (pfilt } \sigma \text{ (nfuse ls1 (nmap (\lambda x. x+ cp) ls2))})) =$
 $(\text{pfilt (ndropn cp } \sigma) \text{ ls2})$
using pfilt-ndropn[of (the-enat(nlength ls1)) (nfuse ls1 (nmap (\lambda x. x+ cp) ls2)) σ]
pfilt-ndropn-nidx[of cp σ ls2]
by (metis 12 17 2 23 5 enat-ord-code(4) enat-the-enat min-def ndropn-nfuse nle-le
ntaken-nlength order-less-imp-le pfilt-nlength)
have 26: $\text{nlength ls1} \leq \text{nlength (pfilt } \sigma \text{ (nfuse ls1 (nmap (\lambda x. x+ cp) ls2)))}$
by (simp add: nfuse-nlength pfilt-nlength)
have 27: $(\exists \text{ls. nidx ls} \wedge \text{nnth ls 0 = 0} \wedge \neg \text{nfinite ls} \wedge \neg \text{nfinite } \sigma \wedge$
 $\text{powerinterval f } \sigma \text{ ls} \wedge$
 $(\exists n \leq \text{nlength (pfilt } \sigma \text{ ls). } g (\text{ntaken n (pfilt } \sigma \text{ ls)}) \wedge h (\text{ndropn n (pfilt } \sigma \text{ ls)})))$
using 11 16 20 24 25 26
by (metis 154 5 nfinite-nlength-enat nfinite-nmap nfuse-nfinite the-enat.simps)
show ?thesis **unfolding** fprojection-d-def oprojection-d-def chop-nfuse nfuse-ntaken-ndropn
finite-defs **by** simp
(metis 27 ndropn-nfirst nfinite-ntaken nfuse-ntaken-ndropn ntaken-nlast)

qed

lemma PJ4sem:

$(\sigma \models f \text{ fproj } (g;h) = (f \text{ fproj } g) ; (f \text{ fproj } h))$

using PJ4semchaina PJ4semchainb unl-lift2 **by** blast

lemma OPJ4sem:

$(\sigma \models f \text{ oproj } ((g \wedge \text{finite});h) = (f \text{ fproj } g) ; (f \text{ oproj } h))$

using *OPJ4semchaina OPJ4semchainb unl-lift2* by *blast*

PJ5

lemma *PJ5sem*:

$(\sigma \models f \text{ fproj } \text{init}(g) \longrightarrow \text{init}(g))$

by (*simp add: fprojection-d-def init-defs*)

(*metis ndropn-0 ndropn-nfirst ntaken-0 pfilt-code(1) pfilt-ntaken zero-enat-def zero-le*)

lemma *OPJ5sem*:

$(\sigma \models f \text{ oproj } \text{init}(g) \longrightarrow \text{init}(g))$

by (*simp add: oprojection-d-def init-defs*)

(*metis nlast-NNil ntaken-0 ntaken-nlast pfilt-nlength pfilt-nnth zero-enat-def zero-le*)

PJ6

lemma *PJ6help1*:

assumes *nidx ls*

nnth ls 0 = 0

nfinite ls

nfinite σ

nlast ls = the-enat(nlength σ)

shows $(\forall i. 0 \leq i \wedge i < \text{nlength } ls \longrightarrow \text{nlength } (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) = (\text{nnth } ls (\text{Suc } i)) - (\text{nnth } ls i))$

proof

fix *i*

show $0 \leq i \wedge i < \text{nlength } ls \longrightarrow$

$\text{nlength } (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) = \text{nnth } ls (\text{Suc } i) - \text{nnth } ls i$

using *assms nsubn-nlength[of σ]* **unfolding** *nidx-expand*

by (*metis assms(1) dual-order.order-iff-strict eSuc-enat enat-minus-mono1 enat-ord-simps(1) idiff-enat-enat ileI1 min.orderE nfinite-conv-nlength-enat nidx-less-last-1 nnth-nlast the-enat.simps*)

qed

lemma *OPJ6help1*:

assumes *nidx ls*

nnth ls 0 = 0

$\neg \text{nfinite } ls$

$\neg \text{nfinite } \sigma$

shows $(\forall i. 0 \leq i \wedge i < \text{nlength } ls \longrightarrow \text{nlength } (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) = (\text{nnth } ls (\text{Suc } i)) - (\text{nnth } ls i))$

proof

fix *i*

show $0 \leq i \wedge i < \text{nlength } ls \longrightarrow$

$\text{nlength } (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) = \text{nnth } ls (\text{Suc } i) - \text{nnth } ls i$

using *assms* **unfolding** *nidx-expand nsubn-def1* **by** (*simp add: nfinite-conv-nlength-enat*)

qed

lemma *PJ6help2*:

assumes *nidx ls*

```

     $nnth\ ls\ 0 = 0$ 
   $nfinite\ ls$ 
   $nfinite\ \sigma$ 
   $nlast\ ls = the-enat(nlength\ \sigma) \wedge$ 
   $(\forall i < nlength\ ls. nnth\ ls\ (Suc\ i) - nnth\ ls\ i = Suc\ 0)$ 
shows  $(\forall i \leq nlength\ ls. nnth\ ls\ i = i)$ 
proof
  fix  $i$ 
  show  $i \leq nlength\ ls \longrightarrow nnth\ ls\ i = i$ 
proof
  (induct  $i$ )
  case  $0$ 
  then show ?case using assms by blast
  next
  case  $(Suc\ i)$ 
  then show ?case
  by (metis One-nat-def Suc-ile-eq assms(1) assms(5) diff-add nid $\bar{x}$ -expand order-less-imp-le plus-1-eq-Suc)
  qed
qed

```

lemma *OPJ6help2:*

```

assumes  $nid\bar{x}\ ls$ 
   $nnth\ ls\ 0 = 0$ 
   $\neg nfinite\ ls$ 
   $\neg nfinite\ \sigma$ 
   $(\forall i < nlength\ ls. nnth\ ls\ (Suc\ i) - nnth\ ls\ i = Suc\ 0)$ 
shows  $(\forall i \leq nlength\ ls. nnth\ ls\ i = i)$ 
proof
  fix  $i$ 
  show  $i \leq nlength\ ls \longrightarrow nnth\ ls\ i = i$ 
proof
  (induct  $i$ )
  case  $0$ 
  then show ?case using assms by blast
  next
  case  $(Suc\ i)$ 
  then show ?case
  by (metis One-nat-def Suc-ile-eq assms(1) assms(5) diff-add nid $\bar{x}$ -expand order-less-imp-le plus-1-eq-Suc)
  qed
qed

```

lemma *PJ6help3:*

```

assumes  $nid\bar{x}\ ls$ 
   $nnth\ ls\ 0 = 0$ 
   $nfinite\ ls$ 
   $nfinite\ \sigma$ 
   $nlast\ ls = the-enat(nlength\ \sigma)$ 

```

$(\forall i \leq \text{nlength } ls. \text{nnth } ls \ i = i)$
shows $(\forall i < \text{nlength } ls. \text{nnth } ls \ (\text{Suc } i) - \text{nnth } ls \ i = \text{Suc } 0)$
proof
fix i
show $i < \text{nlength } ls \longrightarrow \text{nnth } ls \ (\text{Suc } i) - \text{nnth } ls \ i = \text{Suc } 0$
by (*metis One-nat-def add-diff-cancel-right' assms(6) eSuc-enat ileI1 order-less-imp-le plus-1-eq-Suc*)
qed

lemma *OPJ6help3*:
assumes $\text{nidx } ls$
 $\text{nnth } ls \ 0 = 0$
 $\neg \text{nfinite } ls$
 $\neg \text{nfinite } \sigma$
 $(\forall i \leq \text{nlength } ls. \text{nnth } ls \ i = i)$
shows $(\forall i < \text{nlength } ls. \text{nnth } ls \ (\text{Suc } i) - \text{nnth } ls \ i = \text{Suc } 0)$
proof
fix i
show $i < \text{nlength } ls \longrightarrow \text{nnth } ls \ (\text{Suc } i) - \text{nnth } ls \ i = \text{Suc } 0$
by (*metis One-nat-def add-diff-cancel-right' assms(5) eSuc-enat ileI1 order-less-imp-le plus-1-eq-Suc*)
qed

lemma *PJ6help4*:
assumes $\text{nfinite } \sigma$
shows $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge$
 $ls = \text{ntaken } (\text{the-enat } (\text{nlength } \sigma)) \ (\text{niterates } \text{Suc } 0) \wedge$
 $\text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$
 $(\forall i \leq \text{nlength } ls. \text{nnth } ls \ i = i) \wedge$
 $(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 \ i = \text{nnth } \sigma \ (i)) \wedge \sigma 1 = \sigma))$

proof –
have 1: $\text{nidx } (\text{ntaken } (\text{the-enat}(\text{nlength } \sigma)) \ (\text{niterates } \text{Suc } 0))$
using *nidx-ntaken-niterates-Suc* **by** *presburger*
have 2: $\text{nnth } (\text{ntaken } (\text{the-enat } (\text{nlength } \sigma)) \ (\text{niterates } \text{Suc } 0)) \ 0 = 0$
by (*simp add: ntaken-nnth*)
have 3: $\text{nlast } (\text{ntaken } (\text{the-enat } (\text{nlength } \sigma)) \ (\text{niterates } \text{Suc } 0)) = \text{the-enat}(\text{nlength } \sigma)$
by (*simp add: ntaken-nlast*)
have 4: $(\forall i \leq \text{nlength } (\text{ntaken } (\text{the-enat } (\text{nlength } \sigma)) \ (\text{niterates } \text{Suc } 0)).$
 $\text{nnth } (\text{ntaken } (\text{the-enat } (\text{nlength } \sigma)) \ (\text{niterates } \text{Suc } 0)) \ i = i)$
by (*simp add: ntaken-nnth*)
have 5: $(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } (\text{ntaken } (\text{the-enat } (\text{nlength } \sigma)) \ (\text{niterates } \text{Suc } 0)) \wedge$
 $(\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 \ i = \text{nnth } \sigma \ i) \wedge \sigma 1 = \sigma)$
using *assms* **by** (*simp add: enat-the-enat nfinite-nlength-enat*)
show *?thesis*
using 1 2 3 4 5 *assms nfinite-ntaken* **by** *blast*
qed

lemma *OPJ6help4*:

assumes $\neg nfinite\ \sigma$

shows $(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$
 $ls = (niterates\ Suc\ 0) \wedge$
 $(\forall i \leq nlength\ ls. nnth\ ls\ i = i) \wedge$
 $(\exists \sigma 1. nlength\ \sigma 1 = nlength\ ls \wedge (\forall i \leq nlength\ \sigma 1. nnth\ \sigma 1\ i = nnth\ \sigma\ i) \wedge \sigma 1 = \sigma))$

proof –

have 1: $nidx\ ((niterates\ Suc\ 0))$

using *nidx-expand nnth-niterates-Suc* **by** *presburger*

have 2: $nnth\ ((niterates\ Suc\ 0))\ 0 = 0$

by *(simp)*

have 4: $(\forall i \leq nlength\ ((niterates\ Suc\ 0)). nnth\ ((niterates\ Suc\ 0))\ i = i)$

by *(simp)*

have 5: $(\exists \sigma 1. nlength\ \sigma 1 = nlength\ ((niterates\ Suc\ 0)) \wedge$
 $(\forall i \leq nlength\ \sigma 1. nnth\ \sigma 1\ i = nnth\ \sigma\ i) \wedge \sigma 1 = \sigma)$

using *assms* **by** *(simp add: nfinite-conv-nlength-enat)*

show *?thesis*

using 1 2 4 5 *assms* **by** *blast*

qed

lemma *PJ6help5*:

assumes $nfinite\ \sigma$

shows $(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$
 $nlast\ ls = the-enat(nlength\ \sigma) \wedge$
 $(\forall i < nlength\ ls. nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) = Suc\ 0) \wedge$
 $(\exists \sigma 1. nlength\ \sigma 1 = nlength\ ls \wedge (\forall i \leq nlength\ \sigma 1. nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$
 $= g\ \sigma$

proof –

have $(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$

$nlast\ ls = the-enat\ (nlength\ \sigma) \wedge$

$(\forall i < nlength\ ls. nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) = Suc\ 0) \wedge$

$(\exists \sigma 1. nlength\ \sigma 1 = nlength\ ls \wedge (\forall i \leq nlength\ \sigma 1. nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$

$=$

$(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$

$nlast\ ls = the-enat(nlength\ \sigma) \wedge$

$(\forall i < nlength\ ls. nnth\ ls\ (Suc\ i) - nnth\ ls\ i = Suc\ 0) \wedge$

$(\exists \sigma 1. nlength\ \sigma 1 = nlength\ ls \wedge (\forall i \leq nlength\ \sigma 1. nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$

using *PJ6help1[of - σ]* *assms* **by** *auto*

also have $\dots =$

$(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$

$nlast\ ls = the-enat(nlength\ \sigma) \wedge$

$(\forall i \leq nlength\ ls. nnth\ ls\ i = i) \wedge$

$(\exists \sigma 1. nlength\ \sigma 1 = nlength\ ls \wedge (\forall i \leq nlength\ \sigma 1. nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$

using *PJ6help2 PJ6help3* **by** *blast*

also have $\dots =$

$(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$

$nlast\ ls = the-enat(nlength\ \sigma) \wedge$
 $(\forall i \leq nlength\ ls. nnth\ ls\ i = i) \wedge$
 $(\exists \sigma 1. nlength\ \sigma 1 = nlength\ ls \wedge (\forall i \leq nlength\ \sigma 1. nnth\ \sigma 1\ i = nnth\ \sigma\ (i)) \wedge g\ \sigma 1))$

using *assms* **by** *auto* (*metis*, *metis*)

also have ... =

$(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$
 $nlast\ ls = the-enat(nlength\ \sigma) \wedge$
 $(\forall i \leq nlength\ ls. nnth\ ls\ i = i) \wedge$
 $(\exists \sigma 1. nlength\ \sigma 1 = nlength\ ls \wedge (\forall i \leq nlength\ \sigma 1. nnth\ \sigma 1\ i = nnth\ \sigma\ (i)) \wedge \sigma 1 = \sigma \wedge g\ \sigma))$

by *auto*

$(metis\ dual-order.refl\ enat.distinct(2)\ enat-the-enat\ nellist-eq-nnth-eq\ nfinite-nlength-enat$
 $nnth-nlast)$

also have ... =

$g\ \sigma$

using *PJ6help4* *assms* **by** *auto*

finally show $(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge nfinite\ ls \wedge nfinite\ \sigma \wedge$

$nlast\ ls = the-enat(nlength\ \sigma) \wedge$

$(\forall i < nlength\ ls. nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) = Suc\ 0) \wedge$

$(\exists \sigma 1. nlength\ \sigma 1 = nlength\ ls \wedge (\forall i \leq nlength\ \sigma 1. nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$

= $g\ \sigma$

.

qed

lemma *OPJ6help5*:

assumes $\neg nfinite\ \sigma$

shows $(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$

$(\forall i < nlength\ ls. nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) = Suc\ 0) \wedge$

$(\exists \sigma 1. nlength\ \sigma 1 = nlength\ ls \wedge (\forall i \leq nlength\ \sigma 1. nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$

= $g\ \sigma$

proof –

have $(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$

$(\forall i < nlength\ ls. nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) = Suc\ 0) \wedge$

$(\exists \sigma 1. nlength\ \sigma 1 = nlength\ ls \wedge (\forall i \leq nlength\ \sigma 1. nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$

=

$(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$

$(\forall i < nlength\ ls. nnth\ ls\ (Suc\ i) - nnth\ ls\ i = Suc\ 0) \wedge$

$(\exists \sigma 1. nlength\ \sigma 1 = nlength\ ls \wedge (\forall i \leq nlength\ \sigma 1. nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$

using *OPJ6help1*[*of* - σ] *assms* **by** *auto*

also have ... =

$(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$

$(\forall i \leq nlength\ ls. nnth\ ls\ i = i) \wedge$

$(\exists \sigma 1. nlength\ \sigma 1 = nlength\ ls \wedge (\forall i \leq nlength\ \sigma 1. nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ ls\ i)) \wedge g\ \sigma 1))$

using *OPJ6help2*[*of* - σ] *OPJ6help3*[*of* - σ] **by** *blast*

also have ... =

$(\exists ls. nidx\ ls \wedge nnth\ ls\ 0 = 0 \wedge \neg nfinite\ ls \wedge \neg nfinite\ \sigma \wedge$

$(\forall i \leq nlength\ ls. nnth\ ls\ i = i) \wedge$

$(\exists \sigma 1. nlength\ \sigma 1 = nlength\ ls \wedge (\forall i \leq nlength\ \sigma 1. nnth\ \sigma 1\ i = nnth\ \sigma\ (i)) \wedge g\ \sigma 1))$

using *assms* **by** *auto (metis, metis)*
also have ... =

$$(\exists ls. \text{nid}x \text{ } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$$

$$(\forall i \leq \text{nlength } ls. \text{nnth } ls \ i = i) \wedge$$

$$(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 \ i = \text{nnth } \sigma \ (i)) \wedge \sigma 1 = \sigma \wedge g \ \sigma))$$

by *auto (metis OPJ6help4 nellist-eq-nnth-eq)*
also have ... =

$$g \ \sigma$$

using *OPJ6help4 assms* **by** *auto*
finally show $(\exists ls. \text{nid}x \text{ } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$

$$(\forall i < \text{nlength } ls. \text{nlength } (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) = \text{Suc } 0) \wedge$$

$$(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 \ i = \text{nnth } \sigma \ (\text{nnth } ls \ i)) \wedge g \ \sigma 1))$$

$$= g \ \sigma$$

qed

lemma *PJ6sem*:

$(\sigma \models \text{skip } \text{fproj } g = (g \wedge \text{finite}))$

proof –

have 1: $(\sigma \models \text{skip } \text{fproj } g = (g \wedge \text{finite})) =$

$$((\exists l. \text{nid}x \text{ } l \wedge \text{nnth } l \ 0 = 0 \wedge \text{nfinite } l \wedge \text{nfinite } \sigma \wedge \text{nlast } l = \text{the-enat}(\text{nlength } \sigma) \wedge$$

$$(\forall i < \text{nlength } l. (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) \models \text{skip}) \wedge g \ (\text{pfilt } \sigma \ l)) =$$

$$(g \ \sigma \wedge \text{nfinite } \sigma))$$

by *(simp add: fprojection-d-def powerinterval-def finite-defs)*
have 2: $(\exists ls. \text{nid}x \text{ } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge \text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$

$$(\forall i < \text{nlength } ls. (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) \models \text{skip}) \wedge g \ (\text{pfilt } \sigma \ ls)) =$$

$$(\exists ls \ \sigma 1. \text{nid}x \text{ } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge \text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$$

$$(\forall i < \text{nlength } ls. (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) \models \text{skip}) \wedge$$

$$\text{nlength } \sigma 1 = \text{nlength } ls \wedge$$

$$(\forall i \leq \text{nlength } \sigma 1. (\text{nnth } \sigma 1 \ i) = (\text{nnth } \sigma \ (\text{nnth } ls \ i))) \wedge$$

$$g \ \sigma 1)$$

using *pfilt-expand[of - σ]*
by *auto (blast,metis)*
have 3: $(\exists ls \ \sigma 1. \text{nid}x \text{ } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge$

$$\text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$$

$$(\forall i < \text{nlength } ls. (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) \models \text{skip}) \wedge$$

$$\text{nlength } \sigma 1 = \text{nlength } ls \wedge$$

$$(\forall i \leq \text{nlength } \sigma 1. (\text{nnth } \sigma 1 \ i) = (\text{nnth } \sigma \ (\text{nnth } ls \ i))) \wedge$$

$$g \ \sigma 1) =$$

$$(\exists ls. \text{nid}x \text{ } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge \text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$$

$$(\forall i < \text{nlength } ls. \text{nlength } (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) = \text{Suc } 0) \wedge$$

$$(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 \ i = \text{nnth } \sigma \ (\text{nnth } ls \ i)) \wedge g \ \sigma 1))$$

by *(simp add: skip-defs)*
have 4: $(\exists ls. \text{nid}x \text{ } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge \text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$

$$(\forall i < \text{nlength } ls. \text{nlength } (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) = \text{Suc } 0) \wedge$$

$$(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 \ i = \text{nnth } \sigma \ (\text{nnth } ls \ i)) \wedge g \ \sigma 1)) =$$

$$((g \ \sigma) \wedge \text{nfinite } \sigma)$$

using *PJ6help5*[*of* σ *g*] **by** *auto*
from 1 2 3 4 **show** *?thesis*
by *simp*
qed

lemma *OPJ6sem*:

$(\sigma \models \text{skip } \text{oproj } g = (g \wedge \text{inf}))$

proof –

have 1: $(\sigma \models \text{skip } \text{oproj } g = (g \wedge \text{inf})) =$

$((\exists \text{ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \neg \text{nfinite } \text{ls} \wedge \neg \text{nfinite } \sigma \wedge$
 $(\forall i < \text{nlength } \text{ls. } (\text{nsubn } \sigma (\text{nnth } \text{ls } i) (\text{nnth } \text{ls } (\text{Suc } i))) \models \text{skip}) \wedge g (\text{pfilt } \sigma \text{ls})) =$
 $(g \sigma \wedge \neg \text{nfinite } \sigma))$

by (*simp add: oprojection-d-def powerinterval-def infinite-defs*)

have 2: $(\exists \text{ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \neg \text{nfinite } \text{ls} \wedge \neg \text{nfinite } \sigma \wedge$

$(\forall i < \text{nlength } \text{ls. } (\text{nsubn } \sigma (\text{nnth } \text{ls } i) (\text{nnth } \text{ls } (\text{Suc } i))) \models \text{skip}) \wedge g (\text{pfilt } \sigma \text{ls})) =$

$(\exists \text{ls } \sigma 1 . \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \neg \text{nfinite } \text{ls} \wedge \neg \text{nfinite } \sigma \wedge$

$(\forall i < \text{nlength } \text{ls. } (\text{nsubn } \sigma (\text{nnth } \text{ls } i) (\text{nnth } \text{ls } (\text{Suc } i))) \models \text{skip}) \wedge$

$\text{nlength } \sigma 1 = \text{nlength } \text{ls} \wedge$

$(\forall i \leq \text{nlength } \sigma 1. (\text{nnth } \sigma 1 i) = (\text{nnth } \sigma (\text{nnth } \text{ls } i))) \wedge$

$g \sigma 1)$

using *pfilt-expand*[*of* σ] **by** *auto* (*blast,metis*)

have 3: $(\exists \text{ls } \sigma 1 . \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \neg \text{nfinite } \text{ls} \wedge \neg \text{nfinite } \sigma \wedge$

$(\forall i < \text{nlength } \text{ls. } (\text{nsubn } \sigma (\text{nnth } \text{ls } i) (\text{nnth } \text{ls } (\text{Suc } i))) \models \text{skip}) \wedge$

$\text{nlength } \sigma 1 = \text{nlength } \text{ls} \wedge$

$(\forall i \leq \text{nlength } \sigma 1. (\text{nnth } \sigma 1 i) = (\text{nnth } \sigma (\text{nnth } \text{ls } i))) \wedge$

$g \sigma 1) =$

$(\exists \text{ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \neg \text{nfinite } \text{ls} \wedge \neg \text{nfinite } \sigma \wedge$

$(\forall i < \text{nlength } \text{ls. } \text{nlength } (\text{nsubn } \sigma (\text{nnth } \text{ls } i) (\text{nnth } \text{ls } (\text{Suc } i))) = \text{Suc } 0) \wedge$

$(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } \text{ls} \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 i = \text{nnth } \sigma (\text{nnth } \text{ls } i)) \wedge g \sigma 1))$

by (*simp add: skip-defs*)

have 4: $(\exists \text{ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \neg \text{nfinite } \text{ls} \wedge \neg \text{nfinite } \sigma \wedge$

$(\forall i < \text{nlength } \text{ls. } \text{nlength } (\text{nsubn } \sigma (\text{nnth } \text{ls } i) (\text{nnth } \text{ls } (\text{Suc } i))) = \text{Suc } 0) \wedge$

$(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } \text{ls} \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 i = \text{nnth } \sigma (\text{nnth } \text{ls } i)) \wedge g \sigma 1)) =$

$((g \sigma) \wedge \neg \text{nfinite } \sigma)$

using *OPJ6help5*[*of* σ *g*] **by** *auto*

from 1 2 3 4 **show** *?thesis*

by *simp*

qed

PJ7

lemma *PJemptyImp*:

assumes $\text{nlength } \sigma = 0$

shows $(\sigma \models (f \text{ fproj } g) = g)$

using *assms*

by (*auto simp add: fprojection-d-def powerinterval-def nsubn-def1*)

(*metis i0-less less-numeral-extra*(3) *ndropn-0 ndropn-nlast nfinite-conv-nlength-enat*

nidx-less-last-1 nnth-nlast pfilt-code(1) *the-enat.simps the-enat-0*,

metis PJ6help4 ndropn-0 ndropn-nlast nfinite-ntaken not-less-zero ntaken-all ntaken-nlast

pfilt-code(1) *zero-le*)

lemma *PJ7empty:*

assumes $nlength\ \sigma = 0$

shows $(\sigma \models f\ fproj\ (g\ fproj\ h) = (f\ fproj\ g)\ fproj\ h)$

proof –

have 1: $(\sigma \models f\ fproj\ (g\ fproj\ h) = (g\ fproj\ h))$

using *PJemptyImp assms by blast*

have 2: $(\sigma \models (g\ fproj\ h) = h)$

using *PJemptyImp assms by blast*

have 3: $(\sigma \models (f\ fproj\ g)\ fproj\ h = h)$

using *PJemptyImp assms by blast*

from 1 2 3 **show** *?thesis by simp*

qed

lemma *PJ7helpchain1a-help-1:*

assumes $nidx\ ls$

$nnth\ ls\ 0 = 0$

$nfinite\ ls$

$nfinite\ \sigma$

$nlast\ ls = the-enat(nlength\ \sigma)$

$(\forall\ i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$

$nlength\ \sigma > 0$

shows $nidx\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls))) \wedge nnth\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls)))\ 0 = 0$

proof –

have 0: $nlength\ \sigma > 0 \longrightarrow nlength\ ls > 0$

using *assms*

by (*metis enat.distinct(2) enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)

have 01: $nfirst\ ls = 0$

using *assms*

by (*metis ndropn-0 ndropn-nfirst*)

have 02: $nlastnfirst\ (lcppl\ f\ g\ \sigma\ ls)$

using *assms lcppl-nfusecat-nlastnfirst[of ls sigma f g]*

by (*metis 0*)

have 1: $nfirst\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)) = 0$

using *assms 0 01 02 lcppl-nfirst[of ls sigma f g]* **by** (*metis nfirst-nfusecat-nfirst*)

from 1 0 **show** *?thesis using assms lcppl-nfusecat-nidx[of ls sigma f g]*

by (*metis 01 ntaken-0 ntaken-nlast*)

qed

lemma *PJ7helpchain1a-help-1-alt:*

assumes $nidx\ ls$

$nnth\ ls\ 0 = 0$

$\neg nfinite\ ls$

$\neg nfinite\ \sigma$

$(\forall\ i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$

$nlength\ \sigma > 0$

shows $nidx\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls))) \wedge nnth\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls)))\ 0 = 0$

proof –

have 0: $nlength\ \sigma > 0 \longrightarrow nlength\ ls > 0$

using *assms*


```

  by (metis enat-the-enat nfinite-conv-nlength-enat )
have 01: nfirst ls = 0
  using assms
  by (metis ndropn-0 ndropn-nfirst)
have 02: nlastnfirst (lcppl f g σ ls)
  using assms lcppl-nfusecat-nlastnfirst-alt[of ls σ f g]
  by (metis )
have 1: nfirst (nfusecat (lcppl f g σ ls)) = 0
  using assms 0 01 02
  lcppl-nfirst-alt[of ls σ f g ]
  by (metis nfirst-nfusecat-nfirst )
from 1 0 show ?thesis using assms lcppl-nfusecat-nidx-alt[of ls σ f g]
  by (metis 01 ntaken-0 ntaken-nlast)
qed

```

lemma *PJ7helpchain1a-help-2:*

```

assumes nidx ls
  nnth ls 0 = 0
  nfinite ls
  nfinite σ
  nlast ls = the-enat(nlength σ)
  (∀ i<nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f fproj g)
  nlength σ > 0
shows powerinterval f σ (nfusecat ( (lcppl f g σ ls)))
proof –
have 1: (∀ i<nlength ls.
  powerinterval f (nsubn σ (nnth ls i) (nnth ls (Suc i)) )
    (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )))
  using assms cppl-fprojection by blast
have 2: ∀ i<nlength ls.
  ∀ ia<nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ).
  f (nsubn (nsubn σ (nnth ls i) (nnth ls (Suc i)) )
    (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ) ia)
    (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ) (Suc ia))
  )
  using 1 by (simp add: powerinterval-def)
have 3: ∀ i<nlength ls.
  ∀ ia<nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ).
  (nsubn (nsubn σ (nnth ls i) (nnth ls (Suc i)) )
    (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ) ia)
    (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ) (Suc ia))
  ) =
  (nsubn σ
    (nnth (nmap (λx. x+ (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ) ) ia)
    (nnth (nmap (λx. x+ (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ) ) (Suc ia))
  )
proof auto
  fix i
  fix ia
  assume a0: enat i < nlength ls

```

assume *a1*: $\text{enat } ia < \text{nlength } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))))$
show $\text{nsbn } (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))$
 $(\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \ i)$
 $(\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) (\text{Suc } ia)) =$
 $\text{nsbn } \sigma \ (\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \ i + \text{nnth } ls \ i)$
 $(\text{nnth } (\text{nmap } (\lambda x. x + \text{nnth } ls \ i) \ (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))))) (\text{Suc } ia))$
proof –
have *300*: $0 < \text{nlength } \sigma$
using *assms*(7) **by** *auto*
have *301*: $\text{nnth } ls \ i < \text{nnth } ls \ (\text{Suc } i)$
by (*metis* *a0* *assms*(1) *eSuc-enat ileI1 nidx-expand*)
have *302*: $(\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) \models (f \ \text{fproj} \ g)$
by (*simp* *add*: *a0* *assms*(6))
have *303*: $\text{the-enat } (\text{nlength } (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) = (\text{nnth } ls \ (\text{Suc } i)) - (\text{nnth } ls \ i)$
by (*simp* *add*: *PJ6help1* *a0* *assms*(1) *assms*(2) *assms*(3) *assms*(4) *assms*(5))
have *304*: $\text{nlast } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) = (\text{nnth } ls \ (\text{Suc } i)) - (\text{nnth } ls \ i)$
using *a0* *a1* *cppl-fprojection*[*of* *f* *g* $(\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))$]
using *302* *303* **by** *presburger*
have *305*: $(\text{Suc } ia) \leq \text{nlength } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))))$
by (*simp* *add*: *Suc-ile-eq* *a1*)
have *306*: $\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \ i \leq$
 $\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) (\text{Suc } ia)$
by (*metis* *302* *a1* *cppl-fprojection* *eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc*)
have *307*: $\text{nlast } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) =$
 $\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))))$
 $(\text{the-enat}(\text{nlength } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))))))$
using *302* *cppl-fprojection* *nnth-nlast* **by** *auto*
have *308*: $\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) (\text{Suc } ia) \leq$
 $\text{nlast } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))))$
by (*metis* *302* *305* *cppl-fprojection* *nidx-all-le-nlast*)
have *310*: $\text{enat } (\text{nnth } ls \ (\text{Suc } i)) \leq \text{nlength } \sigma$
using *a0* *assms*
by (*metis* *dual-order.refl* *eSuc-enat* *enat-ord-simps*(1) *enat-ord-simps*(3) *enat-the-enat*
ileI1 *nfinite-conv-nlength-enat* *nidx-less-eq* *nnth-nlast* *the-enat.simps*)
have *30*: $\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) (\text{Suc } ia) \leq \text{nnth } ls \ (\text{Suc } i) - \text{nnth } ls \ i$
using *assms* *cppl-bounds*[*of* $(\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)) \ \sigma \ f \ g \ \text{Suc } ia$]
PJ6help1[*of* *ls* σ] *a0* *a1* *cppl-fprojection*[*of* *f* *g* $(\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))$]
by (*metis* *303* *308*)
have *311*: $\text{nsbn } (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))$
 $(\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \ i)$
 $(\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) (\text{Suc } ia)) =$
 $\text{nsbn } \sigma \ (\text{nnth } ls \ i + \text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \ i)$
 $(\text{nnth } ls \ i + \text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) (\text{Suc } ia))$
using *nsbn-nsbn-1*[*of* $(\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \ i)$
 $(\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) (\text{Suc } ia)) \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)) \ \sigma$]
using *30* *301* *306* *310* *order-less-imp-le* **by** *blast*
have *312*: $(\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \ i + \text{nnth } ls \ i =$
 $(\text{nnth } ls \ i + \text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) \ i)$
by *simp*
have *313*: $(\text{nnth } ls \ i + \text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))) (\text{Suc } ia) =$

```

      (nnth (nmap (λx. x + nnth ls i) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))) (Suc ia))
    using 305 by force
  show ?thesis
    using 311 312 313 by presburger
qed
qed
have 4: ∀ i < nlength ls.
  ∀ ia < nlength (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))).
  f (nsubn σ (nnth (nmap (λx. x + (nnth ls i) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))) ia)
    (nnth (nmap (λx. x + (nnth ls i) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))) (Suc ia))
  )
  using 2 3 by auto
have 5: nlength(lcppl f g σ ls) = nlength ls - 1
  using assms lcppl-nlength lcppl-nlength-zero
  by (metis epred-0 epred-conv-minus i0-less)
have 6: nlength σ > 0 ⟶ nlength ls > 0
  using assms
  by (metis enat.distinct(2) enat-the-enat nfinite-nlength-enat nnth-nlast the-enat-0 zero-less-iff-neq-zero)
have 7: ∀ i < nlength ls.
  ∀ ia < nlength ((nnth (lcppl f g σ ls) i)).
  f (nsubn σ (nnth (nnth (lcppl f g σ ls) i) ia)
    (nnth (nnth (lcppl f g σ ls) i) (Suc ia))
  )
  using assms 4 lcppl-nnth by (metis nlength-nmap)
have 8: nlength σ > 0 ⟶
  (∀ i ≤ nlength(lcppl f g σ ls).
    ∀ ia < nlength ((nnth (lcppl f g σ ls) i)).
    f (nsubn σ (nnth (nnth (lcppl f g σ ls) i) ia)
      (nnth (nnth (lcppl f g σ ls) i) (Suc ia))
    ))
  using 5 6 7 assms by (metis co.enat.exhaust-sel i0-less iless-Suc-eq lcppl-nlength)
have 9: nlength σ > 0 ⟶
  powerinterval f σ (nfusecat ( (lcppl f g σ ls))) =
  (∀ i < nlength (nfusecat (lcppl f g σ ls)).
    f (nsubn σ (nnth (nfusecat (lcppl f g σ ls) i) (nnth (nfusecat (lcppl f g σ ls) (Suc i)))))
  by (simp add: powerinterval-def)
have 10: nlength σ > 0 ⟶ nlastnfirst (lcppl f g σ ls)
  using 6 assms lcppl-nfusecat-nlastnfirst
  by (metis nfinite-conv-nlength-enat)
have 11: nlength σ > 0 ⟶
  (∀ j ≤ nlength (lcppl f g σ ls). nlength(nnth (lcppl f g σ ls) j) > 0)

  using assms
  by (metis enat.distinct(2) enat-the-enat lcppl-nlength-all-gr-zero nfinite-conv-nlength-enat)
have 110: all-gr-zero (lcppl f g σ ls)
  using 11 all-gr-zero-nnth-a assms(7) by blast
have 111: all-nfinite (lcppl f g σ ls)
  using all-nfinite-nnth-a assms lcppl-nlength-all-nfinite by blast
have 12: nlength σ > 0 ⟶
  (∀ i ≤ nlength(lcppl f g σ ls).

```

```

    (∀ ia < nlength ((nnth (lcppl f g σ ls) i)).
      f (nsbn σ (nnth (nnth (lcppl f g σ ls) i) ia)
        (nnth (nnth (lcppl f g σ ls) i) (Suc ia))
        ))) =
    (∀ j < nlength (nfusecat (lcppl f g σ ls)).
      f (nsbn σ (nnth (nfusecat (lcppl f g σ ls)) j)
        (nnth (nfusecat (lcppl f g σ ls)) (Suc j))
        ))

using nfusecat-split-nsbn[of (lcppl f g σ ls) f σ] 6 10 11 110 111 assms
unfolding nridx-expand
by (simp add: Suc-ile-eq)
show ?thesis using 12 8 9 using assms
by meson
qed

lemma PJ7helpchain1a-help-2-alt:
assumes nidx ls
  nnth ls 0 = 0
  ¬nfinite ls
  ¬nfinite σ
  (∀ i < nlength ls. (nsbn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f fproj g)
shows powerinterval f σ (nfusecat ( (lcppl f g σ ls)))
proof -
have 1: (∀ i < nlength ls.
  powerinterval f (nsbn σ (nnth ls i) (nnth ls (Suc i)))
    (cppl f g (nsbn σ (nnth ls i) (nnth ls (Suc i)))))
using assms cppl-fprojection by blast
have 2: ∀ i < nlength ls.
  ∀ ia < nlength (cppl f g (nsbn σ (nnth ls i) (nnth ls (Suc i)))).
  f (nsbn (nsbn σ (nnth ls i) (nnth ls (Suc i)))
    (nnth (cppl f g (nsbn σ (nnth ls i) (nnth ls (Suc i))))) ia)
    (nnth (cppl f g (nsbn σ (nnth ls i) (nnth ls (Suc i))))) (Suc ia))
  )
using 1 by (simp add: powerinterval-def)
have 3: ∀ i < nlength ls.
  ∀ ia < nlength (cppl f g (nsbn σ (nnth ls i) (nnth ls (Suc i)))).
  (nsbn (nsbn σ (nnth ls i) (nnth ls (Suc i)))
    (nnth (cppl f g (nsbn σ (nnth ls i) (nnth ls (Suc i))))) ia)
    (nnth (cppl f g (nsbn σ (nnth ls i) (nnth ls (Suc i))))) (Suc ia))
  ) =
  (nsbn σ
    (nnth (nmap (λx. x + (nnth ls i)) (cppl f g (nsbn σ (nnth ls i) (nnth ls (Suc i))))) ia)
    (nnth (nmap (λx. x + (nnth ls i)) (cppl f g (nsbn σ (nnth ls i) (nnth ls (Suc i))))) (Suc ia))
  )
proof auto
fix i
fix ia
assume a0: enat i < nlength ls
assume a1: enat ia < nlength (cppl f g (nsbn σ (nnth ls i) (nnth ls (Suc i))))

```

show $nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))$
 $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ ia)$
 $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ (Suc\ ia)) =$
 $nsubn\ \sigma\ (nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ ia + nnth\ ls\ i)$
 $(nnth\ (nmap\ (\lambda x. x + nnth\ ls\ i)\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))))\ (Suc\ ia))$

proof –

have 300: $0 < nlength\ \sigma$
by (metis assms(4) gr-zeroI nlength-eq-enat-nfiniteD zero-enat-def)
have 301: $nnth\ ls\ i < nnth\ ls\ (Suc\ i)$
by (metis a0 assms(1) eSuc-enat ileI1 nidx-expand)
have 302: $(nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models (f\ fproj\ g)$
by (simp add: a0 assms(5))
have 303: $the-enat\ (nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) = (nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i)$
by (simp add: OPJ6help1 a0 assms(1) assms(2) assms(3) assms(4))
have 304: $nlast\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) = (nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i)$
using a0 a1 cppl-fprojection[of f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))]
using 302 303 **by** presburger
have 305: $(Suc\ ia) \leq nlength\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$
by (simp add: Suc-ile-eq a1)
have 306: $nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ ia \leq$
 $nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ (Suc\ ia)$
by (metis 302 a1 cppl-fprojection eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc)
have 307: $nlast\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) =$
 $nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$
 $(the-enat(nlength\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))))$
using 302 cppl-fprojection nnth-nlast **by** auto
have 308: $nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ (Suc\ ia) \leq$
 $nlast\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$
by (metis 302 305 cppl-fprojection nidx-all-le-nlast)
have 310: $enat\ (nnth\ ls\ (Suc\ i)) \leq nlength\ \sigma$
using a0 assms
by (metis enat-ord-simps(3) enat-the-enat nfinite-conv-nlength-enat)
have 30: $nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ (Suc\ ia) \leq nnth\ ls\ (Suc\ i) - nnth\ ls\ i$
using assms cppl-bounds[of (nnth ls i) (nnth ls (Suc i)) $\sigma\ f\ g\ Suc\ ia$]
PJ6help1[of ls σ] a0 a1 cppl-fprojection[of f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))]
by (metis 303 308)
have 311: $nsubn\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
 $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ ia)$
 $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ (Suc\ ia)) =$
 $nsubn\ \sigma\ (nnth\ ls\ i + nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ ia)$
 $(nnth\ ls\ i + nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ (Suc\ ia))$
using nsubn-nsubn-1[of (nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ia)
 $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ (Suc\ ia))\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ \sigma$]
using 30 301 306 310 order-less-imp-le **by** blast
have 312: $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ ia + nnth\ ls\ i =$
 $(nnth\ ls\ i + nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ ia)$
by simp
have 313: $(nnth\ ls\ i + nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ (Suc\ ia) =$
 $(nnth\ (nmap\ (\lambda x. x + nnth\ ls\ i)\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))))\ (Suc\ ia))$
using 305 **by** force

```

show ?thesis
using 311 312 313 by presburger
qed
qed
have 4:  $\forall i < \text{nlength } ls.$ 
   $\forall ia < \text{nlength } (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ))).$ 
   $f\ (nsubn\ \sigma\ (nnth\ (nmap\ (\lambda x. x + (nnth\ ls\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ )))\ ia)$ 
   $(nnth\ (nmap\ (\lambda x. x + (nnth\ ls\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ )))\ (Suc\ ia))$ 
  )
  using 2 3 by auto
have 5:  $\text{nlength}(lcppl\ f\ g\ \sigma\ ls) = \text{nlength } ls - 1$ 
  using assms lcppl-nlength lcppl-nlength-zero
  by (simp add: epred-conv-minus lcppl-nlength-alt)
have 6:  $\text{nlength } \sigma > 0 \longrightarrow \text{nlength } ls > 0$ 
  using assms
  by (simp add: nfinite-conv-nlength-enat)
have 7:  $\forall i < \text{nlength } ls.$ 
   $\forall ia < \text{nlength } ((nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$ 
   $f\ (nsubn\ \sigma\ (nnth\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)\ ia)$ 
   $(nnth\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)\ (Suc\ ia))$ 
  )
  using assms 4 lcppl-nnth
  by (metis nlength-nmap)
have 8:  $(\forall i \leq \text{nlength}(lcppl\ f\ g\ \sigma\ ls).$ 
   $\forall ia < \text{nlength } ((nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$ 
   $f\ (nsubn\ \sigma\ (nnth\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)\ ia)$ 
   $(nnth\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)\ (Suc\ ia))$ 
  ))
  using 5 6 7 assms enat-ile nfinite-conv-nlength-enat not-le-imp-less by blast
have 9:  $\text{powerinterval } f\ \sigma\ (\text{nfusecat } (lcppl\ f\ g\ \sigma\ ls)) =$ 
   $(\forall i < \text{nlength } (\text{nfusecat } (lcppl\ f\ g\ \sigma\ ls)).$ 
   $f\ (nsubn\ \sigma\ (nnth\ (\text{nfusecat } (lcppl\ f\ g\ \sigma\ ls))\ i)\ (nnth\ (\text{nfusecat } (lcppl\ f\ g\ \sigma\ ls))\ (Suc\ i))\ ))$ 
  by (simp add: powerinterval-def)
have 10:  $\text{nlastnfirst } (lcppl\ f\ g\ \sigma\ ls)$ 
  using 6 assms lcppl-nfusecat-nlastnfirst-alt
  by (metis)
have 11:  $(\forall j \leq \text{nlength } (lcppl\ f\ g\ \sigma\ ls). \text{nlength}(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j) > 0)$ 
  using assms lcppl-nlength-all-gr-zero-alt by blast
have 110:  $\text{all-gr-zero } (lcppl\ f\ g\ \sigma\ ls)$ 
  using 11 all-gr-zero-nnth-a assms by blast
have 111:  $\text{all-nfinite } (lcppl\ f\ g\ \sigma\ ls)$ 
  using all-nfinite-nnth-a assms lcppl-nlength-all-nfinite-alt by blast
have 12:  $(\forall i \leq \text{nlength}(lcppl\ f\ g\ \sigma\ ls).$ 
   $(\forall ia < \text{nlength } ((nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$ 
   $f\ (nsubn\ \sigma\ (nnth\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)\ ia)$ 
   $(nnth\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)\ (Suc\ ia))$ 
  )) =
   $(\forall j < \text{nlength } (\text{nfusecat } (lcppl\ f\ g\ \sigma\ ls)).$ 
   $f\ (nsubn\ \sigma\ (nnth\ (\text{nfusecat } (lcppl\ f\ g\ \sigma\ ls))\ j)$ 
   $(nnth\ (\text{nfusecat } (lcppl\ f\ g\ \sigma\ ls))\ (Suc\ j))$ 
  )

```

))

```

using nfusecat-split-nsubn[of (lcppl f g σ ls) f σ] 10 11 110 111 assms
unfolding nridx-expand
by (simp add: Suc-ile-eq)
show ?thesis using 12 8 9 using assms
by meson
qed

```

lemma *PJ7helpchain1a-help-3*:

```

assumes nidx ls
          nnth ls 0 = 0
          nfinite ls
          nfinite σ
          nlast ls = the-enat(nlength σ)
          (∀ i < nlength ls. (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f fproj g)
          h (pfilt σ ls)
          nlength σ > 0
shows nlast (nfusecat (lcppl f g σ ls)) = nlength σ
proof -
have 0: nlength σ > 0 ⟶ nlength ls > 0
  using assms
  by (metis PJ6help4 nnth-nlast the-enat-0 zero-less-iff-neq-zero)
have 00: nfinite (lcppl f g σ ls)
  using assms(1) assms(3) lcppl-nfinite by blast
have 01: nlastnfirst (lcppl f g σ ls)
  using 0 assms lcppl-nfusecat-nlastnfirst by blast
have 02: all-nfinite (lcppl f g σ ls)
  using all-nfinite-nnth-a assms lcppl-nlength-all-nfinite by blast
have 1: nlength σ > 0 ⟶
  nlast (nfusecat (lcppl f g σ ls)) = nlast(nlast ( (lcppl f g σ ls)))
  using assms 0 nlastfirst-nfusecat-nlast[of (lcppl f g σ ls) ] 00 01 02 by blast
have 2: nlength σ > 0 ⟶
  nlast ( (lcppl f g σ ls))
  = (nnth (lcppl f g σ ls) (the-enat(nlength ls) - 1))
  using assms
  by (metis 0 epred-enat lcppl-nfinite lcppl-nlength nfinite-nlength-enat nnth-nlast
    the-enat.simps)
have 3: nlength σ > 0 ⟶
  (nnth (lcppl f g σ ls) (the-enat(nlength ls) - 1)) =
  (nmap (λx. x + (nnth ls (the-enat(nlength ls) - 1)))
    (cppl f g (nsubn σ (nnth ls (the-enat(nlength ls) - 1))
      (nnth ls (Suc (the-enat(nlength ls) - 1))) )))
  using assms 0 lcppl-nnth[of ls (the-enat(nlength ls) - 1) f g σ]
  by (metis Suc-n-not-le-n add.commute diff-add enat.distinct(2) enat-ord-simps(1)
    enat-ord-simps(4) enat-the-enat ileI1 not-le-imp-less one-eSuc one-enat-def plus-1-eq-Suc)
have 4: nlength σ > 0 ⟶
  nlast( ( cppl f g (nsubn σ (nnth ls (the-enat(nlength ls) - 1))
    (nnth ls (Suc (the-enat(nlength ls) - 1))) ))) =

```

$$\text{the-enat}(\text{nlength } ((\text{nsubn } \sigma (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)) (\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))))))$$

using 0 *assms cppl-fprojection*[of *f g*

$$(\text{nsubn } \sigma (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)) (\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))))]$$

by (*metis Suc-n-not-le-n add.commute diff-add enat.distinct*(2) *enat-ord-simps*(1)
enat-the-enat ileI1 nfinite-conv-nlength-enat not-le-imp-less one-eSuc one-enat-def plus-1-eq-Suc)

have 5: $\text{nlength } \sigma > 0 \longrightarrow$

$$\text{the-enat}(\text{nlength } ((\text{nsubn } \sigma (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)) (\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))))) =$$

$$(\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))) - (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1))$$

using 0 *PJ6help1 assms*
by (*metis Suc-n-not-le-n add.commute diff-add enat-ord-simps*(1) *ileI1 le-add-same-cancel1*
nfinite-conv-nlength-enat not-le-imp-less one-eSuc one-enat-def plus-1-eq-Suc the-enat.simps)

have 60: *nfinite* (*cppl f g* (*nsubn* σ (*nnth* *ls* (*the-enat* (*nlength* *ls*) - 1))
 $(\text{nnth } \text{ls } (\text{Suc } (\text{the-enat } (\text{nlength } \text{ls}) - 1))))$)

by (*metis* 0 *Suc-n-not-le-n add.commute assms*(3) *assms*(6) *assms*(8) *cppl-fprojection diff-add*
enat.distinct(2) *enat-ord-simps*(1) *enat-the-enat ileI1 nfinite-conv-nlength-enat*
not-le-imp-less one-eSuc one-enat-def plus-1-eq-Suc)

have 6: $\text{nlength } \sigma > 0 \longrightarrow$

$$\text{nlast } ((\text{nmap } (\lambda x. x + (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1))) (\text{cppl } f g (\text{nsubn } \sigma (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)) (\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))))) =$$

$$(\lambda x. x + (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))$$

$$(\text{nlast } (\text{cppl } f g (\text{nsubn } \sigma (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)) (\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))))))$$

using *nlast-nmap*[of (*cppl f g* (*nsubn* σ (*nnth* *ls* (*the-enat* (*nlength* *ls*) - 1))
 $(\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))))$)
 $(\lambda x. x + (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))]$ 60 **by** *blast*

have 7: $\text{nlength } \sigma > 0 \longrightarrow$

$$(\lambda x. x + (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))$$

$$(\text{nlast } (\text{cppl } f g (\text{nsubn } \sigma (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)) (\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))))) =$$

$$(\lambda x. x + (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))$$

$$((\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))) - (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))$$

using 4 5 **by** *auto*

have 8: $\text{nlength } \sigma > 0 \longrightarrow$

$$(\lambda x. x + (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))$$

$$((\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))) - (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1))) =$$

$$((\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))) -$$

$$(\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1))) + (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1))$$

by (*simp add: shift-def*)

have 9: $\text{nlength } \sigma > 0 \longrightarrow$

$$((\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))) -$$

$$(\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1))) + (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1))$$

$$= (\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))$$

using *assms* 0

by (*metis Suc-diff-1 cancel-comm-monoid-add-class.diff-cancel diff-add diff-is-0-eq enat-ord-simps*(1)
enat-ord-simps(2) *nfinite-nlength-enat nidx-expand order-less-imp-le the-enat.simps zero-enat-def*)


```

have 10:  $\text{nlength } \sigma > 0 \longrightarrow (\text{nnth } ls \text{ (Suc (the-enat(nlength } ls) - 1))) = \text{nlength } \sigma$ 
using assms 0
by (metis One-nat-def add.commute diff-add eSuc-enat enat.distinct(2) enat-ord-simps(1) enat-the-enat
ileI1 nfinite-conv-nlength-enat nnth-nlast plus-1-eq-Suc zero-enat-def)
show ?thesis
using 1 10 2 3 6 7 8 9 assms by presburger
qed

```

lemma *PJ7helpchain1a-help-4:*

```

assumes nidx ls
           $\text{nnth } ls \ 0 = 0$ 
          nfinite ls
          nfinite  $\sigma$ 
           $\text{nlast } ls = \text{the-enat(nlength } \sigma)$ 
           $(\forall \ i < \text{nlength } ls. (\text{nsubn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)) \ ) \models f \ fproj \ g)$ 
           $h \ (\text{pfilt } \sigma \ ls)$ 
           $\text{nlength } \sigma > 0$ 
shows  $((\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls)))) \models g \ fproj \ h$ 
proof -
have 0:  $\text{nlength } \sigma > 0 \longrightarrow \text{nlength } ls > 0$ 
using assms
by (metis PJ6help4 nnth-nlast the-enat-0 zero-less-iff-neq-zero)
have 00: all-gr-zero ( $\text{lcppl } f \ g \ \sigma \ ls$ )
using all-gr-zero-nnth-a assms lcppl-nlength-all-gr-zero by blast
have 01: all-nfinite ( $\text{lcppl } f \ g \ \sigma \ ls$ )
using all-nfinite-nnth-a assms lcppl-nlength-all-nfinite by blast
have 02:  $\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ 0 = 0$ 
using 0 assms
by (metis addzero-def lcppl-lsum-nlength less-numeral-extra(3) nnth-0)
have 1:  $\text{nlength } \sigma > 0 \longrightarrow \text{nidx } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \wedge \text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ 0 = 0$ 
using assms lsum-addzero-nidx[of (lcppl f g  $\sigma$  ls)] lcppl-nlength-all-gr-zero[of ls  $\sigma$  f g]
using 00 01 02 by blast
have 2:  $\text{nlength } \sigma > 0 \longrightarrow$ 
           $\text{nlast } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) =$ 
           $\text{the-enat(nlength } (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))))$ 
using assms lcppl-lsum-nlength[of ls  $\sigma$  f g] lcppl-nfusecat-pfilt-nlength[of ls  $\sigma$  f g]
by fastforce
have 3:  $\text{nlength } \sigma > 0 \longrightarrow$ 
           $\text{powerinterval } g \ (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))) \ (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0))$ 
by (simp add: assms lcppl-nfusecat-pfilt-powerinterval)
have 4:  $\text{nlength } \sigma > 0 \longrightarrow$ 
           $h \ (\text{pfilt } (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))) \ (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)))$ 
by (simp add: assms pfilt-nmap-pfilt lcppl-pfilt-nfusecat-lsum)
show ?thesis using 1 2 3 4 assms fprojection-d-def
by (metis 0 00 01 lcppl-lsum-nlength lcppl-nfinite lcppl-nfusecat-nlastnfirst
nfinite-conv-nlength-enat nfinite-nmap nfusecat-nfinite pfilt-nmap)
qed

```

lemma *PJ7helpchain1a-help-4-alt*:

assumes *nidx ls*

nnth ls 0 = 0

\neg *nfinite ls*

\neg *nfinite σ*

$(\forall i < \text{length } ls. (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) \models f \text{ fproj } g)$

h (pfilt σ ls)

shows $((\text{pfilt } \sigma (\text{nfusecat } (\text{lcppl } f g \sigma ls))) \models g \text{ oproj } h)$

proof –

have 00: *all-gr-zero (lcppl f g σ ls)*

using *all-gr-zero-nnth-a assms lcppl-nlength-all-gr-zero-alt* **by** *blast*

have 01: *all-nfinite (lcppl f g σ ls)*

using *all-nfinite-nnth-a assms lcppl-nlength-all-nfinite-alt* **by** *blast*

have 02: *nnth (addzero (lsum (lcppl f g σ ls) 0)) 0 = 0*

using *assms*

by *(metis addzero-def lcppl-nfinite nfinite-lsum-conv-b nlength-eq-enat-nfiniteD nnth-0 zero-enat-def)*

have 1: *nidx (addzero (lsum (lcppl f g σ ls) 0)) \wedge nnth (addzero (lsum (lcppl f g σ ls) 0)) 0 = 0*

using *assms lsum-addzero-nidx 00 01 02* **by** *blast*

have 3: *powerinterval g (pfilt σ (nfusecat (lcppl f g σ ls))) (addzero (lsum (lcppl f g σ ls) 0))*

by *(simp add: assms lcppl-nfusecat-pfilt-powerinterval-alt)*

have 4: *h (pfilt (pfilt σ (nfusecat (lcppl f g σ ls))) (addzero (lsum (lcppl f g σ ls) 0)))*

by *(simp add: assms lcppl-pfilt-nfusecat-lsum-alt pfilt-nmap-pfilt)*

have 5: \neg *nfinite (addzero (lsum (lcppl f g σ ls) 0))*

using *assms*

by *(simp add: lcppl-lsum-nlength-alt nfinite-conv-nlength-enat)*

have 6: \neg *nfinite (nfusecat (lcppl f g σ ls))*

using *assms 00 01 lcppl-nfinite lcppl-nfusecat-nlastnfirst-alt nfusecat-nfinite* **by** *blast*

have 7: \neg *nfinite (pfilt σ (nfusecat (lcppl f g σ ls)))*

by *(simp add: 6 pfilt-nmap)*

show *?thesis* **using** 1 3 4 *assms* **unfolding** *oprojection-d-def*

using 5 7 **by** *blast*

qed

lemma *PJ7helpchain1a*:

assumes *nlength $\sigma > 0$*

$(\sigma \models (f \text{ fproj } g) \text{ fproj } h)$

shows $(\sigma \models f \text{ fproj } (g \text{ fproj } h))$

proof –

have 1: *nlength $\sigma > 0$*

using *assms* **by** *auto*

have 2: $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge \text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$

$\text{powerinterval } (\text{LIFT}(f \text{ fproj } g)) \sigma ls \wedge$

$h (\text{pfilt } \sigma ls))$

using *assms* **using** *cppl-fprojection[of LIFT(f fproj g) h σ]* **by** *blast*

obtain *ls* **where** 3: *nidx ls \wedge nnth ls 0 = 0 \wedge nfinite ls \wedge nfinite $\sigma \wedge$*

nlast ls = the-enat(nlength σ) \wedge

powerinterval (LIFT(f fproj g)) σ ls \wedge

h (pfilt σ ls)

using 2 **by** *blast*

have 4: *nidx ls \wedge nnth ls 0 = 0 \wedge nfinite ls \wedge nfinite $\sigma \wedge$ nlast ls = the-enat(nlength σ) \wedge*

$(\forall i < \text{nlength } ls. (\text{nsubn } \sigma (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) \models f \text{ fproj } g) \wedge$
 $h (\text{pfilt } \sigma \text{ } ls)$
using 3 **by** (*simp add: powerinterval-def*)
have 6: $\text{nlength } ls > 0$
using 1 4
by (*metis PJ6help4 nnth-nlast the-enat-0 zero-less-iff-neq-zero*)
have 7: $\text{nidx } (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \text{ } ls)) \wedge \text{nnth } (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \text{ } ls)) \ 0 = 0$
by (*metis (no-types, lifting) 1 4 PJ7helpchain1a-help-1*)
have 8: $\text{powerinterval } f \ \sigma (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \text{ } ls))$
using 4 6 *PJ7helpchain1a-help-2* **assms** **by** *auto*
have 9: $\text{nlast } (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \text{ } ls)) = \text{the-enat}(\text{nlength } \sigma)$
by (*metis 1 4 PJ7helpchain1a-help-3 the-enat.simps*)
have 10: $((\text{pfilt } \sigma (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \text{ } ls))) \models g \text{ fproj } h)$
using 1 4 *PJ7helpchain1a-help-4* **by** *blast*
show ?thesis
using 10 7 8 9 **by** (*metis 3 fprojection-d-def nfinite-nmap pfilt-nmap*)
qed

lemma *OPJ7helpchain1a:*

assumes $(\sigma \models (f \text{ fproj } g) \text{ oproj } h)$

shows $(\sigma \models f \text{ oproj } (g \text{ oproj } h))$

proof –

have 2: $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$
 $\text{powerinterval } (\text{LIFT}(f \text{ fproj } g)) \ \sigma \text{ } ls \wedge$
 $h (\text{pfilt } \sigma \text{ } ls))$
using *assms* **using** *cppl-oprojection[of LIFT(f fproj g) h sigma]* **by** *blast*
obtain *ls* **where** 3: $\text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$
 $\text{powerinterval } (\text{LIFT}(f \text{ fproj } g)) \ \sigma \text{ } ls \wedge$
 $h (\text{pfilt } \sigma \text{ } ls)$
using 2 **by** *blast*
have 4: $\text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$
 $(\forall i < \text{nlength } ls. (\text{nsubn } \sigma (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) \models f \text{ fproj } g) \wedge$
 $h (\text{pfilt } \sigma \text{ } ls)$
using 3 **by** (*simp add: powerinterval-def*)
have 6: $\text{nlength } ls > 0$
by (*metis 4 gr-zeroI nlength-eq-enat-nfiniteD zero-enat-def*)
have 7: $\text{nidx } (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \text{ } ls)) \wedge \text{nnth } (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \text{ } ls)) \ 0 = 0$
using 4 *PJ7helpchain1a-help-1-alt*
by (*metis 6 OPJ6help4*)
have 8: $\text{powerinterval } f \ \sigma (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \text{ } ls))$
using 4 *PJ7helpchain1a-help-2-alt* **by** *blast*
have 10: $((\text{pfilt } \sigma (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \text{ } ls))) \models g \text{ oproj } h)$
using 4 *PJ7helpchain1a-help-4-alt* **by** *blast*
show ?thesis
using 10 7 8 **by** (*metis 3 nfinite-conv-nlength-enat oprojection-d-def pfilt-nlength*)
qed

lemma *PJ7helpchain1b*:
assumes *nlength* $\sigma > 0$
 $(\sigma \models f \text{ fproj } (g \text{ fproj } h))$
shows $(\sigma \models (f \text{ fproj } g) \text{ fproj } h)$
proof –
have 1: *nlength* $\sigma > 0$
using *assms* **by** *auto*
have 2: $(\exists \text{ ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \text{nfinite } \text{ls} \wedge \text{nfinite } \sigma \wedge \text{nlast } \text{ls} = \text{the-enat}(\text{nlength } \sigma) \wedge$
 $\text{powerinterval } f \ \sigma \ \text{ls} \wedge$
 $((\text{pfilt } \sigma \ \text{ls}) \models g \text{ fproj } h))$
using *assms* *cppl-fprojection* **by** *blast*
obtain *ls* **where** 3: $\text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \text{nfinite } \text{ls} \wedge \text{nfinite } \sigma \wedge \text{nlast } \text{ls} = \text{the-enat}(\text{nlength } \sigma) \wedge$
 $\text{powerinterval } f \ \sigma \ \text{ls} \wedge$
 $((\text{pfilt } \sigma \ \text{ls}) \models g \text{ fproj } h)$
using 2 **by** *blast*
have 4: $(\exists \text{ lsa. } \text{nidx } \text{lsa} \wedge \text{nnth } \text{lsa } 0 = 0 \wedge \text{nfinite } \text{lsa} \wedge \text{nfinite } (\text{pfilt } \sigma \ \text{ls}) \wedge$
 $\text{nlast } \text{lsa} = \text{the-enat}(\text{nlength}(\text{pfilt } \sigma \ \text{ls})) \wedge$
 $\text{powerinterval } g \ (\text{pfilt } \sigma \ \text{ls}) \ \text{lsa} \wedge$
 $((\text{pfilt } (\text{pfilt } \sigma \ \text{ls}) \ \text{lsa}) \models h))$
using 3 **using** *cppl-fprojection* **by** *blast*
obtain *lsa* **where** 5: $\text{nidx } \text{lsa} \wedge \text{nnth } \text{lsa } 0 = 0 \wedge \text{nfinite } \text{lsa} \wedge \text{nfinite } (\text{pfilt } \sigma \ \text{ls}) \wedge$
 $\text{nlast } \text{lsa} = \text{the-enat}(\text{nlength}(\text{pfilt } \sigma \ \text{ls})) \wedge$
 $\text{powerinterval } g \ (\text{pfilt } \sigma \ \text{ls}) \ \text{lsa} \wedge$
 $((\text{pfilt } (\text{pfilt } \sigma \ \text{ls}) \ \text{lsa}) \models h)$
using 4 **by** *blast*
have 6: *nlength* *ls* > 0
using 1 3
by (*metis* *enat.distinct*(2) *enat-the-enat* *gr-zeroI* *nfinite-conv-nlength-enat* *nnth-nlast* *the-enat-0*)
have 7: *nlength*(*pfilt* σ *ls*) = *nlength* *ls*
by (*simp* *add*: *pfilt-nlength*)
have 8: (*pfilt* (*pfilt* σ *ls*) *lsa*) = (*pfilt* σ (*pfilt* *ls* *lsa*))
using *pfilt-nmap-pfilt* **by** *blast*
have 9: *nlast* (*pfilt* *ls* *lsa*) = *the-enat*(*nlength* σ)
by (*metis* 3 5 7 *nlast-nmap* *nnth-nlast* *pfilt-nmap*)
have 10: *nlength* *lsa* > 0
using 5 6 7
by (*metis* *enat.distinct*(2) *enat-the-enat* *gr-zeroI* *nfinite-conv-nlength-enat* *nnth-nlast* *the-enat-0*)
have 11: *nlength* (*pfilt* *ls* *lsa*) > 0
by (*metis* 10 *pfilt-nlength*)
have 111: *nnth* (*pfilt* *ls* *lsa*) 0 = 0
by (*metis* 3 5 *pfilt-nnth* *zero-enat-def* *zero-le*)
have 112: *nlength*(*pfilt* *ls* *lsa*) = *nlength* *lsa*
using *pfilt-expand* **by** *blast*
have 113: $(\forall i < \text{nlength } \text{lsa. } (\text{nnth } (\text{pfilt } \text{ls } \text{lsa}) \ i) = (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)))$
by (*simp* *add*: *pfilt-nmap*)
have 114: $(\forall i < \text{nlength } \text{lsa. } (\text{nnth } (\text{pfilt } \text{ls } \text{lsa}) \ (\text{Suc } i)) = (\text{nnth } \text{ls } (\text{nnth } \text{lsa } (\text{Suc } i))))$
by (*metis* 112 *eSuc-enat* *ileI1* *pfilt-nnth*)
have 1141: $\bigwedge j. j \leq \text{nlength } \text{lsa} \longrightarrow \text{nnth } \text{lsa } j \leq \text{nlength } \text{ls}$
by (*metis* 5 7 *enat-ord-simps*(1) *ndropn-eq-NNil* *ndropn-nlast* *nfinite-conv-nlength-enat* *nidx-less-eq* *the-enat.simps*)

have 115: $(\forall i < \text{nlength } \text{lsa}. (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)) < (\text{nnth } \text{ls } (\text{nnth } \text{lsa } (\text{Suc } i))))$)
by (metis 1141 3 5 eSuc-enat ileI1 less-imp-Suc-add nidx-expand nidx-less)
have 12: $\text{nidx } (\text{pfilt } \text{ls } \text{lsa}) \wedge \text{nnth } (\text{pfilt } \text{ls } \text{lsa}) 0 = 0$
by (simp add: 111 112 115 Suc-ile-eq nidx-expand pfilt-nnth)
have 2021: $(\forall i < \text{nlength } \text{lsa}. \text{nfinite } (\text{nsbn } \sigma (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)) (\text{nnth } \text{ls } (\text{nnth } \text{lsa } (\text{Suc } i))))$)
by (simp add: nsbn-def1)
have 2022: $\bigwedge i. i < \text{nlength } \text{lsa} \implies (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)) \leq \text{nlength } \sigma$
by (metis 3 5 7 enat.distinct(2) enat-ord-simps(1) enat-the-enat
nfinite-conv-nlength-enat nidx-all-le-nlast order-less-imp-le)
have 2023: $\bigwedge j. j \leq \text{nlength } \text{lsa} \longrightarrow \text{nnth } \text{lsa } j \leq \text{nlength } \text{ls}$
by (metis 5 7 enat-ord-simps(1) ndropn-eq-NNil ndropn-nlast nfinite-conv-nlength-enat
nidx-less-eq the-enat.simps)
have 203: $(\forall i < \text{nlength } \text{lsa}. \text{nlength } (\text{nsbn } \sigma (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)) (\text{nnth } \text{ls } (\text{nnth } \text{lsa } (\text{Suc } i)))) =$
 $(\text{nnth } \text{ls } (\text{nnth } \text{lsa } (\text{Suc } i))) - (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)))$)
by (metis 112 113 114 12 3 5 9 PJ6help1 le-add1 le-add-same-cancel1
nfinite-conv-nlength-enat)
have 2041: $(\forall i < \text{nlength } \text{lsa}. (\text{nnth } \text{lsa } (\text{Suc } i)) \leq \text{nlength } \text{ls})$
by (metis 2023 eSuc-enat ileI1)
have 204: $(\forall i < \text{nlength } \text{lsa}. \text{nlength } (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i))) (\text{nsbn } \text{ls } (\text{nnth } \text{lsa } i) (\text{nnth } \text{lsa } (\text{Suc } i)))) =$
 $(\text{nnth } \text{lsa } (\text{Suc } i)) - (\text{nnth } \text{lsa } i))$)
by (simp add: 3 5 PJ6help1 pfilt-nlength)
have 205: $(\forall i < \text{nlength } \text{lsa}. (\forall j \leq (\text{nnth } \text{lsa } (\text{Suc } i)) - (\text{nnth } \text{lsa } i). (\text{nnth } (\text{nsbn } \text{ls } (\text{nnth } \text{lsa } i) (\text{nnth } \text{lsa } (\text{Suc } i))) j) =$
 $(\text{nnth } \text{ls } ((\text{nnth } \text{lsa } i) + j)))$)
using 2041 5 by (simp add: nsbn-def1 ntaken-nnth)
have 2060: $(\forall i < \text{nlength } \text{lsa}. (\text{nnth } \text{lsa } i) \leq (\text{nnth } \text{lsa } (\text{Suc } i)))$
using 5 by (metis eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc)
have 206: $(\forall i < \text{nlength } \text{lsa}. (\forall j \leq (\text{nnth } \text{lsa } (\text{Suc } i)) - (\text{nnth } \text{lsa } i). (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)) \leq (\text{nnth } \text{ls } ((\text{nnth } \text{lsa } i) + j)))$)
using 2041 3 2060
by (metis le-add1 nidx-all-le-nlast nidx-less-eq nnth-beyond not-le-imp-less order-less-imp-le)
have 207: $(\forall i < \text{nlength } \text{lsa}. (\forall j \leq (\text{nnth } \text{lsa } (\text{Suc } i)) - (\text{nnth } \text{lsa } i). (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i))) (\text{nsbn } \text{ls } (\text{nnth } \text{lsa } i) (\text{nnth } \text{lsa } (\text{Suc } i)))) j) =$
 $(\text{nnth } \text{ls } ((\text{nnth } \text{lsa } i) + j)) - (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)))$)
using 204 205 by auto
have 208: $(\forall i < \text{nlength } \text{lsa}. (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i))) (\text{nsbn } \text{ls } (\text{nnth } \text{lsa } i) (\text{nnth } \text{lsa } (\text{Suc } i)))) 0) = 0)$)
by (simp add: 207)
have 209: $(\forall i < \text{nlength } \text{lsa}. (\forall j \leq (\text{nnth } \text{lsa } (\text{Suc } i)) - (\text{nnth } \text{lsa } i). (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)) \leq (\text{nnth } \text{ls } ((\text{nnth } \text{lsa } i) + j)))$)

$$\begin{aligned} & \text{nlast } (\text{nmap } (\lambda x. x - (\text{nnth } ls (\text{nnth } lsa \ i))) \\ & \quad (\text{nsbn } ls (\text{nnth } lsa \ i) (\text{nnth } lsa \ (\text{Suc } i)) \)) = \\ & (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i))) - (\text{nnth } ls (\text{nnth } lsa \ i)) \\ &) \end{aligned}$$

using 204 207 5 2060 **by** (simp add: nsbn-def1 ntaken-ndropn-nlast)

have 210: $(\forall \ i < \text{nlength } lsa.$

$$\begin{aligned} & \text{nidx } (\text{nmap } (\lambda x. x - (\text{nnth } ls (\text{nnth } lsa \ i))) (\text{nsbn } ls (\text{nnth } lsa \ i) (\text{nnth } lsa \ (\text{Suc } i)) \)) \wedge \\ & \text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } ls (\text{nnth } lsa \ i))) (\text{nsbn } ls (\text{nnth } lsa \ i) (\text{nnth } lsa \ (\text{Suc } i)) \)) \ 0 = 0 \\ &) \end{aligned}$$

using 3 5 7 nidx-expand nidx-shiftn nidx-nsbn **by** (simp add: 2041 Suc-ile-eq order-less-imp-le)

have 20: $\text{powerinterval } (\text{LIFT}(f \text{ fproj } g)) \ \sigma \ (\text{pfilt } ls \ lsa)$

proof –

have 201: $\text{powerinterval } (\text{LIFT}(f \text{ fproj } g)) \ \sigma \ (\text{pfilt } ls \ lsa) =$

$$(\forall \ i < \text{nlength } lsa. (\text{nsbn } \sigma \ (\text{nnth } ls (\text{nnth } lsa \ i)) (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i)) \)) \models f \text{ fproj } g)$$

unfolding powerinterval-def **by** (auto simp add: Suc-ile-eq pfilt-nlength pfilt-nnth)

have 202: $(\forall \ i < \text{nlength } lsa. (\text{nsbn } \sigma \ (\text{nnth } ls (\text{nnth } lsa \ i)) (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i)) \)) \models f \text{ fproj } g) =$

$$(\forall \ i < \text{nlength } lsa.$$

$$(\exists \ ls1. \text{nidx } ls1 \wedge \text{nnth } ls1 \ 0 = 0 \wedge \text{nfinite } ls1 \wedge$$

$$\text{nfinite } (\text{nsbn } \sigma \ (\text{nnth } ls (\text{nnth } lsa \ i)) (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i)) \)) \wedge$$

$$\text{nlast } ls1 = \text{the-enat}(\text{nlength } (\text{nsbn } \sigma \ (\text{nnth } ls (\text{nnth } lsa \ i)) (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i)) \))) \wedge$$

$$\text{powerinterval } f \ (\text{nsbn } \sigma \ (\text{nnth } ls (\text{nnth } lsa \ i)) (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i)) \)) \ ls1 \wedge$$

$$((\text{pfilt } (\text{nsbn } \sigma \ (\text{nnth } ls (\text{nnth } lsa \ i)) (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i)) \)) \ ls1) \models g)$$

$$))$$

by (simp add: fprojection-d-def)

have fg: $(\forall \ i < \text{nlength } lsa.$

$$(\exists \ ls1. \text{nidx } ls1 \wedge \text{nnth } ls1 \ 0 = 0 \wedge \text{nfinite } ls1 \wedge$$

$$\text{nfinite } (\text{nsbn } \sigma \ (\text{nnth } ls (\text{nnth } lsa \ i)) (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i)) \)) \wedge$$

$$\text{nlast } ls1 = \text{the-enat}(\text{nlength } (\text{nsbn } \sigma \ (\text{nnth } ls (\text{nnth } lsa \ i)) (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i)) \))) \wedge$$

$$\text{powerinterval } f \ (\text{nsbn } \sigma \ (\text{nnth } ls (\text{nnth } lsa \ i)) (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i)) \)) \ ls1 \wedge$$

$$((\text{pfilt } (\text{nsbn } \sigma \ (\text{nnth } ls (\text{nnth } lsa \ i)) (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i)) \)) \ ls1) \models g)$$

$$))$$

proof auto

fix i::nat

assume a: $\text{enat } i < \text{nlength } lsa$

show $\exists \ ls1. \text{nidx } ls1 \wedge$

$$\text{nnth } ls1 \ 0 = 0 \wedge$$

$$\text{nfinite } ls1 \wedge$$

$$\text{nfinite } (\text{nsbn } \sigma \ (\text{nnth } ls (\text{nnth } lsa \ i)) (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i)) \)) \wedge$$

$$\text{nlast } ls1 = \text{the-enat}(\text{nlength } (\text{nsbn } \sigma \ (\text{nnth } ls (\text{nnth } lsa \ i)) (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i)) \))) \wedge$$

$$\text{powerinterval } f \ (\text{nsbn } \sigma \ (\text{nnth } ls (\text{nnth } lsa \ i)) (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i)) \)) \ ls1 \wedge$$

$$g \ (\text{pfilt } (\text{nsbn } \sigma \ (\text{nnth } ls (\text{nnth } lsa \ i)) (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i)) \)) \ ls1)$$

proof –

have fg1: $\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } ls (\text{nnth } lsa \ i))) (\text{nsbn } ls (\text{nnth } lsa \ i) (\text{nnth } lsa \ (\text{Suc } i)) \)) \ 0 = 0$

using 210 a **by** blast

have fg2: $\text{nidx } (\text{nmap } (\lambda x. x - (\text{nnth } ls (\text{nnth } lsa \ i))) (\text{nsbn } ls (\text{nnth } lsa \ i) (\text{nnth } lsa \ (\text{Suc } i)) \))$

using 210 a **by** blast

have fg3: $\text{nfinite } (\text{nsbn } \sigma \ (\text{nnth } ls (\text{nnth } lsa \ i)) (\text{nnth } ls (\text{nnth } lsa \ (\text{Suc } i)) \))$

```

using 2021 a by blast
have fg4: nlast (nmap (λx. x - (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) )) =
  the-enat (nlength (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))
using 203 209 a the-enat.simps by presburger
have fg5: powerinterval f (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
  (nmap (λx. x - (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)) ))
proof (auto simp add: powerinterval-def)
fix ia::nat
assume aa: enat ia < nlength (nsubn ls (nnth lsa i) (nnth lsa (Suc i)))
show f (nsubn (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
  (nnth (nsubn ls (nnth lsa i) (nnth lsa (Suc i))) ia - nnth ls (nnth lsa i))
  (nnth (nmap (λx. x - nnth ls (nnth lsa i)) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)))))
(Suc ia)))
proof -
have f1: ia < (nnth lsa (Suc i)) - (nnth lsa i)
  by (metis 5 7 PJ6help1 a aa enat-ord-simps(2) le-add1 le-add-same-cancel1 nsubn-nlength)
have f2: nsubn (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
  (nnth ls (nnth lsa i + ia) - nnth ls (nnth lsa i))
  (nnth ls (nnth lsa i + Suc ia) - nnth ls (nnth lsa i)) =
  nsubn σ (nnth ls (nnth lsa i + ia)) (nnth ls (nnth lsa i + Suc ia))
proof -
have f3: (nnth ls (nnth lsa i + ia) - nnth ls (nnth lsa i)) ≤
  nnth ls (nnth lsa i + Suc ia) - nnth ls (nnth lsa i)
  by (metis 2041 205 2060 3 Suc-leI a aa diff-le-mono eSuc-enat f1 ileI1
    le-add2 nidx-less-eq nidx-nsubn order-less-imp-le plus-1-eq-Suc)
have f4: nnth ls (nnth lsa i) ≤ nnth ls (nnth lsa (Suc i))
  using 115 a dual-order.order-iff-strict by blast
have f5: enat (nnth ls (nnth lsa (Suc i))) ≤ nlength σ
  by (metis 2041 3 a enat-ord-code(4) enat-ord-simps(1) enat-the-enat nidx-all-le-nlast
    order-less-imp-le)
have f6: nnth ls (nnth lsa i + Suc ia) - nnth ls (nnth lsa i) ≤
  nnth ls (nnth lsa (Suc i)) - nnth ls (nnth lsa i)
  by (metis 2041 2060 3 Suc-leI a add-diff-cancel-left' diff-le-mono f1 le-add1
    le-diff-iff nidx-less-eq)
show ?thesis using nsubn-nsubn-1[of (nnth ls (nnth lsa i + ia) - nnth ls (nnth lsa i))
  (nnth ls (nnth lsa i + Suc ia) - nnth ls (nnth lsa i))
  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) σ ]
  by (metis 206 Suc-leI a f1 f3 f4 f5 f6 le-add-diff-inverse order-less-imp-le)
qed
have f7: (nnth (nmap (λx. x - nnth ls (nnth lsa i))
  (nsubn ls (nnth lsa i) (nnth lsa (Suc i))))) (Suc ia) =
  (nnth (nsubn ls (nnth lsa i) (nnth lsa (Suc i))) (Suc ia) - nnth ls (nnth lsa i))
  by (metis aa eSuc-enat ileI1 nnth-nmap)
have f8: f (nsubn σ (nnth ls ((nnth lsa i) + ia)) ((nnth ls ((nnth lsa i) + (Suc ia)))))
  using 2041 3 unfolding powerinterval-def
  by (metis a add commute add-Suc-right enat-ord-simps(2) f1 less-diff-conv
    order-less-le-trans )
show ?thesis using 205 a f1 f2 f7 f8 by fastforce
qed
qed

```

```

have fg6: g (pfilt (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
  (nmap ( $\lambda x. x -$  (nnth ls (nnth lsa i))) (nsbn ls (nnth lsa i) (nnth lsa (Suc i)) ))))
proof –
  have g1: (pfilt (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
    (nmap ( $\lambda x. x -$  (nnth ls (nnth lsa i))) (nsbn ls (nnth lsa i) (nnth lsa (Suc i)) )))) =
    nmap (nnth (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
      (nmap ( $\lambda x. x -$  nnth ls (nnth lsa i)) (nsbn ls (nnth lsa i) (nnth lsa (Suc i)))))
  using pfilt-nmap[of (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
    (nmap ( $\lambda x. x -$  (nnth ls (nnth lsa i))) (nsbn ls (nnth lsa i) (nnth lsa (Suc i)) )))]
  by blast
have g2: ... =
  nmap (((nnth (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))  $\circ$ 
    ( $\lambda x. x -$  (nnth ls (nnth lsa i))))
    (nsbn ls (nnth lsa i) (nnth lsa (Suc i))))
  using nellist.map-comp by blast
have g3: ... =
  (nsbn (nmap (((nnth (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))  $\circ$ 
    ( $\lambda x. x -$  (nnth ls (nnth lsa i)))) ls)
    (nnth lsa i) (nnth lsa (Suc i))))
  unfolding nsbn-def1 by (simp add: ndropn-nmap)
have g4:  $\bigwedge y. (((nnth (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))  $\circ$ 
  ( $\lambda x. x -$  (nnth ls (nnth lsa i)))) y =$ 
  nnth (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
  (y - nnth ls (nnth lsa i))
  by simp
have g5: (nsbn (nmap (((nnth (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))  $\circ$ 
  ( $\lambda x. x -$  (nnth ls (nnth lsa i)))) ls)
  (nnth lsa i) (nnth lsa (Suc i))) =
  (nsbn (nmap ( $\lambda y. nnth$  (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
  (y - nnth ls (nnth lsa i))) ls)
  (nnth lsa i) (nnth lsa (Suc i)))
  using g4 by presburger
have g6:  $\bigwedge j. (nnth lsa i \leq j \wedge j \leq nnth lsa (Suc i)) \longrightarrow$ 
  nnth (nmap ( $\lambda y. nnth$  (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
  (y - nnth ls (nnth lsa i))) ls) j =
  nnth (nmap (nnth  $\sigma$ ) ls) j
proof
  fix j::nat
  assume aaa: nnth lsa i  $\leq j \wedge j \leq nnth lsa (Suc i)$ 
  show nnth (nmap ( $\lambda y. nnth$  (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
  (y - nnth ls (nnth lsa i))) ls) j = nnth (nmap (nnth  $\sigma$ ) ls) j
  proof –
  have g8: nnth (nmap (nnth  $\sigma$ ) ls) j = nnth  $\sigma$  (nnth ls j)
  by (metis nfinite-ntaken nlast-nmap ntaken-nlast ntaken-nmap)
  have g9: nnth (nmap ( $\lambda y. nnth$  (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
  (y - nnth ls (nnth lsa i))) ls) j =
  ( $\lambda y. nnth$  (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
  (y - nnth ls (nnth lsa i))) (nnth ls j)
  by (metis (no-types, lifting) nfinite-ntaken nlast-nmap ntaken-nlast ntaken-nmap)
  have g10: ... =

```



```

      nnth (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
      (nnth ls j - nnth ls (nnth lsa i))
    by auto
  have g11: ... = nnth  $\sigma$  ((nnth ls (nnth lsa i)) + (nnth ls j - nnth ls (nnth lsa i)))
    using nsubn-nnth[of  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))
      (nnth ls j - nnth ls (nnth lsa i))]
    by (simp add: 2041 3 a aaa diff-le-mono nidx-less-eq)
  have g12: ... = nnth  $\sigma$  (nnth ls j)
    using aaa
    by (metis 3 le-add-diff-inverse nidx-all-le-nlast nidx-less-eq nnth-beyond
      not-le-imp-less order-less-imp-le)
  show ?thesis
    using g11 g12 g8 g9 by presburger
  qed
  qed
  have g13: (pfilt (nsbn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
    (nmap ( $\lambda x. x -$  (nnth ls (nnth lsa i)) (nsbn ls (nnth lsa i) (nnth lsa (Suc i))))) =
    (nsbn (pfilt  $\sigma$  ls) (nnth lsa i) (nnth lsa (Suc i)) )
    by (simp add: 2041 2060 a g2 g3 g5 g6 nsubn-eq pfilt-nmap)
  show ?thesis using 5 a by (simp add: g13 powerinterval-def)
  qed
  show ?thesis
    using 203 2041 2060 209 210 a fg3 fg5 fg6 nsubn-nfinite by fastforce
  qed
  qed
  show ?thesis
    using 201 202 fg by blast
  qed
  show ?thesis
    by (metis 12 20 3 5 8 9 fprojection-d-def nfinite-nmap pfilt-nmap)
  qed

```

lemma *OPJ7helpchain1b*:

assumes ($\sigma \models f \text{ oproj } (g \text{ oproj } h)$)

shows ($\sigma \models (f \text{ fproj } g) \text{ oproj } h$)

proof –

have 2: ($\exists ls. \text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$
 $\text{powerinterval } f \ \sigma \ ls \wedge$
 $((\text{pfilt } \sigma \ ls) \models g \text{ oproj } h)$)

using *assms cppl-oprojection* **by** *blast*

obtain *ls* **where** 3: ($\text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$
 $\text{powerinterval } f \ \sigma \ ls \wedge$
 $((\text{pfilt } \sigma \ ls) \models g \text{ oproj } h)$)

using 2 **by** *blast*

have 4: ($\exists lsa. \text{nidx } lsa \wedge \text{nnth } lsa \ 0 = 0 \wedge \neg \text{nfinite } lsa \wedge \neg \text{nfinite } (\text{pfilt } \sigma \ ls) \wedge$
 $\text{powerinterval } g \ (\text{pfilt } \sigma \ ls) \ lsa \wedge$
 $((\text{pfilt } (\text{pfilt } \sigma \ ls) \ lsa) \models h)$)

using 3 **using** *cppl-oprojection* **by** *blast*

obtain *lsa* **where** 5: ($\text{nidx } lsa \wedge \text{nnth } lsa \ 0 = 0 \wedge \neg \text{nfinite } lsa \wedge \neg \text{nfinite } (\text{pfilt } \sigma \ ls) \wedge$

$\text{powerinterval } g \text{ (pfilt } \sigma \text{ ls) lsa} \wedge$
 $((\text{pfilt (pfilt } \sigma \text{ ls) lsa) } \models h)$
using 4 by blast
have 7: $\text{nlength}(\text{pfilt } \sigma \text{ ls}) = \text{nlength ls}$
by (*simp add: pfilt-nlength*)
have 8: $(\text{pfilt (pfilt } \sigma \text{ ls) lsa}) = (\text{pfilt } \sigma \text{ (pfilt ls lsa)})$
using pfilt-nmap-pfilt by blast
have 11: $\text{nlength (pfilt ls lsa)} > 0$
by (*metis 5 gr-zeroI nlength-eq-enat-nfiniteD pfilt-nlength zero-enat-def*)
have 111: $\text{nnth (pfilt ls lsa) } 0 = 0$
by (*metis 3 5 pfilt-nnth zero-enat-def zero-le*)
have 112: $\text{nlength}(\text{pfilt ls lsa}) = \text{nlength lsa}$
using pfilt-expand by blast
have 113: $(\forall i < \text{nlength lsa}. (\text{nnth (pfilt ls lsa) } i) = (\text{nnth ls (nnth lsa } i)))$
by (*simp add: pfilt-nmap*)
have 114: $(\forall i < \text{nlength lsa}. (\text{nnth (pfilt ls lsa) (Suc } i)}) = (\text{nnth ls (nnth lsa (Suc } i))))$
by (*metis 112 eSuc-enat ileI1 pfilt-nnth*)
have 1141: $\bigwedge j. j \leq \text{nlength lsa} \longrightarrow \text{nnth lsa } j \leq \text{nlength ls}$
by (*meson 3 enat-ile linorder-le-cases nfinite-conv-nlength-enat*)
have 115: $(\forall i < \text{nlength lsa}. (\text{nnth ls (nnth lsa } i)) < (\text{nnth ls (nnth lsa (Suc } i))))$
by (*metis 1141 3 5 eSuc-enat ileI1 less-imp-Suc-add nidx-expand nidx-less*)
have 12: $\text{nidx (pfilt ls lsa)} \wedge \text{nnth (pfilt ls lsa) } 0 = 0$
by (*simp add: 111 112 115 Suc-ile-eq nidx-expand pfilt-nnth*)
have 2021: $(\forall i < \text{nlength lsa}. \text{nfinite (nsbn } \sigma \text{ (nnth ls (nnth lsa } i)) (nnth ls (nnth lsa (Suc } i))))$
by (*simp add: nsbn-def1*)
have 2022: $\bigwedge i. i < \text{nlength lsa} \implies (\text{nnth ls (nnth lsa } i)) \leq \text{nlength } \sigma$
by (*meson 3 enat-ile linorder-le-cases nfinite-conv-nlength-enat*)
have 203: $(\forall i < \text{nlength lsa}. \text{nlength (nsbn } \sigma \text{ (nnth ls (nnth lsa } i)) (nnth ls (nnth lsa (Suc } i))) =$
 $(\text{nnth ls (nnth lsa (Suc } i))} - (\text{nnth ls (nnth lsa } i)))$
by (*metis 112 113 114 12 3 5 OPJ6help1 le-add1 le-add-same-cancel1 nfinite-nmap*
pfilt-nmap)
have 2041: $(\forall i < \text{nlength lsa}. (\text{nnth lsa (Suc } i)) \leq \text{nlength ls})$
by (*metis 1141 eSuc-enat ileI1*)
have 204: $(\forall i < \text{nlength lsa}.$
 $\text{nlength (nmap } (\lambda x. x - (\text{nnth ls (nnth lsa } i))) \text{ (nsbn ls (nnth lsa } i) \text{ (nnth lsa (Suc } i))}) =$
 $(\text{nnth lsa (Suc } i)) - (\text{nnth lsa } i))$
by (*simp add: 3 5 OPJ6help1*)
have 205: $(\forall i < \text{nlength lsa}.$
 $(\forall j \leq (\text{nnth lsa (Suc } i)) - (\text{nnth lsa } i).$
 $(\text{nnth (nsbn ls (nnth lsa } i) \text{ (nnth lsa (Suc } i)) } j) =$
 $(\text{nnth ls ((nnth lsa } i) + j))$
 $)$
 $)$
using 2041 5 by (*simp add: nsbn-def1 ntaken-nnth*)
have 2060: $(\forall i < \text{nlength lsa}. (\text{nnth lsa } i) \leq (\text{nnth lsa (Suc } i)))$
using 5 by (*metis eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc*)
have 206: $(\forall i < \text{nlength lsa}.$
 $(\forall j \leq (\text{nnth lsa (Suc } i)) - (\text{nnth lsa } i).$
 $(\text{nnth ls (nnth lsa } i)) \leq (\text{nnth ls ((nnth lsa } i) + j))$
 $)$
 $)$

```

    )
  using 2041 3 2060 by (simp add: nfinite-conv-nlength-enat nidx-less-eq)
  have 207: (∀ i < nlength lsa.
    (∀ j ≤ (nnth lsa (Suc i)) - (nnth lsa i).
      (nnth (nmap (λx. x - (nnth ls (nnth lsa i)))
        (nsbn ls (nnth lsa i) (nnth lsa (Suc i))) j) =
        (nnth ls ((nnth lsa i) + j)) - (nnth ls (nnth lsa i))
      ))
    )
  using 204 205 by auto
  have 208: (∀ i < nlength lsa.
    (nnth (nmap (λx. x - (nnth ls (nnth lsa i)))
      (nsbn ls (nnth lsa i) (nnth lsa (Suc i))) 0) = 0)
  )
  by (simp add: 207)
  have 209: (∀ i < nlength lsa.
    nlast (nmap (λx. x - (nnth ls (nnth lsa i)))
      (nsbn ls (nnth lsa i) (nnth lsa (Suc i))) ) =
    (nnth ls (nnth lsa (Suc i))) - (nnth ls (nnth lsa i))
  )
  )
  using 204 207 5 2060 by (simp add: nsbn-def1 ntaken-ndropn-nlast)
  have 210: (∀ i < nlength lsa.
    nidx (nmap (λx. x - (nnth ls (nnth lsa i))) (nsbn ls (nnth lsa i) (nnth lsa (Suc i))) ) ∧
    nnth (nmap (λx. x - (nnth ls (nnth lsa i))) (nsbn ls (nnth lsa i) (nnth lsa (Suc i))) ) 0 = 0
  )
  )
  using 3 5 7 nidx-expand nidx-shiftm nidx-nsbn by (simp add: 2041 Suc-ile-eq order-less-imp-le)
  have 20: powerinterval (LIFT(f fproj g)) σ (pfilt ls lsa)
  proof -
    have 201: powerinterval (LIFT(f fproj g)) σ (pfilt ls lsa) =
      (∀ i < nlength lsa. (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ⊨ f fproj g)
    unfolding powerinterval-def by (auto simp add: Suc-ile-eq pfilt-nlength pfilt-nnth)
    have 202: (∀ i < nlength lsa. (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ⊨ f fproj
  g) =
      (∀ i < nlength lsa.
        (∃ ls1. nidx ls1 ∧ nnth ls1 0 = 0 ∧ nfinite ls1 ∧
          nfinite (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ∧
          nlast ls1 = the-enat(nlength (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ) ∧
          powerinterval f (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ls1 ∧
          ((pfilt (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ls1) ⊨ g)
        ))
      )
    by (simp add: fprojection-d-def)
  have fg: (∀ i < nlength lsa.
    (∃ ls1. nidx ls1 ∧ nnth ls1 0 = 0 ∧ nfinite ls1 ∧
      nfinite (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ∧
      nlast ls1 = the-enat(nlength (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ) ∧
      powerinterval f (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ls1 ∧
      ((pfilt (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ls1) ⊨ g)
    ))
  )
  proof auto
    fix i::nat
    assume a: enat i < nlength lsa
    show ∃ ls1. nidx ls1 ∧

```

```

    nnth ls1 0 = 0 ∧
    nfinite ls1 ∧
    nfinite (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))) ∧
    nlast ls1 = the-enat (nlength (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))) ∧
    powerinterval f (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))) ls1 ∧
    g (pfilt (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))) ls1)
  proof -
  have fg1: nnth (nmap (λx. x - (nnth ls (nnth lsa i))) (nsbn ls (nnth lsa i) (nnth lsa (Suc i)))) 0 = 0
    using 210 a by blast
  have fg2: nidx (nmap (λx. x - (nnth ls (nnth lsa i))) (nsbn ls (nnth lsa i) (nnth lsa (Suc i))))
    using 210 a by blast
  have fg3: nfinite (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
    using 2021 a by blast
  have fg4: nlast (nmap (λx. x - (nnth ls (nnth lsa i))) (nsbn ls (nnth lsa i) (nnth lsa (Suc i)))) =
    the-enat (nlength (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))
    using 203 209 a the-enat.simps by presburger
  have fg5: powerinterval f (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
    (nmap (λx. x - (nnth ls (nnth lsa i))) (nsbn ls (nnth lsa i) (nnth lsa (Suc i))))
  proof (auto simp add: powerinterval-def)
  fix ia::nat
  assume aa: enat ia < nlength (nsbn ls (nnth lsa i) (nnth lsa (Suc i)))
  show f (nsbn (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
    (nnth (nsbn ls (nnth lsa i) (nnth lsa (Suc i))) ia - nnth ls (nnth lsa i))
    (nnth (nmap (λx. x - nnth ls (nnth lsa i)) (nsbn ls (nnth lsa i) (nnth lsa (Suc i))))
      (Suc ia)))
  proof -
  have f1: ia < (nnth lsa (Suc i)) - (nnth lsa i)
    by (metis 3 5 OPJ6help1 a aa enat-ord-simps(2) le-add1 le-add-same-cancel1)
  have f2: nsbn (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
    (nnth ls (nnth lsa i + ia) - nnth ls (nnth lsa i))
    (nnth ls (nnth lsa i + Suc ia) - nnth ls (nnth lsa i)) =
    nsbn σ (nnth ls (nnth lsa i + ia) (nnth ls (nnth lsa i + Suc ia)))
  proof -
  have f3: (nnth ls (nnth lsa i + ia) - nnth ls (nnth lsa i)) ≤
    nnth ls (nnth lsa i + Suc ia) - nnth ls (nnth lsa i)
    by (metis 2041 205 2060 3 Suc-leI a aa diff-le-mono eSuc-enat f1 ileI1
      le-add2 nidx-less-eq nidx-nsbn order-less-imp-le plus-1-eq-Suc)
  have f4: nnth ls (nnth lsa i) ≤ nnth ls (nnth lsa (Suc i))
    using 115 a dual-order.order-iff-strict by blast
  have f5: enat (nnth ls (nnth lsa (Suc i))) ≤ nlength σ
    by (metis 2022 5 a eSuc-enat ileI1 nfinite-conv-nlength-enat order-le-imp-less-or-eq)
  have f6: nnth ls (nnth lsa i + Suc ia) - nnth ls (nnth lsa i) ≤
    nnth ls (nnth lsa (Suc i)) - nnth ls (nnth lsa i)
    by (metis 2041 2060 3 Suc-leI a add-diff-cancel-left' diff-le-mono f1 le-add1
      le-diff-iff nidx-less-eq)
  show ?thesis using nsbn-nsbn-1[of (nnth ls (nnth lsa i + ia) - nnth ls (nnth lsa i))
    (nnth ls (nnth lsa i + Suc ia) - nnth ls (nnth lsa i))
    (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) σ]
    by (metis 206 Suc-leI a f1 f3 f4 f5 f6 le-add-diff-inverse order-less-imp-le)
  qed

```

```

have f7: (nnth (nmap (λx. x - nnth ls (nnth lsa i))
  (nsubn ls (nnth lsa i) (nnth lsa (Suc i)))) (Suc ia)) =
  (nnth (nsubn ls (nnth lsa i) (nnth lsa (Suc i))) (Suc ia) - nnth ls (nnth lsa i))
  by (metis aa eSuc-enat ileI1 nnth-nmap)
have f8: f (nsubn σ (nnth ls ((nnth lsa i) + ia)) ((nnth ls ((nnth lsa i) + (Suc ia)))) )
  using 2041 3 unfolding powerinterval-def
  by (metis a add.commute add-Suc-right enat-ord-simps(2) f1 less-diff-conv
    order-less-le-trans )
show ?thesis using 205 a f1 f2 f7 f8 by fastforce
qed
qed
have fg6: g (pfilt (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
  (nmap (λx. x - (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i))) ))
proof -
  have g1: (pfilt (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
    (nmap (λx. x - (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i))) )) =
    nmap (nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
      (nmap (λx. x - nnth ls (nnth lsa i) (nsubn ls (nnth lsa i) (nnth lsa (Suc i))))
    using pfilt-nmap[of (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
      (nmap (λx. x - (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i))) )]]
    by blast
  have g2: ... =
    nmap (((nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))) ∘
      (λx. x - (nnth ls (nnth lsa i))))
      (nsubn ls (nnth lsa i) (nnth lsa (Suc i)))
    using nellist.map-comp by blast
  have g3: ... =
    (nsubn (nmap (((nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))) ∘
      (λx. x - (nnth ls (nnth lsa i)))) ls)
      (nnth lsa i) (nnth lsa (Suc i)))
    unfolding nsubn-def1 by (simp add: ndropn-nmap)
  have g4: ∧y. (((nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))) ∘
    (λx. x - (nnth ls (nnth lsa i)))) y =
    nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
      (y - nnth ls (nnth lsa i))
    by simp
  have g5: (nsubn (nmap (((nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))) ∘
    (λx. x - (nnth ls (nnth lsa i)))) ls)
    (nnth lsa i) (nnth lsa (Suc i)) =
    (nsubn (nmap (λy. nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
      (y - nnth ls (nnth lsa i))) ls)
      (nnth lsa i) (nnth lsa (Suc i)))
    using g4 by presburger
  have g6: ∧j. (nnth lsa i) ≤ j ∧ j ≤ (nnth lsa (Suc i)) →
    nnth (nmap (λy. nnth (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
      (y - nnth ls (nnth lsa i))) ls) j =
    nnth (nmap (nnth σ) ls) j
proof
  fix j::nat
  assume aaa: nnth lsa i ≤ j ∧ j ≤ nnth lsa (Suc i)

```

```

show  $nnth\ (nmap\ (\lambda y. nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))\ (y - nnth\ ls\ (nnth\ lsa\ i)))\ ls)\ j = nnth\ (nmap\ (nnth\ \sigma)\ ls)\ j$ 
proof –
  have  $g8: nnth\ (nmap\ (nnth\ \sigma)\ ls)\ j = nnth\ \sigma\ (nnth\ ls\ j)$ 
    by (metis nfinite-ntaken nlast-nmap ntaken-nlast ntaken-nmap)
  have  $g9: nnth\ (nmap\ (\lambda y. nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))\ (y - nnth\ ls\ (nnth\ lsa\ i)))\ ls)\ j =$ 
     $(\lambda y. nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))\ (y - nnth\ ls\ (nnth\ lsa\ i)))\ (nnth\ ls\ j)$ 
    by (metis (no-types, lifting) nfinite-ntaken nlast-nmap ntaken-nlast ntaken-nmap)
  have  $g10: \dots =$ 
     $nnth\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))\ (nnth\ ls\ j - nnth\ ls\ (nnth\ lsa\ i))$ 
    by auto
  have  $g11: \dots = nnth\ \sigma\ ((nnth\ ls\ (nnth\ lsa\ i)) + (nnth\ ls\ j - nnth\ ls\ (nnth\ lsa\ i)))$ 
    using nsubn-nnth[of  $\sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))\ (nnth\ ls\ j - nnth\ ls\ (nnth\ lsa\ i))$ ]
    by (simp add: 2041 3 a aaa diff-le-mono nidr-less-eq)
  have  $g12: \dots = nnth\ \sigma\ (nnth\ ls\ j)$ 
    using aaa
    by (metis 206 a add-le-imp-le-diff diff-add le-add-diff-inverse)
  show ?thesis
  using  $g11\ g12\ g8\ g9$  by presburger
qed
qed
have  $g13: (pfilt\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))\ (nmap\ (\lambda x. x - (nnth\ ls\ (nnth\ lsa\ i)))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i)))) =$ 
   $(nsubn\ (pfilt\ \sigma\ ls)\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i)))$ 
    by (simp add: 2041 2060 a g2 g3 g5 g6 nsubn-eq pfilt-nmap)
  show ?thesis using 5 a by (simp add: g13 powerinterval-def)
qed
show ?thesis
using 203 2041 2060 209 210 a fg3 fg5 fg6 nsubn-nfinite by fastforce
qed
qed
show ?thesis
using 201 202 fg by blast
qed
show ?thesis
by (metis 12 20 3 5 8 nfinite-nmap oprojection-d-def pfilt-nmap)
qed

```

lemma *PJ7sem:*

$(\sigma \models f\ fproj\ (g\ fproj\ h)) = (f\ fproj\ g)\ fproj\ h)$

proof –

have $1: nlength\ \sigma > 0 \longrightarrow (\sigma \models f\ fproj\ (g\ fproj\ h)) = (f\ fproj\ g)\ fproj\ h)$

using *PJ7helpchain1a PJ7helpchain1b unl-lift2* **by** *blast*

have $2: nlength\ \sigma = 0 \longrightarrow (\sigma \models f\ fproj\ (g\ fproj\ h)) = (f\ fproj\ g)\ fproj\ h)$

using *PJ7empty* **by** *blast*

from $1\ 2$ **show** *?thesis* **by** *auto*

qed

lemma *OPJ7sem:*

$(\sigma \models f \text{ oproj } (g \text{ oproj } h) = (f \text{ fproj } g) \text{ oproj } h)$

using *OPJ7helpchain1a OPJ7helpchain1b unl-lift2 by blast*

PJ8

lemma *PJ8semhelp:*

assumes *nidx ls*

nnth ls 0 = 0

nfinite ls

nfinite σ

nlast ls = the-enat(nlength σ)

$(\forall n \text{ na. } na + n \leq nlength \sigma \longrightarrow f (nsubn \sigma n (n + na)) \longrightarrow g (nsubn \sigma n (n + na)))$

shows

$(\forall i < nlength \text{ ls. } f (nsubn \sigma (nnth \text{ ls } i) (nnth \text{ ls } (Suc \ i))) \longrightarrow g (nsubn \sigma (nnth \text{ ls } i) (nnth \text{ ls } (Suc \ i))))$

using *assms unfolding nidx-expand by auto*

(metis assms(1) diff-add eSuc-enat enat-ord-simps(1) ileI1 le-add-diff-inverse less-imp-le-nat nfinite-nlength-enat nidx-all-le-nlast the-enat.simps)

lemma *PJ8semhelp-alt:*

assumes *nidx ls*

nnth ls 0 = 0

$\neg nfinite \text{ ls}$

$\neg nfinite \sigma$

$(\forall n \text{ na. } na + n \leq nlength \sigma \longrightarrow f (nsubn \sigma n (n + na)) \longrightarrow g (nsubn \sigma n (n + na)))$

shows

$(\forall i < nlength \text{ ls. } f (nsubn \sigma (nnth \text{ ls } i) (nnth \text{ ls } (Suc \ i))) \longrightarrow g (nsubn \sigma (nnth \text{ ls } i) (nnth \text{ ls } (Suc \ i))))$

using *assms unfolding nidx-expand*

by *auto*

(metis enat-ord-simps(3) enat-the-enat le-add-diff-inverse nlength-eq-enat-nfiniteD order-less-imp-le)

lemma *PJ8sem:*

$(\sigma \models ba(f \longrightarrow g) \longrightarrow (f \text{ fproj } h) \longrightarrow (g \text{ fproj } h))$

using *PJ8semhelp*

by *(simp add: fprojection-d-def ba-defs powerinterval-def)*

(metis eSuc-enat enat-ord-simps(1) ileI1 le-add-diff-inverse linorder-le-cases nfinite-nlength-enat nidx-expand nidx-less-eq nnth-nlast order-less-imp-le the-enat.simps)

lemma *OPJ8sem:*

$(\sigma \models ba(f \longrightarrow g) \longrightarrow (f \text{ oproj } h) \longrightarrow (g \text{ oproj } h))$

using *PJ8semhelp-alt*

by *(simp add: oprojection-d-def ba-defs powerinterval-def)*

(metis eSuc-enat enat-ord-code(4) enat-the-enat ileI1 le-add-diff-inverse nidx-expand nlength-eq-enat-nfiniteD order-less-imp-le order-less-imp-le)

PJ9

lemma *PJ9sem*:

$(\sigma \models f \text{ ufproj } (g \longrightarrow h) \longrightarrow f \text{ fproj } g \longrightarrow f \text{ fproj } h)$
by (*simp add: ufpprojection-d-def fprojection-d-def*)
 (*metis less-numeral-extra(3)*)

lemma *OPJ9sem*:

$(\sigma \models f \text{ uproj } (g \longrightarrow h) \longrightarrow f \text{ oproj } g \longrightarrow f \text{ oproj } h)$
by (*simp add: uprojection-d-def oprojection-d-def*)
 (*metis less-numeral-extra(3)*)

2.7.4 Axioms

lemma *FBpGen*:

assumes $\vdash f$
shows $\vdash \text{fbp } f$
using *assms*
by (*simp add: fbp-d-def ufpprojection-d-def fprojection-d-def Valid-def*)

lemma *OBpGen*:

assumes $\vdash f$
shows $\vdash \text{obp } f$
using *assms*
by (*simp add: obp-d-def uprojection-d-def oprojection-d-def Valid-def*)

lemma *PJ00*:

$\vdash \neg(f \text{ fproj } \text{inf})$
using *PJ00sem Valid-def* **by** *blast*

lemma *OPJ00*:

$\vdash \neg(f \text{ oproj } \text{finite})$
using *OPJ00sem Valid-def* **by** *blast*

lemma *PJ01*:

$\vdash (f \text{ fproj } g) \longrightarrow \text{finite}$
using *PJ01sem Valid-def* **by** *blast*

lemma *OPJ01*:

$\vdash (f \text{ oproj } g) \longrightarrow \text{inf}$
using *OPJ01sem Valid-def* **by** *blast*

lemma *PJ02*:

$\vdash (f \text{ fproj } g) = ((f \wedge \text{finite}) \text{ fproj } g)$
using *PJ02sem Valid-def* **by** *blast*

lemma *OPJ02*:

$\vdash (f \text{ oproj } g) = ((f \wedge \text{finite}) \text{ oproj } g)$
using *OPJ02sem Valid-def* **by** *blast*

lemma *PJ03*:

$\vdash (f \text{ fproj } g) = (f \text{ fproj } (g \wedge \text{finite}))$
using *PJ03sem Valid-def* **by** *blast*

lemma *OPJ03*:
 $\vdash (f \text{ oproj } g) = (f \text{ oproj } (g \wedge \text{inf}))$
using *OPJ03sem Valid-def* **by** *blast*

lemma *PJ1*:
 $\vdash f \text{ fproj } (g \vee h) = (f \text{ fproj } g \vee f \text{ fproj } h)$
using *PJ1sem Valid-def* **by** *blast*

lemma *OPJ1*:
 $\vdash f \text{ oproj } (g \vee h) = (f \text{ oproj } g \vee f \text{ oproj } h)$
using *OPJ1sem Valid-def* **by** *blast*

lemma *PJ2*:
 $\vdash f \text{ fproj } \text{empty} = \text{empty}$
using *PJ2sem Valid-def* **by** *blast*

lemma *OPJ2*:
 $\vdash \neg(f \text{ oproj } \text{empty})$
using *OPJ2sem Valid-def* **by** *blast*

lemma *PJ3*:
 $\vdash f \text{ fproj } \text{skip} = (f \wedge \text{more} \wedge \text{finite})$
using *PJ3sem Valid-def* **by** *blast*

lemma *OPJ3*:
 $\vdash \neg(f \text{ oproj } \text{skip})$
using *OPJ3sem Valid-def* **by** *blast*

lemma *PJ4*:
 $\vdash f \text{ fproj } (g;h) = (f \text{ fproj } g) ; (f \text{ fproj } h)$
using *PJ4sem Valid-def* **by** *blast*

lemma *OPJ4*:
 $\vdash f \text{ oproj } ((g \wedge \text{finite});h) = (f \text{ fproj } g) ; (f \text{ oproj } h)$
using *OPJ4sem Valid-def* **by** *blast*

lemma *PJ5*:
 $\vdash f \text{ fproj } \text{init}(g) \longrightarrow \text{init}(g)$
using *PJ5sem Valid-def* **by** *blast*

lemma *OPJ5*:
 $\vdash f \text{ oproj } \text{init}(g) \longrightarrow \text{init}(g)$
using *OPJ5sem Valid-def* **by** *blast*

lemma *PJ6*:
 $\vdash \text{skip } f \text{ fproj } g = (g \wedge \text{finite})$

using *PJ6sem Valid-def* **by** *blast*

lemma *OPJ6*:

$\vdash \text{skip } \text{oproj } g = (g \wedge \text{inf})$

using *OPJ6sem Valid-def* **by** *blast*

lemma *PJ7*:

$\vdash f \text{ fproj } (g \text{ fproj } h) = (f \text{ fproj } g) \text{ fproj } h$

using *PJ7sem Valid-def* **by** *blast*

lemma *OPJ7*:

$\vdash f \text{ oproj } (g \text{ oproj } h) = (f \text{ fproj } g) \text{ oproj } h$

using *OPJ7sem Valid-def* **by** *blast*

lemma *PJ8*:

$\vdash \text{ba}(f \longrightarrow g) \longrightarrow (f \text{ fproj } h) \longrightarrow (g \text{ fproj } h)$

using *PJ8sem Valid-def* **by** *blast*

lemma *OPJ8*:

$\vdash \text{ba}(f \longrightarrow g) \longrightarrow (f \text{ oproj } h) \longrightarrow (g \text{ oproj } h)$

using *OPJ8sem Valid-def* **by** *blast*

lemma *PJ9*:

$\vdash f \text{ ufproj } (g \longrightarrow h) \longrightarrow f \text{ fproj } g \longrightarrow f \text{ fproj } h$

using *PJ9sem Valid-def* **by** *blast*

lemma *OPJ9*:

$\vdash f \text{ uoproj } (g \longrightarrow h) \longrightarrow f \text{ oproj } g \longrightarrow f \text{ oproj } h$

using *OPJ9sem Valid-def* **by** *blast*

2.7.5 Theorems

Projection

lemma *FPowerFProjLen*:

$\vdash f \text{ fproj } \text{len } n = \text{fpower } (f \wedge \text{more}) \text{ } n$

proof

(*induct n*)

case *0*

then show *?case*

by (*metis PJ2 fpower-d-def len-d-def wpow-0*)

next

case (*Suc n*)

then show *?case*

by (*metis AndMoreAndFiniteEqvAndFmore FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite*

FmoreEqvSkipChopFinite PJ3 PJ4sem fpower-d-def intI inteq-reflection len-d-def lift-and-com wpow-Suc)

qed

lemma *FProjLenExist*:

$\vdash f \text{ fproj } (\exists n. \text{ len } n) = (\exists n. f \text{ fproj } \text{ len } n)$
by (*simp add: Valid-def fprojection-d-def, blast*)

lemma *FPowerFProjLenExist*:

$\vdash (\exists n. f \text{ fproj } \text{ len } n) = (\exists n. \text{ fpower } (f \wedge \text{ more}) n)$

using *FPowerFProjLen* **by** (*simp add: Valid-def FPowerFProjLen, blast*)

lemma *RightFProjImpFProj*:

assumes $\vdash g1 \longrightarrow g2$

shows $\vdash f \text{ fproj } g1 \longrightarrow f \text{ fproj } g2$

using *assms*

by (*simp add: Valid-def fprojection-d-def, blast*)

lemma *RightOProjImpOProj*:

assumes $\vdash g1 \longrightarrow g2$

shows $\vdash f \text{ oproj } g1 \longrightarrow f \text{ oproj } g2$

using *assms*

by (*simp add: Valid-def oprojection-d-def, blast*)

lemma *LeftFProjImpFProj*:

assumes $\vdash f1 \longrightarrow f2$

shows $\vdash f1 \text{ fproj } g \longrightarrow f2 \text{ fproj } g$

using *assms*

by (*simp add: Valid-def fprojection-d-def powerinterval-def, blast*)

lemma *LeftOProjImpOProj*:

assumes $\vdash f1 \longrightarrow f2$

shows $\vdash f1 \text{ oproj } g \longrightarrow f2 \text{ oproj } g$

using *assms*

by (*simp add: Valid-def oprojection-d-def powerinterval-def, blast*)

lemma *RightFProjEqvFProj*:

assumes $\vdash g1 = g2$

shows $\vdash f \text{ fproj } g1 = f \text{ fproj } g2$

using *assms*

by (*metis (mono-tags, lifting) Valid-def inteq-reflection unl-lift2*)

lemma *RightOProjEqvOProj*:

assumes $\vdash g1 = g2$

shows $\vdash f \text{ oproj } g1 = f \text{ oproj } g2$

using *assms*

by (*metis (mono-tags, lifting) Valid-def inteq-reflection unl-lift2*)

lemma *LeftFProjEqvFProj*:

assumes $\vdash f1 = f2$

shows $\vdash f1 \text{ fproj } g = f2 \text{ fproj } g$

using *assms*

by (*metis (mono-tags, lifting) Valid-def inteq-reflection unl-lift2*)

```

lemma LeftOProjEqvOProj:
  assumes  $\vdash f1 = f2$ 
  shows  $\vdash f1 \text{ oproj } g = f2 \text{ oproj } g$ 
using assms
by (metis (mono-tags, lifting) Valid-def inteq-reflection unl-lift2)

lemma FProjTrueEqvSChopstar:
   $\vdash f \text{ fproj } \# \text{True} = (\text{schopstar } f)$ 
proof –
  have 1:  $\vdash \text{finite} = (\exists n. \text{len } n)$ 
    by (simp add: Finite-exist-len)
  have 2:  $\vdash f \text{ fproj } \# \text{True} = f \text{ fproj } (\exists n. \text{len } n)$ 
    using 1
    by (metis PJ03 Prop03 Prop10 int-simps(29) inteq-reflection lift-and-com)
  have 3:  $\vdash f \text{ fproj } (\exists n. \text{len } n) = (\exists n. \text{fpower } (f \wedge \text{more}) n)$ 
    using FPowerFProjLenExist FProjLenExist by fastforce
  have 4:  $\vdash (\exists n. \text{fpower } (f \wedge \text{more}) n) = (\text{schopstar } f)$ 
    by (metis FPowerstardef int-eq chopstar-d-def)
  show ?thesis using 2 3 4 by fastforce
qed

lemma OProjTrueEqvAOmega:
   $\vdash f \text{ oproj } \# \text{True} = (\text{aomega } f)$ 
using infinite-nidx-imp-infinite-interval
by (auto simp add: Valid-def oprojection-d-def aomega-d-def powerinterval-def)
(meson enat-ile linorder-le-cases nfinite-conv-nlength-enat not-le-imp-less, blast)

lemma OProjTrueEqvOmega:
   $\vdash f \text{ oproj } \# \text{True} = (\text{omega } f)$ 
by (metis OProjTrueEqvAOmega OmegaEqvAOmega int-eq)

lemma FProjSChopstarEqvSChopstarFProj:
   $\vdash f \text{ fproj } (\text{schopstar } g) = (\text{schopstar } (f \text{ fproj } g))$ 
proof –
  have 1:  $\vdash f \text{ fproj } (\text{schopstar } g) = f \text{ fproj } (g \text{ fproj } \# \text{True})$ 
    by (metis FProjTrueEqvSChopstar RightFProjEqvFProj inteq-reflection)
  have 2:  $\vdash f \text{ fproj } (g \text{ fproj } \# \text{True}) = (f \text{ fproj } g) \text{ fproj } \# \text{True}$ 
    by (simp add: PJ7)
  have 3:  $\vdash (f \text{ fproj } g) \text{ fproj } \# \text{True} = (\text{schopstar } (f \text{ fproj } g))$ 
    by (simp add: FProjTrueEqvSChopstar)
  show ?thesis using 1 2 3 by fastforce
qed

lemma OProjOmegaEqvOmegaFProj:
   $\vdash f \text{ oproj } (\text{omega } g) = (\text{omega } (f \text{ fproj } g))$ 
proof –
  have 1:  $\vdash f \text{ oproj } (\text{omega } g) = f \text{ oproj } (g \text{ oproj } \# \text{True})$ 
    by (metis OProjTrueEqvOmega RightOProjEqvOProj inteq-reflection)
  have 2:  $\vdash f \text{ oproj } (g \text{ oproj } \# \text{True}) = (f \text{ fproj } g) \text{ oproj } \# \text{True}$ 
    by (simp add: OPJ7)

```

have 3: $\vdash (f \text{ fproj } g) \text{ oproj } \# \text{True} = (\text{omega } (f \text{ fproj } g))$
by (*simp add: OProjTrueEqvOmega*)
show ?thesis **using** 1 2 3 **by** fastforce
qed

lemma *OProjAndImp*:

$\vdash f \text{ oproj } (g1 \wedge g2) \longrightarrow f \text{ oproj } g1 \wedge f \text{ oproj } g2$
by (*meson Prop12 RightOProjImpOProj int-iffD1 lift-and-com*)

lemma *FProjAndImp*:

$\vdash f \text{ fproj } (g1 \wedge g2) \longrightarrow f \text{ fproj } g1 \wedge f \text{ fproj } g2$
by (*meson Prop12 RightFProjImpFProj int-iffD1 lift-and-com*)

lemma *FProjOrDist*:

$\vdash \# \text{True } f \text{ fproj } (f \vee g) = (\# \text{True } f \text{ fproj } f \vee \# \text{True } f \text{ fproj } g)$
using PJ1 **by** blast

lemma *OProjOrDist*:

$\vdash \# \text{True } \text{oproj } (f \vee g) = (\# \text{True } \text{oproj } f \vee \# \text{True } \text{oproj } g)$
using OPJ1 **by** blast

lemma *StateImportFProj*:

$\vdash ((\text{init } w) \wedge f \text{ fproj } g) = f \text{ fproj } ((\text{init } w) \wedge g)$
by (*auto simp add: Valid-def init-defs fprojection-d-def pfilt-nnth nidx-expand*)
(metis nlast-NNil ntaken-0 ntaken-nlast pfilt-nlength pfilt-nnth zero-enat-def zero-le,
metis nlast-NNil ntaken-0 ntaken-nlast pfilt-nlength pfilt-nnth zero-enat-def zero-le)

lemma *StateImportOProj*:

$\vdash ((\text{init } w) \wedge f \text{ oproj } g) = f \text{ oproj } ((\text{init } w) \wedge g)$
by (*auto simp add: Valid-def init-defs oprojection-d-def pfilt-nnth nidx-expand*)
(metis ndropn-0 ndropn-nfirst pfilt-nlength pfilt-nnth zero-enat-def zero-le,
metis ndropn-0 ndropn-nfirst pfilt-nlength pfilt-nnth zero-enat-def zero-le)

lemma *FProjStateAndNextEqvStateAndMoreChopFProj*:

$\vdash f \text{ fproj } ((\text{init } w) \wedge \bigcirc g) = ((\text{init } w) \wedge (f \wedge \text{more} \wedge \text{finite}); (f \text{ fproj } g))$
proof –
have 2: $\vdash (f \wedge \text{more} \wedge \text{finite}); (f \text{ fproj } g) = f \text{ fproj } \bigcirc g$
using PJ3 PJ4 **unfolding** next-d-def **by** (*metis integ-reflection*)
have 3: $\vdash f \text{ fproj } ((\text{init } w)) \longrightarrow \text{init } w$
by (*simp add: PJ5*)
have 4: $\vdash (\text{init } w \wedge f \text{ fproj } \bigcirc g) = f \text{ fproj } ((\text{init } w) \wedge \bigcirc g)$
by (*simp add: StateImportFProj*)
have 5: $\vdash f \text{ fproj } ((\text{init } w) \wedge \bigcirc g) \longrightarrow ((\text{init } w) \wedge (f \wedge \text{more} \wedge \text{finite}); (f \text{ fproj } g))$
using 2 3 *FProjAndImp* **by** fastforce
from 5 4 **show** ?thesis **using** 2 **by** fastforce
qed

lemma *OProjStateAndNextEqvStateAndMoreChopFProj*:

$\vdash f \text{ oproj } ((init \ w) \wedge \bigcirc g) = ((init \ w) \wedge (f \wedge more \wedge finite);(f \text{ oproj } g))$
proof –
have 1: $\vdash (skip \wedge finite) = skip$
using *itl-defs(1) itl-defs(5) nlength-eq-enat-nfiniteD* **by** *fastforce*
have 2: $\vdash (f \wedge more \wedge finite);(f \text{ oproj } g) = f \text{ oproj } \bigcirc g$
using *PJ3 OPJ4 unfolding next-d-def* **by** *(metis 1 inteq-reflection)*
have 3: $\vdash f \text{ oproj } ((init \ w)) \longrightarrow init \ w$
by *(simp add: OPJ5)*
have 4: $\vdash (init \ w \wedge f \text{ oproj } \bigcirc g) = f \text{ oproj } ((init \ w) \wedge \bigcirc g)$
by *(simp add: StateImportOProj)*
have 5: $\vdash f \text{ oproj } ((init \ w) \wedge \bigcirc g) \longrightarrow ((init \ w) \wedge (f \wedge more \wedge finite);(f \text{ oproj } g))$
using 2 3 *OProjAndImp* **by** *fastforce*
from 5 4 **show** *?thesis* **using** 2 **by** *fastforce*
qed

lemma *FProjNext*:

$\vdash f \text{ fproj } \bigcirc g = (f \wedge more \wedge finite);(f \text{ fproj } g)$
by *(metis PJ3 PJ4 inteq-reflection next-d-def)*

lemma *OProjNext*:

$\vdash f \text{ oproj } \bigcirc g = (f \wedge more \wedge finite);(f \text{ oproj } g)$
by *(metis DiamondEmptyEqvFinite DiamondEqvEmptyOrNextDiamond FiniteChopEqvDiamond FiniteChopSkipEqvSkipChopFinite NowImpDiamond OPJ4 PJ3 Prop05 Prop10 SkipChopEqvNext inteq-reflection)*

lemma *FProjWnext*:

$\vdash f \text{ fproj } (wnext \ g) = (empty \vee (f \wedge more \wedge finite);(f \text{ fproj } g))$

proof –

have 1: $\vdash f \text{ fproj } (wnext \ g) = f \text{ fproj } (empty \vee \bigcirc g)$
by *(simp add: RightFProjEqvFProj WnextEqvEmptyOrNext)*
have 2: $\vdash f \text{ fproj } (empty \vee \bigcirc g) = (empty \vee f \text{ fproj } (\bigcirc g))$
using *PJ1 PJ2* **by** *fastforce*
have 3: $\vdash f \text{ fproj } (\bigcirc g) = (f \wedge more \wedge finite);(f \text{ fproj } g)$
by *(metis PJ3 PJ4 inteq-reflection next-d-def)*
show *?thesis*
using 1 2 3 **by** *fastforce*
qed

lemma *OProjWnext*:

$\vdash f \text{ oproj } (wnext \ g) = ((f \wedge more \wedge finite);(f \text{ oproj } g))$

proof –

have 1: $\vdash f \text{ oproj } (wnext \ g) = f \text{ oproj } (empty \vee \bigcirc g)$
by *(simp add: RightOProjEqvOProj WnextEqvEmptyOrNext)*
have 2: $\vdash f \text{ oproj } (empty \vee \bigcirc g) = (f \text{ oproj } (\bigcirc g))$
using *OPJ2[of f] OPJ1[of f LIFT empty LIFT $\bigcirc g$]* **by** *fastforce*
have 3: $\vdash f \text{ oproj } (\bigcirc g) = (f \wedge more \wedge finite);(f \text{ oproj } g)$
by *(simp add: OProjNext)*
show *?thesis*
using 1 2 3 **by** *fastforce*

qed

lemma *FProjIntro*:

assumes $\vdash f \longrightarrow ((g \wedge \text{more}) \wedge \text{finite}); f$
shows $\vdash f \wedge \text{finite} \longrightarrow g \text{ fproj } \# \text{True}$
using *assms SCSIntro[of f g] FProjTrueEqvSChopstar[of g] unfolding schop-d-def*
by *fastforce*

lemma *OProjIntro*:

assumes $\vdash f \longrightarrow ((g \wedge \text{more}) \wedge \text{finite}); f$
shows $\vdash f \longrightarrow g \text{ oproj } \# \text{True}$
using *assms OProjTrueEqvOmega[of g]*
by (*metis AndMoreAndFiniteEqvAndFmore OmegaIntro inteq-reflection*)

lemma *RightBoxStateImportFProj*:

$\vdash \Box(\text{init } w) \wedge f \text{ fproj } g \longrightarrow f \text{ fproj } (\Box(\text{init } w) \wedge g)$
by (*simp add: Valid-def always-defs init-defs fprojection-d-def*)
(metis ndropn-all ndropn-nfirst nfinite-conv-nlength-enat nle-le pfilt-code(1) pfilt-nmap-pfilt)

lemma *RightBoxStateImportOProj*:

$\vdash \Box(\text{init } w) \wedge f \text{ oproj } g \longrightarrow f \text{ oproj } (\Box(\text{init } w) \wedge g)$
by (*simp add: Valid-def always-defs init-defs oprojection-d-def*)
(metis min-def ndropn-nfirst nfinite-conv-nlength-enat nfinite-ntaken ntaken-nlength pfilt-nnth)

lemma *LeftBoxStateImportFProjhelp*:

$(\forall n. \text{enat } n \leq \text{nlength } wa \longrightarrow w (\text{NNil } (nfirst (\text{ndropn } n \text{ wa})))) \wedge$
 $(\exists ls. (\forall i. \text{enat } (Suc \ i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls \ i < \text{nnth } ls \ (Suc \ i)) \wedge$
 $\text{nnth } ls \ 0 = 0 \wedge$
 $nfinite \ ls \wedge$
 $nfinite \ wa \wedge$
 $\text{nlast } ls = \text{the-enat } (\text{nlength } wa) \wedge$
 $(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow f (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls \ (Suc \ i)))) \wedge g (\text{pfilt } wa \ ls)) \longrightarrow$
 $(\exists ls. (\forall i. \text{enat } (Suc \ i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls \ i < \text{nnth } ls \ (Suc \ i)) \wedge$
 $\text{nnth } ls \ 0 = 0 \wedge$
 $nfinite \ ls \wedge$
 $nfinite \ wa \wedge$
 $\text{nlast } ls = \text{the-enat } (\text{nlength } wa) \wedge$
 $(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow$
 $f (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls \ (Suc \ i))) \wedge$
 $(\forall n. \text{enat } n \leq \text{nlength } (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls \ (Suc \ i))) \longrightarrow$
 $w (\text{NNil } (nfirst (\text{ndropn } n (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls \ (Suc \ i)))))) \wedge$
 $g (\text{pfilt } wa \ ls))$

proof

assume $0: (\forall n. \text{enat } n \leq \text{nlength } wa \longrightarrow w (\text{NNil } (nfirst (\text{ndropn } n \text{ wa})))) \wedge$
 $(\exists ls. (\forall i. \text{enat } (Suc \ i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls \ i < \text{nnth } ls \ (Suc \ i)) \wedge$
 $\text{nnth } ls \ 0 = 0 \wedge$
 $nfinite \ ls \wedge$
 $nfinite \ wa \wedge$
 $\text{nlast } ls = \text{the-enat } (\text{nlength } wa) \wedge$

$(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow f (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i)))) \wedge g (\text{pfilt } wa \ ls))$
show $(\exists ls. (\forall i. \text{enat } (\text{Suc } i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls \ i < \text{nnth } ls (\text{Suc } i)) \wedge$
 $\text{nnth } ls \ 0 = 0 \wedge$
 $\text{nfinite } ls \wedge$
 $\text{nfinite } wa \wedge$
 $\text{nlast } ls = \text{the-enat } (\text{nlength } wa) \wedge$
 $(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow$
 $f (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) \wedge$
 $(\forall n. \text{enat } n \leq \text{nlength } (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) \longrightarrow$
 $w (\text{NNil } (\text{nfirst } (\text{ndropn } n (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i)))))) \wedge$
 $g (\text{pfilt } wa \ ls))$
proof –
have 1: $(\forall n. \text{enat } n \leq \text{nlength } wa \longrightarrow w (\text{NNil } (\text{nfirst } (\text{ndropn } n \ wa))))$
using 0 **by** *auto*
have 2: $(\exists ls. (\forall i. \text{enat } (\text{Suc } i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls \ i < \text{nnth } ls (\text{Suc } i)) \wedge$
 $\text{nnth } ls \ 0 = 0 \wedge$
 $\text{nfinite } ls \wedge$
 $\text{nfinite } wa \wedge$
 $\text{nlast } ls = \text{the-enat } (\text{nlength } wa) \wedge$
 $(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow$
 $f (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) \wedge g (\text{pfilt } wa \ ls))$
using 0 **by** *auto*
obtain *ls* **where** 3: $(\forall i. \text{enat } (\text{Suc } i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls \ i < \text{nnth } ls (\text{Suc } i)) \wedge$
 $\text{nnth } ls \ 0 = 0 \wedge$
 $\text{nfinite } ls \wedge$
 $\text{nfinite } wa \wedge$
 $\text{nlast } ls = \text{the-enat } (\text{nlength } wa) \wedge$
 $(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow$
 $f (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) \wedge g (\text{pfilt } wa \ ls)$
using 2 **by** *auto*
have 4: $\text{nnth } ls \ 0 = 0$
using 3 **by** *auto*
have 5: $(\forall i. \text{enat } (\text{Suc } i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls \ i < \text{nnth } ls (\text{Suc } i))$
using 3 **by** *auto*
have 6: $\text{nlast } ls = \text{the-enat}(\text{nlength } wa)$
using 3 **by** *auto*
have 7: $(\forall i < \text{nlength } ls. f (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))))$
using 3 **by** *auto*
have 8: $g (\text{pfilt } wa \ ls)$
using 3 **by** *auto*
have 9: $(\forall i < \text{nlength } ls.$
 $f (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) \wedge$
 $(\forall n \leq (\text{nnth } ls (\text{Suc } i)) - (\text{nnth } ls \ i).$
 $w (\text{NNil } (\text{nnth } wa ((\text{nnth } ls \ i) + n))))$
using 1 7
by (*metis* 0 *ndropn-eq-NNil ndropn-nfirst ndropn-nlast nfinite-conv-nlength-enat*
nnth-beyond not-le-imp-less the-enat.simps)
have 10: $(\forall i < \text{nlength } ls.$
 $\text{nlength } (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) =$
 $(\text{nnth } ls (\text{Suc } i)) - (\text{nnth } ls \ i)$

by (simp add: 3 PJ6help1 Suc-ile-eq nidx-expand)
 have 11: $(\forall i < \text{nlength } ls. (\forall n \leq (\text{nnth } ls (Suc i)) - (\text{nnth } ls i). \text{nnth } (\text{nsbn wa } (\text{nnth } ls i) (\text{nnth } ls (Suc i))) n = \text{nnth } wa ((\text{nnth } ls i) + n)))$
 using 3 by (simp add: nsubn-def1 ntaken-nnth)
 have 12: $(\forall i < \text{nlength } ls. f (\text{nsbn wa } (\text{nnth } ls i) (\text{nnth } ls (Suc i))) \wedge (\forall n \leq \text{nlength } (\text{nsbn wa } (\text{nnth } ls i) (\text{nnth } ls (Suc i))). w (NNil (\text{nnth } (\text{nsbn wa } (\text{nnth } ls i) (\text{nnth } ls (Suc i))) n))))$
 using 9 10 11 by simp
 have 13: $(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow f (\text{nsbn wa } (\text{nnth } ls i) (\text{nnth } ls (Suc i))) \wedge (\forall n. \text{enat } n \leq \text{nlength } (\text{nsbn wa } (\text{nnth } ls i) (\text{nnth } ls (Suc i))) \longrightarrow w (NNil (\text{nfirst } (\text{ndropn } n (\text{nsbn wa } (\text{nnth } ls i) (\text{nnth } ls (Suc i)))))))$
 by (simp add: 12 ndropn-nfirst)
 show ?thesis
 using 13 3 by blast
 qed
 qed

lemma LeftBoxStateImportOProjhelp:

$(\forall n. \text{enat } n \leq \text{nlength } wa \longrightarrow w (NNil (\text{nfirst } (\text{ndropn } n wa)))) \wedge$
 $(\exists ls. (\forall i. \text{enat } (Suc i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls i < \text{nnth } ls (Suc i)) \wedge \text{nnth } ls 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } wa \wedge$
 $(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow f (\text{nsbn wa } (\text{nnth } ls i) (\text{nnth } ls (Suc i)))) \wedge g (\text{pfilt } wa ls)) \longrightarrow$
 $(\exists ls. (\forall i. \text{enat } (Suc i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls i < \text{nnth } ls (Suc i)) \wedge \text{nnth } ls 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } wa \wedge$
 $(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow f (\text{nsbn wa } (\text{nnth } ls i) (\text{nnth } ls (Suc i))) \wedge$
 $(\forall n. \text{enat } n \leq \text{nlength } (\text{nsbn wa } (\text{nnth } ls i) (\text{nnth } ls (Suc i))) \longrightarrow w (NNil (\text{nfirst } (\text{ndropn } n (\text{nsbn wa } (\text{nnth } ls i) (\text{nnth } ls (Suc i))))))) \wedge$
 $g (\text{pfilt } wa ls))$

proof

assume 0: $(\forall n. \text{enat } n \leq \text{nlength } wa \longrightarrow w (NNil (\text{nfirst } (\text{ndropn } n wa)))) \wedge$
 $(\exists ls. (\forall i. \text{enat } (Suc i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls i < \text{nnth } ls (Suc i)) \wedge \text{nnth } ls 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } wa \wedge$
 $(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow f (\text{nsbn wa } (\text{nnth } ls i) (\text{nnth } ls (Suc i)))) \wedge g (\text{pfilt } wa ls))$
 show $(\exists ls. (\forall i. \text{enat } (Suc i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls i < \text{nnth } ls (Suc i)) \wedge \text{nnth } ls 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } wa \wedge$
 $(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow f (\text{nsbn wa } (\text{nnth } ls i) (\text{nnth } ls (Suc i))) \wedge$
 $(\forall n. \text{enat } n \leq \text{nlength } (\text{nsbn wa } (\text{nnth } ls i) (\text{nnth } ls (Suc i))) \longrightarrow w (NNil (\text{nfirst } (\text{ndropn } n (\text{nsbn wa } (\text{nnth } ls i) (\text{nnth } ls (Suc i))))))) \wedge$
 $g (\text{pfilt } wa ls))$

$g (pfilt\ wa\ ls))$
proof –
have 1: $(\forall n. enat\ n \leq nlength\ wa \longrightarrow w\ (NNil\ (nfirst\ (ndropn\ n\ wa))))$
using 0 **by** *auto*
have 2: $(\exists ls. (\forall i. enat\ (Suc\ i) \leq nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i)) \wedge$
 $nnth\ ls\ 0 = 0 \wedge$
 $\neg\ nfinite\ ls \wedge \neg\ nfinite\ wa \wedge$
 $(\forall i. enat\ i < nlength\ ls \longrightarrow f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \wedge g\ (pfilt\ wa\ ls))$
using 0 **by** *auto*
obtain *ls* **where** 3: $(\forall i. enat\ (Suc\ i) \leq nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i)) \wedge$
 $nnth\ ls\ 0 = 0 \wedge$
 $\neg\ nfinite\ ls \wedge \neg\ nfinite\ wa \wedge$
 $(\forall i. enat\ i < nlength\ ls \longrightarrow f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \wedge g\ (pfilt\ wa\ ls)$
using 2 **by** *auto*
have 4: $nnth\ ls\ 0 = 0$
using 3 **by** *auto*
have 5: $(\forall i. enat\ (Suc\ i) \leq nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i))$
using 3 **by** *auto*
have 7: $(\forall i < nlength\ ls. f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$
using 3 **by** *auto*
have 8: $g\ (pfilt\ wa\ ls)$
using 3 **by** *auto*
have 9: $(\forall i < nlength\ ls.$
 $f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \wedge$
 $(\forall n \leq (nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i).$
 $w\ (NNil\ (nnth\ wa\ ((nnth\ ls\ i) + n))))$
using 1 7
by (*metis* 0 *linorder-le-cases* *ndropn-eq-NNil* *ndropn-nfirst* *nfinite-NNil* *nfinite-ndropn-b*)
have 10: $(\forall i < nlength\ ls.$
 $nlength\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) =$
 $(nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i)$
by (*simp* *add: 3 OPJ6help1 nidx-expand*)
have 11: $(\forall i < nlength\ ls.$
 $(\forall n \leq (nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i).$
 $nnth\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ n =$
 $nnth\ wa\ ((nnth\ ls\ i) + n))$
using 3 **by** (*simp* *add: nsubn-def1 ntaken-nnth*)
have 12: $(\forall i < nlength\ ls.$
 $f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \wedge$
 $(\forall n \leq nlength\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
 $w\ (NNil\ (nnth\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ n))))$
using 9 10 11 **by** *simp*
have 13: $(\forall i. enat\ i < nlength\ ls \longrightarrow$
 $f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \wedge$
 $(\forall n. enat\ n \leq nlength\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \longrightarrow$
 $w\ (NNil\ (nfirst\ (ndropn\ n\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))$
by (*simp* *add: 12 ndropn-nfirst*)
show ?thesis
using 13 3 **by** *blast*
qed

qed

lemma *LeftBoxStateImportFProj:*

$\vdash \Box(\text{init } w) \wedge f \text{ fproj } g \longrightarrow (f \wedge \Box(\text{init } w)) \text{ fproj } g$

using *LeftBoxStateImportFProjhelp*[of - w f g]

by (*simp add: Valid-def always-defs init-defs fprojection-d-def powerinterval-def nidx-expand*)

lemma *LeftBoxStateImportOProj:*

$\vdash \Box(\text{init } w) \wedge f \text{ oproj } g \longrightarrow (f \wedge \Box(\text{init } w)) \text{ oproj } g$

using *LeftBoxStateImportOProjhelp*[of - w f g]

by (*simp add: Valid-def always-defs init-defs oprojection-d-def powerinterval-def nidx-expand*)

fdp, fbp odp and obp

lemma *NotFDpEqvFBpNot:*

$\vdash (\neg(\text{fdp } f)) = \text{fbp } (\neg f)$

by (*simp add: fbp-d-def fdp-d-def ufprojection-d-def*)

lemma *NotODpEqvOBpNot:*

$\vdash (\neg(\text{odp } f)) = \text{obp } (\neg f)$

by (*simp add: obp-d-def odp-d-def uoprojection-d-def*)

lemma *NotFDBpEqvFDpNot:*

$\vdash (\neg(\text{fbp } f)) = \text{fdp } (\neg f)$

by (*simp add: fbp-d-def fdp-d-def ufprojection-d-def*)

lemma *NotODBpEqvODpNot:*

$\vdash (\neg(\text{obp } f)) = \text{odp } (\neg f)$

by (*simp add: obp-d-def odp-d-def uoprojection-d-def*)

lemma *NowImpFDp:*

$\vdash f \wedge \text{finite} \longrightarrow \text{fdp } f$

proof –

have 1: $\vdash (\text{skip} \longrightarrow \# \text{True})$

by *simp*

have 2: $\vdash \text{ba}(\text{skip} \longrightarrow \# \text{True})$

using 1 **by** (*simp add: BaGen*)

have 3: $\vdash \text{ba}(\text{skip} \longrightarrow \# \text{True}) \longrightarrow (\text{skip } \text{fproj } f \longrightarrow \# \text{True } \text{fproj } f)$

using *PJ8* **by** *blast*

have 4: $\vdash (\text{skip } \text{fproj } f \longrightarrow \# \text{True } \text{fproj } f)$

using 2 3 *MP* **by** *blast*

show *?thesis* **using** 4 *PJ6*

by (*metis* 4 *PJ6* *fdp-d-def inteq-reflection*)

qed

lemma *NowImpODp:*

$\vdash f \wedge \text{inf} \longrightarrow \text{odp } f$

proof –

have 1: $\vdash (\text{skip} \longrightarrow \# \text{True})$

by *simp*

```

have 2:  $\vdash ba(skip \longrightarrow \#True)$ 
  using 1 by (simp add: BaGen)
have 3:  $\vdash ba(skip \longrightarrow \#True) \longrightarrow (skip \text{ oproj } f \longrightarrow \#True \text{ oproj } f)$ 
  using OPJ8 by blast
have 4:  $\vdash (skip \text{ oproj } f \longrightarrow \#True \text{ oproj } f)$ 
  using 2 3 MP by blast
show ?thesis using 4 OPJ6 by (metis int-eq odp-d-def)
qed

```

lemma *FBpElim*:

$\vdash fbp\ f \wedge finite \longrightarrow f$

proof –

have 1: $\vdash \neg f \wedge finite \longrightarrow fdp\ (\neg f)$

by (simp add: NowImpFDp)

hence 2: $\vdash \neg(fdp\ (\neg f)) \longrightarrow f \vee inf$

unfolding finite-d-def by auto

from 2 show ?thesis

by (simp add: Prop13 fbp-d-def fdp-d-def finite-d-def ufprojection-d-def)

qed

lemma *OBpElim*:

$\vdash obp\ f \wedge inf \longrightarrow f$

proof –

have 1: $\vdash \neg f \wedge inf \longrightarrow odp\ (\neg f)$

by (simp add: NowImpODp)

hence 2: $\vdash \neg(odp\ (\neg f)) \longrightarrow f \vee finite$

unfolding finite-d-def by auto

from 2 show ?thesis

by (metis InfEqvNotFinite Prop13 inteq-reflection obp-d-def odp-d-def uoprojection-d-def)

qed

lemma *FBpImpFDpImpFDp*:

$\vdash fbp\ (f \longrightarrow g) \longrightarrow fdp\ f \longrightarrow fdp\ g$

proof –

have 1: $\vdash fbp\ (f \longrightarrow g) \longrightarrow (\#True\ fproj\ f) \longrightarrow (\#True\ fproj\ g)$

by (simp add: PJ9 fbp-d-def)

from 1 show ?thesis by (simp add: fdp-d-def)

qed

lemma *OBpImpODpImpODp*:

$\vdash obp\ (f \longrightarrow g) \longrightarrow odp\ f \longrightarrow odp\ g$

proof –

have 1: $\vdash obp\ (f \longrightarrow g) \longrightarrow (\#True\ oproj\ f) \longrightarrow (\#True\ oproj\ g)$

by (simp add: OPJ9 obp-d-def)

from 1 show ?thesis by (simp add: odp-d-def)

qed

lemma *FBpContraPosImpDist*:

$\vdash fbp\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow (fbp\ f) \longrightarrow (fbp\ g)$

proof –

have 1: $\vdash \text{fbp } (\neg g \longrightarrow \neg f) \longrightarrow (\text{fdp } (\neg g)) \longrightarrow (\text{fdp } (\neg f))$
by (rule *FBpImpFDpImpFDp*)
hence 2: $\vdash \text{fbp } (\neg g \longrightarrow \neg f) \longrightarrow (\neg (\text{fdp } (\neg f))) \longrightarrow (\neg (\text{fdp } (\neg g)))$ **by** *auto*
from 2 **show** ?thesis
by (simp add: *fbp-d-def fdp-d-def ufprojection-d-def*)

qed

lemma *OBpContraPosImpDist*:

$\vdash \text{obp } (\neg g \longrightarrow \neg f) \longrightarrow (\text{obp } f) \longrightarrow (\text{obp } g)$

proof –

have 1: $\vdash \text{obp } (\neg g \longrightarrow \neg f) \longrightarrow (\text{odp } (\neg g)) \longrightarrow (\text{odp } (\neg f))$
by (rule *OBpImpODpImpODp*)
hence 2: $\vdash \text{obp } (\neg g \longrightarrow \neg f) \longrightarrow (\neg (\text{odp } (\neg f))) \longrightarrow (\neg (\text{odp } (\neg g)))$ **by** *auto*
from 2 **show** ?thesis
by (simp add: *obp-d-def odp-d-def uoprojection-d-def*)

qed

lemma *FBpImpDist*:

$\vdash \text{fbp } (f \longrightarrow g) \longrightarrow (\text{fbp } f) \longrightarrow (\text{fbp } g)$

proof –

have 1: $\vdash (f \longrightarrow g) \longrightarrow (\neg g \longrightarrow \neg f)$ **by** *auto*
hence 2: $\vdash \neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g)$ **by** *auto*
hence 3: $\vdash \text{fbp } (\neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g))$ **by** (rule *FBpGen*)
have 4: $\vdash \text{fbp } (\neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g))$
 \longrightarrow
 $\text{fbp } (f \longrightarrow g) \longrightarrow \text{fbp } (\neg g \longrightarrow \neg f)$ **by** (rule *FBpContraPosImpDist*)
have 5: $\vdash \text{fbp } (f \longrightarrow g) \longrightarrow \text{fbp } (\neg g \longrightarrow \neg f)$ **using** 3 4 *MP* **by** *blast*
have 6: $\vdash \text{fbp } (\neg g \longrightarrow \neg f) \longrightarrow (\text{fbp } f) \longrightarrow (\text{fbp } g)$ **by** (rule *FBpContraPosImpDist*)
from 5 6 **show** ?thesis **using** *lift-imp-trans* **by** *blast*

qed

lemma *OBpImpDist*:

$\vdash \text{obp } (f \longrightarrow g) \longrightarrow (\text{obp } f) \longrightarrow (\text{obp } g)$

proof –

have 1: $\vdash (f \longrightarrow g) \longrightarrow (\neg g \longrightarrow \neg f)$ **by** *auto*
hence 2: $\vdash \neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g)$ **by** *auto*
hence 3: $\vdash \text{obp } (\neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g))$ **by** (rule *OBpGen*)
have 4: $\vdash \text{obp } (\neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g))$
 \longrightarrow
 $\text{obp } (f \longrightarrow g) \longrightarrow \text{obp } (\neg g \longrightarrow \neg f)$ **by** (rule *OBpContraPosImpDist*)
have 5: $\vdash \text{obp } (f \longrightarrow g) \longrightarrow \text{obp } (\neg g \longrightarrow \neg f)$ **using** 3 4 *MP* **by** *blast*
have 6: $\vdash \text{obp } (\neg g \longrightarrow \neg f) \longrightarrow (\text{obp } f) \longrightarrow (\text{obp } g)$ **by** (rule *OBpContraPosImpDist*)
from 5 6 **show** ?thesis **using** *lift-imp-trans* **by** *blast*

qed

lemma *FDpImpDpRule*:

assumes $\vdash f \longrightarrow g$

shows $\vdash \text{fdp } f \longrightarrow \text{fdp } g$

proof –

have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence 2: $\vdash \#True \text{ fproj } f \longrightarrow \#True \text{ fproj } g$
by (*metis FBpGen MP PJ9 fbp-d-def*)
from 2 **show** ?thesis **by** (*simp add: fdp-d-def*)
qed

lemma *ODpImpODpRule*:

assumes $\vdash f \longrightarrow g$
shows $\vdash odp \text{ } f \longrightarrow odp \text{ } g$
proof –
have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence 2: $\vdash \#True \text{ oproj } f \longrightarrow \#True \text{ oproj } g$
by (*metis OBpGen MP OPJ9 obp-d-def*)
from 2 **show** ?thesis **by** (*simp add: odp-d-def*)
qed

lemma *FBpImpFBpRule*:

assumes $\vdash f \longrightarrow g$
shows $\vdash fbp \text{ } f \longrightarrow fbp \text{ } g$
proof –
have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence 2: $\vdash \neg g \longrightarrow \neg f$ **by** *auto*
hence 3: $\vdash fdp (\neg g) \longrightarrow fdp (\neg f)$ **by** (*rule FDpImpDpRule*)
hence 4: $\vdash \neg (fdp (\neg f)) \longrightarrow \neg (fdp (\neg g))$ **by** *auto*
from 4 **show** ?thesis
by (*meson FBpGen FBpImpDist MP assms*)
qed

lemma *OBpImpOBpRule*:

assumes $\vdash f \longrightarrow g$
shows $\vdash obp \text{ } f \longrightarrow obp \text{ } g$
proof –
have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence 2: $\vdash \neg g \longrightarrow \neg f$ **by** *auto*
hence 3: $\vdash odp (\neg g) \longrightarrow odp (\neg f)$ **by** (*rule ODpImpODpRule*)
hence 4: $\vdash \neg (odp (\neg f)) \longrightarrow \neg (odp (\neg g))$ **by** *auto*
from 4 **show** ?thesis
by (*meson OBpGen OBpImpDist MP assms*)
qed

lemma *FDpEqvFDpRule*:

assumes $\vdash f = g$
shows $\vdash fdp \text{ } f = fdp \text{ } g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash \#True \text{ fproj } f = \#True \text{ fproj } g$
using *RightFProjEqvFProj* **by** *blast*
from 2 **show** ?thesis **by** (*simp add: fdp-d-def*)
qed

lemma *ODpEqvODpRule*:
assumes $\vdash f = g$
shows $\vdash odp\ f = odp\ g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash \#True\ oproj\ f = \#True\ oproj\ g$
using *RightOProjEqvOProj* **by** *blast*
from 2 **show** *?thesis* **by** (*simp add: odp-d-def*)
qed

lemma *FBpEqvFBpRule*:
assumes $\vdash f = g$
shows $\vdash fbp\ f = fbp\ g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash (\neg f) = (\neg g)$ **by** *auto*
hence 3: $\vdash fdp\ (\neg f) = fdp\ (\neg g)$ **by** (*rule FDpEqvFDpRule*)
hence 4: $\vdash (\neg (fdp\ (\neg f))) = (\neg (fdp\ (\neg g)))$ **by** *auto*
from 4 **show** *?thesis*
by (*metis FBpImpFBpRule assms int-iffD1 int-iffI inteq-reflection*)
qed

lemma *OBpEqvOBpRule*:
assumes $\vdash f = g$
shows $\vdash obp\ f = obp\ g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash (\neg f) = (\neg g)$ **by** *auto*
hence 3: $\vdash odp\ (\neg f) = odp\ (\neg g)$ **by** (*rule ODpEqvODpRule*)
hence 4: $\vdash (\neg (odp\ (\neg f))) = (\neg (odp\ (\neg g)))$ **by** *auto*
from 4 **show** *?thesis*
by (*metis OBpImpOBpRule assms int-iffD1 int-iffI inteq-reflection*)
qed

lemma *FDpState*:
 $\vdash fdp\ (init\ w) = ((init\ w) \wedge finite)$
proof –
have 1: $\vdash init\ w \wedge finite \longrightarrow fdp\ (init\ w)$
using *NowImpFDp[of LIFT (init w)]* **by** *blast*
have 2: $\vdash fdp\ (init\ w) \longrightarrow init\ w$
unfolding *fdp-d-def* **by** (*simp add: PJ5*)
have 3: $\vdash fdp\ (init\ w) \longrightarrow finite$
unfolding *fdp-d-def* **by** (*simp add: PJ01*)
show *?thesis*
by (*simp add: 1 2 3 Prop12 int-iffI*)
qed

lemma *ODpState*:
 $\vdash odp\ (init\ w) = ((init\ w) \wedge inf)$
proof –

have 1: $\vdash \text{init } w \wedge \text{inf} \longrightarrow \text{odp } (\text{init } w)$
using *NowImpODp[of LIFT (init w)]* **by** *blast*
have 2: $\vdash \text{odp } (\text{init } w) \longrightarrow \text{init } w$
unfolding *odp-d-def* **by** (*simp add: OPJ5*)
have 3: $\vdash \text{odp } (\text{init } w) \longrightarrow \text{inf}$
unfolding *odp-d-def* **by** (*simp add: OPJ01*)
show *?thesis*
by (*simp add: 1 2 3 Prop12 int-iffI*)
qed

lemma *StateEqvFBp:*

$\vdash \text{finite} \longrightarrow (\text{init } w) = \text{fbp } (\text{init } w)$
proof –
have 1: $\vdash (\text{init } w) \longrightarrow \text{fbp } (\text{init } w)$
by (*metis (no-types, lifting) DiEqvNotBiNot DiState Initprop(2) NotDiEqvBiNot PJ5 fbp-d-def*
inteq-reflection lift-imp-neg ufprojection-d-def)
have 2: $\vdash \text{fbp } (\text{init } w) \wedge \text{finite} \longrightarrow (\text{init } w)$ **using** *FBpElim* **by** *blast*
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *StateEqvOBp:*

$\vdash \text{inf} \longrightarrow (\text{init } w) = \text{obp } (\text{init } w)$
proof –
have 1: $\vdash (\text{init } w) \longrightarrow \text{obp } (\text{init } w)$
by (*metis (no-types, lifting) DiEqvNotBiNot DiState Initprop(2) NotDiEqvBiNot NotODpEqvOBpNot*
ODpState Prop11 Prop12 inteq-reflection lift-imp-neg)
have 2: $\vdash \text{obp } (\text{init } w) \wedge \text{inf} \longrightarrow (\text{init } w)$ **using** *OBpElim* **by** *blast*
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *FDpFDpEqvFDp:*

$\vdash \text{fdp } (\text{fdp } f) = \text{fdp } f$
proof –
have 2: $\vdash \# \text{True } \text{fproj } (\# \text{True } \text{fproj } f) = (\# \text{True } \text{fproj } \# \text{True}) \text{fproj } f$
by (*simp add: PJ7*)
have 3: $\vdash (\# \text{True } \text{fproj } \# \text{True}) = \text{finite}$
by (*metis ChopEmpty EmptyImpFinite NowImpFDp PJ01 Prop10 TrueChopAndFiniteEqvAndFinite-*
ChopFinite
fdp-d-def int-eq int-iffI)
have 4: $\vdash \text{finite } \text{fproj } f = \# \text{True } \text{fproj } f$
by (*metis PJ02 Prop03 Prop10 finite-d-def int-simps(28) inteq-reflection lift-and-com*)
show *?thesis*
by (*metis 2 3 4 fdp-d-def int-eq*)
qed

lemma *ODpODpEqvODp:*

$\vdash \text{odp } (\text{odp } f) = \text{odp } f$
proof –
have 2: $\vdash \# \text{True } \text{oproj } (\# \text{True } \text{oproj } f) = (\# \text{True } \text{oproj } \# \text{True}) \text{oproj } f$
by (*simp add: OPJ7*)

have 3: $\vdash (\#True \text{ fproj } \#True) = \text{finite}$
by (metis ChopEmpty EmptyImpFinite NowImpFDp PJ01 Prop10 TrueChopAndFiniteEqvAndFiniteChopFinite
fdp-d-def int-eq int-iffI)
have 4: $\vdash \text{finite } \text{oproj } f = \#True \text{ oproj } f$
by (metis OPJ02 Prop03 Prop10 finite-d-def int-simps(28) inteq-reflection lift-and-com)
show ?thesis
by (metis 2 3 4 odp-d-def int-eq)
qed

lemma FBpFBpEqvFBp:
 $\vdash \text{fbp } (\text{fbp } f) = \text{fbp } f$
proof –
have 1: $\vdash \text{fdp } (\text{fdp } (\neg f)) = \text{fdp } (\neg f)$
using FDpFDpEqvFDp **by** blast
have 2: $\vdash (\neg (\text{fdp } (\text{fdp } (\neg f)))) = (\neg (\text{fdp } (\neg f)))$
using 1 **by** auto
have 3: $\vdash (\neg (\text{fdp } (\neg f))) = \text{fbp } f$
by (simp add: fbp-d-def fdp-d-def ufprojection-d-def)
have 4: $\vdash (\neg (\text{fdp } (\text{fdp } (\neg f)))) = \text{fbp } (\text{fbp } f)$
by (simp add: fbp-d-def fdp-d-def ufprojection-d-def)
from 2 3 4 **show** ?thesis
by fastforce
qed

lemma OBpOBpEqvOBp:
 $\vdash \text{obp } (\text{obp } f) = \text{obp } f$
proof –
have 1: $\vdash \text{odp } (\text{odp } (\neg f)) = \text{odp } (\neg f)$
using ODpODpEqvODp **by** blast
have 2: $\vdash (\neg (\text{odp } (\text{odp } (\neg f)))) = (\neg (\text{odp } (\neg f)))$
using 1 **by** auto
have 3: $\vdash (\neg (\text{odp } (\neg f))) = \text{obp } f$
by (simp add: obp-d-def odp-d-def uoprojection-d-def)
have 4: $\vdash (\neg (\text{odp } (\text{odp } (\neg f)))) = \text{obp } (\text{obp } f)$
by (simp add: obp-d-def odp-d-def uoprojection-d-def)
from 2 3 4 **show** ?thesis
by fastforce
qed

lemma FDpOrEqv:
 $\vdash \text{fdp } (f \vee g) = (\text{fdp } f \vee \text{fdp } g)$
proof –
have 1: $\vdash \#True \text{ fproj } (f \vee g) = (\#True \text{ fproj } f \vee \#True \text{ fproj } g)$
using FProjOrDist **by** auto
from 1 **show** ?thesis **by** (simp add: fdp-d-def)
qed

lemma ODpOrEqv:

$\vdash \text{odp } (f \vee g) = (\text{odp } f \vee \text{odp } g)$
proof –
have 1: $\vdash \# \text{True } \text{oproj } (f \vee g) = (\# \text{True } \text{oproj } f \vee \# \text{True } \text{oproj } g)$
using *OProjOrDist* **by** *auto*
from 1 **show** *?thesis* **by** (*simp add: odp-d-def*)
qed

lemma *FBpAndEqv*:
 $\vdash \text{fbp}(f \wedge g) = (\text{fbp } f \wedge \text{fbp } g)$
proof –
have 1: $\vdash \text{fdp } ((\neg f) \vee (\neg g)) = (\text{fdp } (\neg f) \vee \text{fdp } (\neg g))$
using *FDpOrEqv* **by** *auto*
hence 2: $\vdash (\neg (\text{fdp } ((\neg f) \vee (\neg g)))) = (\neg (\text{fdp } (\neg f) \vee \text{fdp } (\neg g)))$
by *auto*
have 3: $\vdash (\neg (\text{fdp } ((\neg f) \vee (\neg g)))) = \text{fbp } (\neg((\neg f) \vee (\neg g)))$
using *NotFDpEqvFBpNot* **by** *blast*
have 4: $\vdash (\neg((\neg f) \vee (\neg g))) = (f \wedge g)$
by *auto*
hence 5: $\vdash \text{fbp}(\neg((\neg f) \vee (\neg g))) = \text{fbp}(f \wedge g)$
by (*simp add: FBpEqvFBpRule*)
have 6: $\vdash (\neg(\text{fdp } (\neg f) \vee \text{fdp } (\neg g))) = ((\neg(\text{fdp } (\neg f))) \wedge (\neg(\text{fdp } (\neg g))))$
by *auto*
have 7: $\vdash ((\neg(\text{fdp } (\neg f))) \wedge (\neg(\text{fdp } (\neg g)))) = (\text{fbp } f \wedge \text{fbp } g)$
by (*simp add: fbp-d-def fdp-d-def ufprojection-d-def*)
show *?thesis*
by (*metis 2 3 4 6 7 inteq-reflection*)
qed

lemma *OBpAndEqv*:
 $\vdash \text{obp}(f \wedge g) = (\text{obp } f \wedge \text{obp } g)$
proof –
have 1: $\vdash \text{odp } ((\neg f) \vee (\neg g)) = (\text{odp } (\neg f) \vee \text{odp } (\neg g))$
using *ODpOrEqv* **by** *auto*
hence 2: $\vdash (\neg (\text{odp } ((\neg f) \vee (\neg g)))) = (\neg (\text{odp } (\neg f) \vee \text{odp } (\neg g)))$
by *auto*
have 3: $\vdash (\neg (\text{odp } ((\neg f) \vee (\neg g)))) = \text{obp } (\neg((\neg f) \vee (\neg g)))$
using *NotODpEqvOBpNot* **by** *blast*
have 4: $\vdash (\neg((\neg f) \vee (\neg g))) = (f \wedge g)$
by *auto*
hence 5: $\vdash \text{obp}(\neg((\neg f) \vee (\neg g))) = \text{obp}(f \wedge g)$
by (*simp add: OBpEqvOBpRule*)
have 6: $\vdash (\neg(\text{odp } (\neg f) \vee \text{odp } (\neg g))) = ((\neg(\text{odp } (\neg f))) \wedge (\neg(\text{odp } (\neg g))))$
by *auto*
have 7: $\vdash ((\neg(\text{odp } (\neg f))) \wedge (\neg(\text{odp } (\neg g)))) = (\text{obp } f \wedge \text{obp } g)$
by (*simp add: obp-d-def odp-d-def uoprojection-d-def*)
show *?thesis*
by (*metis 2 3 4 6 7 inteq-reflection*)
qed

lemma *FDpAndA*:

$\vdash \text{fdp } (f \wedge g) \longrightarrow \text{fdp } f$
proof –
have 1: $\vdash \# \text{True } \text{fproj } (f \wedge g) \longrightarrow \# \text{True } \text{fproj } f$
by (meson Prop12 RightFProjImpFProj int-iffD1 lift-and-com)
from 1 **show** ?thesis **by** (simp add: fdp-d-def)
qed

lemma ODpAndA:
 $\vdash \text{odp } (f \wedge g) \longrightarrow \text{odp } f$
proof –
have 1: $\vdash \# \text{True } \text{oproj } (f \wedge g) \longrightarrow \# \text{True } \text{oproj } f$
by (meson Prop12 RightOProjImpOProj int-iffD1 lift-and-com)
from 1 **show** ?thesis **by** (simp add: odp-d-def)
qed

lemma FBpOrA:
 $\vdash \text{fbp } f \longrightarrow \text{fbp}(f \vee g)$
by (simp add: FBpImpFBpRule intI)

lemma OBpOrA:
 $\vdash \text{obp } f \longrightarrow \text{obp}(f \vee g)$
by (simp add: OBpImpOBpRule intI)

lemma FBpOrB:
 $\vdash \text{fbp } g \longrightarrow \text{fbp}(f \vee g)$
by (simp add: FBpImpFBpRule intI)

lemma OBpOrB:
 $\vdash \text{obp } g \longrightarrow \text{obp}(f \vee g)$
by (simp add: OBpImpOBpRule intI)

lemma FBpOrImpOr:
 $\vdash \text{fbp } f \vee \text{fbp } g \longrightarrow \text{fbp}(f \vee g)$
using FBpOrA FBpOrB **by** fastforce

lemma OBpOrImpOr:
 $\vdash \text{obp } f \vee \text{obp } g \longrightarrow \text{obp}(f \vee g)$
using OBpOrA OBpOrB **by** fastforce

lemma FDpAndB:
 $\vdash \text{fdp } (f \wedge g) \longrightarrow \text{fdp } g$
proof –
have 1: $\vdash \# \text{True } \text{fproj } (f \wedge g) \longrightarrow \# \text{True } \text{fproj } g$
by (meson Prop12 RightFProjImpFProj int-iffD2 lift-and-com)
from 1 **show** ?thesis **by** (simp add: fdp-d-def)
qed

lemma ODpAndB:
 $\vdash \text{odp } (f \wedge g) \longrightarrow \text{odp } g$
proof –

have 1: $\vdash \#True \text{ oproj } (f \wedge g) \longrightarrow \#True \text{ oproj } g$
by (*meson Prop12 RightOProjImpOProj int-iffD2 lift-and-com*)
from 1 **show** ?thesis **by** (*simp add: odp-d-def*)
qed

lemma *FDpAndImpAnd*:
 $\vdash \text{fdp } (f \wedge g) \longrightarrow \text{fdp } f \wedge \text{fdp } g$
proof –
have 1: $\vdash \text{fdp } (f \wedge g) \longrightarrow \text{fdp } f$ **by** (*rule FDpAndA*)
have 2: $\vdash \text{fdp } (f \wedge g) \longrightarrow \text{fdp } g$ **by** (*rule FDpAndB*)
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *ODpAndImpAnd*:
 $\vdash \text{odp } (f \wedge g) \longrightarrow \text{odp } f \wedge \text{odp } g$
proof –
have 1: $\vdash \text{odp } (f \wedge g) \longrightarrow \text{odp } f$ **by** (*rule ODpAndA*)
have 2: $\vdash \text{odp } (f \wedge g) \longrightarrow \text{odp } g$ **by** (*rule ODpAndB*)
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *FDpSkipEqvMore*:
 $\vdash \text{fdp } \text{skip} = (\text{more} \wedge \text{finite})$
proof –
have 1: $\vdash \text{fdp } \text{skip} = \#True \text{ fproj } \text{skip}$
by (*simp add: fdp-d-def*)
have 2: $\vdash \#True \text{ fproj } \text{skip} = (\#True \wedge \text{more} \wedge \text{finite})$
using *PJ3* **by** *blast*
have 3: $\vdash (\#True \wedge \text{more}) = \text{more}$
by *auto*
from 1 2 3 **show** ?thesis **by** *fastforce*
qed

lemma *FDpMoreEqvMore*:
 $\vdash \text{fdp } \text{more} = (\text{more} \wedge \text{finite})$
using *FDpFDpEqvFDp FDpSkipEqvMore*
by (*metis PJ03 fdp-d-def inteq-reflection*)

lemma *ODpMoreEqvInf*:
 $\vdash \text{odp } \text{more} = (\text{inf})$
by (*metis MoreAndInfEqvInf NowImpODp OPJ01 int-iffI inteq-reflection odp-d-def*)

lemma *FBpEmptyEqvEmpty*:
 $\vdash \text{fbp } \text{empty} = (\text{empty} \vee \text{inf})$
proof –
have 1: $\vdash \text{fbp } \text{empty} = (\neg (\text{fdp } \text{more}))$
by (*metis NotFDpEqvFBpNot Prop11 empty-d-def*)
have 2: $\vdash (\neg (\text{fdp } \text{more})) = (\neg (\text{more} \wedge \text{finite}))$
using *FDpMoreEqvMore* **by** *auto*

have 3: $\vdash (\neg (\text{more} \wedge \text{finite})) = (\text{empty} \vee \text{inf})$
unfolding *finite-d-def empty-d-def* **by** *fastforce*
show *?thesis*
by (*metis* 1 2 3 *int-eq*)
qed

lemma *OBpEmptyEqvFinite*:
 $\vdash \text{obp empty} = (\text{finite})$
proof –
have 1: $\vdash \text{obp empty} = (\neg (\text{odp more}))$
by (*metis* *NotODpEqvOBpNot Prop11 empty-d-def*)
have 2: $\vdash (\neg (\text{odp more})) = (\neg (\text{inf}))$
using *ODpMoreEqvInf* **by** *auto*
have 3: $\vdash (\neg (\text{inf})) = (\text{finite})$
unfolding *finite-d-def* **by** *fastforce*
show *?thesis*
by (*metis* 1 2 3 *int-eq*)
qed

lemma *FDpEmptyEqvEmpty*:
 $\vdash \text{fdp empty} = \text{empty}$
proof –
have 1: $\vdash \text{fdp empty} = \# \text{True fproj empty}$
by (*simp add: fdp-d-def*)
have 2: $\vdash \# \text{True fproj empty} = \text{empty}$
by (*simp add: PJ2*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *NotODpEmpty*:
 $\vdash \neg (\text{odp empty})$
proof –
have 1: $\vdash \text{odp empty} = \# \text{True oproj empty}$
by (*simp add: odp-d-def*)
show *?thesis*
by (*metis* 1 *OPJ2 int-eq*)
qed

lemma *FBpMoreEqvMore*:
 $\vdash \text{fbp more} = \text{more}$
by (*metis* *NotFDBpEqvFDpNot PJ2 empty-d-def fdp-d-def int-eq int-simps(4)*)

lemma *OBpMore*:
 $\vdash \text{obp more}$
by (*metis* *OPJ2 empty-d-def obp-d-def uoprojection-d-def*)

lemma *NextFDpImpFDpNext*:
 $\vdash \circ (\text{fdp } f) \longrightarrow \text{fdp } (\circ f)$

proof –
have 1: $\vdash \text{fdp}(\circ f) = \# \text{True } \text{fproj } (\text{skip}; f)$
 by (*simp add: fdp-d-def next-d-def*)
have 2: $\vdash \# \text{True } \text{fproj } (\text{skip}; f) = (\# \text{True } \text{fproj } \text{skip}); (\# \text{True } \text{fproj } f)$
 by (*simp add: PJ4*)
have 3: $\vdash (\# \text{True } \text{fproj } \text{skip}) = (\# \text{True} \wedge \text{more} \wedge \text{finite})$
 using *PJ3* by *blast*
have 4: $\vdash (\# \text{True} \wedge \text{more}) = \text{more}$
 by *auto*
have 40: $\vdash \text{skip} \longrightarrow \text{more}$
 by (*metis DiIntro DiSkipEqvMore int-eq*)
have 41: $\vdash \text{skip} \longrightarrow \text{finite}$
 by (*metis AndChopB EmptySChop FiniteChopSkipImpFinite Prop11 lift-imp-trans schop-d-def*)
have 5: $\vdash \text{skip}; (\# \text{True } \text{fproj } f) \longrightarrow (\text{more} \wedge \text{finite}); (\# \text{True } \text{fproj } f)$
 by (*simp add: 40 41 LeftChopImpChop Prop12*)
show ?thesis by (*metis 2 5 FDpSkipEqvMore fdp-d-def inteq-reflection next-d-def*)
qed

lemma *NextODpImpODpNext*:

$\vdash \circ (\text{odp } f) \longrightarrow \text{odp } (\circ f)$

proof –

have 1: $\vdash \text{odp}(\circ f) = \# \text{True } \text{oproj } (\text{skip}; f)$
 by (*simp add: odp-d-def next-d-def*)
have 10: $\vdash (\text{skip} \wedge \text{finite}) = \text{skip}$
 by (*metis FiniteChopEqvDiamond FiniteChopSkipEqvSkipChopFinite NowImpDiamond Prop10 Prop11 SkipChopFiniteImpFinite lift-imp-trans*)
have 2: $\vdash \# \text{True } \text{oproj } (\text{skip}; f) = (\# \text{True } \text{fproj } \text{skip}); (\# \text{True } \text{oproj } f)$
 using *OPJ4* by (*metis 10 inteq-reflection*)
have 3: $\vdash (\# \text{True } \text{fproj } \text{skip}) = (\# \text{True} \wedge \text{more} \wedge \text{finite})$
 using *PJ3* by *blast*
have 4: $\vdash (\# \text{True} \wedge \text{more}) = \text{more}$
 by *auto*
have 40: $\vdash \text{skip} \longrightarrow \text{more}$
 by (*metis DiIntro DiSkipEqvMore int-eq*)
have 41: $\vdash \text{skip} \longrightarrow \text{finite}$
 by (*metis AndChopB EmptySChop FiniteChopSkipImpFinite Prop11 lift-imp-trans schop-d-def*)
have 5: $\vdash \text{skip}; (\# \text{True } \text{oproj } f) \longrightarrow (\text{more} \wedge \text{finite}); (\# \text{True } \text{oproj } f)$
 by (*simp add: 40 41 LeftChopImpChop Prop12*)
show ?thesis by (*metis 2 5 FDpSkipEqvMore fdp-d-def inteq-reflection next-d-def odp-d-def*)
qed

lemma *BoxStateImportFBp*:

$\vdash \Box(\text{init } w) \longrightarrow \text{fbp } (\Box(\text{init } w))$

proof –

have 1: $\vdash \text{fbp } (\Box(\text{init } w)) = (\neg(\text{fdp } (\Diamond(\neg(\text{init } w)))))$
 by (*metis NotFDpEqvFBpNot always-d-def int-eq*)
have 2: $\vdash (\Diamond(\neg(\text{init } w))) = (\text{finite}; (\neg(\text{init } w)))$
 by (*simp add: sometimes-d-def*)
have 3: $\vdash \text{fdp } (\text{finite}; (\neg(\text{init } w))) = (\text{fdp } \text{finite}); (\text{fdp } (\neg(\text{init } w)))$
 by (*simp add: PJ4 fdp-d-def*)

have 4: $\vdash (fdp \text{ finite}) = \text{finite}$
by (*metis EmptyOrMoreSplit FDPEmptyEqvEmpty FiniteAndEmptyEqvEmpty FiniteChopSkipEqvFinite-AndMore*
FiniteChopSkipImpFinite NowImpFDP PJ01 Prop02 Prop10 fdp-d-def int-iffI inteq-reflection)
have 5: $\vdash (fdp (\neg(\text{init } w))) = ((\neg(\text{init } w)) \wedge \text{finite})$
by (*metis FDPState Initprop(2) inteq-reflection*)
have 6: $\vdash \text{finite};((\neg(\text{init } w)) \wedge \text{finite}) \longrightarrow \Diamond(\neg(\text{init } w))$
by (*metis 2 ChopAndA inteq-reflection*)
show ?thesis
by (*metis 1 2 3 4 5 6 always-d-def inteq-reflection lift-imp-neg*)
qed

lemma *BoxStateImportOBp*:

$\vdash \Box(\text{init } w) \longrightarrow \text{obp}(\Box(\text{init } w))$

proof –

have 1: $\vdash \text{obp}(\Box(\text{init } w)) = (\neg(\text{odp}(\Diamond(\neg(\text{init } w)))))$

by (*metis NotODpEqvOBpNot always-d-def int-eq*)

have 2: $\vdash (\Diamond(\neg(\text{init } w))) = (\text{finite};(\neg(\text{init } w)))$

by (*simp add: sometimes-d-def*)

have 3: $\vdash \text{odp}(\text{finite};(\neg(\text{init } w))) = (fdp \text{ finite});(\text{odp}(\neg(\text{init } w)))$

unfolding *odp-d-def fdp-d-def*

using *OPJ4[of LIFT #True LIFT finite LIFT ($\neg(\text{init } w)$)]*

by (*simp add: OPJ4 odp-d-def fdp-d-def*)

have 4: $\vdash (fdp \text{ finite}) = \text{finite}$

by (*metis EmptyOrMoreSplit FDPEmptyEqvEmpty FiniteAndEmptyEqvEmpty FiniteChopSkipEqvFinite-AndMore*
FiniteChopSkipImpFinite NowImpFDP PJ01 Prop02 Prop10 fdp-d-def int-iffI inteq-reflection)

have 5: $\vdash (\text{odp}(\neg(\text{init } w))) = ((\neg(\text{init } w)) \wedge \text{inf})$

by (*metis ODpState Initprop(2) inteq-reflection*)

have 6: $\vdash \text{finite};((\neg(\text{init } w)) \wedge \text{inf}) \longrightarrow \Diamond(\neg(\text{init } w))$

by (*metis 2 ChopAndA inteq-reflection*)

show ?thesis **by** (*metis 1 2 3 4 5 6 always-d-def int-eq lift-imp-neg*)

qed

lemma *BoxStateEqvFBpBoxState*:

$\vdash \text{finite} \longrightarrow \Box(\text{init } w) = \text{fbp}(\Box(\text{init } w))$

proof –

have 1: $\vdash \text{finite} \longrightarrow \text{fbp}(\Box(\text{init } w)) \longrightarrow \Box(\text{init } w)$

by (*metis FBpElim Prop09 inteq-reflection lift-and-com*)

have 2: $\vdash \text{fbp}(\Box(\text{init } w)) = (\neg(\# \text{True } \text{fproj } (\neg \Box(\text{init } w))))$

by (*simp add: fbp-d-def ufprojection-d-def*)

have 2: $\vdash \Box(\text{init } w) \wedge \text{finite} \longrightarrow fdp(\Box(\text{init } w))$

by (*metis NowImpFDP*)

have 2: $\vdash \Box(\text{init } w) \longrightarrow \text{fbp}(\Box(\text{init } w))$

using *BoxStateImportFBp* **by** *auto*

from 1 2 **show** ?thesis **by** *fastforce*

qed

lemma *BoxStateEqvOBpBoxState*:

$\vdash \text{inf} \longrightarrow \Box(\text{init } w) = \text{obp}(\Box(\text{init } w))$

```

proof –
  have 1:  $\vdash \text{inf} \longrightarrow \text{obp}(\Box (\text{init } w)) \longrightarrow \Box (\text{init } w)$ 
    by (metis OBPElim Prop09 integ-reflection lift-and-com)
  have 2:  $\vdash \text{obp}(\Box (\text{init } w)) = (\neg(\# \text{True } \text{oproj } (\neg \Box (\text{init } w))))$ 
    by (simp add: obp-d-def uoprojection-d-def)
  have 2:  $\vdash \Box (\text{init } w) \wedge \text{inf} \longrightarrow \text{odp} (\Box (\text{init } w))$ 
    by (metis NowImpODp)
  have 2:  $\vdash \Box (\text{init } w) \longrightarrow \text{obp}(\Box (\text{init } w))$ 
  using BoxStateImportOBp by auto
  from 1 2 show ?thesis by fastforce
qed

end

```

2.8 Until operator

```

theory Until
imports Semantics SChopTheorems
begin

```

This theory introduces the weak and strong versions of the until operator. The theorems from [15] are proven in a mostly deductive style.

2.8.1 Definitions

```

definition until-d :: ('a :: world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula
  where until-d  $F\ G \equiv \lambda s. ( (\exists k. k \leq \text{nlength } s \wedge ( \text{ndropn } k\ s ) \models G ) \wedge$ 
     $(\forall j. j < k \longrightarrow ( \text{ndropn } j\ s ) \models F ) ) )$ 

```

```

syntax
  -until-d :: [lift,lift]  $\Rightarrow$  lift          ((-  $\mathcal{U}$  -) [84,84] 83)

```

```

syntax (ASCII)
  -until-d :: [lift,lift]  $\Rightarrow$  lift          ((- until -) [84,84] 83)

```

```

translations
  -until-d  $\equiv$  CONST until-d

```

```

definition suntil-d :: ('a :: world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula
  where suntil-d  $F\ G \equiv \text{LIFT}(\Box(F\ \mathcal{U}\ G))$ 

```

```

definition wait-d :: ('a :: world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula
  where wait-d  $F\ G \equiv \text{LIFT}(\Box F \vee F\ \mathcal{U}\ G)$ 

```

```

definition release-d :: ('a :: world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula
  where release-d  $F\ G \equiv \text{LIFT}(\neg((\neg F)\ \mathcal{U}\ (\neg G)))$ 

```

```

syntax
  -wait-d :: [lift,lift]  $\Rightarrow$  lift          ((-  $\mathcal{W}$  -) [84,84] 83)
  -release-d :: [lift,lift]  $\Rightarrow$  lift        ((-  $\mathcal{R}$  -) [84,84] 83)

```


-suntil-d :: [lift, lift] \Rightarrow lift ((- \mathcal{U}^s -) [84,84] 83)

syntax (*ASCII*)

-wait-d :: [lift, lift] \Rightarrow lift ((- wait -) [84,84] 83)
-release-d :: [lift, lift] \Rightarrow lift ((- release -) [84,84] 83)
-suntil-d :: [lift, lift] \Rightarrow lift ((- until -) [84,84] 83)

translations

-wait-d \Rightarrow *CONST wait-d*
-release-d \Rightarrow *CONST release-d*
-suntil-d \Rightarrow *CONST until-d*

definition *srelease-d* :: ('a :: world) formula \Rightarrow 'a formula \Rightarrow 'a formula
where *srelease-d* F G \equiv *LIFT*($\neg((\neg F) \mathcal{W} (\neg G))$)

syntax

-srelease-d :: [lift, lift] \Rightarrow lift ((- \mathcal{M} -) [84,84] 83)

syntax (*ASCII*)

-srelease-d :: [lift, lift] \Rightarrow lift ((- srelease -) [84,84] 83)

translations

-srelease-d \Rightarrow *CONST srelease-d*

2.8.2 Axioms

NextUntil

lemma *NextUntilsema*:

assumes ($\sigma \models \bigcirc(f \mathcal{U} g)$)
shows ($\sigma \models (\bigcirc f) \mathcal{U} (\bigcirc g)$)

proof –

have 0: $0 < \text{nlength } \sigma \wedge$
 $(\exists k. k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \sigma) \wedge g ((\text{ndropn } (\text{Suc } k) \sigma)) \wedge$
 $(\forall j < k. f ((\text{ndropn } (\text{Suc } j) \sigma)))$)
using *assms zero-enat-def* **by** (*auto simp add: next-defs until-d-def ndropn-ndropn*)

have 1: $0 < \text{nlength } \sigma$

using 0 **by** *auto*

have 2: $(\exists k. k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \sigma) \wedge g ((\text{ndropn } (\text{Suc } k) \sigma)) \wedge$
 $(\forall j < k. f ((\text{ndropn } (\text{Suc } j) \sigma))))$

using 0 **by** *auto*

obtain k **where** 3: $k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \sigma) \wedge g ((\text{ndropn } (\text{Suc } k) \sigma)) \wedge$
 $(\forall j < k. f ((\text{ndropn } (\text{Suc } j) \sigma)))$

using 2 **by** *auto*

have 4: $g ((\text{ndropn } (\text{Suc } k) \sigma))$

using 3 **by** *auto*

have 5: $k \leq \text{nlength } \sigma$

using 3 0

by (*metis diff-le-self dual-order.order-iff-strict enat-ile enat-ord-simps(1) idiff-enat-enat*
less-le-trans ndropn-nlength not-less)

have 6: $0 < \text{nlength } (\text{ndropn } k \sigma)$

```

using 1 3
by (metis gr-zeroI illess-Suc-eq is-NNil-ndropn leD le-numeral-extra(3) ndropn-0
    ndropn-Suc-conv-ndropn nlength-NCons zero-enat-def)
have 7: ( $\forall j < k. 0 < \text{nlength } (\text{ndropn } j \ \sigma) \wedge f \ ((\text{ndropn } (\text{Suc } j) \ \sigma))$ )
using 3 5
by (metis enat-ord-simps(2) is-NNil-ndropn ndropn-0 not-less order.trans zero-le)
have 71:  $\text{nlength } \sigma - \text{enat } k \neq \text{enat } 0$ 
using 6 zero-enat-def by auto
have 72: ( $\forall j < k. \text{nlength } \sigma - \text{enat } j \neq \text{enat } 0 \wedge f \ (\text{ndropn } (\text{Suc } j) \ \sigma)$ )
using 7 zero-enat-def by auto
have 8:  $\exists k. k \leq \text{nlength } \sigma \wedge$ 
 $\text{nlength } \sigma - \text{enat } k \neq \text{enat } 0 \wedge$ 
 $g \ ((\text{ndropn } (\text{Suc } k) \ \sigma)) \wedge$ 
 $(\forall j < k. \text{nlength } \sigma - \text{enat } j \neq \text{enat } 0 \wedge f \ (\text{ndropn } (\text{Suc } j) \ \sigma))$ 
using 3 5 6 71 7 72 by blast
from 8 show ?thesis
by (simp add: next-defs until-d-def ndropn-ndropn)
qed

```

lemma *NextUntilsemb*:

```

assumes ( $\sigma \models (\bigcirc f) \ \mathcal{U} \ (\bigcirc g)$ )
shows ( $\sigma \models \bigcirc(f \ \mathcal{U} \ g)$ )
proof -
have 1:  $\exists k. k \leq \text{nlength } \sigma \wedge 0 < \text{nlength } (\text{ndropn } k \ \sigma) \wedge g \ ((\text{ndropn } (\text{Suc } k) \ \sigma)) \wedge$ 
 $(\forall j < k. j < \text{nlength } \sigma \wedge f \ ((\text{ndropn } (\text{Suc } j) \ \sigma)))$ 
using assms
by (auto simp add: next-defs until-d-def ndropn-ndropn)
(metis is-NNil-ndropn le-zero-eq ndropn-0 ndropn-nlength not-less zero-enat-def)
obtain k where 2:  $k \leq \text{nlength } \sigma \wedge 0 < \text{nlength } (\text{ndropn } k \ \sigma) \wedge$ 
 $g \ ((\text{ndropn } (\text{Suc } k) \ \sigma)) \wedge$ 
 $(\forall j < k. j < \text{nlength } \sigma \wedge f \ ((\text{ndropn } (\text{Suc } j) \ \sigma)))$ 
using 1 by auto
have 3:  $0 < \text{nlength } \sigma$ 
using 2 by auto
have 4:  $k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \ \sigma)$ 
by auto
(metis 2 3 add commute add.right-neutral enat-min illess-Suc-eq ndropn-0
    ndropn-Suc-conv-ndropn ndropn-nlength nlength-NCons zero-enat-def)
have 5:  $g \ ((\text{ndropn } (\text{Suc } k) \ \sigma))$ 
using 2 by auto
have 6: ( $\forall j < k. j < \text{nlength } \sigma \wedge f \ ((\text{ndropn } (\text{Suc } j) \ \sigma))$ )
using 2 by blast
have 7:  $0 < \text{nlength } \sigma \wedge$ 
 $(\exists k. k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \ \sigma) \wedge g \ ((\text{ndropn } (\text{Suc } k) \ \sigma)) \wedge$ 
 $(\forall j < k. f \ ((\text{ndropn } (\text{Suc } j) \ \sigma)))$ )
using 2 3 4 by blast
from 7 show ?thesis by (auto simp: next-defs until-d-def zero-enat-def ndropn-ndropn)
qed

```

lemma *NextUntilsem*:

$\sigma \models \circ(f \mathcal{U} g) = (\circ f) \mathcal{U} (\circ g)$
using *NextUntilsema NextUntilsemb* **using** *unl-lift2* **by** *blast*

lemma *NextUntil*:
 $\vdash \circ(f \mathcal{U} g) = (\circ f) \mathcal{U} (\circ g)$
using *NextUntilsem Valid-def* **by** *blast*

UntilNextUntil

lemma *UntilNextUntilsema*:
assumes $0 < \text{nlength } \sigma \wedge$
 $(\exists k. 0 < k \wedge k \leq \text{nlength } \sigma \wedge g ((\text{ndropn } k \ \sigma)) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f ((\text{ndropn } j \ \sigma))))$
shows $(\sigma) \models \circ (f \mathcal{U} g)$
proof –
have 1: $0 < \text{nlength } \sigma \wedge$
 $(\exists k. 0 < k \wedge k \leq \text{nlength } \sigma \wedge g ((\text{ndropn } k \ \sigma)) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f ((\text{ndropn } j \ \sigma))))$
using *assms* **by** *auto*
have 3: $(\exists k. 0 < k \wedge k \leq \text{nlength } \sigma \wedge g ((\text{ndropn } k \ \sigma)) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f ((\text{ndropn } j \ \sigma))))$
using 1 **by** *auto*
obtain k **where** 4: $0 < (\text{Suc } k) \wedge (\text{Suc } k) \leq \text{nlength } \sigma \wedge g ((\text{ndropn } (\text{Suc } k) \ \sigma)) \wedge$
 $(\forall j. 0 < j \wedge j < (\text{Suc } k) \longrightarrow f ((\text{ndropn } j \ \sigma)))$
using 3 **by** (*metis Suc-pred*)
have 5: $k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \ \sigma)$
by (*metis 4 One-nat-def add commute co.enat.sel(2) eSuc-enat epred-conv-minus le-cases min-absorb1*
ndropn-nlength ntaken-all ntaken-ndropn-swap-nlength ntaken-nlength one-enat-def plus-1-eSuc(2)
plus-1-eq-Suc)
have 6: $g ((\text{ndropn } (\text{Suc } k) \ \sigma))$
using 4 **by** *auto*
have 7: $(\forall j < k. f ((\text{ndropn } (\text{Suc } j) \ \sigma)))$
using 4 **by** *blast*
have 8: $(\exists k. k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \ \sigma) \wedge g ((\text{ndropn } (\text{Suc } k) \ \sigma)) \wedge (\forall j < k. f ((\text{ndropn } (\text{Suc } j) \ \sigma))))$
using 4 5 **by** *blast*
have 9: $0 < \text{nlength } \sigma \wedge$
 $(\exists k. k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \ \sigma) \wedge g ((\text{ndropn } (\text{Suc } k) \ \sigma)) \wedge (\forall j < k. f ((\text{ndropn } (\text{Suc } j) \ \sigma))))$
using 1 8 **by** *blast*
from 9 **show** *?thesis* **by** (*auto simp add: next-defs until-d-def zero-enat-def ndropn-ndropn*)
qed

lemma *UntilNextUntilsemb*:
assumes $\sigma \models \circ (f \mathcal{U} g)$
shows $0 < \text{nlength } \sigma \wedge$
 $(\exists k. 0 < k \wedge k \leq \text{nlength } \sigma \wedge g ((\text{ndropn } k \ \sigma)) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f ((\text{ndropn } j \ \sigma))))$
proof –
have 1: $0 < \text{nlength } \sigma \wedge$
 $(\exists k. k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \ \sigma) \wedge g ((\text{ndropn } (\text{Suc } k) \ \sigma)) \wedge (\forall j < k. f ((\text{ndropn } (\text{Suc } j) \ \sigma))))$
using *assms* **by** (*auto simp add: next-defs until-d-def ndropn-ndropn*) (*simp add: zero-enat-def*)
have 2: $(\exists k. k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \ \sigma) \wedge g ((\text{ndropn } (\text{Suc } k) \ \sigma)) \wedge (\forall j < k. f ((\text{ndropn } (\text{Suc } j) \ \sigma))))$
using 1 **by** *auto*
obtain k **where** 3: $k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \ \sigma) \wedge g ((\text{ndropn } (\text{Suc } k) \ \sigma)) \wedge$
 $(\forall j < k. f ((\text{ndropn } (\text{Suc } j) \ \sigma)))$

```

using 2 by auto
have 4: 0 < (Suc k)
by simp
have 5: g ( (ndropn (Suc k) σ))
using 3 by auto
have 6: (Suc k) ≤ nlength σ
using 3 by auto
(metis 1 dual-order.eq-iff eSuc-enat is-NNil-ndropn le-cases ndropn-0 ndropn-Suc-conv-ndropn
ndropn-ndropn ndropn-nlength nlength-NCons plus-1-eq-Suc zero-enat-def)
have 7: (∀ j. 0 < j ∧ j < (Suc k) → f ( (ndropn j σ)))
using 3 less-Suc-eq-0-disj by auto
have 8: (∃ k. 0 < k ∧ k ≤ nlength σ ∧ g ((ndropn k σ)) ∧ (∀ j. 0 < j ∧ j < k → f ((ndropn j σ))))
using 3 6 7 by blast
show ?thesis using 1 8 by blast
qed

```

lemma *UntilNextUntilsem*:

```

( σ ⊨ ○ ( f U g ) ) =
( 0 < nlength σ ∧
  (∃ k. 0 < k ∧ k ≤ nlength σ ∧ g ((ndropn k σ)) ∧ (∀ j. 0 < j ∧ j < k → f ((ndropn j σ)))) )
using UntilNextUntilsema[of σ g f] UntilNextUntilsemb[of f g σ] by meson

```

lemma *UntilNextUntilsem1*:

```

(σ ⊨ f U g) = (σ ⊨ (g ∨ (f ∧ ○(f U g))))

```

unfolding *UntilNextUntilsem*

proof

```

assume a: (σ ⊨ f U g)
show (σ ⊨ g ∨
      f ∧
      (λσ. 0 < nlength σ ∧
        (∃ k. 0 < k ∧ k ≤ nlength σ ∧ g((ndropn k σ)) ∧ (∀ j. 0 < j ∧ j < k → f((ndropn j σ))))
      ))
using a by (simp add: until-d-def) (metis enat-0-iff(2) i0-less ndropn-0 neq0-conv not-le)
next
next
assume b: (σ ⊨ g ∨
      f ∧
      (λσ. 0 < nlength σ ∧
        (∃ k. 0 < k ∧ k ≤ nlength σ ∧ g((ndropn k σ)) ∧ (∀ j. 0 < j ∧ j < k → f((ndropn j σ))))
      ))
show (σ ⊨ f U g)
using b by (simp add: until-d-def) (metis i0-lb linorder-cases ndropn-0 not-less-zero zero-enat-def)
qed

```

lemma *UntilNextUntil*:

```

⊢ f U g = (g ∨ (f ∧ ○(f U g)))

```

by (simp add: *UntilNextUntilsem1 Valid-def*)

NotUntilFalse

lemma *NotUntilFalse*:

$\vdash \neg (f \mathcal{U} \#False)$
by (*simp add: intI until-d-def*)

UntilOrDist

lemma *UntilOrDistsem*:
 $\sigma \models f \mathcal{U} (g \vee h) = (f \mathcal{U} g \vee f \mathcal{U} h)$
by (*auto simp add: until-d-def*)

lemma *UntilOrDist*:
 $\vdash f \mathcal{U} (g \vee h) = (f \mathcal{U} g \vee f \mathcal{U} h)$
using *UntilOrDistsem Valid-def* **by** *blast*

UntilRightDistOr

lemma *UntilRightDistOr*:
 $\vdash f \mathcal{U} h \vee g \mathcal{U} h \longrightarrow (f \vee g) \mathcal{U} h$
by (*auto simp add: Valid-def until-d-def*)

UntilLeftDistAnd

lemma *UntilLeftDistAnd*:
 $\vdash f \mathcal{U} (g \wedge h) \longrightarrow f \mathcal{U} g \wedge f \mathcal{U} h$
by (*auto simp add: Valid-def until-d-def*)

UntilAndDist

lemma *UntilAndDistsem*:
 $\sigma \models (f \wedge g) \mathcal{U} h = ((f \mathcal{U} h) \wedge (g \mathcal{U} h))$
by (*auto simp add: until-d-def*)
(metis (no-types, lifting) less-le-trans not-less-iff-gr-or-eq order.order-iff-strict)

lemma *UntilAndDist*:
 $\vdash (f \wedge g) \mathcal{U} h = ((f \mathcal{U} h) \wedge (g \mathcal{U} h))$
using *UntilAndDistsem Valid-def* **by** *blast*

untilNotImp

lemma *UntilNotImp*:
 $\vdash f \mathcal{U} g \wedge (\neg g) \mathcal{U} h \longrightarrow f \mathcal{U} h$
by (*simp add: Valid-def until-d-def*)
(metis not-less-iff-gr-or-eq order.strict-trans)

UntilUntil

lemma *UntilUntilsem*:
 $(\sigma \models f \mathcal{U} g) = (\sigma \models f \mathcal{U} (f \mathcal{U} g))$
proof *auto*
show $\sigma \models (f \mathcal{U} g) \implies \sigma \models (f \mathcal{U} (f \mathcal{U} g))$
by (*simp add: until-d-def*)
(metis enat-add-sub-same enat-le-plus-same(1) enat-ord-code(4) enat-ord-simps(4) gen-nlength-def ndropn-0 nlength-code not-less-zero)

```

show  $\sigma \models (f \mathcal{U} (f \mathcal{U} g)) \implies \sigma \models (f \mathcal{U} g)$ 
proof –
  assume  $a: \sigma \models (f \mathcal{U} (f \mathcal{U} g))$ 
  show  $\sigma \models (f \mathcal{U} g)$ 
  proof –
    have  $1: \exists k. \text{enat } k \leq \text{nlength } \sigma \wedge$ 
       $(\exists ka. \text{enat } ka \leq \text{nlength } (\text{ndropn } k \sigma) \wedge g (\text{ndropn } ka (\text{ndropn } k \sigma)) \wedge$ 
       $(\forall j < ka. f (\text{ndropn } j (\text{ndropn } k \sigma)))) \wedge$ 
       $(\forall j < k. f (\text{ndropn } j \sigma))$ 
    using a unfolding until-d-def by blast
    obtain  $k$  where  $2:$ 
       $\text{enat } k \leq \text{nlength } \sigma \wedge$ 
       $(\exists ka. \text{enat } ka \leq \text{nlength } (\text{ndropn } k \sigma) \wedge g (\text{ndropn } ka (\text{ndropn } k \sigma)) \wedge$ 
       $(\forall j < ka. f (\text{ndropn } j (\text{ndropn } k \sigma)))) \wedge$ 
       $(\forall j < k. f (\text{ndropn } j \sigma))$ 
    using 1 by auto
    have  $3: (\exists ka. \text{enat } ka \leq \text{nlength } (\text{ndropn } k \sigma) \wedge g (\text{ndropn } ka (\text{ndropn } k \sigma)) \wedge$ 
       $(\forall j < ka. f (\text{ndropn } j (\text{ndropn } k \sigma))))$ 
    using 2 by auto
    obtain  $ka$  where  $4:$ 
       $\text{enat } ka \leq \text{nlength } (\text{ndropn } k \sigma) \wedge g (\text{ndropn } ka (\text{ndropn } k \sigma)) \wedge$ 
       $(\forall j < ka. f (\text{ndropn } j (\text{ndropn } k \sigma)))$ 
    using 3 by auto
    have  $41: \text{enat } ka \leq \text{nlength } \sigma - (\text{enat } k)$ 
    using 4 by auto
    have  $5: \text{enat } (ka+k) \leq \text{nlength } \sigma$ 
    using 2 41 by auto
       $(\text{metis add.commute antisym-conv2 enat.simps(3) enat-add-sub-same enat-min le-iff-add}$ 
       $\text{less-imp-le order-refl plus-enat-simps(1)})$ 
    have  $6: g (\text{ndropn } (ka+k) \sigma)$ 
    by  $(\text{metis } 4 \text{ add.commute ndropn-ndropn})$ 
    have  $7: (\forall j < (ka+k). f (\text{ndropn } j \sigma))$ 
    by  $(\text{metis } 2 \ 4 \text{ add-diff-inverse-nat less-diff-conv2 linorder-not-less ndropn-ndropn})$ 
    have  $8: \exists k. k \leq \text{nlength } \sigma \wedge g (\text{ndropn } k \sigma) \wedge (\forall j < k. f (\text{ndropn } j \sigma))$ 
    using 5 6 7 by blast
    show ?thesis unfolding until-d-def by  $(\text{simp add: } 8)$ 
  qed
qed
qed

```

lemma *UntilUntil:*

$\vdash f \mathcal{U} g = f \mathcal{U} (f \mathcal{U} g)$

using *UntilUntilsem by fastforce*

UntilRightor

lemma *UntilRightOr:*

$\vdash f \mathcal{U} (g \mathcal{U} h) \longrightarrow (f \vee g) \mathcal{U} h$

proof $(\text{auto simp add: Valid-def until-d-def ndropn-ndropn})$

fix $w :: 'a \text{ nelist}$

```

fix k
fix ka
assume a0: enat k ≤ nlength w
assume a1: ∀ j < k. f (ndropn j w)
assume a2: enat ka ≤ nlength w - enat k
assume a3: h (ndropn (k + ka) w)
assume a4: ∀ j < ka. g (ndropn (k + j) w)
show ∃ k. enat k ≤ nlength w ∧ h (ndropn k w) ∧ (∀ j < k. f (ndropn j w) ∨ g (ndropn j w))
proof -
  have 1: ka + k ≤ nlength w
    by (metis a0 a2 add.commute dual-order.order-iff-strict enat.simps(3) enat-add-sub-same
      enat-less-enat-plusI2 less-eqE plus-enat-simps(1))
  have 2: h (ndropn (ka + k) w)
    using a3 by (simp add: add.commute)
  have 3: (∀ j < (ka + k). f (ndropn j w) ∨ g (ndropn j w))
    by (metis a1 a4 add-diff-inverse-nat less-diff-conv2 not-less)
  show ?thesis using 1 2 3 by blast
qed
qed

```

UntilRightAnd

```

lemma UntilRightAndsem:
assumes (σ ⊨ f U (g ∧ h))
shows (σ ⊨ (f U g) U h)
proof -
  have 1: ∃ k. k ≤ nlength σ ∧ g ((ndropn k σ)) ∧ h ((ndropn k σ)) ∧ (∀ j < k. f ((ndropn j σ)))
    using assms by (simp add: until-d-def)
  obtain k where 2: k ≤ nlength σ ∧ g ((ndropn k σ)) ∧ h ((ndropn k σ)) ∧ (∀ j < k. f ((ndropn j σ)))
    using 1 by auto
  have 3: h ((ndropn k σ))
    using 2 by auto
  have 4: k ≤ nlength σ
    using 2 by auto
  have 5: (∀ j < k.
    ∃ ka. ka ≤ nlength (ndropn j σ) ∧ g ((ndropn (ka + j) σ)) ∧ (∀ ja < ka. f ((ndropn (ja + j) σ))))
  proof auto
    fix j
    assume a0: j < k
    show ∃ ka. enat ka ≤ nlength σ - enat j ∧ g (ndropn (ka + j) σ) ∧ (∀ ja < ka. f (ndropn (ja + j) σ))
    proof -
      have 51: k - j ≤ nlength (ndropn j σ)
        using 4 a0
      by (metis enat-minus-mono1 idiff-enat-enat ndropn-nlength)
      have 52: g ((ndropn ((k - j) + j) σ))
        by (simp add: 2 a0 less-imp-le-nat)
      have 53: (∀ ja < (k - j). f ((ndropn (ja + j) σ)))
        using 2 less-diff-conv by blast
      show ?thesis
        using 51 52 53 by auto
    qed
  qed

```

```

    qed
  qed
  have 6:  $\exists k. k \leq \text{nlength } \sigma \wedge$ 
     $h \ ( \text{ndropn } k \ \sigma ) \wedge$ 
     $(\forall j < k. \exists k' \leq \text{nlength } (\text{ndropn } j \ \sigma). g \ ((\text{ndropn } (k + j) \ \sigma)) \wedge (\forall ja < k. f \ ((\text{ndropn } (ja + j) \ \sigma))))$ 
  using 2 5 by blast
  from 6 show ?thesis by (simp add: until-d-def ndropn-ndropn add commute)
  qed

```

```

lemma UntilRightAnd:
 $\vdash f \ \mathcal{U} \ (g \wedge h) \longrightarrow (f \ \mathcal{U} \ g) \ \mathcal{U} \ h$ 
using UntilRightAndsem Valid-def by auto

```

DiamondEqvTrueUntil

```

lemma DiamondEqvTrueUntil:
 $\vdash \Diamond f = \# \text{True } \mathcal{U} \ f$ 
by (simp add: Valid-def sometimes-defs until-d-def)

```

TrueUntillImpNotUntil

```

lemma nellist-ndropn-first-upto:
assumes  $(\exists i \leq k. f \ ( \text{ndropn } i \ xs ))$ 
shows  $(\exists i \leq k. f \ ( \text{ndropn } i \ xs ) \wedge (\forall j < i. \neg (f \ ( \text{ndropn } j \ xs ))))$ 
using assms
proof (induct k arbitrary: xs)
case 0
then show ?case by simp
next
case (Suc k)
then show ?case
by (metis le-Suc-eq less-Suc-eq-le)
qed

```

```

lemma nellist-ndropn-first:
assumes  $(\exists i \leq \text{nlength } xs. f \ ( \text{ndropn } i \ xs ))$ 
shows  $(\exists i \leq \text{nlength } xs. f \ ( \text{ndropn } i \ xs ) \wedge (\forall j. j < i \longrightarrow \neg (f \ ( \text{ndropn } j \ xs ))))$ 
proof (cases nfinite xs)
case True
then show ?thesis using assms nellist-ndropn-first-upto[of - f xs] nfinite-nlength-enat[of xs]
  by force
next
case False
then show ?thesis using assms nellist-ndropn-first-upto[of - f xs]
proof -
assume a1:  $\bigwedge k. \exists i \leq k. f \ ( \text{ndropn } i \ xs ) \implies \exists i \leq k. f \ ( \text{ndropn } i \ xs ) \wedge (\forall j < i. \neg f \ ( \text{ndropn } j \ xs ))$ 
obtain nn :: nat where
f2:  $f \ ( \text{ndropn } nn \ xs ) \wedge \text{enat } nn \leq \text{nlength } xs$ 
using assms by blast
then have  $\forall e. \neg e \leq \text{enat } nn \vee e \leq \text{nlength } xs$ 
by force

```


then show *?thesis*
 using *f2 a1* by (meson enat-ord-simps(1) less-imp-le-nat)
 qed
 qed

lemma *NotSuffixFirstfinite*:
 assumes $(\exists n \leq \text{nlength } xs. \neg f (\text{ndropn } n \text{ } xs))$
 shows $(\exists n \leq \text{nlength } xs. \neg f (\text{ndropn } n \text{ } xs)) \wedge (\forall k. k < n \longrightarrow f (\text{ndropn } k \text{ } xs))$
 using *assms nellist-ndropn-first[of xs LIFT($\neg f$)]* by auto

lemma *TrueUntilImpNotUntilsem*:
 assumes $\sigma \models \# \text{True } \mathcal{U} \ g$
 shows $\sigma \models (\neg g) \mathcal{U} \ g$
 using *assms*
 by (simp add: until-d-def nellist-ndropn-first)

lemma *TrueUntilImpNotUntil*:
 $\vdash \# \text{True } \mathcal{U} \ g \longrightarrow (\neg g) \mathcal{U} \ g$
 by (simp add: intI nellist-ndropn-first until-d-def)

WaitNotDistUntil

lemma *WaitNotDistUntilsem1*:
 assumes $(\sigma \models \neg(f \ \mathcal{W} \ g))$
 shows $(\sigma \models ((\neg g) \mathcal{U} ((\neg f) \wedge (\neg g))))$
 proof –
 have 1: $(\forall k. g (\text{ndropn } k \text{ } \sigma)) \longrightarrow k \leq \text{nlength } \sigma \longrightarrow (\exists j < k. \neg f (\text{ndropn } j \text{ } \sigma)) \wedge (\exists n \leq \text{nlength } \sigma. \neg f (\text{ndropn } n \text{ } \sigma))$
 using *assms* by (simp add: wait-d-def until-d-def always-defs)
 have 2: $(\forall k. k \leq \text{nlength } \sigma \longrightarrow \neg g (\text{ndropn } k \text{ } \sigma)) \vee (\exists j < k. \neg f (\text{ndropn } j \text{ } \sigma))$
 using 1 by auto
 have 3: $(\exists n \leq \text{nlength } \sigma. \neg f (\text{ndropn } n \text{ } \sigma))$
 using 1 by auto
 obtain *n* where 4: $n \leq \text{nlength } \sigma \wedge \neg f (\text{ndropn } n \text{ } \sigma) \wedge (\forall k < n. f (\text{ndropn } k \text{ } \sigma))$
 using 3 using *NotSuffixFirstfinite* by blast
 have 16: $n \leq \text{nlength } \sigma$
 by (simp add: 4)
 have 17: $\neg g (\text{ndropn } n \text{ } \sigma)$
 using 1 4 by blast
 have 18: $(\forall j < n. \neg g (\text{ndropn } j \text{ } \sigma))$
 by (metis 2 4 dual-order.strict-iff-order dual-order.strict-trans1 enat-ord-simps(2))
 have 19: $\exists k \leq \text{nlength } \sigma. \neg f (\text{ndropn } k \text{ } \sigma) \wedge \neg g (\text{ndropn } k \text{ } \sigma) \wedge (\forall j < k. \neg g (\text{ndropn } j \text{ } \sigma))$
 using 16 17 18 4 by blast
 have 20: $(\sigma \models ((\neg g) \mathcal{U} (\neg f \wedge \neg g)))$
 using 19 by (simp add: until-d-def)
 show *?thesis* using 20 by auto
 qed

lemma *WaitNotDistUntilsem2*:

assumes $(\sigma \models ((\neg g) \mathcal{U} ((\neg f) \wedge (\neg g))))$
shows $(\sigma \models \neg(f \mathcal{W} g))$
using *assms not-less-iff-gr-or-eq* **by** (*auto simp add: always-defs wait-d-def until-d-def*)

lemma *WaitNotDistUntilsem*:
 $(\sigma \models (\neg(f \mathcal{W} g)) = ((\neg g) \mathcal{U} ((\neg f) \wedge (\neg g))))$
using *WaitNotDistUntilsem1 WaitNotDistUntilsem2*
unl-lift2 **by** *blast*

lemma *WaitNotDistUntil*:
 $\vdash (\neg(f \mathcal{W} g)) = ((\neg g) \mathcal{U} (\neg f \wedge \neg g))$
using *WaitNotDistUntilsem Valid-def* **by** (*metis*)

UntilInduction

lemma *LFPUntilsem1*:
assumes $\forall n \leq \text{nlength } \sigma.$
 $(g \text{ (ndropn } n \text{ } \sigma)) \longrightarrow h \text{ (ndropn } n \text{ } \sigma)) \wedge$
 $(f \text{ (ndropn } n \text{ } \sigma)) \wedge n < \text{nlength } \sigma \wedge h \text{ (ndropn (Suc } n \text{) } \sigma)) \longrightarrow$
 $h \text{ (ndropn } n \text{ } \sigma))$
 $k \leq \text{nlength } \sigma$
 $g \text{ (ndropn } k \text{ } \sigma)$
 $\forall j < k. f \text{ (ndropn } j \text{ } \sigma)$
shows $h \text{ (} \sigma \text{)}$
using *assms*
proof (*induct k arbitrary:* σ *)*
case 0
then show *?case* **by** *auto*
next
case (*Suc k*)
then show *?case*
proof –
have 1: $g \text{ (ndropn (Suc } k \text{) } \sigma) \longrightarrow h \text{ (ndropn (Suc } k \text{) } \sigma)$
using *Suc.prem1 Suc.prem2* **by** *blast*
have 2: $(f \text{ (ndropn } k \text{ } \sigma)) \wedge k < \text{nlength } \sigma \wedge h \text{ (ndropn (Suc } k \text{) } \sigma)) \longrightarrow$
 $h \text{ (ndropn } k \text{ } \sigma))$
by (*simp add: Suc.prem1 less-le-not-le*)
have 3: $k < \text{nlength } \sigma$
using *Suc.prem2 Suc-ile-eq* **by** *auto*
have 4: $f \text{ (ndropn } k \text{ } \sigma)$
by (*simp add: Suc.prem4*)
have 5: $h \text{ (ndropn } k \text{ } \sigma)$
using 1 2 3 4 *Suc.prem3* **by** *blast*
have 6: $h \text{ (ndropn } 0 \text{ } \sigma)$
using *zero-induct[of* $\lambda k. h \text{ (ndropn } k \text{ } \sigma) k$ *]*
3 5 Suc.prem nellist-ndropn-first[of σh *]*
by (*metis Suc-ile-eq less-Suc-eq-0-disj less-imp-le not-less-eq*)
show *?thesis*
using 6 **by** *auto*
qed

qed

lemma *LFPUntilsem*:

$\sigma \models \Box((g \vee (f \wedge \bigcirc h)) \longrightarrow h) \longrightarrow (f \mathcal{U} g \longrightarrow h)$

using *LFPUntilsem1*[of σ g h f]

by (*auto simp add: always-defs next-defs until-d-def ndropn-ndropn*)

(*metis canonically-ordered-monoid-add-class.lessE enat.simps(3) enat-add-sub-same zero-enat-def*)

lemma *LFPUntil*:

$\vdash \Box((g \vee (f \wedge \bigcirc h)) \longrightarrow h) \longrightarrow (f \mathcal{U} g \longrightarrow h)$

using *LFPUntilsem Valid-def*

by (*metis*)

lemma *UntilInduction-a*:

$\vdash \Box(f \longrightarrow ((\bigcirc f) \wedge g) \vee h) \longrightarrow (f \longrightarrow \Box g \vee g \mathcal{U} h)$

proof –

have 1: $\vdash (\Box g \vee g \mathcal{U} h) = g \mathcal{W} h$

by (*auto simp add: wait-d-def*)

have 2: $\vdash (f \longrightarrow \Box g \vee g \mathcal{U} h) = ((\neg h) \mathcal{U} (\neg g \wedge \neg h) \longrightarrow \neg f)$

using 1 *WaitNotDistUntil* **by** *fastforce*

have 3: $\vdash \Box(((\neg g \wedge \neg h) \vee (\neg h \wedge \bigcirc(\neg f))) \longrightarrow \neg f) \longrightarrow ((\neg h) \mathcal{U} (\neg g \wedge \neg h) \longrightarrow \neg f)$

using *LFPUntil* **by** *blast*

have 4: $\vdash (f \longrightarrow ((\bigcirc f) \wedge g) \vee h) \longrightarrow (((\neg g \wedge \neg h) \vee (\neg h \wedge \bigcirc(\neg f))) \longrightarrow \neg f)$

using *NextImpNotNextNot*[of f] **by** *auto*

have 5: $\vdash \Box(f \longrightarrow ((\bigcirc f) \wedge g) \vee h) \longrightarrow \Box(((\neg g \wedge \neg h) \vee (\neg h \wedge \bigcirc(\neg f))) \longrightarrow \neg f)$

using 4 **by** (*rule ImpBoxRule*)

show *?thesis*

using 2 3 5 **by** *fastforce*

qed

lemma *UntilInduction-b*:

$\vdash \Box(f \longrightarrow (\bigcirc f) \vee g) \longrightarrow (f \longrightarrow \Box f \vee f \mathcal{U} g)$

proof –

have 1: $\vdash (\Box f \vee f \mathcal{U} g) = f \mathcal{W} g$

by (*auto simp add: wait-d-def*)

have 2: $\vdash (f \longrightarrow \Box f \vee f \mathcal{U} g) = ((\neg g) \mathcal{U} (\neg f \wedge \neg g) \longrightarrow \neg f)$

using 1 *WaitNotDistUntil* **by** *fastforce*

have 3: $\vdash \Box(((\neg f \wedge \neg g) \vee (\neg g \wedge \bigcirc(\neg f))) \longrightarrow \neg f) \longrightarrow ((\neg g) \mathcal{U} (\neg f \wedge \neg g) \longrightarrow \neg f)$

using *LFPUntil* **by** *blast*

have 4: $\vdash (f \longrightarrow (\bigcirc f) \vee g) \longrightarrow (((\neg f \wedge \neg g) \vee (\neg g \wedge \bigcirc(\neg f))) \longrightarrow \neg f)$

using *NextImpNotNextNot*[of f] **by** *auto*

have 5: $\vdash \Box(f \longrightarrow (\bigcirc f) \vee g) \longrightarrow \Box(((\neg f \wedge \neg g) \vee (\neg g \wedge \bigcirc(\neg f))) \longrightarrow \neg f)$

using 4 *BoxImpBoxRule* **by** *blast*

show *?thesis*

using 2 3 5 **by** *fastforce*

qed

2.8.3 Theorems

lemma *NextFalseSUntil*:

$\vdash \bigcirc g = \#False \mathcal{U}^s g$
proof –
have 1: $\vdash \#False \mathcal{U} g = g$
using *UntilNextUntil*[of *LIFT*($\#False$) *g*] **by** *auto*
show *?thesis unfolding suntill-d-def using 1 inteq-reflection by force*
qed

lemma *WNextUntil*:
 $\vdash \text{wnext}(f \mathcal{U} g) = (\text{empty} \vee (\bigcirc f) \mathcal{U} (\bigcirc g))$
by (*meson NextUntil Prop06 WnextEqvEmptyOrNext*)

lemma *UntilRelease*:
 $\vdash f \mathcal{R} g = (\neg (\neg f) \mathcal{U} (\neg g))$
by (*simp add: release-d-def*)

lemma *SReleaseWait*:
 $\vdash f \mathcal{M} g = (\neg (\neg f) \mathcal{W} (\neg g))$
by (*simp add: srelease-d-def*)

lemma *ReleaseUntil*:
 $\vdash f \mathcal{U} g = (\neg (\neg f) \mathcal{R} (\neg g))$
by (*simp add: release-d-def*)

lemma *WaitSRelease*:
 $\vdash f \mathcal{W} g = (\neg (\neg f) \mathcal{M} (\neg g))$
by (*simp add: srelease-d-def*)

lemma *NotUntilRelease*:
 $\vdash \neg(f \mathcal{U} g) = (\neg f) \mathcal{R} (\neg g)$
by (*simp add: ReleaseUntil*)

lemma *NotWaitSRelease*:
 $\vdash \neg(f \mathcal{W} g) = (\neg f) \mathcal{M} (\neg g)$
by (*simp add: WaitSRelease*)

lemma *NotReleaseUntil*:
 $\vdash \neg(f \mathcal{R} g) = (\neg f) \mathcal{U} (\neg g)$
by (*simp add: UntilRelease*)

lemma *NotSReleaseWait*:
 $\vdash \neg(f \mathcal{M} g) = (\neg f) \mathcal{W} (\neg g)$
by (*simp add: SReleaseWait*)

lemma *BoxEqvFalseRelease*:
 $\vdash \Box f = \#False \mathcal{R} f$
unfolding *release-d-def*
by (*metis DiamondEqvTrueUntil Prop11 always-d-def int-simps(3) inteq-reflection lift-imp-neg*)

lemma *UntilTrue*:
 $\vdash f \mathcal{U} \#True$

using *UntilNextUntil* **by** *fastforce*

lemma *UntilIdempotent*:

$\vdash f \mathcal{U} f = f$

using *UntilNextUntil*[*of f f*] **by** *auto*

lemma *UntilImpUntil*:

assumes $\vdash f0 \longrightarrow f1$

$\vdash g0 \longrightarrow g1$

shows $\vdash f0 \mathcal{U} g0 \longrightarrow f1 \mathcal{U} g1$

using *assms*

by (*metis Prop10 Prop12 UntilAndDist UntilLeftDistAnd int-eq*)

lemma *UntilEqvUntil*:

assumes $\vdash f0 = f1$

$\vdash g0 = g1$

shows $\vdash f0 \mathcal{U} g0 = f1 \mathcal{U} g1$

proof –

have 1: $\vdash f0 \longrightarrow f1$

using *assms* **by** *auto*

have 2: $\vdash g0 \longrightarrow g1$

using *assms* **by** *auto*

have 3: $\vdash f0 \mathcal{U} g0 \longrightarrow f1 \mathcal{U} g1$

using 1 2 *UntilImpUntil*[*of f0 f1 g0 g1*] **by** *auto*

have 4: $\vdash f1 \longrightarrow f0$

using *assms* **by** *auto*

have 5: $\vdash g1 \longrightarrow g0$

using *assms* **by** *auto*

have 6: $\vdash f1 \mathcal{U} g1 \longrightarrow f0 \mathcal{U} g0$

using 4 5 *UntilImpUntil*[*of f1 f0 g1 g0*] **by** *auto*

from 3 6 **show** *?thesis* **by** *fastforce*

qed

lemma *UntilRightDistImp*:

$\vdash (f \longrightarrow g) \mathcal{U} h \longrightarrow (f \mathcal{U} h \longrightarrow g \mathcal{U} h)$

proof –

have 1: $\vdash (f \longrightarrow g) \mathcal{U} h \longrightarrow (f \mathcal{U} h \longrightarrow g \mathcal{U} h) =$

$((f \longrightarrow g) \mathcal{U} h \wedge f \mathcal{U} h \longrightarrow g \mathcal{U} h)$

by *auto*

have 2: $\vdash ((f \longrightarrow g) \mathcal{U} h \wedge f \mathcal{U} h) = ((f \longrightarrow g) \wedge f) \mathcal{U} h$

by (*simp add: UntilAndDist int-iffD1 int-iffD2 int-iffI*)

have 3: $\vdash ((f \longrightarrow g) \wedge f) = (f \wedge g)$

by *auto*

have 4: $\vdash h = h$

by *auto*

have 5: $\vdash ((f \longrightarrow g) \wedge f) \mathcal{U} h = (f \wedge g) \mathcal{U} h$

using 3 4 **using** *UntilEqvUntil* **by** *blast*

have 6: $\vdash (f \wedge g) \mathcal{U} h = (f \mathcal{U} h \wedge g \mathcal{U} h)$

by (*simp add: UntilAndDist*)

show *?thesis*

using 2 5 6 by fastforce
qed

lemma *FalseUntil*:

$\vdash \#False \mathcal{U} g = g$

by (metis Prop10 Prop12 TrueW UntilNextUntil int-simps(14) int-simps(21) int-simps(25) int-simps(3) inteq-reflection)

lemma *UntilExclMid*:

$\vdash f \mathcal{U} g \vee f \mathcal{U} (\neg g)$

using UntilOrDist UntilTrue by fastforce

lemma *NotUntilImp*:

$\vdash (\neg f) \mathcal{U} (g \mathcal{U} h) \wedge f \mathcal{U} h \longrightarrow g \mathcal{U} h$

proof –

have 1: $\vdash (\neg f) \mathcal{U} (g \mathcal{U} h) \longrightarrow (\neg f \vee g) \mathcal{U} h$

by (simp add: UntilRightOr)

have 2: $\vdash (\neg f \vee g) = (f \longrightarrow g)$

by auto

have 3: $\vdash h = h$

by auto

have 4: $\vdash (\neg f \vee g) \mathcal{U} h = (f \longrightarrow g) \mathcal{U} h$

by (simp add: 2 UntilEqvUntil)

have 5: $\vdash (f \longrightarrow g) \mathcal{U} h \longrightarrow (f \mathcal{U} h \longrightarrow g \mathcal{U} h)$

by (simp add: UntilRightDistImp)

have 6: $\vdash (\neg f) \mathcal{U} (g \mathcal{U} h) \longrightarrow (f \mathcal{U} h \longrightarrow g \mathcal{U} h)$

using 1 4 5 by fastforce

from 6 show ?thesis by auto

qed

lemma *UntilNotImpa*:

$\vdash f \mathcal{U} ((\neg g) \mathcal{U} h) \wedge g \mathcal{U} h \longrightarrow f \mathcal{U} h$

proof –

have 1: $\vdash f \mathcal{U} ((\neg g) \mathcal{U} h) \longrightarrow (f \vee (\neg g)) \mathcal{U} h$

by (simp add: UntilRightOr)

have 2: $\vdash (f \vee (\neg g)) = (g \longrightarrow f)$

by auto

have 3: $\vdash h = h$

by auto

have 4: $\vdash (f \vee (\neg g)) \mathcal{U} h = (g \longrightarrow f) \mathcal{U} h$

by (simp add: 2 UntilEqvUntil)

have 5: $\vdash (g \longrightarrow f) \mathcal{U} h \longrightarrow (g \mathcal{U} h \longrightarrow f \mathcal{U} h)$

by (simp add: UntilRightDistImp)

have 6: $\vdash f \mathcal{U} ((\neg g) \mathcal{U} h) \longrightarrow (g \mathcal{U} h \longrightarrow f \mathcal{U} h)$

using 1 4 5 by fastforce

from 6 show ?thesis by auto

qed

lemma *UntilNotUntilImp*:

$\vdash f \mathcal{U} g \wedge (\neg g) \mathcal{U} f \longrightarrow f$

proof –
have 1: $\vdash f \mathcal{U} g \wedge (\neg g) \mathcal{U} f \longrightarrow f \mathcal{U} f$
using *UntilNotImp* **by** *auto*
have 2: $\vdash f \mathcal{U} f = f$
using *UntilIdempotent* **by** *auto*
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *AndNotUntilImp*:
 $\vdash f \wedge (\neg f) \mathcal{U} g \longrightarrow g$
proof –
have 1: $\vdash f = f \mathcal{U} f$
by (*simp add: UntilIdempotent int-iffD1 int-iffD2 int-iffI*)
have 2: $\vdash g = \#False \mathcal{U} g$
by (*meson FalseUntil Prop11*)
have 3: $\vdash f \mathcal{U} f \wedge (\neg f) \mathcal{U} g \longrightarrow \#False \mathcal{U} g$
by (*metis 1 FalseUntil UntilNotImp inteq-reflection*)
from 1 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *UntilImpOr*:
 $\vdash f \mathcal{U} g \longrightarrow f \vee g$
proof –
have $\vdash f \wedge \bigcirc(f \mathcal{U} g) \longrightarrow f \vee g$
by *force*
then show *?thesis*
using *UntilNextUntil[of f g]* **by** *auto*
qed

lemma *UntilIntro*:
 $\vdash g \longrightarrow f \mathcal{U} g$
proof –
have 1: $\vdash g = \#False \mathcal{U} g$
by (*meson FalseUntil Prop11*)
have 2: $\vdash \#False \longrightarrow f$
by *auto*
have 3: $\vdash g \longrightarrow g$
by *auto*
have 4: $\vdash \#False \mathcal{U} g \longrightarrow f \mathcal{U} g$
by (*simp add: UntilImpUntil*)
from 1 4 **show** *?thesis* **by** *fastforce*
qed

lemma *OrImpUntil*:
 $\vdash f \wedge g \longrightarrow f \mathcal{U} g$
by (*simp add: Prop01 Prop05 UntilIntro*)

lemma *UntilAbsorp-a*:
 $\vdash (f \vee f \mathcal{U} g) = (f \vee g)$
proof –

have 1: $\vdash (f \vee f \mathcal{U} g) \longrightarrow f \vee g$
using *UntilImpOr* **by** *fastforce*
have 2: $\vdash f \vee g \longrightarrow (f \vee f \mathcal{U} g)$
using *UntilIntro* **by** *fastforce*
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *UntilAbsorp-b*:
 $\vdash (f \mathcal{U} g \vee g) = f \mathcal{U} g$
using *UntilNextUntil* **by** *fastforce*

lemma *UntilAbsorp-c*:
 $\vdash (f \mathcal{U} g \wedge g) = g$
using *UntilIntro* **by** *fastforce*

lemma *UntilAbsorp-d*:
 $\vdash (f \mathcal{U} g \vee (f \wedge g)) = f \mathcal{U} g$
using *UntilNextUntil* **by** *fastforce*

lemma *UntilAbsorp-e*:
 $\vdash (f \mathcal{U} g \wedge (f \vee g)) = f \mathcal{U} g$
by (*meson Prop10 Prop11 UntilImpOr*)

lemma *LeftUntilAbsorp*:
 $\vdash f \mathcal{U} (f \mathcal{U} g) = f \mathcal{U} g$
by (*meson Prop11 UntilUntil*)

lemma *RightUntilAbsorp*:
 $\vdash (f \mathcal{U} g) \mathcal{U} g = f \mathcal{U} g$
by (*metis Prop11 UntilAbsorp-b UntilAbsorp-c UntilImpOr UntilRightAnd UntilUntil inteq-reflection*)

lemma *UntilAbsorpAndDiamond*:
 $\vdash (f \mathcal{U} g \wedge \Diamond g) = f \mathcal{U} g$
by (*metis DiamondEqvTrueUntil Prop11 Prop12 UntilIdempotent UntilImpUntil int-simps(12) inteq-reflection*)

lemma *UntilAbsorpOrDiamond*:
 $\vdash (f \mathcal{U} g \vee \Diamond g) = \Diamond g$
using *UntilAbsorpAndDiamond* **by** *fastforce*

lemma *UntilAbsorpDiamond*:
 $\vdash f \mathcal{U} (\Diamond g) = \Diamond g$
using *DiamondDiamondEqvDiamond UntilAbsorpOrDiamond UntilAbsorp-b* **by** *fastforce*

lemma *UntilImpDiamond*:
 $\vdash f \mathcal{U} g \longrightarrow \Diamond g$
using *UntilAbsorpAndDiamond* **by** *fastforce*

lemma *AlwaysImpNotUntilNot*:
 $\vdash \Box f \longrightarrow \neg(g \mathcal{U} (\neg f))$
by (*simp add: UntilImpDiamond always-d-def*)

lemma *UntilAlwaysAndDist*:

$\vdash \Box f \wedge g \mathcal{U} h \longrightarrow (f \wedge g) \mathcal{U} (f \wedge h)$

proof –

have 1: $\vdash \Box(h \vee g \wedge \bigcirc((f \wedge g) \mathcal{U} (f \wedge h))) \longrightarrow (f \wedge g) \mathcal{U} (f \wedge h) \longrightarrow$
 $g \mathcal{U} h \longrightarrow (f \wedge g) \mathcal{U} (f \wedge h)$

using *LFPUntil* **by** *blast*

have 2: $\vdash f \longrightarrow (h \vee g \wedge \bigcirc((f \wedge g) \mathcal{U} (f \wedge h))) \longrightarrow (f \wedge g) \mathcal{U} (f \wedge h)$

using *UntilNextUntil*[of *LIFT*($f \wedge g$) *LIFT*($f \wedge h$)] **by** *auto*

have 3: $\vdash \Box f \longrightarrow \Box(h \vee g \wedge \bigcirc((f \wedge g) \mathcal{U} (f \wedge h))) \longrightarrow (f \wedge g) \mathcal{U} (f \wedge h)$

using 2 *BoxImpBoxRule* **by** *blast*

have 4: $\vdash \Box f \longrightarrow (g \mathcal{U} h \longrightarrow (f \wedge g) \mathcal{U} (f \wedge h))$

using 1 3 *lift-imp-trans* **by** *blast*

show *?thesis* **using** 4 **by** *fastforce*

qed

lemma *UntilAndImp*:

$\vdash \Box f \wedge \Diamond g \longrightarrow f \mathcal{U} g$

proof –

have 1: $\vdash \Diamond g = \#True \mathcal{U} g$

by (*simp add: DiamondEqvTrueUntil*)

have 2: $\vdash \Box f \wedge \#True \mathcal{U} g \longrightarrow (f \wedge \#True) \mathcal{U} (f \wedge g)$

using *UntilAlwaysAndDist* **by** *blast*

have 3: $\vdash (f \wedge \#True) \mathcal{U} (f \wedge g) = f \mathcal{U} (f \wedge g)$

by *simp*

have 4: $\vdash f \mathcal{U} (f \wedge g) \longrightarrow (f \mathcal{U} f) \mathcal{U} g$

by (*simp add: UntilRightAnd*)

have 5: $\vdash (f \mathcal{U} f) = f$

by (*simp add: UntilIdempotent*)

have 6: $\vdash (f \mathcal{U} f) \mathcal{U} g = f \mathcal{U} g$

by (*simp add: 5 UntilEqvUntil*)

show *?thesis*

by (*metis 1 2 3 4 5 inteq-reflection lift-imp-trans*)

qed

lemma *UntilRightMono*:

$\vdash \Box(f \longrightarrow g) \longrightarrow (h \mathcal{U} f \longrightarrow h \mathcal{U} g)$

proof –

have 1: $\vdash \Box(f \longrightarrow g) \wedge h \mathcal{U} f \longrightarrow ((f \longrightarrow g) \wedge h) \mathcal{U} ((f \longrightarrow g) \wedge f)$

using *UntilAlwaysAndDist* **by** *blast*

have 2: $\vdash ((f \longrightarrow g) \wedge h) \mathcal{U} ((f \longrightarrow g) \wedge f) \longrightarrow h \mathcal{U} ((f \longrightarrow g) \wedge f)$

by (*meson Prop12 UntilImpUntil int-iffD2 lift-and-com*)

have 3: $\vdash ((f \longrightarrow g) \wedge f) \longrightarrow g$

by *auto*

have 4: $\vdash h \mathcal{U} ((f \longrightarrow g) \wedge f) \longrightarrow h \mathcal{U} g$

by (*simp add: 3 UntilImpUntil*)

show *?thesis*

by (*meson 1 2 4 Prop09 lift-imp-trans*)

qed

lemma *UntilLeftMono*:

$\vdash \Box (f \longrightarrow g) \longrightarrow (f \mathcal{U} h \longrightarrow g \mathcal{U} h)$

proof –

have 1: $\vdash \Box (f \longrightarrow g) \wedge f \mathcal{U} h \longrightarrow ((f \longrightarrow g) \wedge f) \mathcal{U} ((f \longrightarrow g) \wedge h)$

by (*simp add: UntilAlwaysAndDist*)

have 2: $\vdash ((f \longrightarrow g) \wedge f) \mathcal{U} ((f \longrightarrow g) \wedge h) \longrightarrow ((f \longrightarrow g) \wedge f) \mathcal{U} h$

by (*meson Prop12 UntilLeftDistAnd*)

have 3: $\vdash ((f \longrightarrow g) \wedge f) \longrightarrow g$

by *auto*

have 4: $\vdash ((f \longrightarrow g) \wedge f) \mathcal{U} h \longrightarrow g \mathcal{U} h$

by (*simp add: 3 UntilImpUntil*)

show *?thesis*

by (*meson 1 2 4 Prop09 lift-imp-trans*)

qed

lemma *UntilCatRule*:

$\vdash \Box ((f \longrightarrow g \mathcal{U} h) \wedge (h \longrightarrow g \mathcal{U} i)) \longrightarrow (f \longrightarrow (g \mathcal{U} i))$

proof –

have 1: $\vdash \Box ((f \longrightarrow g \mathcal{U} h) \wedge (h \longrightarrow g \mathcal{U} i)) \longrightarrow \Box (f \longrightarrow g \mathcal{U} h)$

by (*metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)

have 2: $\vdash \Box ((f \longrightarrow g \mathcal{U} h) \wedge (h \longrightarrow g \mathcal{U} i)) \longrightarrow \Box (h \longrightarrow g \mathcal{U} i)$

by (*metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)

have 3: $\vdash \Box (h \longrightarrow g \mathcal{U} i) \longrightarrow \Box (g \mathcal{U} h \longrightarrow g \mathcal{U} (g \mathcal{U} i))$

by (*metis BoxEqvBoxBox BoxImpBoxRule UntilRightMono inteq-reflection*)

have 4: $\vdash \Box (g \mathcal{U} h \longrightarrow g \mathcal{U} (g \mathcal{U} i)) \longrightarrow \Box (g \mathcal{U} h \longrightarrow g \mathcal{U} i)$

by (*metis BoxEqvBoxBox UntilUntil int-iffD1 inteq-reflection*)

have 5: $\vdash \Box (f \longrightarrow g \mathcal{U} h) \longrightarrow (f \longrightarrow g \mathcal{U} h)$

by (*simp add: BoxElim*)

have 6: $\vdash \Box (g \mathcal{U} h \longrightarrow g \mathcal{U} i) \longrightarrow (g \mathcal{U} h \longrightarrow g \mathcal{U} i)$

by (*simp add: BoxElim*)

have 7: $\vdash (f \longrightarrow g \mathcal{U} h) \wedge (g \mathcal{U} h \longrightarrow g \mathcal{U} i) \longrightarrow (f \longrightarrow g \mathcal{U} i)$

by *auto*

have 8: $\vdash \Box ((f \longrightarrow g \mathcal{U} h) \wedge (h \longrightarrow g \mathcal{U} i)) \longrightarrow$

$(f \longrightarrow g \mathcal{U} h) \wedge (g \mathcal{U} h \longrightarrow g \mathcal{U} i)$

using 1 2 3 4 5 6 **by** *fastforce*

from 7 8 **show** *?thesis* **by** *auto*

qed

lemma *UntilStrengthen*:

$\vdash \Box ((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f \mathcal{U} g \longrightarrow h \mathcal{U} i)$

proof –

have 11: $\vdash \Box ((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow \Box (f \longrightarrow h)$

by (*meson BoxImpBoxRule Prop12 int-iffD2 lift-and-com*)

have 1: $\vdash \Box ((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f \mathcal{U} g \longrightarrow h \mathcal{U} g)$

using 11 *UntilLeftMono*[*of f h g*] **by** *fastforce*

have 21: $\vdash \Box ((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow \Box (g \longrightarrow i)$

by (*simp add: BoxImpBoxRule Prop01 Prop05*)

have 2: $\vdash \Box ((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (h \mathcal{U} g \longrightarrow h \mathcal{U} i)$

using 21 *UntilRightMono*[*of g i h*] **by** *fastforce*

have 3: $\vdash \Box ((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f \mathcal{U} g \longrightarrow h \mathcal{U} g) \wedge (h \mathcal{U} g \longrightarrow h \mathcal{U} i)$

using 1 2 by fastforce
 have 4: $\vdash (f \mathcal{U} g \longrightarrow h \mathcal{U} g) \wedge (h \mathcal{U} g \longrightarrow h \mathcal{U} i) \longrightarrow (f \mathcal{U} g \longrightarrow h \mathcal{U} i)$
 by auto
 from 3 4 show ?thesis by auto
 qed

lemma *UntilInduction*:

$\vdash \Box (f \longrightarrow \neg g \wedge \bigcirc f) \longrightarrow (f \longrightarrow \neg(h \mathcal{U} g))$
proof –
 have 1: $\vdash \Box (\neg g) \longrightarrow \neg(h \mathcal{U} g)$
 by (simp add: *UntilImpDiamond always-d-def*)
 have 15: $\vdash (f \longrightarrow \neg g \wedge \bigcirc f) \longrightarrow (g \vee \bigcirc (\neg f) \longrightarrow \neg f)$
 using *NextImpNotNextNot[of f]* by fastforce
 have 16: $\vdash (f \longrightarrow \neg g \wedge \bigcirc f) \longrightarrow (g \vee \#True \wedge \bigcirc (\neg f) \longrightarrow \neg f)$
 using 15 by auto
 have 2: $\vdash \Box (f \longrightarrow \neg g \wedge \bigcirc f) \longrightarrow \Box (g \vee \#True \wedge \bigcirc (\neg f) \longrightarrow \neg f)$
 using 16 *BoxImpBoxRule* by blast
 have 3: $\vdash \Box (f \longrightarrow \neg g \wedge \bigcirc f) \longrightarrow (\#True \mathcal{U} g \longrightarrow \neg f)$
 using 2 *LFPUntil[of g LIFT(#True) LIFT(\neg f)]*
 by fastforce
 have 4: $\vdash (\#True \mathcal{U} g \longrightarrow \neg f) \longrightarrow (f \longrightarrow \neg(\#True \mathcal{U} g))$
 by auto
 have 5: $\vdash \neg(\#True \mathcal{U} g) = \Box (\neg g)$
 using *BoxEqvFalseRelease NotUntilRelease inteq-reflection* by fastforce
 from 5 4 3 1 show ?thesis by fastforce
 qed

lemma *UntilBoxImp*:

$\vdash f \mathcal{U} (\Box g) \longrightarrow \Box(f \mathcal{U} g)$
proof –
 have 1: $\vdash f \mathcal{U} \Box g \longrightarrow f \mathcal{U} g$
 by (meson *BoxElim BoxGen MP UntilRightMono*)
 have 2: $\vdash \text{wnext} (f \mathcal{U} \Box g) = (\text{empty} \vee \bigcirc (f \mathcal{U} \Box g))$
 by (meson *WnextEqvEmptyOrNext*)
 have 3: $\vdash \Box g = (g \wedge \text{wnext} (\Box g))$
 by (metis (no-types) *BoxEqvAndWnextBox*)
 have 4: $\vdash f \mathcal{U} \Box g = (\Box g \vee f \wedge \bigcirc (f \mathcal{U} \Box g))$
 by (meson *UntilNextUntil*)
 have 5: $\vdash g \wedge \text{wnext} (\Box g) \longrightarrow \text{empty} \vee \bigcirc (f \mathcal{U} \Box g)$
 by (metis *NextUntil Prop01 Prop05 Prop08 UntilIntro WnextEqvEmptyOrNext int-iffD1 inteq-reflection*)
 have 6: $\vdash \text{more} \wedge f \mathcal{U} \Box g \longrightarrow \text{empty} \vee \bigcirc (f \mathcal{U} \Box g)$
 using 5 3 4 by fastforce
 have 7: $\vdash \text{more} \wedge f \mathcal{U} \Box g \longrightarrow \bigcirc (f \mathcal{U} \Box g)$
 using 2 6 *WnextAndMoreEqvNext* by fastforce
 from 1 7 show ?thesis using *BoxIntro[of LIFT(f \mathcal{U} (\Box g)) LIFT(f \mathcal{U} g)]*
 by auto
 qed

lemma *UntilBoxEqvBox*:

$\vdash f \mathcal{U} (\Box f) = \Box f$

proof –

have 1: $\vdash f \mathcal{U} (\Box f) \longrightarrow \Box(f \mathcal{U} f)$

using *UntilBoxImp[of f f]* **by** *auto*

have 2: $\vdash \Box(f \mathcal{U} f) = \Box f$

by (*simp add: BoxEqvBox UntilIdempotent*)

have 3: $\vdash \Box f \longrightarrow f \mathcal{U} (\Box f)$

by (*simp add: UntilIntro*)

from 1 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *UntilRightStrengthen*:

$\vdash f \mathcal{U} (g \wedge h) \longrightarrow f \mathcal{U} (g \mathcal{U} h)$

by (*meson BoxGen MP OrImpUntil UntilRightMono*)

lemma *UntilLeftStrengthen*:

$\vdash (f \wedge g) \mathcal{U} h \longrightarrow (f \mathcal{U} g) \mathcal{U} h$

by (*simp add: OrImpUntil UntilImpUntil*)

lemma *UntilLeftAndOrder*:

$\vdash (f \wedge g) \mathcal{U} h \longrightarrow f \mathcal{U} (g \mathcal{U} h)$

by (*metis Prop12 UntilIdempotent UntilImpUntil UntilIntro inteq-reflection*)

lemma *UntilFrameNext*:

$\vdash \Box f \longrightarrow (\bigcirc g \longrightarrow \bigcirc (f \mathcal{U} g))$

by (*simp add: NextImpNext Prop01 Prop05 Prop09 UntilIntro*)

lemma *UntilFrameDiamond*:

$\vdash \Box f \longrightarrow (\Diamond g \longrightarrow \Diamond (f \mathcal{U} g))$

by (*meson NowImpDiamond Prop09 UntilAndImp lift-imp-trans*)

lemma *UntilFrameBox*:

$\vdash \Box f \longrightarrow (\Box g \longrightarrow \Box (f \mathcal{U} g))$

by (*simp add: BoxAndBoxImpBoxRule OrImpUntil Prop09*)

lemma *UntilImpNot*:

$\vdash f \mathcal{U} g \longrightarrow (f \wedge \neg g) \mathcal{U} g$

proof –

have 1: $\vdash f \mathcal{U} g \longrightarrow \Diamond g$

by (*simp add: UntilImpDiamond*)

have 2: $\vdash \Diamond g = \#True \mathcal{U} g$

by (*simp add: DiamondEqvTrueUntil*)

have 3: $\vdash \#True \mathcal{U} g \longrightarrow (\neg g) \mathcal{U} g$

by (*simp add: TrueUntilImpNotUntil*)

have 4: $\vdash (f \mathcal{U} g \wedge (\neg g) \mathcal{U} g) = (f \wedge \neg g) \mathcal{U} g$

using *UntilAndDist* **by** *fastforce*

show *?thesis*

by (*meson 1 2 3 4 Prop10 int-iffD1 lift-imp-trans*)

qed

lemma *UntilAndRule*:

$\vdash f \mathcal{U} g = (f \wedge \neg g) \mathcal{U} g$

proof –

have 1: $\vdash (f \wedge \neg g) \mathcal{U} g \longrightarrow f \mathcal{U} g$

using *UntilAndDist* **by** *fastforce*

show *?thesis* **by** (*simp add: 1 UntilImpNot int-iffI*)

qed

lemma *UntilWait*:

$\vdash f \mathcal{U} g = (f \mathcal{W} g \wedge \Diamond g)$

proof –

have 1: $\vdash f \mathcal{U} g \longrightarrow f \mathcal{W} g \wedge \Diamond g$

by (*simp add: Prop05 Prop12 UntilImpDiamond wait-d-def*)

have 2: $\vdash (f \mathcal{W} g \wedge \Diamond g) = ((\Box f \vee f \mathcal{U} g) \wedge \Diamond g)$

by (*auto simp add: wait-d-def*)

have 3: $\vdash ((\Box f \vee f \mathcal{U} g) \wedge \Diamond g) = ((\Box f \wedge \Diamond g) \vee (f \mathcal{U} g \wedge \Diamond g))$

by *auto*

have 4: $\vdash (\Box f \wedge \Diamond g) \longrightarrow f \mathcal{U} g$

by (*simp add: UntilAndImp*)

have 5: $\vdash (f \mathcal{U} g \wedge \Diamond g) \longrightarrow f \mathcal{U} g$

by *auto*

show *?thesis*

using 1 2 4 **by** *fastforce*

qed

lemma *WaitLeftDistAnd*:

$\vdash f \mathcal{W} (g \wedge h) \longrightarrow f \mathcal{W} g \wedge f \mathcal{W} h$

proof –

have 1: $\vdash f \mathcal{W} (g \wedge h) \longrightarrow f \mathcal{W} g$

unfolding *wait-d-def*

by (*metis Prop08 Prop12 UntilAbsorp-a UntilLeftDistAnd int-iffD1 inteq-reflection*)

have 2: $\vdash f \mathcal{W} (g \wedge h) \longrightarrow f \mathcal{W} h$

unfolding *wait-d-def*

by (*metis Prop08 Prop12 UntilAbsorp-a UntilLeftDistAnd int-iffD1 inteq-reflection*)

show *?thesis* **by** (*simp add: 1 2 Prop12*)

qed

lemma *WaitRightDistAnd*:

$\vdash (f \wedge g) \mathcal{W} h = (f \mathcal{W} h \wedge g \mathcal{W} h)$

proof –

have 1: $\vdash \Box(f \wedge g) = (\Box f \wedge \Box g)$

by (*metis BoxAndBoxEqvBoxRule inteq-reflection lift-and-com*)

have 2: $\vdash (f \wedge g) \mathcal{U} h = (f \mathcal{U} h \wedge g \mathcal{U} h)$

by (*simp add: UntilAndDist*)

have 3: $\vdash ((\Box f \wedge \Box g) \wedge h) \longrightarrow ((\Box f \vee f \mathcal{U} h) \wedge (\Box g \vee g \mathcal{U} h))$

by (*simp add: intI*)

have 4: $\vdash (f \mathcal{U} h \wedge g \mathcal{U} h) \longrightarrow ((\Box f \vee f \mathcal{U} h) \wedge (\Box g \vee g \mathcal{U} h))$

by *auto*

have 5: $\vdash ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h)) \longrightarrow ((\Box f \vee f \mathcal{U} h) \wedge (\Box g \vee g \mathcal{U} h))$

using 3 4 **by** *fastforce*

have 6: $\vdash \Box f \wedge \Box g \longrightarrow ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$
by *auto*
have 7: $\vdash \Box f \wedge g \mathcal{U} h \longrightarrow ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$
by (*metis 2 Prop05 Prop12 UntilAlwaysAndDist UntilLeftDistAnd inteq-reflection lift-imp-trans*)
have 8: $\vdash f \mathcal{U} h \wedge \Box g \longrightarrow ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$
by (*metis Prop05 Prop08 Prop12 UntilAlwaysAndDist UntilAndDist UntilLeftDistAnd inteq-reflection lift-and-com*)
have 9: $\vdash f \mathcal{U} h \wedge g \mathcal{U} h \longrightarrow ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$
by *auto*
have 10: $\vdash ((\Box f \vee f \mathcal{U} h) \wedge (\Box g \vee g \mathcal{U} h)) \longrightarrow ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$
using 6 7 8 9 **by** *fastforce*
have 11: $\vdash ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h)) = ((\Box f \vee f \mathcal{U} h) \wedge (\Box g \vee g \mathcal{U} h))$
using 5 10 **by** *auto*
have 12: $\vdash (\Box(f \wedge g) \vee (f \wedge g) \mathcal{U} h) = ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$
using 1 2 **by** *fastforce*
show *?thesis unfolding wait-d-def using 11 12*
by (*meson Prop04 UntilIdempotent*)
qed

lemma *WaitAndRule:*

$\vdash f \mathcal{W} g = (f \wedge \neg g) \mathcal{W} g$

proof –

have 1: $\vdash (f \mathcal{W} g \wedge (\neg g) \mathcal{W} g) = (f \wedge \neg g) \mathcal{W} g$

by (*meson Prop11 WaitRightDistAnd*)

have 2: $\vdash (\neg g) \mathcal{W} g$

by (*metis (no-types, opaque-lifting) DiamondEqvTrueUntil FalseUntil UntilAndRule UntilExclMid always-d-def int-simps(17) int-simps(4) inteq-reflection wait-d-def*)

show *?thesis*

using 1 2 **by** *fastforce*

qed

lemma *WaitUntilb:*

$\vdash f \mathcal{W} g = (\Box (f \wedge \neg g) \vee f \mathcal{U} g)$

proof –

have 1: $\vdash f \mathcal{W} g = (f \wedge \neg g) \mathcal{W} g$

by (*simp add: WaitAndRule*)

have 2: $\vdash (f \wedge \neg g) \mathcal{W} g = (\Box (f \wedge \neg g) \vee (f \wedge \neg g) \mathcal{U} g)$

by (*auto simp add: wait-d-def*)

have 3: $\vdash (f \wedge \neg g) \mathcal{U} g = f \mathcal{U} g$

by (*meson Prop11 UntilAndRule*)

show *?thesis*

using 1 2 3 **by** *fastforce*

qed

lemma *UntilNotDistWait:*

$\vdash (\neg (f \mathcal{U} g)) = (\neg g) \mathcal{W} (\neg f \wedge \neg g)$

proof –

have 1: $\vdash (\neg ((\neg g) \mathcal{W} (\neg f \wedge \neg g))) = (\neg(\neg f \wedge \neg g)) \mathcal{U} (\neg(\neg g) \wedge \neg(\neg f \wedge \neg g))$

using *WaitNotDistUntil* **by** *blast*

have 2: $\vdash (\neg(\neg f \wedge \neg g)) = (f \vee g)$

```

  by auto
have 3:  $\vdash (\neg (\neg g) \wedge \neg (\neg f \wedge \neg g)) = g$ 
  by auto
have 4:  $\vdash (\neg (\neg f \wedge \neg g)) \mathcal{U} (\neg (\neg g) \wedge \neg (\neg f \wedge \neg g)) =$ 
       $(f \vee g) \mathcal{U} g$ 
  using 2 3 UntilEqvUntil by blast
have 5:  $\vdash (f \vee g) \mathcal{U} g = ((f \vee g) \wedge \neg g) \mathcal{U} g$ 
  by (simp add: UntilAndRule)
have 6:  $\vdash ((f \vee g) \wedge \neg g) = (f \wedge \neg g)$ 
  by auto
have 7:  $\vdash ((f \vee g) \wedge \neg g) \mathcal{U} g = (f \wedge \neg g) \mathcal{U} g$ 
  using 6 inteq-reflection by fastforce
have 8:  $\vdash (f \wedge \neg g) \mathcal{U} g = f \mathcal{U} g$ 
  by (meson Prop11 UntilAndRule)
have 9:  $\vdash f \mathcal{U} g = (\neg ((\neg g) \mathcal{W} (\neg f \wedge \neg g)))$ 
  using 1 4 5 7 8 by fastforce
show ?thesis using 9 by auto
qed

```

lemma *UntilImpWait*:

```

 $\vdash f \mathcal{U} g \longrightarrow f \mathcal{W} g$ 
by (meson Prop03 WaitUntilb)

```

lemma *WaitAndDist*:

```

 $\vdash (\Box f \wedge g \mathcal{W} h) \longrightarrow (f \wedge g) \mathcal{W} (f \wedge h)$ 
proof -
  have 1:  $\vdash (\Box f \wedge g \mathcal{W} h) = (\Box f \wedge (\Box g \vee g \mathcal{U} h))$ 
    by (auto simp add: wait-d-def)
  have 2:  $\vdash (\Box f \wedge (\Box g \vee g \mathcal{U} h)) = ((\Box f \wedge \Box g) \vee (\Box f \wedge g \mathcal{U} h))$ 
    by auto
  have 3:  $\vdash (\Box f \wedge \Box g) = \Box(f \wedge g)$ 
    by (simp add: BoxAndBoxEqvBoxRule)
  have 4:  $\vdash (\Box f \wedge g \mathcal{U} h) \longrightarrow (f \wedge g) \mathcal{U} (f \wedge h)$ 
    by (simp add: UntilAlwaysAndDist)
  have 5:  $\vdash ((\Box f \wedge \Box g) \vee (\Box f \wedge g \mathcal{U} h)) \longrightarrow \Box(f \wedge g) \vee (f \wedge g) \mathcal{U} (f \wedge h)$ 
    using 3 4 by fastforce
  have 6:  $\vdash (\Box(f \wedge g) \vee (f \wedge g) \mathcal{U} (f \wedge h)) = (f \wedge g) \mathcal{W} (f \wedge h)$ 
    by (auto simp add: wait-d-def)
  show ?thesis
    using 1 5 6 by fastforce
qed

```

lemma *WaitDiamondOr*:

```

 $\vdash f \mathcal{W} (\Diamond g) = (\Box f \vee \Diamond g)$ 
proof -
  have 1:  $\vdash f \mathcal{W} (\Diamond g) = (\Box f \vee f \mathcal{U} (\Diamond g))$ 
    by (auto simp add: wait-d-def)
  have 2:  $\vdash f \mathcal{U} (\Diamond g) = \Diamond g$ 
    by (simp add: UntilAbsorpDiamond)
  show ?thesis using 1 2 Prop06 by blast

```

qed

lemma *WaitBoxImp:*

$\vdash f \mathcal{W} (\Box g) \longrightarrow \Box (f \mathcal{W} g)$

proof –

have 1: $\vdash f \mathcal{W} (\Box g) = (\Box f \vee f \mathcal{U} (\Box g))$

by (*auto simp add: wait-d-def*)

have 2: $\vdash \Box f = \Box (\Box f)$

by (*simp add: BoxEqvBoxBox*)

have 3: $\vdash f \mathcal{U} (\Box g) \longrightarrow \Box (f \mathcal{U} g)$

by (*simp add: UntilBoxImp*)

have 4: $\vdash (\Box f \vee f \mathcal{U} (\Box g)) \longrightarrow (\Box (\Box f) \vee \Box (f \mathcal{U} g))$

using 2 3 **by** *fastforce*

have 5: $\vdash \Box (\Box f) \longrightarrow \Box (\Box f \vee f \mathcal{U} g)$

by (*metis BoxImpBoxRule Prop08 UntilIdempotent UntilIntro int-simps(11) int-simps(25) inteq-reflection*)

have 6: $\vdash \Box (f \mathcal{U} g) \longrightarrow \Box (\Box f \vee f \mathcal{U} g)$

by (*metis BoxImpBoxRule UntilImpWait wait-d-def*)

have 7: $\vdash (\Box (\Box f) \vee \Box (f \mathcal{U} g)) \longrightarrow \Box (\Box f \vee f \mathcal{U} g)$

using 5 6 **by** *fastforce*

have 6: $\vdash \Box (\Box f \vee f \mathcal{U} g) = \Box (f \mathcal{W} g)$

by (*simp add: wait-d-def*)

show *?thesis*

by (*metis 4 7 lift-imp-trans wait-d-def*)

qed

lemma *WaitAbsorbBox:*

$\vdash f \mathcal{W} (\Box f) = \Box f$

by (*metis Prop02 Prop11 UntilBoxEqvBox UntilImpWait inteq-reflection wait-d-def*)

lemma *BoxImpWait:*

$\vdash \Box f \longrightarrow f \mathcal{W} g$

by (*auto simp add: wait-d-def*)

lemma *WaitDistNext:*

$\vdash \bigcirc (f \mathcal{W} g) = (\bigcirc f) \mathcal{W} (\bigcirc g)$

— nitpick finds counterexample, does not hold because of finite intervals

oops

lemma *WaitDistNextInfinite:*

$\vdash inf \longrightarrow \bigcirc (f \mathcal{W} g) = (\bigcirc f) \mathcal{W} (\bigcirc g)$

proof –

have 1: $\vdash \bigcirc (\Box f \vee f \mathcal{U} g) \longrightarrow (\bigcirc (\Box f) \vee \bigcirc (f \mathcal{U} g))$

by (*simp add: ChopOrImpRule next-d-def*)

have 2: $\vdash (\bigcirc (\Box f) \vee \bigcirc (f \mathcal{U} g)) \longrightarrow \bigcirc (\Box f \vee f \mathcal{U} g)$

by (*metis BoxImpWait NextImpNext Prop02 UntilImpWait wait-d-def*)

have 21: $\vdash \bigcirc (\Diamond(\neg f)) = \Diamond(\bigcirc(\neg f))$

by (*metis (no-types, lifting) ChopAssoc FiniteChopSkipEqvSkipChopFinite inteq-reflection next-d-def sometimes-d-def*)

have 22: $\vdash (\neg(\bigcirc f)) = (\text{empty} \vee \bigcirc(\neg f))$
using *WnextEqvEmptyOrNext*[of *LIFT*($\neg f$)] **unfolding** *wnext-d-def* **by** *simp*
have 23: $\vdash \text{inf} \longrightarrow \bigcirc(\neg f) = (\neg(\bigcirc f))$
using 22 *MoreAndInfEqvInf WnextAndMoreEqvNext WnextEqvEmptyOrNext* **by** *fastforce*
have 24: $\vdash \text{inf} \longrightarrow \Diamond(\bigcirc(\neg f)) = \Diamond(\neg(\bigcirc f))$
unfolding *sometimes-d-def* **using** 23
InfRightChopEqvChop[of *LIFT*($\bigcirc(\neg f)$)] *LIFT*($((\neg(\bigcirc f)))$) *LIFT*(*finite*)]
by *simp*
have 25: $\vdash \text{inf} \longrightarrow \bigcirc(\Diamond(\neg f)) = \Diamond(\neg(\bigcirc f))$
using 21 24 **by** *fastforce*
have 26: $\vdash \text{inf} \longrightarrow (\text{empty} \vee \bigcirc(\Diamond(\neg f))) = \bigcirc(\Diamond(\neg f))$
using *MoreAndInfEqvInf WnextAndMoreEqvNext WnextEqvEmptyOrNext* **by** *fastforce*
have 27: $\vdash \text{inf} \longrightarrow \text{wnext}(\Diamond(\neg f)) = \bigcirc(\Diamond(\neg f))$
by (*metis* 26 *WnextEqvEmptyOrNext inteq-reflection*)
have 28: $\vdash \text{inf} \longrightarrow (\neg(\bigcirc(\neg \Diamond(\neg f)))) = \Diamond(\neg \bigcirc f)$
using 21 24 27 **unfolding** *wnext-d-def* **by** *fastforce*
have 3: $\vdash \text{inf} \longrightarrow \bigcirc(\Box f) = \Box(\bigcirc f)$
unfolding *always-d-def* **using** 28 **by** *fastforce*
have 4: $\vdash \bigcirc(f \mathcal{U} g) = \bigcirc f \mathcal{U} \bigcirc g$
by (*simp add: NextUntil*)
have 5: $\vdash \bigcirc(\Box f \vee f \mathcal{U} g) = (\bigcirc(\Box f) \vee \bigcirc(f \mathcal{U} g))$
using 1 2 *int-iffI* **by** *blast*
have 6: $\vdash \text{inf} \longrightarrow \bigcirc(f \mathcal{W} g) = \bigcirc(\Box f \vee f \mathcal{U} g)$
by (*simp add: wait-d-def*)
have 7: $\vdash \text{inf} \longrightarrow \bigcirc(\Box f \vee f \mathcal{U} g) = (\bigcirc(\Box f) \vee \bigcirc(f \mathcal{U} g))$
using 5 **by** *auto*
have 8: $\vdash \text{inf} \longrightarrow (\bigcirc(\Box f) \vee \bigcirc(f \mathcal{U} g)) = (\Box(\bigcirc f) \vee \bigcirc f \mathcal{U} \bigcirc g)$
using 4 3 **by** *fastforce*
show *?thesis*
by (*metis* 5 8 *inteq-reflection wait-d-def*)
qed

lemma *WnextAlwaysEqvAlwaysWnext*:

$\vdash \text{finite} \longrightarrow \text{wnext}(\Box f) = \Box(\text{wnext } f)$

by (*metis* (*mono-tags*, *lifting*) *BoxEqvFiniteYields FiniteChopSkipEqvSkipChopFinite*
SkipYieldsEqvWeakNext YieldsYieldsEqvChopYields intI inteq-reflection unl-lift2)

lemma *WaitExpand*:

$\vdash f \mathcal{W} g = (g \vee (f \wedge \bigcirc(f \mathcal{W} g)))$

— nitpick finds counterexample, does not hold because of finite intervals

oops

lemma *WaitExpand*:

$\vdash f \mathcal{W} g = (g \vee (f \wedge \text{wnext}(f \mathcal{W} g)))$

proof –

have 1: $\vdash f \mathcal{W} g = (\Box f \vee f \mathcal{U} g)$

by (*simp add: wait-d-def*)

have 2: $\vdash \Box f = (f \wedge \text{wnext}(\Box f))$

by (*simp add: BoxEqvAndWnextBox*)

have 3: $\vdash f \mathcal{U} g = (g \vee (f \wedge \bigcirc(f \mathcal{U} g)))$

using *UntilNextUntil* **by** *blast*
have 4: $\vdash (f \wedge \text{wnext}(\Box f)) = (f \wedge (\text{empty} \vee \bigcirc (\Box f)))$
using 2 *BoxEqvAndEmptyOrNextBox*[of *f*] **by** *fastforce*
have 5: $\vdash \text{wnext}(f \mathcal{W} g) = (\text{empty} \vee \bigcirc (f \mathcal{W} g))$
using *WnextEqvEmptyOrNext* **by** *blast*
have 6: $\vdash (f \wedge (\text{empty} \vee \bigcirc (\Box f))) = ((f \wedge \text{empty}) \vee (f \wedge \bigcirc (\Box f)))$
by *auto*
have 7: $\vdash ((f \wedge \text{empty}) \vee (f \wedge \bigcirc (\Box f))) \vee$
 $(g \vee (f \wedge \bigcirc (f \mathcal{U} g))) =$
 $(g \vee (f \wedge (\text{empty} \vee \bigcirc (\Box f) \vee \bigcirc (f \mathcal{U} g))))$
by *auto*
have 8: $\vdash (\bigcirc (\Box f) \vee \bigcirc (f \mathcal{U} g)) = \bigcirc (\Box f \vee f \mathcal{U} g)$
by (*metis ChopOrEqv Prop11 next-d-def*)
show ?thesis
by (*metis 1 2 3 4 5 6 7 8 inteq-reflection*)
qed

lemma *InfImpWnextEqnNext*:
 $\vdash \text{inf} \longrightarrow \text{wnext } f = \bigcirc f$
proof –
have 1: $\vdash \text{wnext } f = (\text{empty} \vee \bigcirc f)$
by (*simp add: WnextEqvEmptyOrNext*)
have 2: $\vdash \text{inf} \longrightarrow \text{more}$
using *MoreAndInfEqvInf* **by** *auto*
have 3: $\vdash \text{inf} \longrightarrow (\text{empty} \vee \bigcirc f) = \bigcirc f$
unfolding *empty-d-def* **using** 2 **by** *fastforce*
show ?thesis
by (*metis 1 3 inteq-reflection*)
qed

lemma *WaitExpandInfinite*:
 $\vdash \text{inf} \longrightarrow f \mathcal{W} g = (g \vee (f \wedge \bigcirc (f \mathcal{W} g)))$
proof –
have 1: $\vdash f \mathcal{W} g = (g \vee (f \wedge \text{wnext}(f \mathcal{W} g)))$
using *WaitExpand* **by** *blast*
have 2: $\vdash \text{inf} \longrightarrow \text{wnext}(f \mathcal{W} g) = \bigcirc (f \mathcal{W} g)$
using *InfImpWnextEqnNext* **by** *blast*
have 3: $\vdash \text{inf} \longrightarrow (g \vee (f \wedge \text{wnext}(f \mathcal{W} g))) = (g \vee (f \wedge \bigcirc (f \mathcal{W} g)))$
using 2 **by** *auto*
show ?thesis **using** 1 3 **by** *fastforce*
qed

lemma *WaitExclMid*:
 $\vdash f \mathcal{W} g \vee f \mathcal{W} (\neg g)$
using *WaitExpand*
proof –
have 1: $\vdash f \mathcal{W} g = (g \vee f \wedge \text{wnext } (f \mathcal{W} g))$
by (*simp add: WaitExpand*)
have 2: $\vdash f \mathcal{W} (\neg g) = ((\neg g) \vee f \wedge \text{wnext } (f \mathcal{W} (\neg g)))$
by (*simp add: WaitExpand*)

have 3: $\vdash (f \mathcal{W} g \vee f \mathcal{W} (\neg g)) =$
 $((g \vee f \wedge \text{wnext } (f \mathcal{W} g)) \vee ((\neg g) \vee f \wedge \text{wnext } (f \mathcal{W} (\neg g))))$
using 1 2 **by** *fastforce*
from 3 **show** *?thesis* **by** *fastforce*
qed

lemma *WaitleftZero*:
 $\vdash \# \text{True } \mathcal{W} g$
by (*meson* *BoxGen* *BoxImpWait* *MP* *TrueW*)

lemma *WaitLeftDistOr*:
 $\vdash f \mathcal{W} (g \vee h) = (f \mathcal{W} g \vee f \mathcal{W} h)$
proof –
have 1: $\vdash f \mathcal{W} (g \vee h) = (\Box f \vee f \mathcal{U} (g \vee h))$
by (*simp* *add*: *wait-d-def*)
have 2: $\vdash (f \mathcal{W} g \vee f \mathcal{W} h) = ((\Box f \vee f \mathcal{U} g) \vee (\Box f \vee f \mathcal{U} h))$
by (*simp* *add*: *wait-d-def*)
have 3: $\vdash f \mathcal{U} (g \vee h) = (f \mathcal{U} g \vee f \mathcal{U} h)$
by (*simp* *add*: *UntilOrDist*)
from 1 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *WaitRightDistOr*:
 $\vdash f \mathcal{W} h \vee g \mathcal{W} h \longrightarrow (f \vee g) \mathcal{W} h$
proof –
have 0: $\vdash \Box g \longrightarrow \Box (f \vee g)$
by (*simp* *add*: *BoxImpBoxRule* *intI*)
have 1: $\vdash \Box f \longrightarrow \Box (f \vee g)$
by (*simp* *add*: *BoxImpBoxRule* *intI*)
have 11: $\vdash \Box f \vee \Box g \longrightarrow \Box (f \vee g)$
using 0 1 *Prop02* **by** *blast*
have 2: $\vdash f \mathcal{W} h = (\Box f \vee f \mathcal{U} h)$
by (*simp* *add*: *wait-d-def*)
have 3: $\vdash (f \vee g) \mathcal{W} h = (\Box (f \vee g) \vee (f \vee g) \mathcal{U} h)$
by (*simp* *add*: *wait-d-def*)
have 4: $\vdash g \mathcal{W} h = (\Box g \vee g \mathcal{U} h)$
by (*simp* *add*: *wait-d-def*)
have 5: $\vdash f \mathcal{U} h \vee g \mathcal{U} h \longrightarrow (f \vee g) \mathcal{U} h$
using *UntilRightDistOr* **by** *simp*
have 6: $\vdash (f \mathcal{W} h \vee g \mathcal{W} h) = ((\Box f \vee \Box g) \vee (f \mathcal{U} h \vee g \mathcal{U} h))$
using 2 4 **by** *fastforce*
from 11 5 6 3 **show** *?thesis*
by (*meson* *BoxImpWait* *Prop02* *Prop11* *UntilImpWait* *lift-imp-trans*)
qed

lemma *WaitOrRule*:
 $\vdash f \mathcal{W} g = (f \vee g) \mathcal{W} g$
proof –
have 1: $\vdash f \mathcal{W} g \longrightarrow (f \vee g) \mathcal{W} g$
by (*metis* (*no-types*, *lifting*) *Prop03* *Prop10* *UntilAbsorp-a* *WaitNotDistUntil* *int-iffD1* *int-simps*(14))

$\text{int-simps}(32) \text{ int-simps}(33) \text{ inteq-reflection})$
have 2: $\vdash (f \vee g) \mathcal{W} g \longrightarrow f \mathcal{W} g$
by (*metis* (*no-types*, *lifting*) *Prop03 Prop10 WaitNotDistUntil int-iffD2 int-simps*(14)
 $\text{int-simps}(32) \text{ int-simps}(33) \text{ inteq-reflection})$
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *UntilOrRule*:
 $\vdash f \mathcal{U} g = (f \vee g) \mathcal{U} g$
by (*metis* *UntilWait WaitOrRule inteq-reflection*)

lemma *WaitRule*:
 $\vdash (\neg f) \mathcal{W} f$
by (*metis* *BoxGen BoxImpWait MP WaitOrRule int-eq-true int-simps*(29) *inteq-reflection*)

lemma *UntilRule*:
 $\vdash (\neg f) \mathcal{U} f = \Diamond f$
using *DiamondEqvTrueUntil UntilOrRule inteq-reflection* **by** fastforce

lemma *WaitImpRule*:
 $\vdash (f \longrightarrow g) \mathcal{W} f$
proof –
have 1: $\vdash (f \longrightarrow g) \mathcal{W} f = ((f \longrightarrow g) \vee f) \mathcal{W} f$
by (*simp* *add*: *WaitOrRule*)
have 2: $\vdash (f \longrightarrow g) \vee f$
by *auto*
have 3: $\vdash ((f \longrightarrow g) \vee f) \mathcal{W} f = \#True \mathcal{W} f$
by (*metis* 1 2 *int-eq-true inteq-reflection*)
show ?thesis
using 1 3 *WaitleftZero* **by** fastforce
qed

lemma *DiamondUntilImpRule*:
 $\vdash \Diamond f \longrightarrow (f \longrightarrow g) \mathcal{U} f$
using *UntilWait WaitImpRule* **by** fastforce

lemma *WaitNotDist*:
 $\vdash (\neg (f \mathcal{W} g)) = (f \wedge \neg g) \mathcal{U} (\neg f \wedge \neg g)$
proof –
have 1: $\vdash (\neg (f \mathcal{W} g)) = (\neg g) \mathcal{U} (\neg f \wedge \neg g)$
using *WaitNotDistUntil* **by** *blast*
have 2: $\vdash (\neg g) \mathcal{U} (\neg f \wedge \neg g) = (\neg g \wedge \neg(\neg f \wedge \neg g)) \mathcal{U} (\neg f \wedge \neg g)$
using *UntilAndRule* **by** *blast*
have 3: $\vdash (\neg g \wedge \neg(\neg f \wedge \neg g)) = (f \wedge \neg g)$
by *auto*
have 4: $\vdash (\neg g \wedge \neg(\neg f \wedge \neg g)) \mathcal{U} (\neg f \wedge \neg g) = (f \wedge \neg g) \mathcal{U} (\neg f \wedge \neg g)$
using 3 *inteq-reflection* **by** *force*
show ?thesis **using** 1 2 4 **by** fastforce
qed

lemma *UntilNotDist*:

$\vdash (\neg (f \mathcal{U} g)) = (f \wedge \neg g) \mathcal{W} (\neg f \wedge \neg g)$

proof –

have 1: $\vdash (\neg (f \mathcal{U} g)) = (\neg g) \mathcal{W} (\neg f \wedge \neg g)$

using *UntilNotDistWait* **by** *blast*

have 2: $\vdash (\neg g) \mathcal{W} (\neg f \wedge \neg g) = (\neg g \wedge \neg(\neg f \wedge \neg g)) \mathcal{W} (\neg f \wedge \neg g)$

by (*simp add: WaitAndRule*)

have 3: $\vdash (\neg g \wedge \neg(\neg f \wedge \neg g)) = (f \wedge \neg g)$

by *auto*

have 4: $\vdash (\neg g \wedge \neg(\neg f \wedge \neg g)) \mathcal{W} (\neg f \wedge \neg g) = (f \wedge \neg g) \mathcal{W} (\neg f \wedge \neg g)$

using 3 *inteq-reflection* **by** *force*

show *?thesis* **using** 1 2 4 **by** *fastforce*

qed

lemma *UntilDuala*:

$\vdash (\neg (\neg f) \mathcal{U} (\neg g))) = g \mathcal{W} (f \wedge g)$

proof –

have 1: $\vdash (\neg (\neg f) \mathcal{U} (\neg g))) = (\neg f \wedge \neg (\neg g)) \mathcal{W} (\neg(\neg f) \wedge \neg (\neg g))$

using *UntilNotDist* **by** *blast*

have 2: $\vdash (\neg f \wedge g) \mathcal{W} (f \wedge g) = g \mathcal{W} (f \wedge g)$

using 1 *UntilNotDistWait int-eq* **by** *fastforce*

show *?thesis*

using 1 2 **by** *fastforce*

qed

lemma *UntilDualb*:

$\vdash (\neg (\neg f) \mathcal{U} (\neg g))) = (\neg f \wedge g) \mathcal{W} (f \wedge g)$

proof –

have 1: $\vdash (\neg (\neg f) \mathcal{U} (\neg g))) = (\neg f \wedge \neg (\neg g)) \mathcal{W} (\neg(\neg f) \wedge \neg (\neg g))$

using *UntilNotDist* **by** *blast*

show *?thesis*

using 1 **by** *auto*

qed

lemma *WaitDuala*:

$\vdash (\neg (\neg f) \mathcal{W} (\neg g))) = g \mathcal{U} (f \wedge g)$

proof –

have 1: $\vdash (\neg (\neg f) \mathcal{W} (\neg g))) = (\neg f \wedge \neg (\neg g)) \mathcal{U} (\neg(\neg f) \wedge \neg (\neg g))$

using *WaitNotDist* **by** *blast*

have 2: $\vdash (\neg f \wedge g) \mathcal{U} (f \wedge g) = g \mathcal{U} (f \wedge g)$

using 1 *WaitNotDistUntil int-eq* **by** *fastforce*

show *?thesis*

using 1 2 **by** *fastforce*

qed

lemma *WaitDualb*:

$\vdash (\neg (\neg f) \mathcal{W} (\neg g))) = (\neg f \wedge g) \mathcal{U} (f \wedge g)$

proof –

have 1: $\vdash (\neg (\neg f) \mathcal{W} (\neg g))) = (\neg f \wedge \neg (\neg g)) \mathcal{U} (\neg(\neg f) \wedge \neg (\neg g))$

using *WaitNotDist* **by** *blast*

show *?thesis* **using** 1 **by** *auto*
qed

lemma *WaitIdempotent*:

$\vdash f \mathcal{W} f = f$

by (*meson BoxElim Prop02 Prop12 UntilIdempotent UntilImpWait UntilIntro WaitUntilb int-iffD1 int-iffI lift-imp-trans*)

lemma *WaitRightZero*:

$\vdash f \mathcal{W} \# \text{True}$

by (*meson MP TrueW UntilImpWait UntilIntro*)

lemma *WaitLeftIdentity*:

$\vdash \# \text{False} \mathcal{W} g = g$

by (*metis (no-types, lifting) UntilAbsorp-c UntilNotDistWait WaitDuala WaitIdempotent WaitSRelease int-eq int-simps(17) int-simps(3) srelease-d-def*)

lemma *WaitImpOr*:

$\vdash f \mathcal{W} g \longrightarrow f \vee g$

by (*metis Prop03 WaitIdempotent WaitLeftDistOr WaitOrRule inteq-reflection*)

lemma *BoxOrImpWait*:

$\vdash \Box(f \vee g) \longrightarrow f \mathcal{W} g$

using *BoxImpWait WaitOrRule* **by** *fastforce*

lemma *BoxImpImpWait*:

$\vdash \Box(\neg g \longrightarrow f) \longrightarrow f \mathcal{W} g$

proof –

have 1: $\vdash (\neg g \longrightarrow f) = (f \vee g)$

by *auto*

have 2: $\vdash \Box(\neg g \longrightarrow f) = \Box(f \vee g)$

using 1 *BoxEqvBox* **by** *blast*

show *?thesis* **using** 2 *BoxOrImpWait* **by** *fastforce*

qed

lemma *WaitInsertion*:

$\vdash g \longrightarrow f \mathcal{W} g$

by (*simp add: Prop05 UntilIntro wait-d-def*)

lemma *WaitFrameNext*:

$\vdash \Box f \longrightarrow (\bigcirc g \longrightarrow \bigcirc (f \mathcal{W} g))$

by (*simp add: NextImpNext Prop01 Prop05 Prop09 WaitInsertion*)

lemma *WaitFrameDiamond*:

$\vdash \Box f \longrightarrow (\Diamond g \longrightarrow \Diamond (f \mathcal{W} g))$

by (*simp add: DiamondImpDiamond Prop01 Prop05 Prop09 WaitInsertion*)

lemma *WaitFrameBox*:

$\vdash \Box f \longrightarrow (\Box g \longrightarrow \Box (f \mathcal{W} g))$

by (*meson BoxAndBoxImpBoxRule OrImpUntil Prop09 UntilImpWait lift-imp-trans*)

lemma *WaitInductiona*:

$\vdash \Box (f \longrightarrow (\bigcirc f \wedge g) \vee h) \longrightarrow (f \longrightarrow g \mathcal{W} h)$

by (*simp add: UntilInduction-a wait-d-def*)

lemma *WaitInductionb*:

$\vdash \Box (f \longrightarrow \bigcirc f \vee g) \longrightarrow (f \longrightarrow f \mathcal{W} g)$

by (*simp add: UntilInduction-b wait-d-def*)

lemma *WaitInductionc*:

$\vdash \Box (f \longrightarrow \bigcirc f) \longrightarrow (f \longrightarrow f \mathcal{W} g)$

proof –

have 1: $\vdash (f \longrightarrow \bigcirc f) \longrightarrow (f \longrightarrow \text{wnext } f)$

unfolding *wnext-d-def* **using** *NextImpNotNextNot[of f]* **by** *auto*

have 2: $\vdash \Box (f \longrightarrow \bigcirc f) \longrightarrow \Box (f \longrightarrow \text{wnext } f)$

using 1 *BoxImpBoxRule* **by** *blast*

show *?thesis* **by** (*meson 2 BoxImpWait BoxInduct Prop09 lift-imp-trans*)

qed

lemma *WaitInductiond*:

$\vdash \Box (f \longrightarrow g \wedge \bigcirc f) \longrightarrow (f \longrightarrow f \mathcal{W} g)$

proof –

have 1: $\vdash (f \longrightarrow g \wedge \bigcirc f) \longrightarrow (f \longrightarrow \text{wnext } f)$

unfolding *wnext-d-def* **using** *NextImpNotNextNot[of f]* **by** *auto*

have 2: $\vdash \Box (f \longrightarrow g \wedge \bigcirc f) \longrightarrow \Box (f \longrightarrow \text{wnext } f)$

using 1 *BoxImpBoxRule* **by** *blast*

show *?thesis* **by** (*meson 2 BoxImpWait BoxInduct Prop09 lift-imp-trans*)

qed

lemma *WaitAbsorba*:

$\vdash (f \vee f \mathcal{W} g) = (f \vee g)$

proof –

have 1: $\vdash (f \vee f \mathcal{W} g) \longrightarrow (f \vee g)$

using *WaitImpOr* **by** *fastforce*

have 2: $\vdash f \vee g \longrightarrow f \vee f \mathcal{W} g$

using *WaitInsertion* **by** *fastforce*

show *?thesis* **using** 1 2 *int-iffI* **by** *blast*

qed

lemma *WaitAbsorbb*:

$\vdash (f \mathcal{W} g \vee g) = f \mathcal{W} g$

using *WaitInsertion[of g f]* **by** *auto*

lemma *WaitAbsorbc*:

$\vdash (f \mathcal{W} g \wedge g) = g$

using *WaitInsertion* **by** *fastforce*

lemma *WaitAbsorbd*:

$\vdash (f \mathcal{W} g \wedge (f \vee g)) = f \mathcal{W} g$

by (*meson Prop10 Prop11 WaitImpOr*)

lemma *WaitAbsorb*:

$\vdash (f \mathcal{W} g \vee (f \wedge g)) = f \mathcal{W} g$

unfolding *wait-d-def*

using *UntilAbsorp-d* **by** *fastforce*

lemma *WaitLeftAbsorb*:

$\vdash f \mathcal{W} (f \mathcal{W} g) = f \mathcal{W} g$

by (*metis* (*no-types*, *lifting*) *BoxEqvBoxBox* *UntilUntil* *WaitAbsorbBox* *WaitAbsorba* *WaitLeftDistOr* *inteq-reflection* *wait-d-def*)

lemma *WaitRightAbsorb*:

$\vdash (f \mathcal{W} g) \mathcal{W} g = f \mathcal{W} g$

by (*metis* (*no-types*, *lifting*) *LeftUntilAbsorp* *Prop10* *WaitInsertion* *WaitNotDistUntil* *int-iffD1* *int-iffI* *int-simps*(32) *inteq-reflection*)

lemma *WaitBox*:

$\vdash \Box f = f \mathcal{W} \#False$

by (*metis* (*no-types*, *lifting*) *BoxGen* *DiamondNotEqvNotBox* *UntilAbsorpAndDiamond* *UntilAbsorp-c* *int-eq-true* *int-simps*(2) *int-simps*(25) *inteq-reflection* *wait-d-def*)

lemma *WaitDiamond*:

$\vdash \Diamond f = (\neg(\neg f) \mathcal{W} \#False)$

using *DiamondNotEqvNotBox* *WaitBox* **by** *fastforce*

lemma *WaitImp*:

$\vdash f \mathcal{W} g \longrightarrow \Box f \vee \Diamond g$

by (*metis* *Prop08* *UntilImpDiamond* *WaitAbsorbb* *WaitImpOr* *WaitRightAbsorb* *int-eq* *wait-d-def*)

lemma *WaitRightUntilAbsorb*:

$\vdash f \mathcal{W} (f \mathcal{U} g) = f \mathcal{W} g$

by (*metis* *UntilUntil* *WaitOrRule* *inteq-reflection* *wait-d-def*)

lemma *WaitLeftUntilAbsorb*:

$\vdash (f \mathcal{U} g) \mathcal{W} g = f \mathcal{U} g$

by (*metis* *Prop11* *RightUntilAbsorp* *UntilAbsorp-b* *UntilImpWait* *WaitImpOr* *inteq-reflection*)

lemma *UntilRightWaitAbsorb*:

$\vdash f \mathcal{U} (f \mathcal{W} g) = f \mathcal{W} g$

using *UntilImpWait* *UntilIntro* *WaitLeftAbsorb* **by** *fastforce*

lemma *UntilLeftWaitAbsorb*:

$\vdash (f \mathcal{W} g) \mathcal{U} g = f \mathcal{U} g$

by (*metis* *UntilWait* *WaitRightAbsorb* *inteq-reflection*)

lemma *WaitDiamondAbsorb*:

$\vdash (\Diamond g) \mathcal{W} g = \Diamond g$

by (*metis* *DiamondEqvTrueUntil* *WaitLeftUntilAbsorb* *inteq-reflection*)

lemma *WaitAndBoxAbsorb*:

$\vdash (\Box f \wedge f \mathcal{W} g) = \Box f$
by (*meson BoxImpWait NotDiamondNotEqvBox Prop04 Prop10*)

lemma *WaitOrBoxAbsorb*:
 $\vdash (\Box f \vee f \mathcal{W} g) = f \mathcal{W} g$
by (*metis UntilRightWaitAbsorb WaitLeftAbsorb inteq-reflection wait-d-def*)

lemma *WaitAndBoxImpBox*:
 $\vdash f \mathcal{W} g \wedge \Box (\neg g) \longrightarrow \Box f$
by (*metis (no-types, opaque-lifting) Prop02 Prop05 Prop07 Prop08 UntilIdempotent UntilImpDiamond UntilIntro always-d-def int-simps(25) int-simps(4) inteq-reflection wait-d-def*)

lemma *BoxImpUntilOrBox*:
 $\vdash \Box f \longrightarrow f \mathcal{U} g \vee \Box (\neg g)$
proof –
have 1: $\vdash (\Box f \longrightarrow f \mathcal{U} g \vee \Box (\neg g)) =$
 $((\Box f \wedge \Diamond g) \longrightarrow f \mathcal{U} g)$
by (*auto simp add: always-d-def*)
have 2: $\vdash (\Box f \wedge \Diamond g) \longrightarrow f \mathcal{U} g$
using *UntilAndImp* **by** *blast*
show *?thesis*
using 1 2 **by** *fastforce*
qed

lemma *NotBoxAndWaitImpDiamond*:
 $\vdash \neg(\Box f) \wedge f \mathcal{W} g \longrightarrow \Diamond g$
using *WaitImp* **by** *fastforce*

lemma *DiamondImpNotBoxOrUntil*:
 $\vdash \Diamond g \longrightarrow \neg(\Box f) \vee f \mathcal{U} g$
proof –
have 1: $\vdash \Diamond g \wedge \Box f \longrightarrow f \mathcal{U} g$
using *UntilAndImp* **by** *fastforce*
show *?thesis* **using** 1 **by** *auto*
qed

lemma *WaitRightDistImp*:
 $\vdash (f \longrightarrow g) \mathcal{W} h \longrightarrow (f \mathcal{W} h \longrightarrow g \mathcal{W} h)$
proof –
have 0: $\vdash \Box(f \longrightarrow g) \wedge \Box f \longrightarrow \Box g$
by (*simp add: BoxAndBoxImpBoxRule intI*)
have 1: $\vdash \Box(f \longrightarrow g) \wedge \Box f \longrightarrow \Box g \vee g \mathcal{U} h$
using 0 **by** *auto*
have 2: $\vdash \Box(f \longrightarrow g) \wedge f \mathcal{U} h \longrightarrow ((f \longrightarrow g) \wedge f) \mathcal{U} ((f \longrightarrow g) \wedge h)$
using *UntilAlwaysAndDist[of LIFT(f \longrightarrow g) f h]* **by** *auto*
have 3: $\vdash (f \longrightarrow g) \wedge f \longrightarrow g$
by *auto*
have 4: $\vdash (f \longrightarrow g) \wedge h \longrightarrow h$
by *auto*
have 5: $\vdash ((f \longrightarrow g) \wedge f) \mathcal{U} ((f \longrightarrow g) \wedge h) \longrightarrow \Box g \vee g \mathcal{U} h$

by (simp add: 3 4 Prop05 UntilImpUntil)
 have 6: $\vdash \Box(f \longrightarrow g) \wedge f \mathcal{U} h \longrightarrow \Box g \vee g \mathcal{U} h$
 using 2 5 lift-imp-trans by blast
 have 7: $\vdash \Box(f \longrightarrow g) \longrightarrow \Box f \vee f \mathcal{U} h \longrightarrow \Box g \vee g \mathcal{U} h$
 by (simp add: 1 6 Prop09 Prop12)
 have 8: $\vdash \Box f \wedge (f \longrightarrow g) \mathcal{U} h \longrightarrow (f \wedge (f \longrightarrow g)) \mathcal{U} (f \wedge h)$
 by (simp add: UntilAlwaysAndDist)
 have 9: $\vdash f \wedge (f \longrightarrow g) \longrightarrow g$
 by auto
 have 10: $\vdash f \wedge h \longrightarrow h$
 by auto
 have 11: $\vdash (f \wedge (f \longrightarrow g)) \mathcal{U} (f \wedge h) \longrightarrow \Box g \vee g \mathcal{U} h$
 by (simp add: 10 9 Prop05 UntilImpUntil)
 have 12: $\vdash (f \longrightarrow g) \mathcal{U} h \wedge \Box f \longrightarrow \Box g \vee g \mathcal{U} h$
 using 8 11 by fastforce
 have 13: $\vdash (f \longrightarrow g) \mathcal{U} h \wedge f \mathcal{U} h \longrightarrow \Box g \vee g \mathcal{U} h$
 by (metis 3 Prop05 UntilAndDist UntilImpUntil UntilIntro UntilUntil inteq-reflection)
 have 14: $\vdash (f \longrightarrow g) \mathcal{U} h \longrightarrow \Box f \vee f \mathcal{U} h \longrightarrow \Box g \vee g \mathcal{U} h$
 by (simp add: 12 13 Prop09 Prop12)
 show ?thesis unfolding wait-d-def using 7 14 Prop02 by blast
 qed

lemma WaitLeftMono:

$\vdash \Box(f \longrightarrow g) \longrightarrow (f \mathcal{W} h \longrightarrow g \mathcal{W} h)$
 by (meson BoxImpWait WaitRightDistImp lift-imp-trans)

lemma WaitRightMono:

$\vdash \Box(f \longrightarrow g) \longrightarrow (h \mathcal{W} f \longrightarrow h \mathcal{W} g)$

proof –

have 1: $\vdash \Box(f \longrightarrow g) \longrightarrow (h \mathcal{U} f \longrightarrow h \mathcal{U} g)$
 by (simp add: UntilRightMono)
 have 2: $\vdash \Box(f \longrightarrow g) \longrightarrow (\Box h \longrightarrow \Box h \vee h \mathcal{U} g)$
 by auto
 have 3: $\vdash \Box(f \longrightarrow g) \longrightarrow (h \mathcal{U} f \longrightarrow \Box h \vee h \mathcal{U} g)$
 using 1 by auto
 have 4: $\vdash \Box(f \longrightarrow g) \longrightarrow (\Box h \vee h \mathcal{U} f \longrightarrow \Box h \vee h \mathcal{U} g)$
 using 2 3 by fastforce
 from 4 show ?thesis by (simp add: wait-d-def)
 qed

lemma WaitStrengthen:

$\vdash \Box((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f \mathcal{W} g \longrightarrow h \mathcal{W} i)$

proof –

have 1: $\vdash \Box((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f \mathcal{W} g \longrightarrow h \mathcal{W} g)$
 by (meson BoxAndBoxEqvBoxRule Prop01 Prop05 Prop11 WaitLeftMono lift-and-com lift-imp-trans)
 have 2: $\vdash \Box((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (h \mathcal{W} g \longrightarrow h \mathcal{W} i)$
 by (meson BoxElim BoxImpBoxBox BoxImpBoxRule Prop12 WaitRightMono lift-imp-trans)
 have 3: $\vdash \Box((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f \mathcal{W} g \longrightarrow h \mathcal{W} g) \wedge (h \mathcal{W} g \longrightarrow h \mathcal{W} i)$
 using 1 2 by fastforce
 have 4: $\vdash (f \mathcal{W} g \longrightarrow h \mathcal{W} g) \wedge (h \mathcal{W} g \longrightarrow h \mathcal{W} i) \longrightarrow (f \mathcal{W} g \longrightarrow h \mathcal{W} i)$

by auto
 from 3 4 show ?thesis by auto
 qed

lemma *WaitCatRule*:

$\vdash \Box (f \longrightarrow g \mathcal{W} h) \wedge (h \longrightarrow g \mathcal{W} i) \longrightarrow (f \longrightarrow g \mathcal{W} i)$

proof –

have 1: $\vdash \Box ((f \longrightarrow g \mathcal{W} h) \wedge (h \longrightarrow g \mathcal{W} i)) \longrightarrow \Box (f \longrightarrow g \mathcal{W} h)$

by (metis *BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)

have 2: $\vdash \Box ((f \longrightarrow g \mathcal{W} h) \wedge (h \longrightarrow g \mathcal{W} i)) \longrightarrow \Box (h \longrightarrow g \mathcal{W} i)$

by (metis *BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection*)

have 3: $\vdash \Box (h \longrightarrow g \mathcal{W} i) \longrightarrow \Box (g \mathcal{W} h \longrightarrow g \mathcal{W} (g \mathcal{W} i))$

by (metis *BoxEqvBoxBox BoxImpBoxRule WaitRightMono inteq-reflection*)

have 4: $\vdash \Box (g \mathcal{W} h \longrightarrow g \mathcal{W} (g \mathcal{W} i)) \longrightarrow \Box (g \mathcal{W} h \longrightarrow g \mathcal{W} i)$

by (metis *BoxEqvBoxBox WaitLeftAbsorb int-iffD1 inteq-reflection*)

have 5: $\vdash \Box (f \longrightarrow g \mathcal{W} h) \longrightarrow (f \longrightarrow g \mathcal{W} h)$

by (simp add: *BoxElim*)

have 6: $\vdash \Box (g \mathcal{W} h \longrightarrow g \mathcal{W} i) \longrightarrow (g \mathcal{W} h \longrightarrow g \mathcal{W} i)$

by (simp add: *BoxElim*)

have 7: $\vdash (f \longrightarrow g \mathcal{W} h) \wedge (g \mathcal{W} h \longrightarrow g \mathcal{W} i) \longrightarrow (f \longrightarrow g \mathcal{W} i)$

by auto

have 8: $\vdash \Box ((f \longrightarrow g \mathcal{W} h) \wedge (h \longrightarrow g \mathcal{W} i)) \longrightarrow$
 $(f \longrightarrow g \mathcal{W} h) \wedge (g \mathcal{W} h \longrightarrow g \mathcal{W} i)$

using 1 2 3 4 5 6 by fastforce

from 7 8 show ?thesis by auto

qed

lemma *LeftUntilWaitImp*:

$\vdash (f \mathcal{U} g) \mathcal{W} h \longrightarrow (f \mathcal{W} g) \mathcal{W} h$

by (meson *BoxGen MP UntilImpWait WaitLeftMono*)

lemma *RightWaitUntilImp*:

$\vdash f \mathcal{W} (g \mathcal{U} h) \longrightarrow f \mathcal{W} (g \mathcal{W} h)$

by (meson *BoxGen MP UntilImpWait WaitRightMono*)

lemma *RightUntilUntilImp*:

$\vdash f \mathcal{U} (g \mathcal{U} h) \longrightarrow f \mathcal{U} (g \mathcal{W} h)$

by (meson *BoxGen MP UntilImpWait UntilRightMono*)

lemma *LeftUntilUntilImp*:

$\vdash (f \mathcal{U} g) \mathcal{U} h \longrightarrow (f \mathcal{W} g) \mathcal{U} h$

by (simp add: *UntilImpUntil UntilImpWait*)

lemma *LeftUntilOrStrengthen*:

$\vdash (f \mathcal{U} g) \mathcal{U} h \longrightarrow (f \vee g) \mathcal{U} h$

by (simp add: *UntilImpOr UntilImpUntil*)

lemma *LeftWaitOrStrengthen*:

$\vdash (f \mathcal{W} g) \mathcal{W} h \longrightarrow (f \vee g) \mathcal{W} h$

by (meson *BoxGen MP WaitImpOr WaitLeftMono*)

lemma *RightWaitOrStrengthen*:

$\vdash f \mathcal{W} (g \mathcal{W} h) \longrightarrow f \mathcal{W} (g \vee h)$

by (*meson BoxGen MP WaitImpOr WaitRightMono*)

lemma *BoxImpBoxOr*:

$\vdash \Box f \longrightarrow \Box(f \vee g)$

by (*metis BoxEqvBoxBox BoxImpBoxRule BoxImpWait Prop12 WaitAbsorbd inteq-reflection*)

lemma *RightWaitOrOrder*:

$\vdash f \mathcal{W} (g \mathcal{W} h) \longrightarrow (f \vee g) \mathcal{W} h$

proof –

have 1: $\vdash f \mathcal{W} (g \mathcal{W} h) = (\Box f \vee f \mathcal{U} (\Box g \vee g \mathcal{U} h))$

by (*simp add: wait-d-def*)

have 2: $\vdash (f \vee g) \mathcal{W} h = (\Box (f \vee g) \vee (f \vee g) \mathcal{U} h)$

by (*simp add: wait-d-def*)

have 3: $\vdash \Box f \longrightarrow (\Box (f \vee g) \vee (f \vee g) \mathcal{U} h)$

using *BoxImpBoxOr* **by** *fastforce*

have 4: $\vdash f \mathcal{U} (\Box g \vee g \mathcal{U} h) = (f \mathcal{U} (\Box g) \vee f \mathcal{U} (g \mathcal{U} h))$

using *UntilOrDist* **by** *blast*

have 5: $\vdash f \mathcal{U} (g \mathcal{U} h) \longrightarrow (\Box (f \vee g) \vee (f \vee g) \mathcal{U} h)$

by (*simp add: Prop05 UntilRightOr*)

have 6: $\vdash f \mathcal{U} (\Box g) \longrightarrow (\Box (f \vee g) \vee (f \vee g) \mathcal{U} h)$

by (*metis BoxImpBoxRule BoxImpWait UntilBoxImp UntilImpOr lift-imp-trans wait-d-def*)

have 7: $\vdash (f \mathcal{U} (\Box g) \vee f \mathcal{U} (g \mathcal{U} h)) \longrightarrow (\Box (f \vee g) \vee (f \vee g) \mathcal{U} h)$

using 5 6 **by** *fastforce*

show *?thesis*

using 1 2 3 4 7 **by** *fastforce*

qed

lemma *RightWaitAndOrder*:

$\vdash f \mathcal{W} (g \wedge h) \longrightarrow f \mathcal{W} (g \mathcal{W} h)$

by (*metis Prop03 WaitAbsorbe WaitLeftDistOr inteq-reflection*)

lemma *UntilOrder*:

$\vdash ((\neg f) \mathcal{U} g \vee (\neg g) \mathcal{U} f) = \Diamond(f \vee g)$

proof –

have 1: $\vdash ((\neg f) \mathcal{U} g \vee (\neg g) \mathcal{U} f) \longrightarrow \Diamond(f \vee g)$

using *UntilAbsorpAndDiamond UntilOrDist* **by** *fastforce*

have 2: $\vdash \Diamond(f \vee g) = \#True \mathcal{U} (f \vee g)$

by (*metis DiamondEqvTrueUntil*)

have 3: $\vdash \#True \mathcal{U} (f \vee g) = (\neg (f \vee g)) \mathcal{U} (f \vee g)$

using 2 *UntilRule* **by** *fastforce*

have 4: $\vdash (\neg (f \vee g)) \mathcal{U} (f \vee g) = (\neg f \wedge \neg g) \mathcal{U} (f \vee g)$

by (*metis UntilAbsorp-c int-eq int-simps(14) int-simps(33)*)

have 5: $\vdash (\neg f \wedge \neg g) \mathcal{U} (f \vee g) \longrightarrow (\neg f \wedge \neg g) \mathcal{U} f \vee (\neg f \wedge \neg g) \mathcal{U} g$

by (*simp add: UntilOrDist int-iffD1*)

have 6: $\vdash (\neg f \wedge \neg g) \mathcal{U} f \longrightarrow (\neg g) \mathcal{U} f$

by (*metis UntilAndRule int-iffD2 inteq-reflection lift-and-com*)

have 7: $\vdash (\neg f \wedge \neg g) \mathcal{U} g \longrightarrow (\neg f) \mathcal{U} g$

by (metis *UntilAndRule int-iffD2*)
 have 8: $\vdash (\neg f \wedge \neg g) \mathcal{U} f \vee (\neg f \wedge \neg g) \mathcal{U} g \longrightarrow (\neg f) \mathcal{U} g \vee (\neg g) \mathcal{U} f$
 using 6 7 by fastforce
 have 9: $\vdash \Diamond(f \vee g) \longrightarrow (\neg f) \mathcal{U} g \vee (\neg g) \mathcal{U} f$
 using 2 3 4 5 8 by fastforce
 show ?thesis using 1 9 by fastforce
 qed

lemma *WaitOrder*:

$\vdash (\neg f) \mathcal{W} g \vee (\neg g) \mathcal{W} f$
 proof –
 have 1: $\vdash (\neg f) \mathcal{W} g = (\Box (\neg f) \vee (\neg f) \mathcal{U} g)$
 by (simp add: wait-d-def)
 have 2: $\vdash (\neg g) \mathcal{W} f = (\Box (\neg g) \vee (\neg g) \mathcal{U} f)$
 by (simp add: wait-d-def)
 have 3: $\vdash ((\Box (\neg f) \vee (\neg f) \mathcal{U} g) \vee (\Box (\neg g) \vee (\neg g) \mathcal{U} f)) =$
 $(\Box (\neg f) \vee \Box (\neg g)) \vee ((\neg f) \mathcal{U} g \vee (\neg g) \mathcal{U} f))$
 by auto
 have 4: $\vdash ((\neg f) \mathcal{U} g \vee (\neg g) \mathcal{U} f) = \Diamond(f \vee g)$
 using *UntilOrder* by blast
 have 5: $\vdash (\Box (\neg f) \vee \Box (\neg g)) = (\neg (\Diamond f) \vee \neg (\Diamond g))$
 by (simp add: always-d-def)
 have 6: $\vdash \Diamond(f \vee g) = (\Diamond f \vee \Diamond g)$
 by (simp add: ChopOrEqv sometimes-d-def)
 have 7: $\vdash (\Box (\neg f) \vee \Box (\neg g)) \vee \Diamond(f \vee g)$
 using 5 6 by fastforce
 show ?thesis
 using 1 2 4 7 by fastforce
 qed

lemma *WaitImpOrder*:

$\vdash f \mathcal{W} g \wedge (\neg g) \mathcal{W} h \longrightarrow f \mathcal{W} h$
 proof –
 have 1: $\vdash f \mathcal{W} g = (\Box f \vee f \mathcal{U} g)$
 by (simp add: wait-d-def)
 have 2: $\vdash f \mathcal{W} h = (\Box f \vee f \mathcal{U} h)$
 by (simp add: wait-d-def)
 have 3: $\vdash (\neg g) \mathcal{W} h = (\Box (\neg g) \vee (\neg g) \mathcal{U} h)$
 by (simp add: wait-d-def)
 have 4: $\vdash \Box f \wedge (\Box (\neg g) \vee (\neg g) \mathcal{U} h) \longrightarrow (\Box f \vee f \mathcal{U} h)$
 by auto
 have 5: $\vdash (\Box f \vee f \mathcal{U} g) \wedge \Box (\neg g) \longrightarrow (\Box f \vee f \mathcal{U} h)$
 using 1 *WaitAndBoxImpBox* by fastforce
 have 6: $\vdash f \mathcal{U} g \wedge (\neg g) \mathcal{U} h \longrightarrow (\Box f \vee f \mathcal{U} h)$
 by (simp add: Prop05 UntilNotImp)
 have 7: $\vdash \Box f \wedge (\neg g) \mathcal{U} h \longrightarrow (\Box f \vee f \mathcal{U} h)$
 by auto
 have 8: $\vdash (\Box f \vee f \mathcal{U} g) \wedge (\neg g) \mathcal{U} h \longrightarrow (\Box f \vee f \mathcal{U} h)$
 using 6 7 by fastforce
 have 9: $\vdash (\Box f \vee f \mathcal{U} g) \wedge (\Box (\neg g) \vee (\neg g) \mathcal{U} h) \longrightarrow (\Box f \vee f \mathcal{U} h)$

```

    using 5 8 by fastforce
    show ?thesis by (simp add: 9 wait-d-def)
qed

```

end

2.9 Pi operator

```

theory Pi
imports Until Omega
begin

```

This theory introduces the Pi operator [13, 10]. The Pi operator is defined in terms of the filter operator. We prove the soundness of the rules and axiom system. The until operator from Until.thy is used as there is a striking similarity of the expressiveness of the until and the Pi operator [10].

2.9.1 Definitions

definition *sxfilt* :: *'a nelist* \Rightarrow (*'a:: world*) *formula* \Rightarrow *'a nelist nelist*
where *sxfilt* *s f* = (*nfilter* ($\lambda \sigma. \sigma \models f$) (*ndropns* *s*))

definition *pfilt* :: *'a nelist* \Rightarrow (*'a:: world*) *formula* \Rightarrow *'a nelist*
where *pfilt* *s f* = (*nmap* ($\lambda s. \text{nnth } s \ 0$) (*sxfilt* *s f*))

definition *pi-d* :: (*'a:: world*) *formula* \Rightarrow *'a formula* \Rightarrow *'a formula*
where *pi-d* *F G* $\equiv \lambda s. ((\exists i \leq \text{nlength } s. (\text{ndropn } i \ s) \models F) \wedge ((\text{pfilt } s \ F) \models G))$

syntax

-pi-d :: [*lift, lift*] \Rightarrow *lift* ((*-* Π *-*) [84,84] 83)

syntax (*ASCII*)

-pi-d :: [*lift, lift*] \Rightarrow *lift* ((*-* Π *-*) [84,84] 83)

translations

-pi-d \Rightarrow *CONST pi-d*

definition *upi-d* :: (*'a:: world*) *formula* \Rightarrow *'a formula* \Rightarrow *'a formula*
where *upi-d* *F G* $\equiv \text{LIFT}(\neg(F \Pi (\neg G)))$

syntax

-upi-d :: [*lift, lift*] \Rightarrow *lift* ((*-* Π^u *-*) [84,84] 83)

syntax (*ASCII*)

-upi-d :: [*lift, lift*] \Rightarrow *lift* ((*-* UPI *-*) [84,84] 83)

translations

$-upi-d \quad \rightleftharpoons \quad CONST \ upi-d$

2.9.2 Semantic Lemmas

lemma *sfxfilt-help*:

$(\exists \ ys \in \ nset \ (ndropns \ xs) . f \ ys) = (\exists \ i \leq \ nlength \ xs . f \ (ndropn \ i \ xs))$

using *in-nset-ndropns* **by** *auto*

lemma *pfiltinit-help*:

$(\exists \ y \in \ nset \ (xs) . w \ (NNil \ y)) = (\exists \ i \leq \ nlength \ xs . w \ (NNil \ (nnth \ xs \ i)))$

by (*metis in-nset-conv-nnth*)

lemma *sfxfilt-nnth*:

assumes $(\exists \ i \leq \ nlength \ \sigma . (ndropn \ i \ \sigma) \models f)$

$i \leq nlength \ (sfxfilt \ \sigma \ f)$

shows $(nnth \ (sfxfilt \ \sigma \ f) \ i) \models f$

using *assms* **by** (*metis in-nset-ndropns nkfilter-nnth-aa sfxfilt-def*)

lemma *pfilt-exists*:

assumes $(\exists \ i \leq nlength \ \sigma . f \ (ndropn \ i \ \sigma))$

shows $(\exists \ i \leq nlength \ (sfxfilt \ \sigma \ f) . (nnth \ (sfxfilt \ \sigma \ f) \ i) \models f)$

using *assms* **by** (*metis sfxfilt-nnth zero-enat-def zero-le*)

lemma *sfxfilt-pfilt-nlength*:

shows $nlength \ (pfilt \ \sigma \ f) = nlength \ (nmap \ (\lambda s . nnth \ s \ 0) \ (sfxfilt \ \sigma \ f))$

by (*simp add: pfilt-def*)

lemma *sfxfilt-pfilt-nnth*:

assumes $j \leq nlength \ (pfilt \ \sigma \ f)$

shows $nnth \ (pfilt \ \sigma \ f) \ j = nnth \ (nmap \ (\lambda s . nnth \ s \ 0) \ (sfxfilt \ \sigma \ f)) \ j$

using *assms* **by** (*simp add: pfilt-def*)

lemma *sfxfilt-pfilt*:

shows $(nmap \ (\lambda s . nnth \ s \ 0) \ (sfxfilt \ \sigma \ f)) = pfilt \ \sigma \ f$

by (*simp add: pfilt-def*)

lemma *sfxfilt-nlength-bound*:

assumes $(\exists \ i \leq nlength \ xs . f \ (ndropn \ i \ xs))$

shows $nlength \ (sfxfilt \ xs \ f) \leq nlength \ xs$

using *assms*

by (*metis length-nfilter-le ndropns-nlength sfxfilt-def*)

lemma *pfilt-nlength-bound*:

assumes $(\exists \ i \leq nlength \ \sigma . f \ (ndropn \ i \ \sigma))$

shows $nlength \ (pfilt \ \sigma \ f) \leq nlength \ \sigma$

using *assms* **by** (*simp add: pfilt-def sfxfilt-nlength-bound*)

lemma *sfxfilt-nlength-nnth-bound*:

assumes $(\exists \ i \leq nlength \ \sigma . f \ (ndropn \ i \ \sigma))$

$j \leq \text{nlength } (\text{sfxfilt } \sigma f)$
shows $\text{nlength } (\text{nnth } (\text{sfxfilt } \sigma f) j) \leq \text{nlength } \sigma$
using *assms* **by** (*simp* *add*: *nfilter-ndropns-nnth-bound* *sfxfilt-def* *sfxfilter-help*)

lemma *sfxfilt-pifilt-nnth-ndropn*:
assumes $(\exists i \leq \text{nlength } \sigma. (\text{ndropn } i \sigma) \models f)$
 $j \leq \text{nlength } (\text{pifilt } \sigma f)$
shows $\text{nnth } (\text{sfxfilt } \sigma f) j = \text{ndropn } (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) j) \sigma$
proof –
have 0: $\exists x \in \text{nset } (\text{ndropns } \sigma). f x$
by (*simp* *add*: *assms*(1) *sfxfilter-help*)
have 1: $\text{nnth } (\text{sfxfilt } \sigma f) j = \text{nnth } (\text{nfilter } f (\text{ndropns } \sigma)) j$
by (*simp* *add*: *sfxfilt-def*)
have 2: $\text{nnth } (\text{nfilter } f (\text{ndropns } \sigma)) j = \text{nnth } (\text{ndropns } \sigma) (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) j)$
using *nkfilter-nfilter*[of *ndropns* $\sigma \lambda s. s \models f j 0$] *assms* **unfolding** *pifilt-def* *sfxfilt-def*
by (*simp* *add*: 0 *nkfilter-nlength*)
have 30: $\text{enat } (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) j) \leq \text{nlength } \sigma$
using 0 *nkfilter-upperbound*[of $(\text{ndropns } \sigma) f j 0$] *assms* **unfolding** *pifilt-def* *sfxfilt-def*
by (*metis* *gen-nlength-def* *ndropns-nlength* *nkfilter-nlength* *nlength-code* *nlength-nmap*)
have 3: $\text{nnth } (\text{ndropns } \sigma) (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) j) =$
 $\text{ndropn } (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) j) \sigma$
using *ndropns-nnth*[of $(\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) j)$] **]**
using 30 **by** *blast*
show ?thesis **by** (*simp* *add*: 1 2 3)
qed

lemma *pifilt-nnth*:
assumes $(\exists i \leq \text{nlength } \sigma. f (\text{ndropn } i \sigma))$
 $i \leq \text{nlength } (\text{pifilt } \sigma f)$
shows $(\exists k \leq \text{nlength } \sigma. \text{nnth } (\text{pifilt } \sigma f) i = \text{nnth } \sigma k)$
using *assms* *sfxfilt-pifilt-nnth-ndropn*[of $\sigma f i$]
by (*simp* *add*: *pifilt-def*)
(metis enat-the-enat infinity-ileE linorder-le-cases ndropn-all ndropn-nfirst)

lemma *sfxfilt-nlength-imp*:
assumes $(\exists i \leq \text{nlength } \sigma. f (\text{ndropn } i \sigma) \wedge g (\text{ndropn } i \sigma))$
shows $\text{nlength } (\text{sfxfilt } \sigma (\text{LIFT}(f \wedge g))) \leq \text{nlength } (\text{sfxfilt } \sigma f)$
proof –
have 1: $\text{nlength } (\text{sfxfilt } \sigma (\text{LIFT}(f \wedge g))) =$
 $\text{nlength } (\text{nfilter } (\lambda ys. f ys \wedge g ys) (\text{ndropns } \sigma))$
by (*simp* *add*: *sfxfilt-def*)
have 2: $\text{nlength } (\text{nfilter } f (\text{ndropns } \sigma)) = \text{nlength } (\text{sfxfilt } \sigma f)$
by (*simp* *add*: *sfxfilt-def*)
have 3: $\exists x \in \text{nset } (\text{ndropns } \sigma). f x \wedge g x$
using *assms* *in-nset-ndropns* **by** *blast*
have 4: $\text{nlength } (\text{nfilter } (\lambda x. f x \wedge g x) (\text{ndropns } \sigma)) \leq \text{nlength } (\text{nfilter } f (\text{ndropns } \sigma))$
using 3 *nfilter-nlength-imp*[of *ndropns* $\sigma f g$] **by** *auto*
show ?thesis **using** 1 2 4 **by** *auto*
qed

lemma *pfilt-nlength-imp*:
assumes $(\exists i \leq nlength \ \sigma. f \ (ndropn \ i \ \sigma) \wedge g \ (ndropn \ i \ \sigma))$
shows $nlength \ (pfilt \ \sigma \ (LIFT(f \wedge g))) \leq nlength \ (pfilt \ \sigma \ f)$
using *assms*
by (*simp add: sfxfilt-nlength-imp pfilt-def*)

lemma *nellist-nset-sfxfilt* [*simp*]:
assumes $(\exists i \leq nlength \ xs. f \ (ndropn \ i \ xs))$
shows $(nset \ (sfxfilt \ xs \ f)) = \{ys. ys \in nset \ (ndropns \ xs) \wedge f \ (ys)\}$
using *assms*
proof –
have 1: $nset \ (sfxfilt \ xs \ f) = nset \ (nfilter \ f \ (ndropns \ xs))$
by (*simp add: sfxfilt-def*)
have 2: $\exists x \in nset \ (ndropns \ xs). f \ x$
using *assms* **using** *in-nset-ndropns* **by** *blast*
have 3: $nset \ (nfilter \ f \ (ndropns \ xs)) = \{ys. ys \in nset \ (ndropns \ xs) \wedge f \ ys\}$
using 2 *nset-nfilter[of ndropns xs f]* **by** *auto*
show ?thesis **by** (*simp add: 1 3*)
qed

lemma *subset-nfilter*:
assumes $\exists x \in nset \ xs. P \ x$
shows $nset(nfilter \ P \ xs) \leq nset(nfilter \ (\lambda x. P \ x \vee Q \ x) \ xs)$
proof –
have 1: $\exists x \in nset \ xs. P \ x \vee Q \ x$
using *assms* **by** *blast*
have 2: $nset(nfilter \ (\lambda x. P \ x \vee Q \ x) \ xs) = \{x \in nset \ xs. P \ x \vee Q \ x\}$
using *assms* *nset-nfilter[of xs (λx. P x ∨ Q x)]* **by** *blast*
have 3: $nset(nfilter \ P \ xs) = \{x \in nset \ xs. P \ x\}$
using *assms* *nset-nfilter[of xs P]* **by** (*simp add: Collect-conj-eq*)
have 4: $\{x \in nset \ xs. P \ x\} \leq \{x \in nset \ xs. P \ x \vee Q \ x\}$
by *auto*
show ?thesis **by** (*simp add: 2 3 4*)
qed

lemma *nellist-subset-sfxfilt* [*simp*]:
assumes $(\exists i \leq nlength \ xs. f \ (ndropn \ i \ xs))$
shows $(nset \ (sfxfilt \ xs \ f)) \leq (nset \ (sfxfilt \ xs \ (LIFT(f \vee g))))$
proof –
have 1: $\exists x \in nset \ (ndropns \ xs). f \ x$
using *assms* **using** *in-nset-ndropns* **by** *blast*
have 2: $nset \ (sfxfilt \ xs \ f) = nset \ (nfilter \ f \ (ndropns \ xs))$
by (*simp add: sfxfilt-def*)
have 3: $nset \ (sfxfilt \ xs \ (LIFT(f \vee g))) = nset \ (nfilter \ (\lambda x. f \ x \vee g \ x) \ (ndropns \ xs))$
by (*simp add: sfxfilt-def*)
have 4: $nset \ (nfilter \ f \ (ndropns \ xs)) \leq nset \ (nfilter \ (\lambda x. f \ x \vee g \ x) \ (ndropns \ xs))$
using 1 *subset-nfilter[of ndropns xs f g]* **by** *auto*
show ?thesis **using** 2 3 4 **by** *blast*
qed

lemma *nellist-nset-nmap*:

$(nset (nmap (\lambda s. nnth s 0) xs)) = \{(nnth ys 0) \mid ys. ys \in nset xs\}$
by (*auto simp add: in-nset-conv-nnth nset-conv-nnth*)
(metis nnth-nmap)

lemma *nellist-nset-pifilt [simp]*:

assumes $(\exists i \leq nlength xs. f (ndropn i xs))$
shows $(nset (pifilt xs f)) = \{(nnth ys 0) \mid ys. ys \in nset(ndropns xs) \wedge f (ys)\}$
proof –
have 1: $(nset (pifilt xs f)) = (nset (nmap (\lambda s. nnth s 0) (sfxfilt xs f)))$
by (*simp add: pifilt-def*)
have 2: $(nset (nmap (\lambda s. nnth s 0) (sfxfilt xs f))) =$
 $\{(nnth ys 0) \mid ys. ys \in nset (sfxfilt xs f)\}$
using *nellist-nset-nmap[of sfxfilt xs f]* **by** *auto*
have 3: $\{(nnth ys 0) \mid ys. ys \in nset (sfxfilt xs f)\} =$
 $\{(nnth ys 0) \mid ys. ys \in nset(ndropns xs) \wedge f (ys)\}$
using *assms* **by** *auto*
show *?thesis* **using** 1 2 3 **by** *auto*
qed

lemma *nellist-nnth-sfxfilt-in-nset*:

$x \in nset (sfxfilt \sigma (LIFT(f \vee g))) =$
 $(\exists i \leq nlength (sfxfilt \sigma (LIFT(f \vee g))). x = (nnth (sfxfilt \sigma (LIFT(f \vee g))) i))$
by (*metis in-nset-conv-nnth*)

lemma *nfilter-nnth-or*:

assumes $\exists x \in nset xs. P x$
shows $\exists x \in nset(nfilter (\lambda x. P x \vee Q x) xs). P x$
proof –
have 0: $\exists x \in nset xs. P x \vee Q x$
using *assms* **by** *auto*
have 1: $\exists i \leq nlength(nfilter (\lambda x. P x \vee Q x) xs). P (nnth (nfilter P xs) i)$
using *assms nkfilter-nnth-aa[of xs (\lambda x. P x)] nkfilter-nnth-aa*
by (*metis (mono-tags, lifting) le-cases le-zero-eq zero-enat-def*)
obtain *i* **where** 2: $i \leq nlength(nfilter (\lambda x. P x \vee Q x) xs) \wedge P (nnth (nfilter P xs) i) \wedge$
 $(nnth (nfilter P xs) i) \in nset (nfilter P xs)$
by (*metis 1 in-nset-conv-nnth le-cases nfinite-ntaken nset-nlast ntaken-all ntaken-nlast*)
have 3: $nset (nfilter P xs) \leq nset(nfilter (\lambda x. P x \vee Q x) xs)$
using *assms subset-nfilter[of xs P Q]* **by** *auto*
have 4: $(nnth (nfilter P xs) i) \in nset (nfilter P xs)$
using 2 **by** *auto*
have 5: $(nnth (nfilter P xs) i) \in nset(nfilter (\lambda x. P x \vee Q x) xs)$
using 3 4 **by** *blast*
show *?thesis* **using** 2 5 **by** *blast*
qed

lemma *sfxfilt-nnth-or*:

assumes $(\exists i \leq nlength \sigma. (ndropn i \sigma) \models f)$
shows $(\exists i \leq nlength (sfxfilt \sigma (LIFT(f \vee g))). (nnth (sfxfilt \sigma (LIFT(f \vee g))) i) \models f)$

proof –

have 1: $\exists x \in \text{nset } (\text{ndropns } \sigma). f x$
using *assms* **by** (*simp* *add*: *sfxfilt-help*)
have 2: *sfxfilt* σ (*LIFT*($f \vee g$)) = *nfilter* ($\lambda x. f x \vee g x$) (*ndropns* σ)
by (*simp* *add*: *sfxfilt-def*)
have 3: $\exists x \in \text{nset}(\text{nfilter } (\lambda x. f x \vee g x) (\text{ndropns } \sigma)). f x$
using 1 *nfilter-nnth-or*[*of* *ndropns* σ f g] **by** *auto*
from 2 3 **show** *?thesis*
by (*metis* (*full-types*) *in-nset-conv-nnth*)
qed

lemma *NotPiFalse*:

$\sigma \models \neg ((\# \text{False}) \Pi f)$
by (*simp* *add*: *pi-d-def*)

lemma *pfilt-true*:

pfilt σ (*LIFT*($\# \text{True}$)) = σ
by (*simp* *add*: *pfilt-def* *sfxfilt-def* *nmap-first-ndropns*)

lemma *pfilt-state-help*:

$(\exists x \in \text{nset } (\text{ndropns } xs) . (\text{LIFT}(\text{init } w)) x) = (\exists x \in \text{nset } xs. w ((\text{NNil } x)))$
proof (*auto* *simp* *add*: *init-defs*)
show $\bigwedge x. x \in \text{nset } (\text{ndropns } xs) \implies w (\text{NNil } (\text{nfirst } x)) \implies \exists x \in \text{nset } xs. w (\text{NNil } x)$
using *in-nset-ndropns* **by** (*metis* *in-nset-conv-nnth* *ndropn-nfirst*)
show $\bigwedge x. x \in \text{nset } xs \implies w (\text{NNil } x) \implies \exists x \in \text{nset } (\text{ndropns } xs). w (\text{NNil } (\text{nfirst } x))$
by (*metis* *in-nset-ndropns-nhd* *ndropn-0* *ndropn-nfirst*)
qed

lemma *pfilt-init*:

shows (*pfilt* xs ($\lambda s. s \models \text{init } w$)) = *nfilter* ($\lambda y. w ((\text{NNil } y))$) xs
proof (*simp* *add*: *init-defs* *pfilt-def* *sfxfilt-def*)
show *nmap* ($\lambda s. \text{nnth } s 0$) (*nfilter* ($\lambda s. w (\text{NNil } (\text{nfirst } s))$) (*ndropns* xs)) =
nfilter ($\lambda y. w (\text{NNil } y)$) xs
proof –
have 1: $\bigwedge x. ((\lambda y. w ((\text{NNil } y))) \circ (\lambda s. \text{nnth } s 0)) x = (\lambda s. w (\text{NNil } (\text{nfirst } s))) x$
by *simp* (*metis* *ndropn-0* *ndropn-nfirst*)
have 2: $((\lambda y. w ((\text{NNil } y))) \circ (\lambda s. \text{nnth } s 0)) = (\lambda s. w (\text{NNil } (\text{nfirst } s)))$
using 1 **by** *blast*
have 4: *nmap* ($\lambda s. \text{nnth } s 0$) (*nfilter* ($\lambda s. w (\text{NNil } (\text{nfirst } s))$) (*ndropns* xs)) =
nfilter ($\lambda y. w ((\text{NNil } y))$) (*nmap* ($\lambda s. \text{nnth } s 0$) (*ndropns* xs))
by (*metis* (*mono-tags*, *lifting*) 1 *nfilter-cong* *nfilter-nmap*)
have 5: (*nmap* ($\lambda s. \text{nnth } s 0$) (*ndropns* xs)) = xs
by (*simp* *add*: *nmap-first-ndropns*)
show *?thesis*
by (*simp* *add*: 4 5)
qed
qed

lemma *pfilt-init-a*:

shows (*pfilt* xs ($\lambda s. w (\text{NNil } (\text{nnth } s 0))$)) = *nfilter* ($\lambda y. w (\text{NNil } y)$) xs

proof –

have 2: $(\text{pifilt } xs \ (\lambda s. s \models \text{init } w)) = (\text{pifilt } xs \ (\lambda s. w \ (NNil \ (\text{nnth } s \ 0)) \))$
by $(\text{metis } (\text{no-types}, \text{lifting}) \text{ in-nset-ndropns init-defs ndropn-nfirst nfilter-cong nnth-zero-ndropn ntaken-0 pifilt-def sfxfilt-def})$
show ?thesis **using** pifilt-init 2 **by** (metis)
qed

lemma pifilt-pifilt :

assumes $(\exists i \leq \text{nlength } xs. f \ (\text{ndropn } i \ xs))$
 $(\exists i \leq \text{nlength } (\text{pifilt } xs \ f). w \ (NNil \ (\text{nnth } (\text{ndropn } i \ (\text{pifilt } xs \ f)) \ 0)))$
shows $(\text{pifilt } (\text{pifilt } xs \ f) \ (\text{LIFT}(\text{init } w))) = \text{pifilt } xs \ (\text{LIFT}(f \wedge \text{init } w))$
proof –
have 1: $\exists i \leq \text{nlength } (\text{pifilt } xs \ f). (\text{LIFT}(\text{init } w)) \ (\text{ndropn } i \ (\text{pifilt } xs \ f))$
using assms **by** (simp add: init-defs ndropn-nfirst)
have 2: $(\text{pifilt } (\text{pifilt } xs \ f) \ (\text{LIFT}(\text{init } w))) = \text{nfilter } (\lambda y. w \ ((NNil \ y))) \ (\text{pifilt } xs \ f)$
using 1 pifilt-init[of $(\text{pifilt } xs \ f) \ w$] **by** auto
have 3: $(\text{pifilt } xs \ f) = \text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{sfxfilt } xs \ f)$
by (simp add: assms sfxfilt-pifilt)
have 4: $(\text{sfxfilt } xs \ f) = \text{nfilter } (\lambda ys. f \ ys) \ (\text{ndropns } xs)$
using sfxfilt-def **by** blast
have 5: $(\text{pifilt } xs \ f) = \text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfilter } (\lambda ys. f \ ys) \ (\text{ndropns } xs))$
by (simp add: 3 4)
have 6: $\text{nfilter } (\lambda y. w \ (NNil \ y)) \ (\text{pifilt } xs \ f) =$
 $\text{nfilter } (\lambda y. w \ (NNil \ y)) \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfilter } (\lambda ys. f \ ys) \ (\text{ndropns } xs)))$
using 5 **by** simp
have 7: $\text{nfilter } (\lambda y. w \ (NNil \ y)) \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfilter } (\lambda ys. f \ ys) \ (\text{ndropns } xs))) =$
 $\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfilter } ((\lambda y. w \ (NNil \ y)) \circ (\lambda s. \text{nnth } s \ 0)) \ (\text{nfilter } (\lambda ys. f \ ys) \ (\text{ndropns } xs)))$
using assms 3 4 **by** (simp add: nfilter-nmap pifiltinit-help)
have 8: $\exists x \in \text{nset } (\text{nfilter } (\lambda ys. f \ ys) \ (\text{ndropns } xs)). ((\lambda y. w \ ((NNil \ y))) \circ (\lambda s. \text{nnth } s \ 0)) \ x$
using assms 3 4 in-nset-conv-nnth[of $-\ (\text{nfilter } f \ (\text{ndropns } xs))$]
by simp
 $(\text{metis in-nset-conv-nnth ndropn-0 ndropn-nfirst nlength-nmap nnth-nmap})$
have 9: $\exists x \in \text{nset } (\text{ndropns } xs). (\lambda ys. f \ ys) \ x$
using assms **by** (simp add: sfxfilter-help)
have 10: $\exists x \in \text{nset } (\text{ndropns } xs). ((\lambda y. w \ (NNil \ y)) \circ (\lambda s. \text{nnth } s \ 0)) \ x \wedge (\lambda ys. f \ ys) \ x$
using 8 9
using exist-conj[of $f \ (\text{ndropns } xs) \ ((\lambda y. w \ (NNil \ y)) \circ (\lambda s. \text{nnth } s \ 0))$] **by** fastforce
have 11: $\text{nfilter } ((\lambda y. w \ (NNil \ y)) \circ (\lambda s. \text{nnth } s \ 0)) \ (\text{nfilter } (\lambda ys. f \ ys) \ (\text{ndropns } xs)) =$
 $\text{nfilter } (\lambda zs. ((\lambda y. w \ (NNil \ y)) \circ (\lambda s. \text{nnth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) \ (\text{ndropns } xs)$
using nfilter-nfilter[of $(\lambda ys. f \ ys) \ (\text{ndropns } xs) \ ((\lambda y. w \ (NNil \ y)) \circ (\lambda s. \text{nnth } s \ 0))$]
8 10 **by** blast
have 12: $\exists i \leq \text{nlength } xs. (\lambda zs. ((\lambda y. w \ (NNil \ y)) \circ (\lambda s. \text{nnth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) \ (\text{ndropn } i \ xs)$
by (metis 10 in-nset-ndropns)
have 13: $(\text{nfilter } (\lambda zs. ((\lambda y. w \ (NNil \ y)) \circ (\lambda s. \text{nnth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) \ (\text{ndropns } xs))$
 $= (\text{sfxfilt } xs \ (\lambda zs. ((\lambda y. w \ (NNil \ y)) \circ (\lambda s. \text{nnth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs))$
by (simp add: sfxfilt-def)
have 14: $\exists i \leq \text{nlength } xs. (\lambda zs. ((\lambda y. w \ (NNil \ y)) \circ (\lambda s. \text{nnth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) \ (\text{ndropn } i \ xs)$
using 12 **by** blast
have 15: $\text{nmap } (\lambda s. \text{nnth } s \ 0)$
 $((\text{sfxfilt } xs \ (\lambda zs. ((\lambda y. w \ (NNil \ y)) \circ (\lambda s. \text{nnth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs)) \) =$

```

    pifilt xs (λ zs. ((λy. w (NNil y))○(λs. nnth s 0)) zs ∧ (λ ys. f ys) zs)
  using 14 by (simp add: sxfilt-pifilt)
  have 16: ∧xs . (λ zs. ((λy. w (NNil y))○(λs. nnth s 0)) zs ∧ (λ ys. f ys) zs) xs =
    (LIFT(f ∧ init w)) xs
  by (simp add: init-defs) (metis ndropn-0 ndropn-nfirst)
  have 17: pifilt xs (λ zs. ((λy. w (NNil y))○(λs. nnth s 0)) zs ∧ (λ ys. f ys) zs) =
    pifilt xs (LIFT(f ∧ init w))
  using 16 by presburger
  show ?thesis
  using 11 13 15 17 2 3 4 7 by auto
qed

```

lemma *PiStatesem*:

```

  (σ ⊨ (init w) Π f) =
  ((∃ i ≤ nlength σ. w (NNil (nnth σ i))) ∧ f (nfilter (λy. w ((NNil y))) σ))
  by (simp add: pi-d-def init-defs ndropn-nfirst pifilt-init)

```

2.9.3 Soundness of Axioms

PiK

lemma *PiKsem*:

```

  σ ⊨ f1 Πu ( f ⟶ g ) ⟶ ( f1 Πu f ⟶ f1 Πu g )
  by (simp add: upi-d-def init-defs pi-d-def) auto

```

lemma *PiK*:

```

  ⊢ f1 Πu ( f ⟶ g ) ⟶ ( f1 Πu f ⟶ f1 Πu g )
  using PiKsem Valid-def by blast

```

PiDc

lemma *PiDcsem*:

```

  σ ⊨ f Π g ⟶ f Πu g
  by (simp add: upi-d-def init-defs pi-d-def)

```

lemma *PiDc*:

```

  ⊢ f Π g ⟶ f Πu g
  using PiDcsem Valid-def by blast

```

PiN

lemma *PiN*:

```

  assumes ⊢ g
  shows ⊢ f Πu g
  using assms by (simp add: Valid-def pi-d-def upi-d-def)

```

PiTrueEqvDiamond

lemma *PiTrueEqvDiamond*:

```

  ⊢ f Π # True = ◇ f
  by (simp add: Valid-def pi-d-def itl-defs)

```

PiOr

lemma *PiOr*:

$\vdash f \Pi (g1 \vee g2) = (f \Pi g1 \vee f \Pi g2)$

by (*simp add: Valid-def pi-d-def*) *blast*

UPiFalseEqvBoxNot:

lemma *UPiFalseEqvBoxNot*:

$\vdash f \Pi^u \#False = \square (\neg f)$

by (*simp add: Valid-def upi-d-def pi-d-def itl-defs*)

BoxEqvImpPiEqv

lemma *BoxEqvImpPiEqvsem*:

assumes $(\sigma \models \square (f1 = f2))$

shows $(\sigma \models (f1 \Pi g = f2 \Pi g))$

proof –

show $(\sigma \models (f1 \Pi g = f2 \Pi g))$

proof –

have 1: $\forall n \leq \text{nlength } \sigma. f1 (\text{ndropn } n \sigma) = f2 (\text{ndropn } n \sigma)$

using *assms* **by** (*simp add: itl-defs*)

have 2: $(\sigma \models (f1 \Pi g)) = ((\exists i \leq \text{nlength } \sigma. f1 (\text{ndropn } i \sigma)) \wedge g (\text{pifilt } \sigma f1))$

by (*simp add: pi-d-def*)

have 3: $(\exists i \leq \text{nlength } \sigma. f1 (\text{ndropn } i \sigma)) = (\exists i \leq \text{nlength } \sigma. f2 (\text{ndropn } i \sigma))$

using 1 **by** *blast*

have 6: $(\exists i \leq \text{nlength } \sigma. f1 (\text{ndropn } i \sigma)) \implies$

$\{ (\text{ndropn } n \sigma) \mid n. n \leq \text{nlength } \sigma \wedge f1 (\text{ndropn } n \sigma) \} =$

$\{ (\text{ndropn } n \sigma) \mid n. n \leq \text{nlength } \sigma \wedge f2 (\text{ndropn } n \sigma) \}$

using 1 **by** *blast*

have 7: $(\exists i \leq \text{nlength } \sigma. f1 (\text{ndropn } i \sigma)) \implies (\text{sfxfilt } \sigma f1) = (\text{sfxfilt } \sigma f2)$

by (*metis 1 in-nset-ndropns nfilter-cong sfxfilt-def*)

have 8: $((\exists i \leq \text{nlength } \sigma. f1 (\text{ndropn } i \sigma)) \wedge g (\text{pifilt } \sigma f1)) =$

$((\exists i \leq \text{nlength } \sigma. f2 (\text{ndropn } i \sigma)) \wedge g (\text{pifilt } \sigma f2))$

by (*metis 3 7 pifilt-def*)

show *?thesis*

by (*simp add: 8 pi-d-def*)

qed

qed

lemma *BoxEqvImpPiEqv*:

$\vdash \square (f1 = f2) \longrightarrow (f1 \Pi g = f2 \Pi g)$

using *BoxEqvImpPiEqvsem* **by** (*simp add: Valid-def,auto*)

PiDiamondImpDiamond

lemma *PiDiamondImpDiamondsem*:

$\sigma \models f \Pi (\Diamond (\text{init } w)) \longrightarrow \Diamond (\text{init } w)$

by (*simp add: Valid-def pi-d-def itl-defs ndropn-nfirst*)

(*metis pifilt-nnth*)

lemma *PiDiamondImp*:

$\vdash f \Pi (\Diamond (init\ w)) \longrightarrow \Diamond (init\ w)$
using *PiDiamondImpDiamondsem Valid-def by blast*

PiAssoc

lemma *PiAssocsem1:*

assumes $i \leq nlength\ \sigma$
 $f\ (ndropn\ i\ \sigma)$
 $ia \leq nlength\ (pifilt\ \sigma\ f)$
 $w\ (NNil\ (nnth\ (pifilt\ \sigma\ f)\ ia))$
shows $\exists i \leq nlength\ \sigma. f\ (ndropn\ i\ \sigma) \wedge w\ (NNil\ (nnth\ (ndropn\ i\ \sigma)\ 0))$
proof –
have 1: $(nnth\ (pifilt\ \sigma\ f)\ ia) = (nnth\ (nmap\ (\lambda s. nnth\ s\ 0)\ (sfxfilt\ \sigma\ f))\ ia)$
using *assms(1) assms(2) assms(3) sfxfilt-pifilt-nnth by blast*
have 2: $(nnth\ (nmap\ (\lambda s. nnth\ s\ 0)\ (sfxfilt\ \sigma\ f))\ ia) =$
 $(\lambda s. nnth\ s\ 0)\ (nnth\ (sfxfilt\ \sigma\ f)\ ia)$
by *(metis assms(3) nlength-nmap nnth-nmap pifilt-def)*
have 3: $f\ (nnth\ (sfxfilt\ \sigma\ f)\ ia)$
using *sfxfilt-nnth*
by *(metis assms(1) assms(2) assms(3) nlength-nmap pifilt-def)*
have 4: $nnth\ (sfxfilt\ \sigma\ f)\ ia = ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ ia)\ \sigma$
using *sfxfilt-pifilt-nnth-ndropn assms(1) assms(2) assms(3) by blast*
have 5: $w\ (NNil\ (nnth\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ ia)\ \sigma)\ 0))$
using 1 2 4 *assms(4) by auto*
show *?thesis by (metis 3 4 5 enat-ile le-cases ndropn-all)*
qed

lemma *PiAssocsem2:*

assumes $i \leq nlength\ \sigma$
 $f\ (ndropn\ i\ \sigma)$
 $w\ (NNil\ (nnth\ \sigma\ i))$
shows $\exists j \leq nlength\ (pifilt\ \sigma\ f). w\ (NNil\ (nnth\ (pifilt\ \sigma\ f)\ j))$
proof –
have 1: $\exists j \leq nlength\ (sfxfilt\ \sigma\ f). f\ (nnth\ (sfxfilt\ \sigma\ f)\ j)$
using *assms pifilt-exists by blast*
have 2: $(LIFT\ (init\ w))\ (ndropn\ i\ \sigma)$
by *(simp add: assms init-defs ndropn-nfirst)*
have 3: $\exists j \leq nlength\ (sfxfilt\ \sigma\ (LIFT\ (init\ w))). (LIFT\ (init\ w))\ (nnth\ (sfxfilt\ \sigma\ (LIFT\ (init\ w)))\ j)$
using *pifilt-exists 2 assms by blast*
have 4: $(LIFT\ (f \wedge init\ w))\ (ndropn\ i\ \sigma)$
by *(simp add: 2 assms)*
have 5: $\exists j \leq nlength\ (sfxfilt\ \sigma\ (LIFT\ (f \wedge init\ w))).$
 $(LIFT\ (f \wedge init\ w))\ (nnth\ (sfxfilt\ \sigma\ (LIFT\ (f \wedge init\ w)))\ j)$
using *pifilt-exists 4 assms by blast*
have 6: $\exists i \leq nlength\ \sigma. ndropn\ i\ \sigma \models f \wedge init\ w$
using 4 *assms by blast*
have 7: $\exists j \leq nlength\ (sfxfilt\ \sigma\ (LIFT\ ((f \wedge init\ w) \vee (f \wedge \neg (init\ w))))).$
 $(LIFT\ (f \wedge init\ w))\ (nnth\ (sfxfilt\ \sigma\ (LIFT\ ((f \wedge init\ w) \vee (f \wedge \neg (init\ w))))\ j)$
using 6 *sfxfilt-nnth-or[of σ $LIFT\ (f \wedge init\ w)$ $LIFT\ (f \wedge \neg (init\ w))$]*
by auto

have 8: $\bigwedge \sigma . (\sigma \models ((f \wedge \text{init } w) \vee (f \wedge \neg (\text{init } w)))) = (\sigma \models f)$
by *auto*
have 9: $(\text{sfxfilt } \sigma (\text{LIFT}((f \wedge \text{init } w) \vee (f \wedge \neg (\text{init } w)))) =$
 $(\text{sfxfilt } \sigma f)$
using 8 **by** (*simp add: sfxfilt-def*)
have 10: $\exists j \leq \text{nlength } (\text{sfxfilt } \sigma f).$
 $(\text{LIFT}(f \wedge \text{init } w)) (\text{nnth } (\text{sfxfilt } \sigma f) j)$
using 7 9 **by** *auto*
have 11: $\text{nlength } (\text{sfxfilt } \sigma f) = \text{nlength } (\text{pifilt } \sigma f)$
by (*simp add: pifilt-def*)
have 12: $\exists j \leq \text{nlength } (\text{pifilt } \sigma f).$
 $(\text{LIFT}(\text{init } w)) (\text{nnth } (\text{sfxfilt } \sigma f) j)$
using 10 11 **by** *auto*
from 12 11 **show** *?thesis unfolding pifilt-def init-defs*
by (*metis nlast-NNil nnth-nmap ntaken-0 ntaken-nlast*)
qed

lemma *PiAssocsema:*

$((\exists i \leq \text{nlength } \sigma. f (\text{ndropn } i \sigma)) \wedge$
 $(\exists i \leq \text{nlength } (\text{pifilt } \sigma f). w (\text{NNil } (\text{nnth } (\text{ndropn } i (\text{pifilt } \sigma f)) 0)))) =$
 $(\exists i \leq \text{nlength } \sigma. f (\text{ndropn } i \sigma) \wedge w (\text{NNil } (\text{nnth } (\text{ndropn } i \sigma) 0)))$
using *PiAssocsem1[of - $\sigma f - w$] PiAssocsem2[of - $\sigma f w$]* **by** *fastforce*

lemma *PiAssocsemb:*

$((\exists i \leq \text{nlength } \sigma. f (\text{ndropn } i \sigma)) \wedge$
 $(\exists i \leq \text{nlength } (\text{pifilt } \sigma f). (\text{LIFT}(\text{init } w)) (\text{ndropn } i (\text{pifilt } \sigma f)))) =$
 $(\exists i \leq \text{nlength } \sigma. (\text{LIFT}(f \wedge \text{init } w)) (\text{ndropn } i \sigma))$
using *PiAssocsem1[of - $\sigma f - w$] PiAssocsem2[of - $\sigma f w$]*
by (*simp add: init-defs ndropn-nfirst*) *blast*

lemma *PiAssocsem:*

$\sigma \models f \Pi ((\text{init } w) \Pi g) = (f \wedge (\text{init } w)) \Pi g$
proof (*auto simp add: pi-d-def init-defs*)
fix *i*
fix *ia*
assume *a0*: $g (\text{pifilt } (\text{pifilt } \sigma f) (\text{LIFT}(\text{init } w)))$
assume *a1*: $(\text{enat } i) \leq \text{nlength } \sigma$
assume *a2*: $f (\text{ndropn } i \sigma)$
assume *a3*: $(\text{enat } ia) \leq \text{nlength } (\text{pifilt } \sigma f)$
assume *a4*: $w (\text{NNil } (\text{nfirst } (\text{ndropn } ia (\text{pifilt } \sigma f))))$
show $\exists i. \text{enat } i \leq \text{nlength } \sigma \wedge f (\text{ndropn } i \sigma) \wedge w (\text{NNil } (\text{nfirst } (\text{ndropn } i \sigma)))$
using *a0 a1 a2 a3 a4 PiAssocsem1*
by (*metis ndropn-nfirst nnth-zero-ndropn*)
next
fix *i*
fix *ia*
assume *a0*: $g (\text{pifilt } (\text{pifilt } \sigma f) (\text{LIFT}(\text{init } w)))$
assume *a1*: $(\text{enat } i) \leq \text{nlength } \sigma$
assume *a2*: $f (\text{ndropn } i \sigma)$


```

assume a3: (enat ia) ≤ nlength (pifilt σ f)
assume a4: w (NNil (nfirst (ndropn ia (pifilt σ f))))
show g (pifilt σ (LIFT(f ∧ init w)))
  using a0 a1 a2 a3 a4 by (metis ndropn-nfirst nnth-zero-ndropn pifilt-pifilt)
next
fix i
assume a0: g (pifilt σ (LIFT(f ∧ init w)))
assume a1: (enat i) ≤ nlength σ
assume a2: f (ndropn i σ)
assume a3: w (NNil (nfirst (ndropn i σ)))
show ∃ i. enat i ≤ nlength (pifilt σ f) ∧ w (NNil (nfirst (ndropn i (pifilt σ f))))
  using a0 a1 a2 a3 by (metis PiAssocsem2 ndropn-nfirst)
next
fix i
assume a0: g (pifilt σ (LIFT(f ∧ init w)))
assume a1: (enat i) ≤ nlength σ
assume a2: f (ndropn i σ)
assume a3: w (NNil (nfirst (ndropn i σ)))
show g (pifilt (pifilt σ f) (LIFT(init w)))
  using a0 a1 a2 a3 by (metis PiAssocsem2 ndropn-nfirst nnth-zero-ndropn pifilt-pifilt)
qed

```

```

lemma PiAssoc:
  ⊢ f Π ((init w) Π g) = (f ∧ (init w)) Π g
  using PiAssocsem Valid-def by blast

```

PiNotEqvDiamondAndNotPi

```

lemma PiNotEqvDiamondAndNotPisem:
  σ ⊨ f Π (¬ g) = (◇ f ∧ ¬(f Π g))
by (simp add: pi-d-def itl-defs ) blast

```

```

lemma PiNotEqvDiamondAndNotPi:
  ⊢ f Π (¬ g) = (◇ f ∧ ¬(f Π g))
using PiNotEqvDiamondAndNotPisem Valid-def by blast

```

PiSChopDist

```

lemma PiSChopDistsema:
  assumes (σ ⊨ (init w) Π (g ∧ h))
  shows (σ ⊨ ((init w) Π g) ∧ ((init w) ∧ ((init w) Π h)))
proof –
  have 1: (σ ⊨ (init w) Π (g ∧ h))
    using assms by auto
  have 2: ((∃ i ≤ nlength σ. (LIFT(init w)) (ndropn i σ)) ∧
    ((pifilt σ (LIFT(init w))) ⊨ g ∧ h))
    using 1 by (simp add: pi-d-def)
  have 3: (∃ i ≤ nlength σ. (LIFT(init w)) (ndropn i σ))
    using 2 by auto
  have 4: ((pifilt σ (LIFT(init w))) ⊨ g ∧ h)

```

```

using 2 by auto
have 5:  $nfilter (\lambda y. w ((NNil y))) \sigma \models g \smallfrown h$ 
using pfilt-init by (metis 4)
have 6:  $(\exists n. (enat\ n) \leq nlength\ (nfilter\ (\lambda y. w\ (NNil\ y))\ \sigma)) \wedge$ 
 $g\ (ntaken\ n\ (nfilter\ (\lambda y. w\ (NNil\ y))\ \sigma)) \wedge$ 
 $h\ (ndropn\ n\ (nfilter\ (\lambda y. w\ (NNil\ y))\ \sigma))$ 
using 5 by (simp add: itl-defs)
obtain  $n$  where 7:  $(enat\ n) \leq nlength\ (nfilter\ (\lambda y. w\ (NNil\ y))\ \sigma) \wedge$ 
 $g\ (ntaken\ n\ (nfilter\ (\lambda y. w\ (NNil\ y))\ \sigma)) \wedge$ 
 $h\ (ndropn\ n\ (nfilter\ (\lambda y. w\ (NNil\ y))\ \sigma))$ 

using 6 by auto
have 8:  $\exists i \leq nlength\ \sigma. w\ (NNil\ (nnth\ \sigma\ i))$ 
using 3 using init-defs PiStatesem assms by blast
have 9:  $\exists x \in nset\ \sigma. w\ (NNil\ x)$ 
using 8 by (simp add: pfiltinit-help)
have 10:  $(ntaken\ n\ (nfilter\ (\lambda y. w\ (NNil\ y))\ \sigma)) =$ 
 $(nfilter\ (\lambda y. w\ (NNil\ y))\ (ntaken\ ((nnth\ (nkfilter\ (\lambda y. w\ (NNil\ y))\ 0\ \sigma)\ n)\ \sigma)\ \sigma))$ 
by (simp add: 7 9 nfilter-nkfilter-ntaken-1 nkfilter-nlength)
have 11:  $(ndropn\ n\ (nfilter\ (\lambda y. w\ (NNil\ y))\ \sigma)) =$ 
 $(nfilter\ (\lambda y. w\ (NNil\ y))\ (ndropn\ ((nnth\ (nkfilter\ (\lambda y. w\ (NNil\ y))\ 0\ \sigma)\ n)\ \sigma)\ \sigma))$ 
by (simp add: 7 9 nfilter-nkfilter-ndropn-1 nkfilter-nlength)
have 12:  $g\ (nfilter\ (\lambda y. w\ (NNil\ y))\ (ntaken\ ((nnth\ (nkfilter\ (\lambda y. w\ (NNil\ y))\ 0\ \sigma)\ n)\ \sigma)\ \sigma))$ 
using 10 7 by auto
have 13:  $h\ (nfilter\ (\lambda y. w\ (NNil\ y))\ (ndropn\ ((nnth\ (nkfilter\ (\lambda y. w\ (NNil\ y))\ 0\ \sigma)\ n)\ \sigma)\ \sigma))$ 
using 11 7 by auto
have 14:  $((nnth\ (nkfilter\ (\lambda y. w\ (NNil\ y))\ 0\ \sigma)\ n)\ \sigma) \leq nlength\ \sigma$ 
using 7 9 nkfilter-upperbound[of  $\sigma\ (\lambda y. w\ (NNil\ y)) - 0$ ]
by (metis gen-nlength-def nkfilter-nlength nlength-code)
have 15:  $w\ (NNil\ (nnth\ (ndropn\ ((nnth\ (nkfilter\ (\lambda y. w\ (NNil\ y))\ 0\ \sigma)\ n)\ \sigma)\ 0))$ 
using 7 9 nkfilter-nmap-nfilter[of  $\sigma\ \lambda x. w\ (NNil\ x)$ ] nkfilter-nnth-aa[of  $\sigma\ \lambda x. w\ (NNil\ x)$ ]
by simp
 $(metis\ (mono-tags,\ lifting)\ 7\ nkfilter-nlength\ nnth-nmap)$ 
have 16:  $(\exists i. (enat\ i) \leq nlength\ (ntaken\ ((nnth\ (nkfilter\ (\lambda y. w\ (NNil\ y))\ 0\ \sigma)\ n)\ \sigma)\ \sigma) \wedge$ 
 $w\ (NNil\ (nnth\ (ndropn\ i\ (ntaken\ ((nnth\ (nkfilter\ (\lambda y. w\ (NNil\ y))\ 0\ \sigma)\ n)\ \sigma)\ 0))))$ 
using 14 15
by (metis le-cases min.orderE ndropn-0 ndropn-nfirst ntaken-nlength ntaken-nnth)
have 17:  $(\exists i. (enat\ i) \leq nlength\ (ndropn\ ((nnth\ (nkfilter\ (\lambda y. w\ (NNil\ y))\ 0\ \sigma)\ n)\ \sigma)\ \sigma) \wedge$ 
 $w\ (NNil\ (nnth\ (ndropn\ (i + ((nnth\ (nkfilter\ (\lambda y. w\ (NNil\ y))\ 0\ \sigma)\ n)\ \sigma)\ 0))))$ 
using 15 by (metis add.left-neutral zero-enat-def zero-le)
have 18:  $(\exists n. (enat\ n) \leq nlength\ \sigma \wedge$ 
 $(\exists i. (enat\ i) \leq nlength\ (ntaken\ n\ \sigma) \wedge w\ (NNil\ (nnth\ (ndropn\ i\ (ntaken\ n\ \sigma))\ 0)) \wedge$ 
 $g\ (nfilter\ (\lambda y. w\ ((NNil\ y)))\ (ntaken\ n\ \sigma)) \wedge$ 
 $w\ (NNil\ (nnth\ (ndropn\ n\ \sigma)\ 0)) \wedge$ 
 $(\exists i. (enat\ i) \leq nlength\ (ndropn\ n\ \sigma) \wedge w\ (NNil\ (nnth\ (ndropn\ (i + n)\ \sigma)\ 0))) \wedge$ 
 $h\ (nfilter\ (\lambda y. w\ ((NNil\ y)))\ (ndropn\ n\ \sigma)))$ 
using 12 13 14 15 16 17 by blast
have 181:  $(\exists n. (enat\ n) \leq nlength\ \sigma \wedge (\exists i. (enat\ i) \leq nlength\ (ntaken\ n\ \sigma) \wedge w\ (NNil\ (nnth\ (ntaken\ n\ \sigma)$ 
 $i)))$ 
using 18 by auto
have 182:  $(\exists n. (enat\ n) \leq nlength\ \sigma \wedge (\exists i. (enat\ i) \leq nlength\ (ndropn\ n\ \sigma) \wedge w\ (NNil\ (nnth\ \sigma\ (i + n))))$ 

```

```

) )
  using 18 by auto
have 19: ( $\exists n. (enat\ n) \leq nlength\ \sigma \wedge$ 
  ( $\exists i. (enat\ i) \leq nlength\ (ntaken\ n\ \sigma) \wedge w\ (NNil\ (nnth\ (ndropn\ i\ (ntaken\ n\ \sigma))\ 0)) \wedge$ 
   $g\ (pifilt\ (ntaken\ n\ \sigma)\ (LIFT(init\ w))) \wedge$ 
   $w\ (NNil\ (nnth\ (ndropn\ n\ \sigma)\ 0)) \wedge$ 
  ( $\exists i. (enat\ i) \leq nlength\ (ndropn\ n\ \sigma) \wedge w\ (NNil\ (nnth\ (ndropn\ (i + n)\ \sigma)\ 0)) \wedge$ 
   $h\ (pifilt\ (ndropn\ n\ \sigma)\ (LIFT(init\ w))))$ )
  using 18 pifilt-init[of - w] by auto
have 20: ( $(\exists n. (enat\ n) \leq nlength\ \sigma \wedge$ 
  ( $\exists i. (enat\ i) \leq nlength\ (ntaken\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ i\ (ntaken\ n\ \sigma))) \wedge$ 
   $g\ (pifilt\ (ntaken\ n\ \sigma)\ (LIFT(init\ w))) \wedge$ 
   $(LIFT(init\ w))\ (ndropn\ n\ \sigma) \wedge$ 
  ( $\exists i. (enat\ i) \leq nlength\ (ndropn\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ (i + n)\ \sigma))$ 
   $\wedge h\ (pifilt\ (ndropn\ n\ \sigma)\ (LIFT(init\ w))))$ )
  using 19
  by (metis init-defs ndropn-nfirst nnth-zero-ndropn ntaken-0)
have 21: ( $\sigma \models ((init\ w) \Pi g) \frown ((init\ w) \wedge ((init\ w) \Pi h))$ )
  using 20 by (simp add: add.commute itl-defs ndropn-ndropn pi-d-def)
show ?thesis by (simp add: 21)
qed

```

lemma *PiSchopDistsemb*:

```

  assumes ( $\sigma \models ((init\ w) \Pi g) \frown ((init\ w) \wedge ((init\ w) \Pi h))$ )
  shows ( $\sigma \models (init\ w) \Pi (g \frown h)$ )
proof -
  have 1: ( $\sigma \models ((init\ w) \Pi g) \frown ((init\ w) \wedge ((init\ w) \Pi h))$ )
    using assms by auto
  have 2: ( $\exists n. (enat\ n) \leq nlength\ \sigma \wedge$ 
    ( $(ntaken\ n\ \sigma) \models ((init\ w) \Pi g) \wedge$ 
    ( $(ndropn\ n\ \sigma) \models ((init\ w) \wedge ((init\ w) \Pi h))$ ))
    using assms schop-defs by blast
  obtain n where 3: ( $(enat\ n) \leq nlength\ \sigma \wedge ((ntaken\ n\ \sigma) \models ((init\ w) \Pi g)) \wedge$ 
    ( $(ndropn\ n\ \sigma) \models ((init\ w) \wedge ((init\ w) \Pi h))$ )
    using 2 by auto
  have 4: ( $(\exists i. (enat\ i) \leq nlength\ (ntaken\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ i\ (ntaken\ n\ \sigma))) \wedge$ 
    ( $(pifilt\ (ntaken\ n\ \sigma)\ (LIFT(init\ w))) \models g$ )
    )
    by (meson 3 pi-d-def)
  have 5: ( $\exists i. (enat\ i) \leq nlength\ (ntaken\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ i\ (ntaken\ n\ \sigma))$ )
    using 4 by auto
  have 6:  $g\ (pifilt\ (ntaken\ n\ \sigma)\ (LIFT(init\ w)))$ 
    using 4 by auto
  have 7:  $g\ (nfilter\ (\lambda y. w\ (NNil\ y))\ (ntaken\ n\ \sigma))$ 
    using 5 6 pifilt-init by (metis)
  have 8: ( $(\exists i. (enat\ i) \leq nlength\ (ndropn\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ i\ (ndropn\ n\ \sigma))) \wedge$ 
    ( $(pifilt\ (ndropn\ n\ \sigma)\ (LIFT(init\ w))) \models h$ )
    )
    by (metis 3 intensional-rews(3) pi-d-def)
  have 9: ( $\exists i. (enat\ i) \leq nlength\ (ndropn\ n\ \sigma) \wedge (LIFT(init\ w))\ (ndropn\ i\ (ndropn\ n\ \sigma))$ )

```

```

using 8 by auto
have 10:  $h$  ( $\text{pifilt}$  ( $\text{ndropn } n \sigma$ ) ( $\text{LIFT}(\text{init } w)$ ))
using 8 by auto
have 11:  $h$  ( $\text{nfilter}$  ( $\lambda y. w ((\text{NNil } y))$ ) ( $\text{ndropn } n \sigma$ ))
using 10 9  $\text{pifilt-init}$  by metis
have 12: ( $\lambda y. w ((\text{NNil } y))$ ) ( $\text{nlast}$  ( $\text{ntaken } n \sigma$ ))
by (metis 3  $\text{init-defs}$   $\text{intensional-rews}(3)$   $\text{ndropn-nfirst}$   $\text{ntaken-0}$   $\text{ntaken-nlast}$ )
have 13:  $\exists x \in \text{nset } \sigma. (\lambda y. w ((\text{NNil } y))) x$ 
using 12 3 by (metis  $\text{in-nset-conv-nnth}$   $\text{ntaken-nlast}$ )
have 14:  $\exists x \in \text{nset} (\text{ntaken } n \sigma) . (\lambda y. w ((\text{NNil } y))) x$ 
using 12 using  $\text{nfinite-ntaken}$   $\text{nset-nlast}$  by blast
have 15:  $\exists x \in \text{nset} (\text{ndropn } n \sigma) . (\lambda y. w ((\text{NNil } y))) x$ 
by (metis 12 3  $\text{dual-order.order-iff-strict}$   $\text{ndropn-Suc-conv-ndropn}$   $\text{ndropn-nlast}$ 
 $\text{nellist.set-intros}(1)$   $\text{nellist.set-intros}(2)$   $\text{nfinite-ntaken}$   $\text{ntaken-all}$   $\text{ntaken-nlast}$   $\text{the-enat.simps}$ )
have 16: ( $\text{nfilter}$  ( $\lambda y. w ((\text{NNil } y))$ ) ( $\text{ntaken } n \sigma$ )) =
 $\text{ntaken}$  ( $\text{the-enat}$  ( $\text{nlength}$  ( $\text{nfilter}$  ( $\lambda y. w ((\text{NNil } y))$ ) ( $\text{ntaken } n \sigma$ ))))
( $\text{nfilter}$  ( $\lambda y. w ((\text{NNil } y))$ )  $\sigma$ )
using 12 13 14 15 3  $\text{nfilter-chop1-ntaken}[of\ n\ \sigma\ (\lambda y. w (\text{NNil } y))]$  by auto
have 17: ( $\text{nfilter}$  ( $\lambda y. w ((\text{NNil } y))$ ) ( $\text{ndropn } n \sigma$ )) =
 $\text{ndropn}$  ( $\text{the-enat}$  ( $\text{nlength}$  ( $\text{nfilter}$  ( $\lambda y. w ((\text{NNil } y))$ ) ( $\text{ntaken } n \sigma$ ))))
( $\text{nfilter}$  ( $\lambda y. w ((\text{NNil } y))$ )  $\sigma$ )
using 12 13 14 15 3  $\text{nfilter-chop1-ndropn}[of\ n\ \sigma\ (\lambda y. w (\text{NNil } y))]$  by auto
have 18:  $\exists n. (\text{enat } n) \leq \text{nlength} (\text{nfilter} (\lambda y. w (\text{NNil } y)) \sigma) \wedge$ 
 $g (\text{ntaken } n (\text{nfilter} (\lambda y. w (\text{NNil } y)) \sigma)) \wedge$ 
 $h (\text{ndropn } n (\text{nfilter} (\lambda y. w (\text{NNil } y)) \sigma))$ 
by (metis 11 16 17 7  $\text{enat-the-enat}$   $\text{infinity-ileE}$   $\text{le-cases}$   $\text{ntaken-all}$ )
have 19:  $\text{nfilter} (\lambda y. w ((\text{NNil } y))) \sigma \models g \smallfrown h$ 
by ( $\text{simp add: 18}$   $\text{schop-defs}$ )
have 20: ( $\text{pifilt } \sigma$  ( $\text{LIFT}(\text{init } w)$ ))  $\models g \smallfrown h$ 
by (metis 19  $\text{pifilt-init}$ )
have 21: ( $\exists i. (\text{enat } i) \leq \text{nlength } \sigma \wedge (\text{LIFT}(\text{init } w)) (\text{ndropn } i \sigma)$ )
using 3 by auto
show ?thesis by ( $\text{simp add: 20 21}$   $\text{pi-d-def}$ )
qed

```

lemma PiSchopDistsem :

$\sigma \models (\text{init } w) \amalg (g \smallfrown h) = ((\text{init } w) \amalg g) \smallfrown ((\text{init } w) \wedge ((\text{init } w) \amalg h))$

using PiSchopDistsema PiSchopDistsemb unl-lift2 by blast

lemma PiSchopDist :

$\vdash (\text{init } w) \amalg (g \smallfrown h) = ((\text{init } w) \amalg g) \smallfrown ((\text{init } w) \wedge ((\text{init } w) \amalg h))$

using PiSchopDistsem Valid-def by blast

PiProp

lemma Pistate :

$(\sigma \models (\text{init } w) \amalg f) =$

$((\exists x \in \text{nset } \sigma. w (\text{NNil } x)) \wedge ((\text{nfilter} (\lambda y. w (\text{NNil } y)) \sigma) \models f))$

proof –

have 1: $(\sigma \models (\text{init } w) \amalg f) =$

($(\exists i \leq \text{nlength } \sigma. w (\text{NNil } (\text{nnth } \sigma i))) \wedge ((\text{pifilt } \sigma (\text{LIFT}(\text{init } w))) \models f)$)
 by (auto simp add: pi-d-def init-defs ndropn-nfirst)
 have 2: $(\exists i \leq \text{nlength } \sigma. (\text{LIFT}(\text{init } w)) (\text{ndropn } i \sigma)) =$
 $(\exists i \leq \text{nlength } \sigma. w (\text{NNil } (\text{nnth } \sigma i)))$
 by (auto simp add: init-defs ndropn-nfirst)
 have 3: $(\exists i \leq \text{nlength } \sigma. w (\text{NNil } (\text{nnth } \sigma i))) \longrightarrow$
 $(\text{pifilt } \sigma (\text{LIFT}(\text{init } w))) = (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma)$
 using pifilt-init using 2 by blast
 have 4: $(\exists i. (\text{enat } i) \leq \text{nlength } \sigma \wedge w (\text{NNil } (\text{nnth } \sigma i))) = (\exists x \in \text{nset } \sigma. w (\text{NNil } x))$
 using in-nset-conv-nnth by force
 show ?thesis
 using 1 3 4 by auto
 qed

lemma PiPropsem1a:

$(\sigma \models f \Pi \$p) =$
 $((\exists i. (\text{enat } i) \leq \text{nlength } \sigma \wedge f (\text{ndropn } i \sigma)) \wedge p (\text{nnth } \sigma (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 0)))$
 using sfxfilt-pifilt-nnth-ndropn[of σf]
 by (simp add: pi-d-def current-val-d-def pifilt-def)
 $(\text{metis ndropn-0 ndropn-nfirst nnth-nmap zero-enat-def zero-order}(1))$

lemma PiPropsem2a:

$(\sigma \models (\neg f) \mathcal{U} (f \wedge \$p)) =$
 $(\exists k \leq \text{nlength } \sigma. f (\text{ndropn } k \sigma) \wedge p (\text{nnth } \sigma k) \wedge (\forall j < k. \neg f (\text{ndropn } j \sigma)))$
 by (simp add: until-d-def current-val-d-def ndropn-nfirst)

lemma PiPropsem3a:

assumes $(\sigma \models f \Pi \$p)$
 shows $(\sigma \models (\neg f) \mathcal{U} (f \wedge \$p))$
 proof –
 have 1: $((\exists i. (\text{enat } i) \leq \text{nlength } \sigma \wedge f (\text{ndropn } i \sigma)) \wedge$
 $p (\text{nnth } \sigma (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 0)))$
 using assms PiPropsem1a by auto
 have 2: $\exists x \in \text{nset}(\text{ndropns } \sigma). f x$
 using 1 by (simp add: sfxfilter-help)
 have 3: $\forall x \in \text{nset}(\text{nkfilter } f 0 (\text{ndropns } \sigma)). f (\text{nnth } (\text{ndropns } \sigma) x)$
 using 2 nkfilter-holds by fastforce
 have 31: $(\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 0) \leq \text{nlength } \sigma$
 by (metis 1 2 dual-order.strict-trans2 enat-ord-simps(2) leI ndropns-nnth nkfilter-not-before)
 have 4: $f (\text{ndropn } (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 0) \sigma)$
 by (metis 3 31 in-nset-ndropns in-nset-ndropns-nhd ndropn-0 ndropns-nnth zero-enat-def zero-le)
 have 5: $(\forall j < (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 0). \neg f (\text{ndropn } j \sigma))$
 using nkfilter-not-before[of $\text{ndropns } \sigma f$] 2
 by (simp add: 31 less-imp-le ndropns-nnth order-less-le-subst2)
 have 6: $(\exists k \leq \text{nlength } \sigma. f (\text{ndropn } k \sigma) \wedge p (\text{nnth } \sigma k) \wedge (\forall j < k. \neg f (\text{ndropn } j \sigma)))$
 using 1 31 4 5 by blast
 show ?thesis using 6 PiPropsem2a by metis
 qed

lemma PiPropsem3b:

assumes $(\sigma \models (\neg f) \mathcal{U} (f \wedge \$p))$
shows $(\sigma \models f \Pi \$p)$
proof –
have 1: $(\exists k \leq \text{nlength } \sigma. f (\text{ndropn } k \sigma) \wedge p (\text{nnth } \sigma k) \wedge (\forall j < k. \neg f (\text{ndropn } j \sigma)))$
using *assms PiPropsem2a* **by** *auto*
obtain *k* **where** 2: $k \leq \text{nlength } \sigma \wedge f (\text{ndropn } k \sigma) \wedge p (\text{nnth } \sigma k) \wedge (\forall j < k. \neg f (\text{ndropn } j \sigma))$
using 1 **by** *auto*
have 3: $(\exists i. (\text{enat } i) \leq \text{nlength } \sigma \wedge f (\text{ndropn } i \sigma))$
using 2 **by** *blast*
have 31: $\exists x \in \text{nset}(\text{ndropns } \sigma). f x$
using 2 *in-nset-ndropns* **by** *auto*
have 331: $(\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0) \leq \text{nlength } \sigma$
by (*metis* 31 *gen-nlength-def ndropns-nlength nkfilter-upperbound nlength-code zero-enat-def zero-le*)
have 32: $f (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0) \sigma)$
using *nkfilter-holds[of ndropns σ $f \ 0$] nkfilter-not-before[of ndropns σ f]*
by (*metis* 2 31 *ndropns-nfilter-nnth sfxfilt-def sfxfilt-pifilt-nnth-ndropn zero-enat-def zero-le*)
have 4: $p (\text{nnth } \sigma (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0))$
by (*metis* 2 31 32 *linorder-neqE-nat ndropns-nnth nkfilter-not-before*)
show ?thesis **using** 4 3 **by** (*simp add: PiPropsem1a*)
qed

lemma *PiPropsema*:
 $\sigma \models f \Pi \$p = (\neg f) \mathcal{U} (f \wedge \$p)$
using *PiPropsem3a PiPropsem3b unl-lift2* **by** *blast*

lemma *PiProp*:
 $\vdash f \Pi \$p = (\neg f) \mathcal{U} (f \wedge \$p)$
using *PiPropsema Valid-def* **by** *blast*

PiNext

lemma *PiNextsem1*:
 $(\sigma \models f \Pi (\bigcirc g)) =$
 $((\exists i \leq \text{nlength } \sigma. f (\text{ndropn } i \sigma)) \wedge$
 $0 < \text{nlength } (\text{nfilter } f (\text{ndropns } \sigma)) \wedge$
 $g (\text{ndropn } (\text{Suc } 0) (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f (\text{ndropns } \sigma)))))$
using *zero-enat-def* **by** (*simp add: pi-d-def itl-defs pifilt-def sfxfilt-def*)

lemma *PiNextsem2*:
 $(\sigma \models (\neg f) \mathcal{U} (f \wedge \bigcirc(f \Pi g))) =$
 $(\exists k \leq \text{nlength } \sigma.$
 $f (\text{ndropn } k \sigma) \wedge$
 $k < \text{nlength } \sigma \wedge$
 $(\exists i \leq \text{nlength } \sigma - \text{Suc } k. f (\text{ndropn } (\text{Suc } (i + k)) \sigma)) \wedge$
 $g (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f (\text{ndropns } (\text{ndropn } (\text{Suc } k) \sigma)))) \wedge (\forall j < k. \neg f (\text{ndropn } j \sigma)))$
by (*simp add: until-d-def itl-defs pi-d-def pifilt-def sfxfilt-def ndropn-ndropn add commute*)
(metis add.right-neutral antisym-conv2 enat.simps(3) enat-add-sub-same less-eqE zero-enat-def)

lemma *PiNextsem3*:
assumes $(\sigma \models f \Pi (\bigcirc g))$

shows $(\sigma \models (\neg f) \mathcal{U} (f \wedge \bigcirc(f \Pi g)))$
proof –
have 1: $((\exists i \leq \text{nlength } \sigma. f (\text{ndropn } i \ \sigma)) \wedge 0 < \text{nlength } (\text{nfilter } f (\text{ndropns } \sigma)) \wedge$
 $g (\text{ndropn } (\text{Suc } 0) (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f (\text{ndropns } \sigma))))$
using *assms* **by** (*simp add: PiNextsem1*)
have 2: $\exists x \in \text{nset}(\text{ndropns } \sigma). f x$
using 1 **by** (*simp add: sxfilter-help*)
have 3: $\forall x \in \text{nset}(\text{nkfilter } f \ 0 (\text{ndropns } \sigma)). f (\text{nnth } (\text{ndropns } \sigma) \ x)$
using 2 *nkfilter-holds* **by** *fastforce*
have 31: $(\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0) \leq \text{nlength } \sigma$
by (*metis 2 gen-nlength-def i0-lb ndropns-nlength nkfilter-upperbound nlength-code zero-enat-def*)
have 4: $f (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0) \ \sigma)$
by (*metis 3 31 i0-lb in-nset-conv-nnth ndropns-nnth zero-enat-def*)
have 41: $(\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 1) \in \text{nset}(\text{nkfilter } f \ 0 (\text{ndropns } \sigma))$
by (*metis 1 2 One-nat-def eSuc-enat ileI1 in-nset-conv-nnth nkfilter-nlength zero-enat-def*)
have 42: $(\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 1) \leq \text{nlength } (\text{ndropns } \sigma)$
by (*metis 2 41 gen-nlength-def in-nset-conv-nnth nkfilter-upperbound nlength-code*)
have 5: $f (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 1) \ \sigma)$
using 3 41 42 **by** (*metis ndropns-nlength ndropns-nnth*)
have 6: $(\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0) < (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 1)$
by (*metis 1 2 One-nat-def ileI1 nidx-nkfilter-expand nkfilter-nlength one-eSuc one-enat-def*)
have 7: $\text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0) \leq \text{nlength } \sigma$
by (*metis 42 6 Suc-leI dual-order.trans enat-ord-simps(1) ndropns-nlength*)
have 8: $0 < \text{nlength } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma))$
using 1 2 *nkfilter-nlength* **by** *fastforce*
have 9: $(\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 1) \leq \text{nlength } \sigma$
by (*metis 42 ndropns-nlength*)
have 10: $(\text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0)) \leq (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 1)$
using 6 *Suc-leI* **by** *blast*
have 11: $(\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 1) =$
 $(\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 1) - (\text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0)) +$
 $(\text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0))$
using 10 **by** *auto*
have 12: $(\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 1) - (\text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0)) \leq$
 $\text{nlength } \sigma - \text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0)$
using 9 *diff-le-mono* **by** (*metis enat-minus-mono1 idiff-enat-enat*)
have 13: $\exists i \leq \text{nlength } \sigma - \text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0).$
 $(\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 1) = (\text{Suc } (i + (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0)))$
using 11 12 **by** *auto*
have 14: $(\exists i \leq \text{nlength } \sigma - \text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0).$
 $f (\text{ndropn } (\text{Suc } (i + (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0))) \ \sigma)$
using 13 5 **by** *auto*
have 15: $(\text{ndropn } (\text{Suc } 0) (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f (\text{ndropns } \sigma)))) =$
 $(\text{nmap } (\lambda s. \text{nnth } s \ 0)$
 $(\text{nfilter } f (\text{ndropns } (\text{ndropn } (\text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0)) \ \sigma))))$
using 2 8
by (*metis One-nat-def ileI1 nfilter-ndropns-nmap nkfilter-nlength one-eSuc one-enat-def*)
have 16: $g (\text{nmap } (\lambda s. \text{nnth } s \ 0)$
 $(\text{nfilter } f (\text{ndropns } (\text{ndropn } (\text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ 0)) \ \sigma))))$
using 1 15 **by** *auto*

have 17: $\forall j < (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0). \neg f (\text{ndropn } j \ \sigma)$
using 2 31 *nkfilter-not-before*[of (*ndropns* σ) *f*]
by (*metis* *enat-ord-simps*(1) *less-imp-le ndropns-nnth order.trans*)
have 18: $(\exists k \leq \text{nlength } \sigma.$
 $f (\text{ndropn } k \ \sigma) \wedge$
 $k < \text{nlength } \sigma \wedge$
 $(\exists i \leq \text{nlength } \sigma - \text{Suc } k. f (\text{ndropn } (\text{Suc } (i + k)) \ \sigma)) \wedge$
 $g (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f (\text{ndropns } (\text{ndropn } (\text{Suc } k) \ \sigma)))) \wedge$
 $(\forall j < k. \neg f (\text{ndropn } j \ \sigma)))$
using 14 16 17 4 7 *Suc-ile-eq less-imp-le* **by** *blast*
show ?thesis **using** 18 **by** (*simp* *add: PiNextsem2*)
qed

lemma *PiNextsem4*:

assumes $(\sigma \models (\neg f) \ \mathcal{U} \ (f \wedge \bigcirc(f \ \Pi \ g)))$

shows $(\sigma \models f \ \Pi \ (\bigcirc \ g))$

proof –

have 1: $(\exists k \leq \text{nlength } \sigma.$
 $f (\text{ndropn } k \ \sigma) \wedge$
 $k < \text{nlength } \sigma \wedge$
 $(\exists i \leq \text{nlength } \sigma - \text{Suc } k. f (\text{ndropn } (\text{Suc } (i + k)) \ \sigma)) \wedge$
 $g (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f (\text{ndropns } (\text{ndropn } (\text{Suc } k) \ \sigma)))) \wedge$
 $(\forall j < k. \neg f (\text{ndropn } j \ \sigma)))$
using *assms* **by** (*simp* *add: PiNextsem2*)
obtain *k* **where** 2: $k \leq \text{nlength } \sigma \wedge f (\text{ndropn } k \ \sigma) \wedge k < \text{nlength } \sigma \wedge (\exists i \leq \text{nlength } \sigma - \text{Suc } k.$
 $f (\text{ndropn } (\text{Suc } (i + k)) \ \sigma)) \wedge$
 $g (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f (\text{ndropns } (\text{ndropn } (\text{Suc } k) \ \sigma)))) \wedge (\forall j < k. \neg f (\text{ndropn } j \ \sigma))$
using 1 **by** *auto*
have 3: $\exists x \in \text{nset } (\text{ndropns } \sigma). f \ x$
using 1 **using** *in-nset-ndropns* **by** *auto*
have 4: $\forall x \in \text{nset}(\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)). f \ (\text{nnth } (\text{ndropns } \sigma) \ x)$
using 3 *nkfilter-holds* **by** *fastforce*
have 41: $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0) \leq \text{nlength } \sigma$
by (*metis* 3 *gen-nlength-def le-cases le-zero-eq ndropns-nlength nkfilter-upperbound nlength-code zero-enat-def*)
have 5: $f (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0) \ \sigma)$
by (*metis* 4 41 *i0-lb in-nset-conv-nnth ndropns-nnth zero-enat-def*)
have 6: $0 < \text{nlength } (\text{nfilter } f (\text{ndropns } \sigma))$
using 2 3 *nfilter-nlength-zero-conv-a*[of (*ndropns* σ) *f*]
by *simp*
 $(\text{metis } (\text{no-types, lifting}) \text{ add.commute eSuc-enat enat.simps}(3) \text{ enat-add-sub-same}$
 $\text{ enat-less-enat-plusI2 ileI1 ileSS-Suc-eq less-add-Suc2 less-eqE ndropn-Suc-conv-ndropn}$
 $\text{ ndropn-nlength ndropns-nlength ndropns-nnth nlength-NCons})$
have 61: $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 1) \in \text{nset}(\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma))$
by (*metis* 3 6 *ileI1 in-nset-conv-nnth nkfilter-nlength one-eSuc one-enat-def*)
have 62: $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 1) \leq \text{nlength } (\text{ndropns } \sigma)$
by (*metis* 3 61 *gen-nlength-def in-nset-conv-nnth nkfilter-upperbound nlength-code*)
have 7: $f (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 1) \ \sigma)$
using 4 61 62 **by** (*metis* *ndropns-nlength ndropns-nnth*)
have 8: $(\exists i \leq \text{nlength } \sigma. f (\text{ndropn } i \ \sigma))$


```

using 1 by blast
have 9: g (ndropn (Suc 0) (nmap (λs. nnth s 0) (nfilter f (ndropns σ))))
  using 2 3 5 6 nfilter-not-before[of (ndropns σ) f] nfilter-ndropns-nmap[of σ f 0]
  by (metis One-nat-def ileI1 ndropns-nnth not-less-iff-gr-or-eq one-eSuc one-enat-def)
have 10: ((∃ i. (enat i) ≤ nlength σ ∧ f (ndropn i σ)) ∧
  0 < nlength (nfilter f (ndropns σ)) ∧
  g (ndropn (Suc 0) (nmap (λs. nnth s 0) (nfilter f (ndropns σ)))))
  using 6 8 9 by blast
show ?thesis using 10
  by (simp add: PiNextsem1)
qed

```

lemma *PiNextsem*:

```

(σ ⊨ f Π (○ g) = (¬ f) U (f ∧ ○(f Π g)))
using PiNextsem3 PiNextsem4 unl-lift2 by blast

```

lemma *PiNext*:

```

⊢ f Π (○ g) = (¬ f) U (f ∧ ○(f Π g))
using PiNextsem Valid-def by blast

```

PiUntil

lemma *PiUntilDistsem1*:

```

(σ ⊨ f Π (g U h)) =
  ((∃ i ≤ nlength σ. f (ndropn i σ)) ∧
  (∃ k ≤ nlength (nfilter f (ndropns σ)).
    h (ndropn k (nmap (λs. nnth s 0) (nfilter f (ndropns σ))))) ∧
  (∀ j < k. g (ndropn j (nmap (λs. nnth s 0) (nfilter f (ndropns σ))))))
by (simp add: pi-d-def pifilt-def sfxfilt-def until-d-def)

```

lemma *PiUntilDistsem2*:

```

(σ ⊨ ( f Π g ) U ( f Π h ) ) =
  (∃ k ≤ nlength σ.
    (∃ i ≤ nlength σ - k. f (ndropn (i + k) σ)) ∧
    h (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn k σ))))) ∧
  (∀ j < k. (∃ i ≤ nlength σ - j. f (ndropn (i + j) σ)) ∧
    g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ)))))
by (simp add: until-d-def pi-d-def pifilt-def sfxfilt-def ndropn-ndropn add commute)

```

lemma *cover*:

```

assumes (∃ i < k. ff(i) < (j::nat) ∧ j ≤ ff(Suc i))
  (∀ i < k. ff(i) < ff(Suc i))
shows ff(0) < j ∧ j ≤ ff(k)
using assms
proof (induct k arbitrary: j)
case 0
then show ?case by simp
next
case (Suc k)
then show ?case

```

```

proof –
  have 1:  $(\exists i < k. \text{ff}(i) < (j :: \text{nat}) \wedge j \leq \text{ff}(\text{Suc } i)) \vee (\text{ff}(k) < j \wedge j \leq \text{ff}(\text{Suc } k))$ 
    using Suc.prem1 less-SucE by blast
  have 2:  $(\forall i < k. \text{ff}(i) < \text{ff}(\text{Suc } i))$ 
    using Suc.prem2 by auto
  have 3:  $\text{ff } k < \text{ff}(\text{Suc } k)$ 
    by (simp add: Suc.prem2)
  have 4:  $(\text{ff}(0) < j \wedge j \leq \text{ff}(k)) \vee (\text{ff}(k) < j \wedge j \leq \text{ff}(\text{Suc } k))$ 
    using 1 2 Suc.hyps by blast
  have 41:  $\text{ff}(0) < j$ 
    using 2 4 Suc.hyps order-refl by force
  have 42:  $j \leq \text{ff}(\text{Suc } k)$ 
    using 3 4 by linarith
  have 5:  $\text{ff}(0) < j \wedge j \leq \text{ff}(\text{Suc } k)$ 
    by (simp add: 41 42)
  show ?thesis
    by (simp add: 5)
qed
qed

```

lemma *cover-a*:

```

assumes  $(\forall j. (j \leq \text{ff } 0) \vee (\exists i < k. \text{ff}(i) < (j :: \text{nat}) \wedge j \leq \text{ff}(\text{Suc } i)) \longrightarrow gg j)$ 
     $(\forall i < k. \text{ff}(i) < \text{ff}(\text{Suc } i))$ 
shows  $(\forall j < \text{ff } k. gg j)$ 
proof –
  have 1:  $(\forall j < \text{ff } 0. gg j)$ 
    by (simp add: assms(1))
  have 2:  $(\forall j. \text{ff } 0 < j \wedge j \leq \text{ff } k \longrightarrow gg j)$ 
    proof
      fix j
      show  $\text{ff } 0 < j \wedge j \leq \text{ff } k \longrightarrow gg j$ 
        using assms
        proof (induct k arbitrary: j)
          case 0
          then show ?case by simp
          next
          case (Suc k)
          then show ?case
            proof –
              have 21:  $(\forall j. (j \leq \text{ff } 0) \vee (\exists i < k. \text{ff}(i) < (j :: \text{nat}) \wedge j \leq \text{ff}(\text{Suc } i)) \vee (\text{ff } k < j \wedge j \leq \text{ff}(\text{Suc } k)) \longrightarrow gg j)$ 
                using Suc.prem1 less-SucI by blast
              have 22:  $(\forall i < k. \text{ff}(i) < \text{ff}(\text{Suc } i))$ 
                using Suc.prem2 by auto
              have 23:  $\text{ff } k < \text{ff}(\text{Suc } k)$ 
                by (simp add: Suc.prem2)
              have 24:  $(\forall j. (j \leq \text{ff } 0) \vee (\exists i < k. \text{ff}(i) < (j :: \text{nat}) \wedge j \leq \text{ff}(\text{Suc } i)) \longrightarrow gg j)$ 

```

```

      (j ≤ ff 0) ∨ (ff 0 < j ∧ j ≤ ff k)
      ∨ (ff k < j ∧ j ≤ ff (Suc k) )
    → gg j)
  using 21 22 Suc.hyps by blast
  have 25: ff 0 < j ∧ j ≤ ff (Suc k) → gg j
    using 24 not-less by blast
  show ?thesis using 25 by blast
qed
qed
qed
show ?thesis
by (metis 2 assms(1) less-or-eq-imp-le linorder-neqE-nat)
qed

```

lemma PiUntilDistsem3:

```

assumes (σ ⊨ f Π (g U h))
shows (σ ⊨ ( f Π g ) U ( f Π h ) )
proof -
  have 1: ((∃ i ≤ nlength σ. f (ndropn i σ)) ∧
    (∃ k ≤ nlength (nfilter f (ndropns σ)).
      h (ndropn k (nmap (λs. nnth s 0) (nfilter f (ndropns σ)))) ∧
      (∀ j < k. g (ndropn j (nmap (λs. nnth s 0) (nfilter f (ndropns σ)))))))
    using assms PiUntilDistsem1 by blast
  have 2: ∃ x ∈ nset(ndropns σ). f x
    using 1 by (simp add: sfxfilter-help)
  have 3: ∀ x ∈ nset(nkfilter f 0 (ndropns σ)). f (nnth (ndropns σ) x)
    using 2 nkfilter-holds by fastforce
  have 4: (∃ k ≤ nlength (nfilter f (ndropns σ)).
    h (ndropn k (nmap (λs. nnth s 0) (nfilter f (ndropns σ)))) ∧
    (∀ j < k. g (ndropn j (nmap (λs. nnth s 0) (nfilter f (ndropns σ))))))
    using 1 by auto
  obtain k where 5: k ≤ nlength (nfilter f (ndropns σ)) ∧
    h (ndropn k (nmap (λs. nnth s 0) (nfilter f (ndropns σ)))) ∧
    (∀ j < k. g (ndropn j (nmap (λs. nnth s 0) (nfilter f (ndropns σ))))))
    using 4 by auto
  have 51: (nnth (nkfilter f 0 (ndropns σ)) k) ≤ nlength σ
    by (metis (mono-tags, lifting) 2 5 gen-nlength-def ndropns-nlength nkfilter-nlength
      nkfilter-upperbound nlength-code)
  have 6: f (ndropn (nnth (nkfilter f 0 (ndropns σ)) k) σ)
    using 2 3 5
    by (metis 1 ndropns-nfilter-nnth nlength-nmap pfilt-def sfxfilt-def sfxfilt-pfilt-nnth-ndropn)
  have 7: k = 0 →
    (∃ i. (enat i) ≤ nlength σ - k ∧ f (ndropn (i + k) σ)) ∧
    h (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn k σ)))) ∧
    (∀ j < k. (∃ i ≤ nlength σ - j. f (ndropn (i + j) σ)) ∧
      g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ)))))
    using 1 5 by auto
  have 71: k > 0 → (nnth (nkfilter f 0 (ndropns σ)) (k - 1)) ∈ nset(nkfilter f 0 (ndropns σ))
    by (metis 2 5 One-nat-def Suc-ile-eq Suc-pred in-nset-conv-nnth less-imp-le nkfilter-nlength)
  have 72: k > 0 → enat (k - 1) ≤ nlength (nkfilter f 0 (ndropns σ))

```

by (metis 2 5 One-nat-def Suc-ile-eq Suc-pred nkfilter-nlength order-less-imp-le)
 have 8: $k > 0 \longrightarrow f \text{ (ndropn (nnth (nkfilter f 0 (ndropns \sigma)) (k-1)) \sigma)}$
 using 3 2 71 72 nkfilter-upperbound[of (ndropns \sigma) f k-1 0]
 by (metis add-0 ndropns-nlength ndropns-nnth zero-enat-def)
 have 9: $k > 0 \longrightarrow (\text{nnth (nkfilter f 0 (ndropns \sigma)) (k-1)}) < (\text{nnth (nkfilter f 0 (ndropns \sigma)) k})$
 by (metis 2 5 One-nat-def Suc-pred nkfilter-mono nkfilter-nlength)
 have 10: $k > 0 \longrightarrow (\text{nnth (nkfilter f 0 (ndropns \sigma)) (k-1)}) \leq \text{nlength } \sigma$
 by (metis 2 71 gen-nlength-def in-nset-conv-nnth ndropns-nlength nkfilter-upperbound
 nlength-code)
 have 11: $k > 0 \longrightarrow (\text{nnth (nkfilter f 0 (ndropns \sigma)) k}) \leq \text{nlength } \sigma$
 using nkfilter-upperbound[of ndropns \sigma f k 0]
 using 51 by blast
 have 12: $k > 0 \longrightarrow$
 $h \text{ (nmap (\lambda s. nnth s 0))}$
 $(\text{nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns \sigma)) k) \sigma)))$
 using nfilter-nkfilter-ndropn[of f (ndropns \sigma) k]
 ndropns-nfilter-ndropn-a[of k f \sigma]
 by (metis 2 5 ndropn-nmap)
 have 121: $k > 0 \longrightarrow$
 $h \text{ (nmap (\lambda s. nnth s 0))}$
 $(\text{nfilter f (ndropns (ndropn (Suc (nnth (nkfilter f 0 (ndropns \sigma)) (k-1))) \sigma)))$
 by (metis 2 5 One-nat-def Suc-pred nfilter-ndropns-nmap)
 have 13: $k > 0 \longrightarrow (\exists i \leq \text{nlength } \sigma - (\text{nnth (nkfilter f 0 (ndropns \sigma)) k}).$
 $f \text{ (ndropn (i + (\text{nnth (nkfilter f 0 (ndropns \sigma)) k)) \sigma)})$
 using 6 by (metis add.left-neutral zero-enat-def zero-le)
 have 130: $k > 0 \longrightarrow (\exists i. i + (\text{Suc (nnth (nkfilter f 0 (ndropns \sigma)) (k-1))) \leq \text{nlength } \sigma \wedge$
 $(\text{ndropn (i + (\text{Suc (nnth (nkfilter f 0 (ndropns \sigma)) (k-1))) \sigma}) =$
 $(\text{ndropn (nnth (nkfilter f 0 (ndropns \sigma)) k) \sigma}))$
 using 9 by (metis 11 Suc-leI diff-add)
 have 131: $k > 0 \longrightarrow (\exists i. i + (\text{Suc (nnth (nkfilter f 0 (ndropns \sigma)) (k-1))) \leq \text{nlength } \sigma \wedge$
 $f \text{ (ndropn (i + (\text{Suc (nnth (nkfilter f 0 (ndropns \sigma)) (k-1))) \sigma}))$
 using 10 6 9 11 130 by auto
 have 132: $k > 0 \longrightarrow (\exists i \leq \text{nlength } \sigma - (\text{Suc (nnth (nkfilter f 0 (ndropns \sigma)) (k-1))).$
 $f \text{ (ndropn (i + (\text{Suc (nnth (nkfilter f 0 (ndropns \sigma)) (k-1))) \sigma}))$
 using 131
 by (metis add-diff-cancel-right' enat-minus-mono1 idiff-enat-enat)
 have 14: $k > 0 \longrightarrow (\forall j < (\text{nnth (nkfilter f 0 (ndropns \sigma)) k}).$
 $(\exists i. i + j \leq \text{nlength } \sigma \wedge f \text{ (ndropn (i + j) \sigma)))$
 using 131 by (metis 11 6 diff-add less-imp-le plus-enat-simps(1))
 have 141: $k > 0 \longrightarrow (\forall j < (\text{Suc (nnth (nkfilter f 0 (ndropns \sigma)) (k-1))).$
 $(\exists i. i + j \leq \text{nlength } \sigma \wedge f \text{ (ndropn (i + j) \sigma)))$
 using 14 9 by auto
 have 142: $k > 0 \longrightarrow (\forall j < (\text{Suc (nnth (nkfilter f 0 (ndropns \sigma)) (k-1))).$
 $(\exists i \leq \text{nlength } \sigma - j. f \text{ (ndropn (i + j) \sigma)))$
 using 141 by (metis add commute enat.simps(3) enat-add-sub-same enat-minus-mono1)
 have 15: $(\forall j < k. g \text{ (ndropn j (nmap (\lambda s. nnth s 0) (nfilter f (ndropns \sigma))))})$
 using 5 by blast
 have 151: $(\forall j < k. g \text{ (nmap (\lambda s. nnth s 0) (ndropn j (nfilter f (ndropns \sigma))))})$
 by (metis 15 ndropn-nmap)
 have 152: $(\forall j < k. (\text{ndropn j (nfilter f (ndropns \sigma))}) =$

```

      (nfilter f (ndropn (nnth (nkfilter f 0 (ndropns σ)) j) (ndropns σ))))
using nfilter-nkfilter-ndropn-1[of ndropns σ f ]
by (simp add: 2 5 less-imp-le nkfilter-nlength order-less-le-subst2)
have 16: (∀ j < k. g (nmap (λs. nnth s 0)
      (nfilter f (ndropn (nnth (nkfilter f 0 (ndropns σ)) j) (ndropns σ)))))
      using 151 152 nfilter-nkfilter-ndropn by simp
have 1610: (nnth (nkfilter f 0 (ndropns σ)) k) ≤ nlength(ndropns σ)
      by (simp add: 51 ndropns-nlength)
have 1611: (∀ j < k. (nnth (nkfilter f 0 (ndropns σ)) j) < (nnth (nkfilter f 0 (ndropns σ)) k))
      by (simp add: 2 5 nidx-nkfilter-gr nkfilter-nlength)
have 1612: (∀ j < k. (nnth (nkfilter f 0 (ndropns σ)) j) ≤ nlength(ndropns σ))
      using 1610 1611 by (meson enat-ord-simps(2) less-imp-le order-less-le-subst2)
have 161: (∀ j < k. ( (ndropn (nnth (nkfilter f 0 (ndropns σ)) j) (ndropns σ))) =
      ( (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) j) σ))))
      using 1612 by (simp add: ndropn-ndropns)
have 17: (∀ j < k. g (nmap (λs. nnth s 0)
      (nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) j) σ)))))
      using 16 161 by auto
have 18: k > 0 ⟶
      g (nmap (λs. nnth s 0)
      (nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) (k-1)) σ))))
      using 17 by simp
have 19: k > 0 ⟶
      g (nmap (λs. nnth s 0)
      (nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) 0) σ))))
      using 17 by blast
have 20: k > 0 ⟶ (nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) 0) σ))) =
      (nfilter f (ndropn (nnth (nkfilter f 0 (ndropns σ)) 0) (ndropns σ)))
      using 161 by auto
have 200: nnth (nkfilter f 0 (ndropns σ)) 0 ≤ nlength (ndropns σ)
      using 1610 1612 by blast
have 21: k > 0 ⟶ (∀ j ≤ (nnth (nkfilter f 0 (ndropns σ)) 0).
      ( (ndropns (ndropn j σ))) =
      ( (ndropn j (ndropns σ))))
      using 200 ndropn-ndropns by (simp add: ndropn-ndropns order-subst2)
have 22: k > 0 ⟶ (∀ j ≤ (nnth (nkfilter f 0 (ndropns σ)) 0).
      (nfilter f (ndropns (ndropn j σ))) =
      (nfilter f (ndropn j (ndropns σ))))
      using 21 by auto
have 23: k > 0 ⟶
      (∀ j ≤ (nnth (nkfilter f 0 (ndropns σ)) 0).
      (nmap (λs. nnth s 0)
      (nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) 0) σ)))) =
      (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ))))
      )
      by (simp add: 2 22 nfilter-ndropns-nmap-help-0)
have 24: k > 0 ⟶
      (∀ j ≤ (nnth (nkfilter f 0 (ndropns σ)) 0).
      g (nmap (λs. nnth s 0)
      (nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) 0) σ)))) =

```

```

      g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ))))
    )
  using 23 by auto
have 241: k>0 → (∀ j ≤ (nnth (nkfilter f 0 (ndropns σ)) 0).
  g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ)))) )
  using 19 24 by blast
have 25: k>0 → (∀ i < k - 1.
  (∀ l. l ≤ (nnth (nkfilter f 0 (ndropns σ)) (Suc i)) ∧
    (nnth (nkfilter f 0 (ndropns σ)) i) < l →
    (nfilter f (ndropn (nnth (nkfilter f 0 (ndropns σ)) (Suc i)) (ndropns σ)) =
    (nfilter f (ndropn l (ndropns σ))) ) ) )
proof auto
  fix i
  fix l
  assume a0: 0 < k
  assume a1: i < k - Suc 0
  assume a2: l ≤ nnth (nkfilter f 0 (ndropns σ)) (Suc i)
  assume a3: nnth (nkfilter f 0 (ndropns σ)) i < l
  show nfilter f (ndropn (nnth (nkfilter f 0 (ndropns σ)) (Suc i)) (ndropns σ)) =
    nfilter f (ndropn l (ndropns σ))
  proof -
    have 251: k=1 ⇒
      nfilter f (ndropn (nnth (nkfilter f 0 (ndropns σ)) (Suc i)) (ndropns σ)) =
      nfilter f (ndropn l (ndropns σ))
    using a1 by force
    have 2511: 1 < k ⇒ enat (Suc i) ≤ nlength (nfilter f (ndropns σ))
      by (metis (mono-tags, opaque-lifting) 5 One-nat-def Suc-diff-1 Suc-mono a0 a1
        enat-ord-simps(1) less-imp-le order-subst2)
    then have 252: 1 < k ⇒
      nfilter f (ndropn (nnth (nkfilter f 0 (ndropns σ)) (Suc i)) (ndropns σ)) =
      nfilter f (ndropn l (ndropns σ))
    using 2 5 a1 nfilter-ndropns-nmap-help-j[of ndropns σ f l i] a2 a3 by fastforce
    show ?thesis
    using 252 a1 by fastforce
  qed
qed
have 261: k>0 → (∀ i < k - 1.
  (∀ l. l ≤ (nnth (nkfilter f 0 (ndropns σ)) (Suc i)) ∧
    (nnth (nkfilter f 0 (ndropns σ)) i) < l →
    l ≤ nlength (ndropns σ))
  using 1612 by (meson diff-less enat-ord-simps(1) less-one less-trans-Suc order-subst2)
have 262: k>0 → (∀ i < k - 1.
  (∀ l. l ≤ (nnth (nkfilter f 0 (ndropns σ)) (Suc i)) ∧
    (nnth (nkfilter f 0 (ndropns σ)) i) < l →
    (ndropn l (ndropns σ)) = (ndropns (ndropn l σ)))
  using 261 by (simp add: ndropn-ndropns)
have 26: k>0 → (∀ i < k - 1.
  (∀ l. l ≤ (nnth (nkfilter f 0 (ndropns σ)) (Suc i)) ∧
    (nnth (nkfilter f 0 (ndropns σ)) i) < l →
    (nmap (λs. nnth s 0)

```

$(\text{nfilter } f \text{ (ndropn (nnth(nkfilter } f \text{ 0 (ndropns } \sigma)) (Suc \ i)) (ndropns } \sigma))) =$
 $(\text{nmap } (\lambda s. \text{nnth } s \text{ 0}) (\text{nfilter } f \text{ (ndropns (ndropn } l \text{ } \sigma))) \text{ }) \text{ })$
using 25 262 **by** auto
have 27: $k > 0 \longrightarrow (\forall \ i < k - 1.$
 $(\forall \ l. l \leq (\text{nnth (nkfilter } f \text{ 0 (ndropns } \sigma)) (Suc \ i)) \wedge$
 $(\text{nnth (nkfilter } f \text{ 0 (ndropns } \sigma)) \ i) < l \longrightarrow$
 $g \text{ (nmap } (\lambda s. \text{nnth } s \text{ 0})$
 $(\text{nfilter } f$
 $(\text{ndropn (nnth(nkfilter } f \text{ 0 (ndropns } \sigma)) (Suc \ i)) (ndropns } \sigma))))))$
by (simp add: 16)
have 28: $k > 0 \longrightarrow (\forall \ i < k - 1.$
 $(\forall \ l. l \leq (\text{nnth (nkfilter } f \text{ 0 (ndropns } \sigma)) (Suc \ i)) \wedge$
 $(\text{nnth (nkfilter } f \text{ 0 (ndropns } \sigma)) \ i) < l \longrightarrow$
 $g \text{ (nmap } (\lambda s. \text{nnth } s \text{ 0}) (\text{nfilter } f \text{ (ndropns (ndropn } l \text{ } \sigma))) \text{ }))$
using 25 262 27 **by** auto
have 281: $k > 0 \longrightarrow$
 $(\forall j.$
 $(j \leq (\text{nnth (nkfilter } f \text{ 0 (ndropns } \sigma)) \ 0) \vee$
 $(\exists \ i. i < k - 1 \wedge$
 $j \leq (\text{nnth (nkfilter } f \text{ 0 (ndropns } \sigma)) (Suc \ i)) \wedge$
 $(\text{nnth (nkfilter } f \text{ 0 (ndropns } \sigma)) \ i) < j) \text{ })$
 $\longrightarrow g \text{ (nmap } (\lambda s. \text{nnth } s \text{ 0}) (\text{nfilter } f \text{ (ndropns (ndropn } j \text{ } \sigma))) \text{ }))$
using 241 28 **by** blast
have 282: $k > 0 \longrightarrow (\forall i < k - 1.$
 $\text{nnth (nkfilter } f \text{ 0 (ndropns } \sigma)) \ i < \text{nnth (nkfilter } f \text{ 0 (ndropns } \sigma)) (Suc \ i))$
using nidx-nkfilter-expand[of ndropns σ f - 0]
by (metis (full-types) 2 72 Suc-ile-eq enat-ord-simps(2) order-less-le-subst2)
have 29: $k > 0 \longrightarrow (\forall j < (Suc (\text{nnth (nkfilter } f \text{ 0 (ndropns } \sigma)) (k - 1))).$
 $g \text{ (nmap } (\lambda s. \text{nnth } s \text{ 0}) (\text{nfilter } f \text{ (ndropns (ndropn } j \text{ } \sigma))))$
using 281 282 cover-a[of $\lambda x. \text{nnth (nkfilter } f \text{ 0 (ndropns } \sigma)) \ x \ k - 1$
 $(\lambda j. g \text{ (nmap } (\lambda s. \text{nnth } s \text{ 0}) (\text{nfilter } f \text{ (ndropns (ndropn } j \text{ } \sigma))))$]
using 18 less-antisym **by** blast
have 30: $k > 0 \longrightarrow (\exists k \leq \text{nlength } \sigma.$
 $(\exists i \leq \text{nlength } \sigma - k. f \text{ (ndropn (i + k) } \sigma)) \wedge$
 $h \text{ (nmap } (\lambda s. \text{nnth } s \text{ 0}) (\text{nfilter } f \text{ (ndropns (ndropn } k \text{ } \sigma)))) \wedge$
 $(\forall j < k. (\exists i \leq \text{nlength } \sigma - j. f \text{ (ndropn (i + j) } \sigma)) \wedge$
 $g \text{ (nmap } (\lambda s. \text{nnth } s \text{ 0}) (\text{nfilter } f \text{ (ndropns (ndropn } j \text{ } \sigma))))$
using 10 11 121 132 142 29 9 **by** simp
 $(\text{metis add-Suc-right antisym-conv2 eSuc-enat enat-ord-simps(1) ileI1 leD})$
have 31: $(\exists k \leq \text{nlength } \sigma.$
 $(\exists i \leq \text{nlength } \sigma - k. f \text{ (ndropn (i + k) } \sigma)) \wedge$
 $h \text{ (nmap } (\lambda s. \text{nnth } s \text{ 0}) (\text{nfilter } f \text{ (ndropns (ndropn } k \text{ } \sigma)))) \wedge$
 $(\forall j < k. (\exists i \leq \text{nlength } \sigma - j. f \text{ (ndropn (i + j) } \sigma)) \wedge$
 $g \text{ (nmap } (\lambda s. \text{nnth } s \text{ 0}) (\text{nfilter } f \text{ (ndropns (ndropn } j \text{ } \sigma))))$
using 30 7 **by** (metis not-gr-zero zero-enat-def zero-le)
show ?thesis **using** 31 **by** (simp add: PiUntilDistsem2)
qed

lemma PiUntilDistsem4:
assumes $(\sigma \models (f \ \Pi \ g) \ \mathcal{U} \ (f \ \Pi \ h))$

shows $(\sigma \models f \Pi (g \mathcal{U} h))$
proof –
have 1: $(\exists k \leq \text{nlength } \sigma.$
 $(\exists i \leq \text{nlength } \sigma - k. f (\text{ndropn } (i + k) \sigma)) \wedge$
 $h (\text{nmap } (\lambda s. \text{nnth } s 0) (\text{nfilter } f (\text{ndropns } (\text{ndropn } k \sigma)))) \wedge$
 $(\forall j < k.$
 $(\exists i \leq \text{nlength } \sigma - j. f (\text{ndropn } (i + j) \sigma)) \wedge$
 $g (\text{nmap } (\lambda s. \text{nnth } s 0) (\text{nfilter } f (\text{ndropns } (\text{ndropn } j \sigma))))))$
using *assms* **by** (*simp add: PiUntilDistsem2*)
obtain *k* **where** 2: $k \leq \text{nlength } \sigma \wedge$
 $(\exists i \leq \text{nlength } \sigma - k. f (\text{ndropn } (i + k) \sigma)) \wedge$
 $h (\text{nmap } (\lambda s. \text{nnth } s 0) (\text{nfilter } f (\text{ndropns } (\text{ndropn } k \sigma)))) \wedge$
 $(\forall j < k.$
 $(\exists i \leq \text{nlength } \sigma - j. f (\text{ndropn } (i + j) \sigma)) \wedge$
 $g (\text{nmap } (\lambda s. \text{nnth } s 0) (\text{nfilter } f (\text{ndropns } (\text{ndropn } j \sigma))))))$
using 1 **by** *auto*
have 3: $(\exists i \leq \text{nlength } \sigma - k. f (\text{ndropn } (i + k) \sigma))$
using 2 **by** *auto*
have 31: $(\exists i \leq \text{nlength } (\text{ndropns } (\text{ndropn } k \sigma)). f (\text{nnth } (\text{ndropns } (\text{ndropn } k \sigma)) i))$
by (*metis 3 add.commute ndropn-ndropn ndropn-nlength ndropns-nlength ndropns-nnth*)
have 4: $k \leq \text{nlength } \sigma$
using 2 **by** *auto*
obtain *i* **where** 5: $(\text{enat } i) \leq \text{nlength } (\text{ndropns } (\text{ndropn } k \sigma)) \wedge f (\text{nnth } (\text{ndropns } (\text{ndropn } k \sigma)) i) \wedge$
 $i = (\text{LEAST } na. \text{enat } na \leq \text{nlength } (\text{ndropns } (\text{ndropn } k \sigma)) \wedge$
 $f (\text{nnth } (\text{ndropns } (\text{ndropn } k \sigma)) na))$
by (*metis (no-types, lifting) 31 LeastI-ex*)
have 51: $i = (\text{LEAST } na. \text{enat } na \leq \text{nlength } (\text{ndropns } (\text{ndropn } k \sigma)) \wedge f (\text{nnth } (\text{ndropns } (\text{ndropn } k \sigma))$
 $na))$
using 5 **by** *auto*
have 60: $(\text{enat } i) \leq \text{nlength } \sigma - k$
by (*metis 5 ndropn-nlength ndropns-nlength*)
have 601: $k + i \leq \text{nlength } \sigma$
by (*metis 4 60 dual-order.eq-iff enat.simps(3) enat-add-sub-same enat-less-enat-plusI2*
 $\text{less-eqE less-imp-le order.not-eq-order-implies-strict plus-enat-simps(1)}$)
have 6: $i + k \leq \text{nlength } \sigma$
by (*metis 601 add.commute*)
have 61: $\exists x \in \text{nset } (\text{ndropns } (\text{ndropn } k \sigma)). f x$
using 31 *exists-Pred-nnth-nset* **by** *blast*
have 62: $\exists x \in \text{nset } (\text{ndropns } (\text{ndropn } (k + i) \sigma)). f x$
by (*metis 5 in-nset-conv-nnth ndropn-ndropn ndropn-ndropns nnth-zero-ndropn zero-enat-def zero-le*)
have 7: $(\exists i \leq \text{nlength } \sigma. f (\text{ndropn } i \sigma))$
by (*metis 5 6 add.commute ndropn-ndropn ndropns-nlength ndropns-nnth*)
have 71: $\exists x \in \text{nset } (\text{ndropns } \sigma). f x$
using 5 6 **using** *in-nset-ndropns* **using** 7 **by** *blast*
have 72: $\exists x \in \text{nset } (\text{nkfilter } f 0 (\text{ndropns } \sigma)). f (\text{nnth } (\text{ndropns } \sigma) x)$
using 71 *nkfilter-holds-b[of (ndropns σ) f] sfxfilter-help[of σ]*
by (*metis 71 Nat.add-0-right ndropns-nlength ndropns-nnth*)
have 722: $(\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } (\text{ndropn } k \sigma))) 0) \leq \text{nlength } (\text{ndropns } (\text{ndropn } k \sigma))$
by (*metis 61 gen-nlength-def nkfilter-upperbound nlength-code zero-enat-def zero-le*)
have 723: $(\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } (\text{ndropn } k \sigma))) 0) \leq \text{nlength } (\text{ndropn } k \sigma)$


```

  by (metis 722 ndropns-nlength)
have 73: f (ndropn (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0) (ndropn k σ))
  using nkfilter-holds[of ndropns (ndropn k σ) f 0]
  by (metis (no-types, lifting) 61 722 diff-zero ndropns-nfilter-nnth ndropns-nlength
    ndropns-nnth nkfilter-nfilter zero-enat-def zero-le)
have 74: f (ndropn ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k) σ)
  using 73 by (simp add: add.commute ndropn-ndropn)
have 75: f (ndropn k (ndropn (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0) σ))
  by (simp add: 74 ndropn-ndropn)
have 76: (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0) = nleast f (ndropns (ndropn k σ))
  by (simp add: 61 nkfilter-nleast)
have 77: nleast f (ndropns (ndropn k σ)) = (LEAST na. enat na ≤ nlength (ndropns (ndropn k σ)) ∧
  f (nnth (ndropns (ndropn k σ)) na))
  using 61 nleast-conv[of ndropns (ndropn k σ) f] by auto
have 78: nleast f (ndropns (ndropn k σ)) = i
  using 5 77 by blast
have 8: h (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn k σ))))
  using 2 by auto
have 10: nfirst (nkfilter f k (ndropn k (ndropns σ))) =
  ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k)
  by (metis 2 61 diff-add ndropn-0 ndropn-ndropns ndropn-nfirst ndropns-nlength
    nkfilter-lowerbound nkfilter-nnth-n-zero zero-enat-def zero-le)
have 101: nfirst (nkfilter f k (ndropn k (ndropns σ))) = k+ i
  by (simp add: 10 76 78)
have 90: f (nlast (ntaken ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k) (ndropns σ)))
  by (metis 6 74 76 78 ndropns-nnth ntaken-nlast)
have 91: ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k) ≤ nlength (ndropns σ)
  using nkfilter-upperbound[of (ndropns (ndropn k σ)) f 0 0]
    ndropn-nlength[of k σ] ndropns-nlength[of ndropn k σ]
  by (simp add: 6 76 78 ndropns-nlength)
have 92: ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k) = ((nnth (nkfilter f k (ndropns (ndropn k σ)))
0))
  by (simp add: 61 nkfilter-nleast)
let ?kf = (the-enat ((nlength(nfilter f (ntaken ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k) (ndropns
σ))))))
let ?nkf = (the-enat ((nlength(nkfilter f 0 (ntaken ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k)
(ndropns σ))))))
let ?pkf = (the-enat ((nlength (nkfilter f 0 (ntaken k (ndropns σ))))))
have 93: ?kf = ?nkf
  by (metis 90 nfinite-ntaken nkfilter-nlength nset-nlast)
have 94: ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k) ≤ nnth (nkfilter f 0 (ndropns σ)) ?nkf
  by (metis 74 90 91 add.left-neutral eq-iff ndropn-nfirst ndropns-nlength ndropns-nnth
    nkfilter-chop1-ndropn nkfilter-nfirst)
have 95: (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0) = 0 ⇒ k = nnth (nkfilter f 0 (ndropns σ)) ?nkf
  by (metis 10 90 91 add-cancel-left-left ndropn-nfirst nkfilter-chop1-ndropn)
have 98: 0 < ?nkf ∧ 0 < (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0) ⇒
  nnth (nkfilter f 0 (ndropns σ)) (?nkf - 1) < k
proof -
  assume b: 0 < ?nkf ∧ 0 < (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)
  have 980: (nnth (nkfilter f 0 (ndropns σ)) (?nkf - 1)) ∈ nset (nkfilter f 0 (ndropns σ))

```

```

  by (metis in-nset-conv-nnth le-cases nfinite-ntaken nset-nlast ntaken-all ntaken-nlast)
have 981: f ( nnth (ndropns σ) (nnth (nkfilter f 0 (ndropns σ)) (?nkf - 1)) )
  using nkfilter-holds[of (ndropns σ) f (nnth (nkfilter f 0 (ndropns σ)) (?nkf - 1)) 0]
  using 71 980 by auto
have 982: ¬ f ( ndropn k σ)
  by (metis 10 4 add-cancel-left-left b gr-implies-not0 ndropn-nfirst ndropns-nnth
    nkfilter-nfirst)
have 984: f (ndropn (nnth (nkfilter f 0 (ndropns σ)) (?nkf - 1)) σ)
  using 71 980 981 ndropns-nnth[of (nnth (nkfilter f 0 (ndropns σ)) (?nkf - 1)) σ]
  by (metis gen-nlength-def in-nset-conv-nnth ndropns-nlength nkfilter-upperbound nlength-code)
have 985:  $\bigwedge j. k \leq j \implies j < k+i \implies \neg f (ndropn j \sigma)$ 
  proof -
    fix j
    assume a0:  $k \leq j$ 
    assume a1:  $j < k+i$ 
    show  $\neg f (ndropn j \sigma)$ 
    proof -
      have 9851:  $\exists x \in \text{nset} (ndropn k (ndropns \sigma)). f x$ 
        by (simp add: 4 61 ndropn-ndropns ndropns-nlength)
      have 9852:  $\text{enat } k \leq \text{nlength} (ndropns \sigma)$ 
        by (simp add: 4 ndropns-nlength)
      have 9853:  $k \leq j$ 
        by (simp add: a0)
      have 9854:  $j < \text{nnth} (nkfilter f k (ndropn k (ndropns \sigma))) 0$ 
        using 10 101 92 9852 a1 ndropn-ndropns by fastforce
      have 9855:  $\neg f (\text{nnth} (ndropns \sigma) j)$ 
        using 9851 9852 9853 9854 nkfilter-n-not-before[of k ndropns σ f j] by auto
      show ?thesis
        by (metis 10 101 6 76 78 9855 a1 enat-ord-simps(2) less-imp-le
          ndropns-nnth order-less-le-subst2)
    qed
  qed
have 986:  $\text{ntaken } ?nkf (nkfilter f 0 (ndropns \sigma)) =$ 
   $nkfilter f 0 (\text{ntaken} (\text{nnth} (nkfilter f 0 (ndropns (ndropn k \sigma))) 0 + k) (ndropns \sigma))$ 
  using nkfilter-chop1-ntaken[of (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0 + k) (ndropns σ) f 0]
  using 90 91 by blast
have 987:  $\neg \text{enat } (?nkf) \leq \text{nlength} (nkfilter f 0 (ndropns \sigma)) \vee$ 
   $\text{nnth} (nkfilter f 0 (ndropns \sigma)) (?nkf - 1) < \text{nnth} (nkfilter f 0 (ndropns \sigma)) (?nkf)$ 
  using 71 by (meson b diff-less less-one nidx-nkfilter-gr)
have 988:  $\text{nlast} (nkfilter f 0 (\text{ntaken} (\text{nnth} (nkfilter f 0 (ndropns (ndropn k \sigma))) 0 + k) (ndropns \sigma)))$ 
   $=$ 
   $0 + (\text{nnth} (nkfilter f 0 (ndropns (ndropn k \sigma))) 0 + k)$ 
  by (simp add: 90 91 nkfilter-nlast)
have 989:  $\text{nnth} (nkfilter f 0 (ndropns \sigma)) (?nkf) = \text{nnth} (nkfilter f 0 (ndropns (ndropn k \sigma))) 0 + k$ 
  using 986
  by (metis 988 add.left-neutral ntaken-nlast)
have 990:  $\text{nnth} (nkfilter f 0 (ndropns \sigma)) (?nkf) = \text{nfirst} (nkfilter f k (ndropn k (ndropns \sigma)))$ 
  using 10 989 by fastforce
have 991:  $\text{nnth} (nkfilter f 0 (ndropns \sigma)) (?nkf) = k + i$ 
  using 10 101 989 by presburger

```

```

show ?thesis
  by (metis 984 985 986 987 991 enat-the-enat infinity-ileE le-cases not-le-imp-less ntaken-all)
qed
have 100: h (nmap (λs. nnth s 0) (ndropn ?kf (nfilter f (ndropns σ))))
  using ndropns-nfilter-ndropn-a[of - f ndropn k σ ]
    nfilter-chop1-ndropn[of ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k) (ndropns σ) f ]
    101 61 76 78 8 90 91
  by simp
  (metis 10 ndropn-0 ndropn-ndropn ndropn-ndropns zero-enat-def zero-le)
have 11: h (ndropn ?kf (nmap (λs. nnth s 0) (nfilter f (ndropns σ))))
  by (simp add: 100 ndropn-nmap)
have 111: ?kf ≤ nlength (nfilter f (ndropns σ))
  by (metis 90 enat-ile le-cases nfilter-chop1-ntaken ntaken-all the-enat.simps)
have 112: nfinite (nfilter f (ntaken ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k) (ndropns σ)))
  by (metis 90 91 nfilter-chop1-ntaken nfinite-ntaken)
have 13: (∀ j < k.
  (∃ x ∈ nset(ndropns (ndropn j σ)). f x) ∧
  g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ)))))
  by (metis 2 add commute in-nset-ndropns ndropn-ndropn ndropn-nlength)
have 151: (∀ jj < ?kf.
  (ndropn jj (nfilter f (ndropns σ))) =
  nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) jj) σ))
  ) using nfilter-nkfilter-ndropn
  by (simp add: 111 71 less-imp-le ndropns-nfilter-ndropn-a order-less-le-subst2)
have 152: (∧ jj. (enat jj) < ?kf ⟹ (nnth (nkfilter f 0 (ndropns σ)) jj) < k )
proof -
  fix jj
  assume a: (enat jj) < ?kf
  show nnth (nkfilter f 0 (ndropns σ)) jj < k
  proof -
    have 1522: (enat jj) ≤ ?nkf - 1
      using 93 a by auto
    have 1523: nnth (nkfilter f 0 (ndropns σ)) (?nkf - 1) < nnth (nkfilter f 0 (ndropns σ)) ?nkf
      by (metis 111 71 93 a diff-less gr-implies-not-zero less-numeral-extra(1)
        nidx-nkfilter-gr nkfilter-nlength not-gr-zero zero-enat-def)
    have 15230: enat (?nkf - 1) ≤ nlength (nkfilter f 0 (ndropns σ))
      by (metis 111 71 93 One-nat-def Suc-ile-eq Suc-pred a less-imp-le nkfilter-nlength
        not-gr-zero not-less-zero zero-enat-def)
    have 1524: nnth (nkfilter f 0 (ndropns σ)) jj ≤ nnth (nkfilter f 0 (ndropns σ)) (?nkf - 1)
      using 1522 1523 71 93 15230
        nidx-less-eq[of (nkfilter f 0 (ndropns σ)) (jj) ?nkf - 1]
        nidx-nkfilter[of (ndropns σ) f 0]
      using enat-ord-simps(1) by blast
    have 1525: k ≤ nnth (nkfilter f 0 (ndropns σ)) ?nkf
      using 94 add-leE by blast
    have 1526: (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0) = 0 ⟹
      nnth (nkfilter f 0 (ndropns σ)) jj < k
      using 1523 1524 95 by linarith
    have 1527: 0 < (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0) ⟹
      nnth (nkfilter f 0 (ndropns σ)) (?nkf - 1) < k

```

```

    using 93 98 a by auto
    show ?thesis using 1524 1526 1527 dual-order.strict-trans2 by blast
qed
qed
have 153: (∀ jj < ?kf.
  g (nmap (λs. nnth s 0)
    (nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) jj) σ)) )
  ))
  using 112 13 152 nfinite-nlength-enat by force
have 1530: (∀ jj < ?kf.
  g (nmap (λs. nnth s 0) (ndropn jj (nfilter f (ndropns σ)))))
  by (simp add: 151 153)
have 1531: (∀ jj < ?kf.
  g (ndropn jj (nmap (λs. nnth s 0) (nfilter f (ndropns σ)))))
  by (simp add: 1530 ndropn-nmap)
have 154: ( (∃ i. (enat i) ≤ nlength σ ∧ f (ndropn i σ)) ∧
  (∃ kk. (enat kk) ≤ nlength (nfilter f (ndropns σ)) ∧
    h (ndropn kk (nmap (λs. nnth s 0) (nfilter f (ndropns σ))))) ∧
  (∀ jj < kk. g (ndropn jj (nmap (λs. nnth s 0) (nfilter f (ndropns σ))))) )
  using 11 111 1531 7 by blast
show ?thesis by (simp add: 154 PiUntilDistsem1)
qed

```

lemma *PiUntilDistsem*:

$\sigma \models f \Pi (g \mathcal{U} h) = (f \Pi g) \mathcal{U} (f \Pi h)$

using *PiUntilDistsem3 PiUntilDistsem4* using *unl-lift2* by blast

lemma *PiUntilDist*:

$\vdash f \Pi (g \mathcal{U} h) = (f \Pi g) \mathcal{U} (f \Pi h)$

using *PiUntilDistsem Valid-def* by blast

PiChopstar

lemma *wnextboxnotstatesem*:

assumes $k \leq \text{nlength } \sigma$

shows $(\forall j \leq \text{nlength } \sigma. k < j \longrightarrow \neg (\lambda y. w (NNil y)) (\text{nnth } \sigma j)) =$
 $(\text{LIFT}(wnext (\Box (\text{init } (\neg w)))) (\text{ndropn } k \sigma))$

using *assms*

proof (auto simp add: *itl-defs ndropn-nfirst*)

show $\bigwedge n. \text{enat } k \leq \text{nlength } \sigma \implies$

$\forall j. \text{enat } j \leq \text{nlength } \sigma \longrightarrow k < j \longrightarrow \neg w (NNil (\text{nnth } \sigma j)) \implies$

$\text{nlength } \sigma - \text{enat } k \neq \text{enat } 0 \implies$

$\text{enat } n \leq \text{nlength } \sigma - \text{enat } k - \text{enat } (\text{Suc } 0) \implies w (NNil (\text{nnth } \sigma (\text{Suc } (k + n)))) \implies \text{False}$

by (*metis add-Suc-right assms diff-self-eq-0 dual-order.order-iff-strict*

enat-ord-simps(2) idiff-enat-enat less-add-same-cancel1 linorder-le-cases

nfinite-nlength-enat nfinite-ntaken not-le-imp-less ntaken-all ntaken-nlast

zero-enat-def zero-less-Suc)

show $\bigwedge j. \text{enat } k \leq \text{nlength } \sigma \implies \text{enat } j \leq \text{nlength } \sigma \implies k < j \implies w (NNil (\text{nnth } \sigma j)) \implies$

$\text{nlength } \sigma - \text{enat } k = \text{enat } 0 \implies \text{False}$

by (metis add.right-neutral enat.inject enat-add-sub-same enat-ord-code(4) leD less-eqE
 min.strict-order-iff min-enat-simps(1) zero-enat-def)
 show $\bigwedge j. \text{enat } k \leq \text{nlength } \sigma \implies$
 $\text{enat } j \leq \text{nlength } \sigma \implies$
 $k < j \implies w (NNil (\text{nnth } \sigma j)) \implies$
 $\forall n. \text{enat } n \leq \text{nlength } \sigma - \text{enat } k - \text{enat } (Suc\ 0) \longrightarrow \neg w (NNil (\text{nnth } \sigma (Suc\ (k + n)))) \implies False$
 proof –
 fix j
 assume a0: $\text{enat } k \leq \text{nlength } \sigma$
 assume a1: $\text{enat } j \leq \text{nlength } \sigma$
 assume a2: $k < j$
 assume a3: $w (NNil (\text{nnth } \sigma j))$
 assume a4: $\forall n. \text{enat } n \leq \text{nlength } \sigma - \text{enat } k - \text{enat } (Suc\ 0) \longrightarrow \neg w (NNil (\text{nnth } \sigma (Suc\ (k + n))))$
 show False
 proof –
 have 1: $j = (Suc\ (k + (j - (Suc\ k))))$
 using a2 by auto
 have 2: $\forall n. \text{enat } n + 1 + k \leq \text{nlength } \sigma \longrightarrow \neg w (NNil (\text{nnth } \sigma (Suc\ (k + n))))$
 by auto
 (metis One-nat-def a4 add commute enat.simps(3) enat-add-sub-same enat-minus-mono1
 one-enat-def)
 have 3: $(j - (Suc\ k)) + 1 + k \leq \text{nlength } \sigma$
 using 1 a1 by simp
 have 4: $\neg w (NNil (\text{nnth } \sigma (Suc\ (k + (j - (Suc\ k))))))$
 using 2 3 eSuc-enat plus-1-eSuc(2) by auto
 from 1 4 a3 show ?thesis by auto
 qed
 qed
 qed

lemma NotStateUntilStateAndsem:

$(\sigma \models (\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge f)) =$
 $(\exists k. (\text{enat } k) \leq \text{nlength } \sigma \wedge w (NNil (\text{nnth } \sigma k)) \wedge f (\text{ndropn } k\ \sigma) \wedge (\forall j < k. \neg w (NNil (\text{nnth } \sigma j))))$

by (auto simp add: until-d-def init-defs ndropn-nfirst)

lemma StateUntilEqWPrevChopsem:

$\sigma \models (\text{init } w) \mathcal{U} f = (wprev (\Box (\text{init } w))) \frown f$

proof (auto simp add: until-d-def itl-defs ntaken-nnth ndropn-nfirst min-absorb1)

show $\bigwedge k. \text{enat } k \leq \text{nlength } \sigma \implies$
 $f (\text{ndropn } k\ \sigma) \implies$
 $\forall j < k. w (NNil (\text{nnth } \sigma j)) \implies$
 $\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\min (\text{enat } n) (\text{nlength } \sigma) = \text{enat } 0 \vee$
 $(\forall na. na \leq \text{the-enat } (\min (\text{enat } (n - Suc\ 0)) (\text{epred } (\text{nlength } \sigma))) \wedge na \leq n \wedge \text{enat } na \leq$
 $\text{nlength } \sigma \longrightarrow$
 $w (NNil (\text{nnth } \sigma na)))) \wedge$
 $f (\text{ndropn } n\ \sigma)$
 by (metis One-nat-def Suc-pred epred-enat epred-min le-imp-less-Suc min.orderE not-gr-zero
 the-enat.simps)

```

next
  fix n
  assume a0: enat n ≤ nlength σ
  assume a1: f (ndropn n σ)
  assume a2: ∀ na. na ≤ the-enat (min (enat (n - Suc 0)) (epred (nlength σ))) ∧ na ≤ n ∧
    enat na ≤ nlength σ → w (NNil (nnth σ na))
  show ∃ k. enat k ≤ nlength σ ∧ f (ndropn k σ) ∧ (∀ j < k. w (NNil (nnth σ j)))
  proof -
  have 1: enat n ≤ nlength σ ∧ f (ndropn n σ)
  using a0 a1 by auto
  have 2: (∀ na. na ≤ the-enat (min (enat (n - Suc 0)) (epred (nlength σ))) ∧ na ≤ n ∧
    enat na ≤ nlength σ → w (NNil (nnth σ na))) →
    (∀ j < n. w (NNil (nnth σ j)))
  by (auto simp add: min-def)
    (simp add: a0 less-imp-le order-less-le-subst2,
    metis One-nat-def a0 epred-enat epred-min min.absorb-iff2)
  have 3: (∀ j < n. w (NNil (nnth σ j)))
  using 2 a2 by blast
  show ?thesis
  using 1 3 by blast
qed
qed

```

lemma *StateUntilEqvWPrevChop*:

```

⊢ (init w) U f = (wprev (□ (init w))) ◊ f
using StateUntilEqvWPrevChopsem Valid-def by blast

```

lemma *UntilChopDist*:

```

⊢ (init w) U (g ◊ h) = ((init w) U g) ◊ h
using StateUntilEqvWPrevChop[of w]
by (metis SChopAssoc inteq-reflection)

```

lemma *PiEmptysem*:

```

σ ⊨ (init w) Π empty = (init (¬ w)) U ((init w) ∧ wnext (□ (init (¬ w))))
proof -
have 1: (σ ⊨ (init w) Π empty) =
  ((∃ x ∈ nset σ. w (NNil x)) ∧ nlength (nfilter (λy. w (NNil y)) σ) = 0)
by (simp add: Pstate itl-defs zero-enat-def)
have 2: ((∃ x ∈ nset σ. w (NNil x)) ∧ nlength (nfilter (λy. w (NNil y)) σ) = 0) =
  (∃ k ≤ nlength σ . (λy. w (NNil y)) (nnth σ k) ∧
    (∀ j. j < k → ¬ (λy. w (NNil y)) (nnth σ j)) ∧
    (∀ j ≤ nlength σ. k < j → ¬ (λy. w (NNil y)) (nnth σ j)))
by (simp add: nfilter-nlength-zero-conv-2)
have 3: (∃ k ≤ nlength σ . (λy. w (NNil y)) (nnth σ k) ∧
  (∀ j. j < k → ¬ (λy. w (NNil y)) (nnth σ j)) ∧
  (∀ j ≤ nlength σ. k < j → ¬ (λy. w (NNil y)) (nnth σ j))) =
  (∃ k ≤ nlength σ.
    w (NNil (nnth σ k)) ∧
    (LIFT(wnext (□ (init (¬ w))))) (ndropn k σ) ∧
    (∀ j < k. ¬ w (NNil (nnth σ j)))) (is ?lhs = ?rhs)

```

```

proof
  assume  $a$ : ?lhs
  show ?rhs using  $a$  wnextboornotstatesem by auto
  next
  assume  $b$ : ?rhs
  show ?lhs
  proof –
    obtain  $k$  where 1:
       $enat\ k \leq nlength\ \sigma \wedge w\ (NNil\ (nnth\ \sigma\ k)) \wedge$ 
       $(LIFT(wnext\ (\Box\ (init\ (\neg\ w))))\ (ndropn\ k\ \sigma) \wedge (\forall j < k. \neg\ w\ (NNil\ (nnth\ \sigma\ j)))) \wedge$ 
      using  $b$  by auto
    have 2:  $enat\ k \leq nlength\ \sigma \wedge w\ (NNil\ (nnth\ \sigma\ k)) \wedge (\forall j < k. \neg\ w\ (NNil\ (nnth\ \sigma\ j))) \wedge$ 
       $(\forall j. enat\ j \leq nlength\ \sigma \longrightarrow k < j \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ j)))$ 
      using 1 wnextboornotstatesem[of  $k\ \sigma\ w$ ]
    proof –
      have  $\forall x0. (x0 < k \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ x0))) = (\neg\ x0 < k \vee \neg\ w\ (NNil\ (nnth\ \sigma\ x0)))$ 
        by meson
      then have f1:  $enat\ k \leq nlength\ \sigma \wedge w\ (NNil\ (nnth\ \sigma\ k)) \wedge$ 
         $(LIFT(wnext\ (\Box\ (init\ (\neg\ w))))\ (ndropn\ k\ \sigma) \wedge (\forall n. \neg\ n < k \vee \neg\ w\ (NNil\ (nnth\ \sigma\ n))))$ 
        by (metis  $\langle enat\ k \leq nlength\ \sigma \wedge w\ (NNil\ (nnth\ \sigma\ k)) \wedge$ 
           $(LIFT(wnext\ (\Box\ (init\ (\neg\ w))))\ (ndropn\ k\ \sigma) \wedge (\forall j < k. \neg\ w\ (NNil\ (nnth\ \sigma\ j)))) \rangle$ )
      have  $(enat\ k \leq nlength\ \sigma \longrightarrow (\forall n. enat\ n \leq nlength\ \sigma \wedge k < n \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ n)))) =$ 
         $(LIFT(wnext\ (\Box\ (init\ (\neg\ w))))\ (ndropn\ k\ \sigma)) =$ 
         $(\neg\ enat\ k \leq nlength\ \sigma \vee (\forall n. (\neg\ enat\ n \leq nlength\ \sigma \vee \neg\ k < n) \vee \neg\ w\ (NNil\ (nnth\ \sigma\ n)))) =$ 
         $(LIFT(wnext\ (\Box\ (init\ (\neg\ w))))\ (ndropn\ k\ \sigma))$ 
        by blast
      then have  $\neg\ enat\ k \leq nlength\ \sigma \vee (\forall n. (\neg\ enat\ n \leq nlength\ \sigma \vee \neg\ k < n) \vee \neg\ w\ (NNil\ (nnth\ \sigma\ n))) =$ 
         $(LIFT(wnext\ (\Box\ (init\ (\neg\ w))))\ (ndropn\ k\ \sigma))$ 
        by (metis  $\langle enat\ k \leq nlength\ \sigma \implies (\forall j. enat\ j \leq nlength\ \sigma \longrightarrow k < j \longrightarrow \neg\ w\ (NNil\ (nnth\ \sigma\ j))) \rangle$ )
    =
       $(LIFT(wnext\ (\Box\ (init\ (\neg\ w))))\ (ndropn\ k\ \sigma))$ 
    then show ?thesis using f1 by presburger
  qed
  show ?thesis using 2 by blast
qed
qed
have 4:  $(\exists k \leq nlength\ \sigma. w\ (NNil\ (nnth\ \sigma\ k)) \wedge$ 
   $(LIFT(wnext\ (\Box\ (init\ (\neg\ w))))\ (ndropn\ k\ \sigma) \wedge$ 
   $(\forall j < k. \neg\ w\ (NNil\ (nnth\ \sigma\ j)))) =$ 
   $(\sigma \models (init\ (\neg\ w))\ \mathcal{U}\ ((init\ w) \wedge wnext\ (\Box\ (init\ (\neg\ w))))$ 
  by (simp add: NotStateUntilStateAndsem)
from 1 2 3 4 show ?thesis by auto
qed

```

lemma *PiEmpty*:

$\vdash (init\ w) \Pi\ empty = (init\ (\neg\ w))\ \mathcal{U}\ ((init\ w) \wedge wnext\ (\Box\ (init\ (\neg\ w))))$
using *PiEmptysem Valid-def* **by** blast

lemma *StatePiBoxStatesem*:

$\sigma \models (\text{init } w) \sqcap f = (\text{init } w) \sqcap (f \wedge \Box (\text{init } w))$

proof –

have 1: $(\sigma \models (\text{init } w) \sqcap f) =$

$((\exists x \in \text{nset } \sigma. w (\text{NNil } x)) \wedge ((\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma) \models f))$

by (*metis Pistate*)

have 2: $(\sigma \models (\text{init } w) \sqcap (f \wedge \Box (\text{init } w))) =$

$((\exists x \in \text{nset } \sigma. w (\text{NNil } x)) \wedge ((\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma) \models f \wedge \Box (\text{init } w)))$

by (*metis Pistate*)

have 3: $((\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma) \models f \wedge \Box (\text{init } w))$

$= (f (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma) \wedge$

$(\forall n \leq \text{nlength } (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma). w (\text{NNil } (\text{nnth } (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma) n))))$

by (*simp add: itl-defs ndropn-nfirst*)

have 4: $((\exists x \in \text{nset } \sigma. w (\text{NNil } x)) \wedge (f (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma) \wedge$

$(\forall n \leq \text{nlength } (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma). w (\text{NNil } (\text{nnth } (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma) n)))) =$

$((\exists x \in \text{nset } \sigma. w (\text{NNil } x)) \wedge f (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma))$

by (*meson nkfilter-nnth-aa*)

show ?thesis **using** 1 2 3 4 **by** *auto*

qed

lemma *StatePiBoxState*:

$\vdash (\text{init } w) \sqcap f = (\text{init } w) \sqcap (f \wedge \Box (\text{init } w))$

using *StatePiBoxStatesem Valid-def* **by** *blast*

lemma *StatePiUntil1*:

$\vdash ((\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge (\text{init } w) \sqcap f)) =$

$(\text{wprev } (\Box (\text{init } (\neg w)))) \frown ((\text{init } w) \wedge (\text{init } w) \sqcap f)$

using *StateUntilEqWPrevChop* **by** *blast*

lemma *StatePiUntilsem2*:

$(\sigma \models (\text{wprev } (\Box (\text{init } (\neg w)))) \frown ((\text{init } w) \wedge (\text{init } w) \sqcap f)) =$

$(\sigma \models ((\text{wprev } (\Box (\text{init } (\neg w)))) \frown ((\text{init } w) \wedge \text{empty})) \frown ((\text{init } w) \wedge (\text{init } w) \sqcap f))$

by (*auto simp add: schop-defs init-defs empty-defs zero-enat-def ndropn-nfirst*)

(metis (no-types, lifting) add.right-neutral dual-order.refl enat.simps(3) enat-add-sub-same

min.orderE min.orderE nfinite-ntaken nnth-nlast ntaken-nlast ntaken-nlength the-enat.simps

zero-enat-def)

lemma *StatePiUntil2*:

$\vdash (\text{wprev } (\Box (\text{init } (\neg w)))) \frown ((\text{init } w) \wedge (\text{init } w) \sqcap f) =$

$((\text{wprev } (\Box (\text{init } (\neg w)))) \frown ((\text{init } w) \wedge \text{empty})) \frown ((\text{init } w) \wedge (\text{init } w) \sqcap f)$

by (*simp add: StatePiUntilsem2 Valid-def*)

lemma *StatePiUntil3*:

$\vdash ((\text{wprev } (\Box (\text{init } (\neg w)))) \frown ((\text{init } w) \wedge \text{empty})) =$

$((\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge \text{wnext } (\Box (\text{init } (\neg w))))) \frown ((\text{init } w) \wedge \text{empty})$

proof –

have 1: $((\text{wprev } (\Box (\text{init } (\neg w)))) \frown ((\text{init } w) \wedge \text{empty})) =$

$(\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge \text{empty})$

by (*meson Prop11 StateUntilEqWPrevChop*)

have 2: $\vdash ((init\ w) \wedge empty) = ((init\ w) \wedge wnext\ (\Box (init\ (\neg w)))) \frown ((init\ w) \wedge empty)$
by (auto simp add: Valid-def itl-defs zero-enat-def ndropn-nfirst)
 (metis ndropn-0 ndropn-nfirst ,
 metis add.right-neutral add-diff-inverse-nat enat.simps(3) enat-add-sub-same enat-ord-simps(2)
 illess-Suc-eq less-eqE min.absorb2 min-def ntaken-all one-eSuc one-enat-def order-refl
 plus-1-eq-Suc zero-enat-def zero-le)
show ?thesis **by** (metis 1 2 UntilChopDist inteq-reflection)
qed

lemma StatePiUntilsem4:

$(\sigma \models ((init\ (\neg w)) \mathcal{U} ((init\ w) \wedge wnext\ (\Box (init\ (\neg w))))) \frown ((init\ w) \wedge empty)) =$
 $(\sigma \models ((init\ w) \Pi empty) \frown ((init\ w) \wedge empty))$
by (metis PiEmpty inteq-reflection)

lemma StatePiUntil4:

$\vdash ((init\ (\neg w)) \mathcal{U} ((init\ w) \wedge wnext\ (\Box (init\ (\neg w))))) \frown ((init\ w) \wedge empty) =$
 $((init\ w) \Pi empty) \frown ((init\ w) \wedge empty)$
by (simp add: StatePiUntilsem4 Valid-def)

lemma StatePiUntilsem:

$\sigma \models (init\ w) \Pi f = (init\ (\neg w)) \mathcal{U} ((init\ w) \wedge (init\ w) \Pi f)$

proof –

have 2: $(\sigma \models (init\ (\neg w)) \mathcal{U} ((init\ w) \wedge (init\ w) \Pi f)) =$
 $(\sigma \models (wprev\ (\Box (init\ (\neg w)))) \frown ((init\ w) \wedge (init\ w) \Pi f))$
using StateUntilEqvWPrevChopsem[of LIFT($\neg w$) LIFT($(init\ w) \wedge (init\ w) \Pi f$) σ]
by simp

have 7: $(\sigma \models (wprev\ (\Box (init\ (\neg w)))) \frown ((init\ w) \wedge (init\ w) \Pi f)) =$
 $(\sigma \models (((init\ w) \Pi empty) \frown ((init\ w) \wedge empty)) \frown ((init\ w) \wedge (init\ w) \Pi f))$

by (metis PiEmpty StatePiUntil2 StatePiUntil3 inteq-reflection)

have 8: $(\sigma \models (((init\ w) \Pi empty) \frown ((init\ w) \wedge empty)) \frown ((init\ w) \wedge (init\ w) \Pi f)) =$
 $(\sigma \models (((init\ w) \Pi empty)) \frown ((init\ w) \wedge (init\ w) \Pi f))$

by (auto simp add: schop-defs init-defs empty-defs zero-enat-def ndropn-nfirst)
 (metis (no-types, opaque-lifting) enat-0-iff(2) min.absorb-iff1 ndropn-all ndropn-nlength
 nle-le nlength-NNil ntaken-nlast ntaken-nlength ntaken-ntaken)

have 9: $(\sigma \models (((init\ w) \Pi empty)) \frown ((init\ w) \wedge (init\ w) \Pi f)) =$
 $(\sigma \models (init\ w) \Pi (empty \frown f))$

using PiSChopDistsema PiSChopDistsemb **by** blast

have 10: $(\sigma \models (init\ w) \Pi (empty \frown f)) = (\sigma \models (init\ w) \Pi f)$

by (metis EmptySChop inteq-reflection)

show ?thesis

by (simp add: 10 2 7 8 9)

qed

lemma StatePiUntil:

$\vdash (init\ w) \Pi f = (init\ (\neg w)) \mathcal{U} ((init\ w) \wedge (init\ w) \Pi f)$

using StatePiUntilsem **by** blast

lemma StateAndPiEmpty:

$\vdash ((init\ w) \wedge (init\ w) \Pi empty) = (w \wedge empty) \frown (wnext\ (\Box (init\ (\neg w))))$

proof –

have 1: $\vdash ((init\ w) \wedge (init\ w) \Pi\ empty) =$
 $((init\ w) \wedge (init\ (\neg\ w)) \mathcal{U}\ ((init\ w) \wedge (wnext\ (\Box\ (init\ (\neg\ w))))))$
using *PiEmpty* **by** *fastforce*
have 2: $\vdash ((init\ w) \wedge (init\ (\neg\ w)) \mathcal{U}\ ((init\ w) \wedge (wnext\ (\Box\ (init\ (\neg\ w))))))$
 $= ((init\ w) \wedge (wnext\ (\Box\ (init\ (\neg\ w)))))$
by (*auto simp add: until-d-def Valid-def init-defs ndropn-nfirst*)
 $(metis\ ndropn-0\ ndropn-nfirst\ neq0-conv,$
 $metis\ gr-implies-not-zero\ ndropn-0\ ndropn-nfirst\ zero-enat-def\ zero-le)$
have 3: $\vdash ((init\ w) \wedge (wnext\ (\Box\ (init\ (\neg\ w))))) = (w \wedge empty) \frown (wnext\ (\Box\ (init\ (\neg\ w)))))$
proof –
have $\bigwedge p\ pa. \vdash ((p::'a\ nellist \Rightarrow bool) \wedge empty) \frown pa = (init\ p \wedge pa)$
by (*metis InitAndEmptyEqvAndEmpty StateAndEmptySChop inteq-reflection*)
then show *?thesis*
by (*simp add: Prop11*)
qed
show *?thesis*
by (*metis 1 2 3 inteq-reflection*)
qed

lemma *PiFPowerExpandsem:*

$(\sigma \models (\exists k. (init\ w) \Pi (fpower\ f\ k))) =$
 $(\sigma \models (init\ w) \Pi\ empty \vee (\exists k. (init\ w) \Pi (f \frown (fpower\ f\ (k)))))$
proof –
have 1: $(\sigma \models (\exists k. (init\ w) \Pi (fpower\ f\ k))) =$
 $(\exists k. (\sigma \models (init\ w) \Pi (fpower\ f\ k)))$
by *simp*
have 2: $(\exists k. (\sigma \models (init\ w) \Pi (fpower\ f\ k))) =$
 $((\sigma \models (init\ w) \Pi (fpower\ f\ 0)) \vee (\exists k. 1 \leq k \wedge (\sigma \models (init\ w) \Pi (fpower\ f\ (k)))))$
by (*metis One-nat-def diff-Suc-1 le-SucE le-add1 plus-1-eq-Suc*)
have 3: $(\sigma \models (init\ w) \Pi (fpower\ f\ 0)) = (\sigma \models (init\ w) \Pi\ empty)$
by (*simp add: itl-def fpower-d-def*)
have 4: $(\exists k. 1 \leq k \wedge (\sigma \models (init\ w) \Pi (fpower\ f\ (k)))) =$
 $(\exists k. (\sigma \models (init\ w) \Pi (fpower\ f\ (Suc\ k))))$
by (*metis le-add1 ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc*)
have 5: $(\exists k. (\sigma \models (init\ w) \Pi (fpower\ f\ (Suc\ k)))) =$
 $(\sigma \models (\exists k. (init\ w) \Pi (fpower\ f\ (Suc\ k))))$
by *simp*
have 6: $((\sigma \models (init\ w) \Pi\ empty) \vee (\sigma \models (\exists k. (init\ w) \Pi (fpower\ f\ (Suc\ k))))) =$
 $(\sigma \models (init\ w) \Pi\ empty \vee (\exists k. (init\ w) \Pi (f \frown fpower\ f\ (k))))$
unfolding *fpower-d-def* **by** (*auto simp add: schop-d-def*)
show *?thesis* **using** 1 2 3 4 5 6 **by** *blast*
qed

lemma *PiFPowerExpandsem1:*

$\forall \sigma. \sigma \models (\exists k. (init\ w) \Pi (fpower\ f\ k)) =$
 $((init\ w) \Pi\ empty \vee (\exists k. (init\ w) \Pi (fpower\ f\ (Suc\ k))))$
proof
fix σ
show $\sigma \models (\exists k. (init\ w) \Pi (fpower\ f\ k)) =$
 $((init\ w) \Pi\ empty \vee (\exists k. (init\ w) \Pi (fpower\ f\ (Suc\ k))))$

```

proof –
  have 1: ( $\sigma \models (\exists k. (init\ w) \sqcap (fpower\ f\ k)) =$ 
    ( $(init\ w) \sqcap empty \vee (\exists k. (init\ w) \sqcap (fpower\ f\ (Suc\ k)) \ ) \ )$ )
    = ( $\sigma \models (\exists k. (init\ w) \sqcap (fpower\ f\ k))$ ) =
    ( $\sigma \models (init\ w) \sqcap empty \vee (\exists k. (init\ w) \sqcap (fpower\ f\ (Suc\ k)) \ ) \ )$ )

  by auto
  have 2: ( $\sigma \models (\exists k. (init\ w) \sqcap (fpower\ f\ k))$ ) =
    ( $\sigma \models (init\ w) \sqcap empty \vee (\exists k. (init\ w) \sqcap (fpower\ f\ (Suc\ k)) \ ) \ )$ )
  using PiFPowerExpandsem[of w f  $\sigma$ ]
  unfolding fpower-d-def by (auto simp add: schop-d-def)
  show ?thesis using 1 2 by blast
qed
qed

```

lemma *PiFPowerExpand*:

```

 $\vdash (\exists k. (init\ w) \sqcap (fpower\ f\ k)) =$ 
  ( $(init\ w) \sqcap empty \vee (\exists k. (init\ w) \sqcap (fpower\ f\ (Suc\ k)) \ ) \ )$ )
using PiFPowerExpandsem1[of w f ] by (auto simp add: Valid-def PiFPowerExpandsem1)

```

lemma *exists-expand-sem*:

```

( $\sigma \models (\exists k. (fpower\ ((init\ w) \sqcap f \wedge fin\ w)\ k))$ ) =
  (( $\sigma \models (fpower\ ((init\ w) \sqcap f \wedge fin\ w)\ 0)$ )  $\vee$ 
  ( $\sigma \models (\exists k. (fpower\ ((init\ w) \sqcap f \wedge fin\ w)\ (Suc\ k))$ )))
by (metis (no-types, lifting) not0-implies-Suc unl-Rex)

```

lemma *exists-expand*:

```

 $\vdash (\exists k. (fpower\ ((init\ w) \sqcap f \wedge fin\ w)\ k)) =$ 
  (( $fpower\ ((init\ w) \sqcap f \wedge fin\ w)\ 0$ )  $\vee (\exists k. (fpower\ ((init\ w) \sqcap f \wedge fin\ w)\ (Suc\ k))$ ))
using exists-expand-sem Valid-def by fastforce

```

TruePiEqv

lemma *TruePiEqvsem*:

```

 $\sigma \models \#True \sqcap f = f$ 
by (simp add: pi-d-def pifilt-true) (metis zero-enat-def zero-le)

```

lemma *TruePiEqv*:

```

 $\vdash (\#True) \sqcap f = f$ 
using TruePiEqvsem by (auto simp add: Valid-def)

```

BoxImpEqvPi

lemma *BoxImpEqvPi*:

```

 $\vdash \Box f \longrightarrow g = f \sqcap g$ 
proof (simp add: Valid-def itl-defs pi-d-def pifilt-def sfxfilt-def)
show  $\forall w. (\forall n. enat\ n \leq nlength\ w \longrightarrow f\ (ndropn\ n\ w)) \longrightarrow$ 
   $g\ w = ((\exists i. enat\ i \leq nlength\ w \wedge f\ (ndropn\ i\ w)) \wedge$ 
     $g\ (nmap\ (\lambda s. nnth\ s\ 0)\ (nfilter\ f\ (ndropns\ w))))$ 
proof
  fix w

```

```

show ( $\forall n. \text{enat } n \leq \text{nlength } w \longrightarrow f (\text{ndropn } n \ w) \longrightarrow$ 
 $g \ w = ((\exists i. \text{enat } i \leq \text{nlength } w \wedge f (\text{ndropn } i \ w)) \wedge$ 
 $g (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f (\text{ndropns } w))))$ )
proof
  assume  $a0: \forall n. \text{enat } n \leq \text{nlength } w \longrightarrow f (\text{ndropn } n \ w)$ 
  show  $g \ w = ((\exists i. \text{enat } i \leq \text{nlength } w \wedge f (\text{ndropn } i \ w)) \wedge$ 
 $g (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f (\text{ndropns } w))))$ 
  proof –
    have  $1: (\text{nfilter } f (\text{ndropns } w)) = (\text{ndropns } w)$ 
      by (metis (mono-tags, lifting) a0 ile0-eq in-nset-ndropns le-cases nfilter-id-conv
 $\text{zero-enat-def}$ )
    have  $2: w = (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f (\text{ndropns } w)))$ 
      by (simp add: 1 nmap-first-ndropns)
    show ?thesis by (metis 1 a0 nmap-first-ndropns zero-enat-def zero-le)
  qed
qed
qed
qed

```

PiEqvDiamondUPi

lemma *PiEqvDiamondUPi*:
 $\vdash f \ \Pi \ g = (\Diamond f \wedge f \ \Pi^u \ g)$
by (*simp add: Valid-def upi-d-def itl-defs pi-d-def,blast*)

PiEqvUntilPi

lemma *PiEqvUntilPi*:
 $\vdash (\text{init } w) \ \Pi \ g = (\text{init } (\neg w)) \ \mathcal{U} ((\text{init } w) \ \Pi \ g)$
by (*metis StatePiUntil UntilUntil int-eq*)

UPiEqvBoxOrPi

lemma *UPiEqvBoxOrPi*:
 $\vdash f \ \Pi^u \ g = (\Box (\neg f) \vee f \ \Pi \ g)$
by (*simp add: Valid-def upi-d-def itl-defs pi-d-def,blast*)

2.9.4 Theorems

lemma *UPiImpRule*:
assumes $\vdash g1 \longrightarrow g2$
shows $\vdash f \ \Pi^u \ g1 \longrightarrow f \ \Pi^u \ g2$
using *assms*
by (*meson MP PiK PiN*)

lemma *UPiEqvRule*:
assumes $\vdash g1 = g2$
shows $\vdash f \ \Pi^u \ g1 = f \ \Pi^u \ g2$
proof –
have $1: \vdash g1 \longrightarrow g2$
using *assms* **by** (*simp add: int-iffD1*)

have 2: $\vdash f \Pi^u g1 \longrightarrow f \Pi^u g2$
using 1 *UPiImpRule* **by** *blast*
have 3: $\vdash g2 \longrightarrow g1$
using *assms* **by** (*simp add: int-iffD2*)
have 4: $\vdash f \Pi^u g2 \longrightarrow f \Pi^u g1$
using 3 *UPiImpRule* **by** *blast*
from 3 4 **show** *?thesis*
by (*simp add: 2 int-iffI*)
qed

lemma *PiEqvNotUPiNot*:
 $\vdash f \Pi g = (\neg (f \Pi^u (\neg g)))$
by (*simp add: upi-d-def*)

lemma *NotPiEqvNotUPi*:
 $\vdash f \Pi (\neg g) = (\neg (f \Pi^u g))$
by (*simp add: upi-d-def*)

lemma *UPiEqvNotPiNot*:
 $\vdash f \Pi^u g = (\neg (f \Pi (\neg g)))$
by (*simp add: upi-d-def*)

lemma *NotUPiEqvNotPi*:
 $\vdash f \Pi^u (\neg g) = (\neg (f \Pi g))$
by (*simp add: upi-d-def*)

lemma *PiImpRule*:
assumes $\vdash g1 \longrightarrow g2$
shows $\vdash f \Pi g1 \longrightarrow f \Pi g2$
proof –
have 1: $\vdash \neg g2 \longrightarrow \neg g1$
by (*simp add: assms*)
have 2: $\vdash f \Pi^u (\neg g2) \longrightarrow f \Pi^u (\neg g1)$
using 1 *UPiImpRule* **by** *blast*
have 3: $\vdash \neg(f \Pi^u (\neg g1)) \longrightarrow \neg(f \Pi^u (\neg g2))$
using 2 **by** *fastforce*
from 3 **show** *?thesis* **using** *PiEqvNotUPiNot* **by** *fastforce*
qed

lemma *PiEqvRule*:
assumes $\vdash g1 = g2$
shows $\vdash f \Pi g1 = f \Pi g2$
proof –
have 1: $\vdash g1 \longrightarrow g2$
using *assms* **by** (*simp add: int-iffD1*)
have 2: $\vdash f \Pi g1 \longrightarrow f \Pi g2$
using 1 *PiImpRule* **by** *blast*
have 3: $\vdash g2 \longrightarrow g1$
using *assms* **by** (*simp add: int-iffD2*)
have 4: $\vdash f \Pi g2 \longrightarrow f \Pi g1$

using 3 PiImpRule by blast
 from 2 4 show ?thesis by (simp add: int-iffI)
 qed

lemma UPiAndPiImpPiAnd:

$\vdash f1 \Pi^u f \wedge f1 \Pi (\neg g) \longrightarrow f1 \Pi (f \wedge \neg g)$

proof –

have 1: $\vdash (\neg(f \longrightarrow g)) = (f \wedge \neg g)$

by fastforce

have 2: $\vdash (\neg(f1 \Pi^u (f \longrightarrow g))) = f1 \Pi (\neg(f \longrightarrow g))$

by (simp add: NotPiEqvNotUPi int-iffD1 int-iffD2 int-iffI)

have 3: $\vdash \neg(f1 \Pi^u f \longrightarrow f1 \Pi^u g) \longrightarrow \neg(f1 \Pi^u (f \longrightarrow g))$

by (simp add: PiK)

have 4: $\vdash (\neg(f1 \Pi^u f \longrightarrow f1 \Pi^u g)) = (f1 \Pi^u f \wedge f1 \Pi (\neg g))$

using NotPiEqvNotUPi[of f1 g] by fastforce

have 5: $\vdash f1 \Pi (\neg(f \longrightarrow g)) = f1 \Pi (f \wedge \neg g)$

using 1 by (simp add: PiEqvRule)

from 1 2 3 4 5 show ?thesis by fastforce

qed

lemma UPiAndPiImpPiAndA:

$\vdash f1 \Pi^u f \wedge f1 \Pi g \longrightarrow f1 \Pi (f \wedge g)$

using UPiAndPiImpPiAnd[of f1 f LIFT($\neg g$)] by fastforce

lemma PiAndPiImpPiAnd:

$\vdash f1 \Pi f \wedge f1 \Pi g \longrightarrow f1 \Pi (f \wedge g)$

proof –

have 1: $\vdash f1 \Pi^u f \wedge f1 \Pi g \longrightarrow f1 \Pi (f \wedge g)$

using UPiAndPiImpPiAndA by fastforce

have 2: $\vdash f1 \Pi f \longrightarrow f1 \Pi^u f$

using PiDc by blast

from 1 2 show ?thesis by fastforce

qed

lemma PiAnd:

$\vdash f \Pi (g1 \wedge g2) = (f \Pi g1 \wedge f \Pi g2)$

proof –

have 1: $\vdash f \Pi (g1 \wedge g2) \longrightarrow f \Pi g1$

by (meson PiImpRule Prop12 int-iffD1 lift-and-com)

have 2: $\vdash f \Pi (g1 \wedge g2) \longrightarrow f \Pi g2$

by (meson PiImpRule Prop12 int-iffD1 lift-and-com)

have 3: $\vdash f \Pi (g1 \wedge g2) \longrightarrow f \Pi g1 \wedge f \Pi g2$

using 1 2 by fastforce

have 4: $\vdash f \Pi g1 \wedge f \Pi g2 \longrightarrow f \Pi (g1 \wedge g2)$

by (simp add: PiAndPiImpPiAnd)

from 3 4 show ?thesis by fastforce

qed

lemma UPiAnd:

$\vdash f \Pi^u (g1 \wedge g2) = (f \Pi^u g1 \wedge f \Pi^u g2)$

proof –

have 1: $\vdash f \Pi (\neg g1 \vee \neg g2) = (f \Pi (\neg g1) \vee f \Pi (\neg g2))$
 by (*simp add: PiOr*)
have 2: $\vdash (\neg(f \Pi (\neg g1 \vee \neg g2))) = (\neg(f \Pi (\neg g1) \vee f \Pi (\neg g2)))$
 using 1 by *fastforce*
have 3: $\vdash (\neg(f \Pi (\neg g1 \vee \neg g2))) = f \Pi^u (\neg(\neg g1 \vee \neg g2))$
 by (*meson NotUPiEqvNotPi Prop11*)
have 4: $\vdash (\neg(\neg g1 \vee \neg g2)) = (g1 \wedge g2)$
 by *fastforce*
have 5: $\vdash f \Pi^u (\neg(\neg g1 \vee \neg g2)) = f \Pi^u (g1 \wedge g2)$
 using 4 by (*simp add: UPiEqvRule*)
have 6: $\vdash (\neg(f \Pi (\neg g1) \vee f \Pi (\neg g2))) = (\neg(f \Pi (\neg g1)) \wedge \neg(f \Pi (\neg g2)))$
 by *fastforce*
have 7: $\vdash \neg(f \Pi (\neg g1)) = f \Pi^u g1$
 by (*simp add: NotPiEqvNotUPi*)
have 8: $\vdash \neg(f \Pi (\neg g2)) = f \Pi^u g2$
 by (*simp add: NotPiEqvNotUPi*)
have 9: $\vdash (\neg(f \Pi (\neg g1)) \wedge \neg(f \Pi (\neg g2))) = (f \Pi^u g1 \wedge f \Pi^u g2)$
 using 6 7 8 by *fastforce*
show ?thesis by (*metis 2 3 5 6 9 inteq-reflection*)
qed

lemma *UPiOr*:

$\vdash f \Pi^u (g1 \vee g2) = (f \Pi^u g1 \vee f \Pi^u g2)$

proof –

have 1: $\vdash f \Pi (\neg g1 \wedge \neg g2) = (f \Pi (\neg g1) \wedge f \Pi (\neg g2))$
 by (*simp add: PiAnd*)
have 2: $\vdash (\neg(f \Pi (\neg g1 \wedge \neg g2))) = (\neg(f \Pi (\neg g1) \wedge f \Pi (\neg g2)))$
 using 1 by *fastforce*
have 3: $\vdash (\neg(f \Pi (\neg g1 \wedge \neg g2))) = f \Pi^u (\neg(\neg g1 \wedge \neg g2))$
 by (*meson NotUPiEqvNotPi Prop11*)
have 4: $\vdash (\neg(\neg g1 \wedge \neg g2)) = (g1 \vee g2)$
 by *fastforce*
have 5: $\vdash f \Pi^u (\neg(\neg g1 \wedge \neg g2)) = f \Pi^u (g1 \vee g2)$
 using 4 *UPiEqvRule* by *blast*
have 6: $\vdash (\neg(f \Pi (\neg g1) \wedge f \Pi (\neg g2))) = (\neg(f \Pi (\neg g1)) \vee \neg(f \Pi (\neg g2)))$
 by *fastforce*
have 7: $\vdash (\neg(f \Pi (\neg g1))) = f \Pi^u g1$
 by (*simp add: upi-d-def*)
have 8: $\vdash (\neg(f \Pi (\neg g2))) = f \Pi^u g2$
 by (*simp add: upi-d-def*)
have 9: $\vdash (\neg(f \Pi (\neg g1)) \vee \neg(f \Pi (\neg g2))) = (f \Pi^u g1 \vee f \Pi^u g2)$
 using 7 8 by *fastforce*
show ?thesis
 by (*metis 2 3 4 6 9 inteq-reflection*)
qed

lemma *UpiAndImp*:

$\vdash f \Pi^u (g1 \longrightarrow g2) \wedge f \Pi g1 \longrightarrow f \Pi g2$

proof –
have 2: $\vdash f \Pi^u ((\neg g2) \longrightarrow (\neg g1)) \longrightarrow (f \Pi^u (\neg g2) \longrightarrow f \Pi^u (\neg g1))$
using *PiK* **by** *blast*
have 3: $\vdash (f \Pi^u (\neg g2) \longrightarrow f \Pi^u (\neg g1)) = ((\neg (f \Pi^u (\neg g1))) \longrightarrow (\neg (f \Pi^u (\neg g2))))$
by *auto*
have 4: $\vdash (\neg (f \Pi^u (\neg g2))) = f \Pi g2$
by (*simp add: upi-d-def*)
have 5: $\vdash (\neg (f \Pi^u (\neg g1))) = f \Pi g1$
by (*simp add: upi-d-def*)
have 6: $\vdash f \Pi^u ((\neg g2) \longrightarrow (\neg g1)) = f \Pi^u (g1 \longrightarrow g2)$
by *simp*
have 7: $\vdash (f \Pi^u (g1 \longrightarrow g2) \wedge f \Pi g1 \longrightarrow f \Pi g2) =$
 $(f \Pi^u (g1 \longrightarrow g2) \longrightarrow (f \Pi g1 \longrightarrow f \Pi g2))$
by *auto*
show *?thesis*
using 2 4 5 **by** *fastforce*
qed

lemma *BoxImpUPiBox*:
 $\vdash \Box (init\ w) \longrightarrow f \Pi^u (\Box (init\ w))$
proof –
have 1: $\vdash f \Pi (\Diamond (init\ (\neg w))) \longrightarrow \Diamond (init\ (\neg w))$
by (*simp add: PiDiamondImp*)
have 2: $\vdash \neg \Diamond (init\ (\neg w)) \longrightarrow \neg (f \Pi (\Diamond (init\ (\neg w))))$
using 1 **by** *auto*
have 3: $\vdash (\neg \Diamond (init\ (\neg w))) = \Box (init\ w)$
by (*metis 2 Initprop(2) Prop10 always-d-def inteq-reflection*)
have 4: $\vdash (\neg (f \Pi (\Diamond (init\ (\neg w)))) = f \Pi^u (\Box (init\ w))$
by (*simp add: upi-d-def*)
 $(metis\ 3\ int-simps(4)\ inteq-reflection)$
show *?thesis*
using 2 3 4 **by** *fastforce*
qed

lemma *WPrevPi*:
 $\vdash (init\ w) \Pi f = (wprev\ (\Box (init\ (\neg w)))) \frown ((init\ w) \wedge (init\ w) \Pi f)$
using *StatePiUntil StatePiUntil1* **by** *fastforce*

lemma *EmptyAndSChopAndMoreEqvAndSChop*:
 $\vdash (w \wedge empty) \frown (f \wedge more) = ((w \wedge empty) \frown f \wedge more)$
proof –
have 1: $\vdash (w \wedge empty) \frown (f \wedge more) \longrightarrow (w \wedge empty) \frown f$
by (*simp add: SChopAndA*)
have 2: $\vdash (w \wedge empty) \frown (f \wedge more) \longrightarrow more$
by (*meson SChopAndB SChopMoreImpMore lift-imp-trans*)
have 3: $\vdash ((w \wedge empty) \frown f \wedge more) \longrightarrow (w \wedge empty) \frown (f \wedge more)$
by (*metis (no-types, opaque-lifting) InitAndEmptyEqvAndEmpty Prop11 Prop12 StateAndEmptySChop int-simps(1) inteq-reflection*)
show *?thesis*
by (*simp add: 1 2 3 Prop12 int-iffI*)

qed

lemma *PiInfImpInf*:

$\vdash f \Pi \text{ inf} \longrightarrow \text{inf}$

unfolding *Valid-def pi-d-def init-defs infinite-defs*

by *auto*

(*metis enat-ile nfinite-conv-nlength-enat pifilt-nlength-bound*)

lemma *DiamondWPrevBoxSChop*:

$\vdash \Diamond (\text{init } w) = (w_{\text{prev}} (\Box (\text{init } (\neg w)))) \frown (\text{init } w)$

by (*metis Initprop(2) StateUntilEqvWPrevChop UntilRule inteq-reflection*)

lemma *PiChopDist1*:

$\vdash ((\text{init } w) \Pi (f;g) \vee ((\text{init } w) \Pi (f \wedge \text{finite}) \wedge \text{inf})) =$
 $((\text{init } w) \Pi f);((\text{init } w) \wedge (\text{init } w) \Pi g)$

proof –

have 1: $\vdash f;g = (f \frown g \vee (f \wedge \text{inf}))$

by (*simp add: ChopSChopdef*)

have 2: $\vdash (\text{init } w) \Pi (f;g) = ((\text{init } w) \Pi (f \frown g) \vee (\text{init } w) \Pi (f \wedge \text{inf}))$

by (*metis 1 PiOr int-eq*)

have 3: $\vdash (\text{init } w) \Pi (f \frown g) = ((\text{init } w) \Pi f) \frown ((\text{init } w) \wedge (\text{init } w) \Pi g)$

by (*simp add: PiSChopDist*)

have 4: $\vdash ((\text{init } w) \Pi f);((\text{init } w) \wedge (\text{init } w) \Pi g) =$
 $((\text{init } w) \Pi f) \frown ((\text{init } w) \wedge (\text{init } w) \Pi g) \vee ((\text{init } w) \Pi f \wedge \text{inf}))$

by (*simp add: ChopSChopdef*)

have 41: $\vdash f = (f \wedge (\text{finite} \vee \text{inf}))$

unfolding *finite-d-def* **by** *fastforce*

have 5: $\vdash (\text{init } w) \Pi f = (\text{init } w) \Pi (f \wedge (\text{finite} \vee \text{inf}))$

by (*simp add: 41 PiEqvRule*)

have 6: $\vdash ((\text{init } w) \Pi f \wedge \text{inf}) =$
 $((\text{init } w) \Pi (f \wedge \text{finite}) \wedge \text{inf}) \vee ((\text{init } w) \Pi (f \wedge \text{inf}) \wedge \text{inf}))$

by (*metis AndInfEqvChopFalse OrChopEqvRule OrFiniteInf PiOr inteq-reflection*)

have 7: $\vdash ((\text{init } w) \Pi (f \wedge \text{inf}) \wedge \text{inf}) = ((\text{init } w) \Pi (f \wedge \text{inf}))$

by (*metis EmptySChop PiImpRule PiInfImpInf Prop01 Prop04 Prop05 Prop10 int-iffD1 lift-imp-trans*)

show *?thesis*

using 2 3 4 6 7 **by** *fastforce*

qed

lemma *PiChopDist2*:

$\vdash ((\text{init } w) \Pi (f;g)) =$

$((\text{init } w) \Pi (f \wedge \text{finite})) \frown ((\text{init } w) \wedge (\text{init } w) \Pi g) \vee ((\text{init } w) \Pi (f \wedge \text{inf}) \wedge \text{inf}))$

proof –

have 1: $\vdash (\text{init } w) \Pi (f;g) = ((\text{init } w) \Pi (f \frown g) \vee (\text{init } w) \Pi (f \wedge \text{inf}))$

by (*metis ChopSChopdef PiOr inteq-reflection*)

have 2: $\vdash (\text{init } w) \Pi (f \frown g) = ((\text{init } w) \Pi f) \frown ((\text{init } w) \wedge (\text{init } w) \Pi g)$

by (simp add: PiSchopDist)
 have 3: $\vdash ((init\ w) \Pi f) \frown ((init\ w) \wedge (init\ w) \Pi g) =$
 $((init\ w) \Pi f \wedge finite); ((init\ w) \wedge (init\ w) \Pi g)$
 by (simp add: schop-d-def)
 have 31: $\vdash f = (f \wedge finite) \vee (f \wedge inf)$
 unfolding finite-d-def by fastforce
 have 32: $\vdash (init\ w) \Pi f = (init\ w) \Pi (f \wedge finite) \vee (f \wedge inf)$
 using 31 PiEqvRule by blast
 have 33: $\vdash (init\ w) \Pi (f \wedge finite) \vee (f \wedge inf) = ((init\ w) \Pi (f \wedge finite) \vee (init\ w) \Pi (f \wedge inf))$
 by (simp add: PiOr)
 have 34: $\vdash \neg((init\ w) \Pi (f \wedge inf) \wedge finite)$
 unfolding finite-d-def using PiAnd[of LIFT init w f LIFT inf] PiInfImpInf[of LIFT init w]
 by fastforce
 have 4: $\vdash ((init\ w) \Pi f \wedge finite) = ((init\ w) \Pi (f \wedge finite) \wedge finite)$
 using 32 33 34 by fastforce
 have 5: $\vdash (init\ w) \Pi (f \wedge inf) = ((init\ w) \Pi (f \wedge inf) \wedge inf)$
 by (metis PiAnd PiInfImpInf Prop10 Prop12 int-eq int-iffD2)
 show ?thesis
 by (metis 1 2 4 5 int-eq schop-d-def)
 qed

lemma InfEqvNotForallNotLen:

$\vdash inf = (\forall n. \neg (len\ n))$

proof –

have 1: $\vdash finite = (\exists n. len\ n)$

by (simp add: Finite-exist-len)

have 2: $\vdash (\neg finite) = (\neg(\exists n. len\ n))$

using 1

by (metis NotEqvYieldsMore int-eq)

have 3: $\vdash (\neg(\exists n. len\ n)) = (\forall n. \neg (len\ n))$

by fastforce

show ?thesis using 2 3 by (metis InfEqvNotFinite int-eq)

qed

lemma PiEx:

$\vdash f \Pi (\exists n. g\ n) = (\exists n. f \Pi (g\ n))$

unfolding Valid-def by (auto simp add: pi-d-def)

lemma PiAll:

$\vdash f \Pi (\forall n. g\ n) = (\forall n. f \Pi (g\ n))$

unfolding Valid-def by (auto simp add: pi-d-def)

lemma PiLenSuc:

$\vdash (init\ w) \Pi (len\ (Suc\ n)) = ((init\ w) \Pi skip) \frown ((init\ w) \wedge (init\ w) \Pi (len\ n))$

proof –

have 1: $\vdash len\ (Suc\ n) = skip \frown (len\ n)$

by (metis NextSchopdef len-d-def next-d-def wpow-Suc)

have 2: $\vdash (init\ w) \Pi (skip \frown (len\ n)) = ((init\ w) \Pi skip) \frown ((init\ w) \wedge (init\ w) \Pi (len\ n))$
by (*simp add: PiSCHopDist*)
show ?thesis **by** (*metis 1 2 int-eq*)
qed

lemma *PiImpDiamond*:
 $\vdash f \Pi g \longrightarrow \Diamond f$
by (*meson PiEqvDiamondUPi Prop12 int-iffD1*)

lemma *PiMPA*:
assumes $\vdash f \Pi g1$
 $\vdash f \Pi (g1 \longrightarrow g2)$
shows $\vdash f \Pi g2$
using *assms*
by (*meson MP PiDc Prop09 UpiAndImp*)

lemma *PiMPB*:
assumes $\vdash f \Pi g1$
 $\vdash g1 \longrightarrow g2$
shows $\vdash f \Pi g2$
using *assms*
by (*metis MP PiAnd Prop10 Prop12 int-iffD2 inteq-reflection*)

lemma *PiMPBC*:
assumes $\vdash f1 \longrightarrow f2 \Pi g1$
 $\vdash g1 \longrightarrow g2$
shows $\vdash f1 \longrightarrow f2 \Pi g2$
using *assms*
by (*meson PiImpRule lift-imp-trans*)

lemma *SlideInitSFin*:
 $\vdash f \frown ((init\ w) \wedge g) = (f \wedge sfin\ w) \frown g$
proof –
have 1: $\vdash sfin\ (init\ w) = sfin\ (w)$
by (*metis InitAndEmptyEqvAndEmpty SFinEqvTrueSCHopAndEmpty int-eq*)
show ?thesis
by (*metis 1 AndSFinSCHopEqvStateAndSCHop inteq-reflection*)
qed

lemma *UPiSCHop*:
 $\vdash (init\ w) \Pi^u (f \frown g) = (\Box (init\ (\neg w)) \vee ((init\ w) \Pi f \wedge sfin\ w) \frown ((init\ w) \Pi g))$
proof –
have 1: $\vdash (init\ w) \Pi^u (f \frown g) = (\Box (init\ (\neg w)) \vee (init\ w) \Pi (f \frown g))$
by (*metis Initprop(2) UPiEqvBoxOrPi int-eq*)

have 2: $\vdash (init\ w) \Pi (f \frown g) = ((init\ w) \Pi f) \frown ((init\ w) \wedge (init\ w) \Pi g)$
by (*simp add: PiSChopDist*)
have 3: $\vdash ((init\ w) \Pi f) \frown ((init\ w) \wedge (init\ w) \Pi g) =$
 $((init\ w) \Pi f \wedge sfin\ w) \frown ((init\ w) \Pi g)$
by (*simp add: SlideInitSFin*)
show ?thesis
using 1 2 3 **by** (*meson Prop06*)
qed

lemma *PiUPiSChop*:

$\vdash ((init\ w) \Pi f \wedge sfin\ w) \frown ((init\ w) \Pi g) = ((init\ w) \Pi^u f \wedge sfin\ w) \frown ((init\ w) \Pi^u g)$

proof –

have 1: $\vdash ((init\ w) \Pi f) = (\Diamond (init\ w) \wedge (init\ w) \Pi^u f)$
by (*simp add: PiEqvDiamondUPi*)
have 2: $\vdash ((init\ w) \Pi f \wedge sfin\ w) = (\Diamond (init\ w) \wedge (init\ w) \Pi^u f \wedge sfin\ w)$
using 1 **by** *auto*
have 3: $\vdash (\Diamond (init\ w) \wedge sfin\ w) = sfin\ w$
by (*metis InitAndEmptyEqvAndEmpty Prop10 SChopAndA SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond*
inteq-reflection lift-and-com)
have 4: $\vdash (\Diamond (init\ w) \wedge (init\ w) \Pi^u f \wedge sfin\ w) = ((init\ w) \Pi^u f \wedge sfin\ w)$
using 3 **by** *auto*
have 5: $\vdash ((init\ w) \Pi g) = (\Diamond (init\ w) \wedge (init\ w) \Pi^u g)$
by (*simp add: PiEqvDiamondUPi*)
have 6: $\vdash (init\ w \wedge (init\ w) \Pi g) = (init\ w \wedge \Diamond (init\ w) \wedge (init\ w) \Pi^u g)$
using 5 **by** *auto*
have 7: $\vdash (init\ w \wedge \Diamond (init\ w)) = init\ w$
by (*meson NowImpDiamond Prop10 Prop11*)
have 8: $\vdash (init\ w \wedge \Diamond (init\ w) \wedge (init\ w) \Pi^u g) = (init\ w \wedge (init\ w) \Pi^u g)$
using 7 **by** *auto*
show ?thesis
by (*metis 2 4 6 8 SlideInitSFin inteq-reflection*)
qed

lemma *PiFiniteAbsorb*:

$\vdash (g \Pi (f \wedge finite) \wedge finite) = (g \Pi f \wedge finite)$

proof –

have 1: $\vdash (g \Pi (f \wedge finite) \wedge finite) \longrightarrow (g \Pi f \wedge finite)$
by (*metis FiniteImp PiAnd PiAndPiImpPiAnd Prop01 Prop05 Prop12 inteq-reflection lift-and-com*)
have 31: $\vdash f = ((f \wedge finite) \vee (f \wedge inf))$
unfolding *finite-d-def* **by** *fastforce*
have 32: $\vdash g \Pi f = g \Pi ((f \wedge finite) \vee (f \wedge inf))$
using 31 *PiEqvRule* **by** *blast*
have 33: $\vdash g \Pi ((f \wedge finite) \vee (f \wedge inf)) = (g \Pi (f \wedge finite) \vee g \Pi (f \wedge inf))$
by (*simp add: PiOr*)
have 34: $\vdash \neg(g \Pi (f \wedge inf) \wedge finite)$

unfolding *finite-d-def* **using** *PiAnd[of g f LIFT inf] PiInfImpInf[of g]*
by *fastforce*
have 4: $\vdash (g \amalg f \wedge \text{finite}) = (g \amalg (f \wedge \text{finite}) \wedge \text{finite})$
using 32 33 34 **by** *fastforce*
have 2: $\vdash (g \amalg f \wedge \text{finite}) \longrightarrow (g \amalg (f \wedge \text{finite}) \wedge \text{finite})$
by (*simp add: 4 int-iffD1*)
show ?thesis **using** 1 2 **by** *fastforce*
qed

lemma *PiInfAbsorb*:
 $\vdash (g \amalg (f \wedge \text{inf}) \wedge \text{inf}) = (g \amalg (f \wedge \text{inf}))$
proof –
have 1: $\vdash (g \amalg (f \wedge \text{inf}) \wedge \text{inf}) \longrightarrow (g \amalg (f \wedge \text{inf}))$
by *fastforce*
have 2: $\vdash (g \amalg (f \wedge \text{inf})) \longrightarrow (g \amalg (f \wedge \text{inf}) \wedge \text{inf})$
by (*metis PiAnd PiInfImpInf Prop10 Prop12 int-iffD1 lift-imp-trans*)
show ?thesis **using** 1 2 **by** *fastforce*
qed

lemma *PiMoreImpMore*:
 $\vdash (g \amalg \text{more}) \longrightarrow \text{more}$
unfolding *Valid-def pi-d-def itl-defs pifilt-def sfxfilt-def*
by *auto*
(metis enat-min-eq-0-iff length-nfilter-le min-def ndropns-nlength)

lemma *PiMoreAbsorb*:
 $\vdash (g \amalg (f \wedge \text{more}) \wedge \text{more}) = (g \amalg (f \wedge \text{more}))$
proof –
have 1: $\vdash (g \amalg (f \wedge \text{more}) \wedge \text{more}) \longrightarrow (g \amalg (f \wedge \text{more}))$
by *auto*
have 2: $\vdash (g \amalg (f \wedge \text{more})) \longrightarrow (g \amalg (f \wedge \text{more}) \wedge \text{more})$
by (*metis PiMoreImpMore PiAnd Prop10 Prop12 int-iffD1 inteq-reflection*)
show ?thesis **using** 1 2 **by** *fastforce*
qed

lemma *EmptyImpPiEmpty*:
 $\vdash (g \amalg f \wedge \text{empty}) \longrightarrow (g \amalg \text{empty})$
unfolding *Valid-def pi-d-def itl-defs pifilt-def sfxfilt-def*
by *auto*
(metis ile0-eq length-nfilter-le ndropns-nlength)

lemma *PiEmptyAbsorb*:
 $\vdash (g \amalg (f \wedge \text{empty}) \wedge \text{empty}) = (g \amalg f \wedge \text{empty})$
proof –
have 1: $\vdash (g \amalg (f \wedge \text{empty}) \wedge \text{empty}) \longrightarrow (g \amalg f \wedge \text{empty})$
using *PiAnd* **by** *fastforce*

have 2: $\vdash (g \Pi f \wedge \text{empty}) \longrightarrow (g \Pi (f \wedge \text{empty}) \wedge \text{empty})$
using *EmptyImpPiEmpty PiAnd* **by** *fastforce*
show *?thesis* **using** 1 2 **by** *fastforce*
qed

lemma *SlideInitSFinVar1*:

$\vdash (\text{init } w) \Pi (((F \wedge \text{more}) \wedge \text{finite}); X) =$
 $((\text{init } w) \Pi (F \wedge \text{more}) \wedge \text{sfin } w) \wedge \text{finite}; ((\text{init } w) \Pi X)$

proof –

have 1: $\vdash (\text{init } w) \Pi (((F \wedge \text{more}) \wedge \text{finite}); X) =$
 $((\text{init } w) \Pi (F \wedge \text{more}) \wedge \text{finite}); ((\text{init } w) \wedge (\text{init } w) \Pi X)$

by (*metis PiSChopDist schop-d-def*)

have 2: $\vdash ((\text{init } w) \Pi (F \wedge \text{more}) \wedge \text{finite}); ((\text{init } w) \wedge (\text{init } w) \Pi X) =$
 $((\text{init } w) \Pi (F \wedge \text{more}) \wedge \text{sfin } w) \wedge \text{finite}; ((\text{init } w) \Pi X)$

using *SlideInitSFin*[of *LIFT* (*init w*) $\Pi (F \wedge \text{more})$ *w LIFT* (*init w*) ΠX] **unfolding** *s chop-d-def*
by *blast*

show *?thesis* **using** 1 2 **by** (*metis int-eq*)

qed

lemma *UPiAbsorp*:

$\vdash (((\text{init } w) \Pi (F \wedge \text{more}) \wedge \text{sfin } w) \wedge \text{finite}); ((\text{init } w) \Pi^u X) =$
 $((\text{init } w) \Pi (F \wedge \text{more}) \wedge \text{sfin } w) \wedge \text{finite}; ((\text{init } w) \Pi X)$

proof –

have 0: $\vdash ((\text{init } w) \Pi^u X) = (\Box (\text{init } (\neg w)) \vee (\text{init } w) \Pi X)$

by (*metis Initprop(2) UPiEqvBoxOrPi inteq-reflection*)

have 1: $\vdash (((\text{init } w) \Pi (F \wedge \text{more}) \wedge \text{sfin } w) \wedge \text{finite}); ((\text{init } w) \Pi^u X) =$
 $((\text{init } w) \Pi (F \wedge \text{more}) \wedge \text{sfin } w) \wedge \text{finite}; (\Box (\text{init } (\neg w)) \vee (\text{init } w) \Pi X)$

using 0 *RightChopEqvChop* **by** *blast*

have 2: $\vdash (((\text{init } w) \Pi (F \wedge \text{more}) \wedge \text{sfin } w) \wedge \text{finite}); (\Box (\text{init } (\neg w)) \vee (\text{init } w) \Pi X) =$
 $((\text{init } w) \Pi (F \wedge \text{more})) \wedge \text{finite}; ((\text{init } w) \wedge (\Box (\text{init } (\neg w)) \vee (\text{init } w) \Pi X))$

using *SlideInitSFin*[of *LIFT* (*init w*) $\Pi (F \wedge \text{more})$ *w LIFT* ($\Box (\text{init } (\neg w)) \vee (\text{init } w) \Pi X$)]

unfolding *s chop-d-def*

by (*metis inteq-reflection*)

have 3: $\vdash ((\text{init } w) \wedge (\Box (\text{init } (\neg w)) \vee (\text{init } w) \Pi X)) =$
 $((\text{init } w) \wedge ((\text{init } w) \Pi X))$

using *BoxElim Initprop(2)* **by** *fastforce*

have 4: $\vdash (((\text{init } w) \Pi (F \wedge \text{more})) \wedge \text{finite}); ((\text{init } w) \wedge (\Box (\text{init } (\neg w)) \vee (\text{init } w) \Pi X)) =$
 $((\text{init } w) \Pi (F \wedge \text{more})) \wedge \text{finite}; ((\text{init } w) \wedge ((\text{init } w) \Pi X))$

using 3 *RightChopEqvChop* **by** *blast*

have 5: $\vdash (((\text{init } w) \Pi (F \wedge \text{more})) \wedge \text{finite}); ((\text{init } w) \wedge ((\text{init } w) \Pi X)) =$
 $((\text{init } w) \Pi (F \wedge \text{more}) \wedge \text{sfin } w) \wedge \text{finite}; ((\text{init } w) \Pi X)$

using *SlideInitSFin*[of *LIFT* ($((\text{init } w) \Pi (F \wedge \text{more}))$) *w LIFT* (*init w*) ΠX]

unfolding *s chop-d-def*

by *auto*

show *?thesis*

by (*metis* 1 2 4 5 *inteq-reflection*)

qed

lemma *PiStateFinite*:

$\vdash (init\ w) \Pi\ finite = \Diamond ((init\ w) \wedge (wnext\ (\Box (init\ (\neg w))))))$

proof –

have 1: $\vdash (init\ w) \Pi\ finite = (init\ w) \Pi\ (\#True \frown empty)$

by (*metis DiamondEmptyEqvFinite DiamondSChopdef PiEqvRule int-eq*)

have 2: $\vdash (init\ w) \Pi\ (\#True \frown empty) = ((init\ w) \Pi\ \#True) \frown ((init\ w) \wedge (init\ w) \Pi\ empty)$

by (*simp add: PiSChopDist*)

have 3: $\vdash ((init\ w) \wedge (init\ w) \Pi\ empty) = ((init\ w) \wedge wnext\ (\Box (init\ (\neg w))))$

by (*metis InitAndEmptyEqvAndEmpty StateAndEmptySChop StateAndPiEmpty inteq-reflection*)

have 4: $\vdash (init\ w) \Pi\ \#True = \Diamond (init\ w)$

by (*simp add: PiTrueEqvDiamond*)

have 5: $\vdash ((init\ w) \Pi\ \#True) \frown ((init\ w) \wedge (init\ w) \Pi\ empty) =$
 $(\Diamond (init\ w)) \frown ((init\ w) \wedge wnext\ (\Box (init\ (\neg w))))$

by (*simp add: 3 4 SChopEqvSChop*)

have 6: $\vdash (\Diamond (init\ w)) \frown ((init\ w) \wedge wnext\ (\Box (init\ (\neg w)))) =$
 $(\Diamond (init\ w) \wedge sfin\ w) \frown (wnext\ (\Box (init\ (\neg w))))$

by (*simp add: SlideInitSFin*)

have 7: $\vdash (\Diamond (init\ w) \wedge sfin\ w) = sfin\ w$

by (*metis DiamondSChopdef InitAndEmptyEqvAndEmpty Prop10 SChopAndA SFinEqvTrueSChopAndEmpty*
inteq-reflection lift-and-com)

have 8: $\vdash (\Diamond (init\ w) \wedge sfin\ w) \frown (wnext\ (\Box (init\ (\neg w)))) =$
 $(sfin\ w) \frown (wnext\ (\Box (init\ (\neg w))))$

using 7 *LeftSChopEqvSChop* **by** *blast*

have 9: $\vdash (sfin\ w) \frown (wnext\ (\Box (init\ (\neg w)))) = \#True \frown ((init\ w) \wedge (wnext\ (\Box (init\ (\neg w))))))$

by (*metis SlideInitSFin int-eq int-simps(17)*)

show *?thesis*

by (*metis 1 2 3 4 6 7 9 TrueSChopEqvDiamond inteq-reflection*)

qed

lemma *PiStateInf*:

$\vdash (init\ w) \Pi\ inf = (\Diamond (init\ w) \wedge \Box ((init\ w) \longrightarrow \bigcirc (\Diamond (init\ w))))$

proof –

have 1: $\vdash (init\ w) \Pi\ inf = (init\ w) \Pi\ (\neg\ finite)$

by (*simp add: InfEqvNotFinite PiEqvRule*)

have 2: $\vdash (init\ w) \Pi\ (\neg\ finite) = (\neg ((init\ w) \Pi^u\ finite))$

by (*simp add: NotPiEqvNotUPi*)

have 3: $\vdash ((init\ w) \Pi^u\ finite) = (\Box (init\ (\neg w)) \vee (init\ w) \Pi\ finite)$

by (*metis Initprop(2) UPiEqvBoxOrPi inteq-reflection*)

have 4: $\vdash (\Box (init\ (\neg w)) \vee (init\ w) \Pi\ finite) =$
 $(\Box (init\ (\neg w)) \vee \Diamond ((init\ w) \wedge (wnext\ (\Box (init\ (\neg w))))))$

using *PiStateFinite* **by** *fastforce*

have 5: $\vdash (\neg ((init\ w) \Pi^u\ finite)) =$
 $(\neg (\Box (init\ (\neg w)) \vee \Diamond ((init\ w) \wedge (wnext\ (\Box (init\ (\neg w)))))))$

using 3 4 **by** *fastforce*

have 6: $\vdash (\neg (\Box (init (\neg w)) \vee \Diamond ((init w) \wedge (wnext (\Box (init (\neg w))))))) =$
 $(\neg (\Box (init (\neg w))) \wedge \neg (\Diamond ((init w) \wedge (wnext (\Box (init (\neg w)))))))$
by force
have 7: $\vdash (\neg (\Box (init (\neg w)))) = \Diamond (init w)$
unfolding always-d-def
by (metis Initprop(2) int-simps(4) inteq-reflection)
have 8: $\vdash (\neg (\Diamond ((init w) \wedge (wnext (\Box (init (\neg w))))))) =$
 $\Box (\neg ((init w) \wedge (wnext (\Box (init (\neg w))))))$
by (simp add: always-d-def)
have 9: $\vdash (\neg ((init w) \wedge (wnext (\Box (init (\neg w)))))) =$
 $(\neg (init w) \vee \neg (wnext (\Box (init (\neg w)))))$
by force
have 10: $\vdash (\neg (init w)) = (init (\neg w))$
by (simp add: Initprop(2))
have 11: $\vdash (\neg (wnext (\Box (init (\neg w)))) = \bigcirc (\neg (\Box (init (\neg w))))$
by (simp add: wnext-d-def)
have 12: $\vdash \bigcirc (\neg (\Box (init (\neg w)))) = \bigcirc (\Diamond (init w))$
by (simp add: 7 NextEqvNext)
have 13: $\vdash (\neg (init w) \vee \neg (wnext (\Box (init (\neg w)))) = ((init w) \longrightarrow \bigcirc (\Diamond (init w)))$
using 10 11 12 by fastforce
have 14: $\vdash \Box (\neg ((init w) \wedge (wnext (\Box (init (\neg w)))))) =$
 $\Box ((init w) \longrightarrow \bigcirc (\Diamond (init w)))$
by (metis 13 8 9 inteq-reflection)
have 15: $\vdash (\neg (\Box (init (\neg w))) \wedge \neg (\Diamond ((init w) \wedge (wnext (\Box (init (\neg w))))))) =$
 $(\Diamond (init w) \wedge \Box ((init w) \longrightarrow \bigcirc (\Diamond (init w))))$
by (metis 13 6 7 8 9 inteq-reflection)
show ?thesis
by (metis 1 15 2 5 6 int-eq)
qed

lemma UPiPiState:

$\vdash (init w) \Pi^u ((init w) \Pi f) = (init w) \Pi^u f$

proof –

have 1: $\vdash (init w) \Pi^u ((init w) \Pi f) = (\Box (init (\neg w)) \vee (init w) \Pi ((init w) \Pi f))$

by (metis Initprop(2) UPiEqvBoxOrPi int-eq)

have 2: $\vdash (init w) \Pi ((init w) \Pi f) = ((init w) \Pi f)$

by (metis PiAssoc Prop10 StateEqvBi StateImpBi inteq-reflection)

have 3: $\vdash (\Box (init (\neg w)) \vee (init w) \Pi ((init w) \Pi f)) =$
 $(\Box (init (\neg w)) \vee ((init w) \Pi f))$

using 2 by auto

have 4: $\vdash (\Box (init (\neg w)) \vee ((init w) \Pi f)) = (init w) \Pi^u f$

by (metis Initprop(2) UPiEqvBoxOrPi int-eq)

show ?thesis

by (metis 1 3 4 int-eq)

qed

lemma PiUPiState:

$\vdash (init\ w) \sqcap ((init\ w) \sqcap^u f) = (init\ w) \sqcap f$
proof –
have 1: $\vdash (init\ w) \sqcap ((init\ w) \sqcap^u f) = (\Diamond (init\ w) \wedge (init\ w) \sqcap^u ((init\ w) \sqcap^u f))$
by (*simp add: PiEqvDiamondUPi*)
have 2: $\vdash (init\ w) \sqcap^u ((init\ w) \sqcap^u f) = ((init\ w) \sqcap^u f)$
by (*metis (no-types, lifting) NotUPiEqvNotPi UPiPiState inteq-reflection upi-d-def*)
have 3: $\vdash (\Diamond (init\ w) \wedge (init\ w) \sqcap^u ((init\ w) \sqcap^u f)) =$
 $(\Diamond (init\ w) \wedge ((init\ w) \sqcap^u f))$
using 2 **by** *auto*
have 4: $\vdash (\Diamond (init\ w) \wedge ((init\ w) \sqcap^u f)) = (init\ w) \sqcap f$
by (*meson PiEqvDiamondUPi Prop11*)
show ?thesis
by (*metis 1 3 4 int-eq*)
qed

lemma *PiStateFiniteAndFinite*:

$\vdash ((init\ w) \sqcap finite \wedge finite) = (\Diamond (init\ w) \wedge finite)$
proof –
have 1: $\vdash ((init\ w) \sqcap finite \wedge finite) \longrightarrow (\Diamond (init\ w) \wedge finite)$
using *PiImpDiamond* **by** *fastforce*
have 2: $\vdash (\Diamond (init\ w) \wedge finite) \longrightarrow ((init\ w) \sqcap finite \wedge finite)$
by (*metis PiFiniteAbsorb PiTrueEqvDiamond TrueW int-simps(13) int-simps(17) inteq-reflection*)
show ?thesis
using 1 2 *int-iffI* **by** *blast*
qed

lemma *PiStateState*:

$\vdash (init\ w) \sqcap (init\ w) = (init\ w) \sqcap \#True$
by (*metis BoxElim PiImpDiamond PiMPBC PiTrueEqvDiamond Prop11 StatePiBoxState int-simps(17) inteq-reflection*)

lemma *StateAndPiState*:

$\vdash ((init\ w) \wedge (init\ w) \sqcap (empty \wedge w)) = ((init\ w) \wedge wnext (\Box (init (\neg w))))$
proof –
have 2: $\vdash ((init\ w) \wedge init\ w \sqcap empty) = (init\ w \wedge wnext (\Box (init (\neg w))))$
by (*metis InitAndEmptyEqvAndEmpty StateAndEmptySChop StateAndPiEmpty inteq-reflection*)
have 5: $\vdash (init\ w) \sqcap \#True = \Diamond (init\ w)$
by (*simp add: PiTrueEqvDiamond*)
show ?thesis
by (*metis 2 5 InitAndEmptyEqvAndEmpty PiAnd PiImpDiamond PiStateState Prop10 inteq-reflection lift-and-com*)
qed

lemma *PiStateFiniteSfin*:

$\vdash ((init\ w) \sqcap finite \wedge sfin\ w) = sfin\ w$
proof –
have 1: $\vdash ((init\ w) \sqcap finite \wedge sfin\ w) =$

$((\text{init } w) \sqcap \text{finite} \wedge \text{finite}) \wedge \text{sfin } w$
by (*metis AndSFinEqvSchopAndEmpty SchopAndEmptyEqvSchopAndEmpty SFinEqvTrueSchopAndEmpty*
inteq-reflection lift-and-com)
have 2: $\vdash ((\text{init } w) \sqcap \text{finite} \wedge \text{finite}) \wedge \text{sfin } w = ((\Diamond (\text{init } w) \wedge \text{finite}) \wedge \text{sfin } w)$
using *PiStateFiniteAndFinite* **by** *fastforce*
have 3: $\vdash ((\Diamond (\text{init } w) \wedge \text{finite}) \wedge \text{sfin } w) = ((\Diamond (\text{init } w)) \wedge \text{sfin } w)$
by (*metis AndSFinEqvSchopAndEmpty SchopAndEmptyEqvSchopAndEmpty SFinEqvTrueSchopAndEmpty*
inteq-reflection lift-and-com)
have 4: $\vdash ((\Diamond (\text{init } w)) \wedge \text{sfin } w) = \text{sfin } w$
by (*metis 3 InitAndEmptyEqvAndEmpty Prop11 Prop12 SchopAndA SFinEqvTrueSchopAndEmpty*
TrueSchopEqvDiamond inteq-reflection)
show *?thesis*
by (*metis 1 2 3 4 inteq-reflection*)
qed

lemma *PiStateSFin*:

$\vdash (\text{init } w) \sqcap (\text{sfin } w) = (\text{sfin } w);(\text{wnext } (\Box (\text{init } (\neg w))))$
proof –
have 1: $\vdash (\text{sfin } w) = \text{finite};(\text{empty} \wedge w)$
by (*metis ChopAndCommute SFinEqvTrueSchopAndEmpty TrueSchopEqvDiamond inteq-reflection sometimes-d-def*)
have 2: $\vdash (\text{init } w) \sqcap (\text{finite};(\text{empty} \wedge w)) =$
 $((\text{init } w) \sqcap \text{finite} \wedge \text{finite});((\text{init } w) \wedge (\text{init } w) \sqcap (\text{empty} \wedge w))$
using *PiSchopDist*[of *w LIFT finite LIFT (empty} \wedge w)*] **unfolding** *schop-d-def*
by *auto*
have 3: $\vdash ((\text{init } w) \sqcap \text{finite} \wedge \text{finite});((\text{init } w) \wedge (\text{init } w) \sqcap (\text{empty} \wedge w)) =$
 $((\text{init } w) \sqcap \text{finite} \wedge \text{finite});((\text{init } w) \wedge \text{wnext } (\Box (\text{init } (\neg w))))$
using *RightChopEqvChop StateAndPiState* **by** *blast*
have 4: $\vdash (\text{sfin } w \wedge \text{finite}) = \text{sfin } w$
by (*metis 1 ChopAndA DiamondEmptyEqvFinite Prop10 inteq-reflection sometimes-d-def*)
have 5: $\vdash ((\text{init } w) \sqcap \text{finite} \wedge \text{finite});((\text{init } w) \wedge \text{wnext } (\Box (\text{init } (\neg w)))) =$
 $((\text{init } w) \sqcap \text{finite} \wedge \text{sfin } w);(\text{wnext } (\Box (\text{init } (\neg w))))$
using *SlideInitSFin*[of *LIFT (init } w) \sqcap \text{finite } w LIFT wnext } (\Box (\text{init } (\neg w)))*] 4
unfolding *schop-d-def*
by (*metis Prop10 Prop12 int-eq int-iffD2 lift-and-com*)
have 6: $\vdash ((\text{init } w) \sqcap \text{finite} \wedge \text{sfin } w);(\text{wnext } (\Box (\text{init } (\neg w)))) = (\text{sfin } w);(\text{wnext } (\Box (\text{init } (\neg w))))$
using *LeftChopEqvChop PiStateFiniteSfin* **by** *blast*
show *?thesis*
by (*metis 1 2 3 5 6 int-eq*)
qed

2.9.5 Schopstar and Pi

lemma *PiChopstarhelp2a*:

$\vdash (w \wedge \text{empty}) \frown (\text{fpower } (f \frown (w \wedge \text{empty})) \wedge \text{more}) k) =$
 $(\text{fpower } (((w \wedge \text{empty}) \frown f) \wedge \text{more}) k) \frown (w \wedge \text{empty})$
proof (*induction k*)

case 0
then show *?case*
proof –
have 1: $\vdash (w \wedge \text{empty}) \multimap \text{empty} = \text{empty} \multimap (w \wedge \text{empty})$
by (*metis EmptySChop InitAndEmptyEqvAndEmpty StateAndEmptySChop int-eq*)
show *?thesis*
by (*metis 1 fpower-d-def wpow-0*)
qed
next
case (*Suc k*)
then show *?case*
proof –
have 1: $\vdash (w \wedge \text{empty}) \multimap \text{fpower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) \text{ (Suc k)} =$
 $(w \wedge \text{empty}) \multimap ((f \multimap (w \wedge \text{empty}) \wedge \text{more}) \multimap \text{fpower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k)$
unfolding *fpower-d-def*
by (*simp add: schop-d-def*)
have 11: $\vdash (f \wedge \text{more}) \multimap (w \wedge \text{empty}) = (\text{sfin } w \wedge (f \wedge \text{more}))$
by (*metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty int-eq*)
have 12: $\vdash (f \multimap (w \wedge \text{empty})) = (\text{sfin } w \wedge f)$
by (*metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection*)
have 13: $\vdash (f \multimap (w \wedge \text{empty}) \wedge \text{more}) = ((\text{sfin } w \wedge f) \wedge \text{more})$
using 12 **by** *auto*
have 14: $\vdash ((\text{sfin } w \wedge f) \wedge \text{more}) = (\text{sfin } w \wedge (f \wedge \text{more}))$
by *fastforce*
have 2: $\vdash (f \multimap (w \wedge \text{empty}) \wedge \text{more}) = (f \wedge \text{more}) \multimap (w \wedge \text{empty})$
using 11 13 **by** *fastforce*
have 20: $\vdash ((f \multimap (w \wedge \text{empty}) \wedge \text{more}) \multimap \text{fpower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k) =$
 $((f \wedge \text{more}) \multimap (w \wedge \text{empty})) \multimap \text{fpower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k$
by (*simp add: 2 LeftSChopEqvSChop*)
have 21: $\vdash ((f \wedge \text{more}) \multimap (w \wedge \text{empty})) \multimap \text{fpower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k =$
 $(f \wedge \text{more}) \multimap ((w \wedge \text{empty}) \multimap \text{fpower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k)$
by (*meson Prop11 SChopAssoc*)
have 22: $\vdash (f \wedge \text{more}) \multimap ((w \wedge \text{empty}) \multimap \text{fpower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k) =$
 $((f \wedge \text{more}) \multimap (\text{fpower } ((w \wedge \text{empty}) \multimap f \wedge \text{more}) k \multimap (w \wedge \text{empty})))$
by (*simp add: RightSChopEqvSChop Suc.IH*)
have 23: $\vdash (w \wedge \text{empty}) \multimap ((f \multimap (w \wedge \text{empty}) \wedge \text{more}) \multimap \text{fpower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k) =$
 $(w \wedge \text{empty}) \multimap (((f \wedge \text{more}) \multimap (w \wedge \text{empty})) \multimap \text{fpower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k)$
using 20 *RightSChopEqvSChop* **by** *blast*
have 24: $\vdash (w \wedge \text{empty}) \multimap (((f \wedge \text{more}) \multimap (w \wedge \text{empty})) \multimap \text{fpower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k) =$
 $(w \wedge \text{empty}) \multimap ((f \wedge \text{more}) \multimap ((w \wedge \text{empty}) \multimap \text{fpower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k))$
using 21 *RightSChopEqvSChop* **by** *blast*
have 25: $\vdash (w \wedge \text{empty}) \multimap ((f \wedge \text{more}) \multimap ((w \wedge \text{empty}) \multimap \text{fpower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k)) =$
 $(w \wedge \text{empty}) \multimap ((f \wedge \text{more}) \multimap (\text{fpower } ((w \wedge \text{empty}) \multimap f \wedge \text{more}) k \multimap (w \wedge \text{empty})))$
using 23 22 *RightSChopEqvSChop* **by** *blast*
have 3: $\vdash (w \wedge \text{empty}) \multimap ((f \multimap (w \wedge \text{empty}) \wedge \text{more}) \multimap \text{fpower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k) =$
 $(w \wedge \text{empty}) \multimap ((f \wedge \text{more}) \multimap (\text{fpower } ((w \wedge \text{empty}) \multimap f \wedge \text{more}) k \multimap (w \wedge \text{empty})))$
using 23 24 25 **by** *fastforce*
have 4: $\vdash (w \wedge \text{empty}) \multimap ((f \wedge \text{more}) \multimap (\text{fpower } ((w \wedge \text{empty}) \multimap f \wedge \text{more}) k \multimap (w \wedge \text{empty}))) =$
 $((w \wedge \text{empty}) \multimap (f \wedge \text{more})) \multimap ((\text{fpower } ((w \wedge \text{empty}) \multimap f \wedge \text{more}) k \multimap (w \wedge \text{empty})))$
using *SChopAssoc* **by** *blast*

have 6: $\vdash ((w \wedge \text{empty}) \neg (f \wedge \text{more})) \neg ((\text{fpower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k \neg (w \wedge \text{empty}))) =$
 $((w \wedge \text{empty}) \neg f \wedge \text{more}) \neg ((\text{fpower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k \neg (w \wedge \text{empty})))$
by (*meson EmptyAndSChopAndMoreEqvAndSChop LeftSChopEqvSChop*)
have 7: $\vdash ((w \wedge \text{empty}) \neg f \wedge \text{more}) \neg ((\text{fpower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k \neg (w \wedge \text{empty}))) =$
 $((\text{fpower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) (\text{Suc } k) \neg (w \wedge \text{empty})))$
by (*metis SChopAssoc fpower-d-def schop-d-def wpow-Suc*)
show *?thesis* **using** 1 3 4 6 7
by (*metis inteq-reflection*)
qed
qed

lemma *PiChopstarhelp2:*

$\vdash (w \wedge \text{empty}) \neg (\text{s chopstar}(f \neg (w \wedge \text{empty}))) = (\text{s chopstar}((w \wedge \text{empty}) \neg f)) \neg (w \wedge \text{empty})$
proof –
have 1: $\vdash (w \wedge \text{empty}) \neg (\text{s chopstar}(f \neg (w \wedge \text{empty}))) =$
 $(w \wedge \text{empty}) \neg (\exists k. \text{fpower } ((f \neg (w \wedge \text{empty})) \wedge \text{more}) k)$
by (*simp add: schopstar-d-def fpowerstar-d-def*)
have 2: $\vdash (w \wedge \text{empty}) \neg (\exists k. \text{fpower } ((f \neg (w \wedge \text{empty})) \wedge \text{more}) k) =$
 $(\exists k. (w \wedge \text{empty}) \neg (\text{fpower } ((f \neg (w \wedge \text{empty})) \wedge \text{more}) k))$
using *SChopExist* **by** *fastforce*
have 3: $\vdash (\exists k. (w \wedge \text{empty}) \neg (\text{fpower } ((f \neg (w \wedge \text{empty})) \wedge \text{more}) k)) =$
 $(\exists k. (\text{fpower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k) \neg (w \wedge \text{empty}))$
by (*simp add: ExEqvRule PiChopstarhelp2a*)
have 4: $\vdash (\exists k. (\text{fpower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k) \neg (w \wedge \text{empty})) =$
 $(\exists k. (\text{fpower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k)) \neg (w \wedge \text{empty})$
using *ExistSChop* **by** *fastforce*
have 5: $\vdash (\exists k. (\text{fpower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k)) \neg (w \wedge \text{empty}) =$
 $(\text{s chopstar}((w \wedge \text{empty}) \neg f)) \neg (w \wedge \text{empty})$
by (*simp add: schopstar-d-def fpowerstar-d-def*)
show *?thesis*
by (*metis 1 2 3 4 5 int-eq*)
qed

lemma *PiFPowerSuca:*

$\vdash (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k)) = ((\text{init } w) \Pi f) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{fpower } f k))$
unfolding *fpower-d-def* **by** (*metis PiSChopDist schop-d-def wpow-Suc*)

lemma *PiFPowerSuch:*

$\vdash (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k)) = ((\text{init } w) \Pi f \wedge \text{sfin } w) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{fpower } f k))$
proof –
have 0: $\vdash (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k)) = ((\text{init } w) \Pi f) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{fpower } f k))$
by (*meson PiFPowerSuca*)
have 1: $\vdash ((\text{init } w) \wedge (\text{init } w) \Pi (\text{fpower } f k)) = (w \wedge \text{empty}) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{fpower } f k))$
by (*metis InitAndEmptyEqvAndEmpty Prop10 Prop12 StateAndEmptySChop int-iffD2 inteq-reflection lift-and-com*)
have 2: $\vdash ((\text{init } w) \Pi f) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{fpower } f k)) =$
 $((\text{init } w) \Pi f) \neg ((w \wedge \text{empty}) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{fpower } f k)))$
using 1 *RightSChopEqvSChop* **by** *blast*
have 3: $\vdash ((\text{init } w) \Pi f) \neg ((w \wedge \text{empty}) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{fpower } f k))) =$

$((init\ w) \sqcap f) \frown (w \wedge empty) \frown ((init\ w) \wedge (init\ w) \sqcap (fpower\ f\ k))$
by (*simp add: SChopAssoc*)
have 4: $\vdash ((init\ w) \sqcap f) \frown (w \wedge empty) = ((init\ w) \sqcap f \wedge sfin\ w)$
by (*metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection lift-and-com*)
have 5: $\vdash ((init\ w) \sqcap f) \frown (w \wedge empty) \frown ((init\ w) \wedge (init\ w) \sqcap (fpower\ f\ k)) =$
 $((init\ w) \sqcap f \wedge sfin\ w) \frown ((init\ w) \wedge (init\ w) \sqcap (fpower\ f\ k))$
by (*simp add: 4 LeftSChopEqvSChop*)
show ?thesis
using 0 2 3 5 **by** *fastforce*
qed

lemma *PiFPowerSucc*:

$\vdash (init\ w) \sqcap (fpower\ f\ (Suc\ k)) =$
 $(fpower\ ((init\ w) \sqcap f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w) \wedge (init\ w) \sqcap empty)$
proof (*induction k*)
case 0
then show ?case
using *PiFPowerSucc[of w f 0]*
unfolding *schop-d-def fpower-d-def wpow-0 wpow-Suc*
proof –
assume a1: $\vdash init\ w \sqcap ((f \wedge finite); empty) =$
 $((init\ w \sqcap f \wedge sfin\ w) \wedge finite); (init\ w \wedge init\ w \sqcap empty)$
have $\vdash (((init\ w \sqcap f \wedge sfin\ w) \wedge finite); empty \wedge finite) = ((init\ w \sqcap f \wedge sfin\ w) \wedge finite)$
by (*metis AndChopB ChopEmpty Prop10 inteq-reflection*)
then show $\vdash init\ w \sqcap ((f \wedge finite); empty) =$
 $((init\ w \sqcap f \wedge sfin\ w) \wedge finite); empty \wedge finite); (init\ w \wedge init\ w \sqcap empty)$
by (*metis a1 int-eq*)
qed
next
case (*Suc k*)
then show ?case
proof –
have 3: $\vdash (init\ w \sqcap f) \frown$
 $(init\ w \wedge fpower\ (init\ w \sqcap f \wedge sfin\ w)\ (Suc\ k)) \frown (init\ w \wedge init\ w \sqcap empty) =$
 $((init\ w \sqcap f \wedge sfin\ w) \frown fpower\ (init\ w \sqcap f \wedge sfin\ w)\ (Suc\ k)) \frown$
 $(init\ w \wedge init\ w \sqcap empty)$
by (*metis (no-types, lifting) AndSFinSChopEqvStateAndSChop InitAndEmptyEqvAndEmpty SChopAssoc*
SFinEqvTrueSChopAndEmpty inteq-reflection)
have 4: $\vdash (init\ w \sqcap f) \frown (init\ w \wedge init\ w \sqcap (f \frown fpower\ f\ k)) =$
 $((init\ w \sqcap f \wedge sfin\ w) \frown$
 $((init\ w \sqcap f \wedge sfin\ w) \frown fpower\ (init\ w \sqcap f \wedge sfin\ w)\ k)) \frown$
 $(init\ w \wedge init\ w \sqcap empty)$
using 3 *Suc.IH* **unfolding** *fpower-d-def wpow-Suc schop-d-def* **by** (*metis int-eq*)
show ?thesis
by (*metis 4 PiFPowerSucca fpower-d-def inteq-reflection schop-d-def wpow-Suc*)
qed
qed

lemma *PiFPowerSuccd*:

$\vdash (init\ w) \sqcap (fpower\ f\ (Suc\ k)) = (fpower\ ((init\ w) \sqcap f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w) \sqcap empty)$

proof (*induction k*)
case 0
then show ?*case unfolding fpower-d-def wpow-0 wpow-Suc*
by (*metis (no-types, opaque-lifting) InitAndEmptyEqvAndEmpty PiSCHopDist SCHopAndEmptyEqvSCHopAndEmpty*)
SCHopAssoc SFinEqvTrueSCHopAndEmpty StateAndEmptySCHop inteq-reflection lift-and-com schop-d-def
next
case (*Suc k*)
then show ?*case*
proof –
have 1: $\vdash \text{init } w \Pi \text{ fpower } f (\text{Suc } (\text{Suc } k)) = (\text{init } w) \Pi (f \frown (\text{fpower } f (\text{Suc } k)))$
by (*metis PiFPowerSuca PiSCHopDist inteq-reflection*)
have 2: $\vdash (\text{init } w) \Pi (f \frown (\text{fpower } f (\text{Suc } k))) = ((\text{init } w) \Pi f) \frown ((\text{init } w) \wedge (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k)))$
by (*simp add: PiSCHopDist*)
have 3: $\vdash ((\text{init } w) \Pi f) \frown ((\text{init } w) \wedge (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k))) =$
 $((\text{init } w) \Pi f) \frown ((\text{init } w) \wedge (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } k))) \frown ((\text{init } w) \wedge (\text{init } w) \Pi \text{empty}))$
by (*metis 2 PiFPowerSucc inteq-reflection*)
have 4: $\vdash ((\text{init } w) \Pi f) \frown ((\text{init } w) \wedge (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } k))) \frown ((\text{init } w) \wedge (\text{init } w) \Pi \text{empty})) =$
 $((\text{init } w) \Pi f \wedge \text{sfin } w) \frown ((\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } k))) \frown ((\text{init } w) \wedge (\text{init } w) \Pi \text{empty}))$
by (*metis AndSFinSCHopEqvStateAndSCHop InitAndEmptyEqvAndEmpty SFinEqvTrueSCHopAndEmpty inteq-reflection*)
have 5: $\vdash ((\text{init } w) \Pi f \wedge \text{sfin } w) \frown ((\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } k))) \frown ((\text{init } w) \wedge (\text{init } w) \Pi \text{empty})) =$
 $((\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } (\text{Suc } k)))) \frown ((\text{init } w) \wedge (\text{init } w) \Pi \text{empty}))$
by (*metis 1 2 4 PiFPowerSucc inteq-reflection*)
have 6: $\vdash ((\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } (\text{Suc } k)))) \frown ((\text{init } w) \wedge (\text{init } w) \Pi \text{empty})) =$
 $((\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } (\text{Suc } k)))) \frown ((\text{init } w) \Pi \text{empty}))$
by (*metis (no-types, lifting) PiFPowerSucc SCHopAssoc Suc fpower-d-def inteq-reflection schop-d-def wpow-Suc*)
show ?*thesis*
by (*metis 1 2 3 4 5 6 inteq-reflection*)
qed
qed

lemma *SFin-SCHop*:

$\vdash (f \wedge \text{sfin } w) \frown g = (f \wedge \text{fin } w) \frown g$

proof –

have 1: $\vdash (f \wedge \text{fin } w) \frown g = (f \wedge (\text{fin } w \wedge \text{finite})) \frown g$

by (*metis (no-types, lifting) LeftChopEqvChop Prop11 Prop12 lift-and-com schop-d-def*)

have 2: $\vdash (\text{fin } w \wedge \text{finite}) = (\text{sfin } w \wedge \text{finite})$

by (*metis (no-types, lifting) EmptyImpFinite FinAndEmpty FinEqvTrueChopAndEmpty Finprop(5) Prop10 SCHopAndB SCHopImpFinite SFinAndEmpty SFinEqvTrueSCHopAndEmpty inteq-reflection lift-imp-trans sfin-d-def*)

have 3: $\vdash (f \wedge \text{sfin } w) \frown g = (f \wedge (\text{sfin } w \wedge \text{finite})) \frown g$

by (*metis AndSCHopCommute DiamondEmptyEqvFinite Prop10 SCHopAndB SFinEqvTrueSCHopAndEmpty*)

$\text{TrueSCHopEqvDiamond int-eq}$
show *?thesis*
using 1 2 3 **by** (*metis inteq-reflection*)
qed

lemma *PiChopstar*:
 $\vdash (\text{init } w) \Pi (\text{schopstar } f) =$
 $(\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge (\text{schopstar}(((\text{init } w) \Pi f) \wedge \text{sfin } w)) \neg \text{wnext}(\Box (\text{init } (\neg w))))$

proof –
have 1: $\vdash (\text{init } w) \Pi (\text{schopstar } f) = (\text{init } w) \Pi (\exists k. \text{fpower } f k)$
by (*metis FPowerstardef PiEqvRule SCHopstar-FPowerstar inteq-reflection*)
have 2: $\vdash (\text{init } w) \Pi (\exists k. \text{fpower } f k) = (\exists k. (\text{init } w) \Pi (\text{fpower } f k))$
by (*simp add: Valid-def pi-d-def*)
have 3: $\vdash (\exists k. (\text{init } w) \Pi (\text{fpower } f k)) =$
 $((\text{init } w) \Pi \text{empty} \vee (\exists k. (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k))))$
using *PiFPowerExpand* **by** *auto*
have 4: $\vdash (\exists k. (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k))) =$
 $(\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } k)) \neg ((\text{init } w) \Pi \text{empty}))$
by (*meson ExEqvRule PiFPowerSuccd*)
have 5: $\vdash (\text{init } w) \Pi \text{empty} = \text{empty} \neg ((\text{init } w) \Pi \text{empty})$
by (*simp add: EmptySCHop int-iffD1 int-iffD2 int-iffI*)
have 6: $\vdash (\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } k)) \neg ((\text{init } w) \Pi \text{empty})) =$
 $(\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } k)) \neg ((\text{init } w) \Pi \text{empty}))$
by (*simp add: ExistSCHop*)
have 7: $\vdash ((\text{init } w) \Pi \text{empty} \vee (\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } k)) \neg ((\text{init } w) \Pi \text{empty}))) =$
 $(\text{empty} \neg ((\text{init } w) \Pi \text{empty}) \vee$
 $(\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } k)) \neg ((\text{init } w) \Pi \text{empty}))$
using 5 6 **by** *fastforce*
have 8: $\vdash (\text{empty} \neg ((\text{init } w) \Pi \text{empty}) \vee$
 $(\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } k)) \neg ((\text{init } w) \Pi \text{empty})) =$
 $(\text{empty} \vee (\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } k)) \neg ((\text{init } w) \Pi \text{empty}))$
by (*meson OrSCHopEqv Prop11*)
have 9: $\vdash \text{empty} = (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) 0)$
unfolding *fpower-d-def* **by** *simp*
have 10: $\vdash (\text{empty} \vee (\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } k)))) =$
 $((\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) 0) \vee (\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } k))))$
unfolding *fpower-d-def* **by** *simp*
have 11: $\vdash ((\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) 0) \vee (\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } k)))) =$
 $(\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) k))$
using *exists-expand[of w f]* **unfolding** *fpower-d-def*
by (*metis (mono-tags) Valid-def inteq-reflection wpow-0 wpowersem1*)
have 12: $\vdash ((\text{init } w) \Pi \text{empty} \vee$
 $(\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (\text{Suc } k)) \neg ((\text{init } w) \Pi \text{empty}))) =$
 $(\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (k)) \neg ((\text{init } w) \Pi \text{empty}))$
by (*metis 11 7 8 9 inteq-reflection*)
have 13: $\vdash (\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) (k)) \neg ((\text{init } w) \Pi \text{empty})) =$
 $(\text{schopstar } ((\text{init } w) \Pi f \wedge \text{sfin } w)) \neg ((\text{init } w) \Pi \text{empty})$
by (*metis FPowerstardef LeftSCHopEqvSCHop SCHopstar-FPowerstar inteq-reflection*)
have 14: $\vdash (\text{schopstar } ((\text{init } w) \Pi f \wedge \text{sfin } w)) \neg ((\text{init } w) \Pi \text{empty}) =$
 $(((\text{init } w) \Pi \text{empty}) \vee$

$((init\ w) \Pi f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi empty)$
by (*metis 5 OrSChopEqvRule SChopstar-unfoldl-eq inteq-reflection*)
have 15: $\vdash ((init\ w) \Pi empty) = (init\ (\neg w)) \mathcal{U} ((init\ w) \wedge wnext\ (\Box (init\ (\neg w))))$
by (*simp add: PiEmpty*)
have 16: $\vdash ((init\ w) \Pi f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi empty) =$
 $((init\ w) \Pi f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown wnext\ (\Box (init\ (\neg w)))$
proof –
have 161: $\vdash ((init\ w) \Pi f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi empty) =$
 $((init\ w) \Pi f \wedge sfin\ w) \frown ((schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi empty))$
by (*meson Prop11 SChopAssoc*)
have 162: $\vdash ((init\ w) \Pi f \wedge sfin\ w) = ((init\ w) \Pi f) \frown (w \wedge empty)$
by (*metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection lift-and-com*)

have 163: $\vdash (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) = (schopstar\ (((init\ w) \Pi f) \frown (w \wedge empty)))$
by (*metis 162 SChopstardef inteq-reflection*)
have 164: $\vdash ((init\ w) \Pi f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) =$
 $((init\ w) \Pi f) \frown ((w \wedge empty) \frown (schopstar\ ((init\ w) \Pi f) \frown (w \wedge empty)))$
using 162 SChopAssoc int-eq by metis
have 165: $\vdash ((init\ w) \Pi f) \frown ((w \wedge empty) \frown (schopstar\ ((init\ w) \Pi f) \frown (w \wedge empty))) =$
 $((init\ w) \Pi f) \frown ((schopstar\ (w \wedge empty) \frown ((init\ w) \Pi f)) \frown (w \wedge empty))$
by (*simp add: PiChopstarhelp2 RightSChopEqvSChop*)
have 166: $\vdash ((init\ w) \Pi f) \frown ((schopstar\ (w \wedge empty) \frown ((init\ w) \Pi f)) \frown (w \wedge empty)) \frown ((init\ w) \Pi empty) =$
 $((init\ w) \Pi f) \frown (schopstar\ (w \wedge empty) \frown ((init\ w) \Pi f)) \frown ((init\ w) \wedge ((init\ w) \Pi empty))$
by (*metis (no-types, lifting) InitAndEmptyEqvAndEmpty SChopAssoc StateAndEmptySChop int-eq*)
have 167: $\vdash ((init\ w) \Pi f) \frown (schopstar\ (w \wedge empty) \frown ((init\ w) \Pi f)) \frown ((init\ w) \wedge ((init\ w) \Pi empty)) =$
 $((init\ w) \Pi f) \frown (schopstar\ (w \wedge empty) \frown ((init\ w) \Pi f)) \frown ((w \wedge empty) \frown wnext\ (\Box (init\ (\neg w))))$
by (*simp add: RightSChopEqvSChop StateAndPiEmpty*)
have 168: $\vdash ((init\ w) \Pi f) \frown (schopstar\ (w \wedge empty) \frown ((init\ w) \Pi f)) \frown ((w \wedge empty) \frown wnext\ (\Box (init\ (\neg w)))) =$
 $((init\ w) \Pi f) \frown (schopstar\ (w \wedge empty) \frown ((init\ w) \Pi f)) \frown (w \wedge empty) \frown wnext\ (\Box (init\ (\neg w)))$
by (*simp add: SChopAssoc*)
have 169: $\vdash ((init\ w) \Pi f) \frown (schopstar\ (w \wedge empty) \frown ((init\ w) \Pi f)) \frown (w \wedge empty) =$
 $((init\ w) \Pi f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi f \wedge sfin\ w))$
using 164 165 SChopAssoc by fastforce
show ?thesis by (*metis 164 165 166 167 168 169 int-eq*)
qed
have 17: $\vdash ((init\ w) \Pi f \wedge sfin\ w) = ((init\ w) \Pi f) \frown ((init\ w) \wedge empty)$
by (*metis InitAndEmptyEqvAndEmpty SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection lift-and-com*)
have 18: $\vdash ((init\ w) \Pi f) = wprev(\Box (init\ (\neg w))) \frown ((init\ w) \wedge (init\ w) \Pi f)$
by (*simp add: WPrevPi*)
have 19: $\vdash wprev(\Box (init\ (\neg w))) \frown ((init\ w) \wedge (init\ w) \Pi f) =$
 $wprev(\Box (init\ (\neg w))) \frown (((init\ w) \wedge empty) \frown ((init\ w) \Pi f))$
by (*meson RightSChopImpSChop StateAndEmptySChop int-iffD1 int-iffD2 int-iffI*)

have 20: $\vdash wprev(\Box (init (\neg w))) \frown (((init w) \wedge empty) \frown ((init w) \amalg f)) =$
 $(init (\neg w)) \mathcal{U} ((init w) \wedge ((init w) \amalg f))$
using 18 19 StatePiUntil by fastforce
have 21: $\vdash (((init w) \amalg f \wedge sfin w) \frown (schopstar ((init w) \amalg f \wedge sfin w))) \frown wnext (\Box (init (\neg w))) =$
 $(init (\neg w)) \mathcal{U} ($
 $(init w) \wedge$
 $((((init w) \amalg f) \wedge sfin w) \frown (schopstar ((init w) \amalg f \wedge sfin w))) \frown wnext (\Box (init (\neg w)))$
 $)$
proof –
have $\vdash (init w \amalg f) \frown (init w \wedge empty) = (init w \amalg f \wedge sfin w)$
by *(metis 17 inteq-reflection)*
then show *?thesis* **using** *StatePiUntil[of w f]*
 $UntilChopDist[of w LIFT((init w \wedge empty)) LIFT((schopstar ((init w) \amalg f \wedge sfin w))) \frown wnext (\Box (init$
 $(\neg w)))]$
by *(metis (no-types, lifting) AndSFinSChopEqvStateAndSChop SChopAssoc StateUntilEqvWPrevChop*
int-eq)
qed
have 22: $\vdash ((init (\neg w)) \mathcal{U} ((init w) \wedge wnext (\Box (init (\neg w))))) \vee$
 $(init (\neg w)) \mathcal{U} ($
 $(init w) \wedge$
 $((((init w) \amalg f) \wedge sfin w) \frown (schopstar ((init w) \amalg f \wedge sfin w))) \frown wnext (\Box (init (\neg w)))$
 $)) =$
 $(init (\neg w)) \mathcal{U} ($
 $((init w) \wedge wnext (\Box (init (\neg w)))) \vee$
 $((init w) \wedge (((init w) \amalg f) \wedge sfin w) \frown (schopstar ((init w) \amalg f \wedge sfin w))) \frown wnext (\Box (init (\neg w)))$
 $)$
using *UntilOrDist by fastforce*
have 23: $\vdash ($
 $((init w) \wedge wnext (\Box (init (\neg w))))) \vee$
 $((init w) \wedge (((init w) \amalg f) \wedge sfin w) \frown (schopstar ((init w) \amalg f \wedge sfin w))) \frown wnext (\Box (init (\neg w)))$
 $) =$
 $((init w) \wedge (schopstar (((init w) \amalg f) \wedge sfin w)) \frown wnext (\Box (init (\neg w))))$
by *(metis (mono-tags, lifting) EmptyOrSChopEqv Prop11 SChopOrEqvRule SChopstar-unfoldl-eq State-*
AndEmptySChop int-eq)
have 24: $\vdash (init w) \amalg (schopstar f) = ((init w) \amalg empty \vee (\exists k. (init w) \amalg (fpower f (Suc k))))$
using 1 2 3 by fastforce
have 25: $\vdash ((init w) \amalg empty \vee (\exists k. (init w) \amalg (fpower f (Suc k)))) =$
 $((init w) \amalg empty \vee (\exists k. (fpower ((init w) \amalg f \wedge sfin w) (Suc k)) \frown ((init w) \amalg empty)))$
using 4 by fastforce
have 26: $\vdash ((init w) \amalg empty \vee$
 $(\exists k. (fpower ((init w) \amalg f \wedge sfin w) (Suc k)) \frown ((init w) \amalg empty))) =$
 $(schopstar ((init w) \amalg f \wedge sfin w)) \frown ((init w) \amalg empty)$
using 12 13 by fastforce
have 27: $\vdash (schopstar ((init w) \amalg f \wedge sfin w)) \frown ((init w) \amalg empty) =$
 $((init (\neg w)) \mathcal{U} ((init w) \wedge wnext (\Box (init (\neg w))))) \vee$
 $(((init w) \amalg f \wedge sfin w) \frown (schopstar ((init w) \amalg f \wedge sfin w))) \frown wnext (\Box (init (\neg w)))$
using 14 15 16 by fastforce
have 28: $\vdash (schopstar ((init w) \amalg f \wedge sfin w)) \frown ((init w) \amalg empty) =$
 $(init (\neg w)) \mathcal{U} ((init w) \wedge (schopstar (((init w) \amalg f) \wedge sfin w)) \frown wnext (\Box (init (\neg w))))$
using 27 21 22 23

by (metis inteq-reflection)
 show ?thesis
 by (metis 24 25 26 28 inteq-reflection)
 qed

lemma *PiChopstar-var1*:

$\vdash (init\ w) \Pi (schopstar\ f) =$
 $(schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \curvearrowright ((init\ w) \Pi empty)$
proof –
have 1: $\vdash (init\ w) \Pi (schopstar\ f) = (init\ w) \Pi (\exists k. fpower\ f\ k)$
 by (metis FPowerstardef PiEqvRule SChopstar-FPowerstar inteq-reflection)
have 2: $\vdash (init\ w) \Pi (\exists k. fpower\ f\ k) = (\exists k. (init\ w) \Pi (fpower\ f\ k))$
 by (simp add: Valid-def pi-d-def)
have 3: $\vdash (\exists k. (init\ w) \Pi (fpower\ f\ k)) =$
 $((init\ w) \Pi empty \vee (\exists k. (init\ w) \Pi (fpower\ f\ (Suc\ k))))$
 using PiFPowerExpand by auto
have 4: $\vdash (\exists k. (init\ w) \Pi (fpower\ f\ (Suc\ k))) =$
 $(\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \curvearrowright ((init\ w) \Pi empty))$
 by (meson ExEqvRule PiFPowerSuccd)
have 5: $\vdash (init\ w) \Pi empty = empty \curvearrowright ((init\ w) \Pi empty)$
 by (simp add: EmptySChop int-iffD1 int-iffD2 int-iffI)
have 6: $\vdash (\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \curvearrowright ((init\ w) \Pi empty)) =$
 $(\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \curvearrowright ((init\ w) \Pi empty))$
 by (simp add: ExistSChop)
have 7: $\vdash ((init\ w) \Pi empty \vee (\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \curvearrowright ((init\ w) \Pi empty))) =$
 $(empty \curvearrowright ((init\ w) \Pi empty) \vee$
 $(\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \curvearrowright ((init\ w) \Pi empty)))$
 using 5 6 by fastforce
have 8: $\vdash (empty \curvearrowright ((init\ w) \Pi empty) \vee$
 $(\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \curvearrowright ((init\ w) \Pi empty)) =$
 $(empty \vee (\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \curvearrowright ((init\ w) \Pi empty)))$
 by (meson OrSChopEqv Prop11)
have 9: $\vdash empty = (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ 0)$
 unfolding fpower-d-def by simp
have 10: $\vdash (empty \vee (\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)))) =$
 $((fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ 0) \vee (\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k))))$
 unfolding fpower-d-def by simp
have 11: $\vdash ((fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ 0) \vee (\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)))) =$
 $(\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ k))$
 using exists-expand[of w f] unfolding fpower-d-def
 by (metis (mono-tags) Valid-def inteq-reflection wpow-0 wpowersem1)
have 12: $\vdash ((init\ w) \Pi empty \vee$
 $(\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \curvearrowright ((init\ w) \Pi empty))) =$
 $(\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (k)) \curvearrowright ((init\ w) \Pi empty))$
 by (metis 11 7 8 9 inteq-reflection)
have 13: $\vdash (\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (k)) \curvearrowright ((init\ w) \Pi empty) =$
 $(schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \curvearrowright ((init\ w) \Pi empty)$
 by (metis FPowerstardef LeftSChopEqvSChop SChopstar-FPowerstar inteq-reflection)
 show ?thesis

by (metis 1 12 13 2 3 4 inteq-reflection)
qed

lemma *PiChopstar-var2*:

⊢ (init w) Π (schopstar f) =
((init w) Π empty ∨
(schopstar ((init w) Π f ∧ sfin w)) ∧ (((init w) Π f ∧ sfin w) ∧ (wnext (□ (init (¬w))))))

proof –

have 1: ⊢ (schopstar ((init w) Π f ∧ sfin w)) =
(empty ∨ (schopstar ((init w) Π f ∧ sfin w)) ∧ ((init w) Π f ∧ sfin w))
by (metis (no-types, lifting) DiamondEmptyEqvFinite Prop10 Prop12 SCSEqvOrChopSCSB SChopAndB

SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond int-eq int-iffD2 lift-and-com)

have 2: ⊢ (schopstar ((init w) Π f ∧ sfin w)) ∧ ((init w) Π empty) =
(((init w) Π empty) ∨ ((schopstar ((init w) Π f ∧ sfin w)) ∧ ((init w) Π f ∧ sfin w)) ∧ ((init w) Π empty))

using 1 *EmptyOrSChopEqvRule* **by** blast

have 3: ⊢ ((schopstar ((init w) Π f ∧ sfin w)) ∧ ((init w) Π f ∧ sfin w)) ∧ ((init w) Π empty) =
(schopstar ((init w) Π f ∧ sfin w)) ∧ (((init w) Π f ∧ sfin w) ∧ ((init w) Π empty))

by (meson Prop11 SChopAssoc)

have 4: ⊢ (((init w) Π f ∧ sfin w) ∧ ((init w) Π empty)) =
(((init w) Π f ∧ sfin w) ∧ (wnext (□ (init (¬w))))

by (metis (no-types, lifting) InitAndEmptyEqvAndEmpty SlideInitSFin StateAndEmptySChop StateAnd-PiEmpty inteq-reflection)

have 5: ⊢ (schopstar ((init w) Π f ∧ sfin w)) ∧ (((init w) Π f ∧ sfin w) ∧ ((init w) Π empty)) =
(schopstar ((init w) Π f ∧ sfin w)) ∧ (((init w) Π f ∧ sfin w) ∧ (wnext (□ (init (¬w))))

by (metis 4 RightChopEqvChop schop-d-def)

show ?thesis

by (metis 2 3 5 PiChopstar-var1 int-eq)

qed

lemma *PiChopstar-var3*:

⊢ (init w) Π (f ∧ (schopstar f)) =
(schopstar ((init w) Π f ∧ sfin w)) ∧ (((init w) Π f ∧ sfin w) ∧ (wnext (□ (init (¬w))))

proof –

have 1: ⊢ (init w) Π (f ∧ (schopstar f)) =
((init w) Π f ∧ sfin w) ∧ ((init w) Π (schopstar f))

by (metis PiSChopDist SlideInitSFin inteq-reflection)

have 2: ⊢ ((init w) Π (schopstar f)) = (schopstar ((init w) Π f ∧ sfin w)) ∧ ((init w) Π empty)

using *PiChopstar-var1* **by** auto

have 3: ⊢ ((init w) Π f ∧ sfin w) ∧ ((init w) Π (schopstar f)) =
((init w) Π f ∧ sfin w) ∧ ((schopstar ((init w) Π f ∧ sfin w)) ∧ ((init w) Π empty))

using 2 *RightSChopEqvSChop* **by** blast

have 4: ⊢ ((init w) Π f ∧ sfin w) ∧ ((schopstar ((init w) Π f ∧ sfin w)) ∧ ((init w) Π empty)) =
(((init w) Π f ∧ sfin w) ∧ (schopstar ((init w) Π f ∧ sfin w))) ∧ ((init w) Π empty)

by (simp add: SChopAssoc)

have 5: $\vdash (((init\ w) \Pi f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi f \wedge sfin\ w))) =$
 $(schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown (((init\ w) \Pi f \wedge sfin\ w) \wedge finite)$
by (*simp add: SChopplusCommute*)
have 6: $\vdash (((init\ w) \Pi f \wedge sfin\ w) \wedge finite) = (((init\ w) \Pi f \wedge sfin\ w))$
using *SFinEqvFinAndFinite* **by** *fastforce*
have 7: $\vdash (((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi empty) =$
 $((init\ w) \Pi f \wedge sfin\ w) \frown (wnext\ (\Box (init\ (\neg w))))$
by (*metis (no-types, lifting) InitAndEmptyEqvAndEmpty SlideInitSFin StateAndEmptySChop*
StateAndPiEmpty inteq-reflection)
show ?thesis
by (*metis (no-types, lifting) 1 2 5 6 7 SChopAssoc inteq-reflection*)
qed

lemma *PiChopstar-var4*:

$\vdash (init\ w) \Pi (f \frown (schopstar\ f)) =$
 $((init\ w) \Pi f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown (wnext\ (\Box (init\ (\neg w))))$
proof –
have 1: $\vdash (init\ w) \Pi (f \frown (schopstar\ f)) =$
 $(schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown (((init\ w) \Pi f \wedge sfin\ w) \frown (wnext\ (\Box (init\ (\neg w))))))$
using *PiChopstar-var3* **by** *blast*
have 2: $\vdash (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown (((init\ w) \Pi f \wedge sfin\ w) \frown (wnext\ (\Box (init\ (\neg w)))))) =$
 $((schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi f \wedge sfin\ w)) \frown (wnext\ (\Box (init\ (\neg w))))$
by (*simp add: SChopAssoc*)
have 3: $\vdash ((init\ w) \Pi f \wedge sfin\ w) = (((init\ w) \Pi f \wedge sfin\ w) \wedge finite)$
using *SFinEqvFinAndFinite* **by** *fastforce*
have 4: $\vdash (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown (((init\ w) \Pi f \wedge sfin\ w) \wedge finite) =$
 $((init\ w) \Pi f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi f \wedge sfin\ w))$
by (*simp add: SChopstar-slide-var*)
show ?thesis
by (*metis 1 2 3 4 int-eq*)
qed

lemma *PiChopstar-var5*:

$\vdash (init\ w) \Pi (f \frown (schopstar\ f)) =$
 $(schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi (f \wedge finite))$
proof –
have 1: $\vdash (init\ w) \Pi (f \frown (schopstar\ f)) =$
 $((init\ w) \Pi f \wedge sfin\ w) \frown ((init\ w) \Pi (schopstar\ f))$
by (*metis PiSChopDist SlideInitSFin inteq-reflection*)
have 2: $\vdash ((init\ w) \Pi (schopstar\ f)) =$
 $(schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi empty)$
using *PiChopstar-var1* **by** *auto*
have 3: $\vdash ((init\ w) \Pi f \wedge sfin\ w) \frown ((init\ w) \Pi (schopstar\ f)) =$
 $((init\ w) \Pi f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi empty)$
using *2 RightSChopEqvSChop* **by** *blast*
have 4: $\vdash ((init\ w) \Pi f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi empty) =$
 $((init\ w) \Pi f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi empty)$
using *SChopAssoc* **by** *blast*
have 5: $\vdash (((init\ w) \Pi f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi f \wedge sfin\ w))) =$
 $(schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown (((init\ w) \Pi f \wedge sfin\ w) \wedge finite)$

by (simp add: *SChopplusCommute*)
 have 6: $\vdash (((init\ w) \Pi f \wedge sfin\ w) \wedge finite) = (((init\ w) \Pi f \wedge sfin\ w))$
 using *SFinEqvFinAndFinite* by fastforce
 have 7: $\vdash (((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi empty) =$
 $(init\ w) \Pi (f \frown empty)$
 by (metis *PiSChopDist SlideInitSFin inteq-reflection*)
 have 8: $\vdash (init\ w) \Pi (f \frown empty) = (init\ w) \Pi (f \wedge finite)$
 by (simp add: *ChopEmpty PiEqvRule schop-d-def*)
 have 9: $\vdash (((init\ w) \Pi f \wedge sfin\ w) \frown (s chopstar ((init\ w) \Pi f \wedge sfin\ w))) \frown ((init\ w) \Pi empty) =$
 $(s chopstar ((init\ w) \Pi f \wedge sfin\ w)) \frown (((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi empty)$
 by (metis 4 5 6 *inteq-reflection*)
 have 10: $\vdash (s chopstar ((init\ w) \Pi f \wedge sfin\ w)) \frown (((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi empty) =$
 $(s chopstar ((init\ w) \Pi f \wedge sfin\ w)) \frown (((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi empty)$
 by (meson *Prop11 SChopAssoc*)
 show ?thesis
 by (metis 1 10 3 4 7 8 9 *int-eq*)
 qed

lemma *PiChopstar-var6*:

$\vdash (init\ w) \Pi (s chopstar f) =$
 $(((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \frown (s chopstar ((init\ w) \Pi (f \vee empty) \wedge sfin\ w)))$
 $\frown (wnext(\Box(init\ (\neg w))))$

proof –

have 1: $\vdash (s chopstar f) = (s chopstar (f \vee empty))$
 by (metis *SChopstar-star2 SChopstar-swap inteq-reflection*)
 have 1: $\vdash (init\ w) \Pi (s chopstar (f \vee empty)) =$
 $(s chopstar ((init\ w) \Pi (f \vee empty) \wedge sfin\ w)) \frown ((init\ w) \Pi empty)$
 by (simp add: *PiChopstar-var1*)
 have 2: $\vdash (s chopstar ((init\ w) \Pi (f \vee empty) \wedge sfin\ w)) =$
 $(empty \vee ((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \frown (s chopstar ((init\ w) \Pi (f \vee empty) \wedge sfin\ w)))$
 by (meson *Prop06 SCSEqvOrChopSCSB SChopstar-slide-var*)
 have 3: $\vdash (empty \vee ((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \frown (s chopstar ((init\ w) \Pi (f \vee empty) \wedge sfin\ w))) \frown$
 $((init\ w) \Pi empty)$
 $= (((init\ w) \Pi empty) \vee (((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \frown (s chopstar ((init\ w) \Pi (f \vee empty)$
 $\wedge sfin\ w)))) \frown ((init\ w) \Pi empty)$

using *EmptyOrSChopEqv* by blast

have 4: $\vdash ((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \frown (s chopstar ((init\ w) \Pi (f \vee empty) \wedge sfin\ w)) =$
 $(s chopstar ((init\ w) \Pi (f \vee empty) \wedge sfin\ w)) \frown (((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \wedge finite)$

by (simp add: *SChopplusCommute*)

have 5: $\vdash (((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \wedge finite) =$
 $((init\ w) \Pi (f \vee empty) \wedge sfin\ w)$

using *SFinEqvFinAndFinite* by fastforce

have 6: $\vdash (((init\ w) \Pi (f \vee empty) \wedge sfin\ w)) \frown ((init\ w) \Pi empty) =$
 $((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \frown (wnext(\Box(init\ (\neg w))))$

by (metis (no-types, lifting) *InitAndEmptyEqvAndEmpty SlideInitSFin StateAndEmptySChop StateAndPiEmpty int-eq*)

have 7: $\vdash (((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \frown (s chopstar ((init\ w) \Pi (f \vee empty) \wedge sfin\ w))) \frown ((init\ w)$

$\Pi \text{ empty}) =$
 $((((\text{init } w) \Pi (f \vee \text{empty}) \wedge \text{sfin } w) \neg (\text{schopstar } ((\text{init } w) \Pi (f \vee \text{empty}) \wedge \text{sfin } w))) \neg (\text{wnext}(\Box(\text{init } (\neg w))))))$
by (*metis* (*no-types, lifting*) 4 5 6 *SChopAssoc inteq-reflection*)
have 8: $\vdash (((\text{init } w) \Pi \text{empty}) \vee (((\text{init } w) \Pi (f \vee \text{empty}) \wedge \text{sfin } w) \neg (\text{schopstar } ((\text{init } w) \Pi (f \vee \text{empty}) \wedge \text{sfin } w))) \neg ((\text{init } w) \Pi \text{empty})) =$
 $((((\text{init } w) \Pi (f \vee \text{empty}) \wedge \text{sfin } w) \neg (\text{schopstar } ((\text{init } w) \Pi (f \vee \text{empty}) \wedge \text{sfin } w))) \neg ((\text{init } w) \Pi \text{empty}))$
by (*metis* 1 2 3 7 *FiniteAndEmptyEqvEmpty PiChopstar-var4 Prop05 SChopstar-sup-id-star1 int-iffD1 inteq-reflection*)
show ?thesis
by (*meson* *PiChopstar-var4 PiEqvRule Prop04 SChopstar-star2 SChopstar-sup-id-star1 SChopstar-swap UntilImpOr UntilIntro lift-imp-trans*)
qed

2.9.6 Omega and Pi

lemma *OmegaImpDiamond*:

$\vdash (\text{omega } ((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{sfin } w)) \longrightarrow \Diamond (\text{init } w)$
proof –
have 1: $\vdash (\text{omega } ((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{sfin } w)) =$
 $(((((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{sfin } w) \wedge \text{more}) \wedge \text{finite}); (\text{omega } ((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{sfin } w)))$
using *OmegaUnroll*[*of LIFT* $((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{sfin } w)]$
by (*simp* *add: schop-d-def*)
have 2: $\vdash (((((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{sfin } w) \wedge \text{more}) \wedge \text{finite}) \longrightarrow \Diamond (\text{init } w))$
using *PiImpDiamond* **by** *fastforce*
have 3: $\vdash (((((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{sfin } w) \wedge \text{more}) \wedge \text{finite}) =$
 $(((((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{more}) \wedge \text{sfin } w) \wedge \text{finite}))$
by *auto*
have 4: $\vdash ((((((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{sfin } w) \wedge \text{more}) \wedge \text{finite}); (\text{omega } ((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{sfin } w)))$
 $=$
 $(((((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{more}) \wedge \text{sfin } w) \wedge \text{finite}); (\text{omega } ((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{sfin } w)))$
using 3 *LeftChopEqvChop* **by** *blast*
have 5: $\vdash ((((((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{more}) \wedge \text{sfin } w) \wedge \text{finite}); (\text{omega } ((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{sfin } w)))$
 $= ((((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{more}) \wedge \text{finite}); ((\text{init } w) \wedge \text{omega } ((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{sfin } w)))$
using *SlideInitSFin*[*of LIFT* $((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{more})$ *w*
 $\text{LIFT } (\text{omega } ((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{sfin } w))]$ **unfolding** *schop-d-def*
by (*metis* *inteq-reflection*)
have 6: $\vdash (((((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{more}) \wedge \text{finite}); ((\text{init } w) \wedge \text{omega } ((\text{init } w) \Pi (f \wedge \text{more}) \wedge \text{sfin } w)))$
 $\longrightarrow \Diamond (\text{init } w))$
by (*metis* *ChopAndB FiniteChopImpDiamond inteq-reflection lift-and-com lift-imp-trans*)
show ?thesis **using** 1 3 4 5 6 **by** (*metis* *int-eq*)
qed

lemma *PiStateOmegaUnroll*:

$\vdash (\text{init } w) \Pi (\text{omega } f) = ((\text{init } w) \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{more}) \wedge \text{sfin } w; ((\text{init } w) \Pi (\text{omega } f))$
proof –
have 1: $\vdash (\text{init } w) \Pi (\text{omega } f) = (\text{init } w) \Pi (((f \wedge \text{more}) \wedge \text{finite}); (\text{omega } f))$

by (*metis OmegaUnroll PiEqvRule schop-d-def*)
have 2: $\vdash ((init\ w) \Pi ((f \wedge more) \wedge finite); (\omega\ f)) =$
 $((init\ w) \Pi (f \wedge more) \wedge finite); ((init\ w) \wedge (init\ w) \Pi (\omega\ f))$
using *PiSChopDist*[of *w LIFT (f \wedge more) LIFT (\omega\ f)*] **unfolding** *s chop-d-def*
by *simp*
have 3: $\vdash ((init\ w) \Pi (f \wedge more) \wedge finite) = ((init\ w) \Pi ((f \wedge more) \wedge finite) \wedge finite)$
by (*meson PiFiniteAbsorb Prop11*)
have 4: $\vdash ((init\ w) \Pi (f \wedge more) \wedge finite); ((init\ w) \wedge (init\ w) \Pi (\omega\ f)) =$
 $((init\ w) \Pi ((f \wedge more) \wedge finite) \wedge finite); ((init\ w) \wedge (init\ w) \Pi (\omega\ f))$
by (*simp add: 3 LeftChopEqvChop*)
have 5: $\vdash ((init\ w) \Pi ((f \wedge more) \wedge finite) \wedge finite); ((init\ w) \wedge (init\ w) \Pi (\omega\ f)) =$
 $((init\ w) \Pi ((f \wedge more) \wedge finite) \wedge sfin\ w) \wedge finite); ((init\ w) \Pi (\omega\ f))$
using *SlideInitSFin*[of *LIFT (init\ w) \Pi ((f \wedge more) \wedge finite) w LIFT (init\ w) \Pi (\omega\ f)*]
unfolding *s chop-d-def* **by** *blast*
have 6: $\vdash (((init\ w) \Pi ((f \wedge more) \wedge finite) \wedge sfin\ w) \wedge finite) =$
 $((init\ w) \Pi ((f \wedge more) \wedge finite) \wedge sfin\ w)$
using *SFinEqvFinAndFinite* **by** *fastforce*
have 7: $\vdash (init\ w) \Pi ((f \wedge more) \wedge finite) = ((init\ w) \Pi ((f \wedge more) \wedge finite) \wedge more)$
by (*metis PiAnd PiAndPiImpPiAnd PiMoreAbsorb Prop10 Prop12 inteq-reflection*)
have 8: $\vdash (((init\ w) \Pi ((f \wedge more) \wedge finite) \wedge sfin\ w)) =$
 $((init\ w) \Pi ((f \wedge more) \wedge finite) \wedge more) \wedge sfin\ w)$
using 7 **by** *auto*
show ?thesis
by (*metis 1 2 4 5 6 8 int-eq*)
qed

lemma *PiAOmega-sfin:*

assumes $\neg nfinite\ l$
 $nnth\ l\ 0 = 0$
 $nidx\ l$
 $(\forall i. enat\ (nnth\ l\ i) \leq nlength\ \sigma)$
shows $((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models sfin\ w) =$
 $w\ (NNil\ (nnth\ \sigma\ (nnth\ l\ (Suc\ i))))$
using *assms nidx-expand*[of *l*]
by (*simp add: sfin-defs*)
(metis diff-add less-imp-le-nat linorder-le-cases ndropn-nlast nfinite-ntaken nsubn-def1 ntaken-all ntaken-ndropn-nlast)

lemma *PiAOmega-sfin-exist:*

assumes $\neg nfinite\ l$
 $nnth\ l\ 0 = 0$
 $nidx\ l$
 $(\forall i. enat\ (nnth\ l\ i) \leq nlength\ \sigma)$
 $w\ (NNil\ (nnth\ \sigma\ (nnth\ l\ (Suc\ i))))$
shows $(\exists x \in nset\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) . w\ (NNil\ x))$
using *assms nidx-expand*[of *l*] *in-nset-conv-nnth*[of $-(nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))$]
unfolding *nsubn-def1*
by (*metis diff-add less-imp-le-nat linorder-le-cases nfinite-ntaken nset-nlast ntaken-all ntaken-ndropn-nlast*)

lemma *PiAOmega-help2*:

$(\sigma \models (\text{aomega } ((\text{init } w) \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfin } w))) =$
 $(\exists l. \neg \text{nfinite } l \wedge$
 $\quad \text{nnth } l \ 0 = 0 \wedge$
 $\quad \text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$
 $\quad (\forall i. ($
 $\quad \quad ((\text{nfilter } (\lambda y. w \ (\text{NNil } y)) (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)))) \models f) \wedge$
 $\quad \quad 0 < \text{nlength } (\text{nfilter } (\lambda y. w \ (\text{NNil } y)) (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)))) \wedge$
 $\quad \quad \text{nfinite } (\text{nfilter } (\lambda y. w \ (\text{NNil } y)) (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))))$
 $\quad \quad \wedge w \ (\text{NNil } (\text{nnth } \sigma \ (\text{nnth } l \ (\text{Suc } i))))$
 $\quad)$
 $\quad)$

proof –

have 1: $(\sigma \models (\text{aomega } ((\text{init } w) \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfin } w))) =$
 $(\exists l. \neg \text{nfinite } l \wedge$
 $\quad \text{nnth } l \ 0 = 0 \wedge$
 $\quad \text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$
 $\quad (\forall i. \text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)) \models \text{init } w \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfin } w))$

unfolding *aomega-d-def* **by** *blast*

have 2: $(\exists l. \neg \text{nfinite } l \wedge$
 $\quad \text{nnth } l \ 0 = 0 \wedge$
 $\quad \text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$
 $\quad (\forall i. \text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)) \models \text{init } w \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfin } w)) =$
 $(\exists l. \neg \text{nfinite } l \wedge$
 $\quad \text{nnth } l \ 0 = 0 \wedge$
 $\quad \text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$
 $\quad (\forall i. ($
 $\quad \quad (\exists x \in \text{nset } (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) . w \ (\text{NNil } x)) \wedge$
 $\quad \quad (\text{nfilter } (\lambda y. w \ (\text{NNil } y)) (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) \models (f \wedge \text{more}) \wedge \text{finite})$
 $\quad \quad \wedge ((\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) \models \text{sfin } w)$
 $\quad)$
 $\quad)$

using *Pistate*[of *w LIFT* $((f \wedge \text{more}) \wedge \text{finite})$] **by** *auto*

have 3: $(\exists l. \neg \text{nfinite } l \wedge$
 $\quad \text{nnth } l \ 0 = 0 \wedge$
 $\quad \text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$
 $\quad (\forall i. ($
 $\quad \quad (\exists x \in \text{nset } (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) . w \ (\text{NNil } x)) \wedge$
 $\quad \quad (\text{nfilter } (\lambda y. w \ (\text{NNil } y)) (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) \models (f \wedge \text{more}) \wedge \text{finite})$
 $\quad \quad \wedge ((\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) \models \text{sfin } w)$
 $\quad)$
 $\quad) =$
 $(\exists l. \neg \text{nfinite } l \wedge$
 $\quad \text{nnth } l \ 0 = 0 \wedge$
 $\quad \text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$
 $\quad (\forall i. ($
 $\quad \quad (\exists x \in \text{nset } (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) . w \ (\text{NNil } x)) \wedge$
 $\quad \quad ((\text{nfilter } (\lambda y. w \ (\text{NNil } y)) (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)))) \models f) \wedge$
 $\quad \quad 0 < \text{nlength } (\text{nfilter } (\lambda y. w \ (\text{NNil } y)) (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)))) \wedge$
 $\quad \quad \text{nfinite } (\text{nfilter } (\lambda y. w \ (\text{NNil } y)) (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))))$
 $\quad \quad \wedge ((\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) \models \text{sfin } w)$
 $\quad)$

)
))
unfolding *more-defs finite-defs by simp*
have 4: $(\exists l. \neg \text{nfinite } l \wedge$
 $\text{nnth } l \ 0 = 0 \wedge$
 $\text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. ((\exists x \in \text{nset } (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))) . w (\text{NNil } x)) \wedge$
 $((\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)))) \models f) \wedge$
 $0 < \text{nlength } (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)))) \wedge$
 $\text{nfinite } (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))))$
 $\wedge ((\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))) \models \text{sfin } w)$
 $)$
 $)) =$
 $(\exists l. \neg \text{nfinite } l \wedge$
 $\text{nnth } l \ 0 = 0 \wedge$
 $\text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. ((\exists x \in \text{nset } (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))) . w (\text{NNil } x)) \wedge$
 $((\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)))) \models f) \wedge$
 $0 < \text{nlength } (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)))) \wedge$
 $\text{nfinite } (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))))$
 $\wedge w (\text{NNil } (\text{nnth } \sigma (\text{nnth } l \ (\text{Suc } i))))$
 $)$
 $))$
using *PiAOmega-sfin[of - σ w] by blast*
have 5: $(\exists l. \neg \text{nfinite } l \wedge$
 $\text{nnth } l \ 0 = 0 \wedge$
 $\text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. ((\exists x \in \text{nset } (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))) . w (\text{NNil } x)) \wedge$
 $((\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)))) \models f) \wedge$
 $0 < \text{nlength } (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)))) \wedge$
 $\text{nfinite } (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))))$
 $\wedge w (\text{NNil } (\text{nnth } \sigma (\text{nnth } l \ (\text{Suc } i))))$
 $)$
 $)) =$
 $(\exists l. \neg \text{nfinite } l \wedge$
 $\text{nnth } l \ 0 = 0 \wedge$
 $\text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. ($
 $((\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)))) \models f) \wedge$
 $0 < \text{nlength } (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)))) \wedge$
 $\text{nfinite } (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))))$
 $\wedge w (\text{NNil } (\text{nnth } \sigma (\text{nnth } l \ (\text{Suc } i))))$
 $)$
 $))$
using *PiAOmega-sfin-exist[of - σ w] by blast*
show ?thesis
using 1 2 3 4 5 **by** presburger
qed

primcorec *oml* :: ('a \Rightarrow bool) \Rightarrow 'a nellist \Rightarrow nat nellist \Rightarrow nat \Rightarrow nat nellist
where

oml *P* *xs* *l* *x* =
 NCons (case *x* of 0 \Rightarrow 0 |
 Suc *j* \Rightarrow (the-enat (nlength(nfilter *P* (ntaken (nnth *l* *x*) *xs*))))
 (*oml* *P* *xs* *l* (*Suc* *x*)))

lemma *oml-nnth-zero*:

nnth (*oml* *P* *xs* *l* 0) 0 = 0
using *oml.disc-iff* *oml.simps*(2) *nhd-conv-nnth*
by (*metis* *old.nat.simps*(4))

lemma *oml-nnth-one*:

assumes \neg *nfinite* *l*
 \neg *nfinite* *xs*
shows *nnth* (*oml* *P* *xs* *l* 0) 1 = (the-enat (nlength(nfilter *P* (ntaken (nnth *l* 1) *xs*))))
using *assms*
oml.code *oml.disc-iff* *oml.simps*(2) *nhd-conv-nnth* *nnth-Suc-NCons*
by (*metis* *One-nat-def* *old.nat.simps*(5))

lemma *oml-nnth*:

assumes \neg *nfinite* *l*
 \neg *nfinite* *xs*
shows *nnth* (*oml* *P* *xs* *l* *x*) *i* =
 (if *x* = 0 then
 (if *i* = 0 then 0 else (the-enat (nlength(nfilter *P* (ntaken (nnth *l* *i*) *xs*))))
 else (the-enat (nlength(nfilter *P* (ntaken (nnth *l* (*i*+*x*)) *xs*))))
using *assms*
proof (induct *i* arbitrary: *x*)
case 0
then show ?case
 by (*metis* *Nitpick.case-nat-unfold* *ndropn-0* *ndropn-nnth* *nhd-conv-nnth* *oml.disc-iff* *oml.simps*(2))
next
case (*Suc* *i*)
then show ?case
 by (*metis* *One-nat-def* *Zero-not-Suc* *add commute* *add-Suc-shift* *nnth-Suc-NCons* *oml.code* *plus-1-eq-Suc*)
qed

lemma *oml-infinite*:

assumes \neg *nfinite* *l*
 \neg *nfinite* *xs*
shows \neg *nfinite* (*oml* *P* *xs* *l* *x*)
proof
assume *nfinite* (*oml* *P* *xs* *l* *x*)
thus *False*
using *assms*
proof (induct *zs* \equiv (*oml* *P* *xs* *l* *x*) arbitrary: *x* rule: *nfinite-induct*)
case (*NNil* *y*)
then show ?case **by** (*metis* *nellist.disc*(1) *oml.disc-iff*)

```

next
case (NCons x nell)
then show ?case by (metis nellist.sel(5) oml.simps(3))
qed
qed

```

lemma *PiAOmegaImpSem:*

```

assumes (σ ⊨ (aomega ((init w) Π ((f ∧ more) ∧ finite) ∧ sfin w)))
shows (σ ⊨ (init w) Π (aomega f))
proof -
  have 1: (∃ l. ¬ nfinite l ∧
    nnth l 0 = 0 ∧
    nidx l ∧ (∀ i. enat (nnth l i) ≤ nlength σ) ∧
    (∀ i. (
      ((nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i)))) ⊨ f) ∧
      0 < nlength (nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i)))) ∧
      nfinite (nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i))))
      ∧ w (NNil (nnth σ (nnth l (Suc i))))
    )
  ) )
  using assms PiAOmega-help2[of w f σ] by blast
  obtain l where 2: ¬ nfinite l ∧
    nnth l 0 = 0 ∧
    nidx l ∧ (∀ i. enat (nnth l i) ≤ nlength σ) ∧
    (∀ i. (
      ((nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i)))) ⊨ f) ∧
      0 < nlength (nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i)))) ∧
      nfinite (nfilter (λy. w (NNil y)) (nsubn σ (nnth l i) (nnth l (Suc i))))
      ∧ w (NNil (nnth σ (nnth l (Suc i))))
    )
  )
  using 1 by auto
  have 3: ∧j. 0 < j ⟶ w (NNil (nnth σ (nnth l j)))
  using 2 by (metis Suc-diff-1)
  have 31: ∧j. 0 < j ⟶ nfinite (nfilter (λy. w (NNil y)) (ntaken (nnth l j) σ))
  by (metis 2 3 nfilter-chop1-ntaken nfinite-ntaken ntaken-nlast)
  have 32: ∧j. (nnth l j) < (nnth l (Suc j))
  by (metis 2 linorder-le-cases nfinite-ntaken nidx-expand ntaken-all)
  have 4: ∧j. 0 < j ⟶
    (nfilter (λy. w (NNil y)) (nsubn σ (nnth l j) (nnth l (Suc j)))) =
    (nsubn (nfilter (λy. w (NNil y)) σ )
      (the-enat (nlength(nfilter (λy. w (NNil y)) (ntaken (nnth l j) σ))))
      (the-enat (nlength(nfilter (λy. w (NNil y)) (ntaken (nnth l (Suc j)) σ))))
    )

  using 2 3 32 nfilter-nsubn[of - σ (λy. w (NNil y))]
  by (metis 31 less-imp-le-nat nfinite-nlength-enat the-enat.simps zero-less-Suc)
  have 5: (nfilter (λy. w (NNil y)) (nsubn σ 0 (nnth l (Suc 0)))) =
    (nsubn (nfilter (λy. w (NNil y)) σ )

```

0
 $(the-enat (nlength(nfilter (\lambda y. w (NNil y)) (ntaken (nnth l (Suc 0)) \sigma))))$
 $)$
by (*simp add: 2 nfilter-nsubn-zero*)
have 6: $0 < (the-enat (nlength(nfilter (\lambda y. w (NNil y)) (ntaken (nnth l (Suc 0)) \sigma))))$
by (*metis 2 5 grOI i0-less nlength-NNil nsubn-def1 ntaken-0 zero-less-diff*)
have 7: $\bigwedge j. 0 < j \longrightarrow (the-enat (nlength(nfilter (\lambda y. w (NNil y)) (ntaken (nnth l j) \sigma)))) <$
 $(the-enat (nlength(nfilter (\lambda y. w (NNil y)) (ntaken (nnth l (Suc j)) \sigma))))$
by (*metis 2 4 bot-nat-0.not-eq-extremum i0-less nlength-NNil nsubn-def1 ntaken-0 zero-less-diff*)
have 8: $\neg nfinite (oml (\lambda y. w (NNil y)) \sigma l 0)$
using 2 *infinite-nidx-imp-infinite-interval oml-infinite* **by** *blast*
have 9: $nnth (oml (\lambda y. w (NNil y)) \sigma l 0) = 0$
by (*simp add: oml-nnth-zero*)
have 10: $\bigwedge j. 0 < j \longrightarrow nnth (oml (\lambda y. w (NNil y)) \sigma l 0) j =$
 $(the-enat (nlength(nfilter (\lambda y. w (NNil y)) (ntaken (nnth l j) \sigma))))$
by (*metis 2 infinite-nidx-imp-infinite-interval less-not-refl2 oml-nnth*)
have 11: $\bigwedge j. nnth (oml (\lambda y. w (NNil y)) \sigma l 0) j < nnth (oml (\lambda y. w (NNil y)) \sigma l 0) (Suc j)$
by (*metis 10 6 7 9 bot-nat-0.not-eq-extremum zero-less-Suc*)
have 12: $nidx (oml (\lambda y. w (NNil y)) \sigma l 0)$
using *nidx-expand[of (oml (\lambda y. w (NNil y)) \sigma l 0)]*
using 11 **by** *blast*
have 13: $(\forall i. enat (nnth (oml (\lambda y. w (NNil y)) \sigma l 0) i) \leq nlength (nfilter (\lambda y. w (NNil y)) \sigma))$
by (*metis 10 3 bot-nat-0.not-eq-extremum enat-ile le-zero-eq linorder-le-cases*
nfilter-chop1-ntaken ntaken-all ntaken-nlast oml-nnth-zero the-enat.simps zero-enat-def)
have 14: $(\forall i. f (nsubn (nfilter (\lambda y. w (NNil y)) \sigma)$
 $(nnth (oml (\lambda y. w (NNil y)) \sigma l 0) i)$
 $(nnth (oml (\lambda y. w (NNil y)) \sigma l 0) (Suc i))))$
by (*metis 10 2 4 5 9 bot-nat-0.not-eq-extremum zero-less-Suc*)
have 15: $(\exists x \in nset \sigma. w (NNil x))$
by (*meson 2 in-nset-conv-nnth*)
have 16: $((\exists x \in nset \sigma. w (NNil x)) \wedge$
 $(\exists l. \neg nfinite l \wedge$
 $nnth l 0 = 0 \wedge$
 $nidx l \wedge$
 $(\forall i. enat (nnth l i) \leq nlength (nfilter (\lambda y. w (NNil y)) \sigma)) \wedge$
 $(\forall i. f (nsubn (nfilter (\lambda y. w (NNil y)) \sigma) (nnth l i) (nnth l (Suc i))))))$
using 12 13 14 15 8 *oml-nnth-zero* **by** *blast*
have 17: $(\sigma \models (init w) \Pi (aomega f))$
using 16
using *Pistate[of w LIFT aomega f \sigma]* **unfolding** *aomega-d-def* **by** *blast*
show *?thesis* **using** 17 **by** *auto*
qed

lemma *PiAOmegaImp*:

$\vdash (aomega ((init w) \Pi ((f \wedge more) \wedge finite) \wedge sfin w)) \longrightarrow (init w) \Pi (aomega f)$

unfolding *Valid-def* **using** *PiAOmegaImpSem*

using *unl-lift2* **by** *blast*

lemma *PiOmegaImpSem*:

assumes $(\sigma \models (\text{omega } ((\text{init } w) \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfine } w)))$
shows $(\sigma \models (\text{init } w) \Pi (\text{omega } f))$
by $(\text{metis } \text{OmegaEqvAOmega } \text{PiAOmegaImpSem } \text{assms } \text{inteq-reflection})$

lemma *PiOmegaImp*:

$\vdash (\text{omega } ((\text{init } w) \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfine } w)) \longrightarrow (\text{init } w) \Pi (\text{omega } f)$
unfolding *Valid-def* **using** *PiOmegaImpSem* **using** *unl-lift2* **by** *blast*

lemma *PiOmega*:

$\vdash (\text{init } w) \Pi (\text{omega } f) = (\text{omega } ((\text{init } w) \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfine } w))$
proof –
have 0: $\vdash (((\text{init } w) \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{more}) \wedge \text{sfine } w) =$
 $(((((\text{init } w) \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfine } w) \wedge \text{more}) \wedge \text{finite})$
using *SFinEqvFinAndFinite* **by** *fastforce*
have 1: $\vdash (\text{init } w) \Pi (\text{omega } f) \longrightarrow \text{inf}$
by $(\text{meson } \text{PiImpRule } \text{PiInfImpInf } \text{lift-imp-trans } \text{omega-imp-inf})$
have 2: $\vdash (\text{init } w) \Pi (\text{omega } f) \longrightarrow$
 $\text{inf} \wedge (((\text{init } w) \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfine } w) \wedge \text{fmore}); ((\text{init } w) \Pi (\text{omega } f))$
by $(\text{metis } 1 \text{ AndMoreSChopEqvAndFmoreChop } 0 \text{ PiStateOmegaUnroll Prop12 int-eq int-iffD1 schop-d-def})$
have 3: $\vdash (\text{init } w) \Pi (\text{omega } f) \longrightarrow (\text{omega } ((\text{init } w) \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfine } w))$
using *OmegaIntro* **by** $(\text{metis } 2 \text{ Prop12})$
have 4: $\vdash (\text{omega } ((\text{init } w) \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfine } w)) \longrightarrow (\text{init } w) \Pi (\text{omega } f)$
by $(\text{simp add: } \text{PiOmegaImp})$
show *?thesis*
by $(\text{simp add: } 3 \ 4 \ \text{int-iffI})$
qed

end

2.10 Quantifiers

theory *FOTheorems*

imports

SChopTheorems Chopstar

begin

We give the proofs of a list of first order (in)finite ITL theorems.

lemma *EExI-unl*:

$w \models f \ x \implies w \models (\exists \exists \ x. f \ x)$
by $(\text{meson } \text{exist-state-d-def})$

lemma *EExNoDep*:

$\vdash (\exists \exists \ x. g) = g$
proof –
have 1: $\vdash g \longrightarrow (\exists \exists \ x. g)$ **by** $(\text{meson } \text{EExI})$

have 2: $\bigwedge x. \vdash g \longrightarrow g$ **by** *simp*
have 3: $\vdash (\exists \exists x. g) \longrightarrow g$ **using** 2 **by** (*meson EExE*)
from 1 3 **show** ?thesis **using** *int-iffI* **by** *blast*
qed

lemma *AAxNoDep*:
 $\vdash (\forall \forall x. g) = g$
using *EExNoDep*[of *LIFT*($\neg g$)] *AAxDef* *EExE* *EExI*
by (*simp add: exist-state-d-def forall-state-d-def intI*)

lemma *EExEqvRule*:
assumes $\bigwedge x. \vdash f x = g x$
shows $\vdash (\exists \exists x. f x) = (\exists \exists x. g x)$
by (*metis EExE EExI assms int-iffD1 int-iffD2 int-iffI lift-imp-trans*)

lemma *AAxImpEEx*:
 $\vdash (\forall \forall x. f x) \longrightarrow (\exists \exists x. f x)$
by (*simp add: exist-state-d-def forall-state-d-def intI*)

lemma *EExImpRule*:
assumes $\vdash f x \longrightarrow g x$
shows $\vdash (\exists \exists x. f x \longrightarrow g x)$
using *assms* **by** (*meson MP EExI*)

lemma *EExImpRuleDist*:
assumes $\vdash f x \longrightarrow g x$
shows $\vdash (\forall \forall x. f x) \longrightarrow (\exists \exists x. g x)$
proof –
have 1: $\vdash (f x) \longrightarrow (\exists \exists x. g x)$ **using** *EExI* *assms* *lift-imp-trans* **by** *blast*
have 2: $\vdash \neg(f x) \vee (\exists \exists x. g x)$ **using** 1 **by** *auto*
have 3: $\vdash \neg(f x) \longrightarrow (\exists \exists x. \neg(f x))$ **by** (*meson EExI*)
have 4: $\vdash (\exists \exists x. \neg(f x)) = (\neg(\forall \forall x. f x))$ **using** *AAxDef* **by** *fastforce*
from 2 3 4 **show** ?thesis **by** *fastforce*
qed

lemma *EExImpNoDepDist*:
assumes $\vdash f \longrightarrow g x$
shows $\vdash f \longrightarrow (\exists \exists x. g x)$
using *assms* **by** (*metis EExI lift-imp-trans*)

lemma *EExOrDist-1*:
 $\vdash (\exists \exists x. h x) \longrightarrow (\exists \exists x. (f x) \vee (h x))$
proof –
have 1: $\bigwedge x. \vdash h x \longrightarrow f x \vee h x$ **by** (*simp add: Valid-def*)
have 2: $\bigwedge x. \vdash f x \vee h x \longrightarrow (\exists \exists x. (f x) \vee (h x))$ **by** (*meson EExI*)
have 3: $\bigwedge x. \vdash h x \longrightarrow (\exists \exists x. (f x) \vee (h x))$ **using** 1 2 **by** (*meson lift-imp-trans*)
from 3 **show** ?thesis **using** *EExE* **by** *blast*
qed

lemma *EExOrDist-2*:

$\vdash (\exists \exists x. f x) \longrightarrow (\exists \exists x. (f x) \vee (h x))$
proof –
have 1: $\bigwedge x. \vdash f x \longrightarrow f x \vee h x$ **by** (*simp add: Valid-def*)
have 2: $\bigwedge x. \vdash f x \vee h x \longrightarrow (\exists \exists x. (f x) \vee (h x))$ **by** (*meson EExI*)
have 3: $\bigwedge x. \vdash f x \longrightarrow (\exists \exists x. (f x) \vee (h x))$ **using** 1 2 **by** (*meson lift-imp-trans*)
from 3 **show** ?thesis **using** EExE **by** blast
qed

lemma EExOrDist-3:
 $\vdash (\exists \exists x. f x) \vee (\exists \exists x. h x) \longrightarrow (\exists \exists x. (f x) \vee (h x))$
using EExOrDist-2 EExOrDist-1 **by** fastforce

lemma EExOrDist-4:
 $\vdash (\exists \exists x. (f x) \vee (h x)) \longrightarrow (\exists \exists x. f x) \vee (\exists \exists x. h x)$
proof –
have 1: $\bigwedge x. \vdash (f x) \vee (h x) \longrightarrow (\exists \exists x. f x) \vee (\exists \exists x. h x)$
by (*simp add: EExI-unl intI*)
from 1 **show** ?thesis **by** (*simp add: EExE*)
qed

lemma EExOrDist:
 $\vdash ((\exists \exists x. f x) \vee (\exists \exists x. h x)) = (\exists \exists x. (f x) \vee (h x))$
using EExOrDist-3 EExOrDist-4 **by** fastforce

lemma EExOrImport-1:
 $\vdash g \longrightarrow (\exists \exists x. g \vee (f x))$
by (*simp add: EExI-unl Valid-def*)

lemma EExOrImport-2:
 $\vdash (\exists \exists x. f x) \longrightarrow (\exists \exists x. g \vee (f x))$
by (*simp add: EExOrDist-1*)

lemma EExOrImport-3:
 $\vdash (g \vee (\exists \exists x. f x)) \longrightarrow (\exists \exists x. g \vee (f x))$
using EExOrImport-1 EExOrImport-2 **by** fastforce

lemma EExOrImport-4:
 $\vdash (\exists \exists x. g \vee f x) \longrightarrow (g \vee (\exists \exists x. f x))$
proof –
have 1: $\bigwedge x. \vdash g \vee f x \longrightarrow g \vee (\exists \exists x. f x)$ **by** (*meson EExI int-iffD2 int-simps(27) Prop04 Prop08*)
from 1 **show** ?thesis **by** (*simp add: EExE*)
qed

lemma EExOrImport:
 $\vdash (g \vee (\exists \exists x. f x)) = (\exists \exists x. g \vee f x)$
by (*metis EExOrImport-3 EExOrImport-4 int-iffI*)

lemma EExAndImport-1:
 $\vdash g \wedge (\exists \exists x. f x) \longrightarrow (\exists \exists x. g \wedge f x)$
proof –

have 1: $\vdash (g \wedge (\exists \exists x. f x) \longrightarrow (\exists \exists x. g \wedge f x)) = ((\exists \exists x. f x) \longrightarrow (g \longrightarrow (\exists \exists x. g \wedge f x)))$
by *fastforce*
have 2: $\bigwedge x. \vdash f x \longrightarrow (g \longrightarrow (\exists \exists x. g \wedge f x))$
using *EEExI[of $\lambda x. LIFT(g \wedge f x)$]* **by** *fastforce*
hence 3: $\vdash (\exists \exists x. f x) \longrightarrow (g \longrightarrow (\exists \exists x. g \wedge f x))$
by (*simp add: EExE*)
from 1 3 **show** *?thesis* **by** *auto*
qed

lemma *EExAndImport-2*:
 $\vdash (\exists \exists x. g \wedge f x) \longrightarrow g \wedge (\exists \exists x. f x)$
proof –
have 1: $\bigwedge x. \vdash g \wedge f x \longrightarrow g \wedge (\exists \exists x. f x)$
by (*metis EExI int-iffD2 lift-and-com lift-imp-trans Prop12*)
from 1 **show** *?thesis* **by** (*simp add: EExE*)
qed

lemma *EExAndImport*:
 $\vdash (g \wedge (\exists \exists x. f x)) = (\exists \exists x. g \wedge f x)$
by (*simp add: EExAndImport-1 EExAndImport-2 int-iffI*)

lemma *EExAndDist*:
assumes $\vdash f x \wedge g x$
shows $\vdash (\exists \exists x. f x) \wedge (\exists \exists x. g x)$
proof –
have 1: $\vdash f x$ **using** *assms* **by** *fastforce*
have 2: $\vdash g x$ **using** *assms* **by** *fastforce*
have 3: $\vdash (\exists \exists x. f x)$ **using** 1 **by** (*meson EExI MP*)
have 4: $\vdash (\exists \exists x. g x)$ **using** 2 **by** (*meson EExI MP*)
from 3 4 **show** *?thesis* **by** *fastforce*
qed

lemma *EExAndNoDepDist*:
assumes $\vdash f \wedge g x$
shows $\vdash f \wedge (\exists \exists x. g x)$
proof –
have 1: $\vdash f$ **using** *assms* **by** *fastforce*
have 2: $\vdash g x$ **using** *assms* **by** *fastforce*
have 3: $\vdash (\exists \exists x. g x)$ **using** 2 **by** (*meson EExI MP*)
from 1 3 **show** *?thesis* **by** *fastforce*
qed

lemma *Spec*:
 $\vdash (\forall \forall x. f x) \longrightarrow f x$
proof –
have 1: $\vdash \neg(f x) \longrightarrow (\exists \exists x. \neg(f x))$ **by** (*meson EExI*)
have 2: $\vdash \neg(\exists \exists x. \neg(f x)) \longrightarrow f x$ **using** 1 **by** *auto*
from 2 **show** *?thesis* **using** *AAxDef* **by** *fastforce*
qed

lemma *AAxE*:
assumes $\vdash (\forall \forall x. f x)$
 $\vdash f x \longrightarrow g$
shows $\vdash g$
using *MP Spec assms(1) assms(2) by blast*

lemma *AAxI*:
assumes $\bigwedge x. \vdash f x$
shows $\vdash (\forall \forall x. f x)$
using *assms* **by** (*simp add: Valid-def exist-state-d-def forall-state-d-def*)

lemma *AAxEqvRule*:
assumes $\bigwedge x. \vdash f x = g x$
shows $\vdash (\forall \forall x. f x) = (\forall \forall x. g x)$
unfolding *forall-state-d-def* **using** *assms EExEqvRule*[*of* $\lambda x. (LIFT(\neg f x)) \lambda x. (LIFT(\neg g x))$] **by** *fastforce*

lemma *AAxAndDist*:
 $\vdash (\forall \forall x. (f x) \wedge (g x)) = ((\forall \forall x. f x) \wedge (\forall \forall x. g x))$
proof –
have 1: $\bigwedge x. \vdash (\neg(f x) \vee \neg(g x)) = (\neg((f x) \wedge (g x)))$
by *auto*
have 2: $\vdash (\exists \exists x. \neg(f x) \vee \neg(g x)) = (\exists \exists x. \neg((f x) \wedge (g x)))$
using 1 **by** (*simp add: EExEqvRule*)
have 3: $\vdash (\exists \exists fa. \neg(f fa \wedge g fa)) = (\exists \exists fa. \neg f fa \vee \neg g fa)$
using 2 **by** *auto*
have 4: $\vdash (\neg(\neg(\exists \exists fa. \neg f fa) \wedge \neg(\exists \exists f. \neg g f))) = ((\exists \exists fa. \neg f fa) \vee (\exists \exists f. \neg g f))$
by *auto*
have $\vdash ((\exists \exists fa. \neg f fa) \vee (\exists \exists f. \neg g f)) = (\exists \exists fa. \neg f fa \vee \neg g fa)$
by (*simp add: EExOrDist inteq-reflection*)
then have 5: $\vdash (\neg(\exists \exists fa. \neg f fa) \wedge \neg(\exists \exists f. \neg g f)) = (\neg(\exists \exists fa. \neg f fa \vee \neg g fa))$
by *auto*
show *?thesis* **using** 3 5 **unfolding** *forall-state-d-def* **by** *fastforce*
qed

lemma *AAxAndImport*:
 $\vdash (g \wedge (\forall \forall x. f x)) = (\forall \forall x. g \wedge f x)$
proof –
have 1: $\vdash (\neg g \vee (\exists \exists x. \neg(f x))) = (\exists \exists x. \neg g \vee \neg(f x))$
by (*simp add: EExOrImport*)
have 2: $\vdash (\neg(\exists \exists x. \neg(f x))) = (\neg(\forall \forall x. f x))$
using *AAxDef* **by** *fastforce*
have 3: $\vdash (\neg g \vee (\exists \exists x. \neg(f x))) = (\neg(g \wedge (\forall \forall x. f x)))$
using 2 **by** *fastforce*
have 4: $\bigwedge x. \vdash (\neg g \vee \neg(f x)) = (\neg(g \wedge f x))$
by *auto*
have 5: $\vdash (\exists \exists x. \neg g \vee \neg(f x)) = (\exists \exists x. \neg(g \wedge f x))$
using 4 **by** (*simp add: EExEqvRule*)
have 6: $\vdash (\exists \exists x. \neg(g \wedge f x)) = (\neg(\forall \forall x. g \wedge f x))$

using *AAxDef* by *fastforce*
 have 7: $\vdash (\neg(g \wedge (\forall\forall x. f x))) = (\neg(\forall\forall x. g \wedge f x))$
 by (*metis 1 3 5 6 inteq-reflection*)
 from 7 show ?thesis by *fastforce*
 qed

lemma *AAxOrImport*:

$\vdash (g \vee (\forall\forall x. f x)) = (\forall\forall x. g \vee f x)$
proof –
 have 1: $\vdash (\neg g \wedge (\exists\exists x. \neg(f x))) = (\exists\exists x. \neg g \wedge \neg(f x))$ by (*simp add: EExAndImport*)
 have 2: $\vdash (\exists\exists x. \neg(f x)) = (\neg((\forall\forall x. f x)))$ using *AAxDef* by *fastforce*
 have 3: $\vdash (\neg g \wedge (\exists\exists x. \neg(f x))) = (\neg(g \vee (\forall\forall x. f x)))$ using 2 by *fastforce*
 have 4: $\bigwedge x. \vdash (\neg g \wedge \neg(f x)) = (\neg(g \vee f x))$ by *auto*
 have 5: $\vdash (\exists\exists x. \neg g \wedge \neg(f x)) = (\exists\exists x. \neg(g \vee f x))$ using 4 by (*simp add: EExEqvRule*)
 have 6: $\vdash (\exists\exists x. \neg(g \vee f x)) = (\neg(\forall\forall x. g \vee f x))$ using *AAxDef* by *fastforce*
 have 7: $\vdash (\neg(g \vee (\forall\forall x. f x))) = (\neg(\forall\forall x. g \vee f x))$ by (*metis 1 3 5 6 inteq-reflection*)
 from 7 show ?thesis by *auto*
 qed

lemma *EExImpChopRule*:

assumes $\vdash f x \longrightarrow g x$
 shows $\vdash (\exists\exists x. h;(f x) \longrightarrow h;(g x))$
 using *RightChopImpChop*[of $f x g x h$]
EExImpRule[of $\lambda x. \text{LIFT}(h;(f x)) x \lambda x. \text{LIFT}(h;(g x))$] *assms* by *auto*

lemma *EExChopRight*:

$\vdash (\exists\exists x. (f x);g) \longrightarrow (\exists\exists x. f x);g$
proof –
 have 1: $\bigwedge x. \vdash (f x);g \longrightarrow (\exists\exists x. f x);g$ by (*simp add: EExI LeftChopImpChop*)
 from 1 show ?thesis by (*simp add: EExE*)
 qed

lemma *EExChopRightNoDep*:

$\vdash (\exists\exists x. (f x);g) = (\exists\exists x. f x);g$
 by (*auto simp add: exist-state-d-def Valid-def itl-defs*)

lemma *EExChopLeft* :

$\vdash (\exists\exists x. g;(f x)) \longrightarrow g;(\exists\exists x. f x)$
proof –
 have 1: $\bigwedge x. \vdash g;(f x) \longrightarrow g;(\exists\exists x. f x)$ by (*simp add: EExI RightChopImpChop*)
 from 1 show ?thesis by (*simp add: EExE*)
 qed

lemma *EExChopLeftNoDep*:

$\vdash (\exists\exists x. g;(f x)) = g;(\exists\exists x. f x)$
 by (*auto simp add: exist-state-d-def Valid-def itl-defs*)

lemma *EExEExChopEqvEExEExChop*:

$\vdash (\exists\exists v. (\exists\exists y. (f v);(g y))) = (\exists\exists y. (\exists\exists v. (f v);(g y)))$
 by (*simp add: exist-state-d-def Valid-def itl-defs*) *blast*

lemma *EExEExChopEqvEExChopEExA*:
 $\vdash (\exists \exists v. (\exists \exists y. (f v); (g y))) = (\exists \exists v. (f v); (\exists \exists y. (g y)))$
by (*simp add: exist-state-d-def Valid-def itl-defs*) *blast*

lemma *EExEExChopEqvEExChopEExB*:
 $\vdash (\exists \exists y. (\exists \exists v. (f v); (g y))) = (\exists \exists y. (\exists \exists v. (f v)); (g y))$
by (*simp add: exist-state-d-def Valid-def itl-defs*) *blast*

lemma *EExEExChopEqvEExChopEExC*:
 $\vdash (\exists \exists v. (\exists \exists y. (f v); (g y))) = (\exists \exists v. (f v)); (\exists \exists y. (g y))$
by (*simp add: exist-state-d-def Valid-def itl-defs*) *blast*

lemma *EExChopDistA*:
 $\vdash (\exists \exists v. (f v); (g v)) \longrightarrow (\exists \exists v. (f v)) ; (\exists \exists v. (g v))$
by (*simp add: exist-state-d-def Valid-def itl-defs*)
blast

lemma *EExChopDistB*:
 $\vdash (\exists \exists v. (f v)) \text{ chop } (\text{skip chop } (\exists \exists v. (g v))) \longrightarrow$
 $(\exists \exists v v1. (f v) \text{ chop } (\text{skip chop } (g v1)))$
by (*auto simp add: exist-state-d-def Valid-def itl-defs*)

lemma *EExChopDistC*:
 $\vdash (\exists \exists v. (f v)) \text{ chop } (\exists \exists v. (g v)) \longrightarrow$
 $(\exists \exists v v1. (f v) \text{ chop } (g v1))$
by (*auto simp add: exist-state-d-def Valid-def itl-defs*)

lemma *ExLenOrInf*:
 $\vdash (\exists n. \text{len}(n)) \vee \text{inf}$
using *nfinite-nlength-enat* **by** (*auto simp add: Valid-def len-defs itl-defs*)

lemma *CSPowerChop*:
 $\vdash (f^*) = (\exists n. \text{fpower } (f \wedge \text{more}) n); (\text{empty} \vee (f \wedge \text{more}) \wedge \text{inf})$
by (*simp add: chopstar-d-def fpowerstar-d-def powerstar-d-def Valid-def*)

lemma *ExChopRightNoDep*:
 $\vdash (\exists x. (f x); g) = (\exists x. (f x)); g$
by (*auto simp add: Valid-def itl-defs*)

lemma *ExChopLeftNoDep*:
 $\vdash (\exists x. g; (f x)) = g; (\exists x. f x)$
by (*auto simp add: Valid-def itl-defs*)

lemma *ExExEqvExEx*:
 $\vdash (\exists x. (\exists y. (f x); (g y))) = (\exists y. (\exists x. (f x); (g y)))$
by (*auto simp add: Valid-def itl-defs*)

end

2.11 Time reversal operator

theory *TimeReversal*

imports

SChopTheorems FOTheorems Omega

begin

Time reversal operator is defined in [9]. Note: in this theory only finite intervals can be reversed.

2.11.1 Definition

definition *freverse-d* :: ('a::world) formula \Rightarrow 'a formula

where *freverse-d* $F \equiv \lambda s. \text{finite } s \wedge (\text{nrev } s \models F)$

definition *ereverse-d* :: ('a::world, 'b) formfun \Rightarrow ('a, 'b) formfun

where *ereverse-d* $f \equiv \lambda s. (\text{if finite } s \text{ then } (\text{nrev } s \models f) \text{ else } (\epsilon (x::'b). x=x))$

syntax

-freverse-d :: lift \Rightarrow lift $((-^r) [85] 85)$

-ereverse-d :: lift \Rightarrow lift $((-^R) [100] 99)$

syntax (*ASCII*)

-freverse-d :: lift \Rightarrow lift $((\text{frev } -) [85] 85)$

-ereverse-d :: lift \Rightarrow lift $((\text{erev } -) [100] 99)$

translations

-freverse-d $\equiv \text{CONST } \text{freverse-d}$

-ereverse-d $\equiv \text{CONST } \text{ereverse-d}$

definition *wfreverse-d* :: ('a::world) formula \Rightarrow 'a formula

where *wfreverse-d* $F \equiv \text{LIFT } (\neg((\neg F)^r))$

syntax

-wfreverse-d :: lift \Rightarrow lift $((\text{wrev } -) [85] 85)$

syntax (*ASCII*)

-wfreverse-d :: lift \Rightarrow lift $((\text{wrev } -) [85] 85)$

translations

-wfreverse-d $\equiv \text{CONST } \text{wfreverse-d}$

2.11.2 Finite theorems

lemma *FiniteChopEqv*:

assumes $\vdash \text{finite} \longrightarrow f = g$

shows $\vdash \text{finite} \longrightarrow f;h = g;h$

using *assms*

by (auto simp add: Valid-def itl-defs)

lemma *FiniteTransEqv*:

assumes $\vdash \text{finite} \longrightarrow f = g$

$\vdash \text{finite} \longrightarrow g = h$

shows $\vdash \text{finite} \longrightarrow f = h$

using *assms*

by (auto simp add: Valid-def itl-defs)

lemma *FiniteOrEqv*:

assumes $\vdash \text{finite} \longrightarrow f1 = f2$

$\vdash \text{finite} \longrightarrow g1 = g2$

shows $\vdash \text{finite} \longrightarrow (f1 \vee g1) = (f2 \vee g2)$

using *assms*

by (auto simp add: Valid-def itl-defs)

lemma *FiniteAndEqv*:

assumes $\vdash \text{finite} \longrightarrow f1 = f2$

$\vdash \text{finite} \longrightarrow g1 = g2$

shows $\vdash \text{finite} \longrightarrow (f1 \wedge g1) = (f2 \wedge g2)$

using *assms*

by (auto simp add: Valid-def itl-defs)

lemma *FiniteNotEqv*:

assumes $\vdash \text{finite} \longrightarrow f1 = f2$

shows $\vdash \text{finite} \longrightarrow (\neg f1) = (\neg f2)$

using *assms*

by (auto simp add: Valid-def itl-defs)

lemma *FiniteWNextEqv*:

assumes $\vdash \text{finite} \longrightarrow f = g$

shows $\vdash \text{finite} \longrightarrow \text{wnext } f = \text{wnext } g$

using *assms*

by (auto simp add: Valid-def itl-defs)

lemma *FiniteWPrevEqv*:

assumes $\vdash \text{finite} \longrightarrow f = g$

shows $\vdash \text{finite} \longrightarrow \text{wprev } f = \text{wprev } g$

using *assms*

by (auto simp add: Valid-def itl-defs)

lemma *FiniteInitonlyEqv*:

assumes $\vdash \text{finite} \longrightarrow f = g$

shows $\vdash \text{finite} \longrightarrow \text{initonly } f = \text{initonly } g$

using *assms*

by (auto simp add: Valid-def itl-defs)

lemma *FiniteKeepEqv*:
assumes $\vdash \text{finite} \longrightarrow f = g$
shows $\vdash \text{finite} \longrightarrow \text{keep } f = \text{keep } g$
using *assms*
by (*auto simp add: Valid-def keep-d-def itl-defs nsubn-nfinite*)

lemma *FiniteHaltEqv*:
assumes $\vdash \text{finite} \longrightarrow f = g$
shows $\vdash \text{finite} \longrightarrow \text{halt } f = \text{halt } g$
using *assms*
by (*auto simp add: Valid-def itl-defs*)

lemma *FiniteDaAlt*:
 $\vdash \text{finite} \longrightarrow \text{da } f = (\# \text{True}; f); \# \text{True}$
unfolding *da-d-def*
by (*metis ChopAssoc FiniteChopEqv int-simps(13) int-simps(9) inteq-reflection*)

lemma *ChopCsImpCSChop*:
 $\vdash (f \wedge \text{finite}); (\text{schopstar } f) \longrightarrow (\text{schopstar } f); (f \wedge \text{finite})$
by (*metis SChopstar-WPowerstar WPowerstar-slide-var int-iffD2 inteq-reflection*)

lemma *CSChopImpChopCS*:
 $\vdash (\text{schopstar } f); (f \wedge \text{finite}) \longrightarrow (f \wedge \text{finite}); (\text{schopstar } f)$
by (*metis SChopstar-WPowerstar WPowerstar-slide-var int-iffD1 inteq-reflection*)

lemma *NextDiamondEqvDiamondNext*:
 $\vdash \bigcirc (\Diamond f) = \Diamond (\bigcirc f)$
proof –
have *1*: $\vdash \text{finite}; \text{skip} = \text{skip}; \text{finite}$
by (*simp add: FiniteChopSkipEqvSkipChopFinite*)
hence *2*: $\vdash (\text{finite}; \text{skip}); f = (\text{skip}; \text{finite}); f$ **using** *LeftChopEqvChop* **by** *blast*
have *3*: $\vdash (\text{finite}; \text{skip}); f = \text{finite}; (\text{skip}; f)$
by (*meson ChopAssoc ChopEmpty Prop04*)
have *4*: $\vdash (\text{skip}; \text{finite}); f = \text{skip}; (\text{finite}; f)$
by (*meson ChopAssoc Prop11*)
from *2 3 4* **show** *?thesis* **by** (*metis int-eq next-d-def sometimes-d-def*)
qed

lemma *WeakNextBoxInduct*:
assumes $\vdash \text{wnext } (\Box f) \longrightarrow f$
shows $\vdash \text{finite} \longrightarrow f$
proof –
have *1*: $\vdash \text{wnext } (\Box f) \longrightarrow f$ **using** *assms* **by** *blast*
hence *2*: $\vdash \neg f \longrightarrow \neg (\text{wnext } (\Box f))$ **by** *fastforce*
hence *3*: $\vdash \neg f \longrightarrow \bigcirc (\neg (\Box f))$ **by** (*simp add: wnext-d-def*)
have *4*: $\vdash (\neg (\Box f)) = (\Diamond (\neg f))$ **by** (*auto simp: always-d-def*)

hence 5: $\vdash \bigcirc(\neg(\Box f)) = \bigcirc(\Diamond(\neg f))$ **using** *NextEqvNext* **by** *blast*
have 6: $\vdash \neg f \longrightarrow \bigcirc(\Diamond(\neg f))$ **using** 3 5 **by** *fastforce*
have 7: $\vdash \bigcirc(\Diamond(\neg f)) = \Diamond(\bigcirc(\neg f))$ **using** *NextDiamondEqvDiamondNext* **by** *blast*
have 8: $\vdash \neg f \longrightarrow \Diamond(\bigcirc(\neg f))$ **using** 6 7 **by** *fastforce*
have 9: $\vdash \Diamond(\neg f) \longrightarrow \Diamond(\Diamond(\bigcirc(\neg f)))$ **using** 8 *DiamondImpDiamond* **by** *blast*
have 10: $\vdash \Diamond(\Diamond(\bigcirc(\neg f))) = \Diamond(\bigcirc(\neg f))$ **using** *DiamondDiamondEqvDiamond* **by** *blast*
have 11: $\vdash \Diamond(\neg f) \longrightarrow \Diamond(\bigcirc(\neg f))$ **using** 9 10 **by** *fastforce*
have 12: $\vdash \Diamond(\neg f) \longrightarrow \bigcirc(\Diamond(\neg f))$ **using** 7 11 **by** *fastforce*
hence 13: $\vdash \text{finite} \longrightarrow \neg(\Diamond(\neg f))$ **using** *NextLoop* **by** *blast*
hence 14: $\vdash \text{finite} \longrightarrow \Box f$ **by** (*simp add: always-d-def*)
have 15: $\vdash \Box f \longrightarrow f$ **using** *BoxElim* **by** *blast*
from 14 15 **show** *?thesis* **by** (*metis lift-imp-trans*)
qed

2.11.3 Time reversal Rules

lemma *WRevRev*:

$\vdash (\text{wrev } f) = (\text{finite} \longrightarrow f^r)$
by (*simp add: Valid-def wfreverse-d-def freverse-d-def finite-defs nfinite-nrev*)

lemma *RevWRev*:

$\vdash (\text{frev } f) = (\text{finite} \wedge \text{wrev } f)$
by (*auto simp add: Valid-def wfreverse-d-def freverse-d-def finite-defs nfinite-nrev*)

lemma *RevWRevAlt*:

$\vdash (\text{frev } f) = (\neg((\text{wrev } (\neg f))))$
unfolding *wfreverse-d-def*
by *simp*

lemma *EExRev* :

$\vdash (\exists \exists x. F x)^r = (\exists \exists x. (F x)^r)$
by (*simp add: Valid-def exist-state-d-def freverse-d-def*)

lemma *EExWRev* :

$\vdash (\text{wrev } (\exists \exists x. F x)) = (\exists \exists x. (\text{wrev } (F x)))$
by (*simp add: Valid-def exist-state-d-def wfreverse-d-def freverse-d-def*)

lemma *rev-const* :

$\vdash \text{finite} \longrightarrow (\#c)^R = \#c$
by (*auto simp add: Valid-def itl-defs ereverse-d-def*)

lemma *rev-fun1* :

$\vdash \text{finite} \longrightarrow (f \langle x \rangle)^r = f \langle x^r \rangle$
by (*auto simp: itl-defs freverse-d-def*)

lemma *rev-fun1-alt* :

$\vdash (f \langle x \rangle)^r = (f \langle x^r \rangle \wedge \text{finite})$
by (*auto simp: itl-defs freverse-d-def*)

lemma *wrev-fun1* :

$\vdash \text{finite} \longrightarrow (\text{wrev } (f \langle x \rangle)) = f \langle (\text{wrev } x) \rangle$

by (*auto simp add: itl-defs wfreverse-d-def freverse-d-def*)

lemma *wrev-fun1-alt* :

$\vdash (\text{wrev } (f \langle x \rangle)) = (\text{finite} \longrightarrow f \langle (\text{wrev } x) \rangle)$

by (*auto simp add: itl-defs wfreverse-d-def freverse-d-def*)

lemma *rev-fun1-e* :

$\vdash \text{finite} \longrightarrow (f \langle x \rangle)^R = f \langle x^R \rangle$

by (*auto simp: itl-defs ereverse-d-def*)

lemma *rev-fun2*:

$\vdash \text{finite} \longrightarrow (f \langle x, y \rangle)^r = f \langle x^r, y^r \rangle$

by (*auto simp: itl-defs freverse-d-def*)

lemma *rev-fun2-alt*:

$\vdash (f \langle x, y \rangle)^r = (f \langle x^r, y^r \rangle \wedge \text{finite})$

by (*auto simp: itl-defs freverse-d-def*)

lemma *wrev-fun2*:

$\vdash \text{finite} \longrightarrow (\text{wrev } (f \langle x, y \rangle)) = f \langle (\text{wrev } x), (\text{wrev } y) \rangle$

by (*auto simp add: itl-defs wfreverse-d-def freverse-d-def*)

lemma *wrev-fun2-alt* :

$\vdash (\text{wrev } (f \langle x, y \rangle)) = (\text{finite} \longrightarrow f \langle (\text{wrev } x), (\text{wrev } y) \rangle)$

by (*auto simp add: itl-defs wfreverse-d-def freverse-d-def*)

lemma *rev-fun2-e*:

$\vdash \text{finite} \longrightarrow (f \langle x, y \rangle)^R = f \langle x^R, y^R \rangle$

by (*auto simp: itl-defs ereverse-d-def*)

lemma *rev-fun3*:

$\vdash \text{finite} \longrightarrow (f \langle x, y, z \rangle)^r = f \langle x^r, y^r, z^r \rangle$

by (*auto simp: itl-defs freverse-d-def*)

lemma *rev-fun3-alt*:

$\vdash (f \langle x, y, z \rangle)^r = (f \langle x^r, y^r, z^r \rangle \wedge \text{finite})$

by (*auto simp: itl-defs freverse-d-def*)

lemma *wrev-fun3*:

$\vdash \text{finite} \longrightarrow (\text{wrev } (f \langle x, y, z \rangle)) = f \langle (\text{wrev } x), (\text{wrev } y), (\text{wrev } z) \rangle$

by (*auto simp add: itl-defs wfreverse-d-def freverse-d-def*)

lemma *wrev-fun3-alt* :

$\vdash (\text{wrev } (f \langle x, y, z \rangle)) = (\text{finite} \longrightarrow f \langle (\text{wrev } x), (\text{wrev } y), (\text{wrev } z) \rangle)$

by (*auto simp add: itl-defs wfreverse-d-def freverse-d-def*)

lemma *rev-fun3-e*:

$\vdash \text{finite} \longrightarrow (f \langle x, y, z \rangle)^R = f \langle x^R, y^R, z^R \rangle$

by (*auto simp: itl-defs ereverse-d-def*)

lemma *rev-forall*:

$\vdash (\forall x. P x)^r = (\forall x. (P x)^r)$

by (*auto simp: freverse-d-def*)

lemma *wrev-forall*:

$\vdash (wrev (\forall x. P x)) = (\forall x. (wrev (P x)))$

by (*auto simp: wfreverse-d-def freverse-d-def*)

lemma *rev-exists*:

$\vdash (\exists x. P x)^r = (\exists x. (P x)^r)$

by (*auto simp: freverse-d-def*)

lemma *wrev-exists*:

$\vdash (wrev (\exists x. P x)) = (\exists x. (wrev (P x)))$

by (*auto simp: wfreverse-d-def freverse-d-def*)

lemma *rev-exists1*:

$\vdash (\exists! x. P x)^r = (\exists! x. (P x)^r)$

by (*auto simp: freverse-d-def*)

lemma *wrev-exists1*:

$\vdash wrev (\exists! x. P x) = (finite \longrightarrow (\exists! x. wrev (P x)))$

proof –

have 1: $\vdash wrev (\exists! x. P x) = (finite \longrightarrow frev (\exists! x. P x))$

using *WRevRev* **by** *blast*

have 2: $\vdash (finite \longrightarrow frev (\exists! x. P x)) = (finite \longrightarrow (\exists! x. frev (P x)))$

by (*metis int-simps(27) inteq-reflection rev-exists1*)

have 3: $\bigwedge x. \vdash frev (P x) = (wrev (P x) \wedge finite)$

by (*metis RevWRev inteq-reflection lift-and-com*)

have 4: $\vdash ((\exists! x. frev (P x))) = (\exists! x. wrev (P x) \wedge finite)$

using 3 **by** (*simp add: Valid-def freverse-d-def itl-defs*)

have 5: $\vdash (\exists! x. wrev (P x) \wedge finite) = ((\exists! x. wrev (P x)) \wedge finite)$

by *auto*

have 6: $\vdash (finite \longrightarrow ((\exists! x. wrev (P x)) \wedge finite)) = (finite \longrightarrow (\exists! x. wrev (P x)))$

by *force*

show *?thesis*

by (*metis 1 2 4 5 6 inteq-reflection*)

qed

lemma *rev-current*:

$\vdash finite \longrightarrow (\$v)^R = (!v)$

by (*auto simp: nnth-nrev itl-def ereverse-d-def nfirst-nrev*)

lemma *rev-next*:

$\vdash finite \longrightarrow (v\$)^R = (v!)$

by (*auto simp: itl-defs ereverse-d-def nfinite-nrev*)
 (*metis One-nat-def enat-ord-simps(2) epred-conv-minus idiff-enat-enat less-Suc0 linorder-not-less*
nfinite-nlength-enat nnth-nrev one-enat-def the-enat.simps)

lemma *rev-penult*:

$\vdash \text{finite} \longrightarrow (v!)^R = (v\$)$

by (*auto simp: itl-defs ereverse-d-def nfinite-nrev*)

(*metis One-nat-def Suc-ile-eq epred-enat i0-less ndropn-nfirst nfinite-ndropn*
nfinite-nlength-enat nlast-nrev nrev-ndropn ntaken-nlast the-enat.simps zero-enat-def)

lemma *rev-fin*:

$\vdash \text{finite} \longrightarrow (!v)^R = (\$v)$

by (*auto simp: itl-defs ereverse-d-def nfinite-nrev nlast-nrev*)

lemma *EqvReverseReverse*:

$\vdash \text{finite} \longrightarrow (f^r)^r = f$

by (*simp add: Valid-def itl-defs freverse-d-def nfinite-nrev nrev-nrev*)

lemma *EqvWReverseWReverse*:

$\vdash \text{finite} \longrightarrow (\text{wrev} (\text{wrev } f)) = f$

by (*simp add: Valid-def itl-defs wfreverse-d-def freverse-d-def nfinite-nrev nrev-nrev*)

lemma *EqvReverseReverseAlt*:

$\vdash (f^r)^r = (f \wedge \text{finite})$

by (*auto simp add: Valid-def itl-defs freverse-d-def nfinite-nrev nrev-nrev*)

lemma *EqvWReverseWReverseAlt*:

$\vdash (\text{wrev} (\text{wrev } f)) = (\text{finite} \longrightarrow f)$

by (*simp add: Valid-def itl-defs wfreverse-d-def freverse-d-def nfinite-nrev nrev-nrev*)

lemma *ReverseEqv*:

$(\vdash \text{finite} \longrightarrow f) \longleftrightarrow (\vdash \text{finite} \longrightarrow f^r)$

by (*simp add: Valid-def itl-defs freverse-d-def*)
 (*metis nfinite-nrev nrev-nrev*)

lemma *WReverseEqv*:

$(\vdash \text{finite} \longrightarrow f) \longleftrightarrow (\vdash \text{finite} \longrightarrow \text{wrev } f)$

by (*simp add: Valid-def itl-defs wfreverse-d-def freverse-d-def*)
 (*metis nfinite-nrev nrev-nrev*)

lemma *ReverseEqvAlt*:

$(\vdash f \wedge \text{finite}) \longleftrightarrow (\vdash f^r)$

by (*auto simp add: Valid-def itl-defs freverse-d-def*)

lemma *WReverseEqvAlt*:

$(\vdash \text{finite} \longrightarrow f) \longleftrightarrow (\vdash \text{wrev } f)$

by (*simp add: Valid-def itl-defs wfreverse-d-def freverse-d-def*)
 (*metis nfinite-nrev nrev-nrev*)

lemma *RevSkip*:

$\vdash \text{skip}^r = \text{skip}$
using *nlength-eq-enat-nfiniteD* **by** (*simp add: Valid-def freverse-d-def itl-defs, blast*)

lemma *WRevSkip*:
 $\vdash (\text{wrev skip}) = (\text{inf} \vee \text{skip})$
by (*auto simp add: Valid-def wfreverse-d-def freverse-d-def itl-defs*)

lemma *RevFinite*:
 $\vdash \text{finite}^r = \text{finite}$
by (*simp add: Valid-def freverse-d-def finite-defs nfinite-nrev*)

lemma *WRevFinite*:
 $\vdash (\text{wrev finite})$
by (*auto simp add: Valid-def wfreverse-d-def freverse-d-def itl-defs nfinite-nrev*)

lemma *NotRevInfinite*:
 $\vdash \neg (\text{inf}^r)$
by (*simp add: Valid-def freverse-d-def infinite-defs nfinite-nrev*)

lemma *WRevInfinite*:
 $\vdash (\text{wrev inf}) = \text{inf}$
by (*simp add: Valid-def wfreverse-d-def freverse-d-def infinite-defs nfinite-nrev*)

lemma *RevChop*:
 $\vdash \text{finite} \longrightarrow (f;g)^r = (g^r;f^r)$
by (*simp add: Valid-def itl-defs freverse-d-def nfinite-nrev*)
(metis diff-diff-cancel diff-le-self enat-ord-simps(1) nfinite-nlength-enat nrev-ndropn nrev-ntaken the-enat.simps)

lemma *RevYields*:
 $\vdash \text{finite} \longrightarrow (f \text{ yields } g)^r = ((g^r) \text{ revyields } f^r)$
unfolding *yields-d-def revyields-d-def*
by (*meson FiniteChopEqv FiniteNotEqv FiniteTransEqv RevChop rev-fun1*)

lemma *WRevChop*:
 $\vdash \text{finite} \longrightarrow (\text{wrev } (f;g)) = ((\text{wrev } g);(\text{wrev } f))$
by (*simp add: Valid-def itl-defs wfreverse-d-def freverse-d-def nfinite-nrev*)
(metis diff-diff-cancel diff-le-self enat-ord-simps(1) nfinite-nlength-enat nrev-ndropn nrev-ntaken the-enat.simps)

lemma *RevChopAlt*:
 $\vdash (f;g)^r = (g^r;f^r)$
by (*simp add: Valid-def itl-defs freverse-d-def nfinite-nrev*)
(metis diff-diff-cancel diff-le-self enat-ord-simps(1) nfinite-nlength-enat nrev-ndropn nrev-ntaken the-enat.simps)

lemma *RevSChopAlt*:
 $\vdash (f \frown g)^r = (g^r \frown f^r)$
by (*metis (no-types, opaque-lifting) EquReverseReverseAlt RevChopAlt int-simps(27)*)

inteq-reflection rev-fun2-alt chop-d-def)

lemma *WRevChopAlt:*

$\vdash (wrev (f;g)) = ((wrev g);(wrev f))$

by (*simp add: Valid-def itl-defs wreverse-d-def freverse-d-def nfinite-nrev*)
(metis diff-diff-cancel diff-le-self enat-ord-simps(1) nfinite-nlength-enat nrev-ndropn
nrev-ntaken the-enat.simps)

lemma *WRevSChopAlt:*

$\vdash (wrev (f \frown g)) = ((wrev g);(wrev f))$

unfolding *chop-d-def*

by (*metis (no-types, opaque-lifting) EquReverseReverseAlt EquWReverseWReverse FiniteImpAnd*
WRevChopAlt int-simps(4) inteq-reflection rev-fun1-alt wreverse-d-def)

lemma *RAnd:*

$\vdash (f \wedge g)^r = (f^r \wedge g^r)$

by (*auto simp add: Valid-def freverse-d-def*)

lemma *WRAnd:*

$\vdash (wrev (f \wedge g)) = (wrev f \wedge wrev g)$

by (*auto simp add: Valid-def wreverse-d-def freverse-d-def*)

lemma *RevAndFinite:*

$\vdash (f^r \wedge finite) = (f \wedge finite)^r$

by (*metis RAnd RevFinite inteq-reflection*)

lemma *RevAndFiniteAlt:*

$\vdash (f^r \wedge finite) = (f)^r$

by (*metis int-simps(20) inteq-reflection rev-fun2-alt*)

lemma *WRevAndFinite:*

$\vdash (wrev f) = (wrev (f \wedge finite))$

proof –

have 1: $\vdash (wrev (f \wedge finite)) = ((wrev f) \wedge wrev finite)$

by (*simp add: WRAnd*)

have 2: $\vdash ((wrev f) \wedge wrev finite) = (wrev f)$

using *RevFinite WRevRev* **by** *fastforce*

show *?thesis*

using 1 2 **by** *fastforce*

qed

lemma *WRevAndFiniteAlt:*

$\vdash ((wrev f) \wedge finite) = (frev f)$

by (*metis int-simps(4) inteq-reflection rev-fun1-alt wreverse-d-def*)

lemma *WRevFiniteImp:*

$\vdash (finite \longrightarrow wrev f) = wrev f$

by (*metis FiniteImp RevFinite WRevAndFiniteAlt WRevRev inteq-reflection*)

lemma *RMoreEqvMore*:

$\vdash \text{finite} \longrightarrow \text{more}^r = \text{more}$

by (*simp add: Valid-def itl-defs freverse-d-def*)

lemma *WRMoreEqvMore*:

$\vdash \text{finite} \longrightarrow \text{wrev more} = \text{more}$

by (*simp add: Valid-def itl-defs wfreverse-d-def freverse-d-def*)

lemma *RMoreEqvMoreAlt*:

$\vdash \text{more}^r = (\text{more} \wedge \text{finite})$

by (*auto simp add: Valid-def itl-defs freverse-d-def*)

lemma *WRMoreEqvMoreAlt*:

$\vdash \text{wrev more} = \text{more}$

by (*auto simp add: Valid-def itl-defs wfreverse-d-def freverse-d-def
nfinite-conv-nlength-enat zero-enat-def*)

lemma *REmptyEqvEmpty*:

$\vdash \text{empty}^r = \text{empty}$

by (*auto simp add: Valid-def itl-defs freverse-d-def nlength-eq-enat-nfiniteD zero-enat-def*)

lemma *WREmptyEqvEmpty*:

$\vdash \text{wrev empty} = (\text{finite} \longrightarrow \text{empty})$

by (*auto simp add: Valid-def itl-defs wfreverse-d-def freverse-d-def
nlength-eq-enat-nfiniteD zero-enat-def*)

lemma *RImpRule*:

assumes $\vdash f \longrightarrow g$

shows $\vdash (f^r) \longrightarrow (g^r)$

using *assms*

by (*auto simp add: Valid-def itl-defs freverse-d-def*)

lemma *WRImpRule*:

assumes $\vdash f \longrightarrow g$

shows $\vdash (\text{wrev } f) \longrightarrow (\text{wrev } g)$

using *assms*

by (*auto simp add: Valid-def itl-defs wfreverse-d-def freverse-d-def*)

lemma *REqvRule*:

assumes $\vdash f = g$

shows $\vdash (f^r) = (g^r)$

using *assms*

using *inteq-reflection* **by** *force*

lemma *WREqvRule*:

assumes $\vdash f = g$
shows $\vdash (\text{wrev } f) = (\text{wrev } g)$
using *assms*
by (*simp add: Prop11 WRImpRule*)

lemma *FPowerFinite*:

$\vdash (\text{fpower } (f \wedge \text{more}) \ n) = ((\text{fpower } (f \wedge \text{more}) \ n) \wedge \text{finite})$

proof (*induct n*)

case 0

then show ?*case*

by (*metis EmptyImpFinite Prop10 fpower-d-def wpow-0*)

next

case (*Suc n*)

then show ?*case*

proof –

have 1: $\vdash \text{fpower } (f \wedge \text{more}) \ (\text{Suc } n) = (f \wedge \text{fmore}); (\text{fpower } (f \wedge \text{more}) \ n)$

by (*metis AndMoreSChopEqvAndFmoreChop fpower-d-def schop-d-def wpow-Suc*)

have 2: $\vdash (f \wedge \text{fmore}); (\text{fpower } (f \wedge \text{more}) \ n) =$
 $((f \wedge \text{fmore}); (\text{fpower } (f \wedge \text{more}) \ n) \wedge \text{finite})$

by (*metis ChopImpChop FiniteChopFiniteEqvFinite Prop01 Prop05 Prop10 Suc fmore-d-def int-eq int-iffD1*)

show ?*thesis*

by (*metis 1 2 inteq-reflection*)

qed

qed

lemma *RevFPowerChop*:

$\vdash (\text{fpower } (f \wedge \text{more}) \ n)^r = (\text{fpower } ((f \wedge \text{more})^r) \ n)$

proof

(*induct n*)

case 0

then show ?*case* **using** *REmptyEqvEmpty*

by (*metis fpower-d-def wpow-0*)

next

case (*Suc n*)

then show ?*case*

proof –

have 1: $\vdash (\text{fpower } (f \wedge \text{more}) \ (\text{Suc } n))^r = ((f \wedge \text{fmore}); (\text{fpower } (f \wedge \text{more}) \ n))^r$

by (*metis AndMoreSChopEqvAndFmoreChop ChopEmpty fpower-d-def inteq-reflection schop-d-def wpow-Suc*)

have 2: $\vdash ((f \wedge \text{fmore}); (\text{fpower } (f \wedge \text{more}) \ n))^r = ((\text{fpower } (f \wedge \text{more}) \ n); (f \wedge \text{fmore}))^r$

by (*metis (no-types, lifting) AndMoreAndFiniteEqvAndFmore FPowerCommute REqvRule inteq-reflection*)

have 3: $\vdash (\text{fpower } (f \wedge \text{more}) \ n) = ((\text{fpower } (f \wedge \text{more}) \ n) \wedge \text{finite})$

by (*simp add: FPowerFinite*)

have 4: $\vdash ((\text{fpower } (f \wedge \text{more}) \ n); (f \wedge \text{fmore}))^r = ((f \wedge \text{fmore})^r; (\text{fpower } (f \wedge \text{more}) \ n)^r)$

by (*metis RevChopAlt*)

show ?*thesis*

by (*metis 2 4 AndMoreAndFiniteEqvAndFmore RevAndFinite Suc fpower-d-def inteq-reflection wpow-Suc*)

qed
qed

lemma *FPowerWRevFiniteAbsorb*:

$\vdash (\text{fpower } (\text{finite} \wedge \text{wrev } (f \wedge \text{more})) \text{ } n) = (\text{fpower } (\text{wrev } (f \wedge \text{more})) \text{ } n)$

proof (*induct n*)

case 0

then show ?case

by (*metis REmptyEqvEmpty fpower-d-def inteq-reflection wpow-0*)

next

case (*Suc n*)

then show ?case

by (*metis (no-types, opaque-lifting) EqvReverseReverseAlt EqvWReverseWReverseAlt RevWRev WRevAndFinite*

WRevRev fpower-d-def int-eq int-simps(27))

qed

lemma *WRevFPowerChop*:

$\vdash \text{wrev } (\text{fpower } (f \wedge \text{more}) \text{ } n) = (\text{finite} \longrightarrow \text{fpower } (\text{wrev } (f \wedge \text{more})) \text{ } n)$

proof –

have 1: $\vdash \text{wrev } (\text{fpower } (f \wedge \text{more}) \text{ } n) = (\text{finite} \longrightarrow \text{frev } (\text{fpower } (f \wedge \text{more}) \text{ } n))$

by (*simp add: WRevRev*)

have 2: $\vdash (\text{finite} \longrightarrow \text{frev } (\text{fpower } (f \wedge \text{more}) \text{ } n)) =$
 $(\text{finite} \longrightarrow (\text{fpower } ((f \wedge \text{more})^r) \text{ } n))$

using *RevFPowerChop* **by** *fastforce*

have 3: $\vdash (\text{fpower } ((f \wedge \text{more})^r) \text{ } n) = (\text{fpower } (\text{finite} \wedge \text{wrev } (f \wedge \text{more})) \text{ } n)$

by (*metis RevFPowerChop RevWRev inteq-reflection*)

have 4: $\vdash (\text{fpower } (\text{finite} \wedge \text{wrev } (f \wedge \text{more})) \text{ } n) = (\text{fpower } (\text{wrev } (f \wedge \text{more})) \text{ } n)$

by (*simp add: FPowerWRevFiniteAbsorb*)

show ?thesis

using 1 2 3 4 **by** *fastforce*

qed

lemma *RevChopstar*:

$\vdash (\text{schopstar } f)^r = (\text{schopstar } (f^r))$

proof –

have 1: $\vdash (\text{schopstar } f) = (\exists n. \text{fpower } (f \wedge \text{more}) \text{ } n)$

by (*simp add: FPowerstardef chopstar-d-def*)

have 2: $\vdash (\text{schopstar } f)^r = (\exists n. \text{fpower } (f \wedge \text{more}) \text{ } n)^r$

using *REqvRule 1* **by** *blast*

have 3: $\vdash (\exists n. \text{fpower } (f \wedge \text{more}) \text{ } n)^r = (\exists n. (\text{fpower } (f \wedge \text{more}) \text{ } n)^r)$

by (*simp add: rev-exists*)

have 4: $\vdash (\exists n. (\text{fpower } (f \wedge \text{more}) \text{ } n)^r) = (\exists n. (\text{fpower } ((f \wedge \text{more})^r) \text{ } n))$

by (*simp add: RevFPowerChop ExEqvRule*)

have 5: $\vdash (f \wedge \text{fmore})^r = (f^r \wedge \text{fmore})$

by (*metis (no-types, opaque-lifting) FiniteImpAnd RAnd RMoreEqvMore RevAndFinite*
inteq-reflection itl-def(26))

hence 6: $\vdash (\exists n. (\text{fpower } ((f \wedge \text{more})^r) \text{ } n)) = (\exists n. (\text{fpower } ((f^r \wedge \text{more})) \text{ } n))$

by (*metis AndMoreAndFiniteEqvAndFmore FPowerstar-WPowerstar FPowerstardef RAnd RMoreEqvMo-*

reAlt

fmore-d-def inteq-reflection rev-fun2-alt
have 7: $\vdash (\exists n. (fpower ((f^r \wedge more)) n)) = schopstar (f^r)$
by (*simp add: fpowerstar-d-def schopstar-d-def*)
from 2 3 4 6 7 **show** ?thesis **by** fastforce
qed

lemma *WRevAndFMore:*

$\vdash finite \longrightarrow wrev (f \wedge fmore) = (wrev f \wedge fmore)$
proof –
have 1: $\vdash finite \longrightarrow wrev (f \wedge fmore) = (f \wedge fmore)^r$
using *WRevAndFiniteAlt[of LIFT (f \wedge fmore)]* **by** auto
have 2: $\vdash finite \longrightarrow (f \wedge fmore)^r = (f^r \wedge fmore)$
by (*metis EqvReverseReverseAlt RMoreEqvMoreAlt fmore-d-def inteq-reflection rev-fun2*)
have 3: $\vdash finite \longrightarrow (f^r \wedge fmore) = (wrev f \wedge fmore)$
by (*metis (no-types, lifting) 2 AndMoreAndFiniteEqvAndFmore EqvReverseReverseAlt RAnd RMoreEqvMoreAlt WRevAndFiniteAlt fmore-d-def inteq-reflection*)
show ?thesis
by (*metis (no-types, lifting) 1 AndMoreAndFiniteEqvAndFmore WRAnd WRMoreEqvMoreAlt WRevAndFinite*
nite
WRevAndFiniteAlt inteq-reflection)
qed

lemma *WRevChopstar:*

$\vdash wrev (s chopstar f) = (finite \longrightarrow schopstar (wrev f))$
proof –
have 1: $\vdash (s chopstar f) = (\exists n. fpower (f \wedge more) n)$
by (*simp add: FPowerstardef schopstar-d-def*)
have 2: $\vdash wrev (s chopstar f) = wrev (\exists n. fpower (f \wedge more) n)$
using *WREqvRule 1* **by** blast
have 3: $\vdash wrev (\exists n. fpower (f \wedge more) n) = (\exists n. wrev (fpower (f \wedge more) n))$
by (*simp add: wrev-exists*)
have 4: $\vdash (\exists n. wrev (fpower (f \wedge more) n)) = (\exists n. (finite \longrightarrow fpower (wrev (f \wedge more)) n))$
by (*simp add: WRevFPowerChop ExEqvRule*)
have 5: $\vdash finite \longrightarrow wrev (f \wedge fmore) = (wrev f \wedge fmore)$
by (*simp add: WRevAndFMore*)
hence 6: $\vdash (\exists n. (finite \longrightarrow fpower (wrev (f \wedge more)) n)) =$
 $(\exists n. (finite \longrightarrow fpower ((wrev f \wedge more)) n))$
by (*metis (no-types, lifting) ExEqvRule WRAnd WRMoreEqvMoreAlt WRevFPowerChop inteq-reflection*)
have 65: $\vdash (\exists n. (finite \longrightarrow fpower ((wrev f \wedge more)) n)) =$
 $(finite \longrightarrow (\exists n. (fpower ((wrev f \wedge more)) n)))$
by auto
have 7: $\vdash (finite \longrightarrow (\exists n. (fpower ((wrev f \wedge more)) n))) =$
 $(finite \longrightarrow schopstar (wrev f))$
by (*simp add: fpowerstar-d-def schopstar-d-def*)
from 2 3 4 6 7 **show** ?thesis
by (*metis 65 inteq-reflection*)
qed

lemma *RevLen:*

$\vdash (\text{len } k)^r = \text{len } k$
by (*auto simp add: Valid-def itl-defs freverse-d-def len-defs nlength-eq-enat-nfiniteD*)

lemma *WRevLen*:
 $\vdash \text{wrev } (\text{len } k) = (\text{finite} \longrightarrow \text{len } k)$
by (*auto simp add: Valid-def itl-defs wfreverse-d-def freverse-d-def len-defs nlength-eq-enat-nfiniteD*)

lemma *NotRevOmega*:
 $\vdash \neg \text{frev } (\text{omega } f)$
by (*meson MP NotRevInfinite RImpRule lift-imp-neg omega-imp-inf*)

lemma *WRevOmega*:
 $\vdash \text{wrev } (\text{omega } f) = \text{inf}$
proof –
have 1: $\vdash \text{wrev } (\text{omega } f) \longrightarrow \text{inf}$
by (*metis WRImpRule WRevInfinite inteq-reflection omega-imp-inf*)
have 2: $\vdash \text{inf} \longrightarrow \text{wrev } (\text{omega } f)$
by (*simp add: freverse-d-def intI itl-defs(4) wfreverse-d-def*)
show ?thesis
by (*simp add: 1 2 Prop11*)
qed

lemmas *all-rev = rev-const rev-fun1 rev-fun2 rev-fun3 rev-fun1-alt rev-fun2-alt rev-fun3-alt*
rev-forall rev-exists rev-exists1 rev-current rev-next rev-penult rev-fin RevSkip
RevChop RevChopAlt RevChopstar RevFinite RevAndFinite RevLen

lemmas *all-wrev = wrev-fun1 wrev-fun2 wrev-fun3 wrev-fun1-alt wrev-fun2-alt wrev-fun3-alt*
wrev-forall wrev-exists wrev-exists1 WRevSkip
WRevChop WRevChopAlt WRevChopstar WRevFinite WRevAndFinite WRevLen

lemmas *all-rev-b = rev-fun1-alt rev-fun2-alt rev-fun3-alt rev-forall rev-exists RevChopAlt*
rev-exists1 RevSkip RevChopstar RevFinite RevAndFinite RevLen

lemmas *all-wrev-b = wrev-fun1-alt wrev-fun2-alt wrev-fun3-alt*
wrev-forall wrev-exists WRevSkip WRevChopAlt WRevChopstar WRevAndFinite WRevLen

lemmas *all-rev-unl = all-rev[THEN intD]*

lemmas *all-wrev-unl = all-wrev[THEN intD]*

lemmas *all-rev-eq = all-rev-b[THEN inteq-reflection]*

lemmas *all-wrev-eq = all-wrev-b[THEN inteq-reflection]*

2.11.4 Properties of Time Reversal

lemma *NotRev*:
 $\vdash (\neg(f^r)) = (\text{wrev } (\neg f))$

by (*simp add: wfreverse-d-def*)

lemma *NotWRev*:

$\vdash (\neg(wrev\ f)) = (\neg f)^r$

by (*simp add: wfreverse-d-def*)

lemma *RNot*:

$\vdash finite \longrightarrow (\neg f)^r = (\neg f^r)$

by (*simp add: rev-fun1*)

lemma *RNotAlt*:

$\vdash (\neg f)^r = ((\neg f^r) \wedge finite)$

by (*simp add: rev-fun1-alt*)

lemma *WRNot*:

$\vdash finite \longrightarrow wrev(\neg f) = (\neg(wrev\ f))$

by (*simp add: wrev-fun1*)

lemma *WRNotAlt*:

$\vdash wrev(\neg f) = (finite \longrightarrow \neg(wrev\ f))$

by (*simp add: wrev-fun1-alt*)

lemma *RRNot*:

$\vdash finite \longrightarrow (\neg(f^r))^r = (\neg f)$

by (*simp add: freverse-d-def intI itl-defs(5) nfinite-nrev nrev-nrev*)

lemma *WRWRNot*:

$\vdash finite \longrightarrow wrev(\neg(wrev\ f)) = (\neg f)$

by (*metis (no-types, lifting) EquWReverseWReverse EquWReverseWReverseAlt WRevAndFiniteAlt WRevRev all-wrev-eq(1) all-wrev-eq(9) inteq-reflection*)

lemma *RRNotAlt*:

$\vdash (\neg(f^r))^r = (\neg f \wedge finite)$

by (*auto simp add: Valid-def freverse-d-def intI itl-defs(5) nfinite-nrev nrev-nrev*)

lemma *WRWRNotAlt*:

$\vdash wrev(\neg(wrev\ f)) = (finite \longrightarrow \neg f)$

by (*metis EquReverseReverseAlt EquWReverseWReverseAlt WRevRev all-wrev-eq(9) int-simps(4) inteq-reflection wfreverse-d-def*)

lemma *RTrue*:

$\vdash finite \longrightarrow (\#True)^r = \#True$

by (*metis int-simps(1) inteq-reflection rev-fun2*)

lemma *WRTrue*:

$\vdash finite \longrightarrow wrev(\#True) = \#True$

by (*metis int-simps(1) inteq-reflection wrev-fun2*)

lemma *RTrueAlt*:

$\vdash (\#True)^r = \text{finite}$
by (*simp add: freverse-d-def intI itl-defs(5)*)

lemma *WRTrueAlt*:
 $\vdash \text{wrev } (\#True)$
by (*meson Prop11 Prop12 WReverseEqvAlt int-simps(17)*)

lemma *ROr*:
 $\vdash \text{finite} \longrightarrow (f \vee g)^r = (f^r \vee g^r)$
by (*simp add: rev-fun2*)

lemma *WROr*:
 $\vdash \text{finite} \longrightarrow \text{wrev } (f \vee g) = (\text{wrev } f \vee \text{wrev } g)$
by (*simp add: wrev-fun2*)

lemma *ROrAlt*:
 $\vdash (f \vee g)^r = (f^r \vee g^r)$
by (*auto simp add: Valid-def freverse-d-def*)

lemma *WROrAlt*:
 $\vdash \text{wrev } (f \vee g) = (\text{wrev } f \vee \text{wrev } g)$
by (*auto simp add: Valid-def wfreverse-d-def freverse-d-def*)

lemma *RROr*:
 $\vdash \text{finite} \longrightarrow (f^r \vee g^r)^r = (f \vee g)$
proof –
have 1: $\vdash \text{finite} \longrightarrow (f^r \vee g^r)^r = ((f^r)^r \vee (g^r)^r)$ **using** *ROr* **by** *blast*
have 2: $\vdash \text{finite} \longrightarrow ((f^r)^r \vee (g^r)^r) = (f \vee g)$ **using** *EqvReverseReverse*
by (*simp add: freverse-d-def intI itl-defs(5) nfinite-nrev nrev-nrev*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *WRWROr*:
 $\vdash \text{finite} \longrightarrow \text{wrev } (\text{wrev } f \vee \text{wrev } g) = (f \vee g)$
by (*metis EqvWReverseWReverse WROrAlt inteq-reflection*)

lemma *RROrAlt*:
 $\vdash (f^r \vee g^r)^r = ((f \vee g) \wedge \text{finite})$
by (*auto simp add: Valid-def freverse-d-def itl-defs nfinite-nrev nrev-nrev*)

lemma *WRWROrAlt*:
 $\vdash \text{wrev } (\text{wrev } f \vee \text{wrev } g) = (\text{finite} \longrightarrow f \vee g)$
by (*metis EqvWReverseWReverseAlt WROrAlt inteq-reflection*)

lemma *RRAnd*:
 $\vdash \text{finite} \longrightarrow (f^r \wedge g^r)^r = (f \wedge g)$
proof –

have 1: $\vdash (f^r \wedge g^r)^r = ((f^r)^r \wedge (g^r)^r)$ **using** *RAnd* **by** *blast*
have 2: $\vdash \text{finite} \longrightarrow ((f^r)^r \wedge (g^r)^r) = (f \wedge g)$
by (*metis* *EqvReverseReverse* *RAnd* *inteq-reflection*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *WRWRAnd*:

$\vdash \text{finite} \longrightarrow \text{wrev} (\text{wrev } f \wedge \text{wrev } g) = (f \wedge g)$
by (*metis* *EqvWReverseWReverse* *WRAnd* *inteq-reflection*)

lemma *RRAndAlt*:

$\vdash (f^r \wedge g^r)^r = (f \wedge g \wedge \text{finite})$
by (*auto simp add: Valid-def freverse-d-def itl-defs nfinite-nrev nrev-nrev*)

lemma *WRWRAndAlt*:

$\vdash \text{wrev} (\text{wrev } f \wedge \text{wrev } g) = (\text{finite} \longrightarrow (f \wedge g))$
by (*metis* *EqvWReverseWReverseAlt* *WRAnd* *inteq-reflection*)

lemma *RAndEmptyEqvAndEmpty*:

$\vdash (f \wedge \text{empty})^r = (f \wedge \text{empty})$
by (*simp add: Valid-def itl-defs freverse-d-def*)
(*metis* *ndropn-0 ndropn-nlast nlength-eq-enat-nfiniteD nrev-NNil the-enat-0 zero-enat-def*)

lemma *WRAndEmptyEqvAndEmpty*:

$\vdash \text{wrev} (f \wedge \text{empty}) = (\text{finite} \longrightarrow (f \wedge \text{empty}))$
by (*simp add: Valid-def itl-defs wfreverse-d-def freverse-d-def*)
(*metis* *ndropn-0 ndropn-nlast nrev-NNil the-enat-0*)

lemma *RNextEqvPrev*:

$\vdash \text{finite} \longrightarrow (\odot f)^r = \text{prev } (f^r)$
by (*metis* *RevChop* *RevSkip* *inteq-reflection* *next-d-def* *prev-d-def*)

lemma *WRNextEqvPrev*:

$\vdash \text{finite} \longrightarrow \text{wrev} (\odot f) = \text{prev} (\text{wrev } f)$

proof –

have 3: $\vdash \text{finite} \longrightarrow \text{wrev } f ; (\text{finite} \longrightarrow \text{skip}) = \text{wrev } f ; (\text{skip})$
by (*metis* *EqvWReverseWReverse* *EqvWReverseWReverseAlt* *FiniteRightChopEqvChop* *inteq-reflection*)

show *?thesis*

by (*metis* 3 *WRevRev* *all-rev-eq(8)* *all-wrev-eq(7)* *inteq-reflection* *next-d-def* *prev-d-def*)
qed

lemma *RNextEqvPrevAlt*:

$\vdash (\odot f)^r = \text{prev } (f^r)$
by (*metis* *RevChopAlt* *all-rev-eq(8)* *next-d-def* *prev-d-def*)

lemma *WRNextEqvPrevAlt*:

$\vdash \text{wrev} (\odot f) = (\text{finite} \longrightarrow \text{prev} (\text{wrev } f))$
by (*metis* (*no-types, lifting*) *EqvWReverseWReverseAlt* *WRevAndFiniteAlt* *WRevRev* *all-rev-eq(8)*)

all-wrev-eq(7) all-wrev-eq(9) inteq-reflection next-d-def prev-d-def)

lemma *RRNextEqvPrev*:

$\vdash \text{finite} \longrightarrow (\odot (f^r))^r = \text{prev} (f)$

proof –

have 1: $\vdash \text{finite} \longrightarrow (\odot (f^r))^r = \text{prev} ((f^r)^r)$ **using** *RNextEqvPrev* **by** *blast*

have 2: $\vdash \text{finite} \longrightarrow (f^r)^r = f$

by (*simp add: EqvReverseReverse*)

have 3: $\vdash \text{finite} \longrightarrow \text{prev} ((f^r)^r) = \text{prev} f$

unfolding *prev-d-def* **using** 2 **by** (*simp add: FiniteChopEqv*)

from 1 3 **show** *?thesis* **using** *FiniteTransEqv* **by** *blast*

qed

lemma *WRWRNextEqvPrev*:

$\vdash \text{finite} \longrightarrow \text{wrev} (\odot (\text{wrev} f)) = \text{prev} (f)$

by (*metis EqvWReverseWReverse FiniteChopEqv FiniteTransEqv WRNextEqvPrev prev-d-def*)

lemma *RRNextEqvPrevAlt*:

$\vdash (\odot (f^r))^r = (\text{finite} \wedge \text{prev} f)$

by (*metis EqvReverseReverseAlt RevChopAlt all-rev-eq(8) inteq-reflection lift-and-com next-d-def prev-d-def*)

lemma *WRWRNextEqvPrevAlt*:

$\vdash \text{wrev} (\odot (\text{wrev} f)) = (\text{finite} \longrightarrow \text{prev} f)$

by (*metis (no-types, lifting) EqvWReverseWReverseAlt WRevAndFiniteAlt all-rev-eq(8)*

all-wrev-eq(7) all-wrev-eq(9) inteq-reflection next-d-def prev-d-def)

lemma *RWNextEqvWPrev*:

$\vdash \text{finite} \longrightarrow (\text{wnext} f)^r = \text{wprev}(f^r)$

proof –

have 1: $\vdash \text{finite} \longrightarrow (\text{wnext} f)^r = (\text{empty} \vee \text{next} f)^r$

by (*metis ROr ROrAlt WnextEqvEmptyOrNext inteq-reflection*)

have 2: $\vdash \text{finite} \longrightarrow (\text{empty} \vee \text{next} f)^r = (\text{empty}^r \vee (\text{next} f)^r)$

by (*simp add: ROr*)

have 3: $\vdash \text{finite} \longrightarrow (\text{empty}^r \vee (\text{next} f)^r) = (\text{empty} \vee (\text{prev} (f^r)))$

using *REmptyEqvEmpty RNextEqvPrev[of f]* **by** *fastforce*

have 4: $\vdash \text{finite} \longrightarrow (\text{empty} \vee (\text{prev} (f^r))) = \text{wprev}(f^r)$

by (*metis WprevEqvEmptyOrPrev int-simps(1) int-simps(12) inteq-reflection*)

show *?thesis*

by (*metis 3 FiniteTransEqv WnextEqvEmptyOrNext WprevEqvEmptyOrPrev inteq-reflection rev-fun2*)

qed

lemma *WRWNextEqvWPrev*:

$\vdash \text{finite} \longrightarrow \text{wrev} (\text{wnext} f) = \text{wprev}(\text{wrev} f)$

by (*metis (no-types, opaque-lifting) EqvReverseReverse EqvWReverseWReverseAlt FiniteTransEqv RNextEqvPrevAlt WRevRev int-simps(4) inteq-reflection wreverse-d-def wnext-d-def wprev-d-def wrev-fun1*)

lemma *RWNextEqvWPrevAlt*:

$\vdash (\text{wnext} f)^r = \text{wprev}(f^r)$

proof –
have 1: $\vdash (wnext\ f)^r = (empty \vee next\ f)^r$
by (*simp add: REquivRule WnextEqvEmptyOrNext*)
have 2: $\vdash (empty \vee next\ f)^r = (empty^r \vee (next\ f)^r)$
by (*simp add: ROrAlt*)
have 3: $\vdash (empty^r \vee (next\ f)^r) = (empty \vee (prev\ (f^r)))$
by (*metis 2 REmptyEqvEmpty RNextEqvPrevAlt inteq-reflection*)
have 4: $\vdash (empty \vee (prev\ (f^r))) = wprev(f^r)$
by (*meson WprevEqvEmptyOrPrev int-iffD1 int-iffD2 int-iffI*)
show ?thesis
by (*metis 1 2 3 4 inteq-reflection*)
qed

lemma *WRWNextEqvWPrevAlt*:
 $\vdash wrev\ (wnext\ f) = wprev(wrev\ f)$
by (*metis (no-types, lifting) RNextEqvPrevAlt int-simps(4) inteq-reflection itl-def(20) wfreverse-d-def wprev-d-def*)

lemma *RRWNextEqvWPrev*:
 $\vdash finite \longrightarrow (wnext\ (f^r))^r = wprev(f)$
proof –
have 1: $\vdash finite \longrightarrow (wnext\ (f^r))^r = wprev\ ((f^r)^r)$ **using** *RRWNextEqvWPrev* **by** *blast*
have 2: $\vdash finite \longrightarrow (f^r)^r = f$
by (*simp add: EqvReverseReverse*)
have 3: $\vdash finite \longrightarrow wprev\ ((f^r)^r) = wprev\ f$
by (*simp add: 2 FiniteWPrevEqv*)
from 1 3 **show** ?thesis **using** *FiniteTransEqv* **by** *blast*
qed

lemma *WRWRWNextEqvWPrev*:
 $\vdash finite \longrightarrow wrev\ (wnext\ (wrev\ f)) = wprev(f)$
using *EqvWReverseWReverse FiniteTransEqv FiniteWPrevEqv WRWNextEqvWPrev* **by** *blast*

lemma *RRWNextEqvWPrevAlt*:
 $\vdash (wnext\ (f^r))^r = (finite \wedge wprev\ f)$
by (*metis FiniteImpAnd ROrAlt RRWNextEqvWPrev WnextEqvEmptyOrNext all-rev-eq(2) inteq-reflection lift-and-com*)

lemma *WRWRWNextEqvWPrevAlt*:
 $\vdash wrev\ (wnext\ (wrev\ f)) = (finite \longrightarrow wprev(f))$
using *WRWRWNextEqvWPrev WRevRev* **by** *fastforce*

lemma *RPrevEqvNext*:
 $\vdash finite \longrightarrow (prev\ f)^r = \bigcirc\ (f^r)$
by (*metis RevChop RevSkip inteq-reflection next-d-def prev-d-def*)

lemma *WRPrevEqvNext*:
 $\vdash finite \longrightarrow wrev\ (prev\ f) = \bigcirc\ (wrev\ f)$
by (*metis (no-types, opaque-lifting) EqvWReverseWReverse EqvWReverseWReverseAlt FiniteChopEqv WRevRev all-rev-eq(8) all-wrev-eq(7) inteq-reflection next-d-def prev-d-def*)

lemma *RPrevEqvNextAlt*:

$\vdash (\text{prev } f)^r = \bigcirc (f^r)$

by (*metis* *RevChopAlt* *all-rev-eq(8)* *next-d-def* *prev-d-def*)

lemma *WRPrevEqvNextAlt*:

$\vdash \text{wrev } (\text{prev } f) = (\text{finite} \longrightarrow \bigcirc (\text{wrev } f))$

by (*metis* (*no-types*, *lifting*) *EqvWReverseWReverseAlt* *WRWRNextEqvPrevAlt* *WRevAndFiniteAlt* *WRevRev*

all-rev-eq(10) *all-rev-eq(2)* *inteq-reflection*)

lemma *RRPrevEqvNext*:

$\vdash \text{finite} \longrightarrow (\text{prev } (f^r))^r = \bigcirc (f)$

proof –

have 1: $\vdash \text{finite} \longrightarrow (\text{prev } (f^r))^r = \bigcirc ((f^r)^r)$ **using** *RPrevEqvNext* **by** *blast*

have 2: $\vdash \text{finite} \longrightarrow \bigcirc ((f^r)^r) = \bigcirc f$

by (*simp* *add*: *EqvReverseReverse* *FiniteRightChopEqvChop* *next-d-def*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *WRWRPrevEqvNext*:

$\vdash \text{finite} \longrightarrow \text{wrev } (\text{prev } (\text{wrev } f)) = \bigcirc (f)$

by (*metis* *EqvWReverseWReverse* *FiniteRightChopEqvChop* *FiniteTransEqv* *WRPrevEqvNext* *next-d-def*)

lemma *RRPrevEqvNextAlt*:

$\vdash (\text{prev } (f^r))^r = (\text{finite} \wedge \bigcirc f)$

by (*metis* *EqvReverseReverseAlt* *RNextEqvPrevAlt* *inteq-reflection* *lift-and-com*)

lemma *WRWRPrevEqvNextAlt*:

$\vdash \text{wrev } (\text{prev } (\text{wrev } f)) = (\text{finite} \longrightarrow \bigcirc f)$

by (*metis* *EqvWReverseWReverseAlt* *WRNextEqvPrevAlt* *WRPrevEqvNextAlt* *WRWRNextEqvPrevAlt* *inteq-reflection*)

lemma *RWPrevEqvWNext*:

$\vdash \text{finite} \longrightarrow (\text{wprev } f)^r = \text{wnext}(f^r)$

proof –

have 1: $\vdash \text{finite} \longrightarrow (\text{wprev } f)^r = (\text{empty} \vee \text{prev } f)^r$

by (*metis* *WprevEqvEmptyOrPrev* *int-simps(1)* *int-simps(12)* *inteq-reflection*)

have 2: $\vdash \text{finite} \longrightarrow (\text{empty} \vee \text{prev } f)^r = (\text{empty}^r \vee (\text{prev } f)^r)$

by (*simp* *add*: *ROr*)

have 3: $\vdash \text{finite} \longrightarrow (\text{prev } f)^r = (\text{next } (f^r))$

by (*simp* *add*: *RPrevEqvNext*)

have 4: $\vdash \text{finite} \longrightarrow \text{empty}^r = \text{empty}$

using *REmptyEqvEmpty* **by** *auto*

have 5: $\vdash \text{finite} \longrightarrow (\text{empty}^r \vee (\text{prev } f)^r) = (\text{empty} \vee (\text{next } (f^r)))$

by (*simp* *add*: 3 4 *FiniteOrEqv*)

have 6: $\vdash \text{finite} \longrightarrow (\text{empty} \vee (\text{next } (f^r))) = \text{wnext}(f^r)$

by (*metis* *WnextEqvEmptyOrNext* *int-simps(1)* *int-simps(12)* *inteq-reflection*)

show *?thesis*
by (*metis* 5 *FiniteTransEqv WnextEqvEmptyOrNext WprevEqvEmptyOrPrev inteq-reflection rev-fun2*)
qed

lemma *WRWPrevEqvWNext*:

$\vdash \text{finite} \longrightarrow \text{wrev}(\text{wprev } f) = \text{wnext}(\text{wrev } f)$
by (*metis* (*no-types, opaque-lifting*) *RPrevEqvNextAlt WRTrueAlt WReverseEqvAlt int-simps(1)*
int-simps(4) inteq-reflection wreverse-d-def wnext-d-def wprev-d-def)

lemma *RWPrevEqvWNextAlt*:

$\vdash (\text{wprev } f)^r = \text{wnext}(f^r)$
proof –
have 1: $\vdash (\text{wprev } f)^r = (\text{empty} \vee \text{prev } f)^r$
by (*metis* *WprevEqvEmptyOrPrev int-simps(1) inteq-reflection*)
have 2: $\vdash (\text{empty} \vee \text{prev } f)^r = (\text{empty}^r \vee (\text{prev } f)^r)$
by (*simp add: ROrAlt*)
have 3: $\vdash (\text{prev } f)^r = (\text{next } (f^r))$
by (*simp add: RPrevEqvNextAlt*)
have 4: $\vdash \text{empty}^r = \text{empty}$
using *REmptyEqvEmpty* **by** *auto*
have 5: $\vdash (\text{empty}^r \vee (\text{prev } f)^r) = (\text{empty} \vee (\text{next } (f^r)))$
by (*metis* 2 3 *REmptyEqvEmpty inteq-reflection*)
have 6: $\vdash (\text{empty} \vee (\text{next } (f^r))) = \text{wnext}(f^r)$
by (*metis* *WnextEqvEmptyOrNext inteq-reflection*)
show *?thesis*
by (*metis* 1 2 5 6 *inteq-reflection*)
qed

lemma *WRWPrevEqvWNextAlt*:

$\vdash \text{wrev}(\text{wprev } f) = (\text{finite} \longrightarrow \text{wnext}(\text{wrev } f))$
by (*metis* (*no-types, opaque-lifting*) *EqvReverseReverseAlt RPrevEqvNextAlt WRWRNotAlt*
WRevAndFiniteAlt int-simps(4) inteq-reflection wreverse-d-def wnext-d-def wprev-d-def)

lemma *RRWPrevEqvWNext*:

$\vdash \text{finite} \longrightarrow (\text{wprev } (f^r))^r = \text{wnext}(f)$
proof –
have 1: $\vdash \text{finite} \longrightarrow (\text{wprev } (f^r))^r = \text{wnext}((f^r)^r)$ **using** *RWPrevEqvWNext* **by** *blast*
have 2: $\vdash \text{finite} \longrightarrow \text{wnext}((f^r)^r) = \text{wnext } f$
by (*simp add: EqvReverseReverse FiniteWNextEqv*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *WRWRWPrevEqvWNext*:

$\vdash \text{finite} \longrightarrow \text{wrev}(\text{wprev}(\text{wrev } f)) = \text{wnext}(f)$
by (*metis* *EqvWReverseWReverse WRWNextEqvWPrevAlt inteq-reflection*)

lemma *RRWPrevEqvWNextAlt*:

$\vdash (\text{wprev } (f^r))^r = (\text{finite} \wedge \text{wnext } f)$
by (*meson* *EqvReverseReverseAlt Prop04 REqvRule RWNextEqvWPrevAlt lift-and-com*)

lemma *WRWRWPrevEqWNextAlt:*

$\vdash \text{wrev} (\text{wprev} (\text{wrev } f)) = (\text{finite} \longrightarrow \text{wnext } f)$

by (*metis EqWReverseWReverseAlt WRWNextEqWPrevAlt inteq-reflection*)

lemma *RDiamondEqvDi:*

$\vdash \text{finite} \longrightarrow (\Diamond f)^r = \text{di } (f^r)$

by (*metis FiniteRightChopEqvChop FiniteTransEqv Prop11 RevChop all-rev-eq(10) di-d-def int-simps(9) sometimes-d-def*)

lemma *WRDiamondEqvDi:*

$\vdash \text{finite} \longrightarrow \text{wrev} (\Diamond f) = \text{di } (\text{wrev } f)$

unfolding *di-d-def sometimes-d-def*

by (*metis FiniteRightChopEqvChop FiniteTransEqv Prop11 WRevChop WReverseEqv int-simps(9) lift-imp-trans*)

lemma *RDiamondEqvDiAlt:*

$\vdash (\Diamond f)^r = (f^r); \text{finite}$

by (*metis RevChopAlt all-rev-eq(10) sometimes-d-def*)

lemma *WRDiamondEqvDiAlt:*

$\vdash \text{wrev} (\Diamond f) = \text{di } (\text{wrev } f)$

unfolding *di-d-def sometimes-d-def*

by (*metis WRevRev all-rev-eq(10) all-wrev-eq(7) int-simps(13) inteq-reflection*)

lemma *RRDiamondEqvDi:*

$\vdash \text{finite} \longrightarrow (\Diamond (f^r))^r = \text{di } (f)$

proof –

have 1: $\vdash \text{finite} \longrightarrow (\Diamond (f^r))^r = \text{di } ((f^r)^r)$ **using** *RDiamondEqvDi* **by** *blast*

have 2: $\vdash \text{finite} \longrightarrow \text{di } ((f^r)^r) = \text{di } f$

by (*simp add: EqvReverseReverse FiniteChopEqv di-d-def*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *WRWRDiamondEqvDi:*

$\vdash \text{finite} \longrightarrow \text{wrev} (\Diamond (\text{wrev } f)) = \text{di } (f)$

by (*metis EqWReverseWReverse FiniteChopEqv FiniteTransEqv WRDiamondEqvDi di-d-def*)

lemma *RRDiamondEqvDiAlt:*

$\vdash (\Diamond (f^r))^r = (\text{finite} \wedge \text{di } f)$

by (*metis (no-types, opaque-lifting) EqvReverseReverseAlt FiniteImpAnd RRAndAlt RRDiamondEqvDi all-rev-eq(10) all-rev-eq(6) inteq-reflection lift-and-com sometimes-d-def*)

lemma *WRWRDiamondEqvDiAlt:*

$\vdash \text{wrev} (\Diamond (\text{wrev } f)) = (\text{finite} \longrightarrow \text{di } f)$

using *WRWRDiamondEqvDi WRevRev* **by** *fastforce*

lemma *RBoxEqvBi:*

$\vdash \text{finite} \longrightarrow (\Box f)^r = \text{bi } (f^r)$

unfolding *always-d-def bi-d-def*

by (*metis FiniteChopEqv FiniteNotEqv FiniteTransEqv RDiamondEqvDi RNot di-d-def*)

lemma *WRBoxEqvBi*:

$\vdash \text{finite} \longrightarrow \text{wrev } (\Box f) = \text{bi } (\text{wrev } f)$

unfolding *always-d-def bi-d-def*

by (*simp add: FiniteNotEqv RDiamondEqvDi wfreverse-d-def*)

lemma *RBoxEqvBiAlt*:

$\vdash (\Box f)^r = (\text{bi } (f^r) \wedge \text{finite})$

by (*metis FiniteImpAnd RAnd RBoxEqvBi RTrueAlt int-simps(17) inteq-reflection lift-and-com*)

lemma *WRBoxEqvBiAlt*:

$\vdash \text{wrev } (\Box f) = (\text{finite} \longrightarrow \text{bi } (\text{wrev } f))$

using *WRBoxEqvBi WRevRev* **by** *fastforce*

lemma *RRBoxEqvBi*:

$\vdash \text{finite} \longrightarrow (\Box (f^r))^r = \text{bi } (f)$

proof –

have 1: $\vdash \text{finite} \longrightarrow (\Box (f^r))^r = \text{bi } ((f^r)^r)$ **using** *RBoxEqvBi* **by** *blast*

have 2: $\vdash \text{finite} \longrightarrow \text{bi } ((f^r)^r) = \text{bi } f$

by (*simp add: EqvReverseReverse FiniteChopEqv FiniteNotEqv bi-d-def di-d-def*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *WRWRBoxEqvBi*:

$\vdash \text{finite} \longrightarrow \text{wrev } (\Box (\text{wrev } f)) = \text{bi } (f)$

by (*metis (no-types, opaque-lifting) EqvWReverseWReverse FiniteChopEqv FiniteNotEqv FiniteTransEqv WRBoxEqvBi bi-d-def di-d-def*)

lemma *RRBoxEqvBiAlt*:

$\vdash (\Box (f^r))^r = (\text{bi } f \wedge \text{finite})$

by (*metis FiniteImpAnd RAnd RRBoxEqvBi RTrueAlt int-simps(17) inteq-reflection lift-and-com*)

lemma *WRWRBoxEqvBiAlt*:

$\vdash \text{wrev } (\Box (\text{wrev } f)) = (\text{finite} \longrightarrow \text{bi } f)$

using *WRWRBoxEqvBi WRevRev* **by** *fastforce*

lemma *RDIEqvDiamond*:

$\vdash \text{finite} \longrightarrow (\text{di } f)^r = \Diamond (f^r)$

unfolding *sometimes-d-def di-d-def*

by (*meson FiniteChopEqv FiniteTransEqv Prop11 RTrue RevChop int-simps(8)*)

lemma *WRDIEqvDiamond*:

$\vdash \text{finite} \longrightarrow \text{wrev } (\text{di } f) = \Diamond (\text{wrev } f)$

by (*metis (no-types, opaque-lifting) EqvWReverseWReverseAlt RTrueAlt WRevAndFiniteAlt all-rev-eq(2) all-rev-eq(6) all-wrev-eq(2) all-wrev-eq(9) di-d-def int-simps(1) int-simps(12)*)

inteq-reflection sometimes-d-def)

lemma *RDiEqvDiamondAlt:*

$\vdash (di\ f)^r = \Diamond (f^r)$

by (*metis RTrueAlt RevChopAlt di-d-def inteq-reflection sometimes-d-def*)

lemma *WRDiEqvDiamondAlt:*

$\vdash wrev\ (di\ f) = (finite \longrightarrow \Diamond (wrev\ f))$

by (*metis (no-types, lifting) EqvWReverseWReverseAlt WRWRDiamondEqvDiAlt WRevAndFiniteAlt WRevRev all-wrev-eq(9) inteq-reflection*)

lemma *RRDiEqvDiamond:*

$\vdash finite \longrightarrow (di\ (f^r))^r = \Diamond (f)$

proof –

have 1: $\vdash finite \longrightarrow (di\ (f^r))^r = \Diamond ((f^r)^r)$ **using** *RDiEqvDiamond* **by** *blast*

have 2: $\vdash finite \longrightarrow \Diamond ((f^r)^r) = \Diamond f$

by (*simp add: EqvReverseReverse FiniteRightChopEqvChop sometimes-d-def*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *WRWRDiEqvDiamond:*

$\vdash finite \longrightarrow wrev\ (di\ (wrev\ f)) = \Diamond (f)$

by (*metis EqvWReverseWReverse WRDiamondEqvDiAlt int-eq*)

lemma *RRDiEqvDiamondAlt:*

$\vdash (di\ (f^r))^r = (finite \wedge \Diamond f)$

by (*metis EqvReverseReverseAlt RDiEqvDiamondAlt all-rev-eq(10) all-rev-eq(6) inteq-reflection lift-and-com sometimes-d-def*)

lemma *WRWRDiEqvDiamondAlt:*

$\vdash wrev\ (di\ (wrev\ f)) = (finite \longrightarrow \Diamond f)$

by (*meson EqvWReverseWReverseAlt Prop04 WRDiamondEqvDiAlt WREqvRule*)

lemma *RBiEqvBox:*

$\vdash finite \longrightarrow (bi\ f)^r = \Box (f^r)$

unfolding *always-d-def bi-d-def*

by (*metis FiniteNotEqv FiniteRightChopEqvChop FiniteTransEqv RDiEqvDiamond RNot sometimes-d-def*)

lemma *WRBiEqvBox:*

$\vdash finite \longrightarrow wrev\ (bi\ f) = \Box (wrev\ f)$

unfolding *always-d-def bi-d-def*

by (*metis (no-types, opaque-lifting) EqvReverseReverse EqvReverseReverseAlt FiniteTransEqv RDiEqvDiamondAlt all-rev-eq(1) int-simps(4) inteq-reflection wreverse-d-def wrev-fun1*)

lemma *RBiEqvBoxAlt*:

$\vdash (bi\ f)^r = (\Box (f^r) \wedge finite)$

by (*metis FiniteImpAnd RAnd RBiEqvBox RTrueAlt int-simps(17) inteq-reflection lift-and-com*)

lemma *WRBiEqvBoxAlt*:

$\vdash wrev\ (bi\ f) = (finite \longrightarrow \Box (wrev\ f))$

using *WRBiEqvBox WRevRev* **by** *fastforce*

lemma *RRBiEqvBox*:

$\vdash finite \longrightarrow (bi\ (f^r))^r = \Box (f)$

proof –

have 1: $\vdash finite \longrightarrow (bi\ (f^r))^r = \Box ((f^r)^r)$ **using** *RBiEqvBox* **by** *blast*

have 2: $\vdash finite \longrightarrow \Box ((f^r)^r) = \Box f$

by (*simp add: EqvReverseReverse FiniteNotEqv FiniteRightChopEqvChop always-d-def sometimes-d-def*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *WRWRBiEqvBox*:

$\vdash finite \longrightarrow wrev\ (bi\ (wrev\ f)) = \Box (f)$

by (*metis (no-types, opaque-lifting) EqvWReverseWReverse FiniteNotEqv FiniteRightChopEqvChop FiniteTransEqv WRBiEqvBox always-d-def sometimes-d-def*)

lemma *RRBiEqvBoxAlt*:

$\vdash (bi\ (f^r))^r = (\Box f \wedge finite)$

by (*meson FiniteImpAnd Prop04 Prop11 RBiEqvBox RBiEqvBoxAlt RRBiEqvBox*)

lemma *WRWRBiEqvBoxAlt*:

$\vdash wrev\ (bi\ (wrev\ f)) = (finite \longrightarrow \Box f)$

by (*metis EqvWReverseWReverseAlt WRBiEqvBoxAlt WRBoxEqvBiAlt WRWRBoxEqvBiAlt inteq-reflection*)

lemma *RDaEqvDa*:

$\vdash finite \longrightarrow (da\ f)^r = da(f^r)$

proof –

have 1: $\vdash finite \longrightarrow (finite;(f;\#True))^r = (f;\#True)^r; finite^r$ **using** *RevChop* **by** *blast*

have 2: $\vdash finite \longrightarrow (f;\#True)^r; finite^r = (f;\#True)^r; finite$

by (*simp add: all-rev-eq(10)*)

have 3: $\vdash finite \longrightarrow (f;\#True)^r; finite = (\#True^r;f^r);finite$

by (*simp add: FiniteChopEqv RevChop*)

have 4: $\vdash finite \longrightarrow (\#True^r;f^r);finite = (\#True;f^r);finite$

using *FiniteChopEqv RTrue* **by** *blast*

have 5: $\vdash finite \longrightarrow (\#True;f^r);finite = \#True;(f^r;finite)$

using *ChopAssoc* **by** *fastforce*

have 6: $\vdash finite \longrightarrow (finite;(f;\#True))^r = \#True;(f^r;finite)$ **using** 1 2 3 4 5 **by** *fastforce*

have 7: $\vdash finite \longrightarrow \#True;(f^r;finite) = finite;(f^r;\#True)$

by (*meson FiniteChopEqv FiniteRightChopEqvChop FiniteTransEqv Prop11 int-simps(8) int-simps(9)*)

from 6 7 **show** *?thesis*

unfolding *da-d-def*

using *FiniteTransEqv* **by** *blast*

qed

lemma *WRDaEqvDa*:

$\vdash \text{finite} \longrightarrow \text{wrev} (\text{da } f) = \text{da}(\text{wrev } f)$

proof –

have 1: $\vdash \text{finite} \longrightarrow \text{wrev} (\text{finite};(f;\# \text{True})) = \text{wrev} (f;\# \text{True}); \text{wrev } \text{finite}$

by (*simp add: WRevChop*)

have 2: $\vdash \text{finite} \longrightarrow \text{wrev} (f;\# \text{True}); \text{wrev } \text{finite} = \text{wrev} (f;\# \text{True}); \text{finite}$

by (*metis FiniteRightChopEqvChop InfEqvNotFinite all-rev-eq(10) finite-d-def inteq-reflection wfreverse-d-def wrev-fun1*)

have 3: $\vdash \text{finite} \longrightarrow \text{wrev} (f;\# \text{True}) ; \text{finite} = (\text{wrev } \# \text{True}; \text{wrev } f); \text{finite}$

by (*simp add: FiniteChopEqv WRevChop*)

have 4: $\vdash \text{finite} \longrightarrow (\text{wrev } \# \text{True}; \text{wrev } f); \text{finite} = (\# \text{True}; \text{wrev } f); \text{finite}$

using *FiniteChopEqv WRTrue* **by** *blast*

have 5: $\vdash \text{finite} \longrightarrow (\# \text{True}; \text{wrev } f); \text{finite} = \# \text{True}; (\text{wrev } f; \text{finite})$

using *ChopAssoc* **by** *fastforce*

have 6: $\vdash \text{finite} \longrightarrow \text{wrev} (\text{finite};(f;\# \text{True})) = \# \text{True}; (\text{wrev } f; \text{finite})$

using 1 2 3 4 5 **by** *fastforce*

have 7: $\vdash \text{finite} \longrightarrow \# \text{True}; (\text{wrev } f; \text{finite}) = \text{finite}; (\text{wrev } f; \# \text{True})$

by (*meson FiniteChopEqv FiniteRightChopEqvChop FiniteTransEqv Prop11 int-simps(8) int-simps(9)*)

from 6 7 **show** *?thesis*

by (*metis FiniteTransEqv da-d-def*)

qed

lemma *RDaEqvDaAlt*:

$\vdash (\text{da } f)^r = (\text{finite} \wedge \text{da } (f^r))$

by (*metis (no-types, lifting) EqvReverseReverseAlt FiniteImpAnd RDaEqvDa RRAndAlt WRevAndFiniteAlt*

inteq-reflection lift-and-com)

lemma *WRDaEqvDaAlt*:

$\vdash \text{wrev} (\text{da } f) = (\text{finite} \longrightarrow \text{da } (\text{wrev } f))$

by (*metis FiniteImp FiniteImpAnd Prop10 WRDaEqvDa WRevAndFiniteAlt WRevRev int-iffD1 inteq-reflection*)

lemma *RRDaEqvDa*:

$\vdash \text{finite} \longrightarrow (\text{da } (f^r))^r = \text{da}(f)$

proof –

have 1: $\vdash \text{finite} \longrightarrow (\text{da } (f^r))^r = \text{da } ((f^r)^r)$ **using** *RDaEqvDa* **by** *blast*

have 2: $\vdash \text{finite} \longrightarrow \text{da } ((f^r)^r) = \text{da } f$

by (*simp add: EqvReverseReverse FiniteChopEqv FiniteRightChopEqvChop da-d-def*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *WRWRDaEqvDa*:

$\vdash \text{finite} \longrightarrow \text{wrev} (\text{da } (\text{wrev } f)) = \text{da}(f)$

by (*metis EqvWReverseWReverse FiniteChopEqv FiniteRightChopEqvChop FiniteTransEqv WRDaEqvDa da-d-def*)

lemma *RRDaEqvDaAlt*:

$\vdash (da (f^r))^r = (finite \wedge da f)$
by (*metis FiniteImpAnd RDaEqvDa RDaEqvDaAlt RRDaeqvDa inteq-reflection lift-and-com*)

lemma *WRWRDaEqvDaAlt*:

$\vdash wrev (da (wrev f)) = (finite \longrightarrow da f)$
by (*metis EqvWReverseWReverseAlt WRDaEqvDaAlt WRevFiniteImp inteq-reflection*)

lemma *RBaEqvBa*:

$\vdash finite \longrightarrow (ba f)^r = ba(f^r)$

unfolding *ba-d-def*

by (*metis FiniteChopEqv FiniteNotEqv FiniteRightChopEqvChop FiniteTransEqv RDaEqvDa RNot itl-def(12)*)

lemma *WRBaEqvBa*:

$\vdash finite \longrightarrow wrev (ba f) = ba(wrev f)$

unfolding *ba-d-def wreverse-d-def*

by (*simp add: FiniteNotEqv RDaEqvDa*)

lemma *RBaEqvBaAlt*:

$\vdash (ba f)^r = (ba(f^r) \wedge finite)$

by (*metis FiniteImpAnd RAnd RBaEqvBa RTrueAlt int-simps(17) inteq-reflection lift-and-com*)

lemma *WRBaEqvBaAlt*:

$\vdash wrev (ba f) = (finite \longrightarrow ba(wrev f))$

by (*metis (no-types, opaque-lifting) EqvWReverseWReverseAlt FiniteImpAnd RDaEqvDa WRevAndFiniteAlt*

WRevRev all-wrev-eq(9) ba-d-def int-simps(4) inteq-reflection wreverse-d-def)

lemma *RRBaEqvBa*:

$\vdash finite \longrightarrow (ba (f^r))^r = ba(f)$

proof –

have 1: $\vdash finite \longrightarrow (ba (f^r))^r = ba ((f^r)^r)$ **using** *RBaEqvBa* **by** *blast*

have 2: $\vdash finite \longrightarrow ba ((f^r)^r) = ba f$

by (*simp add: EqvReverseReverse FiniteChopEqv FiniteNotEqv FiniteRightChopEqvChop ba-d-def da-d-def*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *WRWRBaEqvBa*:

$\vdash finite \longrightarrow wrev (ba (wrev f)) = ba(f)$

by (*metis (no-types, opaque-lifting) EqvWReverseWReverse FiniteChopEqv FiniteNotEqv FiniteRightChopEqvChop FiniteTransEqv WRBaEqvBa ba-d-def da-d-def*)

lemma *RRBaEqvBaAlt*:

$\vdash (ba (f^r))^r = (ba f \wedge finite)$

by (*meson FiniteImpAnd Prop04 Prop11 RBaEqvBa RBaEqvBaAlt RRBaEqvBa*)

lemma *WRWRBaEqvBaAlt*:

$\vdash wrev (ba (wrev f)) = (finite \longrightarrow ba f)$

by (*metis (no-types, opaque-lifting) EqvReverseReverseAlt EqvWReverseWReverseAlt FiniteImpAnd WRWRBaEqvBa all-wrev-eq(9) int-simps(4) inteq-reflection wreverse-d-def*)

lemma *RInitEqvFin*:

$\vdash \text{finite} \longrightarrow (\text{init } f)^r = \text{fin}(f)$

proof –

have 1: $\vdash \text{finite} \longrightarrow (\text{init } f)^r = ((f \wedge \text{empty}); \# \text{True})^r$

by (*metis init-d-def int-simps(1) int-simps(12) inteq-reflection lift-and-com*)

have 2: $\vdash \text{finite} \longrightarrow ((f \wedge \text{empty}); \# \text{True})^r = (\# \text{True}; (f \wedge \text{empty})^r)$

by (*meson FiniteChopEqv FiniteTransEqv RTrue RevChop*)

have 3: $\vdash \# \text{True}; (f \wedge \text{empty})^r = \# \text{True}; (f^r \wedge \text{empty})$

by (*metis RAnd REmptyEqvEmpty RightChopEqvChop int-eq*)

have 4: $\vdash \text{finite} \longrightarrow \# \text{True}; (f^r \wedge \text{empty}) = \# \text{True}; (f \wedge \text{empty})$

using *RAndEmptyEqvAndEmpty*

by (*metis 3 FiniteRightChopEqvChop REmptyEqvEmpty inteq-reflection rev-fun2*)

have 5: $\vdash \# \text{True}; (f \wedge \text{empty}) = \text{fin}(f)$

using *FinEqvTrueChopAndEmpty* **by** *fastforce*

from 1 2 3 4 5 **show** *?thesis* **by** *fastforce*

qed

lemma *WRInitEqvFin*:

$\vdash \text{finite} \longrightarrow \text{wrev } (\text{init } f) = \text{fin}(f)$

by (*metis EqvWReverseWReverse EqvWReverseWReverseAlt FiniteTransEqv RInitEqvFin WRevRev inteq-reflection*)

lemma *RInitEqvFinAlt*:

$\vdash (\text{init } f)^r = (\text{sfinit}(f))$

proof –

have 1: $\vdash (\text{init } f)^r = ((f \wedge \text{empty}); \# \text{True})^r$

by (*metis init-d-def int-simps(1) inteq-reflection lift-and-com*)

have 2: $\vdash ((f \wedge \text{empty}); \# \text{True})^r = (\text{finite}; (f \wedge \text{empty})^r)$

by (*metis RTrueAlt RevChopAlt inteq-reflection*)

have 3: $\vdash \text{finite}; (f \wedge \text{empty})^r = \text{finite}; (f^r \wedge \text{empty})$

by (*metis RAnd REmptyEqvEmpty RightChopEqvChop int-eq*)

have 4: $\vdash \text{finite}; (f^r \wedge \text{empty}) = \text{finite}; (f \wedge \text{empty})$

by (*metis 3 RAndEmptyEqvAndEmpty inteq-reflection*)

have 5: $\vdash \text{finite}; (f \wedge \text{empty}) = \text{sfinit}(f)$

by (*metis DiamondSChopdef FiniteChopEqvDiamond SFinEqvTrueSChopAndEmpty inteq-reflection*)

from 1 2 3 4 5 **show** *?thesis* **by** *fastforce*

qed

lemma *WRInitEqvFinAlt*:

$\vdash \text{wrev } (\text{init } f) = (\text{finite} \longrightarrow \text{fin}(f))$

by (*metis EqvWReverseWReverseAlt FiniteImpAnd RInitEqvFin WRevRev all-wrev-eq(9) inteq-reflection*)

lemma *RFinEqvInit*:

$\vdash \text{finite} \longrightarrow (\text{fin } f)^r = \text{init } (f)$

proof –

have 1: $\vdash \text{fin } f = \# \text{True}; (f \wedge \text{empty})$

using *FinEqvTrueChopAndEmpty* **by** *auto*

have 2: $\vdash (\text{fin } f)^r = (\# \text{True}; (f \wedge \text{empty}))^r$

using 1 *REqvRule* **by** *blast*

have 3: $\vdash \text{finite} \longrightarrow (\# \text{True}; (f \wedge \text{empty}))^r = (f \wedge \text{empty})^r; \# \text{True}$

by (meson FiniteRightChopEqvChop FiniteTransEqv RTrue RevChop)
 have 4: $\vdash (f \wedge \text{empty})^r; \# \text{True} = (f^r \wedge \text{empty}); \# \text{True}$
 using LeftChopEqvChop RAnd REmptyEqvEmpty by (metis int-eq)
 have 5: $\vdash (f \wedge \text{empty})^r; \# \text{True} = (f \wedge \text{empty}); \# \text{True}$
 by (simp add: RAndEmptyEqvAndEmpty LeftChopEqvChop)
 have 6: $\vdash (f \wedge \text{empty}); \# \text{True} = \text{init}(f)$
 by (simp add: AndChopCommutate init-d-def)
 from 1 2 3 4 5 6 show ?thesis by fastforce
 qed

lemma WRFineqvInit:

$\vdash \text{finite} \longrightarrow \text{wrev}(\text{fin } f) = \text{init } (f)$
 by (metis EqvWReverseWReverse EqvWReverseWReverseAlt FiniteTransEqv RFinEqvInit WRevRev in-
 teq-reflection)

lemma RFinEqvInitAlt:

$\vdash (\text{fin } f)^r = (\text{finite} \wedge \text{init } f)$

proof –

have 1: $\vdash \text{fin } f = \# \text{True}; (f \wedge \text{empty})$
 using FinEqvTrueChopAndEmpty by auto
 have 2: $\vdash (\text{fin } f)^r = (\# \text{True}; (f \wedge \text{empty}))^r$
 using 1 REqvRule by blast
 have 3: $\vdash (\# \text{True}; (f \wedge \text{empty}))^r = (f \wedge \text{empty})^r; \text{finite}$
 by (metis RTrueAlt RevChopAlt inteq-reflection)
 have 4: $\vdash (f \wedge \text{empty})^r; \text{finite} = (f^r \wedge \text{empty}); \text{finite}$
 using LeftChopEqvChop RAnd REmptyEqvEmpty by (metis int-eq)
 have 5: $\vdash (f \wedge \text{empty})^r; \text{finite} = (f \wedge \text{empty}); \text{finite}$
 by (simp add: RAndEmptyEqvAndEmpty LeftChopEqvChop)
 have 6: $\vdash (f \wedge \text{empty}); \text{finite} = (\text{finite} \wedge \text{init } f)$
 by (meson InitAndEmptyEqvAndEmpty LeftChopEqvChop Prop04 StateAndEmptyChop lift-and-com)
 from 1 2 3 4 5 6 show ?thesis by fastforce
 qed

lemma WRFineqvInitAlt:

$\vdash \text{wrev}(\text{fin } f) = (\text{finite} \longrightarrow \text{init } f)$

by (metis EqvWReverseWReverseAlt FiniteImpAnd WRInitEqvFin all-wrev-eq(9) inteq-reflection)

lemma RHaltEqvInitonly:

$\vdash \text{finite} \longrightarrow (\text{halt } f)^r = \text{initonly}(f^r)$

proof –

have 1: $\vdash (\text{halt } f)^r = (\Box (\text{empty} = f))^r$ by (simp add: halt-d-def)
 have 2: $\vdash \text{finite} \longrightarrow (\Box (\text{empty} = f))^r = \text{bi} (\text{empty} = f)^r$ by (simp add: RBoxEqvBi)
 have 3: $\vdash \text{finite} \longrightarrow (\text{empty} = f)^r = (\text{empty} = f^r)$ by (metis REmptyEqvEmpty inteq-reflection rev-fun2)
 hence 4: $\vdash \text{finite} \longrightarrow \text{bi} (\text{empty} = f)^r = \text{bi}(\text{empty} = f^r)$
 by (metis (no-types, opaque-lifting) FiniteChopEqv FiniteNotEqv bi-d-def di-d-def)
 have 5: $\vdash \text{bi}(\text{empty} = f^r) = \text{initonly}(f^r)$ by (simp add: initonly-d-def)
 from 1 2 4 5 show ?thesis by fastforce
 qed

lemma *WRHaltEqvInitonly*:

$\vdash \text{finite} \longrightarrow \text{wrev}(\text{halt } f) = \text{initonly}(\text{wrev } f)$

by (*metis* (*no-types*, *opaque-lifting*) *EqvReverseReverse FiniteImpAnd FiniteInitonlyEqv RHaltEqvInitonly WRevAndFiniteAlt WReverseEqvAlt all-wrev-eq(2) all-wrev-eq(9) int-eq*)

lemma *RHaltEqvInitonlyAlt*:

$\vdash (\text{halt } f)^r = (\text{initonly } f^r) \wedge \text{finite}$

proof –

have 1: $\vdash (\text{halt } f)^r = (\Box (\text{empty} = f))^r$ **by** (*simp add: halt-d-def*)

have 2: $\vdash (\Box (\text{empty} = f))^r = (\text{bi } (\text{empty} = f)^r) \wedge \text{finite}$ **by** (*simp add: RBoxEqvBiAlt*)

have 3: $\vdash (\text{empty} = f)^r = ((\text{empty} = f^r) \wedge \text{finite})$

using *REmptyEqvEmpty FiniteImpAnd*[*of LIFT (empty = f)^r*]

by (*metis* (*no-types*, *opaque-lifting*) *RAnd RTrueAlt int-simps(17) inteq-reflection lift-and-com rev-fun2*)

hence 4: $\vdash (\text{bi } (\text{empty} = f)^r) \wedge \text{finite} = (\text{bi } (\text{empty} = f^r) \wedge \text{finite}) \wedge \text{finite}$

by (*metis* 2 *inteq-reflection*)

have 39: $\vdash \text{bi } \text{finite} = \text{finite}$

by (*metis AndInfChopEqvAndInf InfEqvNotFinite bi-d-def di-d-def finite-d-def int-simps(17) inteq-reflection*)

have 40: $\vdash (\text{bi } (\text{empty} = f^r) \wedge \text{finite}) \wedge \text{finite} = (\text{bi } (\text{empty} = f^r)) \wedge \text{finite}$

by (*metis BiAndEqv BiElim 39 Prop10 Prop12 inteq-reflection*)

have 5: $\vdash (\text{bi } (\text{empty} = f^r) \wedge \text{finite}) = (\text{initonly } f^r) \wedge \text{finite}$ **by** (*simp add: initonly-d-def*)

from 1 2 4 5 **show** ?thesis **by** (*metis* 40 *inteq-reflection*)

qed

lemma *WRHaltEqvInitonlyAlt*:

$\vdash \text{wrev}(\text{halt } f) = (\text{finite} \longrightarrow \text{initonly}(\text{wrev } f))$

using *WRHaltEqvInitonly WRevRev* **by** *fastforce*

lemma *RInitonlyEqvHalt*:

$\vdash \text{finite} \longrightarrow (\text{initonly } f)^r = \text{halt}(f^r)$

proof –

have 1: $\vdash (\text{initonly } f)^r = (\text{bi } (\text{empty} = f))^r$ **by** (*simp add: initonly-d-def*)

have 2: $\vdash \text{finite} \longrightarrow (\text{bi } (\text{empty} = f))^r = \Box((\text{empty} = f)^r)$ **by** (*simp add: RBiEqvBox*)

have 3: $\vdash \text{finite} \longrightarrow (\text{empty} = f)^r = (\text{empty} = f^r)$ **by** (*metis REmptyEqvEmpty inteq-reflection rev-fun2*)

hence 4: $\vdash \text{finite} \longrightarrow \Box((\text{empty} = f)^r) = \Box(\text{empty} = f^r)$

by (*metis* (*no-types*, *opaque-lifting*) *FiniteNotEqv FiniteRightChopEqvChop always-d-def sometimes-d-def*)

have 5: $\vdash \Box(\text{empty} = f^r) = \text{halt}(f^r)$ **by** (*simp add: halt-d-def*)

from 1 2 4 5 **show** ?thesis **by** *fastforce*

qed

lemma *WRInitonlyEqvHalt*:

$\vdash \text{finite} \longrightarrow \text{wrev}(\text{initonly } f) = \text{halt}(\text{wrev } f)$

by (*metis* (*no-types*, *lifting*) *RInitonlyEqvHalt WRHaltEqvInitonlyAlt WRevAndFiniteAlt WReverseEqvAlt*

all-wrev-eq(2) all-wrev-eq(9) inteq-reflection)

lemma *RInitonlyEqvHaltAlt*:

$\vdash (\text{initonly } f)^r = (\text{halt}(f^r) \wedge \text{finite})$

by (*metis* (*no-types*, *lifting*) *EqvReverseReverseAlt FiniteImpAnd RInitonlyEqvHalt RRAAndAlt WRevAndFiniteAlt all-rev-eq(10) inteq-reflection*)

lemma *WRInitonlyEqvHaltAlt*:

$\vdash \text{wrev } (\text{initonly } f) = (\text{finite} \longrightarrow \text{halt}(\text{wrev } f))$

by (*metis* (*no-types*, *lifting*) *EqvWReverseWReverseAlt RInitonlyEqvHaltAlt WRHaltEqvInitonlyAlt WRevAndFiniteAlt WRevRev all-wrev-eq(9) inteq-reflection*)

lemma *RRHaltEqvInitonly*:

$\vdash \text{finite} \longrightarrow (\text{halt } (f^r))^r = \text{initonly } (f)$

proof –

have 1: $\vdash \text{finite} \longrightarrow (\text{halt } (f^r))^r = \text{initonly } ((f^r)^r)$ **using** *RHaltEqvInitonly* **by** *blast*

have 2: $\vdash \text{finite} \longrightarrow \text{initonly } ((f^r)^r) = \text{initonly}(f)$

by (*simp add: EqvReverseReverse FiniteInitonlyEqv*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *WRWRHaltEqvInitonly*:

$\vdash \text{finite} \longrightarrow \text{wrev } (\text{halt } (\text{wrev } f)) = \text{initonly } (f)$

by (*meson EqvWReverseWReverse FiniteInitonlyEqv FiniteTransEqv WRHaltEqvInitonly*)

lemma *RRHaltEqvInitonlyAlt*:

$\vdash (\text{halt } (f^r))^r = (\text{initonly } (f) \wedge \text{finite})$

by (*metis EqvReverseReverse FiniteImpAnd FiniteInitonlyEqv RHaltEqvInitonlyAlt inteq-reflection*)

lemma *WRWRHaltEqvInitonlyAlt*:

$\vdash \text{wrev } (\text{halt } (\text{wrev } f)) = (\text{finite} \longrightarrow \text{initonly } (f))$

by (*metis* (*no-types*, *lifting*) *EqvWReverseWReverseAlt WRInitonlyEqvHaltAlt WRevAndFiniteAlt WRevRev all-wrev-eq(9) inteq-reflection*)

lemma *RRInitonlyEqvHalt* :

$\vdash \text{finite} \longrightarrow (\text{initonly } (f^r))^r = \text{halt}(f)$

proof –

have 1: $\vdash \text{finite} \longrightarrow (\text{initonly } (f^r))^r = \text{halt}((f^r)^r)$ **using** *RInitonlyEqvHalt* **by** *blast*

have 2: $\vdash \text{finite} \longrightarrow \text{halt}((f^r)^r) = \text{halt}(f)$

by (*simp add: EqvReverseReverse FiniteHaltEqv*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *WRWRInitonlyEqvHalt* :

$\vdash \text{finite} \longrightarrow \text{wrev } (\text{initonly } (\text{wrev } f)) = \text{halt}(f)$

by (*meson EqvWReverseWReverse FiniteHaltEqv FiniteTransEqv WRInitonlyEqvHalt*)

lemma *RRInitonlyEqvHaltAlt* :

$\vdash (\text{initonly } (f^r))^r = (\text{halt}(f) \wedge \text{finite})$

by (*metis EqvReverseReverse FiniteHaltEqv FiniteImpAnd RInitonlyEqvHaltAlt inteq-reflection*)

lemma *WRWRInitonlyEqvHaltAlt* :

$\vdash \text{wrev} (\text{initonly} (\text{wrev } f)) = (\text{finite} \longrightarrow \text{halt}(f))$

by (*metis* *EqvWReverseWReverseAlt* *WRHaltEqvInitonlyAlt* *WRInitonlyEqvHaltAlt* *WRWRHaltEqvInitonlyAlt* *inteq-reflection*)

lemma *RKeepEqvKeep* :

$\vdash \text{finite} \longrightarrow (\text{keep } f)^r = \text{keep}(f^r)$

proof –

have 1: $\vdash (\text{keep } f)^r = (\text{ba}(\text{skip} \longrightarrow f))^r$ **by** (*simp* *add: keep-d-def*)

have 2: $\vdash \text{finite} \longrightarrow (\text{ba}(\text{skip} \longrightarrow f))^r = \text{ba}((\text{skip} \longrightarrow f)^r)$ **by** (*simp* *add: RBaEqvBa*)

have 3: $\vdash \text{finite} \longrightarrow (\text{skip} \longrightarrow f)^r = (\text{skip} \longrightarrow f^r)$

by (*metis* *all-rev-eq(8)* *rev-fun2*)

hence 4: $\vdash \text{finite} \longrightarrow \text{ba}((\text{skip} \longrightarrow f)^r) = \text{ba}(\text{skip} \longrightarrow f^r)$

by (*simp* *add: FiniteChopEqv FiniteNotEqv FiniteRightChopEqvChop* *ba-d-def* *da-d-def*)

have 5: $\vdash \text{ba}(\text{skip} \longrightarrow f^r) = \text{keep}(f^r)$ **by** (*simp* *add: keep-d-def*)

from 1 2 4 5 **show** *?thesis* **by** *fastforce*

qed

lemma *WRKeepEqvKeep* :

$\vdash \text{finite} \longrightarrow \text{wrev} (\text{keep } f) = \text{keep}(\text{wrev } f)$

by (*metis* (*no-types*, *opaque-lifting*) *EqvReverseReverseAlt* *FiniteImpAnd* *RKeepEqvKeep* *WRevAndFiniteAlt*

WReverseEqvAlt *all-wrev-eq(2)* *all-wrev-eq(9)* *inteq-reflection*)

lemma *RKeepEqvKeepAlt* :

$\vdash (\text{keep } f)^r = (\text{keep}(f^r) \wedge \text{finite})$

using *FiniteImpAnd[of LIFT (keep f)^r RKeepEqvKeep*

by (*metis* (*no-types*, *opaque-lifting*) *Prop11* *Prop12* *RImpRule* *RTrueAlt* *int-simps(12)* *int-simps(13)* *inteq-reflection*)

lemma *WRKeepEqvKeepAlt* :

$\vdash \text{wrev} (\text{keep } f) = (\text{finite} \longrightarrow \text{keep}(\text{wrev } f))$

using *WRKeepEqvKeep* *WRevRev* **by** *fastforce*

lemma *RRKeepEqvKeep* :

$\vdash \text{finite} \longrightarrow (\text{keep } (f^r))^r = \text{keep}(f)$

proof –

have 1: $\vdash \text{finite} \longrightarrow (\text{keep } (f^r))^r = \text{keep}((f^r)^r)$ **using** *RKeepEqvKeep* **by** *blast*

have 2: $\vdash \text{finite} \longrightarrow \text{keep}((f^r)^r) = \text{keep}(f)$

by (*simp* *add: EqvReverseReverse* *FiniteKeepEqv*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *WRWRKeepEqvKeep* :

$\vdash \text{finite} \longrightarrow \text{wrev} (\text{keep } (\text{wrev } f)) = \text{keep}(f)$

by (*meson* *EqvWReverseWReverse* *FiniteKeepEqv* *FiniteTransEqv* *WRKeepEqvKeep*)

lemma *RRKeepEqvKeepAlt* :

$\vdash (\text{keep } (f^r))^r = (\text{keep } f) \wedge \text{finite}$

by (*metis EqvReverseReverse FiniteImpAnd FiniteKeepEqv RRKeepEqvKeepAlt inteq-reflection*)

lemma *WRWRKeepEqvKeepAlt* :

$\vdash \text{wrev } (\text{keep } (\text{wrev } f)) = (\text{finite} \longrightarrow \text{keep } f)$

by (*metis (no-types, lifting) EqvWReverseWReverseAlt WRKeepEqvKeepAlt WRevAndFiniteAlt WRevRev*

all-wrev-eq(9) inteq-reflection)

lemma *RassignEqvTAssign*:

$\vdash \text{finite} \longrightarrow (\$v = e)^R = (v \leftarrow e^R)$

proof –

have 1: $\vdash \text{finite} \longrightarrow (\$v = e)^R = ((\$v)^R = e^R)$

using *rev-fun2-e* **by** *blast*

have 20: $\vdash \text{finite} \longrightarrow (\$v)^R = (!v)$

by (*simp add: rev-current*)

have 2: $\vdash \text{finite} \longrightarrow ((\$v)^R = e^R) = (!v = e^R)$

using 20 **by** (*auto simp add: Valid-def itl-defs*)

have 3: $\vdash \text{finite} \longrightarrow (!v = e^R) = (v \leftarrow e^R)$ **by** (*simp add: intI temporal-assign-d-def*)

from 1 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *RassignEqvTAssignAlt*:

$\vdash ((\$v = e)^R \wedge \text{finite}) = ((v \leftarrow e^R) \wedge \text{finite})$

by (*simp add: FiniteImpAnd RassignEqvTAssign*)

lemma *RTAssignEqvAssign*:

$\vdash \text{finite} \longrightarrow (v \leftarrow e)^r = (\$v = e^R)$

proof –

have 1: $\vdash \text{finite} \longrightarrow (v \leftarrow e)^r = (\text{finite} \longrightarrow !v = e)^r$

unfolding *temporal-assign-d-def*

by (*metis int-simps(1) int-simps(12) inteq-reflection*)

have 2: $\vdash \text{finite} \longrightarrow (\text{finite} \longrightarrow !v = e)^r = (\$v = e^R)$

by (*auto simp add: Valid-def itl-defs freverse-d-def ereverse-d-def nfinite-nrev nlast-nrev*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *WRTAssignEqvAssign*:

$\vdash \text{finite} \longrightarrow \text{wrev } (v \leftarrow e) = (\$v = e^R)$

by (*metis (no-types, opaque-lifting) FiniteNotEqv FiniteTransEqv RTAssignEqvAssign int-simps(4) inteq-reflection rev-fun1 wfreverse-d-def*)

lemma *RTAssignEqvAssignAlt*:

$\vdash (v \leftarrow e)^r = ((\$v = e^R) \wedge \text{finite})$

by (*metis FiniteImpAnd RAnd RTAssignEqvAssign RTrueAlt int-simps(17) inteq-reflection lift-and-com*)

lemma *WRTAssignEqvAssignAlt*:

$\vdash \text{wrev } (v \leftarrow e) = (\text{finite} \longrightarrow (\$v = e^R))$

using *RTAssignEqvAssignAlt WRevRev* **by** *fastforce*

lemma *RNextAssignEqvPrevAssign*:

$\vdash \text{finite} \longrightarrow (v := e)^r = (v =: e^R)$

proof –

have 1: $\vdash \text{finite} \longrightarrow (v := e)^r = (v\$ = e)^r$ **by** (*simp add: REqvRule intI next-assign-d-def*)

have 20: $\vdash \text{finite} \longrightarrow (v\$ = e)^r = ((v\$)^R = e^R)$

by (*simp add: ereverse-d-def freverse-d-def intI itl-defs(5)*)

have 21: $\vdash \text{finite} \longrightarrow ((v\$)^R = e^R) = (v! = e^R)$

by (*simp add: Valid-def itl-defs freverse-d-def ereverse-d-def*)

(*metis One-nat-def Suc-ile-eq epred-conv-minus gr-zeroI idiff-enat-enat nfinite-nlength-enat
nnth-nrev one-enat-def the-enat.simps zero-enat-def*)

have 2: $\vdash \text{finite} \longrightarrow (v\$ = e)^r = (v! = e^R)$

using 20 21 *FiniteTransEqv* **by** *blast*

have 3: $\vdash \text{finite} \longrightarrow (v! = e^R) = (v =: e^R)$

by (*auto simp add: Valid-def itl-defs freverse-d-def ereverse-d-def*)

from 1 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *WRNextAssignEqvPrevAssign*:

$\vdash \text{finite} \longrightarrow \text{wrev } (v := e) = (v =: e^R)$

by (*metis EqvWReverseWReverse EqvWReverseWReverseAlt FiniteTransEqv RNextAssignEqvPrevAssign
WRevRev*

inteq-reflection)

lemma *RNextAssignEqvPrevAssignAlt*:

$\vdash (v := e)^r = ((v =: e^R) \wedge \text{finite})$

using *FiniteImpAnd[of LIFT (v := e)^r] RNextAssignEqvPrevAssign[of v e]*

by (*metis (no-types, opaque-lifting) Prop11 Prop12 RImpRule RTrueAlt int-simps(12) int-simps(13)
inteq-reflection*)

lemma *WRNextAssignEqvPrevAssignAlt*:

$\vdash \text{wrev } (v := e) = (\text{finite} \longrightarrow (v =: e^R))$

by (*metis EqvWReverseWReverseAlt RNextAssignEqvPrevAssignAlt WRevRev all-wrev-eq(9) inteq-reflection*)

lemma *RPrevAssignEqvNextAssign*:

$\vdash \text{finite} \longrightarrow (v =: e)^r = (v := e^R)$

proof –

have 1: $\vdash \text{finite} \longrightarrow (v =: e)^r = (v! = e)^r$

by (*auto simp add: Valid-def itl-defs freverse-d-def ereverse-d-def nfinite-nrev*)

have 2: $\vdash \text{finite} \longrightarrow (v! = e)^r = (v\$ = e^R)$

by (*simp add: Valid-def itl-defs freverse-d-def ereverse-d-def nfinite-nrev*)

(*metis One-nat-def Suc-ile-eq epred-enat i0-less ndropn-nfirst nfinite-ndropn nfinite-nlength-enat
nlast-nrev nrev-ndropn ntaken-nlast the-enat.simps zero-enat-def*)

have 3: $\vdash \text{finite} \longrightarrow (v\$ = e^R) = (v := e^R)$ **by** (*simp add: next-assign-d-def*)

from 1 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *WRPrevAssignEqvNextAssign*:

$\vdash \text{finite} \longrightarrow \text{wrev } (v =: e) = (v := e^R)$

by (*metis* (*no-types*, *opaque-lifting*) *FiniteNotEqv FiniteTransEqv RPrevAssignEqvNextAssign int-simps*(4)

inteq-reflection rev-fun1 wfreverse-d-def)

lemma *RPrevAssignEqvNextAssignAlt*:

$\vdash (v =: e)^r = ((v := e^R) \wedge \text{finite})$

using *FiniteImpAnd*[of *LIFT* ($v =: e$)^r] *RPrevAssignEqvNextAssign*

by (*metis* (*no-types*, *opaque-lifting*) *Prop11 Prop12 RImpRule RTrueAlt int-simps*(12)

int-simps(13) *inteq-reflection*)

lemma *WRPrevAssignEqvNextAssignAlt*:

$\vdash \text{wrev } (v =: e) = (\text{finite} \longrightarrow (v := e^R))$

by (*metis* *EqvWReverseWReverseAlt RPrevAssignEqvNextAssignAlt WRevRev all-wrev-eq*(9) *inteq-reflection*)

lemma *RGetsEqvBaSkipImp*:

$\vdash \text{finite} \longrightarrow (v \text{ gets } e)^r = \text{ba}(\text{skip} \longrightarrow (\$v = e^R))$

proof –

have 01: $\vdash (\text{skip} \longrightarrow \text{finite} \longrightarrow !v = e) =$
 $(\text{skip} \longrightarrow !v = e)$

using *nlength-eq-enat-nfiniteD* **by** (*simp add: Valid-def itl-defs, blast*)

have 1: $\vdash \text{finite} \longrightarrow (v \text{ gets } e)^r = (\text{ba}(\text{skip} \longrightarrow (!v = e)))^r$

unfolding *gets-d-def keep-d-def temporal-assign-d-def*

by (*metis* 01 *EqvReverseReverse REmptyEqvEmpty int-simps*(1) *inteq-reflection*)

have 2: $\vdash \text{finite} \longrightarrow (\text{ba}(\text{skip} \longrightarrow (!v = e)))^r = \text{ba} (\text{skip} \longrightarrow (!v = e))^r$

by (*simp add: RBaEqvBa*)

have 3: $\vdash \text{finite} \longrightarrow (\text{skip} \longrightarrow (!v = e))^r = (\text{skip} \longrightarrow (\$v = e^R))$

by (*auto simp add: Valid-def itl-defs freverse-d-def ereverse-d-def nfinite-nrev nlast-nrev*)

hence 4: $\vdash \text{finite} \longrightarrow \text{ba} ((\text{skip} \longrightarrow (!v = e))^r) = \text{ba} (\text{skip} \longrightarrow (\$v = e^R))$

by (*simp add: FiniteChopEqv FiniteNotEqv FiniteRightChopEqvChop ba-d-def da-d-def*)

from 1 2 4 **show** *?thesis* **by** *fastforce*

qed

lemma *WRGetsEqvBaSkipImp*:

$\vdash \text{finite} \longrightarrow \text{wrev } (v \text{ gets } e) = \text{ba}(\text{skip} \longrightarrow (\$v = e^R))$

by (*metis* *FiniteKeepEqv FiniteTransEqv WRKeepEqvKeep WRTAssignEqvAssign gets-d-def keep-d-def*)

lemma *RGetsEqvBaSkipImpAlt*:

$\vdash (v \text{ gets } e)^r = (\text{ba}(\text{skip} \longrightarrow (\$v = e^R)) \wedge \text{finite})$

by (*metis* *FiniteImpAnd RAnd RGetsEqvBaSkipImp RTrueAlt int-simps*(17) *inteq-reflection lift-and-com*)

lemma *WRGetsEqvBaSkipImpAlt*:

$\vdash \text{wrev } (v \text{ gets } e) = (\text{finite} \longrightarrow \text{ba}(\text{skip} \longrightarrow (\$v = e^R)))$

by (*metis* *EqvWReverseWReverseAlt RGetsEqvBaSkipImpAlt WRevRev all-wrev-eq*(9) *inteq-reflection*)

lemma *RIfThenElse*:

$\vdash \text{finite} \longrightarrow (\text{if}_i f0 \text{ then } f1 \text{ else } f2)^r = \text{if}_i (f0^r) \text{ then } (f1^r) \text{ else } (f2^r)$

unfolding *ifthenelse-d-def*

by (*auto simp add: Valid-def itl-defs freverse-d-def ereverse-d-def*)

lemma *WRIfThenElse*:

$\vdash \text{finite} \longrightarrow \text{wrev} (\text{if}_i f0 \text{ then } f1 \text{ else } f2) = \text{if}_i (\text{wrev } f0) \text{ then } (\text{wrev } f1) \text{ else } (\text{wrev } f2)$

unfolding *ifthenelse-d-def*

by (*metis (no-types, lifting) FiniteAndEqv FiniteOrEqv WRAnd WRNot WROrAlt int-simps(1) inteq-reflection wrev-fun2*)

lemma *RIfThenElseAlt*:

$\vdash (\text{if}_i f0 \text{ then } f1 \text{ else } f2)^r = (\text{if}_i (f0^r) \text{ then } (f1^r) \text{ else } (f2^r)) \wedge \text{finite}$

using *RIfThenElse[of f0 f1 f2]*

FiniteImpAnd[of LIFT (if_i f0 then f1 else f2)^r]

by (*metis RAnd RTrueAlt int-simps(17) inteq-reflection lift-and-com*)

lemma *WRIfThenElseAlt*:

$\vdash \text{wrev} (\text{if}_i f0 \text{ then } f1 \text{ else } f2) = (\text{finite} \longrightarrow \text{if}_i (\text{wrev } f0) \text{ then } (\text{wrev } f1) \text{ else } (\text{wrev } f2))$

by (*metis (no-types, lifting) FiniteImp FiniteImpAnd Prop12 WRIfThenElse WRevAndFiniteAlt WRevFinite WRevRev int-iffD1 int-iffI inteq-reflection*)

lemma *RWhile*:

$\vdash (\text{init } f \wedge \text{swhile } f0 \text{ do } f1)^r = (\text{fin}(f) \wedge (\text{schopstar} ((f0^r) \wedge (f1^r))) \wedge \text{init} (\neg(f0)))$

proof –

have 1: $\vdash (\text{init } f \wedge \text{swhile } f0 \text{ do } f1)^r = (\text{init } f \wedge (\text{schopstar} (f0 \wedge f1)) \wedge \text{sfin} (\neg f0))^r$

by (*simp add: swile-d-def*)

have 2: $\vdash (\text{init } f \wedge (\text{schopstar} (f0 \wedge f1)) \wedge \text{sfin} (\neg f0))^r = ((\text{init } f)^r \wedge (\text{schopstar} (f0 \wedge f1))^r \wedge (\text{sfin} (\neg f0))^r)$

by (*metis RAnd inteq-reflection*)

have 3: $\vdash \text{finite} \longrightarrow (\text{init } f)^r = \text{fin}(f)$

by (*simp add: RInitEqvFin*)

have 4: $\vdash (\text{schopstar} (f0 \wedge f1))^r = (\text{schopstar} ((f0^r) \wedge (f1^r)))$

by (*metis RAnd RevChopstar inteq-reflection*)

have 40: $\vdash \text{finite} \longrightarrow (\text{sfin} (\neg f0)) = (\text{fin} (\neg f0))$

by (*metis (no-types, opaque-lifting) FinEqvTrueChopAndEmpty Finprop(4) int-simps(4) inteq-reflection sfin-d-def*)

have 41: $\vdash \text{finite} \longrightarrow (\text{sfin} (\neg f0))^r = (\text{fin} (\neg f0))^r$

by (*metis 40 Prop12 ReverseEqv all-rev-eq(2)*)

have 5: $\vdash \text{finite} \longrightarrow (\text{sfin} (\neg f0))^r = \text{init} (\neg(f0))$

using 41 *FiniteTransEqv RFinEqvInit* **by** *blast*

have 50: $\vdash (\text{schopstar} (f0 \wedge f1))^r \longrightarrow \text{finite}$

using 4 *SChopstar-finite* **by** *fastforce*

have 51: $\vdash (\text{schopstar} ((f0^r) \wedge (f1^r))) \longrightarrow \text{finite}$

using *SChopstar-finite* **by** *blast*

have 6: $\vdash ((\text{init } f)^r \wedge (\text{schopstar} (f0 \wedge f1))^r \wedge (\text{sfin} (\neg f0))^r) =$

$(\text{fin}(f) \wedge (\text{schopstar}((f0^r) \wedge (f1^r))) \wedge \text{init}(\neg(f0)))$
using 3 4 5 50 51 **by** fastforce
from 1 2 6 **show** ?thesis **by** fastforce
qed

lemma WRWhile:

$\vdash \text{wrev}(\text{init } f \wedge \text{swhile } f0 \text{ do } f1) =$
 $(\text{finite} \longrightarrow \text{fin}(f) \wedge (\text{schopstar}((\text{wrev } f0) \wedge (\text{wrev } f1))) \wedge \text{init}(\neg(f0)))$
by (metis (no-types, opaque-lifting) EqvReverseReverseAlt RAnd RWhile SChopstar-Chopstar
WRevAndFiniteAlt WRevRev all-wrev-eq(9) int-eq)

lemma AAXRev:

$\vdash (\forall \forall x. F x)^r = (\forall \forall x. (F x)^r)$

proof –

have 1: $\vdash (\forall \forall x. F x) = (\neg(\exists \exists x. \neg(F x)))$ **using** AAXDef **by** blast
have 2: $\vdash (\forall \forall x. F x)^r = (\neg(\exists \exists x. \neg(F x)))^r$ **using** REqvRule 1 **by** blast
have 3: $\vdash (\neg(\exists \exists x. \neg(F x)))^r = (\neg((\exists \exists x. \neg(F x))^r) \wedge \text{finite})$
using FiniteImpAnd[of LIFT $(\neg(\exists \exists x. \neg(F x)))^r$]
by (simp add: rev-fun1-alt)
have 4: $\vdash ((\exists \exists x. \neg(F x))^r) = ((\exists \exists x. \neg(F x))^r)$ **by** (simp add: EExRev)
hence 5: $\vdash (\neg((\exists \exists x. \neg(F x))^r) \wedge \text{finite}) = (\neg(\exists \exists x. \neg(F x))^r \wedge \text{finite})$ **by** auto
have 51: $\bigwedge x. \vdash ((\neg(F x))^r) = ((\neg(F x))^r \wedge \text{finite})$
using rev-fun1-alt **by** blast
hence 52: $\vdash (\exists \exists x. ((\neg(F x))^r)) = (\exists \exists x. ((\neg(F x))^r \wedge \text{finite}))$
using EExEqvRule[of $(\lambda x. \text{LIFT}(\neg(F x))^r)$ $(\lambda x. \text{LIFT}((\neg(F x))^r \wedge \text{finite}))$]
by fastforce
hence 6: $\vdash (\neg(\exists \exists x. ((\neg(F x))^r) \wedge \text{finite})) = (\neg(\exists \exists x. ((\neg(F x))^r \wedge \text{finite})) \wedge \text{finite})$
by auto
have 60: $\vdash (\neg(\exists \exists x. ((\neg(F x))^r \wedge \text{finite})) \wedge \text{finite}) =$
 $(\neg(\exists \exists x. ((\neg(F x))^r \vee \neg \text{finite}))) \wedge \text{finite}$
by (auto simp add: Valid-def exist-state-d-def)
have 7: $\vdash (\neg(\exists \exists x. ((\neg(F x))^r \vee \neg \text{finite})) \wedge \text{finite}) = ((\forall \forall x. ((F x)^r \vee \neg \text{finite})) \wedge \text{finite})$ **using**
AAXDef
by (simp add: forall-state-d-def)
have 69: $\vdash ((\forall \forall x. ((F x)^r \vee \neg \text{finite})) \wedge \text{finite}) = (\forall \forall x. (\text{finite} \wedge ((F x)^r \vee \neg \text{finite})))$
using AAXAndImport[of LIFT finite $\lambda x. \text{LIFT} (F x)^r \vee \neg \text{finite}$] **by** auto
have 70: $\bigwedge x. \vdash (\text{finite} \wedge ((F x)^r \vee \neg \text{finite})) = (F x)^r$
by (auto simp add: freverse-d-def itl-defs(5))
have 71: $\vdash ((\forall \forall x. (\text{finite} \wedge ((F x)^r \vee \neg \text{finite})))) = (\forall \forall x. (F x)^r)$
by (simp add: 70 AAXEqvRule)
from 1 2 3 5 6 7 70 **show** ?thesis
by (metis 60 69 71 inteq-reflection)
qed

lemma AAXWRev:

$\vdash \text{wrev}(\forall \forall x. F x) = (\forall \forall x. \text{wrev}(F x))$

proof –

have 1: $\vdash \text{wrev}(\forall \forall x. F x) = (\text{finite} \longrightarrow \text{frev}(\forall \forall x. F x))$
by (simp add: WRevRev)


```

have 2:  $\vdash (finite \longrightarrow frev (\forall\forall x. F x)) = (finite \longrightarrow (\forall\forall x. frev (F x)))$ 
  using AAxRev[of F] by auto
have 3:  $\vdash (finite \longrightarrow (\forall\forall x. frev (F x))) = (\forall\forall x. \neg finite \vee frev (F x))$ 
  using AAxOrImport[of LIFT  $\neg finite$   $\lambda x. LIFT frev (F x)$ ] by auto
have 4:  $\bigwedge x. \vdash (\neg finite \vee frev (F x)) = (wrev (F x))$ 
  using WRevRev by fastforce
have 5:  $\vdash (\forall\forall x. \neg finite \vee frev (F x)) = (\forall\forall x. wrev (F x))$ 
by (meson 4 AAxEqRule)
show ?thesis
by (metis 1 2 3 5 inteq-reflection)
qed

```

end

2.12 First occurrence operator

```

theory First
imports
  Omega
begin

```

Runtime verification (RV) has gained significant interest in recent years. The behaviour of a program can be verified in real time by analysing its evolving trace. This approach has two significant benefits over static verification techniques such as model checking. Firstly, it is only necessary to verify actual execution paths rather than all possible paths. Secondly, it is possible to react at runtime should the program diverge from its specified behaviour. RV does not replace traditional verification techniques but it does provide an extra layer of security.

Linear Temporal Logic (LTL) is a popular formalism for writing specifications from which RV monitors can be derived automatically. By contrast, Interval Temporal Logic (ITL) has not been as widely represented in this field despite being more expressive and compositional. The principal issue is efficiency. ITL uses non-deterministic operators to IConstruct sequential and iterative specifications (chop and chop-star, respectively) and these introduce combinatorial complexity. Approaches to mitigate this include using a deterministic subset of ITL or adapting the semantics to include a deterministic chop operator. This work proposes an alternative approach, wholly within existing ITL, and based upon a new, derived operator called “first occurrence”.

A theory of first occurrence is developed and used to derive an algebra of RV monitors.

2.12.1 Definitions

Definitions Strict Initial and Final

```

definition fprev-d :: ('a::world) formula  $\Rightarrow$  'a formula
where fprev-d F  $\equiv LIFT(F \frown skip)$ 

```

syntax

$-fprev-d \quad :: lift \Rightarrow lift ((fprev -) [88] 87)$

syntax (*ASCII*)

$-fprev-d \quad :: lift \Rightarrow lift ((fprev -) [88] 87)$

translations

$-fprev-d \quad \Rightarrow CONST fprev-d$

definition $wfprev-d :: ('a::world) formula \Rightarrow 'a formula$

where $wfprev-d F \equiv LIFT(\neg(fprev(\neg F)))$

syntax

$-wfprev-d \quad :: lift \Rightarrow lift ((wfprev -) [88] 87)$

syntax (*ASCII*)

$-wfprev-d \quad :: lift \Rightarrow lift ((wfprev -) [88] 87)$

translations

$-wfprev-d \quad \Rightarrow CONST wfprev-d$

definition $bs-d :: ('a::world) formula \Rightarrow 'a formula$

where

$bs-d f \equiv LIFT(empty \vee ((bi f) \frown skip))$

syntax

$-bs-d :: lift \Rightarrow lift ((bs -) [88] 87)$

syntax (*ASCII*)

$-bs-d :: lift \Rightarrow lift ((bs -) [88] 87)$

translations

$-bs-d \Rightarrow CONST bs-d$

definition $ds-d :: ('a::world) formula \Rightarrow 'a formula$

where

$ds-d f \equiv LIFT(\neg(bs(\neg f)))$

syntax

$-ds-d :: lift \Rightarrow lift ((ds -) [88] 87)$

syntax (*ASCII*)

$-ds-d :: lift \Rightarrow lift ((ds -) [88] 87)$

translations

$-ds-d \Rightarrow CONST ds-d$

Definition First and Last Operators

definition *first-d* :: ('a::world) formula \Rightarrow 'a formula

where

first-d *f* \equiv *LIFT* (*f* \wedge (*bs* (\neg *f*)))

syntax

-*first-d* :: *lift* \Rightarrow *lift* ((\triangleright -) [88] 87)

syntax (*ASCII*)

-*first-d* :: *lift* \Rightarrow *lift* ((*first* -) [88] 87)

translations

-*first-d* \Rightarrow *CONST first-d*

2.12.2 Semantic Theorems

2.12.3 Bi induction

lemma *finite-ntaken*:

nfinite w \Longrightarrow *f w* = *f* (*ntaken* (*the-enat* (*nlength w*)) *w*)

by (*metis ndropn-eq-NNil ndropn-nlast ntaken-all*)

lemma *biinduct-help*:

($\forall n. \text{enat } n \leq \text{nlength } w \longrightarrow f (\text{ntaken } n \ w) \longrightarrow n = 0 \vee f (\text{ntaken } (n - \text{Suc } 0) \ w) \Longrightarrow$

f w \Longrightarrow

nfinite w \Longrightarrow

enat j \leq *nlength w* \Longrightarrow

enat k = *nlength w* \Longrightarrow

f (*ntaken j w*)

proof (*induct k arbitrary: w j*)

case 0

then show ?*case*

by (*metis le-zero-eq ntaken-all zero-enat-def*)

next

case (*Suc k*)

then show ?*case*

proof –

have 1: *f* (*ntaken* (*k+1*) *w*)

by (*simp add: Suc.premis(2) Suc.premis(5) ntaken-all*)

have 2: *f* (*ntaken* (*k*) *w*)

by (*metis 1 One-nat-def Suc.premis(1) Suc.premis(5) Suc-eq-plus1 Zero-not-Suc diff-Suc-1 nle-le*)

have 3: *w* = *nappend* (*ntaken k w*) (*NNil* (*nlast(w)*))

by (*metis Suc.premis(5) nappend-ntaken-ndropn ndropn-all order-refl*)

have 4: $\forall n. \text{enat } n \leq \text{nlength } (\text{ntaken } (k) \ w) \longrightarrow f (\text{ntaken } n (\text{ntaken } (k) \ w)) \longrightarrow$

$n = 0 \vee f (\text{ntaken } (n - \text{Suc } 0) (\text{ntaken } (k) \ w))$

by (*metis 3 Suc.premis(1) diff-le-self min.bounded-iff ntaken-eq-ntaken-antimono*

ntaken-nappend1 ntaken-nlength)

have 5: *j* = *nlength w* \Longrightarrow ?*thesis*

```

  by (simp add: Suc.premis(2) ntaken-all)
have 6:  $j < \text{nlength } w \implies ?thesis$ 
using 2 3 4 Suc.hyps[of (ntaken k w)]
by (metis (no-types, lifting) Suc.premis(5) Suc-ile-eq linorder-not-less min.orderE
    nfinite-ntaken nle-le ntaken-nappend1 ntaken-nlength)
show ?thesis
using 5 6 Suc.premis(4) order.order-iff-strict by blast
qed
qed

```

lemma BiInduct:

```

 $\vdash bi(f \longrightarrow wprev\ f) \wedge f \wedge finite \longrightarrow bi\ f$ 
proof –
  have 1:  $\bigwedge j\ w.$ 
    ( $\forall n. \text{enat } n \leq \text{nlength } w \longrightarrow f\ (\text{ntaken } n\ w) \longrightarrow n = 0 \vee f\ (\text{ntaken } (n - \text{Suc } 0)\ w) \implies$ 
 $f\ w \implies$ 
 $nfinite\ w \implies$ 
 $\text{enat } j \leq \text{nlength } w \implies$ 
 $f\ (\text{ntaken } j\ w)$ 
  by (metis biinduct-help nfinite-conv-nlength-enat)
  have 2: ( $\vdash bi(f \longrightarrow wprev\ f) \wedge f \wedge finite \longrightarrow bi\ f$ ) =
    ( $\forall w\ n. (\forall n. \text{enat } n \leq \text{nlength } w \longrightarrow f\ (\text{ntaken } n\ w) \longrightarrow n = 0 \vee f\ (\text{ntaken } (n - \text{Suc } 0)\ w) \longrightarrow$ 
 $f\ w \longrightarrow nfinite\ w \longrightarrow \text{enat } n \leq \text{nlength } w \longrightarrow f\ (\text{ntaken } n\ w)) \longrightarrow$ 
    by (auto simp add: Valid-def itl-defs)

  show ?thesis using 1 2 by blast
qed

```

Semantics First Operator

lemma FstAndBfsem:

```

( $\text{nlength } \sigma > 0 \wedge (\sigma \models f) \wedge (\sigma \models bi\ (\neg f) \frown skip)) =$ 
( $\text{nlength } \sigma > 0 \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge (\forall ia < \text{nlength } (\sigma). (\text{ntaken } ia\ \sigma \models \neg f))$ )
proof –
  have ( $\text{nlength } \sigma > 0 \wedge (\sigma \models f) \wedge (\sigma \models bi\ (\neg f) \frown skip) =$ 
    ( $0 < \text{nlength } \sigma \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge$ 
    ( $\exists i. (i \leq \text{nlength } \sigma \longrightarrow (\forall ia \leq i. \neg (\text{ntaken } ia\ (\text{ntaken } i\ \sigma) \models f)) \wedge$ 
     $\text{nlength } \sigma - i = \text{Suc } 0) \wedge i \leq \text{nlength } \sigma)$ 
    )
  unfolding itl-defs by simp
  (metis enat-ord-simps(1) ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD order-subst2)
also have ... =
  ( $0 < \text{nlength } \sigma \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge$ 
  ( $\exists i. (i \leq \text{nlength } \sigma \longrightarrow (\forall ia \leq i. \neg (\text{ntaken } ia\ (\text{ntaken } i\ \sigma) \models f)) \wedge$ 
   $i = \text{nlength } \sigma - \text{Suc } 0) \wedge i \leq \text{nlength } \sigma)$ 
  )
  by (auto simp add: nfinite-conv-nlength-enat)
  (metis Suc-diff-Suc cancel-comm-monoid-add-class.diff-cancel diff-zero lessI ntaken-all)

```

$order.order\text{-}iff\text{-}strict\ zero\text{-}less\text{-}iff\text{-}neq\text{-}zero)$
also have ... =
 $(0 < nlength\ \sigma \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge$
 $(\forall ia. enat\ ia \leq (nlength\ \sigma - Suc\ 0) \longrightarrow \neg (ntaken\ ia\ (ntaken\ (the\text{-}enat\ (nlength\ \sigma - Suc\ 0))\ \sigma) \models$
 $f)))$
 $)$
using *diff-le-self* **by** (*auto simp add: min-def nfinite-conv-nlength-enat, blast*)
also have ... =
 $(nlength\ \sigma > 0 \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge$
 $(\forall ia. ia < nlength\ (\sigma) \longrightarrow \neg (ntaken\ ia\ (ntaken\ (the\text{-}enat\ (nlength\ \sigma - Suc\ 0))\ \sigma) \models f))$
 $)$
by (*metis Suc-pred enat-ord-simps(1) enat-ord-simps(2) idiff-enat-enat less-Suc-eq-le*
nfinite-conv-nlength-enat zero-enat-def)
also have ... =
 $(nlength\ \sigma > 0 \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge$
 $(\forall ia. ia < nlength\ (\sigma) \longrightarrow (ntaken\ ia\ (ntaken\ (the\text{-}enat\ (nlength\ \sigma - Suc\ 0))\ \sigma) \models \neg f))$
 $)$
by *auto*
also have ... =
 $(nlength\ \sigma > 0 \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge (\forall ia < nlength\ (\sigma). (ntaken\ ia\ \sigma \models \neg f)))$
by (*metis Suc-pred enat-ord-simps(2) idiff-enat-enat less-Suc-eq-le min-def*
nfinite-conv-nlength-enat ntaken-ntaken the-enat.simps zero-enat-def)
finally show $(nlength\ \sigma > 0 \wedge (\sigma \models f) \wedge (\sigma \models bi\ (\neg f) \frown skip)) =$
 $(nlength\ \sigma > 0 \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge (\forall ia < nlength\ (\sigma). (ntaken\ ia\ \sigma \models \neg f)))$.
qed

lemma *Fstsem-0*:

$(\sigma \models \triangleright f) =$
 $($
 $(\sigma \models f \wedge empty) \vee (nlength\ \sigma > 0 \wedge (\sigma \models f) \wedge (\sigma \models bi\ (\neg f) \frown skip))$
 $)$

using *empty-defs* **by** (*auto simp add: first-d-def bs-d-def*)

lemma *Emptysem*:

$(\sigma \models f \wedge empty) = ((\sigma \models f) \wedge nlength\ \sigma = 0)$
using *empty-defs* **by** *auto*

lemma *Fstsem*:

$(\sigma \models \triangleright f) =$
 $($
 $(\sigma \models f) \wedge nlength\ \sigma = 0 \vee$
 $(nlength\ \sigma > 0 \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge (\forall ia < nlength\ (\sigma). (ntaken\ ia\ \sigma \models \neg f)))$
 $)$

using *Fstsem-0 Emptysem FstAndBfsem* **by** *metis*

Various Semantic Lemmas

lemma *DiLensem*:

$(\sigma \models di\ (f \wedge len(i))) =$
 $((ntaken\ i\ \sigma \models f) \wedge i \leq nlength\ \sigma)$

using *nlength-eq-enat-nfiniteD* **by** (*auto simp add: itl-defs len-defs*)

lemma *DfLensem*:

($\sigma \models df (f \wedge len(i))$) =
 ($(ntaken\ i\ \sigma \models f) \wedge i \leq nlength\ \sigma$)
by (*auto simp add: itl-defs len-defs*)

lemma *PrefixFstsem*:

($(ntaken\ i\ \sigma \models \triangleright f) \wedge i \leq nlength\ \sigma$) =
 ($i \leq nlength\ \sigma \wedge$
 ($(ntaken\ i\ \sigma \models f) \wedge i = 0$) \vee
 ($i > 0 \wedge (ntaken\ i\ \sigma \models f) \wedge (\forall ia < i. (ntaken\ ia\ \sigma \models \neg f))$))
)
)

proof –

have 1: ($((ntaken\ i\ \sigma) \models \triangleright f)$) =
 ($((ntaken\ i\ \sigma) \models f) \wedge nlength\ (ntaken\ i\ \sigma) = 0$) \vee
 ($nlength\ (ntaken\ i\ \sigma) > 0 \wedge ((ntaken\ i\ \sigma) \models f) \wedge$
 ($\forall ia < nlength\ (ntaken\ i\ \sigma). (ntaken\ ia\ (ntaken\ i\ \sigma) \models \neg f)$))
)

using *Fstsem* **by** (*metis nfinite-ntaken*)

hence 2: ($((ntaken\ i\ \sigma) \models \triangleright f) \wedge i \leq nlength\ \sigma$) =
 ($i \leq nlength\ \sigma \wedge$
 ($((ntaken\ i\ \sigma) \models f) \wedge nlength\ (ntaken\ i\ \sigma) = 0$) \vee
 ($nlength\ (ntaken\ i\ \sigma) > 0 \wedge ((ntaken\ i\ \sigma) \models f) \wedge$
 ($\forall ia < nlength\ (ntaken\ i\ \sigma). (ntaken\ ia\ (ntaken\ i\ \sigma) \models \neg f)$))
)
)

by *auto*

hence 3: ($((ntaken\ i\ \sigma) \models \triangleright f) \wedge i \leq nlength\ \sigma$) =
 ($i \leq nlength\ \sigma \wedge$
 ($((ntaken\ i\ \sigma) \models f) \wedge i = 0$) \vee
 ($i > 0 \wedge ((ntaken\ i\ \sigma) \models f) \wedge (\forall ia < i. (ntaken\ ia\ (ntaken\ i\ \sigma) \models \neg f))$))
)
)

by (*metis diff-zero enat-ord-simps(2) gr-zeroI less-numeral-extra(3) min.orderE ndropn-0 nlength-NNil nsubn-def1 nsubn-nlength-gr-one ntaken-0 ntaken-nlength*)

hence 4: ($((ntaken\ i\ \sigma) \models \triangleright f) \wedge i \leq nlength\ \sigma$) =
 ($i \leq nlength\ \sigma \wedge$
 ($((ntaken\ i\ \sigma) \models f) \wedge i = 0$) \vee
 ($i > 0 \wedge ((ntaken\ i\ \sigma) \models f) \wedge (\forall ia < i. (ntaken\ ia\ \sigma \models \neg f))$))
)
)

using *less-imp-add-positive* **by** *fastforce*

from 4 **show** *?thesis* **by** *auto*

qed

lemma *PrefixFstAndsem*:

```

( (ntaken i σ ⊢ ▷f ∧ g) ∧ i ≤ nlength σ) =
( i ≤ nlength σ ∧
(
( (ntaken i σ ⊢ f ∧ g) ∧ i = 0) ∨
( i > 0 ∧ (ntaken i σ ⊢ f ∧ g) ∧ (∀ ia < i. (ntaken ia σ ⊢ ¬f)))
)
)
using PrefixFstsem[of f i σ] by (metis unl-lift2)

```

lemma *DiLenFstsem*:

```

(σ ⊢ di (▷f ∧ len(i))) =
( i ≤ nlength σ ∧
(
( (ntaken i σ ⊢ f) ∧ i = 0) ∨
( i > 0 ∧ (ntaken i σ ⊢ f) ∧ (∀ ia < i. (ntaken ia σ ⊢ ¬f)))
)
)
by (simp add: DiLensem PrefixFstsem)

```

lemma *DfLenFstsem*:

```

(σ ⊢ df (▷f ∧ len(i))) =
( i ≤ nlength σ ∧
(
( (ntaken i σ ⊢ f) ∧ i = 0) ∨
( i > 0 ∧ (ntaken i σ ⊢ f) ∧ (∀ ia < i. (ntaken ia σ ⊢ ¬f)))
)
)
by (simp add: DfLensem PrefixFstsem)

```

lemma *DiLenFstAndsem*:

```

(σ ⊢ di ((▷f ∧ g) ∧ len(i))) =
( i ≤ nlength σ ∧
(
( (ntaken i σ ⊢ f ∧ g) ∧ i = 0) ∨
( i > 0 ∧ (ntaken i σ ⊢ f ∧ g) ∧ (∀ ia < i. (ntaken ia σ ⊢ ¬f)))
)
)
using DiLensem PrefixFstAndsem by metis

```

lemma *DfLenFstAndsem*:

```

(σ ⊢ df ((▷f ∧ g) ∧ len(i))) =
( i ≤ nlength σ ∧
(
( (ntaken i σ ⊢ f ∧ g) ∧ i = 0) ∨
( i > 0 ∧ (ntaken i σ ⊢ f ∧ g) ∧ (∀ ia < i. (ntaken ia σ ⊢ ¬f)))
)
)
using DfLensem PrefixFstAndsem by metis

```

lemma *FstLenSamesem*:

```
( ( i ≤ nlength σ ∧
  (
    ( (ntaken i σ ⊨ f) ∧ i = 0) ∨
    ( i > 0 ∧ (ntaken i σ ⊨ f) ∧ (∀ ia < i. (ntaken ia σ ⊨ ¬f)))
  )
) ∧
( j ≤ nlength σ ∧
  (
    ( (ntaken j σ ⊨ f) ∧ j = 0) ∨
    ( j > 0 ∧ (ntaken j σ ⊨ f) ∧ (∀ ia < j. (ntaken ia σ ⊨ ¬f)))
  )
)
) → (i = j)
```

by (*metis not-less-iff-gr-or-eq unl-lift*)

2.12.4 Theorems

Fixed length intervals

lemma *LenZeroEqvEmpty*:

$\vdash \text{len}(0) = \text{empty}$

by (*simp add: len-d-def*)

lemma *LenOneEqvSkip*:

$\vdash \text{len}(1) = \text{skip}$

by (*simp add: len-d-def ChopEmpty*)

lemma *LenNPlusOneA*:

$\vdash \text{len}(n+1) = \text{skip}; \text{len}(n)$

by (*simp add: len-d-def*)

lemma *SkipFiniteEqvFiniteSkip*:

$\vdash \text{skip}; \text{finite} = \text{finite}; \text{skip}$

using *FiniteChopSkipEqvSkipChopFinite* **by** *auto*

lemma *TrueEqvFiniteOrInfinite*:

$\vdash \# \text{True} = \text{finite} \vee \text{inf}$

by (*simp add: FiniteOrInfinite*)

lemma *SkipInfEqvinfSkip*:

$\vdash \text{skip}; \text{inf} = \text{inf}; \text{skip}$

by (*metis ChopAndInf MoreAndInfEqvInf MoreEqvSkipChopTrue PowerstarEqvSemhelp2 int-simps(16) integ-reflection lift-and-com*)

lemma *SkipTrueEqvTrueSkip*:

$\vdash \text{skip}; \# \text{True} = \# \text{True}; \text{skip}$

proof –

have 1: $\vdash \text{skip}; \# \text{True} = (\text{skip}; \text{finite} \vee \text{skip}; \text{inf})$


```

  by (meson ChopOrEqv FiniteOrInfinite Prop04 RightChopEqvChop int-eq-true)
have 2:  $\vdash (finite;skip \vee inf;skip) = \#True;skip$ 
  by (meson FiniteOrInfinite LeftChopEqvChop OrChopEqv Prop04 int-eq-true)
have 3:  $\vdash (skip;finite \vee skip;inf) = (finite;skip \vee inf;skip)$ 
  by (metis 2 SkipFiniteEqvFiniteSkip SkipInfEqvinfSkip integ-reflection)
show ?thesis
by (metis 1 2 3 int-eq)
qed

```

lemma *AndExistsLen*:
 $\vdash (f \wedge finite) = (f \wedge (\exists k. len(k)))$
using *Finite-exist-len* **by** *fastforce*

lemma *AndExistsLenChop*:
 $\vdash (f \frown g) = (\exists k. (f \wedge len(k));g)$
by (*simp add: Valid-def len-defs chop-defs schop-defs*)
(*meson min.absorb1 nlength-eq-enat-nfiniteD*)

lemma *AndExistsLenSChop*:
 $\vdash (f \frown g) = (\exists k. (f \wedge len(k)) \frown g)$
by (*simp add: Valid-def len-defs chop-defs schop-defs*)
(*meson min.absorb1 nlength-eq-enat-nfiniteD*)

lemma *AndExistsLenChopR*:
 $\vdash (f;(g \wedge finite)) = (\exists k. f;(g \wedge len(k)))$
by (*simp add: Valid-def len-defs chop-defs finite-defs*)
(*metis ndropn-nlength nfinite-conv-nlength-enat nfinite-ndropn*)

lemma *AndExistsLenSChopR*:
 $\vdash (f \frown (g \wedge finite)) = (\exists k. f \frown (g \wedge len(k)))$
by (*simp add: Valid-def len-defs schop-defs finite-defs*)
(*metis ndropn-nlength nfinite-conv-nlength-enat nfinite-ndropn*)

lemma *LFixedAndDistr*:
 $\vdash ((f0 \wedge len(k));g0 \wedge (f1 \wedge len(k));g1) = ((f0 \wedge f1) \wedge len(k));(g0 \wedge g1)$
by (*auto simp add: Valid-def len-defs chop-defs nlength-eq-enat-nfiniteD*)

lemma *LFixedAndDistrS*:
 $\vdash ((f0 \wedge len(k)) \frown g0 \wedge (f1 \wedge len(k)) \frown g1) = ((f0 \wedge f1) \wedge len(k)) \frown (g0 \wedge g1)$
by (*auto simp add: Valid-def len-defs schop-defs*)

lemma *RFixedAndDistr*:
 $\vdash (f0;(g0 \wedge len(k)) \wedge f1;(g1 \wedge len(k))) = (f0 \wedge f1);((g0 \wedge g1) \wedge len(k))$
unfolding *Valid-def itl-defs len-defs*
by *simp*
(*metis enat.simps(3) enat-add2-eq enat-add-sub-same le-iff-add ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD the-enat.simps*)

lemma *RFixedAndDistrS*:

$\vdash (f0 \frown (g0 \wedge \text{len}(k)) \wedge f1 \frown (g1 \wedge \text{len}(k))) = (f0 \wedge f1) \frown ((g0 \wedge g1) \wedge \text{len}(k))$

by (*auto simp add: Valid-def len-defs schop-defs*)

(*metis enat.simps(3) enat-add2-eq enat-add-sub-same le-iff-add the-enat.simps*)

lemma *LFixedChopEqvFixedSChop*:

$\vdash (f \wedge \text{len}(k));g = (f \wedge \text{len}(k)) \frown g$

by (*auto simp add: Valid-def len-defs schop-defs chop-defs nlength-eq-enat-nfiniteD*)

lemma *LFixedAndDistrA*:

$\vdash ((f0 \wedge \text{len}(k));g0 \wedge (f1 \wedge \text{len}(k));g0) = ((f0 \wedge f1) \wedge \text{len}(k));g0$

proof –

have 1: $\vdash ((f0 \wedge \text{len}(k));g0 \wedge (f1 \wedge \text{len}(k));g0) = ((f0 \wedge f1) \wedge \text{len}(k));(g0 \wedge g0)$

by (*rule LFixedAndDistr*)

have 2: $\vdash ((f0 \wedge f1) \wedge \text{len}(k));(g0 \wedge g0) = ((f0 \wedge f1) \wedge \text{len}(k));g0$

by *auto*

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *LFixedAndDistrB*:

$\vdash ((f0 \wedge \text{len}(k));g0 \wedge (f0 \wedge \text{len}(k));g1) = (f0 \wedge \text{len}(k));(g0 \wedge g1)$

proof –

have 1: $\vdash ((f0 \wedge \text{len}(k));g0 \wedge (f0 \wedge \text{len}(k));g1) = ((f0 \wedge f0) \wedge \text{len}(k));(g0 \wedge g1)$

by (*rule LFixedAndDistr*)

have 2: $\vdash ((f0 \wedge f0) \wedge \text{len}(k));(g0 \wedge g1) = (f0 \wedge \text{len}(k));(g0 \wedge g1)$

by *auto*

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *LFixedAndDistrB1*:

$\vdash (\text{len}(k);f \wedge \text{len}(k);g) = \text{len}(k);(f \wedge g)$

proof –

have 1: $\vdash \text{len}(k);f = (\# \text{True} \wedge \text{len}(k));f$

by *auto*

have 2: $\vdash \text{len}(k);g = (\# \text{True} \wedge \text{len}(k));g$

by *auto*

have 3: $\vdash (\text{len}(k);f \wedge \text{len}(k);g) = ((\# \text{True} \wedge \text{len}(k));f \wedge (\# \text{True} \wedge \text{len}(k));g)$

using 1 2 **by** *auto*

have 4: $\vdash ((\# \text{True} \wedge \text{len}(k));f \wedge (\# \text{True} \wedge \text{len}(k));g) = (\# \text{True} \wedge \text{len}(k));(f \wedge g)$

using *LFixedAndDistrB* **by** *blast*

have 5: $\vdash (\# \text{True} \wedge \text{len}(k));(f \wedge g) = (\text{len}(k));(f \wedge g)$

by *auto*

from 1 2 3 4 5 **show** *?thesis* **by** *auto*

qed

lemma *RFixedAndDistrA*:

$\vdash (f0;(g0 \wedge \text{len}(k)) \wedge f0;(g1 \wedge \text{len}(k))) = f0;((g0 \wedge g1) \wedge \text{len}(k))$

proof –

have 1: $\vdash (f0;(g0 \wedge \text{len}(k)) \wedge f0;(g1 \wedge \text{len}(k))) = (f0 \wedge f0);((g0 \wedge g1) \wedge \text{len}(k))$

by (*rule RFixedAndDistr*)

have 2: $\vdash (f0 \wedge f0);((g0 \wedge g1) \wedge \text{len}(k)) = f0;((g0 \wedge g1) \wedge \text{len}(k))$
by *auto*
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *RFixedAndDistrAS*:
 $\vdash (f0 \frown (g0 \wedge \text{len}(k)) \wedge f0 \frown (g1 \wedge \text{len}(k))) = f0 \frown ((g0 \wedge g1) \wedge \text{len}(k))$
proof –
have 1: $\vdash (f0 \frown (g0 \wedge \text{len}(k)) \wedge f0 \frown (g1 \wedge \text{len}(k))) = (f0 \wedge f0) \frown ((g0 \wedge g1) \wedge \text{len}(k))$
by (rule *RFixedAndDistrS*)
have 2: $\vdash (f0 \wedge f0) \frown ((g0 \wedge g1) \wedge \text{len}(k)) = f0 \frown ((g0 \wedge g1) \wedge \text{len}(k))$
by *auto*
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *RFixedAndDistrB*:
 $\vdash (f0;(g0 \wedge \text{len}(k)) \wedge f1;(g0 \wedge \text{len}(k))) = (f0 \wedge f1);(g0 \wedge \text{len}(k))$
proof –
have 1: $\vdash (f0;(g0 \wedge \text{len}(k)) \wedge f1;(g0 \wedge \text{len}(k))) = (f0 \wedge f1);((g0 \wedge g0) \wedge \text{len}(k))$
by (rule *RFixedAndDistrS*)
have 2: $\vdash (f0 \wedge f1);((g0 \wedge g0) \wedge \text{len}(k)) = (f0 \wedge f1);(g0 \wedge \text{len}(k))$
by *auto*
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *RFixedAndDistrBS*:
 $\vdash (f0 \frown (g0 \wedge \text{len}(k)) \wedge f1 \frown (g0 \wedge \text{len}(k))) = (f0 \wedge f1) \frown (g0 \wedge \text{len}(k))$
proof –
have 1: $\vdash (f0 \frown (g0 \wedge \text{len}(k)) \wedge f1 \frown (g0 \wedge \text{len}(k))) = (f0 \wedge f1) \frown ((g0 \wedge g0) \wedge \text{len}(k))$
by (rule *RFixedAndDistrS*)
have 2: $\vdash (f0 \wedge f1) \frown ((g0 \wedge g0) \wedge \text{len}(k)) = (f0 \wedge f1) \frown (g0 \wedge \text{len}(k))$
by *auto*
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *ChopSkipAndChopSkip*:
 $\vdash (f0;\text{skip} \wedge f1;\text{skip}) = (f0 \wedge f1);\text{skip}$
proof –
have 1: $\vdash (f0;(\# \text{True} \wedge \text{len}(1)) \wedge f1;(\# \text{True} \wedge \text{len}(1))) = (f0 \wedge f1);(\# \text{True} \wedge \text{len}(1))$
by (rule *RFixedAndDistrB*)
have 2: $\vdash (\# \text{True} \wedge \text{len}(1)) = \text{skip}$
using *LenOneEqvSkip* **by** *fastforce*
hence 3: $\vdash f0;(\# \text{True} \wedge \text{len}(1)) = f0;\text{skip}$
using *RightChopEqvChop* **by** *blast*
have 4: $\vdash f1;(\# \text{True} \wedge \text{len}(1)) = f1;\text{skip}$
using 2 *RightChopEqvChop* **by** *blast*
have 5: $\vdash (f0;(\# \text{True} \wedge \text{len}(1)) \wedge f1;(\# \text{True} \wedge \text{len}(1))) = (f0;\text{skip} \wedge f1;\text{skip})$
using 3 4 **by** *fastforce*
have 6: $\vdash (f0 \wedge f1);(\# \text{True} \wedge \text{len}(1)) = (f0 \wedge f1);\text{skip}$
using 2 *RightChopEqvChop* **by** *blast*

from 1 5 6 show ?thesis by fastforce
qed

lemma *SChopSkipAndSChopSkip*:

$\vdash (f0 \frown skip \wedge f1 \frown skip) = (f0 \wedge f1) \frown skip$

proof –

have 1: $\vdash (f0 \frown (\# True \wedge len(1)) \wedge f1 \frown (\# True \wedge len(1))) = (f0 \wedge f1) \frown (\# True \wedge len(1))$
by (rule *RFixedAndDistrBS*)

have 2: $\vdash (\# True \wedge len(1)) = skip$

using *LenOneEqvSkip* by fastforce

hence 3: $\vdash f0 \frown (\# True \wedge len(1)) = f0 \frown skip$

using *RightSChopEqvSChop* by blast

have 4: $\vdash f1 \frown (\# True \wedge len(1)) = f1 \frown skip$

using 2 *RightSChopEqvSChop* by blast

have 5: $\vdash (f0 \frown (\# True \wedge len(1)) \wedge f1 \frown (\# True \wedge len(1))) = (f0 \frown skip \wedge f1 \frown skip)$

using 3 4 by fastforce

have 6: $\vdash (f0 \wedge f1) \frown (\# True \wedge len(1)) = (f0 \wedge f1) \frown skip$

using 2 *RightSChopEqvSChop* by blast

from 1 5 6 show ?thesis by fastforce

qed

lemma *BiAndChopSkipEqv*:

$\vdash (bi (f \wedge g)); skip = ((bi f); skip \wedge (bi g); skip)$

proof –

have 1: $\vdash bi (f \wedge g) = ((bi f) \wedge (bi g))$

by (auto simp add: bi-defs Valid-def)

hence 2: $\vdash (bi (f \wedge g)); skip = (bi f \wedge bi g); skip$

by (rule *LeftChopEqvChop*)

have 3: $\vdash (bi f \wedge bi g); skip = ((bi f); skip \wedge (bi g); skip)$

using *ChopSkipAndChopSkip* by fastforce

from 2 3 show ?thesis by fastforce

qed

lemma *BfAndSChopSkipEqv*:

$\vdash (bf (f \wedge g)) \frown skip = ((bf f) \frown skip \wedge (bf g) \frown skip)$

proof –

have 1: $\vdash bf (f \wedge g) = ((bf f) \wedge (bf g))$

by (auto simp add: bf-defs Valid-def)

hence 2: $\vdash (bf (f \wedge g)) \frown skip = (bf f \wedge bf g) \frown skip$

by (rule *LeftSChopEqvSChop*)

have 3: $\vdash (bf f \wedge bf g) \frown skip = ((bf f) \frown skip \wedge (bf g) \frown skip)$

using *SChopSkipAndSChopSkip* by fastforce

from 2 3 show ?thesis by fastforce

qed

lemma *DiAndChopSkipEqv*:

$\vdash (di (f \wedge g)); skip \longrightarrow (di f); skip \wedge (di g); skip$

proof –

have 1: $\vdash di (f \wedge g) \longrightarrow (di f) \wedge (di g)$

by (simp add: DiAndImpAnd)
 hence 2: $\vdash (di\ f \wedge g);skip \longrightarrow (di\ f \wedge di\ g);skip$
 by (rule LeftChopImpChop)
 have 3: $\vdash (di\ f \wedge di\ g);skip = ((di\ f);skip \wedge (di\ g);skip)$
 using ChopSkipAndChopSkip by fastforce
 from 2 3 show ?thesis by fastforce
 qed

lemma DfAndSChopSkipEqv:
 $\vdash (df\ (f \wedge g)) \frown skip \longrightarrow (df\ f) \frown skip \wedge (df\ g) \frown skip$
proof –
 have 1: $\vdash df\ (f \wedge g) \longrightarrow (df\ f) \wedge (df\ g)$
 by (simp add: DfAndImpAnd)
 hence 2: $\vdash (df\ (f \wedge g)) \frown skip \longrightarrow (df\ f \wedge df\ g) \frown skip$
 by (rule LeftSChopImpSChop)
 have 3: $\vdash (df\ f \wedge df\ g) \frown skip = ((df\ f) \frown skip \wedge (df\ g) \frown skip)$
 using SChopSkipAndSChopSkip by fastforce
 from 2 3 show ?thesis by fastforce
 qed

lemma ChopEmptyAndEmpty:
 $\vdash (f;g \wedge empty) = (f \wedge g \wedge empty)$
 by (simp add: Valid-def itl-defs)
 (metis enat-0-iff(2) le-zero-eq ndropn-0 nfinite-ntaken ntaken-all)

lemma SChopEmptyAndEmpty:
 $\vdash (f \frown g \wedge empty) = (f \wedge g \wedge empty)$
 by (simp add: Valid-def itl-defs)
 (metis is-NNil-ndropn le-numeral-extra(3) ndropn-0 ndropn-is-NNil ntaken-all zero-enat-def)

lemma ChopSkipImpMore:
 $\vdash f;skip \longrightarrow more$
 by (metis DiIntro DiSkipEqvMore RightChopImpMoreRule inteq-reflection)

lemma SChopSkipImpMore:
 $\vdash f \frown skip \longrightarrow more$
 by (metis DiIntro DiSkipEqvMore RightSChopImpMoreRule inteq-reflection)

lemma MoreEqvMoreChopTrue:
 $\vdash more = more;\#True$
proof –
 have 1: $\vdash more = skip;\#True$
 using MoreEqvSkipChopTrue by blast
 have 2: $\vdash \#True = \#True;\#True$
 by (auto simp add: Valid-def chop-defs)
 (metis zero-enat-def zero-le)
 hence 3: $\vdash skip;\#True = skip;(\#True;\#True)$
 using RightChopEqvChop by blast
 have 4: $\vdash skip;(\#True;\#True) = (skip;\#True);\#True$

```

    using ChopAssoc by blast
  have 5:  $\vdash (skip; \# True); \# True = more; \# True$ 
    using MoreEqvSkipChopTrue by (simp add: more-d-def next-d-def)
  from 1 3 4 5 show ?thesis by fastforce
qed

```

```

lemma MoreEqvMoreSChopTrue:
 $\vdash more = more \frown \# True$ 
by (metis MoreEqvSkipSChopTrue SChopAssoc TrueEqvTrueSChopTrue inteq-reflection)

```

```

lemma NotNotChopSkip:
 $\vdash (\neg((\neg f); skip)) = (empty \vee (f; skip))$ 
by (metis WprevEqvEmptyOrPrev prev-d-def wprev-d-def)

```

```

lemma NotSChopFixed:
 $\vdash (\neg(f \frown (g \wedge len(k)))) = (\neg(\Diamond(g \wedge len(k))) \vee ((\neg f) \frown (g \wedge len(k))))$ 
by (auto simp add: itl-defs Valid-def len-defs)
  (metis diff-diff-cancel enat-ord-simps(1) idiff-enat-enat ndropn-nlength
    nfinite-conv-nlength-enat nfinite-ndropn the-enat.simps)

```

```

lemma NotNotSChopSkip:
 $\vdash (\neg((\neg f) \frown skip)) = (empty \vee inf \vee (f \frown skip))$ 
proof -
  have 1:  $\vdash ((\neg f) \frown skip \vee f \frown skip) = (\neg f \vee f) \frown skip$ 
  by (meson OrSChopEqv Prop11)
  have 2:  $\vdash (\neg f \vee f) = \# True$ 
  by simp
  have 3:  $\vdash more \wedge finite \longrightarrow \# True \frown skip$ 
  by (metis DiamondSChopdef FmoreEqvSkipChopFinite SkipFiniteEqvFiniteSkip fmore-d-def
    int-simps(13) int-simps(9) inteq-reflection sometimes-d-def)
  have 4:  $\vdash more \wedge finite \longrightarrow ((\neg f) \frown skip) \vee (f \frown skip)$ 
  using 1 3 by fastforce
  have 5:  $\vdash more \wedge finite \wedge \neg((f \frown skip)) \longrightarrow ((\neg f) \frown skip)$ 
  using 4 by fastforce
  have 6:  $\vdash (\neg((\neg f) \frown skip)) \longrightarrow empty \vee inf \vee (f \frown skip)$ 
  using 5 unfolding empty-d-def finite-d-def by fastforce
  have 7:  $\vdash ((\neg f) \frown skip) \longrightarrow finite$ 
  by (metis LenOneEqvSkip SChopImpFinite inteq-reflection len-k-finite)
  have 8:  $\vdash ((\neg f) \frown skip) \longrightarrow more$ 
  by (simp add: SChopSkipImpMore)
  have 9:  $\vdash ((\neg f) \frown skip) \longrightarrow \neg((f \frown skip))$ 
  by (meson NotSChopSkipEqvFmoreAndNotSChopSkip Prop11 Prop12)
  have 10:  $\vdash ((\neg f) \frown skip) \longrightarrow more \wedge finite \wedge \neg((f \frown skip))$ 
  by (simp add: 7 8 9 Prop12)
  have 11:  $\vdash (empty \vee inf \vee (f \frown skip)) \longrightarrow (\neg((\neg f) \frown skip))$ 
  using 10 unfolding finite-d-def empty-d-def by fastforce
  show ?thesis
  by (meson 11 6 Prop11)
qed

```

lemma *NotFixedSChop*:
 $\vdash (\neg((g \wedge \text{len}(k)) \frown f)) = (\neg(df(g \wedge \text{len}(k))) \vee ((g \wedge \text{len}(k)) \frown (\neg f)))$
proof –
have 1: $\vdash (g \wedge \text{len}(k)) \frown f = (g \wedge \text{len}(k)); f$
by (*meson LFixedChopEqvFixedSChop Prop11*)
have 2: $\vdash (g \wedge \text{len}(k)) \frown (\neg f) = ((g \wedge \text{len}(k)); (\neg f))$
by (*meson LFixedChopEqvFixedSChop Prop11*)
have 3: $\vdash df(g \wedge \text{len}(k)) = di(g \wedge \text{len}(k))$
by (*metis LFixedChopEqvFixedSChop df-d-def di-d-def inteq-reflection*)
show ?thesis **using** 1 2 3
by (*metis NotFixedChop inteq-reflection*)
qed

lemma *NotChopNotSkip*:
 $\vdash (\neg(f; \text{skip})) = (\text{empty} \vee ((\neg f); \text{skip}))$
proof –
have 1: $\vdash (\neg((\neg(\neg f)); \text{skip})) = (\text{empty} \vee ((\neg f); \text{skip}))$ **using** *NotNotChopSkip* **by** *blast*
have 2: $\vdash (\neg((\neg(\neg f)); \text{skip})) = (\neg(f; \text{skip}))$ **by** *auto*
from 1 2 **show** ?thesis **by** *auto*
qed

lemma *NotSChopNotSkip*:
 $\vdash (\neg(f \frown \text{skip})) = (\text{empty} \vee \text{inf} \vee ((\neg f) \frown \text{skip}))$
by (*metis NotNotSChopSkip int-simps(4) inteq-reflection*)

lemma *NotNotSChopInf*:
 $\vdash (\neg(\neg f \frown \text{inf})) = (\text{finite} \vee (bf f))$
by (*auto simp add: Valid-def itl-defs*)

Additional ITL theorems

lemma *DiAndFiniteEqvDfAndFinite*:
 $\vdash (di f \wedge \text{finite}) = (df f \wedge \text{finite})$
by (*auto simp add: Valid-def di-defs df-defs finite-defs*)

lemma *DiEqvDfOrInf*:
 $\vdash di f = (df f \vee (f \wedge \text{inf}))$
by (*simp add: ChopSChopdef df-d-def di-d-def*)

lemma *BiEqvBfAndfinite*:
 $\vdash bi f = (bf f \wedge (\text{inf} \longrightarrow f))$
proof –
have 1: $\vdash di(\neg f) = (df(\neg f) \vee \neg f \wedge \text{inf})$
using *DiEqvDfOrInf[of LIFT $\neg f$]* **by** *blast*
have 2: $\vdash (\neg di(\neg f)) = (\neg df(\neg f) \wedge (\text{inf} \longrightarrow f))$
using 1 **by** *fastforce*
show ?thesis

unfolding *bf-d-def bi-d-def* **by** (*simp add: 2*)
qed

lemma *DfEqvDiAndFinite*:
 $\vdash df\ f = di\ (f \wedge finite)$
by (*simp add: df-d-def di-d-def schop-d-def*)

lemma *BfEqvBiOrInf*:
 $\vdash bf\ f = bi\ (f \vee inf)$
proof –
have 1: $\vdash df\ (\neg f) = di\ (\neg f \wedge finite)$
by (*simp add: DfEqvDiAndFinite*)
have 2: $\vdash (\neg df\ (\neg f)) = (\neg di\ (\neg f \wedge finite))$
using 1 **by** *auto*
have 3: $\vdash (\neg di\ (\neg f \wedge finite)) = (bi\ (\neg (\neg f \wedge finite)))$
by (*simp add: NotDiEqvBiNot*)
have 4: $\vdash (\neg (\neg f \wedge finite)) = (f \vee inf)$
unfolding *finite-d-def* **by** *fastforce*
have 5: $\vdash bi\ (\neg (\neg f \wedge finite)) = bi\ (f \vee inf)$
by (*metis 3 4 inteq-reflection*)
have 6: $\vdash bf\ f = (\neg (df\ (\neg f)))$
by (*simp add: bf-d-def*)
show *?thesis*
by (*metis 2 3 5 6 inteq-reflection*)
qed

lemma *BiOrBiImpBiOr*:
 $\vdash bi\ f \vee bi\ g \longrightarrow bi(f \vee g)$
proof –
have 1: $\vdash f \longrightarrow f \vee g$ **by** *auto*
hence 2: $\vdash bi\ f \longrightarrow bi(f \vee g)$ **by** (*rule BiImpBiRule*)
have 3: $\vdash g \longrightarrow f \vee g$ **by** *auto*
hence 4: $\vdash bi\ g \longrightarrow bi(f \vee g)$ **by** (*rule BiImpBiRule*)
from 2 4 **show** *?thesis* **by** *fastforce*
qed

lemma *BfOrBfImpBfOr*:
 $\vdash bf\ f \vee bf\ g \longrightarrow bf(f \vee g)$
proof –
have 1: $\vdash f \longrightarrow f \vee g$ **by** *auto*
hence 2: $\vdash bf\ f \longrightarrow bf(f \vee g)$ **by** (*rule BfImpBfRule*)
have 3: $\vdash g \longrightarrow f \vee g$ **by** *auto*
hence 4: $\vdash bf\ g \longrightarrow bf(f \vee g)$ **by** (*rule BfImpBfRule*)
from 2 4 **show** *?thesis* **by** *fastforce*
qed

lemma *MoreAndBiImpBiChopSkip*:
 $\vdash more \wedge bi\ f \longrightarrow (bi\ f);skip$

proof –

have 1: $\vdash (bi\ f);skip = ((\neg(di\ (\neg\ f))));skip$ **by** (*simp add: bi-d-def*)
have 2: $\vdash (\neg(\neg(di\ (\neg\ f))));skip = (empty \vee (di\ (\neg\ f));skip)$ **by** (*rule NotNotChopSkip*)
have 3: $\vdash empty \longrightarrow empty \vee di\ (\neg\ f)$ **by** *auto*
have 4: $\vdash (di\ (\neg\ f));skip \longrightarrow di\ (\neg\ f)$ **using** *ChopImpDi DiEqvDiDi* **by** *fastforce*
hence 5: $\vdash (di\ (\neg\ f));skip \longrightarrow empty \vee di\ (\neg\ f)$ **by** (*rule Prop05*)
have 6: $\vdash \neg(\neg(di\ (\neg\ f)));skip \longrightarrow empty \vee di\ (\neg\ f)$ **using** 2 3 5 **by** *fastforce*
hence 7: $\vdash \neg(empty \vee di\ (\neg\ f)) \longrightarrow \neg(\neg(\neg(di\ (\neg\ f)));skip))$ **by** *fastforce*
have 8: $\vdash (\neg(\neg(\neg(di\ (\neg\ f)));skip))) = ((\neg(di\ (\neg\ f)));skip)$ **by** *auto*
have 9: $\vdash (\neg(empty \vee di\ (\neg\ f))) = (more \wedge \neg(di\ (\neg\ f)))$
unfolding *empty-d-def* **by** *force*
have 10: $\vdash (more \wedge \neg(di\ (\neg\ f))) = (more \wedge bi\ f)$ **by** (*simp add: bi-d-def*)
from 1 6 7 8 9 10 **show** *?thesis* **by** (*metis int-eq*)
qed

lemma *MoreAndBiImpBiSChopSkip*:

$\vdash more \wedge finite \wedge bi\ f \longrightarrow (bi\ f) \frown skip$

proof –

have 1: $\vdash (bi\ f) \frown skip = ((\neg(di\ (\neg\ f)))) \frown skip$ **by** (*simp add: bi-d-def*)
have 2: $\vdash (\neg(\neg(di\ (\neg\ f)))) \frown skip = (empty \vee inf \vee (di\ (\neg\ f)) \frown skip)$ **by** (*rule NotNotSChopSkip*)
have 3: $\vdash empty \longrightarrow empty \vee di\ (\neg\ f)$ **by** *auto*
have 03: $\vdash skip \longrightarrow \#True$ **by** *simp*
have 04: $\vdash (di\ (\neg\ f)) \frown skip \longrightarrow di\ (\neg\ f) \frown \#True$ **by** (*simp add: RightSChopImpSChop*)
have 4: $\vdash (di\ (\neg\ f)) \frown skip \longrightarrow df\ (\neg\ f)$ **unfolding** *di-d-def df-d-def*
by (*metis (no-types, opaque-lifting) ChopAssoc ChopImpDi ChopTrueAndFiniteEqvAndFiniteChopFinite di-d-def inteq-reflection schop-d-def*)
hence 5: $\vdash (di\ (\neg\ f)) \frown skip \longrightarrow empty \vee df\ (\neg\ f)$ **by** (*rule Prop05*)
have 6: $\vdash \neg(\neg(di\ (\neg\ f))) \frown skip \longrightarrow empty \vee inf \vee df\ (\neg\ f)$ **using** 2 3 5 **by** *fastforce*
hence 7: $\vdash \neg(empty \vee inf \vee df\ (\neg\ f)) \longrightarrow \neg(\neg(\neg(di\ (\neg\ f))) \frown skip))$ **by** *fastforce*
have 8: $\vdash (\neg(\neg(\neg(di\ (\neg\ f))) \frown skip))) = ((\neg(di\ (\neg\ f))) \frown skip)$ **by** *auto*
have 9: $\vdash (\neg(empty \vee inf \vee df\ (\neg\ f))) = (more \wedge finite \wedge \neg(df\ (\neg\ f)))$
unfolding *empty-d-def finite-d-def* **by** *force*
have 10: $\vdash (more \wedge finite \wedge \neg(df\ (\neg\ f))) = (more \wedge finite \wedge bf\ f)$
by (*simp add: bf-d-def*)
have 11: $\vdash (more \wedge finite \wedge bf\ f) = (more \wedge finite \wedge bi\ f)$
using *BfElim BiEqvBfAndfinite* **by** *fastforce*
from 1 6 7 8 9 10 11 **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *MoreAndBfImpBfSChopSkip*:

$\vdash more \wedge bf\ f \longrightarrow inf \vee (bf\ f) \frown skip$

proof –

have 1: $\vdash (bf\ f) \frown skip = ((\neg(df\ (\neg\ f)))) \frown skip$ **by** (*simp add: bf-d-def*)
have 2: $\vdash (\neg(\neg(df\ (\neg\ f)))) \frown skip = (empty \vee inf \vee (df\ (\neg\ f)) \frown skip)$ **by** (*rule NotNotSChopSkip*)
have 3: $\vdash empty \longrightarrow empty \vee inf \vee df\ (\neg\ f)$ **by** *auto*
have 4: $\vdash (df\ (\neg\ f)) \frown skip \longrightarrow df\ (\neg\ f)$
by (*metis DfEqvDfDf SChopImpDf inteq-reflection*)
hence 5: $\vdash (df\ (\neg\ f)) \frown skip \longrightarrow empty \vee inf \vee df\ (\neg\ f)$
by (*simp add: Prop05*)
have 6: $\vdash \neg(\neg(df\ (\neg\ f))) \frown skip \longrightarrow empty \vee inf \vee df\ (\neg\ f)$ **using** 2 3 5 **by** *fastforce*

hence 7: $\vdash \neg(\text{empty} \vee \text{inf} \vee \text{df } (\neg f)) \longrightarrow \neg(\neg(\neg(\text{df } (\neg f))) \frown \text{skip}))$ **by** *fastforce*
have 8: $\vdash (\neg(\neg(\neg(\text{df } (\neg f))) \frown \text{skip}))) = ((\neg(\text{df } (\neg f))) \frown \text{skip})$ **by** *auto*
have 9: $\vdash (\neg(\text{empty} \vee \text{inf} \vee \text{df } (\neg f))) = (\text{more} \wedge \text{finite} \wedge \neg(\text{df } (\neg f)))$
unfolding *empty-d-def* **unfolding** *finite-d-def* **by** *fastforce*
have 10: $\vdash (\text{more} \wedge \text{finite} \wedge \neg(\text{df } (\neg f))) = (\text{more} \wedge \text{finite} \wedge \text{bf } f)$ **by** (*simp add: bf-d-def*)
from 1 6 7 8 9 10 **show** *?thesis* **unfolding** *finite-d-def*
using *NotNotSChopSkip Prop11* **by** *fastforce*
qed

lemma *DiChopEqvChopDi*:
 $\vdash \text{di}(f;g) = f;(di \ g)$
by (*metis ChopAssoc Prop11 di-d-def*)

lemma *DiChopImpDiB*:
 $\vdash \text{di}(f;g) \longrightarrow \text{di } f$
proof –
have 1: $\vdash f ; (g;\# \text{True}) \longrightarrow \text{di } f$ **by** (*rule ChopImpDi*)
have 2: $\vdash f ; (g;\# \text{True}) = (f;g);\# \text{True}$ **by** (*rule ChopAssoc*)
from 1 2 **show** *?thesis* **by** (*metis di-d-def int-eq*)
qed

lemma *DfSChopEqvSChopDf*:
 $\vdash \text{df}(f;g) = f \frown (\text{df } g)$
by (*metis (no-types, opaque-lifting) ChopAndFiniteDist ChopEmpty SChopAssoc int-eq itl-def(15) itl-def(9)*)

lemma *DfChopEqvDfSChop*:
 $\vdash \text{df}(f;g) = \text{df}(f \frown g)$
by (*metis DfSChopEqvSChopDf Prop04 Prop11 SChopAssoc df-d-def*)

lemma *DfSChopImpDfB*:
 $\vdash \text{df}(f \frown g) \longrightarrow \text{df } f$
proof –
have 1: $\vdash f \frown (g \frown \# \text{True}) \longrightarrow \text{df } f$ **by** (*simp add: SChopImpDf*)
have 2: $\vdash f \frown (g \frown \# \text{True}) = (f \frown g) \frown \# \text{True}$ **by** (*rule SChopAssoc*)
from 1 2 **show** *?thesis* **by** (*metis df-d-def int-eq*)
qed

lemma *BiBiOrImpBi*:
 $\vdash \text{bi } (\text{bi } f \vee \text{bi } g) \longrightarrow \text{bi } f \vee \text{bi } g$
using *BiElim* **by** *auto*

lemma *BfBfOrImpBf*:
 $\vdash \text{bf } (\text{bf } f \vee \text{bf } g) \longrightarrow \text{bf } f \vee \text{bf } g$
proof –
have 1: $\vdash \text{bf } (\text{bf } f \vee \text{bf } g) \wedge \text{finite} \longrightarrow \text{bf } f \vee \text{bf } g$
using *BfElim* **by** *fastforce*
have 2: $\vdash \text{bf } (\text{bf } f \vee \text{bf } g) \wedge \text{inf} \longrightarrow \text{bf } f \vee \text{bf } g$
by (*auto simp add: Valid-def itl-defs*)
(meson nle-le)

show *?thesis* **using** 1 2 **unfolding** *finite-d-def*
by (*meson* 1 *OrFiniteInf Prop02 Prop11 lift-imp-trans*)
qed

lemma *BiImpBiBiOr*:

$\vdash bi\ f \longrightarrow bi\ (bi\ f \vee bi\ g)$

proof –

have 1: $\vdash bi\ f \longrightarrow bi\ f \vee bi\ g$ **by** *auto*

hence 2: $\vdash bi\ (bi\ f) \longrightarrow bi\ (bi\ f \vee bi\ g)$ **using** *BiImpBiRule* **by** *blast*

have 3: $\vdash bi\ (bi\ f) = bi\ f$ **using** *BiEqvBiBi* **by** *fastforce*

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *BfImpBfBfOr*:

$\vdash bf\ f \longrightarrow bf\ (bf\ f \vee bf\ g)$

proof –

have 1: $\vdash bf\ f \longrightarrow bf\ f \vee bf\ g$ **by** *auto*

hence 2: $\vdash bf\ (bf\ f) \longrightarrow bf\ (bf\ f \vee bf\ g)$ **using** *BfImpBfRule* **by** *blast*

have 3: $\vdash bf\ (bf\ f) = bf\ f$ **using** *BfEqvBfBf* **by** *fastforce*

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *BiImpBiBiOrB*:

$\vdash bi\ g \longrightarrow bi\ (bi\ f \vee bi\ g)$

proof –

have 1: $\vdash bi\ g \longrightarrow bi\ f \vee bi\ g$ **by** *auto*

hence 2: $\vdash bi\ (bi\ g) \longrightarrow bi\ (bi\ f \vee bi\ g)$ **using** *BiImpBiRule* **by** *blast*

have 3: $\vdash bi\ (bi\ g) = bi\ g$ **using** *BiEqvBiBi* **by** *fastforce*

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *BfImpBfBfOrB*:

$\vdash bf\ g \longrightarrow bf\ (bf\ f \vee bf\ g)$

proof –

have 1: $\vdash bf\ g \longrightarrow bf\ f \vee bf\ g$ **by** *auto*

hence 2: $\vdash bf\ (bf\ g) \longrightarrow bf\ (bf\ f \vee bf\ g)$ **using** *BfImpBfRule* **by** *blast*

have 3: $\vdash bf\ (bf\ g) = bf\ g$ **using** *BfEqvBfBf* **by** *fastforce*

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *BiBiOrEqvBi*:

$\vdash bi\ (bi\ f \vee bi\ g) = (bi\ f \vee bi\ g)$

proof –

have 1: $\vdash bi\ (bi\ f \vee bi\ g) \longrightarrow bi\ f \vee bi\ g$ **by** (*rule* *BiBiOrImpBi*)

have 2: $\vdash bi\ f \longrightarrow bi\ (bi\ f \vee bi\ g)$ **by** (*rule* *BiImpBiBiOr*)

have 3: $\vdash bi\ g \longrightarrow bi\ (bi\ f \vee bi\ g)$ **by** (*rule* *BiImpBiBiOrB*)

have 4: $\vdash bi\ f \vee bi\ g \longrightarrow bi\ (bi\ f \vee bi\ g)$ **using** 2 3 **by** *fastforce*

from 1 4 **show** *?thesis* **by** *fastforce*

qed

lemma *BfBfOrEqvBf*:

$\vdash bf (bf f \vee bf g) = (bf f \vee bf g)$

by (*meson BfBfOrImpBf BfImpBfBfOr BfImpBfBfOrB Prop02 int-iffI*)

lemma *DiEqvOrDiChopSkipA*:

$\vdash di f = (f \vee di(f;skip))$

proof –

have 1: $\vdash di f = f ;\#True$ **by** (*simp add: di-d-def*)

hence 2: $\vdash di f = f ; (empty \vee more)$ **by** (*simp add: empty-d-def*)

hence 3: $\vdash f ; (empty \vee more) = (f;empty \vee f;more)$ **using** *ChopOrEqv* **by** *blast*

have 4: $\vdash f;empty = f$ **by** (*rule ChopEmpty*)

have 5: $\vdash more = skip;\#True$ **using** *MoreEqvSkipChopTrue* **by** *blast*

hence 6: $\vdash f;more = f;(skip;\#True)$ **using** *RightChopEqvChop* **by** *blast*

have 7: $\vdash f;(skip;\#True) = (f;skip);\#True$ **by** (*rule ChopAssoc*)

from 2 3 4 6 7 **show** *?thesis* **by** (*metis di-d-def int-eq*)

qed

lemma *DfEqvOrDfSChopSkipA*:

$\vdash df f = ((f \wedge finite) \vee df(f \frown skip))$

proof –

have 1: $\vdash df f = f \frown \#True$ **by** (*simp add: df-d-def*)

hence 2: $\vdash df f = f \frown (empty \vee more)$ **by** (*simp add: empty-d-def*)

hence 3: $\vdash f \frown (empty \vee more) = (f \frown empty \vee f \frown more)$ **using** *SChopOrEqv* **by** *blast*

have 4: $\vdash f \frown empty = (f \wedge finite)$ **by** (*simp add: ChopEmpty schop-d-def*)

have 5: $\vdash more = skip \frown \#True$ **using** *MoreEqvSkipSChopTrue* **by** *simp*

hence 6: $\vdash f \frown more = f \frown (skip \frown \#True)$ **using** *RightSChopEqvSChop* **by** *blast*

have 7: $\vdash f \frown (skip \frown \#True) = (f \frown skip) \frown \#True$ **by** (*rule SChopAssoc*)

from 2 3 4 6 7 **show** *?thesis* **by** (*metis df-d-def integ-reflection*)

qed

lemma *DiEqvOrDiChopSkipB*:

$\vdash di f = (f \vee (di f);skip)$

proof –

have 1: $\vdash (di f) = (f \vee di(f;skip))$ **by** (*rule DiEqvOrDiChopSkipA*)

have 2: $\vdash di(f;skip) = (f;skip);\#True$ **by** (*simp add: di-d-def*)

have 3: $\vdash (f;skip);\#True = f;(skip;\#True)$

by (*meson ChopAssoc Prop11*)

have 4: $\vdash di(f;skip) = f;(skip;\#True)$ **using** 2 3 **by** *fastforce*

have 5: $\vdash skip;\#True = \#True;skip$

by (*simp add: SkipTrueEqvTrueSkip*)

hence 6: $\vdash f;(skip;\#True) = f;(\#True;skip)$ **using** *RightChopEqvChop* **by** *blast*

have 7: $\vdash di(f;skip) = f;(\#True;skip)$ **using** 4 6 **by** *fastforce*

have 8: $\vdash f;(\#True;skip) = (f;\#True);skip$ **by** (*rule ChopAssoc*)

have 9: $\vdash (f;\#True);skip = (di f);skip$ **by** (*simp add: di-d-def*)

have 10: $\vdash di(f;skip) = (di f);skip$ **using** 7 8 9 **by** *fastforce*

hence 11: $\vdash (f \vee di(f;skip)) = (f \vee (di f);skip)$ **by** *auto*

from 1 11 **show** *?thesis* **by** *fastforce*

qed

lemma *DfEqvOrDfSChopSkipB*:

$\vdash df\ f = ((f \wedge finite) \vee (df\ f) \frown skip \vee (df\ f) \frown inf)$

proof –

have 1: $\vdash (df\ f) = ((f \wedge finite) \vee df(f \frown skip))$ **by** (rule *DfEqvOrDfSChopSkipA*)

have 2: $\vdash df(f \frown skip) = (f \frown skip) \frown \#True$ **by** (simp add: *df-d-def*)

have 3: $\vdash (f \frown skip) \frown \#True = f \frown (skip \frown \#True)$

by (meson *SChopAssoc Prop11*)

have 4: $\vdash df(f \frown skip) = f \frown (skip \frown \#True)$ **using** 2 3 **by** *fastforce*

have 5: $\vdash skip \frown \#True = (\#True \frown skip \vee \#True \frown inf)$

by (metis *FiniteChopInfEqvInf FmoreEqvSkipChopFinite MoreAndInfEqvInf MoreEqvSkipSChopTrue OrFiniteInf SkipFiniteEqvFiniteSkip fmore-d-def int-simps(17) inteq-reflection schop-d-def*)

hence 6: $\vdash f \frown (skip \frown \#True) = f \frown (\#True \frown skip \vee \#True \frown inf)$ **using** *RightSChopEqvSChop* **by** *blast*

have 7: $\vdash df(f \frown skip) = f \frown (\#True \frown skip \vee \#True \frown inf)$ **using** 4 6 **by** *fastforce*

have 71: $\vdash f \frown (\#True \frown skip \vee \#True \frown inf) = (f \frown (\#True \frown skip) \vee f \frown (\#True \frown inf))$

by (simp add: *SChopOrEqv*)

have 8: $\vdash f \frown (\#True \frown skip) = (f \frown \#True) \frown skip$ **by** (rule *SChopAssoc*)

have 81: $\vdash f \frown (\#True \frown inf) = (f \frown \#True) \frown inf$ **by** (rule *SChopAssoc*)

have 9: $\vdash (f \frown \#True) \frown skip = (df\ f) \frown skip$ **by** (simp add: *df-d-def*)

have 91: $\vdash (f \frown \#True) \frown inf = (df\ f) \frown inf$ **by** (simp add: *df-d-def*)

have 10 : $\vdash df(f \frown skip) = ((df\ f) \frown skip \vee (df\ f) \frown inf)$

using 8 81 9 91 7 71 **by** (metis *int-eq*)

hence 11: $\vdash ((f \wedge finite) \vee df(f \frown skip)) = ((f \wedge finite) \vee (df\ f) \frown skip \vee (df\ f) \frown inf)$ **by** *auto*

from 1 11 **show** *?thesis* **by** *fastforce*

qed

lemma *BiEqvAndEmptyOrBiChopSkip*:

$\vdash bi\ f = (f \wedge (empty \vee (bi\ f); skip))$

proof –

have 1: $\vdash di\ (\neg f) = (\neg f \vee (di\ (\neg f); skip))$ **by** (rule *DiEqvOrDiChopSkipB*)

have 2: $\vdash di\ (\neg f) = (\neg(bi\ f))$ **by** (rule *DiNotEqvNotBi*)

have 3: $\vdash (\neg(bi\ f)) = (\neg f \vee (di\ (\neg f); skip))$ **using** 1 2 **by** *fastforce*

hence 4: $\vdash bi\ f = (\neg(\neg f \vee (di\ (\neg f); skip)))$ **by** *auto*

have 5: $\vdash (\neg(\neg f \vee (di\ (\neg f); skip))) = (f \wedge \neg(di\ (\neg f); skip))$ **by** *auto*

have 6: $\vdash di\ (\neg f); skip = ((\neg(bi\ f)); skip)$ **by** (simp add: *2 LeftChopEqvChop*)

hence 7: $\vdash (\neg(di\ (\neg f); skip)) = (\neg((\neg(bi\ f)); skip))$ **by** *auto*

have 8: $\vdash (\neg((\neg(bi\ f)); skip)) = (empty \vee (bi\ f); skip)$ **using** *NotNotChopSkip* **by** *blast*

hence 9: $\vdash (f \wedge \neg(di\ (\neg f); skip)) = (f \wedge (empty \vee (bi\ f); skip))$ **using** 7 8 **by** *fastforce*

from 4 5 9 **show** *?thesis* **by** *fastforce*

qed

lemma *BfEqvAndEmptyOrBfSChopSkip*:

$\vdash bf\ f = ((finite \longrightarrow f) \wedge (empty \vee inf \vee (bf\ f) \frown skip) \wedge (finite \vee bf\ f))$

proof –

have 1: $\vdash (df\ (\neg f)) = ((\neg f \wedge finite) \vee (df\ (\neg f) \frown skip) \vee (df\ (\neg f) \frown inf))$

by (rule *DfEqvOrDfSChopSkipB*)

have 2: $\vdash (df\ (\neg f)) = (\neg(bf\ f))$ **by** (rule *DfNotEqvNotBf*)

have 3: $\vdash (\neg(bf\ f)) = ((\neg f \wedge finite) \vee (df\ (\neg f) \frown skip) \vee (df\ (\neg f) \frown inf))$

using 1 2 by fastforce
 hence 4: $\vdash (bf\ f) = (\neg((\neg f \wedge finite) \vee (df\ (\neg f) \frown skip) \vee (df\ (\neg f) \frown inf)))$ by auto
 have 5: $\vdash (\neg((\neg f \wedge finite) \vee (df\ (\neg f) \frown skip) \vee (df\ (\neg f) \frown inf))) =$
 $((finite \longrightarrow f) \wedge (\neg(df\ (\neg f) \frown skip)) \wedge (\neg(df\ (\neg f) \frown inf)))$ by auto
 have 50: $\vdash (bf\ f) = ((finite \longrightarrow f) \wedge \neg(df\ (\neg f) \frown skip) \wedge \neg(df\ (\neg f) \frown inf))$
 using 4 by auto
 have 6: $\vdash df\ (\neg f) \frown skip = ((\neg(bf\ f)) \frown skip)$ by (simp add: 2 LeftSChopEqvSChop)
 hence 7: $\vdash (\neg(df\ (\neg f) \frown skip)) = (\neg((\neg(bf\ f)) \frown skip))$ by auto
 have 8: $\vdash (\neg((\neg(bf\ f)) \frown skip)) = (empty \vee inf \vee (bf\ f) \frown skip)$ using NotNotSChopSkip by blast
 have 61: $\vdash df\ (\neg f) \frown inf = ((\neg(bf\ f)) \frown inf)$ by (simp add: 2 LeftSChopEqvSChop)
 hence 71: $\vdash (\neg(df\ (\neg f) \frown inf)) = (\neg((\neg(bf\ f)) \frown inf))$ by auto
 have 81: $\vdash (\neg(df\ (\neg f) \frown inf)) = (finite \vee (bf\ (bf\ f)))$
 by (metis 2 NotNotSChopInf int-eq)
 have 82: $\vdash (bf\ (bf\ f)) = bf\ f$
 by (simp add: BfEqvBfBf BfImpBfBf int-iffD2 int-iffI)
 have 83: $\vdash (finite \vee (bf\ (bf\ f))) = (finite \vee bf\ f)$
 using 82 by auto
 hence 9: $\vdash ((finite \longrightarrow f) \wedge \neg(df\ (\neg f) \frown skip) \wedge \neg(df\ (\neg f) \frown inf)) =$
 $((finite \longrightarrow f) \wedge (empty \vee inf \vee (bf\ f) \frown skip) \wedge (finite \vee bf\ f))$
 by (metis 2 8 81 inteq-reflection lift-and-com)
 from 4 5 9 show ?thesis
 by (metis inteq-reflection)
 qed

lemma DiDiAndEqvDi:

$\vdash di\ (di\ f \wedge di\ g) = (di\ f \wedge di\ g)$
 proof –
 have 1: $\vdash bi\ (bi\ (\neg f) \vee bi\ (\neg g)) = (bi\ (\neg f) \vee bi\ (\neg g))$
 by (meson BiBiOrImpBi BiImpBiBiOr BiImpBiBiOrB Prop02 int-iffI)
 have 2: $\vdash bi\ (\neg f) = (\neg (di\ f))$
 by (simp add: bi-d-def)
 have 3: $\vdash bi\ (\neg g) = (\neg (di\ g))$
 by (simp add: bi-d-def)
 have 4: $\vdash (bi\ (\neg f) \vee bi\ (\neg g)) = (\neg (di\ f) \vee \neg (di\ g))$
 using 2 3 by fastforce
 have 5: $\vdash (\neg (di\ f) \vee \neg (di\ g)) = (\neg(di\ f \wedge di\ g))$
 by auto
 have 6: $\vdash bi\ (bi\ (\neg f) \vee bi\ (\neg g)) = (\neg(di\ f \wedge di\ g))$
 using 1 5 4 by fastforce
 hence 7: $\vdash (\neg(bi\ (bi\ (\neg f) \vee bi\ (\neg g)))) = (di\ f \wedge di\ g)$
 by auto
 have 8 : $\vdash (\neg(bi\ (bi\ (\neg f) \vee bi\ (\neg g)))) = di\ (\neg(bi\ (\neg f) \vee bi\ (\neg g)))$
 using DiNotEqvNotBi by fastforce
 have 9 : $\vdash (\neg(bi\ (\neg f) \vee bi\ (\neg g))) = (di\ f \wedge di\ g)$
 using 1 7 by fastforce
 hence 10: $\vdash di\ (\neg(bi\ (\neg f) \vee bi\ (\neg g))) = di\ (di\ f \wedge di\ g)$
 using DiEqvDi by blast
 from 7 8 10 show ?thesis by fastforce
 qed

lemma *DfDfAndEqvDf*:
 $\vdash df (df f \wedge df g) = (df f \wedge df g)$
proof –
have 1: $\vdash bf (bf (\neg f) \vee bf (\neg g)) = (bf (\neg f) \vee bf (\neg g))$
by (*meson BfBfOrImpBf BfImpBfBfOr BfImpBfBfOrB Prop02 int-iffI*)
have 2: $\vdash bf (\neg f) = (\neg (df f))$
by (*simp add: bf-d-def*)
have 3: $\vdash bf (\neg g) = (\neg (df g))$
by (*simp add: bf-d-def*)
have 4: $\vdash (bf (\neg f) \vee bf (\neg g)) = (\neg (df f) \vee \neg (df g))$
using 2 3 **by** *fastforce*
have 5: $\vdash (\neg (df f) \vee \neg (df g)) = (\neg (df f \wedge df g))$
by *auto*
have 6: $\vdash bf (bf (\neg f) \vee bf (\neg g)) = (\neg (df f \wedge df g))$
using 1 5 4 **by** *fastforce*
hence 7: $\vdash (\neg (bf (bf (\neg f) \vee bf (\neg g)))) = (df f \wedge df g)$
by *auto*
have 8: $\vdash (\neg (bf (bf (\neg f) \vee bf (\neg g)))) = df (\neg (bf (\neg f) \vee bf (\neg g)))$
using *DfNotEqvNotBf* **by** *fastforce*
have 9: $\vdash (\neg (bf (\neg f) \vee bf (\neg g))) = (df f \wedge df g)$
using 1 7 **by** *fastforce*
hence 10: $\vdash df (\neg (bf (\neg f) \vee bf (\neg g))) = df (df f \wedge df g)$
using *DfEqvDf* **by** *blast*
from 7 8 10 **show** *?thesis* **by** *fastforce*
qed

lemma *BiInductLen*:
 $\vdash bi(f \longrightarrow wprev f) \wedge f \wedge (len k) \longrightarrow bi f$
proof –
have 1: $\vdash len k \longrightarrow finite$
by (*simp add: len-k-finite*)
have 2: $\vdash bi(f \longrightarrow wprev f) \wedge f \wedge finite \longrightarrow bi f$
by (*simp add: BiInduct*)
have 3: $\vdash finite = (\exists k. len k)$
by (*simp add: Finite-exist-len*)
have 4: $\vdash bi(f \longrightarrow wprev f) \wedge f \wedge (\exists k. len k) \longrightarrow bi f$
by (*metis 2 3 inteq-reflection*)
show *?thesis* **using** 4 **by** *fastforce*
qed

lemma *NotFinitePrevEqv*:
 $\vdash (\neg((f \wedge finite);skip)) = (empty \vee (\neg f \vee inf);skip)$
proof –
have 1: $\vdash (\neg((f \wedge finite);skip)) = (empty \vee (\neg(f \wedge finite));skip)$
by (*simp add: NotChopNotSkip*)
have 2: $\vdash (\neg(f \wedge finite)) = (\neg f \vee inf)$
unfolding *finite-d-def* **by** *fastforce*

have 3: $\vdash (\text{empty} \vee (\neg(f \wedge \text{finite})); \text{skip}) = (\text{empty} \vee (\neg f \vee \text{inf}); \text{skip})$
by (metis 1 2 integ-reflection)
show ?thesis
by (metis 1 3 integ-reflection)
qed

lemma WPrevAndFiniteEqv:
 $\vdash \text{wprev}(f \wedge \text{finite}) = (\text{empty} \vee f \frown \text{skip})$
unfolding wprev-d-def prev-d-def schop-d-def
by (simp add: NotNotChopSkip)

lemma BiInductB:
 $\vdash \text{bi}(f \longrightarrow \text{wprev}(f \wedge \text{finite})) \wedge f \wedge \text{finite} \longrightarrow \text{bi } f$
unfolding Valid-def itl-defs
by simp
 (metis biinduct-help nfinite-conv-nlength-enat)

lemma BfInduct:
 $\vdash \text{bf}(f \longrightarrow \text{wprev } f) \wedge f \wedge \text{finite} \longrightarrow \text{bf } f$
by (auto simp add: Valid-def itl-defs)
 (metis biinduct-help nfinite-conv-nlength-enat)

lemma PrevLoop:
assumes $\vdash f \longrightarrow \text{prev } f$
shows $\vdash \text{finite} \longrightarrow \neg f$
proof –
have 1: $\vdash f \longrightarrow \text{prev } f$ **using** assms **by** auto
hence 2: $\vdash f \longrightarrow (\text{more} \wedge \text{wprev } f)$
by (metis ChopSkipImpMore Prop05 Prop12 WprevEqvEmptyOrPrev integ-reflection lift-imp-trans prev-d-def)
hence 3: $\vdash f \longrightarrow \text{wprev } f$ **by** auto
hence 4: $\vdash \text{bi}(f \longrightarrow \text{wprev } f)$ **by** (rule BiGen)
have 5: $\vdash \text{bi}(f \longrightarrow \text{wprev } f) \wedge f \wedge \text{finite} \longrightarrow \text{bi } f$ **by** (rule BiInduct)
hence 6: $\vdash \text{bi}(f \longrightarrow \text{wprev } f) \longrightarrow (f \wedge \text{finite} \longrightarrow \text{bi } f)$ **by** fastforce
have 7: $\vdash (f \wedge \text{finite} \longrightarrow \text{bi } f)$ **using** 4 6 MP **by** blast
have 8: $\vdash \text{bi } f \longrightarrow f$ **by** (rule BiElim)
have 9: $\vdash \text{finite} \longrightarrow f = \text{bi } f$ **using** 7 8 **by** fastforce
have 10: $\vdash f \longrightarrow \text{more}$ **using** 2 **by** auto
hence 11: $\vdash \text{bi } f \longrightarrow \text{bi more}$ **using** BiImpBiRule **by** blast
have 12: $\vdash \neg(\text{bi more})$ **using** DiEmpty bi-d-def empty-d-def **by** (simp add: bi-d-def empty-d-def)
from 7 9 11 12 **show** ?thesis **using** MP **by** fastforce
qed

lemma FinitePrevLoop:
assumes $\vdash f \longrightarrow \text{prev}(f \wedge \text{finite})$
shows $\vdash \text{finite} \longrightarrow \neg f$
by (metis AndChopA PrevLoop assms lift-imp-trans prev-d-def)

lemma *PrevImpNotPrevNot*:

$\vdash \text{prev } f \longrightarrow \neg(\text{prev } (\neg f))$

by (*metis NotNotChopSkip Prop03 prev-d-def*)

lemma *BiEqvAndWprevBi*:

$\vdash \text{bi } f = (f \wedge \text{wprev}(\text{bi } f))$

by (*metis BiEqvAndEmptyOrBiChopSkip WprevEqvEmptyOrPrev inteq-reflection prev-d-def*)

lemma *DiIntroB*:

assumes $\vdash (f \wedge \neg g) \longrightarrow \text{prev } f$

shows $\vdash f \wedge \text{finite} \longrightarrow \text{di } g$

proof –

have 1: $\vdash f \wedge \neg g \longrightarrow \text{prev } f$

using *assms* **by** *auto*

hence 2: $\vdash f \wedge \neg g \wedge (\text{bi } (\neg g)) \longrightarrow (\text{prev } f) \wedge (\text{bi } (\neg g))$

by *auto*

have 3: $\vdash (\text{bi } (\neg g)) \longrightarrow \neg g$

by (*rule BiElim*)

hence 4: $\vdash \text{bi } (\neg g) = ((\text{bi } (\neg g)) \wedge \neg g)$

by *fastforce*

have 5: $\vdash f \wedge (\text{bi } (\neg g)) \longrightarrow \text{prev } f \wedge \text{bi } (\neg g)$

using 2 4 **by** *fastforce*

have 6: $\vdash \text{bi } (\neg g) = ((\neg g) \wedge \text{wprev}(\text{bi } (\neg g)))$

using *BiEqvAndWprevBi* **by** *blast*

have 7: $\vdash \text{prev } f \wedge \text{bi } (\neg g) \longrightarrow \text{prev } f \wedge \text{wprev}(\text{bi } (\neg g))$

using 6 **by** *auto*

have 8: $\vdash f \wedge (\text{bi } (\neg g)) \longrightarrow \text{prev } f \wedge \text{wprev}(\text{bi } (\neg g))$

using 5 7 **using** *lift-imp-trans* **by** *blast*

hence 9: $\vdash f \wedge (\text{bi } (\neg g)) \longrightarrow \text{more} \wedge \text{wprev } f \wedge \text{wprev}(\text{bi } (\neg g))$

using *zero-enat-def* **by** (*auto simp: Valid-def itl-defs*)

(*metis le-zero-eq nlength-eq-enat-nfiniteD*)

hence 10: $\vdash f \wedge (\text{bi } (\neg g)) \longrightarrow \text{wprev } f \wedge \text{wprev}(\text{bi } (\neg g))$

by *auto*

hence 11: $\vdash f \wedge (\text{bi } (\neg g)) \longrightarrow \text{wprev } (f \wedge \text{bi } (\neg g))$

by (*auto simp: Valid-def itl-defs*)

hence 12: $\vdash \text{bi}(f \wedge (\text{bi } (\neg g))) \longrightarrow \text{wprev } (f \wedge \text{bi } (\neg g))$

by (*rule BiGen*)

have 13: $\vdash \text{bi}(f \wedge (\text{bi } (\neg g))) \longrightarrow \text{wprev } (f \wedge \text{bi } (\neg g)) \wedge$

$(f \wedge (\text{bi } (\neg g))) \wedge \text{finite} \longrightarrow \text{bi}(f \wedge (\text{bi } (\neg g)))$

using *BiInduct* **by** *auto*

hence 14: $\vdash \text{bi}(f \wedge (\text{bi } (\neg g))) \longrightarrow \text{wprev } (f \wedge \text{bi } (\neg g)) \longrightarrow$

$((f \wedge (\text{bi } (\neg g))) \wedge \text{finite} \longrightarrow \text{bi}(f \wedge (\text{bi } (\neg g))))$

by *fastforce*

have 15: $\vdash ((f \wedge (\text{bi } (\neg g))) \wedge \text{finite} \longrightarrow \text{bi}(f \wedge (\text{bi } (\neg g))))$

using 12 14 *MP* **by** *blast*

have 16: $\vdash \text{bi}(f \wedge (\text{bi } (\neg g))) \longrightarrow (f \wedge (\text{bi } (\neg g)))$

by (*rule BiElim*)

```

have 17:  $\vdash \text{finite} \longrightarrow \text{bi}(f \wedge (\neg g)) = (f \wedge (\text{bi}(\neg g)))$ 
  using 16 15 by fastforce
have 18:  $\vdash (f \wedge (\text{bi}(\neg g))) \longrightarrow \text{more}$ 
  using 9 by auto
hence 19:  $\vdash \text{bi}(f \wedge (\text{bi}(\neg g))) \longrightarrow \text{bi more}$ 
  by (simp add: BiImpBiRule)
have 191:  $\vdash (\neg \text{more}) = \text{empty}$ 
  by (simp add: empty-d-def)
have 20:  $\vdash (\neg(\text{bi more}))$ 
  unfolding bi-d-def di-d-def
  by (metis EmptyChop TrueW empty-d-def int-simps(2) int-simps(3) inteq-reflection)
have 21:  $\vdash \text{finite} \longrightarrow \neg(f \wedge (\text{bi}(\neg g)))$ 
  using 17 19 20 by fastforce
hence 22:  $\vdash \text{finite} \longrightarrow \neg f \vee \neg(\text{bi}(\neg g))$ 
  by auto
have 23:  $\vdash (\neg(\text{bi}(\neg g))) = \text{di } g$ 
  by (auto simp: bi-d-def)
show ?thesis using 22 23 by fastforce
qed

```

lemma *DiIntroC*:

```

assumes  $\vdash (f \wedge \neg g) \longrightarrow \text{prev}(f \wedge \text{finite})$ 
shows  $\vdash f \wedge \text{finite} \longrightarrow \text{di } g$ 
using assms
by (metis AndChopA DiIntroB lift-imp-trans prev-d-def)

```

lemma *DfIntroB*:

```

assumes  $\vdash (f \wedge \neg g) \longrightarrow \text{prev } f$ 
shows  $\vdash f \wedge \text{finite} \longrightarrow \text{df } g$ 
proof –
  have 1:  $\vdash f \wedge \text{finite} \longrightarrow \text{di } g$ 
    by (simp add: DiIntroB assms)
  have 2:  $\vdash \text{di } g = (\text{df } g \vee (g \wedge \text{inf}))$ 
    by (simp add: DiEqvDfOrInf)
  have 3:  $\vdash f \wedge \text{finite} \longrightarrow \text{finite} \wedge (\text{df } g \vee (g \wedge \text{inf}))$ 
    by (metis 1 ChopSChopdef FiniteImp df-d-def di-d-def int-eq lift-and-com)
  have 4:  $\vdash \text{finite} \wedge (\text{df } g \vee (g \wedge \text{inf})) \longrightarrow \text{df } g$ 
    unfolding finite-d-def by fastforce
  show ?thesis
  using 3 4 lift-imp-trans by blast
qed

```

lemma *DfIntroC*:

```

assumes  $\vdash (f \wedge \neg g) \longrightarrow \text{prev}(f \wedge \text{finite})$ 
shows  $\vdash f \wedge \text{finite} \longrightarrow \text{df } g$ 
using assms
by (metis AndChopA DfIntroB lift-imp-trans prev-d-def)

```

lemma *DiEqvOrChopMore*:

$\vdash di\ f = (f \vee f;more)$
proof –
have 1: $\vdash di\ f = f; \#True$ **by** (*simp add: di-d-def*)
hence 2: $\vdash di\ f = f; (empty \vee more)$ **by** (*simp add: empty-d-def*)
have 3: $\vdash f; (empty \vee more) = (f;empty \vee f;more)$ **by** (*simp add: ChopOrEqv*)
have 4: $\vdash f;empty = f$ **by** (*rule ChopEmpty*)
from 2 3 4 **show** ?thesis **by** *fastforce*
qed

lemma *DfEqvOrSChopMore*:
 $\vdash df\ f = ((f \wedge finite) \vee f \frown more)$
proof –
have 1: $\vdash df\ f = f \frown \#True$ **by** (*simp add: df-d-def*)
hence 2: $\vdash df\ f = f \frown (empty \vee more)$ **by** (*simp add: empty-d-def*)
have 3: $\vdash f \frown (empty \vee more) = (f \frown empty \vee f \frown more)$ **by** (*simp add: SChopOrEqv*)
have 4: $\vdash f \frown empty = (f \wedge finite)$ **by** (*simp add: ChopEmpty schop-d-def*)
from 2 3 4 **show** ?thesis **by** *fastforce*
qed

lemma *DiAndDiEqvDiAndDiOrDiAndDi*:
 $\vdash (di\ f \wedge di\ g) = (di(f \wedge di\ g) \vee di(g \wedge di\ f))$
unfolding *Valid-def*
apply *auto*
unfolding *itl-defs*
apply *simp*
apply (*metis min.absorb1 nle-le*)
apply *simp*
apply (*metis*)
apply *simp*
apply (*metis min.absorb1*)
apply *simp*
apply (*metis min.absorb1*)
apply *simp*
apply (*metis*)
done

lemma *DfAndDfEqvDiAndDfOrDfAndDf*:
 $\vdash (df\ f \wedge df\ g) = (df(f \wedge df\ g) \vee df(g \wedge df\ f))$
unfolding *Valid-def itl-defs*
using *linorder-le-cases* **by** *fastforce*

lemma *BoxStateEqvBiFinState*:
 $\vdash \Box (init\ w) = bi\ (finite \longrightarrow fin\ (init\ w))$
proof –
have 1: $\vdash \Diamond (\neg (init\ w)) = finite ; (\neg (init\ w))$
by (*simp add: sometimes-d-def*)
have 2: $\vdash \Diamond (init(\neg w)) = finite ; init\ (\neg w)$
by (*simp add: sometimes-d-def*)

have 3: $\vdash di (finite \wedge fin (init (\neg w))) = finite ; init (\neg w)$
by (*metis DiAndFinEqvChopState NowImpDiamond Prop10 inteq-reflection lift-and-com*)
have 4: $\vdash \diamond (init(\neg w)) = di (finite \wedge fin (init (\neg w)))$
using 1 2 3 **by** *fastforce*
have 5: $\vdash (\neg (\diamond (init(\neg w)))) = (\neg (di (finite \wedge fin (init (\neg w)))))$
using 4 **by** *fastforce*
have 51: $\vdash (\neg (finite \wedge fin (init (\neg w)))) = (finite \longrightarrow \neg (fin (init (\neg w))))$
by *fastforce*
have 6: $\vdash \square (init w) = (\neg (di (finite \wedge fin (init (\neg w)))))$
using 5 *always-d-def Initprop(2)* **by** (*metis int-eq*)
have 7: $\vdash \square (init w) = bi (finite \longrightarrow \neg (fin (init (\neg w))))$
using 6 51 **unfolding** *bi-d-def*
by (*metis int-simps(4) inteq-reflection*)
have 8: $\vdash init (\neg w) = (\neg (init w))$
using *Initprop(2)* **by** *fastforce*
have 9: $\vdash fin (init (\neg w)) = fin (\neg (init w))$
using 8 *FinEqvFin* **by** *blast*
have 10: $\vdash finite \longrightarrow_{fin} (init (\neg w)) = (\neg (fin (init w)))$
using 8 *FinNotStateEqvNotFinState FinEqvFin* **by** *fastforce*
have 11: $\vdash finite \longrightarrow ((\neg (fin (init (\neg w)))) = (fin (init w)))$
using 10 **by** *fastforce*
have 111: $\vdash (finite \longrightarrow \neg (fin (init (\neg w)))) = (finite \longrightarrow_{fin} (init w))$
using 11 **by** *fastforce*
have 12: $\vdash bi (finite \longrightarrow \neg (fin (init (\neg w)))) = bi (finite \longrightarrow_{fin} (init w))$
using 111 **by** (*simp add: BiEqvBi*)
have 13: $\vdash \square (init w) = bi (finite \longrightarrow_{fin} (init w))$
using 7 12 **by** *fastforce*
from 13 **show** *?thesis* **by** *simp*
qed

lemma *BoxStateEqvBfFinState*:

$\vdash \square (init w) = bf (fin (init w))$

proof –

have 1: $\vdash bi (finite \longrightarrow_{fin} (init w)) =$
 $(bf (finite \longrightarrow_{fin} (init w)) \wedge (inf \longrightarrow finite \longrightarrow_{fin} (init w)))$
by (*simp add: BiEqvBfAndfinite*)
have 2: $\vdash (bf (finite \longrightarrow_{fin} (init w)) \wedge (inf \longrightarrow finite \longrightarrow_{fin} (init w))) =$
 $bf (finite \longrightarrow_{fin} (init w))$
unfolding *finite-d-def* **by** *fastforce*
have 3: $\vdash \square (init w) = bi (finite \longrightarrow fin (init w))$
by (*simp add: BoxStateEqvBiFinState*)
have 4: $\vdash bf (finite \longrightarrow_{fin} (init w)) = bf (fin (init w))$
by (*simp add: FiniteImpBfEqvRule intI*)
show *?thesis*
by (*metis 1 2 3 4 inteq-reflection*)

qed

lemma *DiamondStateEqvDiFinState*:

$\vdash \diamond (init w) = di (finite \wedge fin (init w))$

proof –
have 1: $\vdash \Box (init (\neg w)) = bi (finite \longrightarrow fin (init (\neg w)))$
using *BoxStateEqvBiFinState* **by** *blast*
have 2: $\vdash (\neg (\Box (init (\neg w)))) = (\neg (bi (finite \longrightarrow fin (init (\neg w)))))$
using 1 **by** *auto*
have 3: $\vdash \Diamond (\neg (init (\neg w))) = di (finite \wedge \neg (fin (init (\neg w))))$
using 2
by (*metis* (*no-types*, *opaque-lifting*) *FinChopEqvDiamond* *FinNotStateEqvNotFinState* *Initprop*(2) *di-d-def* *int-simps*(17) *inteq-reflection* *lift-and-com*)
have 4: $\vdash \Diamond (init w) = di (finite \wedge \neg (fin (init (\neg w))))$
by (*metis* 3 *DiEqvNotBiNot* *DiState* *Initprop*(2) *StateEqvBi* *int-eq*)
have 5: $\vdash \Diamond (init w) = di (finite \wedge fin (init w))$ **using** 4 *FinNotStateEqvNotFinState*
by (*metis* *int-simps*(4) *inteq-reflection* *lift-and-com*)
from 1 2 3 4 5 **show** *?thesis* **by** *simp*
qed

lemma *DiamondStateEqvDfFinState*:
 $\vdash \Diamond (init w) = df (fin (init w))$
by (*metis* *DiamondStateEqvDiFinState* *df-d-def* *di-d-def* *inteq-reflection* *lift-and-com* *schop-d-def*)

lemma *OrDiEqvDi*:
 $\vdash (f \vee di f) = di f$
by (*simp* *add*: *Prop05* *Prop11* *DiIntro*)

lemma *OrDfEqvDf*:
 $\vdash ((f \wedge finite) \vee df f) = df f$
by (*simp* *add*: *AndFiniteImpDf* *Prop05* *Prop11*)

lemma *AndDiEqv*:
 $\vdash (f \wedge di f) = f$

proof –
have 1: $\vdash f \longrightarrow di f$ **using** *DiIntro* **by** *blast*
from 1 **show** *?thesis* **by** *auto*
qed

lemma *AndDfEqv*:
 $\vdash ((f \wedge finite) \wedge df f) = (f \wedge finite)$
by (*meson* *AndFiniteImpDf* *Prop10* *Prop11*)

lemma *BiEmptyEqvEmpty*:
 $\vdash bi\ empty = empty$

proof –
have 1: $\vdash bi\ empty = (\neg (di (\neg empty)))$ **by** (*simp* *add*: *bi-d-def*)
have 2: $\vdash (\neg (di (\neg empty))) = (\neg ((\neg empty); \# True))$ **by** (*simp* *add*: *di-d-def*)
have 3: $\vdash (\neg ((\neg empty); \# True)) = (\neg (more; \# True))$ **by** (*simp* *add*: *empty-d-def*)
have 4: $\vdash more; \# True = more$ **using** *MoreEqvMoreChopTrue* **by** *auto*
hence 5: $\vdash (\neg (more; \# True)) = (\neg more)$ **by** *fastforce*

from 1 2 3 5 show ?thesis by (metis empty-d-def inteq-reflection)
qed

lemma BfEmptyEqvEmpty:

$\vdash \text{bf empty} = \text{empty}$

by (metis MoreEqvMoreSChopTrue empty-d-def int-eq int-simps(4) itl-def(15) itl-def(23))

lemma EmptyChopSkipInduct:

assumes $\vdash \text{empty} \longrightarrow f$

$\vdash \text{prev } f \longrightarrow f$

shows $\vdash \text{finite} \longrightarrow f$

proof –

have 1: $\vdash \text{empty} \longrightarrow f$ **using** *assms(1)* **by** *auto*

have 2: $\vdash \text{prev } f \longrightarrow f$ **using** *assms(2)* **by** *blast*

have 3: $\vdash (\text{empty} \vee \text{prev } f) \longrightarrow f$ **using** 1 2 **by** *fastforce*

have 4: $\vdash \text{wprev } f = (\text{empty} \vee \text{prev } f)$ **by** (*simp add: WprevEqvEmptyOrPrev*)

hence 5: $\vdash \text{wprev } f \longrightarrow f$ **using** 3 **by** *fastforce*

hence 6: $\vdash \neg f \longrightarrow \neg (\text{wprev } f)$ **by** *fastforce*

hence 7: $\vdash \neg f \longrightarrow \text{prev } (\neg f)$ **by** (*simp add: wprev-d-def*)

hence 8: $\vdash \text{finite} \longrightarrow \neg \neg f$ **by** (*rule PrevLoop*)

from 8 show ?thesis by auto

qed

lemma PrevOrInfEqv:

$\vdash \text{prev } (f \vee \text{inf}) = (\text{prev } (f \wedge \text{finite}) \vee \text{inf})$

proof –

have 1: $\vdash \text{prev } (f \vee \text{inf}) = (f \vee \text{inf}); \text{skip}$

unfolding *prev-d-def* **by** *auto*

have 2: $\vdash (f \vee \text{inf}); \text{skip} = (f; \text{skip} \vee \text{inf}; \text{skip})$

by (*simp add: OrChopEqv*)

have 3: $\vdash f; \text{skip} = ((f \wedge \text{finite}); \text{skip} \vee (f \wedge \text{inf}); \text{skip})$

by (*simp add: OrChopEqvRule OrFiniteInf*)

have 4: $\vdash (f \wedge \text{inf}); \text{skip} = (f \wedge \text{inf})$

by (*simp add: AndInfChopEqvAndInf*)

have 5: $\vdash \text{inf}; \text{skip} = \text{inf}$

by (*metis AndInfChopEqvAndInf MoreAndInfEqvInf int-eq*)

have 6: $\vdash ((f \wedge \text{inf}) \vee \text{inf}) = (\text{inf})$

by *fastforce*

have 7: $\vdash (f; \text{skip} \vee \text{inf}; \text{skip}) = ((f \wedge \text{finite}); \text{skip} \vee (f \wedge \text{inf}) \vee \text{inf})$

using 3 4 5 6 **by** *fastforce*

have 8: $\vdash ((f \wedge \text{finite}); \text{skip} \vee (f \wedge \text{inf}) \vee \text{inf}) =$

$((f \wedge \text{finite}); \text{skip} \vee \text{inf})$

using 6 **by** *fastforce*

show ?thesis

by (*metis 2 7 8 inteq-reflection prev-d-def*)

qed

lemma EmptySChopSkipInduct:

assumes $\vdash \text{empty} \longrightarrow f$

$\vdash \text{prev } (f \vee \text{inf}) \longrightarrow f$
shows $\vdash \text{finite} \longrightarrow f$
proof –
have 1: $\vdash \text{empty} \longrightarrow f$ **using** *assms(1)* **by** *auto*
have 2: $\vdash \text{prev } (f \vee \text{inf}) \longrightarrow f$ **using** *assms(2)* **by** *blast*
have 3: $\vdash (\text{empty} \vee \text{prev } (f \vee \text{inf})) \longrightarrow f$ **using** 1 2 **by** *fastforce*
have 4: $\vdash \text{wprev } (f \vee \text{inf}) = (\text{empty} \vee \text{prev } (f \vee \text{inf}))$ **by** (*simp add: WprevEqvEmptyOrPrev*)
hence 5: $\vdash \text{wprev } (f \vee \text{inf}) \longrightarrow f$ **using** 3 **by** *fastforce*
hence 6: $\vdash \neg f \longrightarrow \neg (\text{wprev } (f \vee \text{inf}))$ **by** *fastforce*
hence 7: $\vdash \neg f \longrightarrow \text{prev } (\neg (f \vee \text{inf}))$ **by** (*simp add: wprev-d-def*)
have 8: $\vdash (\neg (f \vee \text{inf})) = (\neg f \wedge \text{finite})$ **unfolding** *finite-d-def* **by** *fastforce*
hence 9: $\vdash \text{finite} \longrightarrow \neg \neg f$ **using** *FinitePrevLoop[of LIFT $\neg f$]*
by (*metis 7 inteq-reflection*)
from 9 **show** *?thesis* **by** *auto*
qed

lemma *EmptySChopSkipInductB*:
assumes $\vdash \text{empty} \longrightarrow f$
 $\vdash \text{prev } (f \wedge \text{finite}) \longrightarrow f$
shows $\vdash \text{finite} \longrightarrow f$
by (*metis EmptyChopSkipInduct EmptyImpFinite Prop12 SChopImpFinite WPowerstar-ext*
WPowerstar-skip-finite assms(1) assms(2) int-eq itl-def(14) itl-def(9))

lemma *MoreImpImpChopSkipEqv*:
 $\vdash \text{more} \longrightarrow ((f \longrightarrow g); \text{skip} = ((f; \text{skip}) \longrightarrow (g; \text{skip})))$
proof –
have 01: $\vdash (f \longrightarrow g) = (\neg f \vee g)$ **by** *auto*
hence 02: $\vdash (f \longrightarrow g); \text{skip} = (\neg f \vee g); \text{skip}$ **by** (*simp add: LeftChopEqvChop*)
hence 1: $\vdash (\text{more} \wedge (f \longrightarrow g); \text{skip}) = (\text{more} \wedge (\neg f \vee g); \text{skip})$ **by** *fastforce*
have 2: $\vdash (\neg f \vee g); \text{skip} = ((\neg f); \text{skip} \vee g; \text{skip})$
using *OrChopEqv* **by** *auto*
hence 3: $\vdash (\text{more} \wedge (\neg f \vee g); \text{skip}) = (\text{more} \wedge ((\neg f); \text{skip} \vee g; \text{skip}))$
by *auto*
have 4: $\vdash (\neg((\neg f); \text{skip})) = (\text{empty} \vee (f; \text{skip}))$
using *NotNotChopSkip* **by** *blast*
hence 5: $\vdash ((\neg f); \text{skip}) = (\neg(\text{empty} \vee (f; \text{skip})))$
by *fastforce*
have 6: $\vdash \neg(\text{empty} \vee (f; \text{skip})) = (\text{more} \wedge \neg(f; \text{skip}))$
using 5 *NotChopSkipEqvMoreAndNotChopSkip* **by** *fastforce*
have 7: $\vdash ((\neg f); \text{skip} \vee g; \text{skip}) = ((\text{more} \wedge \neg(f; \text{skip})) \vee g; \text{skip})$
using 5 6 **by** *fastforce*
hence 8: $\vdash (\text{more} \wedge (\neg f; \text{skip} \vee g; \text{skip})) = (\text{more} \wedge ((\text{more} \wedge \neg(f; \text{skip})) \vee g; \text{skip}))$
by *auto*
have 9: $\vdash (\text{more} \wedge ((\text{more} \wedge \neg(f; \text{skip})) \vee g; \text{skip})) = (\text{more} \wedge (\neg(f; \text{skip}) \vee g; \text{skip}))$
by *auto*
have 10: $\vdash (\text{more} \wedge (\neg(f; \text{skip}) \vee g; \text{skip})) = (\text{more} \wedge ((f; \text{skip}) \longrightarrow (g; \text{skip})))$
by *auto*
have 11: $\vdash (\text{more} \wedge (f \longrightarrow g); \text{skip}) = (\text{more} \wedge ((f; \text{skip}) \longrightarrow (g; \text{skip})))$

using 1 2 3 8 9 10 7 by fastforce
 from 11 show ?thesis using MP by fastforce
 qed

lemma *MoreImpImpSChopSkipEqv*:

$\vdash \text{more} \wedge \text{finite} \longrightarrow ((f \longrightarrow g) \frown \text{skip} = ((f \frown \text{skip}) \longrightarrow (g \frown \text{skip})))$

proof –

have 01: $\vdash (f \longrightarrow g) = (\neg f \vee g)$ **by** *auto*

hence 02: $\vdash (f \longrightarrow g) \frown \text{skip} = (\neg f \vee g) \frown \text{skip}$ **by** (*simp add: LeftSChopEqvSChop*)

have 03: $\vdash (f \longrightarrow g) \frown \text{skip} \longrightarrow \text{finite}$

by (*metis LenOneEqvSkip SChopImpFinite inteq-reflection len-k-finite*)

have 04: $\vdash (\text{more} \wedge (f \longrightarrow g) \frown \text{skip}) = (\text{more} \wedge \text{finite} \wedge (f \longrightarrow g) \frown \text{skip})$

using 03 **by** *auto*

have 1: $\vdash (\text{more} \wedge \text{finite} \wedge (f \longrightarrow g) \frown \text{skip}) = (\text{more} \wedge \text{finite} \wedge (\neg f \vee g) \frown \text{skip})$

by (*metis 01 04 inteq-reflection*)

have 2: $\vdash (\neg f \vee g) \frown \text{skip} = ((\neg f) \frown \text{skip} \vee g \frown \text{skip})$

using *OrSChopEqv* **by** *auto*

hence 3: $\vdash (\text{more} \wedge \text{finite} \wedge (\neg f \vee g) \frown \text{skip}) = (\text{more} \wedge \text{finite} \wedge ((\neg f) \frown \text{skip} \vee g \frown \text{skip}))$

by *auto*

have 4: $\vdash \neg((\neg f) \frown \text{skip}) = (\text{empty} \vee \text{inf} \vee (f \frown \text{skip}))$

using *NotNotSChopSkip* **by** *blast*

hence 5: $\vdash ((\neg f) \frown \text{skip}) = \neg(\text{empty} \vee \text{inf} \vee (f \frown \text{skip}))$

by *fastforce*

have 6: $\vdash \neg(\text{empty} \vee \text{inf} \vee (f \frown \text{skip})) = (\text{more} \wedge \text{finite} \wedge \neg(f \frown \text{skip}))$

using 5 **unfolding** *finite-d-def empty-d-def* **by** *fastforce*

have 7: $\vdash ((\neg f) \frown \text{skip} \vee g \frown \text{skip}) = ((\text{more} \wedge \text{finite} \wedge \neg(f \frown \text{skip})) \vee g \frown \text{skip})$

using 5 6 **by** *fastforce*

hence 8: $\vdash (\text{more} \wedge \text{finite} \wedge ((\neg f) \frown \text{skip} \vee g \frown \text{skip})) =$

$(\text{more} \wedge \text{finite} \wedge ((\text{more} \wedge \text{finite} \wedge \neg(f \frown \text{skip})) \vee g \frown \text{skip}))$

by *fastforce*

have 9: $\vdash (\text{more} \wedge \text{finite} \wedge ((\text{more} \wedge \text{finite} \wedge \neg(f \frown \text{skip})) \vee g \frown \text{skip})) =$

$(\text{more} \wedge \text{finite} \wedge (\neg(f \frown \text{skip}) \vee g \frown \text{skip}))$

by *auto*

have 10: $\vdash (\text{more} \wedge \text{finite} \wedge (\neg(f \frown \text{skip}) \vee g \frown \text{skip})) =$

$(\text{more} \wedge \text{finite} \wedge ((f \frown \text{skip}) \longrightarrow (g \frown \text{skip})))$

by *auto*

have 11: $\vdash (\text{more} \wedge \text{finite} \wedge (f \longrightarrow g) \frown \text{skip}) = (\text{more} \wedge \text{finite} \wedge ((f \frown \text{skip}) \longrightarrow (g \frown \text{skip})))$

using 1 2 3 8 9 10 7

by (*metis inteq-reflection*)

from 11 **show** ?thesis **using** MP **by** *fastforce*

qed

lemma *MoreImpImpPrevEqv*:

$\vdash \text{more} \longrightarrow (\text{prev}(f \longrightarrow g) = (\text{prev } f \longrightarrow \text{prev } g))$

by (*simp add: MoreImpImpChopSkipEqv prev-d-def*)

lemma *BiBoxNotEqvNotFiniteChopChopTrue*:

$\vdash \text{bi}(\Box (\neg f)) = (\neg((\text{finite}; f); \# \text{True}))$

by (*simp add: bi-d-def always-d-def di-d-def sometimes-d-def*)

lemma *BfBoxNotEqvNotTrueSChopSChopTrue*:

$\vdash bf(\Box (\neg f)) = (\neg((\#True \frown f) \frown \#True))$

by (*simp add: bf-d-def always-d-def df-d-def sometimes-d-def schop-d-def*)

lemma *DiAndEmptyEqvAndEmpty*:

$\vdash (di\ f \wedge empty) = (f \wedge empty)$

by (*metis ChopEmptyAndEmpty di-d-def int-simps(17) inteq-reflection*)

lemma *DfAndEmptyEqvAndEmpty*:

$\vdash (df\ f \wedge empty) = (f \wedge empty)$

by (*metis ChopEmptyAndEmpty DiAndEmptyEqvAndEmpty Prop04 SChopEmptyAndEmpty df-d-def di-d-def*)

lemma *FiniteImpBfEqvBi*:

$\vdash finite \longrightarrow bf\ f = bi\ f$

proof –

have 1: $\vdash bi\ f = (bf\ f \wedge (inf \longrightarrow f))$

by (*simp add: BiEqvBfAndfinite*)

have 2: $\vdash finite \longrightarrow (bf\ f \wedge (inf \longrightarrow f)) = bf\ f$

unfolding *finite-d-def* **by** *fastforce*

show *?thesis*

using 1 2 **by** *fastforce*

qed

lemma *FiniteImpDfEqvDi*:

$\vdash finite \longrightarrow df\ f = di\ f$

proof –

have 1: $\vdash di\ f = (df\ f \vee (f \wedge inf))$

by (*simp add: DiEqvDfOrInf*)

have 2: $\vdash finite \longrightarrow (df\ f \vee (f \wedge inf)) = df\ f$

unfolding *finite-d-def* **by** *fastforce*

show *?thesis* **using** 1 2 **by** *fastforce*

qed

Strict initial intervals

lemma *BsEqvEmptyOrBiSChopSkip*:

$\vdash bs\ f = (empty \vee ((bi\ f) \frown skip))$

unfolding *bs-d-def* **by** *simp*

lemma *DsMoreDi*:

$\vdash ds\ f = (more \wedge (finite \longrightarrow (di\ f) \frown skip))$

proof –

have 1: $\vdash ds\ f = (\neg(bs\ (\neg f)))$

by (*simp add: ds-d-def*)

have 2: $\vdash (\neg(bs\ (\neg f))) = (\neg(empty \vee (bi\ (\neg f) \frown skip)))$

by (*simp add: bs-d-def*)
have 3: $\vdash (\neg(\text{empty} \vee (\text{bi } (\neg f)) \frown \text{skip})) = (\neg \text{empty} \wedge \neg((\text{bi } (\neg f)) \frown \text{skip}))$
 by *auto*
have 4: $\vdash (\neg \text{empty} \wedge \neg((\text{bi } (\neg f)) \frown \text{skip})) = (\text{more} \wedge \neg((\text{bi } (\neg f)) \frown \text{skip}))$
 unfolding *empty-d-def* by *auto*
have 40: $\vdash (\text{bi } (\neg f)) = (\neg(\text{di } f))$
 by (*meson NotDiEqvBiNot Prop11*)
have 41: $\vdash (\neg((\text{bi } (\neg f)) \frown \text{skip})) = (\text{empty} \vee \text{inf} \vee ((\text{di } f) \frown \text{skip}))$
 using 40 *NotNotSChopSkip[of LIFT (di f)]* by (*metis inteq-reflection*)
have 5: $\vdash (\text{more} \wedge \neg((\text{bi } (\neg f)) \frown \text{skip})) = (\text{more} \wedge (\text{empty} \vee \text{inf} \vee (\text{di } f) \frown \text{skip}))$
 using 41 by *auto*
have 6: $\vdash (\text{more} \wedge (\text{empty} \vee \text{inf} \vee (\text{di } f) \frown \text{skip})) =$
 $(\text{more} \wedge (\text{inf} \vee (\text{di } f) \frown \text{skip}))$
 unfolding *empty-d-def* by *fastforce*
show ?thesis unfolding *finite-d-def* using 1 2 3 4 5
 by *fastforce*
qed

lemma *DsDi*:

$\vdash \text{ds } f = (\text{finite} \longrightarrow (\text{di } f) \frown \text{skip})$
proof –
have 1: $\vdash \text{ds } f = (\text{more} \wedge (\text{finite} \longrightarrow (\text{di } f) \frown \text{skip}))$ by (*rule DsMoreDi*)
have 2: $\vdash (\text{di } f) \frown \text{skip} \longrightarrow \text{more}$ by (*simp add: SChopSkipImpMore*)
hence 3: $\vdash (\text{more} \wedge (\text{finite} \longrightarrow (\text{di } f) \frown \text{skip})) =$
 $(\text{more} \wedge \text{inf}) \vee (\text{more} \wedge (\text{di } f) \frown \text{skip})$
 unfolding *finite-d-def* by *fastforce*
have 4: $\vdash ((\text{more} \wedge \text{inf}) \vee (\text{more} \wedge (\text{di } f) \frown \text{skip})) =$
 $(\text{inf} \vee (\text{di } f) \frown \text{skip})$
 by (*metis 2 3 MoreAndInfEqvInf Prop10 inteq-reflection lift-and-com*)
from 1 2 4 **show** ?thesis unfolding *finite-d-def*
 by *fastforce*
qed

lemma *DsDf*:

$\vdash \text{ds } f = (\text{finite} \longrightarrow (\text{df } f) \frown \text{skip})$
proof –
have 1: $\vdash \text{finite} \longrightarrow (\text{di } f) = (\text{df } f)$
 using *FiniteImpDfEqvDi* by *fastforce*
have 2: $\vdash \text{ds } f = (\text{finite} \longrightarrow (\text{di } f) \frown \text{skip})$
 by (*simp add: DsDi*)
have 3: $\vdash (\text{finite} \longrightarrow (\text{di } f) \frown \text{skip}) = (\text{finite} \longrightarrow (\text{df } f) \frown \text{skip})$
 by (*metis 1 2 FiniteImpAnd inteq-reflection schop-d-def*)
show ?thesis
 by (*metis 2 3 inteq-reflection*)
qed

lemma *BsEqvNotDsNot*:

$\vdash \text{bs } f = (\neg(\text{ds } (\neg f)))$
proof –
have 1: $\vdash \text{ds } (\neg f) = (\text{more} \wedge (\text{finite} \longrightarrow \text{di } (\neg f) \frown \text{skip}))$

using *DsMoreDi*[of *LIFT* $\neg f$] by *blast*
 hence 2: $\vdash (\neg(ds (\neg f))) = (\neg(more \wedge (finite \longrightarrow di (\neg f) \frown skip)))$
 by *auto*
 have 3: $\vdash (\neg(more \wedge (finite \longrightarrow di (\neg f) \frown skip))) =$
 $(empty \vee \neg(finite \longrightarrow (di (\neg f) \frown skip)))$
 unfolding *empty-d-def* by *fastforce*
 have 31: $\vdash di (\neg f) = (\neg(bi f))$
 by (*simp add: DiNotEqvNotBi*)
 have 4: $\vdash (empty \vee \neg(finite \longrightarrow (di (\neg f) \frown skip))) =$
 $(empty \vee (finite \wedge \neg((di (\neg f) \frown skip))))$
 by *fastforce*
 have 41: $\vdash (\neg((di (\neg f) \frown skip))) = (\neg(\neg(bi f) \frown skip))$
 by (*metis 31 int-simps(15) inteq-reflection*)
 have 5: $\vdash (\neg(\neg(bi f) \frown skip)) = (empty \vee inf \vee (bi f) \frown skip)$
 by (*rule NotNotSChopSkip*)
 have 51: $\vdash (empty \vee (finite \wedge \neg((di (\neg f) \frown skip)))) =$
 $(empty \vee (finite \wedge (empty \vee inf \vee (bi f) \frown skip)))$
 by (*metis 31 4 5 inteq-reflection*)
 have 52: $\vdash (finite \wedge (empty \vee inf \vee (bi f) \frown skip)) =$
 $(finite \wedge (empty \vee (bi f) \frown skip))$
 unfolding *finite-d-def* by *fastforce*
 have 53: $\vdash (finite \wedge (empty \vee (bi f) \frown skip)) =$
 $(empty \vee (finite \wedge (bi f) \frown skip))$
 by (*metis (no-types, opaque-lifting) ChopEmpty EmptyOrSChopEqv EmptySChop*
FiniteImportSChopRight int-eq itl-def(9))
 have 54: $\vdash (finite \wedge (bi f) \frown skip) = (bi f) \frown skip$
 by (*meson Prop10 Prop11 SChopImpFinite WPowerstar-ext WPowerstar-skip-finite lift-and-com lift-imp-trans*)
 have 55: $\vdash (empty \vee (finite \wedge (empty \vee inf \vee (bi f) \frown skip))) =$
 $(empty \vee (bi f) \frown skip)$
 using 52 54 by *fastforce*
 have 6: $\vdash (empty \vee (finite \wedge \neg((di (\neg f) \frown skip)))) =$
 $(empty \vee (bi f) \frown skip)$
 by (*metis 51 55 inteq-reflection*)
 from 2 3 4 6 show ?thesis
 by (*simp add: ds-d-def*)
 qed

lemma *NotBsEqvDsNot*:

$\vdash (\neg(bs f)) = ds (\neg f)$

proof –

have 1: $\vdash bs f = (\neg(ds (\neg f)))$ by (*rule BsEqvNotDsNot*)

hence 2: $\vdash (\neg(bs f)) = (\neg\neg(ds (\neg f)))$ by *auto*

from 2 show ?thesis by *auto*

qed

lemma *NotDsEqvBsNot*:

$\vdash (\neg(ds f)) = bs (\neg f)$

proof –

have 1: $\vdash (\neg(ds f)) = (\neg\neg(bs (\neg f)))$ by (*simp add: ds-d-def*)

from 1 show ?thesis by *auto*

qed

lemma *NotDsAndEmpty*:

$\vdash \neg(ds\ f \wedge empty)$

proof –

have 1: $\vdash ds\ f = (more \wedge (finite \longrightarrow (di\ f) \frown skip))$ **by** (*rule DsMoreDi*)

have 2: $\vdash (more \wedge (finite \longrightarrow (di\ f) \frown skip)) \wedge empty \longrightarrow \#False$

unfolding *empty-d-def* **by** *fastforce*

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *BsMoreEqvEmpty*:

$\vdash bs\ more = empty$

proof –

have 1: $\vdash bs\ more = (empty \vee (bi\ more) \frown skip)$

by (*simp add: bs-d-def*)

have 2: $\vdash bi\ more \longrightarrow \#False$

by (*metis EmptyChop TrueW bi-d-def di-d-def empty-d-def int-simps(14) int-simps(4) inteq-reflection*)

hence 3: $\vdash (bi\ more) \frown skip \longrightarrow \#False \frown skip$

using *LeftSChopImpSChop* **by** *blast*

have 31: $\vdash \#False \frown skip \longrightarrow \#False$

by (*simp add: Valid-def skip-defs schop-defs*)

have 32: $\vdash (bi\ more) \frown skip \longrightarrow \#False$ **using** 3 31 **by** *fastforce*

hence 4: $\vdash (empty \vee ((bi\ more) \frown skip)) = empty$ **by** *fastforce*

from 1 4 **show** *?thesis* **by** *fastforce*

qed

lemma *BsAndEqv*:

$\vdash (bs\ f \wedge bs\ g) = bs(f \wedge g)$

proof –

have 1: $\vdash bs\ f = (empty \vee (bi\ f) \frown skip)$

by (*simp add: bs-d-def*)

have 2: $\vdash bs\ g = (empty \vee (bi\ g) \frown skip)$

by (*simp add: bs-d-def*)

have 3: $\vdash (bs\ f \wedge bs\ g) = ((empty \vee (bi\ f) \frown skip) \wedge (empty \vee (bi\ g) \frown skip))$

using 1 2 **by** *fastforce*

have 4: $\vdash ((empty \vee (bi\ f) \frown skip) \wedge (empty \vee (bi\ g) \frown skip)) =$
 $(empty \vee ((bi\ f) \frown skip \wedge (bi\ g) \frown skip))$

by *auto*

have 5: $\vdash (((bi\ f) \frown skip \wedge (bi\ g) \frown skip)) = bi(f \wedge g) \frown skip$

by (*metis BiAndEqv SChopSkipAndSChopSkip inteq-reflection*)

hence 6: $\vdash (empty \vee ((bi\ f) \frown skip \wedge (bi\ g) \frown skip)) = (empty \vee bi(f \wedge g) \frown skip)$

by *auto*

from 3 4 6 **show** *?thesis* **by** (*metis bs-d-def inteq-reflection*)

qed

lemma *BsEqvRule*:

assumes $\vdash f = g$

shows $\vdash bs\ f = bs\ g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash bi(f) = bi(g)$ **by** (*simp add: BiEqvBi*)
hence 3: $\vdash bi(f) \frown skip = bi(g) \frown skip$ **by** (*simp add: LeftSChopEqvSChop*)
hence 4: $\vdash (empty \vee bi(f) \frown skip) = (empty \vee bi(g) \frown skip)$ **by** *auto*
hence 5: $\vdash bs(f) = bs(g)$ **by** (*simp add: bs-d-def*)
from 1 2 3 4 5 **show** *?thesis* **by** *auto*
qed

lemma *DsEqvRule*:
assumes $\vdash f = g$
shows $\vdash ds\ f = ds\ g$
using *assms* **using** *int-eq* **by** *force*

lemma *DsOrEqv*:
 $\vdash (ds\ f \vee ds\ g) = ds\ (f \vee g)$
proof –
have 1: $\vdash ds\ f = (\neg(bs\ (\neg f)))$ **by** (*simp add: ds-d-def*)
have 2: $\vdash ds\ g = (\neg(bs\ (\neg g)))$ **by** (*simp add: ds-d-def*)
have 3: $\vdash (ds\ f \vee ds\ g) = (\neg(bs\ (\neg f)) \vee \neg(bs\ (\neg g)))$ **using** 1 2 **by** *fastforce*
have 4: $\vdash (\neg(bs\ (\neg f)) \vee \neg(bs\ (\neg g))) = (\neg(bs\ (\neg f) \wedge bs\ (\neg g)))$ **by** *auto*
have 5: $\vdash (bs\ (\neg f) \wedge bs\ (\neg g)) = bs\ (\neg f \wedge \neg g)$ **by** (*rule BsAndEqv*)
hence 6: $\vdash (\neg(bs\ (\neg f) \wedge bs\ (\neg g))) = (\neg(bs\ (\neg f \wedge \neg g)))$ **by** *auto*
have 7: $\vdash (\neg(bs\ (\neg f \wedge \neg g))) = ds\ (\neg(\neg f \wedge \neg g))$ **by** (*rule NotBsEqvDsNot*)
have 8: $\vdash (\neg(\neg f \wedge \neg g)) = (f \vee g)$ **by** *auto*
hence 9: $\vdash ds(\neg(\neg f \wedge \neg g)) = ds\ (f \vee g)$ **by** (*rule DsEqvRule*)
from 3 4 6 7 9 **show** *?thesis* **by** *fastforce*
qed

lemma *BsOrImp*:
 $\vdash bs\ f \vee bs\ g \longrightarrow bs(f \vee g)$
proof –
have 1: $\vdash bi\ f \vee bi\ g \longrightarrow bi(f \vee g)$
by (*rule BiOrBiImpBiOr*)
hence 2: $\vdash (bi\ f \vee bi\ g) \frown skip \longrightarrow (bi(f \vee g)) \frown skip$
by (*rule LeftSChopImpSChop*)
have 3: $\vdash (bi\ f) \frown skip \vee (bi\ g) \frown skip \longrightarrow (bi(f \vee g)) \frown skip$
using 1 *OrSChopEqv* 2 **by** *fastforce*
hence 4: $\vdash empty \vee (bi\ f) \frown skip \vee (bi\ g) \frown skip \longrightarrow empty \vee (bi(f \vee g)) \frown skip$
by *auto*
hence 5: $\vdash (empty \vee (bi\ f) \frown skip) \vee (empty \vee (bi\ g) \frown skip) \longrightarrow empty \vee (bi(f \vee g)) \frown skip$
by *auto*
from 5 **show** *?thesis* **by** (*simp add: bs-d-def*)
qed

lemma *DsAndImp*:
 $\vdash ds\ (f \wedge g) \longrightarrow ds\ f \wedge ds\ g$
proof –
have 1: $\vdash bs\ (\neg f) \vee bs\ (\neg g) \longrightarrow bs(\neg f \vee \neg g)$ **by** (*rule BsOrImp*)

have 2: $\vdash (\neg f \vee \neg g) = (\neg(f \wedge g))$ **by** *auto*
hence 3: $\vdash bs(\neg f \vee \neg g) = bs(\neg(f \wedge g))$ **by** (*rule BsequivRule*)
have 4: $\vdash bs(\neg f) \vee bs(\neg g) \longrightarrow bs(\neg(f \wedge g))$ **using** 1 3 **by** *fastforce*
have 5: $\vdash bs(\neg f) = (\neg(ds f))$ **using** *NotDsEqvBsNot* **by** *fastforce*
have 6: $\vdash bs(\neg g) = (\neg(ds g))$ **using** *NotDsEqvBsNot* **by** *fastforce*
have 7: $\vdash bs(\neg(f \wedge g)) = (\neg(ds(f \wedge g)))$ **using** *NotDsEqvBsNot* **by** *fastforce*
have 8: $\vdash \neg(ds f) \vee \neg(ds g) \longrightarrow \neg(ds(f \wedge g))$ **using** 4 5 6 7 **by** *fastforce*
hence 9: $\vdash \neg(ds f \wedge ds g) \longrightarrow \neg(ds(f \wedge g))$ **by** *auto*
from 9 **show** *?thesis* **by** *auto*
qed

lemma *DsAndImpElimL*:
 $\vdash ds(f \wedge g) \longrightarrow ds f$
using *DsAndImp* **by** *fastforce*

lemma *DsAndImpElimR*:
 $\vdash ds(f \wedge g) \longrightarrow ds g$
using *DsAndImp* **by** *fastforce*

lemma *BiImpBs*:
 $\vdash bi f \wedge finite \longrightarrow bs f$

proof –

have 1: $\vdash empty \longrightarrow empty \vee (bi f) \frown skip$ **by** *auto*
hence 10: $\vdash empty \wedge bi f \longrightarrow empty \vee (bi f) \frown skip$ **by** *auto*
have 11: $\vdash more \wedge bi f \longrightarrow (f \wedge inf) \vee (more \wedge bi f \wedge finite)$
using *BiElim[of f]* *OrFiniteInf[of LIFT bi f]* **by** *fastforce*
have 12: $\vdash more \wedge bi f \wedge finite \longrightarrow (bi f) \frown skip$
using *MoreAndBiImpBiSChopSkip* **by** *fastforce*
have 2: $\vdash more \wedge bi f \longrightarrow inf \vee (bi f) \frown skip$
using 12 **unfolding** *finite-d-def* **by** *fastforce*
have 3: $\vdash more \wedge bi f \wedge finite \longrightarrow empty \vee (bi f) \frown skip$
by (*metis* 12 *Prop05*)
have 4: $\vdash (bi f \wedge finite) = ((bi f \wedge empty \wedge finite) \vee (bi f \wedge more \wedge finite))$
by (*auto simp add: empty-d-def*)
have 41: $\vdash (bi f \wedge empty \wedge finite) = (bi f \wedge empty)$
using *FiniteAndEmptyEqvEmpty* **by** *fastforce*
have 5: $\vdash (empty \vee (bi f) \frown skip) = bs f$ **by** (*simp add: bs-d-def*)
have 6: $\vdash (bi f \wedge finite) = (bf f \wedge finite)$
by (*meson FiniteImpAnd FiniteImpBfEqvBi Prop11*)
from 2 3 4 5 **show** *?thesis* **by** *fastforce*
qed

lemma *BfImpBs*:
 $\vdash bf f \wedge finite \longrightarrow bs f$

proof –

have 1: $\vdash empty \longrightarrow empty \vee (bi f) \frown skip$ **by** *auto*
hence 2: $\vdash empty \wedge bi f \longrightarrow empty \vee (bi f) \frown skip$ **by** *auto*
have 20: $\vdash more \wedge bi f \longrightarrow inf \vee (bi f) \frown skip$
using *MoreAndBiImpBiSChopSkip* **unfolding** *finite-d-def* **by** *fastforce*

have 21: $\vdash \text{more} \wedge \text{bi } f \wedge \text{finite} \longrightarrow (\text{bi } f) \frown \text{skip}$
unfolding *finite-d-def* **using** 20 **by** *fastforce*
have 3: $\vdash \text{more} \wedge \text{bi } f \wedge \text{finite} \longrightarrow \text{empty} \vee (\text{bi } f) \frown \text{skip}$ **using** 21 **by** *auto*
have 4: $\vdash (\text{bi } f \wedge \text{finite}) = ((\text{bi } f \wedge \text{empty} \wedge \text{finite}) \vee (\text{bi } f \wedge \text{more} \wedge \text{finite}))$
by (*auto simp add: empty-d-def*)
have 41: $\vdash (\text{bi } f \wedge \text{empty} \wedge \text{finite}) = (\text{bi } f \wedge \text{empty})$
using *FiniteAndEmptyEqvEmpty* **by** *fastforce*
have 5: $\vdash (\text{empty} \vee (\text{bi } f) \frown \text{skip}) = \text{bs } f$ **by** (*simp add: bs-d-def*)
from 2 3 4 5 **show** ?thesis
by (*meson BiImpBs FiniteImpAnd FiniteImpBfEqvBi Prop11 lift-imp-trans*)
qed

lemma *BiFiniteEqvFinite*:
 $\vdash \text{bi } \text{finite} = \text{finite}$
by (*auto simp add: Valid-def itl-defs*)

lemma *BsImpBsBs*:
 $\vdash \text{bs } f \longrightarrow \text{bs } (\text{bs } f)$
proof –
have 1: $\vdash \text{bi } f \wedge \text{finite} \longrightarrow \text{bs } f$
using *BiImpBs* **by** *auto*
hence 2: $\vdash \text{bi } (\text{bi } f \wedge \text{finite}) \longrightarrow \text{bi}(\text{bs } f)$ **by** (*rule BiImpBiRule*)
have 21: $\vdash \text{bi } (\text{bi } f \wedge \text{finite}) = (\text{bi } f \wedge \text{finite})$
by (*metis BiAndEqv BiEqvBiBi BiFiniteEqvFinite inteq-reflection*)
have 3: $\vdash (\text{bi } f) \wedge \text{finite} \longrightarrow \text{bi}(\text{bs } f)$ **using** 2 21 **by** *fastforce*
have 4: $\vdash (\text{bi } f) \frown \text{skip} \longrightarrow (\text{bi}(\text{bs } f)) \frown \text{skip}$
by (*metis (no-types, opaque-lifting) 3 LeftChopImpChop Prop12 SChopstar-ext SChopstar-finite lift-imp-trans schop-d-def*)
have 5: $\vdash \text{empty} \vee (\text{bi } f) \frown \text{skip} \longrightarrow \text{empty} \vee (\text{bi}(\text{bs } f)) \frown \text{skip}$
using 4 **by** *fastforce*
have 6: $\vdash (\text{bi}(\text{bs } f)) \frown \text{skip} = (\text{bf}(\text{bs } f)) \frown \text{skip}$
by (*meson BfEqvImpSChopEqvSChop FiniteBfGen FiniteImpBfEqvBi MP Prop11*)
from 5 6 **show** ?thesis
by (*metis BsEqvEmptyOrBiSChopSkip inteq-reflection*)
qed

lemma *DsImpDi*:
 $\vdash \text{ds } f \longrightarrow \text{inf} \vee \text{di } f$
proof –
have 1: $\vdash \text{bi } (\neg f) \wedge \text{finite} \longrightarrow \text{bs } (\neg f)$ **by** (*rule BiImpBs*)
hence 2: $\vdash \neg(\text{bs } (\neg f)) \longrightarrow \text{inf} \vee \neg(\text{bi } (\neg f))$ **unfolding** *finite-d-def* **by** *fastforce*
from 2 **show** ?thesis
using *NotBsEqvDsNot DiEqvNotBiNot* **by** *fastforce*
qed

lemma *DsImpDf*:
 $\vdash \text{ds } f \longrightarrow \text{inf} \vee \text{df } f$
proof –
have 1: $\vdash \text{bf } (\neg f) \wedge \text{finite} \longrightarrow \text{bs } (\neg f)$ **by** (*rule BfImpBs*)

hence 2: $\vdash \neg(bs (\neg f)) \longrightarrow \inf \vee \neg(bf (\neg f))$ **unfolding** *finite-d-def* **by** *fastforce*
from 2 show ?thesis by (*simp add: bf-d-def ds-d-def*)
qed

lemma *BsImpBsRule*:

assumes $\vdash f \longrightarrow g$

shows $\vdash bs f \longrightarrow bs g$

proof –

have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*

hence 2: $\vdash bi f \longrightarrow bi g$ **by** (*rule BiImpBiRule*)

hence 3: $\vdash (bi f) \frown skip \longrightarrow (bi g) \frown skip$ **by** (*rule LeftSChopImpSChop*)

hence 4: $\vdash empty \vee (bi f) \frown skip \longrightarrow empty \vee (bi g) \frown skip$ **by** *auto*

from 4 show ?thesis by (*simp add: bs-d-def*)

qed

lemma *DsChopImpDsB*:

$\vdash ds (f;g) \longrightarrow ds f$

proof –

have 1: $\vdash di(f;g) \longrightarrow di f$ **by** (*rule DiChopImpDiB*)

hence 2: $\vdash (di(f;g)) \frown skip \longrightarrow (di f) \frown skip$ **by** (*rule LeftSChopImpSChop*)

from 2 show ?thesis using *DsDi*

by (*metis ChopImpDi DiEqvDiDi DsAndImpElimR Prop10 inteq-reflection*)

qed

lemma *DsSChopImpDsB*:

$\vdash ds (f \frown g) \longrightarrow ds f$

by (*metis DsAndImpElimL DsChopImpDsB lift-imp-trans schop-d-def*)

lemma *NotBsImpBsNotChop*:

$\vdash bs (\neg f) \longrightarrow bs (\neg(f;g))$

proof –

have 1: $\vdash ds (f;g) \longrightarrow ds f$ **by** (*rule DsChopImpDsB*)

hence 2: $\vdash \neg(ds f) \longrightarrow \neg(ds (f;g))$ **by** *fastforce*

from 2 show ?thesis using *NotDsEqvBsNot* **by** *fastforce*

qed

lemma *BsOrBsEqvBsBiOrBi*:

$\vdash (bs f \vee bs g) = bs(bi f \vee bi g)$

proof –

have 1: $\vdash (bs f \vee bs g) = ((empty \vee (bi f) \frown skip) \vee (empty \vee (bi g) \frown skip))$

by (*simp add: bs-d-def*)

have 2: $\vdash ((empty \vee (bi f) \frown skip) \vee (empty \vee (bi g) \frown skip)) = (empty \vee (bi f) \frown skip \vee (bi g) \frown skip)$

by *auto*

have 3: $\vdash ((bi f) \frown skip \vee (bi g) \frown skip) = (bi f \vee bi g) \frown skip$

by (*meson OrSChopEqv Prop11*)

hence 4: $\vdash (empty \vee (bi f) \frown skip \vee (bi g) \frown skip) = (empty \vee (bi f \vee bi g) \frown skip)$

by *auto*

have 5: $\vdash (bi f \vee bi g) = bi (bi f \vee bi g)$

by (*meson BiBiOrEqvBi Prop11*)

hence 6: $\vdash (bi\ f \vee bi\ g) \frown skip = bi\ (bi\ f \vee bi\ g) \frown skip$
by (*simp add: LeftSChopEqvSChop*)
hence 7: $\vdash (empty \vee bi\ (bi\ f \vee bi\ g) \frown skip) = (empty \vee (bi\ f \vee bi\ g) \frown skip)$
by *auto*
have 8: $\vdash (empty \vee (bi\ f \vee bi\ g) \frown skip) = bs(bi\ f \vee bi\ g)$
by (*metis BiBiOrEqvBi BsEqvEmptyOrBiSChopSkip inteq-reflection*)
from 1 2 4 8 show ?thesis by (*metis inteq-reflection*)
qed

lemma BsOrBsEqvBsBfOrBf:
 $\vdash (bs\ f \vee bs\ g) = bs(bf\ f \vee bf\ g)$
proof –
have 1: $\vdash (bs\ f \vee bs\ g) = bs(bi\ f \vee bi\ g)$
by (*simp add: BsOrBsEqvBsBiOrBi*)
have 2: $\vdash (bi\ f \vee bi\ g) \frown skip = (bf\ f \vee bf\ g) \frown skip$
by (*metis FiniteImpAnd FiniteImpBfEqvBi OrSChopEqv inteq-reflection schop-d-def*)
from 1 2 show ?thesis unfolding *bs-d-def*
by (*metis BfBfOrEqvBf BiBiOrEqvBi FiniteImpAnd FiniteImpBfEqvBi inteq-reflection schop-d-def*)
qed

lemma DiOrDsEqvDi:
 $\vdash di\ f \vee (ds\ f \wedge finite) = di\ f$
proof –
have 1: $\vdash di\ f \longrightarrow di\ f \vee ds\ f$ **by** *auto*
have 2: $\vdash di\ f \longrightarrow di\ f$ **by** *auto*
have 3: $\vdash ds\ f \longrightarrow inf \vee di\ f$ **by** (*rule DsImpDi*)
have 4: $\vdash di\ f \vee ds\ f \longrightarrow inf \vee di\ f$ **using 2 3 by** *auto*
from 1 4 show ?thesis unfolding *finite-d-def* **by** *fastforce*
qed

lemma DiAndDsEqvDs:
 $\vdash (di\ f \wedge (ds\ f \wedge finite)) = (ds\ f \wedge finite)$
proof –
have 1: $\vdash di\ f \wedge ds\ f \longrightarrow ds\ f$ **by** *auto*
have 2: $\vdash ds\ f \longrightarrow ds\ f$ **by** *auto*
have 3: $\vdash ds\ f \longrightarrow inf \vee di\ f$ **by** (*rule DsImpDi*)
have 4: $\vdash (ds\ f \wedge finite) \longrightarrow di\ f \wedge (ds\ f \wedge finite)$
using 2 3 unfolding *finite-d-def* **by** *auto*
from 1 4 show ?thesis by *auto*
qed

lemma DiFiniteOrInf:
 $\vdash di\ f = ((f \wedge finite) \vee (di\ f) \frown skip \vee (di\ f \wedge inf))$
proof –
have 1: $\vdash (di\ f) = ((di\ f \wedge finite) \vee (di\ f \wedge inf))$
by (*simp add: OrFiniteInf*)
have 2: $\vdash (di\ f \wedge finite) = (di\ f) \frown finite$
by (*metis ChopTrueAndFiniteEqvAndFiniteChopFinite DiEqvDiDi di-d-def inteq-reflection schop-d-def*)

have 3: $\vdash (\text{more} \wedge \text{finite}) = (\# \text{True} \frown \text{skip})$
by (metis *DiamondSChopdef FmoreEqvSkipChopFinite SkipFiniteEqvFiniteSkip fmore-d-def*
inteq-reflection sometimes-d-def)
have 4: $\vdash \text{finite} = (\# \text{True} \frown \text{skip} \vee \text{empty})$
using 3 **unfolding** *empty-d-def*
by (metis *DiamondEmptyEqvFinite FmoreEqvSkipChopFinite WPowerstarChopEqvChopOrRule*
WPowerstar-skip-finite empty-d-def fmore-d-def int-eq sometimes-d-def)
have 5: $\vdash (\text{di } f) \frown \text{finite} = (\text{di } f) \frown (\# \text{True} \frown \text{skip} \vee \text{empty})$
by (simp add: 4 *RightSChopEqvSChop*)
have 6: $\vdash (\text{di } f) \frown (\# \text{True} \frown \text{skip}) = (\text{di } f) \frown \text{skip}$
unfolding *di-d-def*
by (metis 2 3 *ChopAssoc FmoreEqvSkipChopFinite SkipFiniteEqvFiniteSkip di-d-def fmore-d-def*
inteq-reflection schop-d-def)
have 7: $\vdash (\text{di } f) \frown \text{finite} = (f \frown \text{finite}) \frown \text{finite}$
by (metis (no-types, lifting) 4 *ChopTrueAndFiniteEqvAndFiniteChopFinite Prop10 Prop11*
SChopImpFinite di-d-def inteq-reflection schop-d-def)
have 8: $\vdash (f \frown \text{finite}) \frown \text{finite} = (f \frown \text{finite})$
by (metis 2 7 *ChopTrueAndFiniteEqvAndFiniteChopFinite Prop04 di-d-def schop-d-def*)
have 9: $\vdash (f \frown \text{finite}) = f \frown (\# \text{True} \frown \text{skip} \vee \text{empty})$
by (simp add: 4 *RightSChopEqvSChop*)
have 10: $\vdash f \frown (\# \text{True} \frown \text{skip}) = (\text{di } f) \frown \text{skip}$
by (metis (no-types, opaque-lifting) 3 *ChopAssoc ChopTrueAndFiniteEqvAndFiniteChopFinite*
FmoreEqvSkipChopFinite SkipFiniteEqvFiniteSkip di-d-def fmore-d-def inteq-reflection schop-d-def)
have 11: $\vdash f \frown \text{empty} = (f \wedge \text{finite})$
by (simp add: *ChopEmpty schop-d-def*)
have 12: $\vdash f \frown (\# \text{True} \frown \text{skip} \vee \text{empty}) = (f \frown (\# \text{True} \frown \text{skip}) \vee f \frown \text{empty})$
by (simp add: *SChopOrEqv*)
have 13: $\vdash (\text{di } f \wedge \text{finite}) = ((f \wedge \text{finite}) \vee (\text{di } f) \frown \text{skip})$
by (metis 10 11 2 3 7 8 *FmoreEqvSkipChopFinite SChopOrEqvRule WPowerstarEqv*
WPowerstar-skip-finite fmore-d-def int-eq)
have 14: $\vdash ((\text{di } f \wedge \text{finite}) \vee (\text{di } f \wedge \text{inf})) =$
 $((f \wedge \text{finite}) \vee (\text{di } f) \frown \text{skip} \vee (\text{di } f \wedge \text{inf}))$
using 13 **by** *fastforce*
show ?thesis **using** 1 14 **by** *fastforce*
qed

lemma *DiFiniteEqv*:

$\vdash (\text{di } f \wedge \text{finite}) = ((f \wedge \text{finite}) \vee (\text{di } f) \frown \text{skip})$

proof –

have 1: $\vdash (\text{di } f \wedge \text{finite}) =$
 $((f \wedge \text{finite}) \vee (\text{di } f) \frown \text{skip} \vee (\text{di } f \wedge \text{inf})) \wedge \text{finite}$
using *DiFiniteOrInf* **by** (metis *int-simps(1) inteq-reflection*)
have 2: $\vdash (((f \wedge \text{finite}) \vee (\text{di } f) \frown \text{skip} \vee (\text{di } f \wedge \text{inf})) \wedge \text{finite}) =$
 $((f \wedge \text{finite}) \vee ((\text{di } f) \frown \text{skip} \wedge \text{finite}))$
by (metis (no-types, opaque-lifting) *ChopAndFiniteDist ChopEmpty DiEqvOrDiChopSkipB*
DiFiniteOrInf OrSChopEqv SChopAssoc inteq-reflection itl-def(9))
have 3: $\vdash ((\text{di } f) \frown \text{skip} \wedge \text{finite}) = ((\text{di } f) \frown \text{skip})$
by (metis *Prop10 SChopImpFinite WPowerstar-ext WPowerstar-skip-finite inteq-reflection*)
show ?thesis
by (metis 1 2 3 *inteq-reflection*)

qed

lemma *OrDsEqvDi*:

$\vdash ((f \wedge \text{finite}) \vee (ds\ f \wedge \text{finite})) = (di\ f \wedge \text{finite})$

proof –

have 1: $\vdash (ds\ f \wedge \text{finite}) = ((\text{finite} \longrightarrow (di\ f) \frown skip) \wedge \text{finite})$

using *DsDi* **by** *fastforce*

have 11: $\vdash ((\text{finite} \longrightarrow (di\ f) \frown skip) \wedge \text{finite}) = (((di\ f) \frown skip) \wedge \text{finite})$

by *fastforce*

have 12: $\vdash (((di\ f) \frown skip) \wedge \text{finite}) = (di\ f) \frown skip$

by (*metis Prop10 SChopImpFinite WPowerstar-ext WPowerstar-skip-finite inteq-reflection*)

have 2: $\vdash ((f \wedge \text{finite}) \vee (ds\ f \wedge \text{finite})) =$

$((f \wedge \text{finite}) \vee (di\ f) \frown skip)$

by (*metis 1 11 12 DiFiniteEqv inteq-reflection*)

have 3: $\vdash ((f \wedge \text{finite}) \vee (di\ f) \frown skip) = (di\ f \wedge \text{finite})$

by (*meson DiFiniteEqv Prop11*)

from 2 3 **show** *?thesis*

by (*metis int-eq*)

qed

lemma *BiFiniteEqv*:

$\vdash (bi\ f \wedge \text{finite}) = ((f \wedge \text{finite}) \wedge (\text{empty} \vee (bi\ f) \frown skip))$

by (*metis (no-types, opaque-lifting) BiAndEqv BiEqvAndWprevBi BiFiniteEqvFinite WPrevAndFiniteEqv inteq-reflection*)

lemma *AndSChopSkipEqVFiniteAndSChopSkip*:

$\vdash (f \wedge g \frown skip) = ((f \wedge \text{finite}) \wedge g \frown skip)$

proof –

have 1: $\vdash g \frown skip \longrightarrow \text{finite}$

by (*metis LenOneEqvSkip SChopImpFinite inteq-reflection len-k-finite*)

from 1 **show** *?thesis*

by *fastforce*

qed

lemma *BiFiniteEqvB*:

$\vdash (bi\ f \wedge \text{finite}) = (f \wedge (\text{empty} \vee (bi\ f) \frown skip))$

proof –

have 1: $\vdash ((f \wedge \text{finite}) \wedge \text{empty}) = (f \wedge \text{empty})$

using *FiniteAndEmptyEqvEmpty* **by** *auto*

have 2: $\vdash ((f \wedge \text{finite}) \wedge (bi\ f) \frown skip) = (f \wedge (bi\ f) \frown skip)$

by (*meson AndSChopSkipEqVFiniteAndSChopSkip Prop11*)

have 4: $\vdash (((f \wedge \text{finite}) \wedge \text{empty}) \vee ((f \wedge \text{finite}) \wedge (bi\ f) \frown skip)) =$
 $((f \wedge \text{empty}) \vee (f \wedge (bi\ f) \frown skip))$

using 1 2 **by** *fastforce*

have 3: $\vdash (bi\ f \wedge \text{finite}) = ((f \wedge \text{finite}) \wedge (\text{empty} \vee (bi\ f) \frown skip))$

by (*simp add: BiFiniteEqv*)

show *?thesis* **using** 3 4 **by** *fastforce*
qed

lemma *AndBsEqvBi*:

$\vdash (f \wedge bs\ f) = (bi\ f \wedge finite)$

by (*metis BiFiniteEqvB BsEqvEmptyOrBiSChopSkip inteq-reflection*)

lemma *BsEqvBsBi*:

$\vdash bs\ f = bs\ (bi\ f)$

proof –

have 1: $\vdash bs\ f = (empty \vee (bi\ f) \frown skip)$ **by** (*simp add: bs-d-def*)

have 2: $\vdash bi\ f = bi\ (bi\ f)$ **by** (*rule BiEqvBiBi*)

hence 3: $\vdash (bi\ f) \frown skip = bi\ (bi\ f) \frown skip$ **using** *LeftSChopEqvSChop* **by** *blast*

hence 4: $\vdash (empty \vee (bi\ f) \frown skip) = (empty \vee bi\ (bi\ f) \frown skip)$ **by** *auto*

from 1 4 **show** *?thesis* **unfolding** *bs-d-def* **by** *auto*

qed

lemma *StateImpBs*:

$\vdash init\ w \wedge finite \longrightarrow bs\ (init\ w)$

proof –

have 1: $\vdash init\ w = bi\ (init\ w)$ **by** (*rule StateEqvBi*)

have 2: $\vdash bi\ (init\ w) \wedge finite \longrightarrow bs\ (init\ w)$ **by** (*rule BiImpBs*)

from 1 2 **show** *?thesis* **using** *StateImpBi* **by** *fastforce*

qed

lemma *DsAndDsEqvDsAndDiOrDsAndDi*:

$\vdash (ds\ f \wedge ds\ g) = (ds\ (f \wedge di\ g) \vee ds\ (g \wedge di\ f))$

proof –

have 1: $\vdash (di\ f \wedge di\ g) = (di\ (f \wedge di\ g) \vee di\ (g \wedge di\ f))$

by (*rule DiAndDiEqvDiAndDiOrDiAndDi*)

hence 2: $\vdash (di\ f \wedge di\ g) \frown skip = (di\ (f \wedge di\ g) \vee di\ (g \wedge di\ f)) \frown skip$

by (*rule LeftSChopEqvSChop*)

have 3: $\vdash (di\ f \wedge di\ g) \frown skip = ((di\ f) \frown skip \wedge (di\ g) \frown skip)$

using *SChopSkipAndSChopSkip* **by** *fastforce*

have 4: $\vdash ((di\ f) \frown skip \wedge (di\ g) \frown skip) = (di\ (f \wedge di\ g) \vee di\ (g \wedge di\ f)) \frown skip$

using 2 3 **by** *fastforce*

have 5: $\vdash (di\ (f \wedge di\ g) \vee di\ (g \wedge di\ f)) \frown skip = (di\ (f \wedge di\ g) \frown skip \vee di\ (g \wedge di\ f) \frown skip)$

using *OrSChopEqv* **by** *blast*

have 6: $\vdash ds\ f = (finite \longrightarrow (di\ f) \frown skip)$

using *DsDi* **by** *blast*

have 7: $\vdash ds\ g = (finite \longrightarrow (di\ g) \frown skip)$

using *DsDi* **by** *blast*

have 8: $\vdash ((finite \longrightarrow (di\ f) \frown skip) \wedge (finite \longrightarrow (di\ g) \frown skip)) = (ds\ f \wedge ds\ g)$

using 6 7 **by** *fastforce*

have 81: $\vdash ((finite \longrightarrow di\ f \frown skip) \wedge (finite \longrightarrow di\ g \frown skip)) =$

$(finite \longrightarrow di\ f \frown skip \wedge di\ g \frown skip)$

by *fastforce*

have 9: $\vdash ds\ (f \wedge di\ g) = (finite \longrightarrow di\ (f \wedge di\ g) \frown skip)$

using *DsDi* **by** *blast*

have 10: $\vdash ds\ (g \wedge di\ f) = (finite \longrightarrow di\ (g \wedge di\ f) \frown skip)$

using *DsDi* by *blast*
 have 11: $\vdash ((\text{finite} \longrightarrow \text{di}(f \wedge \text{di } g) \frown \text{skip}) \vee (\text{finite} \longrightarrow \text{di}(g \wedge \text{di } f) \frown \text{skip})) =$
 $(\text{ds}(f \wedge \text{di } g) \vee \text{ds}(g \wedge \text{di } f))$
 using 9 10 by *fastforce*
 have 12: $\vdash ((\text{finite} \longrightarrow \text{di}(f \wedge \text{di } g) \frown \text{skip}) \vee (\text{finite} \longrightarrow \text{di}(g \wedge \text{di } f) \frown \text{skip})) =$
 $(\text{finite} \longrightarrow \text{di}(f \wedge \text{di } g) \frown \text{skip} \vee \text{di}(g \wedge \text{di } f) \frown \text{skip})$
 by *fastforce*
 from 4 5 8 11 12 81 show ?thesis by (metis *inteq-reflection*)
 qed

lemma *SChopFiniteEqvSChopTrueAndFinite*:

$\vdash g \frown \text{finite} = (g \frown \# \text{True} \wedge \text{finite})$
 by (meson *FiniteImportSChopRight Prop04 RightSChopEqvSChop int-simps(17) lift-and-com*)

lemma *BsEqvBiMoreImpChop*:

$\vdash (\text{bs } f) = (\text{bi}(\text{more} \wedge \text{finite} \longrightarrow f \frown \text{skip}) \wedge \text{finite})$
 proof –
 have 1: $\vdash (\text{bs } f \vee \text{inf}) = (\text{empty} \vee \text{inf} \vee (\text{bi } f \frown \text{skip}))$
 unfolding *bs-d-def* by *fastforce*
 have 2: $\vdash (\text{empty} \vee \text{inf} \vee (\text{bi } f \frown \text{skip})) = ((\neg(\neg(\text{bi } f)) \frown \text{skip}))$
 using *NotNotSChopSkip* by *fastforce*
 have 3: $\vdash \neg((\neg(\text{bi } f)) \frown \text{skip}) = (\neg(\text{di } (\neg f) \frown \text{skip}))$
 by (*simp add: bi-d-def*)
 have 4: $\vdash (\neg(\text{di } (\neg f) \frown \text{skip})) = (\neg(((\neg f) \frown \# \text{True}) \frown \text{skip}))$
 by (metis *FiniteImpAnd FiniteImpDfEqvDi LeftChopImpChop Prop11 itl-def(15) lift-imp-neg schop-d-def*)
 have 5: $\vdash (\neg(((\neg f) \frown \# \text{True}) \frown \text{skip})) = (\neg((\neg f) \frown (\# \text{True} \frown \text{skip})))$
 by (meson *Prop11 SChopAssoc lift-imp-neg*)
 have 05: $\vdash \# \text{True} \frown \text{skip} = \text{skip} \frown \text{finite}$
 by (metis *DiamondSChopdef Prop10 SkipFiniteEqvFiniteSkip WPowerstar-ext WPowerstar-skip-finite*
inteq-reflection itl-def(10) itl-def(9))
 have 6: $\vdash (\neg((\neg f) \frown (\# \text{True} \frown \text{skip}))) = (\neg((\neg f) \frown (\text{skip} \frown \text{finite})))$
 by (metis 05 5 *inteq-reflection*)
 have 7: $\vdash (\neg((\neg f) \frown (\text{skip} \frown \text{finite}))) = (\neg(((\neg f) \frown \text{skip}) \frown \text{finite}))$
 using *SChopAssoc* by *fastforce*
 have 07: $\vdash (\neg(((\neg f) \frown \text{skip}) \frown \text{finite})) = ((\neg(((\neg f) \frown \text{skip}) \frown \# \text{True})) \vee \text{inf})$
 using *InfEqvNotFinite SChopFiniteEqvSChopTrueAndFinite* by *fastforce*
 have 8: $\vdash (\neg(((\neg f) \frown \text{skip}) \frown \text{finite})) = ((\neg(\text{di } ((\neg f) \frown \text{skip}))) \vee \text{inf})$
 by (metis 07 *FiniteImportSChopRight Prop10 WPowerstar-ext WPowerstar-skip-finite di-d-def*
inteq-reflection lift-and-com schop-d-def)
 have 9: $\vdash (\neg(\text{di } ((\neg f) \frown \text{skip}))) = \text{bi } (\neg((\neg f) \frown \text{skip}))$
 using *NotDiEqvBiNot* by *blast*
 have 10: $\vdash \text{bi } (\neg((\neg f) \frown \text{skip})) = \text{bi } (\text{empty} \vee \text{inf} \vee (f \frown \text{skip}))$
 using *NotNotSChopSkip* using *BiEqvBi* by *blast*
 have 010: $\vdash (\text{empty} \vee \text{inf} \vee (f \frown \text{skip})) = (\neg(\text{more} \wedge \text{finite}) \vee (f \frown \text{skip}))$
 unfolding *empty-d-def finite-d-def* by *fastforce*
 have 11: $\vdash \text{bi } (\text{empty} \vee \text{inf} \vee (f \frown \text{skip})) = \text{bi } (\neg(\text{more} \wedge \text{finite}) \vee (f \frown \text{skip}))$
 by (*simp add: 010 BiEqvBi*)
 have 12: $\vdash (\neg(\text{more} \wedge \text{finite}) \vee (f \frown \text{skip})) = (\text{more} \wedge \text{finite} \longrightarrow f \frown \text{skip})$ by *auto*
 have 13: $\vdash \text{bi } (\neg(\text{more} \wedge \text{finite}) \vee (f \frown \text{skip})) = \text{bi}(\text{more} \wedge \text{finite} \longrightarrow f \frown \text{skip})$
 using 12 using *BiEqvBi* by *blast*

have 14: $\vdash (bs\ f \vee inf) = (\neg (((\neg f) \frown skip) \frown finite))$ **using** 1 2 3 4 5 6 7
by (metis NotSChopNotSkip bi-d-def inteq-reflection)
have 15: $\vdash (\neg (((\neg f) \frown skip) \frown finite)) = (bi(more \wedge finite \longrightarrow f \frown skip) \vee inf)$
using 8 9 10 11 13 **by** (metis inteq-reflection)
have 16: $\vdash (bs\ f \vee inf) = (bi(more \wedge finite \longrightarrow f \frown skip) \vee inf)$
using 14 15 **by** fastforce
have 17: $\vdash bs\ f \longrightarrow finite$
by (metis AndChopB EmptyImpFinite FiniteChopSkipImpFinite Prop02 bs-d-def lift-imp-trans schop-d-def)
have 18: $\vdash ((bs\ f \vee inf) \wedge finite) = bs\ f$
using 17 **unfolding** finite-d-def **by** fastforce
have 19: $\vdash ((bi(more \wedge finite \longrightarrow f \frown skip) \vee inf) \wedge finite) =$
 $(bi(more \wedge finite \longrightarrow f \frown skip) \wedge finite)$
unfolding finite-d-def **by** fastforce
have 20: $\vdash ((bs\ f \vee inf) \wedge finite) = ((bi(more \wedge finite \longrightarrow f \frown skip) \vee inf) \wedge finite)$
using 16 **by** fastforce
show ?thesis **using** 16 18 19 20 **by** (metis inteq-reflection)
qed

lemma NotSChopInf:

$\vdash (\neg (f \frown inf)) = (finite \vee \neg(f \frown \#True))$

proof –

have 1: $\vdash (\neg (f \frown inf)) = (finite \vee (bf\ (\neg f)))$
by (metis NotNotSChopInf int-simps(4) inteq-reflection)
have 2: $\vdash bf\ (\neg f) = (\neg (f \frown \#True))$
unfolding bf-d-def df-d-def **by** auto
show ?thesis **using** 1 2 Prop06 **by** blast
qed

lemma BoxMoreStateEqvBsFinState:

$\vdash (\Box(more \longrightarrow \neg (init\ w)) \wedge finite) = bs(\neg(fin(init\ w)))$

proof –

have 1: $\vdash \Box(more \longrightarrow \neg (init\ w)) = (\neg(\Diamond(\neg(more \longrightarrow \neg (init\ w)))))$
by (simp add: always-d-def)
have 01: $\vdash (\neg(more \longrightarrow \neg (init\ w))) = (init\ w \wedge more)$ **by** auto
hence 2: $\vdash \neg(\Diamond(\neg(more \longrightarrow \neg (init\ w)))) = (\neg(\#True \frown (init\ w \wedge more)))$
by (metis DiamondSChopdef TrueW int-simps(3) int-simps(6) inteq-reflection)
have 3: $\vdash more = (\#True \frown skip \vee inf)$
by (metis FmoreEqvSkipChopFinite MoreAndInfEqvInf OrFiniteInf SkipFiniteEqvFiniteSkip
fmore-d-def int-simps(17) inteq-reflection schop-d-def)
have 4: $\vdash (init\ w \wedge more) = (init\ w \wedge ((\#True \frown skip \vee inf)))$
using 3 **by** auto
have 5: $\vdash (init\ w \wedge (\#True \frown skip \vee inf)) =$
 $((init\ w \wedge empty) \frown (\#True \frown skip) \vee (init\ w \wedge empty) \frown inf)$
by (meson Prop04 SChopOrEqv StateAndEmptySChop)
have 6: $\vdash (init\ w \wedge more) = ((init\ w \wedge empty) \frown (\#True \frown skip) \vee (init\ w \wedge empty) \frown inf)$
using 4 5 **by** fastforce
have 7: $\vdash (\#True \frown (init\ w \wedge more)) =$
 $(\#True \frown ((init\ w \wedge empty) \frown (\#True \frown skip) \vee (init\ w \wedge empty) \frown inf))$
using 6 **by** (simp add: RightSChopEqvSChop)
have 8: $\vdash (\#True \frown ((init\ w \wedge empty) \frown (\#True \frown skip))) =$

$((\#True \wedge (init\ w \wedge empty)) \wedge (\#True \wedge skip))$)
using *SChopAssoc* **by** *blast*
have 9: $\vdash (((\#True \wedge (init\ w \wedge empty)) \wedge (\#True \wedge skip)) =$
 $((\#True \wedge (init\ w \wedge empty)) \wedge \#True \wedge skip))$)
using *SChopAssoc* **by** *blast*
have 10: $\vdash (\#True \wedge (init\ w \wedge more)) =$
 $((\#True \wedge (init\ w \wedge empty)) \wedge \#True \wedge skip) \vee \#True \wedge ((init\ w \wedge empty) \wedge inf)$)
using 7 8 9 **by** (*metis* *SChopOrEqv* *inteq-reflection*)
hence 11: $\vdash (\neg(\#True \wedge (init\ w \wedge more))) =$
 $(\neg(((\#True \wedge (init\ w \wedge empty)) \wedge \#True \wedge skip) \vee \#True \wedge ((init\ w \wedge empty) \wedge inf)))$
by *auto*
have 011: $\vdash (\neg(((\#True \wedge (init\ w \wedge empty)) \wedge \#True \wedge skip) \vee \#True \wedge ((init\ w \wedge empty) \wedge inf))) =$
 $(\neg(((\#True \wedge (init\ w \wedge empty)) \wedge \#True \wedge skip)) \wedge \neg(\#True \wedge ((init\ w \wedge empty) \wedge inf)))$
by *fastforce*
have 12: $\vdash (\neg(((\#True \wedge (init\ w \wedge empty)) \wedge \#True \wedge skip))) =$
 $(empty \vee inf \vee (\neg(\#True \wedge (init\ w \wedge empty)) \wedge \#True \wedge skip))$
using *NotSChopNotSkip* **by** *fastforce*
have 012: $\vdash (\neg(\#True \wedge ((init\ w \wedge empty) \wedge inf))) = (finite \vee (\neg(\#True \wedge (init\ w \wedge empty))) \wedge \#True)$

using *NotSChopInf* *SChopAssoc* **by** *fastforce*
have 0130: $\vdash (finite; (init\ w \wedge empty)) = finite; (w \wedge empty)$
by (*simp* *add: InitAndEmptyEqvAndEmpty RightChopEqvChop*)
have 0131: $\vdash ((init\ w \wedge empty) \wedge finite) = (w \wedge empty)$
by (*metis* *ChopEmpty InitAndEmptyEqvAndEmpty StateAndEmptySChop* *inteq-reflection* *schop-d-def*)
have 0132: $\vdash \#True \wedge ((init\ w \wedge empty) \wedge finite) = finite; (w \wedge empty)$
by (*simp* *add: 0131 ChopEqvChop* *schop-d-def*)
have 013: $\vdash (finite; (init\ w \wedge empty)) = \#True \wedge ((init\ w \wedge empty) \wedge finite)$
using 0130 0132 **by** *fastforce*
have 13: $\vdash (\neg(\#True \wedge (init\ w \wedge empty)) \wedge \#True) =$
 $bi(\Box (\neg(init\ w \wedge empty)))$
using *BiBoxNotEqvNotFiniteChopChopTrue* [of *LIFT* $((init\ w \wedge empty))$]
by (*metis* (*no-types*, *lifting*) 013 *ChopAssoc* *SChopAssoc* *inteq-reflection* *schop-d-def*)
hence 14: $\vdash (\neg(\#True \wedge (init\ w \wedge empty)) \wedge \#True) \wedge skip =$
 $(bi(\Box (\neg(init\ w \wedge empty)))) \wedge skip$ **using** *LeftSChopEqvSChop* **by** *blast*
hence 15: $\vdash (empty \vee inf \vee (\neg(\#True \wedge (init\ w \wedge empty)) \wedge \#True) \wedge skip) =$
 $(empty \vee inf \vee (bi(\Box (\neg(init\ w \wedge empty)))) \wedge skip)$
by *auto*
have 015: $\vdash (finite \vee (\neg(\#True \wedge (init\ w \wedge empty)) \wedge \#True)) =$
 $(finite \vee bi(\Box (\neg(init\ w \wedge empty))))$
using 13 **by** *auto*
have 16: $\vdash (\neg(((\#True \wedge (init\ w \wedge empty)) \wedge \#True) \wedge skip) \vee \#True \wedge ((init\ w \wedge empty) \wedge inf)) =$
 $((empty \vee inf \vee (bi(\Box (\neg(init\ w \wedge empty)))) \wedge skip) \wedge (finite \vee bi(\Box (\neg(init\ w \wedge empty)))))$
by (*metis* 011 012 13 *NotSChopNotSkip* *inteq-reflection*)
have 171: $\vdash (\neg(init\ w \wedge empty)) = (\neg(init\ w) \vee \neg empty)$
by *auto*
hence 172: $\vdash \Box(\neg(init\ w \wedge empty)) = \Box(\neg(init\ w) \vee \neg empty)$
by (*simp* *add: BoxEqvBox*)
hence 173: $\vdash bi(\Box(\neg(init\ w \wedge empty))) = bi(\Box(\neg(init\ w) \vee \neg empty))$
by (*simp* *add: BiEqvBi*)
hence 174: $\vdash bi(\Box(\neg(init\ w \wedge empty))) \wedge skip = bi(\Box(\neg(init\ w) \vee \neg empty)) \wedge skip$

using *LeftSChopEqvSChop* by *blast*
hence 17: $\vdash (empty \vee inf \vee (bi(\Box (\neg(init\ w \wedge empty))) \frown skip)) =$
 $(empty \vee inf \vee (bi(\Box (\neg(init\ w) \vee \neg empty)) \frown skip))$
 by *auto*
have 175: $\vdash (finite \vee bi(\Box (\neg(init\ w \wedge empty)))) =$
 $(finite \vee bi(\Box (\neg(init\ w) \vee \neg empty)))$
 using 173 by *fastforce*
have 181: $\vdash (\neg(init\ w) \vee \neg empty) = (\neg empty \vee \neg(init\ w))$
 by *auto*
hence 18: $\vdash \Box (\neg(init\ w) \vee \neg empty) = \Box (\neg empty \vee \neg(init\ w))$
 by (*simp add: BoxEqvBox*)
have 191: $\vdash (\neg empty \vee \neg(init\ w)) = (empty \longrightarrow \neg(init\ w))$
 by *auto*
hence 19: $\vdash \Box (\neg empty \vee \neg(init\ w)) = \Box (empty \longrightarrow \neg(init\ w))$
 by (*simp add: BoxEqvBox*)
have 20: $\vdash \Box (empty \longrightarrow \neg(init\ w)) = fin\ (\neg(init\ w))$
 by (*simp add: fin-d-def*)
have 21: $\vdash (fin\ (\neg(init\ w)) \wedge finite) = ((\neg(fin\ (init\ w))) \wedge finite)$
 by (*metis FinNotStateEqvNotFinState Initprop(2) inteq-reflection*)
have 22: $\vdash (bi(\Box (\neg(init\ w) \vee \neg empty)) \wedge finite) = (bi\ (\neg(fin\ (init\ w)))) \wedge finite$
 by (*metis 181 191 20 21 BiAndEqv BiFiniteEqvFinite inteq-reflection*)
hence 23: $\vdash (bi(\Box (\neg(init\ w) \vee \neg empty))) \frown skip = (bi\ (\neg(fin\ (init\ w)))) \frown skip$
 by (*simp add: LeftChopEqvChop schop-d-def*)
hence 24: $\vdash (empty \vee inf \vee (bi(\Box (\neg(init\ w) \vee \neg empty))) \frown skip) =$
 $(empty \vee inf \vee (bi\ (\neg(fin\ (init\ w)))) \frown skip)$
 by *auto*
have 241: $\vdash (bi\ (\neg(fin\ (init\ w)))) \frown skip = bf\ (\neg\ fin\ (init\ w)) \frown skip$
 by (*metis (no-types, opaque-lifting) EmptyChop FiniteImpAnd FiniteImpBfEqvBi LeftChopEqvChop Prop04 schop-d-def*)
hence 25: $\vdash (empty \vee inf \vee (bi\ (\neg(fin\ (init\ w)))) \frown skip) = (inf \vee bs(\neg(fin\ (init\ w))))$
 unfolding *bs-d-def* by *auto*
have 26: $\vdash (\Box (more \longrightarrow \neg(init\ w)) \wedge finite) =$
 $((empty \vee inf \vee (bi(\Box (\neg(init\ w \wedge empty))) \frown skip)) \wedge (finite \vee bi(\Box (\neg(init\ w \wedge empty)))))$
 $\wedge finite)$
 by (*metis (no-types, opaque-lifting) 01 10 16 TrueSChopEqvDiamond always-d-def int-simps(4) inteq-reflection*)
have 27: $\vdash (((empty \vee inf \vee (bi(\Box (\neg(init\ w \wedge empty))) \frown skip)) \wedge (finite \vee bi(\Box (\neg(init\ w \wedge empty)))))$
 $\wedge finite) =$
 $((empty \vee inf \vee (bi(\Box (\neg(init\ w \wedge empty))) \frown skip)) \wedge finite)$
 by *fastforce*
have 28: $\vdash (bi(\Box (\neg(init\ w \wedge empty))) \frown skip) \longrightarrow finite$
 by (*metis LenOneEqvSkip SChopImpFinite inteq-reflection len-k-finite*)
have 29: $\vdash empty \longrightarrow finite$
 by (*simp add: EmptyImpFinite*)
have 29: $\vdash (((empty \vee inf \vee (bi(\Box (\neg(init\ w \wedge empty))) \frown skip)) \wedge finite) =$
 $((empty \vee (bi(\Box (\neg(init\ w \wedge empty))) \frown skip)) \wedge finite)$
 using 28 29 unfolding *finite-d-def* by *fastforce*
have 30: $\vdash (((empty \vee (bi(\Box (\neg(init\ w \wedge empty))) \frown skip)) \wedge finite) = bs(\neg(fin\ (init\ w))))$
 by (*metis 171 23 241 29 bs-d-def inteq-reflection*)


```

show ?thesis
by (metis 26 27 29 30 inteq-reflection)
qed

lemma BsFalseEqvEmpty:
   $\vdash bs \#False = empty$ 
proof -
  have 1:  $\vdash bs \#False = (empty \vee bi \#False \frown skip)$ 
    by (simp add: bs-d-def)
  have 2:  $\vdash \neg(bi \#False \frown skip)$ 
  by (metis BiFiniteEqvB MoreEqvSkipChopTrue NotChopSkipEqvMoreAndNotChopSkip SkipTrueEqvTrueSkip

    TrueW int-simps(19) int-simps(2) int-simps(21) int-simps(3) inteq-reflection schop-d-def)
  from 1 2 show ?thesis by fastforce
qed

```

First occurrence

```

lemma FstEqvRule:
  assumes  $\vdash f = g$ 
  shows  $\vdash \triangleright f = \triangleright g$ 
proof -
  have 1:  $\vdash f = g$  using assms by auto
  hence 2:  $\vdash (\neg f) = (\neg g)$  by auto
  hence 3:  $\vdash bs(\neg f) = bs(\neg g)$  by (simp add: BsEqvRule)
  hence 4:  $\vdash (f \wedge bs(\neg f)) = (g \wedge bs(\neg g))$  using 1 by fastforce
  from 4 show ?thesis by (simp add: first-d-def)
qed

```

```

lemma FstImpFinite:
   $\vdash \triangleright f \longrightarrow finite$ 
unfolding first-d-def using BsEqvBiMoreImpChop by fastforce

```

```

lemma FstWithAndImp:
   $\vdash \triangleright f \wedge g \longrightarrow \triangleright (f \wedge g)$ 
proof -
  have 1:  $\vdash (\triangleright f \wedge g) = ((f \wedge (bs(\neg f))) \wedge g)$ 
    by (simp add: first-d-def)
  have 2:  $\vdash ((f \wedge (bs(\neg f))) \wedge g) = (f \wedge \neg(ds f) \wedge g)$ 
    using NotDsEqvBsNot by fastforce
  have 3:  $\vdash \neg(ds f) \longrightarrow \neg(ds(f \wedge g))$ 
    using DsAndImpElimL by fastforce
  hence 4:  $\vdash f \wedge \neg(ds f) \wedge g \longrightarrow f \wedge g \wedge \neg(ds(f \wedge g))$ 
    by auto
  have 5:  $\vdash (f \wedge g \wedge \neg(ds(f \wedge g))) = ((f \wedge g) \wedge (bs(\neg(f \wedge g))))$ 
    using NotDsEqvBsNot by fastforce
  have 6:  $\vdash ((f \wedge g) \wedge (bs(\neg(f \wedge g)))) = \triangleright(f \wedge g)$ 
    by (simp add: first-d-def)
  from 1 2 4 5 6 show ?thesis by fastforce
qed

```

lemma *FstWithOrEqv*:

$\vdash \triangleright(f \vee g) = ((\triangleright f \wedge bs(\neg g)) \vee (\triangleright g \wedge bs(\neg f)))$

proof –

have 1: $\vdash \triangleright(f \vee g) = ((f \vee g) \wedge bs(\neg(f \vee g)))$

by (*simp add: first-d-def*)

have 2: $\vdash (\neg(f \vee g)) = (\neg f \wedge \neg g)$

by *auto*

hence 3: $\vdash bs(\neg(f \vee g)) = bs(\neg f \wedge \neg g)$

using *BsEqvRule* **by** *blast*

have 4: $\vdash bs(\neg f \wedge \neg g) = (bs(\neg f) \wedge bs(\neg g))$

using *BsAndEqv* **by** *fastforce*

have 5: $\vdash ((f \vee g) \wedge bs(\neg(f \vee g))) = ((f \vee g) \wedge bs(\neg f) \wedge bs(\neg g))$

using 3 4 **by** *fastforce*

have 6: $\vdash ((f \vee g) \wedge bs(\neg f) \wedge bs(\neg g)) =$
 $((f \wedge bs(\neg f)) \wedge bs(\neg g)) \vee (g \wedge bs(\neg f) \wedge bs(\neg g))$

by *auto*

have 7: $\vdash ((f \wedge bs(\neg f)) \wedge bs(\neg g)) = (\triangleright f \wedge bs(\neg g))$

by (*simp add: first-d-def*)

have 8: $\vdash (g \wedge bs(\neg f) \wedge bs(\neg g)) = ((g \wedge bs(\neg g)) \wedge bs(\neg f))$

by *auto*

have 9: $\vdash ((g \wedge bs(\neg g)) \wedge bs(\neg f)) = (\triangleright g \wedge bs(\neg f))$

by (*simp add: first-d-def*)

have 10: $\vdash ((f \wedge bs(\neg f)) \wedge bs(\neg g)) \vee (g \wedge bs(\neg f) \wedge bs(\neg g)) =$
 $(\triangleright f \wedge bs(\neg g)) \vee (\triangleright g \wedge bs(\neg f))$

using 7 8 9 **by** *fastforce*

from 1 5 6 10 **show** *?thesis* **by** (*metis* 7 8 9 *int-eq*)

qed

lemma *FstFstAndEqvFstAnd*:

$\vdash \triangleright(\triangleright f \wedge g) = (\triangleright f \wedge g)$

proof –

have 1: $\vdash (\triangleright f \wedge g) = ((f \wedge (bs(\neg f))) \wedge g)$ **by** (*simp add: first-d-def*)

hence 2: $\vdash \triangleright f \wedge g \longrightarrow (bs(\neg f))$ **by** *auto*

hence 3: $\vdash \triangleright f \wedge g \longrightarrow \triangleright f \wedge g \wedge (bs(\neg f))$ **by** *auto*

have 4: $\vdash \neg f \longrightarrow \neg f \vee \neg(bs(\neg f)) \vee \neg g$ **by** *auto*

hence 5: $\vdash bs(\neg f) \longrightarrow bs(\neg f \vee \neg(bs(\neg f)) \vee \neg g)$ **using** *BsImpBsRule* **by** *blast*

have 6: $\vdash (\neg f \vee \neg(bs(\neg f)) \vee \neg g) = (\neg(f \wedge bs(\neg f) \wedge g))$ **by** *auto*

hence 7: $\vdash bs(\neg f \vee \neg(bs(\neg f)) \vee \neg g) = bs(\neg(f \wedge bs(\neg f) \wedge g))$ **using** *BsEqvRule* **by** *blast*

have 8: $\vdash ((f \wedge bs(\neg f)) \wedge g) = (\triangleright f \wedge g)$ **by** (*simp add: first-d-def*)

hence 9: $\vdash (\neg(f \wedge bs(\neg f) \wedge g)) = (\neg(\triangleright f \wedge g))$ **by** *auto*

hence 10: $\vdash bs(\neg(f \wedge bs(\neg f) \wedge g)) = bs(\neg(\triangleright f \wedge g))$ **using** *BsEqvRule* **by** *blast*

have 11: $\vdash \triangleright f \wedge g \longrightarrow (\triangleright f \wedge g) \wedge bs(\neg(\triangleright f \wedge g))$ **using** 3 5 7 10 **by** *fastforce*

hence 12: $\vdash \triangleright f \wedge g \longrightarrow \triangleright(\triangleright f \wedge g)$ **by** (*simp add: first-d-def*)

have 13: $\vdash \triangleright(\triangleright f \wedge g) = ((\triangleright f \wedge g) \wedge bs(\neg(\triangleright f \wedge g)))$ **by** (*simp add: first-d-def*)

hence 14: $\vdash \triangleright(\triangleright f \wedge g) \longrightarrow \triangleright f \wedge g$ **by** *auto*

from 12 14 **show** *?thesis* **by** *fastforce*

qed

lemma *FstTrue*:

$\vdash \triangleright \#True = \text{empty}$
proof –
have 1: $\vdash \triangleright \#True = (\#True \wedge bs (\neg \#True))$
by (*simp add: first-d-def*)
have 2: $\vdash bs (\neg \#True) = (\text{empty} \vee (bi (\neg \#True)) \frown skip)$
by (*simp add: bs-d-def*)
have 3: $\vdash \neg(bi (\neg \#True))$
using *BiElim* **by** *fastforce*
have 4: $\vdash \neg((bi (\neg \#True)) \frown skip)$
by (*metis (no-types, opaque-lifting) BiEqvBiBi DiSkipEqvMore NotChopSkipEqvMoreAndNotChopSkip SkipTrueEqvTrueSkip TrueEqvTrueSChopTrue di-d-def int-simps(17) int-simps(19) int-simps(21) int-simps(7) int-simps(9) inteq-reflection itl-def(18) schop-d-def*)
have 5: $\vdash bs (\neg \#True) = \text{empty}$
using 2 4 **by** *fastforce*
from 1 5 **show** *?thesis* **by** *fastforce*
qed

lemma *FstFalse*:
 $\vdash \neg(\triangleright \#False)$
proof –
have 1: $\vdash \triangleright \#False = (\#False \wedge bs \#True)$ **by** (*simp add: first-d-def*)
from 1 **show** *?thesis* **by** *auto*
qed

lemma *FstChopFalseEqvFalse*:
 $\vdash \neg(\triangleright f \frown \#False)$
by (*simp add: Valid-def itl-defs first-d-def bs-d-def*)

lemma *FstEmpty*:
 $\vdash \triangleright \text{empty} = \text{empty}$
proof –
have 1: $\vdash \triangleright \text{empty} = (\text{empty} \wedge bs (\neg \text{empty}))$ **by** (*simp add: first-d-def*)
have 2: $\vdash bs (\neg \text{empty}) = (\text{empty} \vee bi (\neg \text{empty})) \frown skip$ **by** (*simp add: bs-d-def*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndEmptyEqvAndEmpty*:
 $\vdash (\triangleright f \wedge \text{empty}) = (f \wedge \text{empty})$
proof –
have 1: $\vdash (\triangleright f \wedge \text{empty}) = ((f \wedge bs (\neg f)) \wedge \text{empty})$ **by** (*simp add: first-d-def*)
have 2: $\vdash bs (\neg f) = (\text{empty} \vee bi (\neg f)) \frown skip$ **by** (*simp add: bs-d-def*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *FstEmptyOrEqvEmpty*:
 $\vdash \triangleright(\text{empty} \vee f) = \text{empty}$
proof –
have 1: $\vdash \triangleright(\text{empty} \vee f) = ((\triangleright \text{empty} \wedge bs (\neg f)) \vee (\triangleright f \wedge bs (\neg \text{empty})))$ **using** *FstWithOrEqv* **by** *blast*
have 2: $\vdash (\neg \text{empty}) = \text{more}$ **by** (*simp add: empty-d-def*)
hence 3: $\vdash bs (\neg \text{empty}) = bs \text{ more}$ **using** *BsEqvRule* **by** *blast*

have 4: $\vdash bs \text{ more} = \text{empty}$ **using** *BsMoreEqvEmpty* **by** *blast*
have 5: $\vdash (\triangleright f \wedge bs (\neg \text{empty})) = (\triangleright f \wedge \text{empty})$ **using** 3 4 **by** *fastforce*
have 6: $\vdash \triangleright \text{empty} = \text{empty}$ **using** *FstEmpty* **by** *blast*
hence 7: $\vdash (\triangleright \text{empty} \wedge bs (\neg f)) = (\text{empty} \wedge bs (\neg f))$ **by** *auto*
have 8: $\vdash (\text{empty} \wedge bs (\neg f)) = (\text{empty} \wedge (\text{empty} \vee bi (\neg f) \frown skip))$ **by** (*simp add:bs-d-def*)
have 9: $\vdash (\text{empty} \wedge (\text{empty} \vee bi (\neg f) \frown skip)) = \text{empty}$ **by** *auto*
have 10: $\vdash (\text{empty} \wedge bs (\neg f)) = \text{empty}$ **using** 8 9 **by** *auto*
have 11: $\vdash ((\triangleright \text{empty} \wedge bs (\neg f)) \vee (\triangleright f \wedge bs (\neg \text{empty}))) =$
 $(\text{empty} \vee (\triangleright f \wedge \text{empty}))$ **using** 7 10 5 **by** *fastforce*
have 12: $\vdash (\text{empty} \vee (\triangleright f \wedge \text{empty})) = \text{empty}$ **by** *auto*
from 1 11 12 **show** *?thesis* **by** *fastforce*
qed

lemma *FstChopEmptyEqvFstChopFstEmpty*:

$\vdash (\triangleright f \frown g \wedge \text{empty}) = (\triangleright f \frown \triangleright g \wedge \text{empty})$

proof –

have 1: $\vdash (\triangleright f \frown g \wedge \text{empty}) = (\triangleright f \wedge g \wedge \text{empty})$ **using** *SChopEmptyAndEmpty* **by** *blast*
have 2: $\vdash (\triangleright g \wedge \text{empty}) = (g \wedge \text{empty})$ **using** *FstAndEmptyEqvAndEmpty* **by** *blast*
hence 3: $\vdash (\triangleright f \wedge g \wedge \text{empty}) = (\triangleright f \wedge \triangleright g \wedge \text{empty})$ **by** *auto*
have 4: $\vdash (\triangleright f \frown \triangleright g \wedge \text{empty}) = (\triangleright f \wedge \triangleright g \wedge \text{empty})$ **using** *SChopEmptyAndEmpty* **by** *blast*
from 1 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *FstMoreEqvSkip*:

$\vdash \triangleright \text{more} = \text{skip}$

proof –

have 1: $\vdash \triangleright \text{more} = (\text{more} \wedge bs (\neg \text{more}))$ **by** (*simp add: first-d-def*)
have 2: $\vdash (\text{more} \wedge bs (\neg \text{more})) = (\text{more} \wedge (\text{empty} \vee bi (\neg \text{more}) \frown skip))$ **by** (*simp add:bs-d-def*)
have 3: $\vdash (\text{more} \wedge (\text{empty} \vee bi (\neg \text{more}) \frown skip)) = (\text{more} \wedge bi (\neg \text{more}) \frown skip)$
unfolding *empty-d-def* **by** *fastforce*
have 4: $\vdash (\text{more} \wedge ((bi (\neg \text{more})) \frown skip)) = ((bi (\neg \text{more})) \frown skip)$
using *SChopSkipImpMore* **by** *fastforce*
have 5: $\vdash ((bi (\neg \text{more})) \frown skip) = bi \text{ empty} \frown skip$ **by** (*simp add: empty-d-def*)
have 6: $\vdash bi \text{ empty} = \text{empty}$ **using** *BiEmptyEqvEmpty* **by** *auto*
hence 7: $\vdash bi \text{ empty} \frown skip = \text{empty} \frown skip$
using *LeftSChopEqvSChop* **by** *blast*
have 8: $\vdash \text{empty} \frown skip = \text{skip}$ **using** *EmptySChop* **by** *blast*
from 1 2 3 4 5 7 8 **show** *?thesis* **by** (*metis int-eq*)

qed

lemma *FstEqvBsNotAndDi*:

$\vdash \triangleright f = (bs (\neg f) \wedge di f)$

proof –

have 1: $\vdash bs (\neg f) = (\neg(ds f))$ **by** (*simp add: ds-d-def*)
hence 2: $\vdash (bs (\neg f) \wedge di f) = (\neg(ds f) \wedge di f)$ **by** *auto*
have 3: $\vdash (di f \wedge \text{finite}) = ((ds f \wedge \text{finite}) \vee (f \wedge \text{finite}))$
using *OrDsEqvDi* **by** *fastforce*
have 03: $\vdash (\neg(ds f)) \longrightarrow \text{finite}$
by (*metis BsEqvBiMoreImpChop Prop11 Prop12 ds-d-def int-simps(4) lift-imp-trans*)
have 4: $\vdash (\neg(ds f) \wedge di f) = (\neg(ds f) \wedge ((ds f \wedge \text{finite}) \vee (f \wedge \text{finite})))$

```

    using 03 3 by fastforce
  have 5 :  $\vdash (\neg(ds\ f) \wedge ((ds\ f \wedge finite) \vee (f \wedge finite))) =$ 
     $(\neg(ds\ f) \wedge f \wedge finite)$  by auto
  have 6 :  $\vdash (\neg(ds\ f) \wedge f \wedge finite) = (f \wedge bs\ (\neg\ f))$  using 1
  using 03 by fastforce
  from 2 4 5 6 show ?thesis by (metis first-d-def int-eq)
qed

```

lemma *FstOrDiEqvDi*:

```

 $\vdash (\triangleright f \vee di\ f) = di\ f$ 
proof -
  have 1:  $\vdash (\triangleright f \vee di\ f) = ((f \wedge bs\ (\neg\ f)) \vee di\ f)$  by (simp add: first-d-def)
  have 2:  $\vdash ((f \wedge bs\ (\neg\ f)) \vee di\ f) = ((f \vee di\ f) \wedge (bs\ (\neg\ f) \vee di\ f))$  by auto
  have 3:  $\vdash (f \vee di\ f) = di\ f$  by (simp add: OrDiEqvDi)
  hence 4:  $\vdash ((f \vee di\ f) \wedge (bs\ (\neg\ f) \vee di\ f)) = (di\ f \wedge (bs\ (\neg\ f) \vee di\ f))$  by auto
  have 5:  $\vdash (di\ f \wedge (bs\ (\neg\ f) \vee di\ f)) = di\ f$  by auto
  from 1 2 4 5 show ?thesis by fastforce
qed

```

lemma *FstAndDiEqvFst*:

```

 $\vdash (\triangleright f \wedge di\ f) = \triangleright f$ 
proof -
  have 1:  $\vdash (\triangleright f \wedge di\ f) = ((f \wedge bs\ (\neg\ f)) \wedge di\ f)$  by (simp add: first-d-def)
  have 2:  $\vdash (f \wedge di\ f) = f$  by (meson DiIntro Prop10 Prop11)
  hence 3:  $\vdash (f \wedge bs\ (\neg\ f) \wedge di\ f) = (f \wedge bs\ (\neg\ f))$  by auto
  from 1 3 show ?thesis by (metis first-d-def int-iffD2 int-iffI Prop12)
qed

```

lemma *WPrevFiniteEqv*:

```

 $\vdash wprev(f \wedge finite) = (\neg prev(\neg f \vee inf))$ 
proof -
  have 1:  $\vdash (\neg (f \wedge \neg inf)) = (\neg f \vee inf)$ 
    by fastforce
  show ?thesis unfolding wprev-d-def finite-d-def using 1
    by (metis int-simps(10) inteq-reflection)
qed

```

lemma *FPrevEqv*:

```

 $\vdash f \frown skip = (more \wedge wprev(f \wedge finite))$ 
proof -
  have 1:  $\vdash wprev(f \wedge finite) = (empty \vee f \frown skip)$ 
    using WPrevAndFiniteEqv by blast
  have 2:  $\vdash (more \wedge wprev(f \wedge finite)) = f \frown skip$ 
    using 1 unfolding empty-d-def
    by (simp add: Prop01 Prop03 Prop11 Prop12 SChopSkipImpMore)
  show ?thesis
    using 2 by auto
qed

```

```

lemma DiEqvDiFst:
   $\vdash \text{finite} \longrightarrow \text{di } (f) = \text{di } (\triangleright f)$ 
proof –
  have 1:  $\vdash \text{di } (\triangleright f) = \text{di } (f \wedge \text{bs } (\neg f))$ 
    by (simp add: first-d-def)
  have 2:  $\vdash \text{di } (f \wedge \text{bs } (\neg f)) \longrightarrow \text{di } f \wedge \text{di } (\text{bs } (\neg f))$ 
    using DiAndImpAnd by auto
  hence 3:  $\vdash \text{di } (f \wedge \text{bs } (\neg f)) \longrightarrow \text{di } f$ 
    by auto
  have 03:  $\vdash \text{di } (f \wedge \text{bs } (\neg f)) \longrightarrow \text{di } (f)$ 
    by (metis AndDiEqv DiImpDi Prop11 Prop12 inteq-reflection)
  have 4:  $\vdash \text{di } (\triangleright f) \longrightarrow \text{di } (f)$  using 1 03
    by fastforce

  have 5:  $\vdash (\text{di } (f) \wedge \text{empty}) = (f \wedge \text{empty})$ 
    by (simp add: DiAndEmptyEqvAndEmpty)
  have 6:  $\vdash (\triangleright f \wedge \text{empty}) = (f \wedge \text{empty})$ 
    using FstAndEmptyEqvAndEmpty by auto
  have 7:  $\vdash \text{di } (f) \wedge \text{empty} \longrightarrow \triangleright f$ 
    using 5 6 by fastforce
  have 8:  $\vdash \triangleright f \longrightarrow \text{di } (\triangleright f)$ 
    using DiIntro by auto
  have 9:  $\vdash \text{di } (f) \wedge \text{empty} \longrightarrow \text{di } (\triangleright f)$ 
    using 7 8 using lift-imp-trans by blast
  hence 10:  $\vdash \text{empty} \longrightarrow (\text{di } (f) \longrightarrow \text{di } (\triangleright f))$ 
    by auto

  have 11:  $\vdash (\text{di } (f) \longrightarrow \text{di } (\triangleright f)) \frown \text{skip} \longrightarrow \text{more} \wedge \text{finite}$ 
    by (metis FstImpFinite FstMoreEqvSkip Prop12 SChopImpFinite SChopSkipImpMore inteq-reflection)
  have 12:  $\vdash \text{more} \wedge \text{finite} \longrightarrow$ 
     $(\text{di } (f) \longrightarrow \text{di } (\triangleright f)) \frown \text{skip} =$ 
     $(\text{di } (f)) \frown \text{skip} \longrightarrow (\text{di } (\triangleright f)) \frown \text{skip} )$ 
    by (simp add: MoreImpImpSChopSkipEqv)
  have 13:  $\vdash (\text{more} \wedge \text{finite} \wedge (\text{di } (f) \longrightarrow \text{di } (\triangleright f)) \frown \text{skip}) =$ 
     $(\text{more} \wedge \text{finite} \wedge ((\text{di } (f)) \frown \text{skip} \longrightarrow (\text{di } (\triangleright f)) \frown \text{skip}))$ 
    using 12 by fastforce
  have 14:  $\vdash (\text{di } (f) \longrightarrow \text{di } (\triangleright f)) \frown \text{skip} =$ 
     $(\text{more} \wedge \text{finite} \wedge ((\text{di } (f)) \frown \text{skip} \longrightarrow (\text{di } (\triangleright f)) \frown \text{skip}))$ 
    using 11 13 by fastforce
  have 15:  $\vdash (\text{di } (f) \wedge \text{finite}) = ((f \wedge \text{finite}) \vee (\text{ds } f \wedge \text{finite}))$ 
    using OrDsEqvDi by (metis (no-types, opaque-lifting) int-eq lift-and-com)
  have 16:  $\vdash \text{di } (f) = (\text{di } (f) \wedge (\text{bs } (\neg f) \vee \neg(\text{bs } (\neg f))))$ 
    by auto
  have 17:  $\vdash (\text{di } (f) \wedge (\text{bs } (\neg f) \vee \neg(\text{bs } (\neg f)))) =$ 
     $((\text{di } (f) \wedge \text{bs } (\neg f)) \vee (\text{di } (f) \wedge \neg(\text{bs } (\neg f))))$ 
    by auto
  have 171:  $\vdash \text{bs } (\neg f) \longrightarrow \text{finite}$ 
    using BsEqvBiMoreImpChop by fastforce
  have 18:  $\vdash (\text{di } (f) \wedge \text{bs } (\neg f)) = ((f \vee \text{ds } f) \wedge \text{bs } (\neg f))$ 

```

using 15 171 by fastforce
 have 19: $\vdash ((f \vee ds\ f) \wedge bs\ (\neg f)) = ((f \wedge bs\ (\neg f)) \vee (ds\ f \wedge bs\ (\neg f)))$
 by auto
 have 20: $\vdash \neg(ds\ f \wedge bs\ (\neg f))$
 by (simp add: ds-d-def)
 have 21: $\vdash ((f \wedge bs\ (\neg f)) \vee (ds\ f \wedge bs\ (\neg f))) = (f \wedge bs\ (\neg f))$
 using 20 by auto
 have 22: $\vdash (di\ (f) \wedge bs\ (\neg f)) = (f \wedge bs\ (\neg f))$
 using 18 19 21 by fastforce
 have 23: $\vdash (f \wedge bs\ (\neg f)) = \triangleright f$
 by (simp add: first-d-def)
 have 24: $\vdash (\triangleright f) \longrightarrow di\ (\triangleright f)$
 using DiIntro by auto
 have 25: $\vdash (f \wedge bs\ (\neg f)) \longrightarrow di\ (\triangleright f)$
 using 23 24 by fastforce
 have 26: $\vdash (di\ (f) \wedge bs\ (\neg f)) \longrightarrow di\ (\triangleright f)$
 using 25 22 by fastforce
 hence 27: $\vdash (di\ (f) \wedge bs\ (\neg f) \wedge ((di\ (f) \longrightarrow di\ (\triangleright f)) \frown skip)) \longrightarrow di\ (\triangleright f)$
 by auto

 have 28: $\vdash (di\ (f) \wedge \neg(bs\ (\neg f))) = (di\ (f) \wedge ds\ f)$
 by (simp add: ds-d-def)
 hence 29: $\vdash (di\ (f) \wedge \neg(bs\ (\neg f)) \wedge ((di\ (f) \longrightarrow di\ (\triangleright f)) \frown skip)) =$
 $(di\ (f) \wedge ds\ f \wedge finite \wedge ((di\ (f) \longrightarrow di\ (\triangleright f)) \frown skip))$
 using 11 by fastforce
 have 030: $\vdash finite \longrightarrow (di\ f) \frown skip = (di\ f);skip$
 by (auto simp add: Valid-def itl-defs)
 have 031: $\vdash finite \longrightarrow (di\ f);skip = (di\ (f \wedge finite));skip$
 by (auto simp add: Valid-def itl-defs)
 have 032: $\vdash finite \longrightarrow (di\ f) \frown skip = (di\ (f \wedge finite));skip$
 by (auto simp add: Valid-def itl-defs)
 have 30: $\vdash ds\ f = (finite \longrightarrow prev(di\ f))$
 using DsDi 030 unfolding prev-d-def by fastforce
 have 033: $\vdash ds\ f = (finite \longrightarrow (di\ (f)) \frown skip)$
 by (simp add: DsDi)
 hence 31: $\vdash (di\ (f) \wedge ds\ f \wedge finite \wedge ((di\ (f) \longrightarrow di\ (\triangleright f)) \frown skip)) =$
 $(di\ (f) \wedge finite \wedge ((di\ (f)) \frown skip) \wedge ((di\ (f) \longrightarrow di\ (\triangleright f)) \frown skip))$
 by auto
 have 32: $\vdash ((di\ (f) \longrightarrow di\ (\triangleright f)) \frown skip) \longrightarrow ((di\ (f)) \frown skip \longrightarrow (di\ (\triangleright f)) \frown skip)$
 using 14 by auto
 hence 33: $\vdash di\ (f) \wedge finite \wedge ((di\ (f)) \frown skip) \wedge ((di\ (f) \longrightarrow di\ (\triangleright f)) \frown skip) \longrightarrow$
 $di\ (f) \wedge finite \wedge ((di\ (f)) \frown skip) \wedge ((di\ (f)) \frown skip \longrightarrow (di\ (\triangleright f)) \frown skip)$
 by auto
 have 34: $\vdash di\ (f) \wedge finite \wedge ((di\ (f)) \frown skip) \wedge$
 $((di\ (f)) \frown skip \longrightarrow (di\ (\triangleright f)) \frown skip) \longrightarrow (di\ (\triangleright f)) \frown skip$
 by fastforce
 have 36: $\vdash (di\ (\triangleright f)) \frown skip \longrightarrow di\ (di\ (\triangleright f))$
 by (metis AndChopA ChopImpDi lift-imp-trans schop-d-def)
 have 37: $\vdash di\ (di\ (\triangleright f)) = di\ (\triangleright f)$
 using DiEqvDiDi by fastforce

have 38: $\vdash di\ f \wedge finite \wedge (di\ f) \frown skip \wedge ((di\ f)) \frown skip \longrightarrow (di\ (\triangleright f)) \frown skip \longrightarrow di\ (\triangleright f)$
using 37 36
by (metis 34 inteq-reflection lift-imp-trans)
have 39: $\vdash di\ f \wedge finite \wedge \neg(bs\ (\neg f)) \wedge ((di\ f) \longrightarrow di\ (\triangleright f)) \frown skip \longrightarrow di\ (\triangleright f)$
using 29 31 33 38 36 37 **by** fastforce
hence 40: $\vdash \neg(bs\ (\neg f)) \wedge finite \wedge ((di\ f) \longrightarrow di\ (\triangleright f)) \frown skip \longrightarrow (di\ f) \longrightarrow di\ (\triangleright f)$
by fastforce
have 41: $\vdash bs\ (\neg f) \wedge ((di\ f) \longrightarrow di\ (\triangleright f)) \frown skip \longrightarrow (di\ f) \longrightarrow di\ (\triangleright f)$
using 27 **by** fastforce
have 42: $\vdash (\neg(bs\ (\neg f)) \vee bs\ (\neg f)) \wedge ((di\ f) \longrightarrow di\ (\triangleright f)) \frown skip \longrightarrow (di\ f) \longrightarrow di\ (\triangleright f)$
using 40 41 29 **by** fastforce
have 43: $\vdash (\neg(bs\ (\neg f)) \vee bs\ (\neg f))$
by auto
have 44: $\vdash ((di\ f) \longrightarrow di\ (\triangleright f)) \frown skip \longrightarrow (di\ f) \longrightarrow di\ (\triangleright f)$
using 42 43 **by** fastforce
have 45: $\vdash finite \longrightarrow di\ f \longrightarrow di\ (\triangleright f)$
using 10 44 EmptySChopSkipInductB[of LIFT $di\ f \longrightarrow di\ (\triangleright f)$]
unfolding prev-d-def schop-d-def **by** fast
from 4 45 **show** ?thesis **by** fastforce
qed

lemma FiniteImpDiEqvImpDfEqv:

assumes $\vdash finite \longrightarrow di\ f = di\ g$

shows $\vdash df\ f = df\ g$

proof –

have 1: $\vdash (di\ f \wedge finite) = (di\ g \wedge finite)$
using assms **by** fastforce
have 2: $\vdash (di\ f \wedge finite) = f \frown finite$
unfolding di-d-def
by (simp add: ChopTrueAndFiniteEqvAndFiniteChopFinite schop-d-def)
have 3: $\vdash (di\ g \wedge finite) = g \frown finite$
unfolding di-d-def
by (simp add: ChopTrueAndFiniteEqvAndFiniteChopFinite schop-d-def)
have 4: $\vdash ((f \frown finite) = (g \frown finite)) \longrightarrow (f \frown \# True = g \frown \# True)$
by (metis (no-types, lifting) 1 2 3 Prop11 SChopAssoc SChopFiniteEqvSChopTrueAndFinite TrueEqvTrueSChopTrue int-simps(1) inteq-reflection schop-d-def)
show ?thesis **unfolding** df-d-def **by** (metis 1 2 3 4 MP inteq-reflection)
qed

lemma DfEqvDfFst:

$\vdash df\ f = df\ (\triangleright f)$

by (simp add: DiEqvDiFst FiniteImpDiEqvImpDfEqv)

lemma FstDiEqvFst:

$\vdash \triangleright(di\ f) = \triangleright f$

proof –

have 1: $\vdash \triangleright(di\ f) = (di\ f \wedge bs\ (\neg (di\ f)))$ **by** (simp add: first-d-def)

have 2: $\vdash (\neg (di\ f)) = bi\ (\neg f)$ **by** (simp add: NotDiEqvBiNot)

hence 3: $\vdash bs\ (\neg (di\ f)) = bs\ (bi\ (\neg f))$ **using** BsEqvRule **by** blast

have 4: $\vdash bs (bi (\neg f)) = bs (\neg f)$ **using** *BsEqvBsBi* **by** *fastforce*
hence 5: $\vdash (di f \wedge bs (\neg (di f))) = (di f \wedge bs (\neg f))$ **using** 3 **by** *fastforce*
have 05: $\vdash bs (\neg (di f)) \longrightarrow finite$ **using** *BsEqvBiMoreImpChop* **by** *fastforce*
have 06: $\vdash bs (\neg f) \longrightarrow finite$ **using** 05 3 4 **by** *fastforce*
have 6 : $\vdash (di f \wedge finite) = ((f \vee ds f) \wedge finite)$ **using** *OrDsEqvDi* **by** (*metis FiniteOr inteq-reflection*)
have 07: $\vdash (di f \wedge bs (\neg f)) = (di f \wedge finite \wedge bs (\neg f))$ **using** 06 **by** *fastforce*
have 08: $\vdash (di f \wedge finite \wedge bs (\neg f)) = ((f \vee ds f) \wedge finite \wedge bs (\neg f))$ **using** 6 **by** *fastforce*
have 09: $\vdash ((f \vee ds f) \wedge finite \wedge bs (\neg f)) = ((f \vee ds f) \wedge bs (\neg f))$ **using** 06 **by** *fastforce*
have 7: $\vdash (di f \wedge bs (\neg f)) = ((f \vee ds f) \wedge bs (\neg f))$ **by** (*metis 07 08 09 inteq-reflection*)
have 8: $\vdash ((f \vee ds f) \wedge bs (\neg f)) = ((f \wedge bs (\neg f)) \vee (ds f \wedge bs (\neg f)))$ **by** *auto*
have 9: $\vdash \neg(ds f \wedge bs (\neg f))$ **by** (*simp add: ds-d-def*)
have 10: $\vdash (f \wedge bs (\neg f)) = \triangleright f$ **by** (*simp add: first-d-def*)
have 11: $\vdash ((f \wedge bs (\neg f)) \vee (ds f \wedge bs (\neg f))) = \triangleright f$ **using** 9 10 **by** *fastforce*
from 1 5 7 8 11 **show** *?thesis* **by** (*metis int-eq*)
qed

lemma *FstDfEqvFst*:

$\vdash \triangleright(df f) = \triangleright f$
by (*metis (no-types, opaque-lifting) DfEqvDfFst DfEqvDiAndFinite FstDiEqvFst FstFstAndEqvFstAnd FstImpFinite Prop10 int-eq*)

lemma *DiAndFstOrEqvFstOrDiAnd*:

$\vdash (di f \wedge (\triangleright f \vee g)) = (\triangleright f \vee (di f \wedge g))$

proof –

have 1: $\vdash (di f \wedge (\triangleright f \vee g)) = (\triangleright f \wedge di f) \vee (di f \wedge g)$ **by** *auto*
have 2: $\vdash (\triangleright f \wedge di f) = \triangleright f$ **using** *FstAndDiEqvFst* **by** *blast*
from 1 2 **show** *?thesis* **by** *auto*

qed

lemma *DiOrFstAndEqvDi*:

$\vdash di f \vee (\triangleright f \wedge g) = di f$

proof –

have 1: $\vdash (di f \vee (\triangleright f \wedge g)) = ((\triangleright f \vee di f) \wedge (di f \vee g))$ **by** *auto*
have 2: $\vdash (\triangleright f \vee di f) = di f$ **using** *FstOrDiEqvDi* **by** *blast*
from 1 2 **show** *?thesis* **by** *auto*

qed

lemma *FstDiAndDiEqv*:

$\vdash \triangleright(di f \wedge di g) = ((\triangleright f \wedge di g) \vee (\triangleright g \wedge di f))$

proof –

have 1: $\vdash \triangleright(di f \wedge di g) = ((di f \wedge di g) \wedge bs (\neg(di f \wedge di g)))$ **by** (*simp add: first-d-def*)
have 2: $\vdash (\neg(di f \wedge di g)) = (bi (\neg f) \vee bi (\neg g))$ **by** (*auto simp add: bi-d-def*)
hence 3: $\vdash bs (\neg(di f \wedge di g)) = bs(bi (\neg f) \vee bi (\neg g))$ **using** *BsEqvRule* **by** *blast*
hence 4: $\vdash ((di f \wedge di g) \wedge bs (\neg(di f \wedge di g))) =$
 $(di f \wedge di g \wedge bs(bi (\neg f) \vee bi (\neg g)))$ **by** *auto*
have 5: $\vdash (bs (\neg f) \vee bs (\neg g)) = bs(bi (\neg f) \vee bi (\neg g))$ **using** *BsOrBsEqvBsBiOrBi* **by** *blast*
hence 6: $\vdash (di f \wedge di g \wedge bs(bi (\neg f) \vee bi (\neg g))) =$
 $(di f \wedge di g \wedge (bs (\neg f) \vee bs (\neg g)))$ **by** *auto*
have 7: $\vdash (di f \wedge di g \wedge (bs (\neg f) \vee bs (\neg g))) =$
 $((bs (\neg f) \wedge di f \wedge di g) \vee (di f \wedge bs (\neg g) \wedge di g))$ **by** *auto*

have 8: $\vdash \triangleright f = (bs (\neg f) \wedge di f)$ **using** *FstEqvBsNotAndDi* **by** *blast*
hence 9: $\vdash (bs (\neg f) \wedge di f \wedge di g) = (\triangleright f \wedge di g)$ **by** *auto*
have 10: $\vdash \triangleright g = (bs (\neg g) \wedge di g)$ **using** *FstEqvBsNotAndDi* **by** *blast*
hence 11: $\vdash (di f \wedge bs (\neg g) \wedge di g) = (di f \wedge \triangleright g)$ **by** *auto*
show *?thesis*
by (*metis* 11 4 6 7 9 *first-d-def* *inteq-reflection* *lift-and-com*)
qed

lemma *BiNotFstEqvBiNot*:

$\vdash finite \longrightarrow bi (\neg (\triangleright f)) = bi (\neg f)$

proof –

have 1: $\vdash finite \longrightarrow di f = di (\triangleright f)$ **using** *DiEqvDiFst* **by** *blast*
hence 2: $\vdash finite \longrightarrow (\neg (di f)) = (\neg (di (\triangleright f)))$ **by** *auto*
from 1 2 **show** *?thesis* **using** *NotDiEqvBiNot* **by** *fastforce*

qed

lemma *BsNotFstEqvBsNot*:

$\vdash bs (\neg (\triangleright f)) = bs (\neg f)$

proof –

have 1: $\vdash bs (\neg (\triangleright f)) = (empty \vee bi (\neg (\triangleright f)) \neg skip)$ **by** (*simp* *add: bs-d-def*)
have 2: $\vdash finite \longrightarrow bi (\neg (\triangleright f)) = bi (\neg f)$ **using** *BiNotFstEqvBiNot* **by** *fastforce*
hence 3: $\vdash bi (\neg (\triangleright f)) \neg skip = bi (\neg f) \neg skip$ **using** *LeftSChopEqvSChop*
by (*simp* *add: FiniteImpAnd LeftChopEqvChop schop-d-def*)
hence 4: $\vdash (empty \vee bi (\neg (\triangleright f)) \neg skip) = (empty \vee bi (\neg f) \neg skip)$ **by** *auto*
from 1 4 **show** *?thesis* **by** (*metis* *BsEqvEmptyOrBiSChopSkip* *inteq-reflection*)

qed

lemma *FstState*:

$\vdash \triangleright (init w) = (empty \wedge init w)$

proof –

have 1: $\vdash \triangleright (init w) = (init w \wedge bs (\neg (init w)))$ **by** (*simp* *add: first-d-def*)
hence 2: $\vdash \triangleright (init w) \longrightarrow init w$ **by** *auto*
have 3: $\vdash init w \wedge finite \longrightarrow bs (init w)$ **using** *StateImpBs* **by** *auto*
have 4: $\vdash \triangleright (init w) \longrightarrow bs (init w)$ **using** 2 3
using *FstImpFinite* **by** *fastforce*
have 5: $\vdash \triangleright (init w) \longrightarrow bs (\neg (init w))$ **using** 1 **by** *auto*
have 6: $\vdash \triangleright (init w) \longrightarrow bs (init w) \wedge bs (\neg (init w))$ **using** 4 5 **by** *fastforce*
have 7: $\vdash (bs (init w) \wedge bs (\neg (init w))) = (bs ((init w) \wedge \neg (init w)))$ **using** *BsAndEqv* **by** *blast*
have 8: $\vdash ((init w) \wedge \neg (init w)) = \#False$ **by** *auto*
hence 9: $\vdash (bs ((init w) \wedge \neg (init w))) = bs \#False$ **using** *BsEqvRule* **by** *blast*
have 10: $\vdash bs \#False = empty$ **using** *BsFalseEqvEmpty* **by** *auto*
have 11: $\vdash \triangleright (init w) \longrightarrow empty$ **using** 10 9 7 6 **by** *fastforce*
have 12: $\vdash \triangleright (init w) \longrightarrow empty \wedge init w$ **using** 11 2 **by** *fastforce*
have 13: $\vdash empty \wedge init w \longrightarrow empty$ **by** *auto*
hence 14: $\vdash empty \wedge init w \longrightarrow empty \vee bi (\neg (init w)) \neg skip$ **by** *auto*
hence 15: $\vdash empty \wedge init w \longrightarrow bs (\neg (init w))$ **by** (*simp* *add: bs-d-def*)
have 16: $\vdash empty \wedge init w \longrightarrow init w$ **by** *auto*
have 17: $\vdash empty \wedge init w \longrightarrow init w \wedge bs (\neg (init w))$ **using** 16 15 **by** *auto*
hence 18: $\vdash empty \wedge init w \longrightarrow \triangleright (init w)$ **by** (*simp* *add: first-d-def*)
from 12 18 **show** *?thesis* **by** *fastforce*

qed

lemma *FstStateAndBsNotEmpty*:

$\vdash (\triangleright (init\ w) \wedge bs\ (\neg\ empty)) = \triangleright (init\ w)$

proof –

have 1: $\vdash (\triangleright (init\ w) \wedge bs\ (\neg\ empty)) = (\triangleright (init\ w) \wedge bs\ more)$

by (*simp add: empty-d-def*)

have 2: $\vdash (\triangleright (init\ w) \wedge bs\ more) = (\triangleright (init\ w) \wedge empty)$

using *BsMoreEqvEmpty* **by** *fastforce*

have 3: $\vdash \triangleright (init\ w) = (empty \wedge (init\ w))$

using *FstState* **by** *blast*

hence 4: $\vdash (\triangleright (init\ w) \wedge empty) = (empty \wedge (init\ w) \wedge empty)$

by *auto*

have 5: $\vdash (empty \wedge (init\ w) \wedge empty) = (empty \wedge (init\ w))$

by *auto*

have 6: $\vdash (empty \wedge (init\ w)) = \triangleright (init\ w)$

using *FstState* **by** *fastforce*

from 1 2 4 5 6 **show** *?thesis* **by** *fastforce*

qed

lemma *FstStateImpFstStateOr*:

$\vdash \triangleright (init\ w) \longrightarrow \triangleright (init\ w \vee f)$

proof –

have 1: $\vdash \triangleright (init\ w) = (empty \wedge init\ w)$

using *FstState* **by** *blast*

have 2: $\vdash (empty \wedge init\ w) = (empty \wedge (empty \vee bi\ (\neg\ f) \frown skip) \wedge init\ w)$

by *auto*

have 3: $\vdash (empty \wedge (empty \vee bi\ (\neg\ f) \frown skip) \wedge init\ w) =$

$(empty \wedge bs\ (\neg\ f) \wedge init\ w)$

by (*simp add: bs-d-def*)

have 4: $\vdash (empty \wedge bs\ (\neg\ f) \wedge init\ w) = (empty \wedge init\ w \wedge bs\ (\neg\ f))$

by *auto*

have 5: $\vdash (empty \wedge init\ w) = \triangleright (init\ w)$

using *FstState* **by** *fastforce*

hence 6: $\vdash (empty \wedge init\ w \wedge bs\ (\neg\ f)) = (\triangleright (init\ w) \wedge bs\ (\neg\ f))$

by *auto*

have 7: $\vdash \triangleright (init\ w) \wedge bs\ (\neg\ f) \longrightarrow (\triangleright (init\ w) \wedge bs\ (\neg\ f)) \vee (\triangleright f \wedge bs\ (\neg (init\ w)))$

by *auto*

have 8: $\vdash \triangleright (init\ w \vee f) = ((\triangleright (init\ w) \wedge bs\ (\neg\ f)) \vee (\triangleright f \wedge bs\ (\neg (init\ w))))$

using *FstWithOrEqv* **by** *blast*

show *?thesis* **using** 1 3 8 **by** *fastforce*

qed

lemma *FstLenSame*:

$(\forall\ \sigma. (\sigma \models di\ (\triangleright f \wedge len(i)) \wedge di\ (\triangleright f \wedge len(j))) \longrightarrow (i=j))$

by (*simp add: DiLenFstsem FstLenSamesem*)

lemma *FstLenSame-1*:

$\vdash di\ (\triangleright f \wedge len(i)) \wedge di\ (\triangleright f \wedge len(j)) \longrightarrow (\#i=\#j)$

using *FstLenSame Valid-def* by *fastforce*

lemma *FstAndLenSame*:

$(\forall \sigma. (\sigma \models di ((\triangleright f \wedge g1) \wedge len(i)) \wedge di ((\triangleright f \wedge g2) \wedge len(j))) \longrightarrow (i=j))$

using *linorder-neqE-nat* by (*simp add: DiLenFstAndsem*) *blast*

lemma *FstAndLenSame-1*:

$\vdash di ((\triangleright f \wedge g1) \wedge len(i)) \wedge di ((\triangleright f \wedge g2) \wedge len(j)) \longrightarrow (\#i=\#j)$

using *FstAndLenSame Valid-def* by *fastforce*

lemma *FstLenSameChop*:

$(\forall \sigma. (\sigma \models ((\triangleright f \wedge g1) \wedge len(i));h1 \wedge ((\triangleright f \wedge g2) \wedge len(j));h2) \longrightarrow (i=j))$

proof

fix σ

show $(\sigma \models ((\triangleright f \wedge g1) \wedge len(i));h1 \wedge ((\triangleright f \wedge g2) \wedge len(j));h2) \longrightarrow (i=j)$

proof

assume 0: $(\sigma \models ((\triangleright f \wedge g1) \wedge len(i));h1 \wedge ((\triangleright f \wedge g2) \wedge len(j));h2)$

have 1: $(\sigma \models ((\triangleright f \wedge g1) \wedge len(i));h1)$ using 0 by *auto*

have 2: $(\sigma \models ((\triangleright f \wedge g1) \wedge len(i));h1) \longrightarrow$

$(\sigma \models ((\triangleright f \wedge g1) \wedge len(i));\#True)$ by (*metis ChopImpDi Valid-def di-d-def unl-lift2*)

have 3: $(\sigma \models di((\triangleright f \wedge g1) \wedge len(i)))$ using 1 2 by (*simp add: di-d-def*)

have 4: $(\sigma \models ((\triangleright f \wedge g2) \wedge len(j));h2)$ using 0 by *auto*

have 5: $(\sigma \models ((\triangleright f \wedge g2) \wedge len(j));h2) \longrightarrow$

$(\sigma \models ((\triangleright f \wedge g2) \wedge len(j));\#True)$ by (*metis ChopImpDi Valid-def di-d-def unl-lift2*)

have 6: $(\sigma \models di((\triangleright f \wedge g2) \wedge len(j)))$ using 4 5 by (*simp add: di-d-def*)

have 7: $(\sigma \models di((\triangleright f \wedge g1) \wedge len(i)) \wedge di((\triangleright f \wedge g2) \wedge len(j)))$ using 3 6 by *auto*

thus $(i=j)$ using *FstAndLenSame* by *blast*

qed

qed

lemma *FstLenSameChop-1*:

$\vdash ((\triangleright f \wedge g1) \wedge len(i));h1 \wedge ((\triangleright f \wedge g2) \wedge len(j));h2 \longrightarrow (\#i=\#j)$

using *FstLenSameChop Valid-def* by *fastforce*

lemma *DiImpExistsOneDiLenAndFst*:

assumes *nfinite* σ

$(\sigma \models di f)$

shows $(\exists! k. (\sigma \models di(\triangleright f \wedge len(k))))$

proof –

have 1: $(\sigma \models di(\triangleright f))$

using *assms DiEqvDiFst*

by (*metis (no-types, lifting) FiniteImpAnd finite-defs inteq-reflection unl-lift2*)

have 2: $(\sigma \models \triangleright f) = ((\sigma \models \triangleright f) \wedge (\exists k. (\sigma \models len(k))))$

using *AndExistsLen[of TEMP $\triangleright f$]*

using *assms len-defs nfinite-conv-nlength-enat* by *blast*

have 3: $((\sigma \models \triangleright f) \wedge (\exists k. (\sigma \models len(k)))) =$

$(\exists k. (\sigma \models \triangleright f) \wedge (\sigma \models len(k)))$

by *auto*

have 4: $(\sigma \models di(\triangleright f)) = (\exists k. (\sigma \models di(\triangleright f \wedge len(k))))$

using *assms* by (*metis DiLensem itl-defs(10)*)

have 5: $(\exists k. (\sigma \models di(\triangleright f \wedge len(k))))$
 using 1 using 4 by auto
 then obtain i where 6: $(\sigma \models di(\triangleright f \wedge len(i)))$ by blast
 from 5 obtain j where 7: $(\sigma \models di(\triangleright f \wedge len(j)))$ by blast
 have 8: $(\sigma \models di(\triangleright f \wedge len(i))) \wedge (\sigma \models di(\triangleright f \wedge len(j)))$
 using 6 7 by auto
 hence 9: $(\sigma \models di(\triangleright f \wedge len(i)) \wedge di(\triangleright f \wedge len(j)))$
 by simp
 hence 10: $i=j$
 using FstLenSame by blast
 have 11: $\bigwedge j. (\sigma \models di(\triangleright f \wedge len(j))) \longrightarrow (j=i)$
 using 9 10 using FstLenSame by auto
 thus $(\exists! k. (\sigma \models di(\triangleright f \wedge len(k))))$
 using 11 5 by blast
 qed

lemma *DiImpExistsOneDiLenAndFst-1*:
 $\vdash finite \longrightarrow di\ f \longrightarrow (\exists! k. (di(\triangleright f \wedge len(k))))$
 using *DiImpExistsOneDiLenAndFst* **unfolding** *Valid-def finite-defs* by fastforce

lemma *LFstAndDist-help*:
 $(\sigma \models ((\triangleright f \wedge g1) \wedge len(k)); h1 \wedge ((\triangleright f \wedge g2) \wedge len(k)); h2) =$
 $(\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge len(k)); (h1 \wedge h2))$
 using *LFixedAndDistr* by fastforce

lemma *LFstAndDist-help-1*:
 $(\exists k. (\sigma \models ((\triangleright f \wedge g1) \wedge len(k)); h1 \wedge ((\triangleright f \wedge g2) \wedge len(k)); h2)) =$
 $(\exists k. (\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge len(k)); (h1 \wedge h2)))$

proof

assume 0: $\exists k. \sigma \models ((\triangleright f \wedge g1) \wedge len(k)) ; h1 \wedge ((\triangleright f \wedge g2) \wedge len(k)) ; h2$
 obtain k where 1: $\sigma \models ((\triangleright f \wedge g1) \wedge len(k)) ; h1 \wedge ((\triangleright f \wedge g2) \wedge len(k)) ; h2$
 using 0 by auto
 hence 2: $(\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge len(k)); (h1 \wedge h2))$
 using *LFstAndDist-help* by blast
 show $(\exists k. (\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge len(k)); (h1 \wedge h2)))$
 using 2 by auto
 next
 assume 3: $(\exists k. (\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge len(k)); (h1 \wedge h2)))$
 obtain k where 4: $(\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge len(k)); (h1 \wedge h2))$
 using 3 by auto
 hence 5: $(\sigma \models ((\triangleright f \wedge g1) \wedge len(k)); h1 \wedge ((\triangleright f \wedge g2) \wedge len(k)); h2)$
 using *LFstAndDist-help* by blast
 show $(\exists k. (\sigma \models ((\triangleright f \wedge g1) \wedge len(k)); h1 \wedge ((\triangleright f \wedge g2) \wedge len(k)); h2))$
 using 5 by auto
 qed

lemma *LFstAndDistrsem*:
 $(\sigma \models ((\triangleright f \wedge g1); h1 \wedge (\triangleright f \wedge g2); h2) = (\triangleright f \wedge g1 \wedge g2); (h1 \wedge h2)))$
proof –

```

have 01: (σ ⊨ (▷f ∧ g1);h1) = (σ ⊨ (▷f ∧ g1)¬h1)
  using FstImpFinite schop-d-def
  by (metis (no-types, lifting) FstFstAndEqvFstAnd Prop10 inteq-reflection)
have 02: (σ ⊨ (▷f ∧ g2);h2) = (σ ⊨ (▷f ∧ g2)¬h2)
  using FstImpFinite schop-d-def
  by (metis (no-types, lifting) FstFstAndEqvFstAnd Prop10 inteq-reflection)
have 1: (σ ⊨ (▷f ∧ g1)¬h1) = (∃ i. (σ ⊨ ((▷f ∧ g1) ∧ len(i));h1) )
  using AndExistsLenChop[of TEMP (▷ f ∧ g1)] by fastforce
have 2: (σ ⊨ (▷f ∧ g2)¬h2) = (∃ j. (σ ⊨ ((▷f ∧ g2) ∧ len(j));h2) )
  using AndExistsLenChop[of TEMP (▷ f ∧ g2)] by fastforce
have 3: (σ ⊨ (▷f ∧ g1)¬h1 ∧ (▷f ∧ g2)¬h2) =
  ( (∃ i j. (σ ⊨ ((▷f ∧ g1) ∧ len(i));h1 ∧ ((▷f ∧ g2) ∧ len(j));h2) )
  )
  using 1 2 by auto
have 4: ( (∃ i j. (σ ⊨ ((▷f ∧ g1) ∧ len(i));h1 ∧ ((▷f ∧ g2) ∧ len(j));h2) )
  ) =
  ( (∃ k. (σ ⊨ ((▷f ∧ g1) ∧ len(k));h1 ∧ ((▷f ∧ g2) ∧ len(k));h2) )
  )
  using FstLenSameChop by blast
have 5: (∃ k. (σ ⊨ ((▷f ∧ g1) ∧ len(k));h1 ∧ ((▷f ∧ g2) ∧ len(k));h2)) =
  (∃ k. (σ ⊨ (((▷f ∧ g1) ∧ (▷f ∧ g2)) ∧ len(k));(h1 ∧ h2) ))
  using LFstAndDist-help-1 by blast
have 6 : (∃ k. (σ ⊨ (((▷f ∧ g1) ∧ (▷f ∧ g2)) ∧ len(k));(h1 ∧ h2) )) =
  (σ ⊨ ((▷f ∧ g1) ∧ (▷f ∧ g2))¬(h1 ∧ h2))
  using AndExistsLenChop[of TEMP ((▷ f ∧ g1) ∧ ▷ f ∧ g2)] by fastforce
have 7 : (σ ⊨ ((▷f ∧ g1) ∧ (▷f ∧ g2));(h1 ∧ h2)) =
  (σ ⊨ (▷f ∧ g1 ∧ g2);(h1 ∧ h2))
  by (auto simp add: chop-defs)
have 8: (σ ⊨ (▷f ∧ g1 ∧ g2)¬(h1 ∧ h2)) = (σ ⊨ (▷f ∧ g1 ∧ g2);(h1 ∧ h2))
  using FstImpFinite schop-d-def
  by (metis (no-types, lifting) FstFstAndEqvFstAnd Prop10 inteq-reflection)
have 9: (σ ⊨ (▷f ∧ g1)¬h1 ∧ (▷f ∧ g2)¬h2) =
  (σ ⊨ (▷f ∧ g1 ∧ g2);(h1 ∧ h2))
  using 01 02 3 4 5 6 7 8
  by (metis (no-types, lifting) FstFstAndEqvFstAnd FstImpFinite Prop10 inteq-reflection schop-d-def)
show ?thesis
  unfolding schop-d-def using 01 02 9 by auto
qed

```

lemma LFstAndDistr:

```

⊢ ((▷f ∧ g1);h1 ∧ (▷f ∧ g2);h2) = (▷f ∧ g1 ∧ g2);(h1 ∧ h2)
using LFstAndDistrsem by fastforce

```

lemma LFstAndDistrA:

```

⊢ ((▷f ∧ g1);h ∧ (▷f ∧ g2);h) = (▷f ∧ g1 ∧ g2);h

```

proof –

```

have 1: ⊢ ((▷f ∧ g1);h ∧ (▷f ∧ g2);h) = (▷f ∧ g1 ∧ g2);(h ∧ h) using LFstAndDistr by blast

```

```

have 2: ⊢ (▷f ∧ g1 ∧ g2);(h ∧ h) = (▷f ∧ g1 ∧ g2);h by auto

```

```

from 1 2 show ?thesis by auto

```

qed

lemma *LFstAndDistrB*:

$\vdash ((\triangleright f \wedge g); h1 \wedge (\triangleright f \wedge g); h2) = (\triangleright f \wedge g); (h1 \wedge h2)$

proof –

have 1: $\vdash ((\triangleright f \wedge g); h1 \wedge (\triangleright f \wedge g); h2) = (\triangleright f \wedge g \wedge g); (h1 \wedge h2)$ **using** *LFstAndDistr* **by** *blast*

have 2: $\vdash (\triangleright f \wedge g \wedge g); (h1 \wedge h2) = (\triangleright f \wedge g); (h1 \wedge h2)$ **by** *auto*

from 1 2 **show** *?thesis* **by** *auto*

qed

lemma *LFstAndDistrC*:

$\vdash ((\triangleright f); h1 \wedge (\triangleright f); h2) = (\triangleright f); (h1 \wedge h2)$

proof –

have 1: $\vdash ((\triangleright f \wedge \#True); h1 \wedge (\triangleright f \wedge \#True); h2) = (\triangleright f \wedge \#True \wedge \#True); (h1 \wedge h2)$
using *LFstAndDistr* **by** *blast*

have 2: $\vdash (\triangleright f \wedge \#True); h1 = (\triangleright f); h1$
by *auto*

have 3: $\vdash (\triangleright f \wedge \#True); h2 = (\triangleright f); h2$
by *auto*

have 4: $\vdash (\triangleright f \wedge \#True \wedge \#True); (h1 \wedge h2) = (\triangleright f); (h1 \wedge h2)$
by *auto*

from 1 2 3 4 **show** *?thesis* **by** *auto*

qed

lemma *LFstAndDistrD*:

$\vdash (di(\triangleright f \wedge g1) \wedge di(\triangleright f \wedge g2)) = di(\triangleright f \wedge g1 \wedge g2)$

proof –

have 1: $\vdash ((\triangleright f \wedge g1); \#True \wedge (\triangleright f \wedge g2); \#True) = (\triangleright f \wedge g1 \wedge g2); (\#True \wedge \#True)$
using *LFstAndDistr* **by** *blast*

have 2: $\vdash (\triangleright f \wedge g1); \#True = di(\triangleright f \wedge g1)$
by (*simp add: di-d-def*)

have 3: $\vdash (\triangleright f \wedge g2); \#True = di(\triangleright f \wedge g2)$
by (*simp add: di-d-def*)

have 4: $\vdash (\triangleright f \wedge g1 \wedge g2); (\#True \wedge \#True) = di(\triangleright f \wedge g1 \wedge g2)$
by (*simp add: di-d-def*)

from 1 2 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *NotFstChop*:

$\vdash (\neg(\triangleright f; g)) = (\neg(di(\triangleright f)) \vee (\triangleright f; (\neg g)))$

proof –

have 1: $\vdash g \longrightarrow \#True$ **by** *auto*

hence 2: $\vdash \triangleright f; g \longrightarrow \triangleright f; \#True$ **using** *RightChopImpChop* **by** *blast*

hence 3: $\vdash \triangleright f; g \longrightarrow di(\triangleright f)$ **by** (*simp add: di-d-def*)

hence 4: $\vdash \neg(di(\triangleright f)) \longrightarrow \neg(\triangleright f; g)$ **by** *auto*

have 5: $\vdash (\triangleright f; (\neg g)) \longrightarrow \neg(\triangleright f; g) = ((\triangleright f; (\neg g)) \wedge (\triangleright f; g) \longrightarrow \#False)$ **by** *auto*

have 6: $\vdash ((\triangleright f; (\neg g)) \wedge (\triangleright f; g)) = \triangleright f; (\neg g \wedge g)$ **using** *LFstAndDistrC* **by** *blast*

have 06: $\vdash \triangleright f \longrightarrow finite$ **by** (*simp add: FstImpFinite*)

have 7: $\vdash \neg(\triangleright f; (\neg g \wedge g))$ **using** 06 **by** (*auto simp add: Valid-def itl-defs*)

have 8: $\vdash \triangleright f; (\neg g) \longrightarrow \neg(\triangleright f; g)$ **using** 5 6 7 **by** *fastforce*
have 9: $\vdash \neg(di(\triangleright f)) \vee (\triangleright f; (\neg g)) \longrightarrow \neg(\triangleright f; g)$ **using** 4 8 **by** *fastforce*
have 10: $\vdash di(\triangleright f) \vee \neg(di(\triangleright f))$ **by** *auto*
hence 11: $\vdash (\triangleright f; \#True) \vee \neg(di(\triangleright f))$ **by** (*simp add: di-d-def*)
hence 12: $\vdash (\triangleright f; (g \vee \neg g)) \vee \neg(di(\triangleright f))$ **by** *auto*
have 13: $\vdash (\triangleright f; (g \vee \neg g)) = ((\triangleright f; g) \vee (\triangleright f; (\neg g)))$ **using** *ChopOrEqv* **by** *fastforce*
have 14: $\vdash ((\triangleright f; g) \vee (\triangleright f; (\neg g))) \vee \neg(di(\triangleright f))$ **using** 12 13 **by** *fastforce*
hence 15: $\vdash \neg(\triangleright f; g) \longrightarrow \neg(di(\triangleright f)) \vee (\triangleright f; (\neg g))$ **by** *auto*
from 9 15 **show** *?thesis* **by** *fastforce*
qed

lemma *BsNotFstChop*:

$\vdash bs(\neg(\triangleright f; g)) = (empty \vee (finite \wedge \neg(di(\triangleright f))) \vee \triangleright f; (bs(\neg g)))$

proof –

have 01: $\vdash (\triangleright f; g) = (\triangleright f \frown g)$
by (*simp add: FstImpFinite LeftChopEqvChop Prop10 schop-d-def*)
have 1: $\vdash bs(\neg(\triangleright f; g)) = (empty \vee bi(\neg(\triangleright f \frown g)) \frown skip)$
using 01 **unfolding** *bs-d-def* **by** (*metis WPrevAndFiniteEqv inteq-reflection*)
have 000: $\vdash (empty \vee bi(\neg(\triangleright f \frown g)) \frown skip) = (empty \vee bf(\neg(\triangleright f \frown g)) \frown skip)$
by (*metis FiniteImpAnd FiniteImpBfEqvBi WPrevAndFiniteEqv inteq-reflection*)
have 2: $\vdash (empty \vee bf(\neg(\triangleright f \frown g)) \frown skip) = (empty \vee (\neg(df(\triangleright f \frown g))) \frown skip)$
by (*simp add: bf-d-def*)
have 3: $\vdash (empty \vee (\neg(df(\triangleright f \frown g))) \frown skip) = (empty \vee (\neg((\triangleright f \frown g) \frown \#True)) \frown skip)$
by (*simp add: df-d-def*)
have 4: $\vdash (\neg((\triangleright f \frown g) \frown \#True)) \frown skip = (\neg(\triangleright f \frown (g \frown \#True))) \frown skip$
by (*metis LeftSChopEqvSChop SChopAssoc int-simps(14) inteq-reflection*)
hence 5: $\vdash (empty \vee (\neg((\triangleright f \frown g) \frown \#True)) \frown skip) = (empty \vee (\neg(\triangleright f \frown (g \frown \#True))) \frown skip)$
by *auto*
have 6: $\vdash (empty \vee (\neg(\triangleright f \frown (g \frown \#True))) \frown skip) = (empty \vee (\neg(\triangleright f \frown df(g))) \frown skip)$
by (*simp add: df-d-def*)
have 7: $\vdash (empty \vee (\neg(\triangleright f \frown df(g))) \frown skip) = (empty \vee \neg(\neg((\triangleright f \frown df(g))) \frown skip))$
by *auto*
have 8: $\vdash (\neg(\neg((\neg(\triangleright f \frown df(g))) \frown skip))) = (\neg(empty \vee inf \vee (\triangleright f \frown df(g)) \frown skip))$
using *NotNotSChopSkip* **by** *fastforce*
hence 9: $\vdash (empty \vee \neg(\neg((\neg(\triangleright f \frown df(g))) \frown skip))) = (empty \vee \neg(empty \vee inf \vee (\triangleright f \frown df(g)) \frown skip))$
by *auto*
have 09: $\vdash (\neg(empty \vee inf \vee (\triangleright f \frown df(g)) \frown skip)) = (more \wedge finite \wedge \neg((\triangleright f \frown df(g)) \frown skip))$
unfolding *empty-d-def finite-d-def* **by** *fastforce*
have 10: $\vdash (empty \vee \neg(empty \vee inf \vee (\triangleright f \frown df(g)) \frown skip)) = (empty \vee (more \wedge finite \wedge \neg((\triangleright f \frown df(g)) \frown skip)))$
using 6 7 8 9 09 **by** *auto*
have 11: $\vdash (empty \vee (more \wedge finite \wedge \neg((\triangleright f \frown df(g)) \frown skip))) = (empty \vee (finite \wedge \neg((\triangleright f \frown df(g)) \frown skip)))$
by (*auto simp add: empty-d-def*)
have 12: $\vdash (empty \vee (finite \wedge \neg((\triangleright f \frown df(g)) \frown skip))) = (empty \vee (finite \wedge \neg(\triangleright f \frown (df(g) \frown skip))))$
by (*metis 11 SChopAssoc inteq-reflection*)
have 012: $\vdash (\neg(\triangleright f \frown (inf \vee df(g) \frown skip))) = (\neg(\triangleright f \frown inf) \wedge \neg(\triangleright f \frown (df(g) \frown skip)))$
using *SChopOrEqv* **by** *fastforce*
have 013: $\vdash (\neg(\triangleright f \frown inf)) = (\neg di(\triangleright f) \vee \triangleright f \frown (\neg inf))$
by (*metis FstImpFinite NotFstChop Prop10 inteq-reflection schop-d-def*)

have 014: $\vdash (empty \vee (finite \wedge \neg(\triangleright f \wedge (df(g) \wedge skip)))) =$
 $(empty \vee (finite \wedge (\neg(di \triangleright f)) \vee (\triangleright f \wedge (\neg(df(g) \wedge skip)))))$
using *NotFstChop*
by (*metis FstImpFinite Prop10 int-simps(1) inteq-reflection schop-d-def*)
have 015: $\vdash (\neg(df(g) \wedge skip)) = (empty \vee inf \vee bf(\neg g) \wedge skip)$
by (*metis NotDfEqvBfNot NotSChopNotSkip inteq-reflection*)
have 16: $\vdash (\triangleright f \wedge (\neg(df(g) \wedge skip))) = (\triangleright f \wedge (empty \vee inf \vee bf(\neg g) \wedge skip))$
by (*simp add: 015 RightSChopEqvSChop*)
have 0160: $\vdash bf(\neg g) \wedge skip = bi(\neg g) \wedge skip$
by (*simp add: FiniteImpAnd FiniteImpBfEqvBi LeftChopEqvChop schop-d-def*)
have 016: $\vdash (empty \vee inf \vee bf(\neg g) \wedge skip) = (inf \vee bs(\neg g))$
unfolding *bs-d-def* **using** 0160 **by** *fastforce*
have 017: $\vdash (\triangleright f \wedge (\neg(df(g) \wedge skip))) = (\triangleright f \wedge (inf \vee bs(\neg g)))$
by (*metis 016 16 inteq-reflection*)
have 018: $\vdash (\triangleright f \wedge (inf \vee bs(\neg g))) = (\triangleright f \wedge inf \vee \triangleright f \wedge (bs(\neg g)))$
by (*simp add: SChopOrEqv*)
have 019: $\vdash (finite \wedge (\neg(di \triangleright f)) \vee (\triangleright f \wedge (\neg(df(g) \wedge skip)))) =$
 $((finite \wedge \neg(di \triangleright f))) \vee (finite \wedge (\triangleright f \wedge (\neg(df(g) \wedge skip))))$
by *force*
have 020: $\vdash (finite \wedge (\triangleright f \wedge (\neg(df(g) \wedge skip)))) = (finite \wedge (\triangleright f \wedge inf \vee \triangleright f \wedge (bs(\neg g))))$
using 017 018 **by** *fastforce*
have 021: $\vdash \triangleright f \wedge inf \longrightarrow inf$
by (*metis AndChopB FiniteChopInfEqvInf Prop11 lift-imp-trans schop-d-def*)
have 022: $\vdash (finite \wedge (\triangleright f \wedge inf \vee \triangleright f \wedge (bs(\neg g)))) = (finite \wedge \triangleright f \wedge (bs(\neg g)))$
using 021 **unfolding** *finite-d-def* **by** *fastforce*
have 023: $\vdash \triangleright f \wedge (bs(\neg g)) \longrightarrow finite$
by (*metis AndChopB EmptyImpFinite FiniteChopSkipImpFinite Prop02 SChopImpFinite bs-d-def*
lift-imp-trans schop-d-def)
have 024: $\vdash (finite \wedge \triangleright f \wedge (bs(\neg g))) = \triangleright f \wedge (bs(\neg g))$
using 023 **by** *fastforce*
have 025: $\vdash (empty \vee (finite \wedge (\neg(di \triangleright f)) \vee (\triangleright f \wedge (\neg(df(g) \wedge skip))))) =$
 $(empty \vee (finite \wedge \neg(di \triangleright f))) \vee \triangleright f \wedge (bs(\neg g))$
by (*metis 014 019 020 022 024 inteq-reflection*)
have 026: $\vdash bs(\neg(\triangleright f;g)) = (empty \vee (finite \wedge (\neg(di \triangleright f)) \vee (\triangleright f \wedge (\neg(df(g) \wedge skip)))))$
using 014 09 1 11 12 2 3 5 6 7 8
by (*metis 000 inteq-reflection*)
show ?thesis using 025 026 by (*metis FstImpFinite Prop10 inteq-reflection schop-d-def*)
qed

lemma *FstFstChopEqvFstChopFst*:

$\vdash \triangleright(\triangleright f;g) = \triangleright f; \triangleright g$

proof –

have 1: $\vdash \triangleright(\triangleright f;g) = ((\triangleright f;g) \wedge bs(\neg(\triangleright f;g)))$

by (*simp add: first-d-def*)

have 2: $\vdash bs(\neg(\triangleright f;g)) = (empty \vee (finite \wedge \neg(di \triangleright f))) \vee (\triangleright f; bs(\neg g))$

using *BsNotFstChop* **by** *auto*

hence 3: $\vdash ((\triangleright f;g) \wedge bs(\neg(\triangleright f;g))) = ((\triangleright f;g) \wedge (empty \vee (finite \wedge \neg(di \triangleright f))) \vee (\triangleright f; bs(\neg g)))$

by *auto*

have 4: $\vdash ((\triangleright f;g) \wedge (empty \vee (finite \wedge \neg(di \triangleright f))) \vee (\triangleright f; bs(\neg g))) =$
 $((\triangleright f;g) \wedge empty) \vee ((\triangleright f;g) \wedge \neg(di \triangleright f)) \wedge finite \vee ((\triangleright f;g) \wedge (\triangleright f; bs(\neg g)))$

by *auto*
have 5: $\vdash \neg((\triangleright f;g) \wedge \neg(di(\triangleright f)) \wedge finite)$
 using *ChopImpDi* by *fastforce*
hence 6: $\vdash (((\triangleright f;g) \wedge empty) \vee ((\triangleright f;g) \wedge \neg(di(\triangleright f)) \wedge finite) \vee ((\triangleright f;g) \wedge (\triangleright f;bs(\neg g)))) =$
 $((\triangleright f;g) \wedge empty) \vee ((\triangleright f;g) \wedge (\triangleright f;bs(\neg g))))$
 by *auto*
have 7: $\vdash ((\triangleright f;g) \wedge (\triangleright f;(bs(\neg g)))) = ((\triangleright f;(g \wedge (bs(\neg g)))))$
 using *LFstAndDistrC* by *blast*
hence 8: $\vdash (((\triangleright f;g) \wedge empty) \vee ((\triangleright f;g) \wedge (\triangleright f;(bs(\neg g))))) =$
 $((\triangleright f;g) \wedge empty) \vee ((\triangleright f;(g \wedge (bs(\neg g)))))$
 by *auto*
have 9: $\vdash (((\triangleright f;g) \wedge empty) \vee ((\triangleright f;(g \wedge (bs(\neg g))))) = (((\triangleright f;g) \wedge empty) \vee \triangleright f;\triangleright g)$
 by (*simp add: first-d-def*)
have 10: $\vdash ((\triangleright f;g) \wedge empty) = ((\triangleright f;\triangleright g) \wedge empty)$
 using *FstChopEmptyEqvFstChopFstEmpty*
 by (*metis ChopEmptyAndEmpty SChopEmptyAndEmpty inteq-reflection*)
hence 11: $\vdash (((\triangleright f;g) \wedge empty) \vee \triangleright f;\triangleright g) = (((\triangleright f;\triangleright g) \wedge empty) \vee \triangleright f;\triangleright g)$
 by *auto*
have 12: $\vdash (((\triangleright f;\triangleright g) \wedge empty) \vee \triangleright f;\triangleright g) = \triangleright f;\triangleright g$
 by *auto*
from 1 3 4 6 8 9 11 12 **show** *?thesis* by (*metis inteq-reflection*)
qed

lemma *FstFixFst*:

$\vdash \triangleright(\triangleright f) = \triangleright f$

proof –

have 1: $\vdash \triangleright f = (\triangleright f);empty$ using *ChopEmpty* by (*metis int-eq*)
hence 2: $\vdash \triangleright(\triangleright f) = \triangleright((\triangleright f);empty)$ using *FstEqvRule* by *blast*
have 3: $\vdash \triangleright((\triangleright f);empty) = \triangleright f;\triangleright empty$ using *FstFstChopEqvFstChopFst* by *auto*
have 4: $\vdash \triangleright f;\triangleright empty = \triangleright f;empty$ using *FstEmpty* using *RightChopEqvChop* by *blast*
have 5: $\vdash \triangleright f;empty = \triangleright f$ using *ChopEmpty* by *blast*
from 2 3 4 5 **show** *?thesis* by *fastforce*

qed

lemma *FstCSEqvEmpty*:

$\vdash \triangleright(f^*) = empty$

proof –

have 1: $\vdash \triangleright(f^*) = \triangleright(empty \vee ((f \wedge more);f^*))$ using *ChopstarEqv FstEqvRule* by *blast*
from 1 **show** *?thesis* using *FstEmptyOrEqvEmpty* by *fastforce*

qed

lemma *FstIterFixFst*:

$\vdash fpower(\triangleright f) n = \triangleright(fpower(\triangleright f) n)$

proof

(*induct n*)

case 0

then show *?case*

proof –

have 1: $\vdash fpower(\triangleright f) 0 = empty$
 by (*simp add: fpower-d-def*)

```

have 2:  $\vdash \text{empty} = \triangleright \text{empty}$  using FstEmpty by auto
have 3:  $\vdash \triangleright \text{empty} = \triangleright(\text{fpower } (\triangleright f) \ 0)$ 
by (metis 1 2 inteq-reflection)
from 1 2 3 show ?thesis
by fastforce
qed
next
case (Suc n)
then show ?case
proof –
  have 4:  $\vdash (\text{fpower } (\triangleright f) (\text{Suc } n)) = (\triangleright f) ; (\text{fpower } (\triangleright f) \ n)$ 
    by (metis FstImpFinite LeftChopEqvChop Prop10 Prop11 fpower-d-def wpow-Suc)
  have 5:  $\vdash (\triangleright f) ; (\text{fpower } (\triangleright f) \ n) = (\triangleright f) ; \triangleright (\text{fpower } (\triangleright f) \ n)$ 
    using RightChopEqvChop Suc.hyps by blast
  have 6:  $\vdash (\triangleright f) ; \triangleright (\text{fpower } (\triangleright f) \ n) = \triangleright(\triangleright f; (\text{fpower } (\triangleright f) \ n))$ 
    using FstFstChopEqvFstChopFst by fastforce
  have 7:  $\vdash \triangleright(\triangleright f; (\text{fpower } (\triangleright f) \ n)) = \triangleright(\text{fpower } (\triangleright f) (\text{Suc } n))$ 
    by (metis 4 6 inteq-reflection)
  from 4 5 6 7 show ?thesis by fastforce
qed
qed

lemma DsImpNotFst:
 $\vdash \text{ds } f \longrightarrow (\neg(\triangleright f))$ 
proof –
  have 1:  $\vdash (\text{ds } f \wedge \triangleright f) = (\text{ds } f \wedge (f \wedge \text{bs } (\neg f)))$  by (simp add: first-d-def)
  have 2:  $\vdash (\text{ds } f \wedge (f \wedge \text{bs } (\neg f))) = (\text{ds } f \wedge f \wedge \neg(\text{ds } f))$  using NotDsEqvBsNot by fastforce
  from 1 2 show ?thesis by fastforce
qed

lemma FstLenAndEqvLenAnd:
 $\vdash \triangleright(\text{len}(k) \wedge f) = (\text{len}(k) \wedge f)$ 
proof –
  have 1:  $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow \text{ds } (\text{len}(k))$ 
    using DsAndImpElimL by fastforce
  have 01:  $\vdash \text{ds}(\text{len}(k) \wedge f) = (\text{finite} \longrightarrow \text{di}(\text{len}(k) \wedge f) \frown \text{skip})$ 
    by (simp add: DsDi)
  have 02:  $\vdash \text{len } k \longrightarrow \text{finite}$ 
    by (simp add: len-k-finite)
  have 03:  $\vdash \text{di}(\text{len}(k) \wedge f) \frown \text{skip} \longrightarrow (\text{di}(\text{len}(k))) \frown \text{skip}$ 
    by (simp add: DiAndA LeftSChopImpSChop)
  hence 2:  $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow (\text{di}(\text{len}(k))) \frown \text{skip}$ 
    using 01 02 03 by fastforce
  hence 3:  $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow ((\text{len}(k) \frown \# \text{True})) \frown \text{skip}$ 
    unfolding di-d-def
    by (metis Prop10 inteq-reflection len-k-finite schop-d-def)
  hence 4:  $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow (\text{len}(k) \frown (\# \text{True} \frown \text{skip}))$ 
    using SChopAssoc by fastforce
  hence 5:  $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow (\text{len}(k) \frown (\text{skip} \frown \text{finite}))$ 
    by (metis Prop10 SkipFiniteEqvFiniteSkip WPowerstar-ext WPowerstar-skip-finite int-simps(17))

```

inteq-reflection schop-d-def
hence 6: $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow (\text{len}(k) \frown (\text{skip} \frown \text{finite})) \wedge \text{len}(k)$
by *auto*
hence 7: $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow (\text{len}(k) \frown (\text{skip} \frown \text{finite})) \wedge \text{len}(k) \frown \text{empty}$
by (*metis ChopEmpty Prop12 inteq-reflection len-k-finite lift-imp-trans schop-d-def*)
hence 8: $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow (\text{len}(k) \frown ((\text{skip} \frown \text{finite}) \wedge \text{empty}))$
by (*metis LenSChopAnd Prop11 lift-imp-trans*)
have 08: $\vdash \neg((\text{skip} \frown \text{finite}) \wedge \text{empty})$
by (*metis ChopEmptyAndEmpty FmoreEqvSkipChopFinite NotFmoreAndEmpty SChopEmptyAndEmpty inteq-reflection lift-and-com*)
have 09: $\vdash ((\text{skip} \frown \text{finite}) \wedge \text{empty}) = \#False$
using 08 by (*simp add: Prop11*)
have 010: $\vdash \text{len}(k) \frown ((\text{skip} \frown \text{finite}) \wedge \text{empty}) = \#False$
by (*metis 09 SChopRightFalse inteq-reflection*)
have 9: $\vdash \neg(\text{len}(k) \frown ((\text{skip} \frown \text{finite}) \wedge \text{empty}))$
using 010 by *auto*
have 10: $\vdash \text{len}(k) \wedge f \longrightarrow \neg(\text{ds}(\text{len}(k) \wedge f))$
using 8 9 by *fastforce*
hence 11: $\vdash \text{len}(k) \wedge f \longrightarrow \text{bs } (\neg(\text{len}(k) \wedge f))$
using *NotDsEqvBsNot* **by** *fastforce*
hence 12: $\vdash \text{len}(k) \wedge f \longrightarrow (\text{len}(k) \wedge f) \wedge \text{bs } (\neg(\text{len}(k) \wedge f))$
by *auto*
hence 13: $\vdash \text{len}(k) \wedge f \longrightarrow \triangleright(\text{len}(k) \wedge f)$
by (*simp add: first-d-def*)
have 14: $\vdash \triangleright(\text{len}(k) \wedge f) \longrightarrow \text{len}(k) \wedge f$
by (*auto simp add: first-d-def*)
from 13 14 show ?thesis by *fastforce*
qed

lemma *FstAndElimL:*

$\vdash \triangleright f \longrightarrow f$
by (*auto simp add: first-d-def*)

lemma *FstImpNotDiChopSkip:*

$\vdash \triangleright f \longrightarrow \neg(\text{di } f \frown \text{skip})$
proof –
have 1: $\vdash \triangleright f \longrightarrow \text{bs } (\neg f)$ **by** (*auto simp add: first-d-def*)
hence 2: $\vdash \triangleright f \longrightarrow \neg(\text{ds } f)$ **using** *NotDsEqvBsNot* **by** *fastforce*
have 3: $\vdash \text{ds } f = (\text{finite} \longrightarrow \text{di } f \frown \text{skip})$ **using** *DsDi* **by** *blast*
hence 4: $\vdash (\neg(\text{ds } f)) = (\text{finite} \wedge \neg(\text{di } f \frown \text{skip}))$ **by** *auto*
from 2 4 show ?thesis by *fastforce*
qed

lemma *FstImpNotDiChopSkipB:*

$\vdash \triangleright f \longrightarrow \neg(\text{di } (f \frown \text{skip}))$
proof –
have 1: $\vdash \triangleright f \longrightarrow \text{bs } (\neg f)$
by (*auto simp add: first-d-def*)
hence 2: $\vdash \triangleright f \longrightarrow \neg(\text{ds } f)$
using *NotDsEqvBsNot* **by** *fastforce*

```

have 3:  $\vdash ds\ f = (finite \longrightarrow di\ f \frown skip)$ 
  using DsDi by blast
have 4:  $\vdash di\ f \frown skip = (f \frown \#True) \frown skip$ 
  by (metis ChopTrueAndFiniteEqvAndFiniteChopFinite SChopFiniteEqvSChopTrueAndFinite
    di-d-def int-simps(1) inteq-reflection schop-d-def)
have 5:  $\vdash (f \frown \#True) \frown skip = f \frown (\#True \frown skip)$ 
  by (meson Prop11 SChopAssoc)
have 6:  $\vdash f \frown (\#True \frown skip) = f \frown (skip \frown finite)$ 
  by (metis ChopTrueAndFiniteEqvAndFiniteChopFinite DiSkipEqvMore DiamondSChopdef
    FmoreEqvSkipChopFinite RightSChopEqvSChop SkipFiniteEqvFiniteSkip di-d-def fmore-d-def
    inteq-reflection schop-d-def sometimes-d-def)
have 7:  $\vdash f \frown (skip \frown finite) = (f \frown skip) \frown finite$ 
  using SChopAssoc by blast
have 8:  $\vdash (f \frown skip) \frown finite = (di(f \frown skip) \wedge finite)$ 
  by (metis ChopTrueAndFiniteEqvAndFiniteChopFinite Prop11 di-d-def schop-d-def)
have 9:  $\vdash (\neg(ds\ f)) = (\neg(di(f \frown skip)) \wedge finite)$ 
  using 3 4 5 6 7 8 by fastforce
from 2 9 show ?thesis by fastforce
qed

```

lemma *FstImpDiEqv*:

$\vdash \triangleright f \longrightarrow (di\ f = f)$

proof –

```

have 1:  $\vdash \triangleright f \longrightarrow \neg(di\ f \frown skip)$  using FstImpNotDiChopSkip by blast
have 2:  $\vdash di\ f \wedge finite \longrightarrow f \vee (di\ f \frown skip)$  using DiEqvOrDiChopSkipB
using DiFiniteEqv by fastforce
have 3:  $\vdash \triangleright f \wedge di\ f \longrightarrow (f \vee (di\ f \frown skip)) \wedge \neg(di\ f \frown skip)$  using 1 2
using FstAndElimL by fastforce
have 4:  $\vdash ((f \vee (di\ f \frown skip)) \wedge \neg(di\ f \frown skip)) = (f \wedge \neg(di\ f \frown skip))$  by auto
have 5:  $\vdash \triangleright f \wedge di\ f \longrightarrow f \wedge \neg(di\ f \frown skip)$  using 3 4 by fastforce
hence 6:  $\vdash \triangleright f \wedge di\ f \longrightarrow f$  by fastforce
hence 7:  $\vdash \triangleright f \longrightarrow (di\ f \longrightarrow f)$  using FstAndElimL by fastforce
have 8:  $\vdash f \longrightarrow di\ f$  using DiIntro by auto
hence 9:  $\vdash \triangleright f \longrightarrow (f \longrightarrow (di\ f))$  by auto
from 7 9 show ?thesis by fastforce
qed

```

lemma *FstAndDiFstAndEqvFstAnd*:

$\vdash (\triangleright f \wedge di(\triangleright f \wedge g)) = (\triangleright f \wedge g)$

proof –

```

have 1:  $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow \triangleright f$ 
  by auto
have 2:  $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow di(\triangleright f \wedge g)$ 
  by auto
have 3:  $\vdash finite \longrightarrow di(\triangleright f \wedge g) = ((\triangleright f \wedge g) \vee di((\triangleright f \wedge g) \frown skip))$ 
  using DiEqvOrDiChopSkipA
  by (metis (no-types, lifting) DiEqvDiFst FstFstAndEqvFstAnd FstImpFinite Prop10 inteq-reflection
    s chop-d-def)
have 4:  $\vdash finite \longrightarrow di((\triangleright f \wedge g) \frown skip) = ((\triangleright f \wedge g) \frown skip) \frown finite$ 
  unfolding di-d-def schop-d-def

```

using *ChopTrueAndFiniteEqvAndFiniteChopFinite* **by** *fastforce*
have 04: $\vdash \triangleright f \longrightarrow \text{finite}$
by (*simp add: FstImpFinite*)
have 5: $\vdash \triangleright f \wedge \text{di}(\triangleright f \wedge g) \longrightarrow (\triangleright f \wedge g) \vee ((\triangleright f \wedge g) \frown \text{skip}) \frown \text{finite}$
using 2 3 4 04 **by** *fastforce*
have 6: $\vdash \triangleright f \wedge g \longrightarrow f$
using *FstAndElimL* **by** *fastforce*
hence 7: $\vdash ((\triangleright f \wedge g) \frown \text{skip}) \frown \text{finite} \longrightarrow (f \frown \text{skip}) \frown \text{finite}$
by (*simp add: LeftSChopImpSChop*)
hence 8: $\vdash ((\triangleright f \wedge g) \frown \text{skip}) \frown \text{finite} \longrightarrow \text{di}(f \frown \text{skip})$
by (*metis AndChopA ChopImpDi lift-imp-trans schop-d-def*)
have 9: $\vdash \triangleright f \longrightarrow \neg(\text{di}(f \frown \text{skip}))$
using *FstImpNotDiChopSkipB* **by** *blast*
have 10: $\vdash \triangleright f \wedge \text{di}(\triangleright f \wedge g) \longrightarrow ((\triangleright f \wedge g) \vee \text{di}(f \frown \text{skip}))$
using 5 8 **by** *fastforce*
have 11: $\vdash \triangleright f \wedge \text{di}(\triangleright f \wedge g) \longrightarrow \neg(\text{di}(f \frown \text{skip})) \wedge ((\triangleright f \wedge g) \vee \text{di}(f \frown \text{skip}))$
using 9 10 1 **by** *fastforce*
have 12: $\vdash (\neg(\text{di}(f \frown \text{skip})) \wedge ((\triangleright f \wedge g) \vee \text{di}(f \frown \text{skip}))) = (\neg(\text{di}(f \frown \text{skip})) \wedge ((\triangleright f \wedge g)))$
by *auto*
have 13: $\vdash \triangleright f \wedge \text{di}(\triangleright f \wedge g) \longrightarrow (\triangleright f \wedge g)$
using 11 12 **by** *auto*
have 14: $\vdash (\triangleright f \wedge g) \longrightarrow \triangleright f$
by *auto*
hence 15: $\vdash (\triangleright f \wedge g) \longrightarrow \text{di}(\triangleright f \wedge g)$
using *DiIntro* **by** *auto*
have 16: $\vdash (\triangleright f \wedge g) \longrightarrow \triangleright f \wedge \text{di}(\triangleright f \wedge g)$
using 14 15 **by** *auto*
from 13 16 **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndDiImpBsNotAndDi*:

$\vdash (\triangleright f \wedge \text{di } g) \longrightarrow (\text{bs } (\neg(\text{di } f \wedge g)))$

proof –

have 1: $\vdash (\triangleright f \wedge \text{di } g) \wedge \neg(\text{bs } (\neg(\text{di } f \wedge g))) \longrightarrow \text{ds}(\text{di } f \wedge g)$

by (*auto simp add: ds-d-def*)

hence 2: $\vdash (\triangleright f \wedge \text{di } g) \wedge \neg(\text{bs } (\neg(\text{di } f \wedge g))) \longrightarrow \text{ds}(\text{di } f)$

using *DsAndImp* **by** *fastforce*

hence 3: $\vdash (\triangleright f \wedge \text{di } g) \wedge \neg(\text{bs } (\neg(\text{di } f \wedge g))) \longrightarrow \text{di}(\text{di } f) \frown \text{skip}$

using *DsDi FstImpFinite* **by** *fastforce*

hence 4: $\vdash (\triangleright f \wedge \text{di } g) \wedge \neg(\text{bs } (\neg(\text{di } f \wedge g))) \longrightarrow \text{di } f \frown \text{skip}$

using *DiEqvDiDi* **by** (*metis int-eq*)

hence 5: $\vdash (\triangleright f \wedge \text{di } g) \wedge \neg(\text{bs } (\neg(\text{di } f \wedge g))) \longrightarrow \text{ds } f$

using *DsDi* **by** *fastforce*

hence 6: $\vdash (\triangleright f \wedge \text{di } g) \wedge \neg(\text{bs } (\neg(\text{di } f \wedge g))) \longrightarrow \neg(\triangleright f)$

using *DsImpNotFst* **by** *fastforce*

from 6 **show** *?thesis* **by** *auto*

qed

lemma *FstFstOrEqvFstOrL*:

$\vdash \triangleright(\triangleright f \vee g) = \triangleright(f \vee g)$

proof –

have 1: $\vdash \triangleright(f \vee g) = ((f \vee g) \wedge bs(\neg(f \vee g)))$
by (*simp add: first-d-def*)
have 2: $\vdash (\neg(f \vee g)) = (\neg f \wedge \neg g)$
by *auto*
hence 3: $\vdash bs(\neg(f \vee g)) = bs(\neg f \wedge \neg g)$
using *BsEqvRule* **by** *blast*
have 4: $\vdash bs(\neg f \wedge \neg g) = (bs(\neg f) \wedge bs(\neg g))$
using *BsAndEqv* **by** *fastforce*
hence 5: $\vdash ((f \vee g) \wedge bs(\neg(f \vee g))) = ((f \vee g) \wedge bs(\neg f) \wedge bs(\neg g))$
using 4 3 **by** *fastforce*
have 6: $\vdash ((f \vee g) \wedge bs(\neg f) \wedge bs(\neg g)) =$
 $((f \wedge bs(\neg f)) \vee (g \wedge bs(\neg f))) \wedge bs(\neg g)$
by *auto*
have 7: $\vdash (((f \wedge bs(\neg f)) \vee (g \wedge bs(\neg f))) \wedge bs(\neg g)) =$
 $((\triangleright f \vee (g \wedge bs(\neg f))) \wedge bs(\neg g))$
by (*simp add: first-d-def*)
have 8: $\vdash ((\triangleright f \vee (g \wedge bs(\neg f))) \wedge bs(\neg g)) =$
 $((\triangleright f \vee g) \wedge (\triangleright f \vee bs(\neg f))) \wedge bs(\neg g)$
by *auto*
have 9: $\vdash (((\triangleright f \vee g) \wedge (\triangleright f \vee bs(\neg f))) \wedge bs(\neg g)) =$
 $((\triangleright f \vee g) \wedge ((f \wedge bs(\neg f)) \vee bs(\neg f))) \wedge bs(\neg g)$
by (*simp add: first-d-def*)
have 10: $\vdash (((\triangleright f \vee g) \wedge ((f \wedge bs(\neg f)) \vee bs(\neg f))) \wedge bs(\neg g)) =$
 $((\triangleright f \vee g) \wedge bs(\neg f) \wedge bs(\neg g))$
by *auto*
have 11: $\vdash ((\triangleright f \vee g) \wedge bs(\neg f) \wedge bs(\neg g)) =$
 $((\triangleright f \vee g) \wedge bs(\neg(\triangleright f)) \wedge bs(\neg g))$
using *BsNotFstEqvBsNot* **by** *fastforce*
have 12: $\vdash ((\triangleright f \vee g) \wedge bs(\neg(\triangleright f)) \wedge bs(\neg g)) =$
 $((\triangleright f \vee g) \wedge bs(\neg(\triangleright f) \wedge \neg g))$
using *BsAndEqv* **by** *fastforce*
have 13: $\vdash (\neg(\triangleright f) \wedge \neg g) = (\neg(\triangleright f \vee g))$
by *auto*
hence 14: $\vdash bs(\neg(\triangleright f) \wedge \neg g) = bs(\neg(\triangleright f \vee g))$
using *BsEqvRule* **by** *blast*
hence 15: $\vdash ((\triangleright f \vee g) \wedge bs(\neg(\triangleright f) \wedge \neg g)) = ((\triangleright f \vee g) \wedge bs(\neg(\triangleright f \vee g)))$
by *auto*
have 16: $\vdash ((\triangleright f \vee g) \wedge bs(\neg(\triangleright f \vee g))) = \triangleright(\triangleright f \vee g)$
by (*simp add: first-d-def*)
from 16 15 12 11 10 9 8 7 6 5 1 **show** *?thesis* **by** (*metis int-eq*)
qed

lemma *FstFstOrEqvFstOrR*:

$\vdash \triangleright(f \vee \triangleright g) = \triangleright(f \vee g)$

proof –

have 1: $\vdash (f \vee \triangleright g) = (\triangleright g \vee f)$ **by** *auto*
hence 2: $\vdash \triangleright(f \vee \triangleright g) = \triangleright(\triangleright g \vee f)$ **using** *FstEqvRule* **by** *blast*
have 3: $\vdash \triangleright(\triangleright g \vee f) = \triangleright(g \vee f)$ **using** *FstFstOrEqvFstOrL* **by** *blast*
have 4: $\vdash (g \vee f) = (f \vee g)$ **by** *auto*

hence 5: $\vdash \triangleright(g \vee f) = \triangleright(f \vee g)$ using *FstEqvRule* by *blast*
 from 2 3 5 show *?thesis* by *fastforce*
 qed

lemma *FstFstOrEqvFstOr*:

$\vdash \triangleright(\triangleright f \vee \triangleright g) = \triangleright(f \vee g)$

proof –

have 1: $\vdash \triangleright(\triangleright f \vee \triangleright g) = \triangleright(f \vee \triangleright g)$ using *FstFstOrEqvFstOrL* by *blast*

have 2: $\vdash \triangleright(f \vee \triangleright g) = \triangleright(f \vee g)$ using *FstFstOrEqvFstOrR* by *blast*

from 1 2 show *?thesis* by *fastforce*

qed

lemma *FstLenEqvLen*:

$\vdash \triangleright(\text{len}(k)) = \text{len}(k)$

proof –

have 1: $\vdash \triangleright(\text{len}(k) \wedge \# \text{True}) = (\text{len}(k) \wedge \# \text{True})$ using *FstLenAndEqvLenAnd* by *blast*

have 2: $\vdash (\text{len}(k) \wedge \# \text{True}) = \text{len}(k)$ by *auto*

hence 3: $\vdash \triangleright(\text{len}(k) \wedge \# \text{True}) = \triangleright(\text{len}(k))$ using *FstEqvRule* by *blast*

from 1 2 3 show *?thesis* by *auto*

qed

lemma *FstSkip*:

$\vdash \triangleright \text{skip} = \text{skip}$

proof –

have 1: $\vdash \text{skip} = \text{len}(1)$ using *LenOneEqvSkip* by *fastforce*

hence 2: $\vdash \triangleright \text{skip} = \triangleright(\text{len}(1))$ using *FstEqvRule* by *blast*

have 3: $\vdash \triangleright(\text{len}(1)) = \text{len}(1)$ using *FstLenEqvLen* by *blast*

from 1 2 3 show *?thesis* using *LenOneEqvSkip* by *fastforce*

qed

lemma *HaltStateEqvFstFinState*:

$\vdash (\text{halt}(\text{init } w) \wedge \text{finite}) = \triangleright(\text{fin}(\text{init } w))$

proof –

have 1: $\vdash \text{halt}(\text{init } w) = \Box(\text{empty} = (\text{init } w))$ by (*simp add: halt-d-def*)

have 21: $\vdash (\text{empty} = (\text{init } w)) = (((\text{empty} \longrightarrow (\text{init } w)) \wedge ((\text{init } w) \longrightarrow \text{empty})))$
 by *auto*

hence 2: $\vdash \Box(\text{empty} = (\text{init } w)) = (\Box((\text{empty} \longrightarrow (\text{init } w)) \wedge ((\text{init } w) \longrightarrow \text{empty})))$
 by (*simp add: BoxEqvBox*)

have 3: $\vdash (\Box((\text{empty} \longrightarrow (\text{init } w)) \wedge ((\text{init } w) \longrightarrow \text{empty}))) =$
 $(\Box((\text{empty} \longrightarrow (\text{init } w))) \wedge \Box((\text{init } w) \longrightarrow \text{empty}))$

by (*metis 21 BoxAndBoxEqvBoxRule int-eq*)

have 4: $\vdash ((\text{init } w) \longrightarrow \text{empty}) = (\text{more} \longrightarrow \neg(\text{init } w))$

by (*auto simp add: empty-d-def*)

hence 5: $\vdash \Box((\text{init } w) \longrightarrow \text{empty}) = \Box(\text{more} \longrightarrow \neg(\text{init } w))$ using *BoxEqvBox* by *blast*

have 6: $\vdash (\Box(\text{more} \longrightarrow \neg(\text{init } w)) \wedge \text{finite}) = \text{bs}(\neg(\text{fin}(\text{init } w)))$ using *BoxMoreStateEqvBsFinState* by *blast*

have 7: $\vdash \Box((\text{empty} \longrightarrow (\text{init } w))) = \text{fin}(\text{init } w)$ by (*simp add: fin-d-def*)

have 8: $\vdash (\Box((\text{empty} \longrightarrow (\text{init } w))) \wedge \Box((\text{init } w) \longrightarrow \text{empty}) \wedge \text{finite}) =$
 $(\text{fin}(\text{init } w) \wedge \text{bs}(\neg(\text{fin}(\text{init } w))))$ using 5 6 7 by *fastforce*

from 1 2 3 8 show ?thesis unfolding first-d-def
 by fastforce
 qed

lemma *FstLenEqvLenFst*:

$\vdash \triangleright(\text{len } k ; f) = \text{len } k ; \triangleright f$

proof –

have 1: $\vdash \text{len } k ; f = \triangleright(\text{len } k) ; f$ using *FstLenEqvLen LeftChopEqvChop* by fastforce
 have 2: $\vdash \triangleright(\text{len } k ; f) = \triangleright(\triangleright(\text{len } k) ; f)$ using 1 *FstEqvRule* by blast
 have 3: $\vdash \triangleright(\triangleright(\text{len } k) ; f) = \triangleright(\text{len } k) ; \triangleright f$ using *FstFstChopEqvFstChopFst* by blast
 have 4: $\vdash \triangleright(\text{len } k) ; \triangleright f = \text{len } k ; \triangleright f$ using *FstLenEqvLen LeftChopEqvChop* by fastforce
 from 2 3 4 show ?thesis by fastforce

qed

lemma *FstNextEqvNextFst*:

$\vdash \triangleright(\bigcirc f) = \bigcirc(\triangleright f)$

proof –

have 1: $\vdash \triangleright(\bigcirc f) = \triangleright(\text{skip} ; f)$ using *FstEqvRule* by (simp add: next-d-def)
 have 2: $\vdash \text{skip} ; f = \triangleright \text{skip} ; f$ using *FstSkip* using *LeftChopEqvChop* by fastforce
 have 3: $\vdash \triangleright(\text{skip} ; f) = \triangleright(\triangleright \text{skip} ; f)$ using 2 *FstEqvRule LeftChopEqvChop* by blast
 have 4: $\vdash \triangleright(\triangleright \text{skip} ; f) = \triangleright \text{skip} ; \triangleright f$ using 3 *FstFstChopEqvFstChopFst* by blast
 have 5: $\vdash \triangleright \text{skip} ; \triangleright f = \text{skip} ; \triangleright f$ using 4 *FstSkip LeftChopEqvChop* by blast
 have 6: $\vdash \text{skip} ; \triangleright f = \bigcirc(\triangleright f)$ by (simp add: next-d-def)
 from 1 2 3 4 5 6 show ?thesis by fastforce

qed

lemma *FstDiamondStateEqvHalt*:

$\vdash \triangleright(\Diamond(\text{init } w)) = (\text{halt }(\text{init } w) \wedge \text{finite})$

proof –

have 1: $\vdash \Diamond(\text{init } w) = \Diamond((\text{init } w) \wedge \# \text{True})$ by simp
 have 2: $\vdash \text{fin }(\text{init } w) \frown \# \text{True} = \Diamond((\text{init } w) \wedge \# \text{True})$ using 1 *FinChopEqvDiamond*
 by (metis schop-d-def)
 have 3: $\vdash \text{fin }(\text{init } w) \frown \# \text{True} = \text{df }(\text{fin }(\text{init } w))$ unfolding *df-d-def* by simp
 have 4: $\vdash \Diamond(\text{init } w) = (\text{df }(\text{fin }(\text{init } w)))$ using 1 2 3 by fastforce
 have 5: $\vdash \triangleright(\Diamond(\text{init } w)) = \triangleright(\text{df }(\text{fin }(\text{init } w)))$ using 4 *FstEqvRule* by blast
 hence 6: $\vdash \triangleright(\Diamond(\text{init } w)) = \triangleright(\text{fin }(\text{init } w))$ using *FstDfEqvFst* by fastforce
 hence 7: $\vdash \triangleright(\Diamond(\text{init } w)) = (\text{halt }(\text{init } w) \wedge \text{finite})$ using *HaltStateEqvFstFinState* by fastforce
 from 7 show ?thesis by simp

qed

lemma *DiAndFiniteEqvDiFst*:

$\vdash \text{di } (f \wedge \text{finite}) = \text{di}(\triangleright f)$

proof –

have 1: $\vdash (\triangleright f \wedge \text{finite}) = \triangleright f$
 by (meson *FstImpFinite Prop10 Prop11*)
 have 2: $\vdash (f \wedge \text{finite}); \# \text{True} = (\triangleright f \wedge \text{finite}); \# \text{True}$
 using *DfEqvDfFst FstImpFinite* unfolding *df-d-def schop-d-def* by auto
 show ?thesis
 by (metis 1 2 *di-d-def inteq-reflection*)

qed

lemma *FstEqvFstAndFinite*:

$\vdash \triangleright f = \triangleright (f \wedge \text{finite})$

by (*metis FstDfEqvFst FstDiEqvFst di-d-def inteq-reflection itl-def(15) schop-d-def*)

lemma *FstDiAndDiEqvFstDfAndDf*:

$\vdash \triangleright (di\ f \wedge di\ g) = \triangleright (df\ f \wedge df\ g)$

proof –

have 1: $\vdash \triangleright (di\ f \wedge di\ g) = \triangleright ((di\ f \wedge di\ g) \wedge \text{finite})$

by (*simp add: FstEqvFstAndFinite*)

have 2: $\vdash ((di\ f \wedge di\ g) \wedge \text{finite}) = ((df\ f \wedge df\ g) \wedge \text{finite})$

using *DiAndFiniteEqvDfAndFinite[of f] DiAndFiniteEqvDfAndFinite[of g]*

by *fastforce*

have 3: $\vdash \triangleright ((di\ f \wedge di\ g) \wedge \text{finite}) = \triangleright ((df\ f \wedge df\ g) \wedge \text{finite})$

by (*simp add: 2 FstEqvRule*)

have 4: $\vdash \triangleright ((df\ f \wedge df\ g) \wedge \text{finite}) = \triangleright (df\ f \wedge df\ g)$

by (*meson FstEqvFstAndFinite Prop11*)

show *?thesis*

by (*metis 1 3 4 inteq-reflection*)

qed

lemma *FstBoxStateEqvStateAndEmpty*:

$\vdash \triangleright (\Box (init\ w)) = ((init\ w) \wedge \text{empty})$

proof –

have 1: $\vdash ((init\ w) \wedge (\Box (init\ w))^*) = \Box (init\ w)$

using *BoxCSEqvBox* **by** *blast*

have 2: $\vdash \Box (init\ w) = ((init\ w) \wedge (\Box (init\ w))^*)$

using 1 **by** *auto*

hence 3: $\vdash \Box (init\ w) = ((init\ w) \wedge (\Box (init\ w))^*)$

by *blast*

have 4: $\vdash ((init\ w) \wedge \text{empty}) ; (\Box (init\ w))^* = ((init\ w) \wedge (\Box (init\ w))^*)$

using *StateAndEmptyChop* **by** *blast*

have 5: $\vdash ((init\ w) \wedge (\Box (init\ w))^*) = ((init\ w) \wedge \text{empty}) ; (\Box (init\ w))^*$

using 4 **by** *fastforce*

have 6: $\vdash \Box (init\ w) = ((init\ w) \wedge \text{empty}) ; (\Box (init\ w))^*$

using 3 5 **by** *fastforce*

have 7: $\vdash ((init\ w) \wedge \text{empty}) ; (\Box (init\ w))^* = \triangleright (init\ w) ; (\Box (init\ w))^*$

using *FstState* **by** (*metis AndChopCommute int-eq*)

have 8: $\vdash \Box (init\ w) = \triangleright (init\ w) ; (\Box (init\ w))^*$

using 6 7 **by** *fastforce*

have 9: $\vdash \triangleright (\Box (init\ w)) = \triangleright (\triangleright (init\ w) ; (\Box (init\ w))^*)$

using 8 *FstEqvRule* **by** *blast*

have 10: $\vdash \triangleright (\triangleright (init\ w) ; (\Box (init\ w))^*) = \triangleright (init\ w) ; \triangleright ((\Box (init\ w))^*)$

using *FstFstChopEqvFstChopFst* **by** *blast*

have 11: $\vdash \triangleright (init\ w) ; \triangleright ((\Box (init\ w))^*) = \triangleright (init\ w) ; \text{empty}$

using *RightChopEqvChop FstCSEqvEmpty* **by** *blast*

have 12: $\vdash \triangleright (\text{init } w) ; \text{empty} = \triangleright (\text{init } w)$
using *RightChopEqvChop ChopEmpty* **by** *blast*
have 13: $\vdash \triangleright (\text{init } w) = ((\text{init } w) \wedge \text{empty})$
using *FstState* **by** *fastforce*
from 9 10 11 12 13 **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndFstStarEqvFst*:

$\vdash (\text{schopstar } \triangleright f) \wedge \triangleright f = \triangleright f$

proof –

have 1: $\vdash (\text{schopstar } \triangleright f) = (\text{empty} \vee (\triangleright f) \neg (\text{schopstar } \triangleright f))$
by (*meson Prop11 SChopstar-unfoldl-eq*)
have 2: $\vdash ((\text{schopstar } \triangleright f) \wedge \triangleright f) = ((\text{empty} \vee (\triangleright f) \neg (\text{schopstar } \triangleright f)) \wedge \triangleright f)$
using 1 **by** *fastforce*
have 3: $\vdash ((\text{empty} \vee (\triangleright f) \neg (\text{schopstar } \triangleright f)) \wedge \triangleright f) =$
 $((\text{empty} \wedge \triangleright f) \vee ((\triangleright f) \neg (\text{schopstar } \triangleright f)) \wedge \triangleright f)$
by *auto*
have 4: $\vdash ((\text{schopstar } \triangleright f) \wedge \triangleright f) = ((\text{empty} \wedge \triangleright f) \vee ((\triangleright f) \neg (\text{schopstar } \triangleright f)) \wedge \triangleright f)$
using 2 3 **by** *fastforce*
have 5: $\vdash ((\triangleright f) \neg (\text{schopstar } \triangleright f)) \wedge \triangleright f = ((\triangleright f) \neg (\text{schopstar } \triangleright f)) \wedge \triangleright f \neg \text{empty}$
by (*metis ChopEmpty FstImpFinite Prop10 inteq-reflection schop-d-def*)
have 6: $\vdash ((\triangleright f) \neg (\text{schopstar } \triangleright f)) \wedge \triangleright f \neg \text{empty} = (\triangleright f) \neg ((\text{schopstar } \triangleright f) \wedge \text{empty})$
by (*simp add: LFstAndDistrB schop-d-def*)
have 7: $\vdash (\text{schopstar } \triangleright f) \wedge \text{empty} = \text{empty}$
using *SChopstar-imp-empty* **by** *fastforce*

have 8: $\vdash (\triangleright f) \neg (\text{schopstar } \triangleright f) \wedge \text{empty} = \triangleright f$
using 7 *ChopEmpty*
by (*metis FstImpFinite Prop11 Prop12 inteq-reflection schop-d-def*)
have 9: $\vdash ((\triangleright f) \neg (\text{schopstar } \triangleright f)) \wedge \triangleright f = \triangleright f$
using 5 6 8 **by** *fastforce*
have 10: $\vdash (\text{schopstar } \triangleright f) \wedge \triangleright f = ((\text{empty} \wedge \triangleright f) \vee \triangleright f)$
using 4 9 **by** *fastforce*
have 11: $\vdash ((\text{empty} \wedge \triangleright f) \vee \triangleright f) = \triangleright f$
by *auto*
have 12: $\vdash (\text{schopstar } \triangleright f) \wedge \triangleright f = \triangleright f$
using 10 11 **by** *fastforce*
from 12 **show** *?thesis* **by** *auto*
qed

lemma *HaltStateEqvFstHaltState*:

$\vdash (\text{halt}(\text{init}(w)) \wedge \text{finite}) = \triangleright (\text{halt}(\text{init}(w)))$

proof –

have 1: $\vdash (\text{halt}(\text{init } w) \wedge \text{finite}) = \triangleright (\text{fin}(\text{init } w))$
by (*simp add: HaltStateEqvFstFinState*)
have 2: $\vdash \triangleright (\text{fin}(\text{init } w)) = \triangleright (\triangleright (\text{fin}(\text{init } w)))$
using *FstEqvRule FstFixFst* **by** *fastforce*
have 3: $\vdash \triangleright (\triangleright (\text{fin}(\text{init } w))) = \triangleright (\text{halt}(\text{init}(w)) \wedge \text{finite})$
using *FstEqvRule HaltStateEqvFstFinState* **by** (*metis inteq-reflection*)
have 4: $\vdash \triangleright (\text{halt}(\text{init}(w)) \wedge \text{finite}) = \triangleright (\text{halt}(\text{init}(w)))$

by (metis FstDfEqvFst FstDiEqvFst Prop04 di-d-def itl-def(15) schop-d-def)
 from 1 2 3 4 show ?thesis by fastforce
 qed

lemma *DiHaltAndDiHaltAndEqvDiHaltAndAnd*:

$\vdash (df(halt (init w) \wedge f) \wedge df(halt (init w) \wedge g)) = df(halt (init w) \wedge f \wedge g)$

proof –

have 01: $\vdash (halt (init w) \wedge f \wedge finite) = (\triangleright(fin (init w)) \wedge f \wedge finite)$
 using *HaltStateEqvFstFinState* by fastforce
 have 02: $\vdash (halt (init w) \wedge g \wedge finite) = (\triangleright(fin (init w)) \wedge g \wedge finite)$
 using *HaltStateEqvFstFinState* by fastforce
 have 03: $\vdash (halt (init w) \wedge f \wedge finite) \frown \#True = (\triangleright(fin (init w)) \wedge f \wedge finite) \frown \#True$
 by (simp add: 01 LeftSCHopEqvSCHop)
 have 04: $\vdash (halt (init w) \wedge g \wedge finite) \frown \#True = (\triangleright(fin (init w)) \wedge g \wedge finite) \frown \#True$
 by (simp add: 02 LeftSCHopEqvSCHop)
 have 05: $\vdash (\triangleright(fin (init w)) \wedge f \wedge finite) \frown \#True = (\triangleright(fin (init w)) \wedge f) \frown \#True$
 by (metis (no-types, lifting) FstFstAndEqvFstAnd FstImpFinite LeftSCHopEqvSCHop Prop11 Prop12
 inteq-reflection)
 have 06: $\vdash (\triangleright(fin (init w)) \wedge g \wedge finite) \frown \#True = (\triangleright(fin (init w)) \wedge g) \frown \#True$
 by (metis (no-types, lifting) FstFstAndEqvFstAnd FstImpFinite LeftSCHopEqvSCHop Prop11 Prop12
 inteq-reflection)
 have 07: $\vdash (halt (init w) \wedge f \wedge finite) \frown \#True = (halt (init w) \wedge f) \frown \#True$
 unfolding *schop-d-def*
 by (metis (no-types, lifting) LeftChopEqvChop Prop11 Prop12 lift-and-com)
 have 08: $\vdash (halt (init w) \wedge g \wedge finite) \frown \#True = (halt (init w) \wedge g) \frown \#True$
 unfolding *schop-d-def*
 by (metis (no-types, lifting) LeftChopEqvChop Prop11 Prop12 lift-and-com)
 have 1: $\vdash (df(halt (init w) \wedge f) \wedge df(halt (init w) \wedge g)) =$
 $(df(\triangleright(fin (init w)) \wedge f) \wedge df(\triangleright(fin (init w)) \wedge g))$
 unfolding *df-d-def*
 by (metis 01 02 05 06 07 08 inteq-reflection lift-and-com)
 have 2: $\vdash (df(\triangleright(fin (init w)) \wedge f) \wedge df(\triangleright(fin (init w)) \wedge g)) =$
 $df(\triangleright(fin (init w)) \wedge f \wedge g)$
 using *LFstAndDistrD*
 by (metis (no-types, lifting) FstFstAndEqvFstAnd FstImpFinite Prop10 df-d-def di-d-def inteq-reflection
 schop-d-def)
 have 08: $\vdash (\triangleright(fin (init w)) \wedge f \wedge g) = (halt (init w) \wedge f \wedge g \wedge finite)$
 using *HaltStateEqvFstFinState* by fastforce
 have 09: $\vdash (\triangleright(fin (init w)) \wedge f \wedge g) \frown \#True = (halt (init w) \wedge f \wedge g \wedge finite) \frown \#True$
 by (simp add: 08 LeftSCHopEqvSCHop)
 have 010: $\vdash (halt (init w) \wedge f \wedge g \wedge finite) \frown \#True = (halt (init w) \wedge f \wedge g) \frown \#True$
 unfolding *schop-d-def*
 by (metis (no-types, lifting) LeftChopEqvChop Prop11 Prop12 lift-and-com)
 have 3: $\vdash df(\triangleright(fin (init w)) \wedge f \wedge g) = df(halt (init w) \wedge f \wedge g)$
 unfolding *df-d-def*
 by (metis 010 08 inteq-reflection)
 from 1 2 3 show ?thesis using *int-eq* by metis
 qed

lemma counter-ex-lhs:

$\vdash ((\triangleright(\text{len}(5)) \wedge \triangleright(\text{len}(2))) ; (\text{len}(5) \vee \text{len}(2))) = \#False$

proof –

have 1: $\vdash ((\triangleright(\text{len}(5)) \wedge \triangleright(\text{len}(2))) ; (\text{len}(5) \vee \text{len}(2))) =$
 $(\text{len}(5) \wedge \text{len}(2)); (\text{len}(5) \vee \text{len}(2))$

by (*metis FstLenAndEqvLenAnd FstLenEqvLen LeftChopEqvChop inteq-reflection*)

have 2: $\vdash (\text{len}(5) \wedge \text{len}(2)) = \#False$

by (*simp add: Valid-def len-defs*)

have 3: $\vdash ((\text{len}(5) \wedge \text{len}(2)); (\text{len}(5) \vee \text{len}(2))) = (\#False; (\text{len}(5) \vee \text{len}(2)))$

by (*simp add: 2 LeftChopEqvChop*)

have 4: $\vdash (\#False; (\text{len}(5) \vee \text{len}(2))) = \#False$

by (*simp add: Valid-def chop-defs*)

from 1 3 4 show ?thesis by fastforce

qed

lemma counter-ex-rhs:

$\vdash ((\triangleright(\text{len}(5)) ; (\text{len}(5) \vee \text{len}(2))) \wedge (\triangleright(\text{len}(2)) ; (\text{len}(5) \vee \text{len}(2)))) = \text{len}(7)$

proof –

have 1: $\vdash (\triangleright(\text{len}(5)) ; (\text{len}(5) \vee \text{len}(2))) =$
 $\text{len}(5); (\text{len}(5) \vee \text{len}(2))$

using *FstLenEqvLen LeftChopEqvChop* **by** *blast*

have 2: $\vdash (\triangleright(\text{len}(2)) ; (\text{len}(5) \vee \text{len}(2))) =$
 $\text{len}(2); (\text{len}(5) \vee \text{len}(2))$

using *FstLenEqvLen LeftChopEqvChop* **by** *blast*

have 3: $\vdash \text{len}(5); (\text{len}(5) \vee \text{len}(2)) =$
 $((\text{len}(5); \text{len}(5)) \vee (\text{len}(5); \text{len}(2)))$

by (*simp add: ChopOrEqv*)

have 4: $\vdash ((\text{len}(5); \text{len}(5)) \vee (\text{len}(5); \text{len}(2))) =$
 $(\text{len}(10) \vee \text{len}(7))$

using *LenEqvLenChopLen inteq-reflection* **by** *fastforce*

have 5: $\vdash \text{len}(2); (\text{len}(5) \vee \text{len}(2)) =$
 $((\text{len}(2); \text{len}(5)) \vee (\text{len}(2); \text{len}(2)))$

by (*simp add: ChopOrEqv*)

have 6: $\vdash ((\text{len}(2); \text{len}(5)) \vee (\text{len}(2); \text{len}(2))) =$
 $(\text{len}(7) \vee \text{len}(4))$

using *LenEqvLenChopLen inteq-reflection* **by** *fastforce*

have 7: $\vdash ((\text{len}(10) \vee \text{len}(7)) \wedge (\text{len}(7) \vee \text{len}(4))) =$
 $((\text{len}(7) \vee \text{len}(10)) \wedge (\text{len}(7) \vee \text{len}(4)))$

by *fastforce*

have 8: $\vdash ((\text{len}(7) \vee \text{len}(10)) \wedge (\text{len}(7) \vee \text{len}(4))) =$
 $(\text{len}(7) \vee (\text{len}(10) \wedge \text{len}(4)))$

by *fastforce*

have 9: $\vdash (\text{len}(10) \wedge \text{len}(4)) = \#False$

by (*simp add: Valid-def len-defs*)

have 10: $\vdash (\text{len}(7) \vee (\text{len}(10) \wedge \text{len}(4))) = \text{len}(7)$

using 9 by auto

have 11: $\vdash ((\triangleright(\text{len}(5)) ; (\text{len}(5) \vee \text{len}(2))) \wedge (\triangleright(\text{len}(2)) ; (\text{len}(5) \vee \text{len}(2)))) =$

```

      (len(5);(len(5) ∨ len(2)) ∧ len(2) ;(len(5) ∨ len(2)))
    using 1 2 by fastforce
  have 12: ⊢ (len(5);(len(5) ∨ len(2)) ∧ len(2) ;(len(5) ∨ len(2))) = len(7)
    by (metis 10 4 6 7 8 ChopOrEqv inteq-reflection)
  from 11 12 show ?thesis by fastforce
qed

```

end

2.13 Monitors

```

theory Monitor
imports First

```

begin

The RV monitors language is introduced plus the algebraic properties of the monitor operators.

2.13.1 Syntax

```

datatype ('a::world) monitor =
  mFIRST-d 'a formula                ((FIRST -) [84] 83)
| mUPTO-d 'a monitor 'a monitor      ((- UPTO -) [84,84] 83)
| mTHRU-d 'a monitor 'a monitor      ((- THRU -) [84,84] 83)
| mTHEN-d 'a monitor 'a monitor      ((- THEN -) [84,84] 83)
| mWITH-d 'a monitor 'a formula      ((- WITH -) [84,84] 83)

fun MON :: ('a::world) monitor ⇒ 'a formula
where (MON (FIRST f)) = LIFT(▷ f)
      | (MON (a UPTO b)) = LIFT(▷((MON a) ∨ (MON b)))
      | (MON (a THRU b)) = LIFT(▷(di(MON a) ∧ di(MON b)))
      | (MON (a THEN b)) = LIFT((MON a)⌢(MON b))
      | (MON (a WITH f)) = LIFT((MON a) ∧ f)

syntax
  -MON :: 'a monitor ⇒ lift ((M -) [80] 80)

translations
  -MON == CONST MON

```

```

definition eq-d :: ('a:: world) monitor ⇒ 'a monitor ⇒ bool ((- ≃ -) [84,84] 83)
where
  eq-d a b ≡ (⊢ (M a) = (M b))

```

```

lemma MonEqRefl:
  a ≃ a
by (simp add: eq-d-def)

```

lemma *MonEqSym*:

assumes $a \simeq b$

shows $b \simeq a$

using *assms* **by** (*metis eq-d-def inteq-reflection*)

lemma *MonEqTrans*:

assumes $a \simeq b$

$b \simeq c$

shows $a \simeq c$

using *assms*(1) *assms*(2) **by** (*metis eq-d-def inteq-reflection*)

lemma *MonEq*:

$(a \simeq b) = (\vdash (\mathcal{M} \ a) = (\mathcal{M} \ b))$

by (*simp add: eq-d-def*)

lemma *MonEqSubstWith*:

assumes $a \simeq b$

shows $(a \text{ WITH } f) \simeq (b \text{ WITH } f)$

using *assms* **by** (*metis MON.simps*(5) *eq-d-def inteq-reflection lift-and-com*)

lemma *MonEqSubstThen*:

assumes $a1 \simeq b1$

$a2 \simeq b2$

shows $(a1 \text{ THEN } a2) \simeq (b1 \text{ THEN } b2)$

using *assms* **by** (*simp add: SChopEqvSChop eq-d-def*)

lemma *MonEqSubstUpto*:

assumes $a1 \simeq b1$

$a2 \simeq b2$

shows $(a1 \text{ UPTO } a2) \simeq (b1 \text{ UPTO } b2)$

using *assms* **by** (*metis (mono-tags, lifting) MON.simps*(2) *eq-d-def int-eq MonEqRefl*)

lemma *MonEqSubstThru*:

assumes $a1 \simeq b1$

$a2 \simeq b2$

shows $(a1 \text{ THRU } a2) \simeq (b1 \text{ THRU } b2)$

using *assms* **by** (*metis (mono-tags, lifting) MON.simps*(3) *eq-d-def int-eq MonEqRefl*)

2.13.2 Derived Monitors

definition *HALT-d* :: $('a :: \text{world}) \text{ formula} \Rightarrow 'a \text{ monitor}$

where *HALT-d* $w \equiv \text{FIRST}(\text{LIFT}(\text{fin}(\text{init } w)))$

definition *LEN-d* :: $\text{nat} \Rightarrow ('a :: \text{world}) \text{ monitor}$

where

LEN-d $k \equiv \text{FIRST}(\text{LIFT}(\text{len } k))$

definition *EMPTY-d* :: $('a :: \text{world}) \text{ monitor}$

where

$EMPTY-d \equiv FIRST (LIFT(empty))$

definition $SKIP-d :: ('a :: world) \text{ monitor}$

where

$SKIP-d \equiv FIRST (LIFT (skip))$

syntax

$-HALT-d :: lift \Rightarrow 'a \text{ monitor} \quad ((HALT -) [84] 83)$
 $-LEN-d :: nat \Rightarrow 'a \text{ monitor} \quad ((LEN -) [84] 83)$
 $-EMPTY-d :: 'a \text{ monitor} \quad ((EMPTY))$
 $-SKIP-d :: 'a \text{ monitor} \quad ((SKIP))$

syntax (ASCII)

$-HALT-d :: lift \Rightarrow 'a \text{ monitor} \quad ((HALT -) [84] 83)$
 $-LEN-d :: nat \Rightarrow 'a \text{ monitor} \quad ((LEN -) [84] 83)$
 $-EMPTY-d :: 'a \text{ monitor} \quad ((EMPTY))$
 $-SKIP-d :: 'a \text{ monitor} \quad ((SKIP))$

translations

$-HALT-d \Rightarrow CONST HALT-d$
 $-LEN-d \Rightarrow CONST LEN-d$
 $-EMPTY-d \Rightarrow CONST EMPTY-d$
 $-SKIP-d \Rightarrow CONST SKIP-d$

definition $GUARD-d :: ('a :: world) \text{ formula} \Rightarrow 'a \text{ monitor}$

where

$GUARD-d w \equiv (EMPTY WITH LIFT(init w))$

primrec $TIMES-d :: ('a :: world) \text{ monitor} \Rightarrow nat \Rightarrow 'a \text{ monitor}$

where

$TIMES-0 : TIMES-d a 0 = EMPTY$
 $| TIMES-Suc: TIMES-d a (Suc k) = (a THEN (TIMES-d a k))$

syntax

$-GUARD-d :: lift \Rightarrow 'a \text{ monitor} \quad ((GUARD -) [84] 83)$
 $-TIMES-d :: ['a \text{ monitor}, nat] \Rightarrow 'a \text{ monitor} \quad ((- TIMES -) [84, 84] 83)$

syntax (ASCII)

$-GUARD-d :: lift \Rightarrow 'a \text{ monitor} \quad ((GUARD -) [84] 83)$
 $-TIMES-d :: ['a \text{ monitor}, nat] \Rightarrow 'a \text{ monitor} \quad ((- TIMES -) [84, 84] 83)$

translations

$-GUARD-d \Rightarrow CONST GUARD-d$
 $-TIMES-d \Rightarrow CONST TIMES-d$

definition $FAIL-d :: ('a :: world) \text{ monitor}$

where

$FAIL-d \equiv GUARD (\#False)$

definition $ALWAYS-d :: ('a :: world) \text{ monitor} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ monitor}$
where

$ALWAYS-d \ a \ w \equiv (a \text{ WITH LIFT}((bi \ (finite \longrightarrow fin \ (init \ w))))))$

definition $SOMETIME-d :: ('a :: world) \text{ monitor} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ monitor}$
where

$SOMETIME-d \ a \ w \equiv (a \text{ WITH LIFT}((di \ (finite \wedge fin \ (init \ w)))))$

definition $LIMIT-d :: ('a :: world) \text{ formula} \Rightarrow 'a \text{ formula}$
where

$LIMIT-d \ f \equiv LIFT(bs \ (\neg f))$

definition $UNTIL-d :: ('a :: world) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ monitor}$
where

$UNTIL-d \ w1 \ w2 \equiv (HALT \ w2) \text{ WITH } (LIFT(bm \ w1))$

syntax

- $FAIL-d \quad :: 'a \text{ monitor} \quad (FAIL)$
- $ALWAYS-d \quad :: ['a \text{ monitor}, lift] \Rightarrow 'a \text{ monitor} \ ((- \ ALWAYS \ -) \ [84,84] \ 83)$
- $SOMETIME-d \quad :: ['a \text{ monitor}, lift] \Rightarrow 'a \text{ monitor} \ ((- \ SOMETIME \ -) \ [84,84] \ 83)$
- $LIMIT-d \quad :: lift \Rightarrow lift \quad ((Limit \ -) \ [84] \ 83)$
- $UNTIL-d \quad :: [lift, lift] \Rightarrow 'a \text{ monitor} \quad ((- \ UNTIL \ -) \ [84,84] \ 83)$

syntax (ASCII)

- $FAIL-d \quad :: 'a \text{ monitor} \quad (FAIL)$
- $ALWAYS-d \quad :: ['a \text{ monitor}, lift] \Rightarrow 'a \text{ monitor} \ ((- \ ALWAYS \ -) \ [84,84] \ 83)$
- $SOMETIME-d \quad :: ['a \text{ monitor}, lift] \Rightarrow 'a \text{ monitor} \ ((- \ SOMETIME \ -) \ [84,84] \ 83)$
- $LIMIT-d \quad :: lift \Rightarrow lift \quad ((Limit \ -) \ [84] \ 83)$
- $UNTIL-d \quad :: [lift, lift] \Rightarrow 'a \text{ monitor} \quad ((- \ UNTIL \ -) \ [84,84] \ 83)$

translations

- $FAIL-d \quad \Rightarrow CONST \ FAIL-d$
- $ALWAYS-d \quad \Rightarrow CONST \ ALWAYS-d$
- $SOMETIME-d \quad \Rightarrow CONST \ SOMETIME-d$
- $LIMIT-d \quad \Rightarrow CONST \ LIMIT-d$
- $UNTIL-d \quad \Rightarrow CONST \ UNTIL-d$

definition $WITHIN-d :: ('a :: world) \text{ monitor} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ monitor}$
where

$WITHIN-d \ a \ f \equiv (a \text{ WITH LIFT}(Limit \ f))$

syntax

- $WITHIN-d \quad :: ['a \text{ monitor}, lift] \Rightarrow 'a \text{ monitor} \ ((- \ WITHIN \ -) \ [84,84] \ 83)$

syntax (ASCII)

- $WITHIN-d \quad :: ['a \text{ monitor}, lift] \Rightarrow 'a \text{ monitor} \ ((- \ WITHIN \ -) \ [84,84] \ 83)$

translations

$-WITHIN-d \Rightarrow CONST WITHIN-d$

definition $AND-d :: ('a :: world) \text{ monitor} \Rightarrow 'a \text{ monitor} \Rightarrow 'a \text{ monitor}$

where

$AND-d \ a \ b \equiv (a \text{ WITH LIFT}(\mathcal{M} \ b))$

definition $ITERATE-d :: ('a :: world) \text{ monitor} \Rightarrow 'a \text{ monitor} \Rightarrow 'a \text{ monitor}$

where

$ITERATE-d \ a \ b \equiv (a \text{ WITH (LIFT (schopstar } (\mathcal{M} \ b))))$

syntax

$-AND-d \quad :: ['a \text{ monitor}, 'a \text{ monitor}] \Rightarrow 'a \text{ monitor} \ ((- \text{ AND } -) [84,84] \ 83)$

$-ITERATE-d \quad :: ['a \text{ monitor}, 'a \text{ monitor}] \Rightarrow 'a \text{ monitor} \ ((- \text{ ITERATE } -) [84,84] \ 83)$

syntax (ASCII)

$-AND-d \quad :: ['a \text{ monitor}, 'a \text{ monitor}] \Rightarrow 'a \text{ monitor} \ ((- \text{ AND } -) [84,84] \ 83)$

$-ITERATE-d \quad :: ['a \text{ monitor}, 'a \text{ monitor}] \Rightarrow 'a \text{ monitor} \ ((- \text{ ITERATE } -) [84,84] \ 83)$

translations

$-AND-d \quad \Rightarrow CONST AND-d$

$-ITERATE-d \Rightarrow CONST ITERATE-d$

definition $STAR-d :: ('a :: world) \text{ monitor} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ monitor}$

where

$STAR-d \ a \ f \equiv ((FIRST \ LIFT(\Diamond \ f)) \text{ ITERATE } (a))$

definition $REPEAT-d :: ('a :: world) \text{ monitor} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ monitor}$

where

$REPEAT-d \ a \ w \equiv ((HALT \ w) \text{ ITERATE } (a \text{ WITH LIFT}(\text{keep}(\neg \ (init \ w)))))$

syntax

$-STAR-d \quad :: ['a \text{ monitor}, \text{lift}] \Rightarrow 'a \text{ monitor} \ ((- \text{ STAR } -) [84,84] \ 83)$

$-REPEAT-d \quad :: ['a \text{ monitor}, \text{lift}] \Rightarrow 'a \text{ monitor} \ ((- \text{ REPEATUNTIL } -) [84,84] \ 83)$

syntax (ASCII)

$-STAR-d \quad :: ['a \text{ monitor}, \text{lift}] \Rightarrow 'a \text{ monitor} \ ((- \text{ STAR } -) [84,84] \ 83)$

$-REPEAT-d \quad :: ['a \text{ monitor}, \text{lift}] \Rightarrow 'a \text{ monitor} \ ((- \text{ REPEATUNTIL } -) [84,84] \ 83)$

translations

$-STAR-d \quad \Rightarrow CONST STAR-d$

$-REPEAT-d \Rightarrow CONST REPEAT-d$

2.13.3 Monitor Laws

lemma $MFixFst$:

$\vdash (\mathcal{M} \ a) = \triangleright (\mathcal{M} \ a)$

```

proof
  (induct a )
  case (mFIRST-d x)
  then show ?case
  proof -
    have 1:  $\vdash (\mathcal{M} (FIRST\ x)) = \triangleright x$  by simp
    have 2:  $\vdash \triangleright x = \triangleright (\triangleright x)$  using FstFixFst by fastforce
    have 3:  $\vdash \triangleright (\triangleright x) = \triangleright (\mathcal{M} (FIRST\ x))$  by simp
    from 1 2 3 show ?thesis by fastforce
  qed
next
  case (mUPTO-d a1 a2)
  then show ?case
  proof -
    have 1:  $\vdash (\mathcal{M} (a1\ UPTO\ a2)) = \triangleright (\mathcal{M}\ a1 \vee (\mathcal{M}\ a2))$ 
      by (simp)
    have 2:  $\vdash \triangleright (\mathcal{M}\ a1 \vee (\mathcal{M}\ a2)) = \triangleright (\triangleright (\mathcal{M}\ a1 \vee (\mathcal{M}\ a2)))$ 
      using FstFixFst by fastforce
    have 3:  $\vdash \triangleright (\triangleright (\mathcal{M}\ a1 \vee (\mathcal{M}\ a2))) = \triangleright (\mathcal{M} (a1\ UPTO\ a2))$ 
      using 2 by simp
    from 1 2 3 show ?thesis by fastforce
  qed
next
  case (mTHRU-d a1 a2)
  then show ?case
  proof -
    have 1:  $\vdash (\mathcal{M} (a1\ THRU\ a2)) = \triangleright (di\ (\mathcal{M}\ a1) \wedge di(\mathcal{M}\ a2))$ 
      by (simp)
    have 2:  $\vdash \triangleright (di\ (\mathcal{M}\ a1) \wedge di(\mathcal{M}\ a2)) = \triangleright (\triangleright (di(\mathcal{M}\ a1) \wedge di(\mathcal{M}\ a2)))$ 
      using FstFixFst by fastforce
    have 3:  $\vdash \triangleright (\triangleright (di\ (\mathcal{M}\ a1) \wedge di(\mathcal{M}\ a2))) = \triangleright (\mathcal{M} (a1\ THRU\ a2))$ 
      using 2 by simp
    from 1 2 3 show ?thesis by fastforce
  qed
next
  case (mTHEN-d a1 a2)
  then show ?case
  proof -
    have 1:  $\vdash (\mathcal{M} (a1\ THEN\ a2)) = (\mathcal{M}\ a1) \frown (\mathcal{M}\ a2)$ 
      by (simp)
    have 2:  $\vdash (\mathcal{M}\ a1) \frown (\mathcal{M}\ a2) = \triangleright (\mathcal{M}\ a1) \frown \triangleright (\mathcal{M}\ a2)$ 
      using SChopEqvSChop mTHEN-d.hyps(1) mTHEN-d.hyps(2) by blast
    have 3:  $\vdash \triangleright (\mathcal{M}\ a1) \frown \triangleright (\mathcal{M}\ a2) = \triangleright (\triangleright (\mathcal{M}\ a1) \frown (\mathcal{M}\ a2))$ 
      using FstFstChopEqvFstChopFst
      by (metis FstImpFinite Prop10 inteq-reflection schop-d-def)
    have 4:  $\vdash \triangleright (\triangleright (\mathcal{M}\ a1) \frown (\mathcal{M}\ a2)) = \triangleright ((\mathcal{M}\ a1) \frown (\mathcal{M}\ a2))$ 
      using FstEqvRule LeftSChopEqvSChop mTHEN-d.hyps(1) by (metis inteq-reflection)
    have 5:  $\vdash \triangleright ((\mathcal{M}\ a1) \frown (\mathcal{M}\ a2)) = \triangleright (\mathcal{M} (a1\ THEN\ a2))$ 
      using 4 by simp
    from 1 2 3 4 5 show ?thesis by fastforce
  
```

```

qed
next
case (mWITH-d a x2)
then show ?case
proof -
  have 1:  $\vdash (\mathcal{M} (a \text{ WITH } x2)) = ((\mathcal{M} a) \wedge (x2))$ 
    by (simp)
  have 2:  $\vdash ((\mathcal{M} a) \wedge (x2)) = (\triangleright(\mathcal{M} a) \wedge (x2))$ 
    using mWITH-d.hyps by fastforce
  have 3:  $\vdash (\triangleright(\mathcal{M} a) \wedge (x2)) = \triangleright(\triangleright(\mathcal{M} a) \wedge (x2))$ 
    using FstFstAndEqvFstAnd by fastforce
  have 4:  $\vdash \triangleright(\triangleright(\mathcal{M} a) \wedge (x2)) = \triangleright((\mathcal{M} a) \wedge (x2))$ 
    using 2 FstEqvRule by fastforce
  have 5:  $\vdash \triangleright((\mathcal{M} a) \wedge (x2)) = \triangleright(\mathcal{M} (a \text{ WITH } x2))$ 
    using 4 by simp
  from 1 2 3 4 5 show ?thesis by (metis inteq-reflection)
qed
qed

```

lemma *MGuardFalseEqvFalse:*

$\vdash \mathcal{M}(\text{GUARD } \#False) = \#False$

proof -

```

have 1:  $\vdash \mathcal{M}(\text{GUARD } \#False) = \mathcal{M}(\text{EMPTY WITH LIFT}(\text{init } \#False))$  by (simp add: GUARD-d-def)
have 2:  $\vdash \mathcal{M}(\text{EMPTY WITH LIFT}(\text{init } \#False)) = (\mathcal{M}(\text{EMPTY}) \wedge (\text{init } \#False))$  by (simp)
have 3:  $\vdash \#False = (\text{init } \#False)$  by (simp add: init-defs Valid-def)
have 4:  $\vdash (\mathcal{M}(\text{EMPTY}) \wedge (\text{init } \#False)) = (\mathcal{M}(\text{EMPTY}) \wedge \#False)$  using 3 by auto
have 5:  $\vdash (\mathcal{M}(\text{EMPTY}) \wedge \#False) = \#False$  by simp
have 6:  $\vdash (\mathcal{M}(\text{EMPTY}) \wedge (\text{init } \#False)) = \#False$  using 4 5 by simp
have 7:  $\vdash \mathcal{M}(\text{EMPTY WITH LIFT}(\text{init } \#False)) = \#False$  using 2 6 by fastforce
have 8:  $\vdash \mathcal{M}(\text{GUARD } \#False) = \#False$  using 1 7 by fastforce
from 8 show ?thesis by auto

```

qed

lemma *MFirstFalseEqvFalse:*

$\vdash \mathcal{M}(\text{FIRST LIFT } \#False) = \#False$

proof -

```

have 1:  $\vdash \mathcal{M}(\text{FIRST LIFT } \#False) = \triangleright \#False$  by (simp)
have 2:  $\vdash \mathcal{M}(\text{FIRST LIFT } \#False) = \#False$  using FstFalse by fastforce
from 2 show ?thesis by auto

```

qed

lemma *MFailAlt:*

$\vdash \mathcal{M} \text{ FAIL} = \#False$

proof -

```

have 1:  $\vdash \mathcal{M} \text{ FAIL} = \mathcal{M}(\text{GUARD } (\#False))$  by (simp add: FAIL-d-def)
have 2:  $\vdash \mathcal{M}(\text{GUARD } (\#False)) = \#False$  using MGuardFalseEqvFalse by auto
from 1 2 show ?thesis by fastforce

```

qed

lemma *MFailEqvFirstFalseWithinEmpty:*

$FAIL \simeq ((FIRST\ LIFT\ \#False)\ WITHIN\ empty)$
proof –
have 1: $\vdash \mathcal{M}\ ((FIRST\ LIFT\ \#False)\ WITHIN\ (empty)) =$
 $\mathcal{M}((FIRST\ LIFT\ \#False)\ WITH\ LIFT(Limit\ empty))$
by (*simp add: WITHIN-d-def*)
have 2: $\vdash \mathcal{M}((FIRST\ LIFT\ \#False)\ WITH\ LIFT(Limit\ empty)) =$
 $(\mathcal{M}(FIRST\ LIFT\ \#False) \wedge (Limit\ empty))$
by (*simp*)
have 3: $\vdash \mathcal{M}((FIRST\ LIFT\ \#False)\ WITH\ LIFT(Limit\ empty)) = \#False$
using *MFirstFalseEqvFalse* **by** *auto*
have 4: $\vdash \mathcal{M}((FIRST\ LIFT\ \#False)\ WITHIN\ (empty)) = \#False$
using 1 3 **by** *fastforce*
have 5: $\vdash \mathcal{M}(FAIL) = \#False$
using *MFailAlt* **by** *simp*
from 4 5 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MEEmptyAlt*:
 $\vdash \mathcal{M}\ EMPTY = empty$
proof –
have 1: $\vdash \mathcal{M}\ (EMPTY) = \mathcal{M}\ ((FIRST\ LIFT\ empty))$ **by** (*simp add: EMPTY-d-def*)
have 2: $\vdash \mathcal{M}\ ((FIRST\ LIFT\ empty)) = \triangleright\ empty$ **by** (*simp*)
have 3: $\vdash \triangleright\ empty = empty$ **using** *FstEmpty* **by** *auto*
from 1 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *MSkipAlt*:
 $\vdash \mathcal{M}\ SKIP = skip$
proof –
have 1: $\vdash \mathcal{M}\ SKIP = \mathcal{M}\ (FIRST\ LIFT\ skip)$ **by** (*simp add: SKIP-d-def*)
have 2: $\vdash \mathcal{M}\ (FIRST\ LIFT\ skip) = \triangleright\ skip$ **by** (*simp*)
have 3: $\vdash \triangleright\ skip = skip$ **using** *FstSkip* **by** *simp*
from 1 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *MGuardAlt*:
 $\vdash \mathcal{M}\ (GUARD(w)) = (empty \wedge init\ w)$
proof –
have 1: $\vdash \mathcal{M}(GUARD(w)) = \mathcal{M}(EMPTY\ WITH\ (LIFT\ (init\ w)))$ **by** (*simp add: GUARD-d-def*)
have 2: $\vdash \mathcal{M}(EMPTY\ WITH\ (LIFT\ (init\ w))) = (\mathcal{M}\ EMPTY) \wedge (init\ w)$ **by** (*simp*)
have 3: $\vdash (\mathcal{M}\ EMPTY) \wedge (init\ w) = (empty \wedge (init\ w))$ **using** *MEEmptyAlt* **by** *fastforce*
have 4: $\vdash (empty \wedge (init\ w)) = (empty \wedge init\ w)$ **by** *simp*
from 1 2 3 4 **show** *?thesis* **by** *fastforce*
qed

lemma *MLengthAlt*:
 $\vdash \mathcal{M}\ (LEN(k)) = len(k)$
proof –
have 1: $\vdash \mathcal{M}(LEN(k)) = \mathcal{M}(FIRST\ LIFT(len(k)))$ **by** (*simp add: LEN-d-def*)
have 2: $\vdash \mathcal{M}(FIRST\ LIFT(len(k))) = \triangleright(len(k))$ **by** (*simp*)

have 3: $\vdash \triangleright(\text{len}(k)) = \text{len}(k)$ **using** *FstLenEqvLen* **by** *blast*
 from 1 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *MAalwaysAlt*:

$\vdash \mathcal{M}(a \text{ ALWAYS } w) = (\mathcal{M}(a) \wedge \Box (\text{init } w))$

proof –

have 1: $\vdash \mathcal{M}(a \text{ ALWAYS } w) = \mathcal{M}(a \text{ WITH LIFT}(bi (\text{finite} \longrightarrow \text{fin } (\text{init } w))))$
 by (*simp add: ALWAYS-d-def*)
 have 2: $\vdash \mathcal{M}(a \text{ WITH LIFT}(bi (\text{finite} \longrightarrow \text{fin } (\text{init } w)))) = (\mathcal{M}(a) \wedge (bi (\text{finite} \longrightarrow \text{fin } (\text{init } w))))$
 by (*simp*)
 have 3: $\vdash (\mathcal{M}(a) \wedge (bi (\text{finite} \longrightarrow \text{fin } (\text{init } w)))) = (\mathcal{M}(a) \wedge \Box (\text{init } w))$
 using *BoxStateEqvBiFinState* **by** *fastforce*
 from 1 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *MSometimeAlt*:

$\vdash \mathcal{M}(a \text{ SOMETIME } w) = (\mathcal{M}(a) \wedge \Diamond (\text{init } w))$

proof –

have 1: $\vdash \mathcal{M}(a \text{ SOMETIME } w) = \mathcal{M}(a \text{ WITH LIFT}(di (\text{finite} \wedge \text{fin } (\text{init } w))))$
 by (*simp add: SOMETIME-d-def*)
 have 2: $\vdash \mathcal{M}(a \text{ WITH LIFT}(di (\text{finite} \wedge \text{fin } (\text{init } w)))) = (\mathcal{M}(a) \wedge (di (\text{finite} \wedge \text{fin } (\text{init } w))))$
 by (*simp*)
 have 3: $\vdash \mathcal{M}(a \text{ WITH LIFT}(di (\text{finite} \wedge \text{fin } (\text{init } w)))) = (\mathcal{M}(a) \wedge \Diamond (\text{init } w))$
 using *DiamondStateEqvDiFinState* **by** *fastforce*
 from 1 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *MWithinAlt*:

$\vdash \mathcal{M}(a \text{ WITHIN } f) = (\mathcal{M}(a) \wedge (bs (\neg f)))$

proof –

have 1: $\vdash \mathcal{M}(a \text{ WITHIN } f) = \mathcal{M}(a \text{ WITH LIFT}(bs (\neg f)))$
 by (*simp add: WITHIN-d-def LIMIT-d-def*)
 have 2: $\vdash \mathcal{M}(a \text{ WITH LIFT}(bs (\neg f))) = (\mathcal{M}(a) \wedge (bs (\neg f)))$
 by (*simp*)
 from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *MTimesAlt*:

$\vdash \mathcal{M}(a \text{ TIMES } k) = \text{fpower } (\mathcal{M}(a)) \ k$

proof

(*induct k*)

case 0

then show *?case*

proof –

have 1: $\vdash \mathcal{M}(a \text{ TIMES } 0) = \mathcal{M} \text{ EMPTY}$ **by** *simp*
 have 2: $\vdash \mathcal{M} \text{ EMPTY} = \text{empty}$ **using** *MEmpyAlt* **by** *simp*
 have 3: $\vdash \text{empty} = \text{fpower } (\mathcal{M} \ a) \ 0$ **by** (*simp add: fpower-d-def*)
 from 1 2 3 **show** *?thesis* **by** *fastforce*

```

qed
next
case (Suc k)
then show ?case
proof -
  have 1:  $\vdash \mathcal{M}(a \text{ TIMES } Suc \ k) = \mathcal{M}(a \text{ THEN } (a \text{ TIMES } k))$ 
    by simp
  have 2:  $\vdash \mathcal{M}(a \text{ THEN } (a \text{ TIMES } k)) = (\mathcal{M} \ a) \frown (\mathcal{M} \ (a \text{ TIMES } k))$ 
    by (simp)
  have 3:  $\vdash (\mathcal{M} \ a) \frown (\mathcal{M} \ (a \text{ TIMES } k)) = (\mathcal{M} \ a) \frown (fpower \ (\mathcal{M} \ a) \ k)$ 
    using RightSChopEqvSChop Suc.hyps by blast
  have 4:  $\vdash (\mathcal{M} \ a) \frown (fpower \ (\mathcal{M} \ a) \ k) = fpower \ (\mathcal{M} \ a) \ (Suc \ k)$ 
    by (simp add: fpower-d-def schop-d-def)
  from 1 2 3 4 show ?thesis by fastforce
qed
qed

```

lemma MUptoAlt:

```

 $\vdash \mathcal{M}(a \text{ UPTO } b) =$ 
 $((\mathcal{M} \ a) \wedge bi \ (\neg(\mathcal{M} \ b)) \wedge finite) \vee ((\mathcal{M} \ b) \wedge bi \ (\neg(\mathcal{M} \ a)) \wedge finite) \vee ((\mathcal{M} \ a) \wedge (\mathcal{M} \ b))$ 
proof -
  have 1:  $\vdash \mathcal{M} \ (a \text{ UPTO } b) = \triangleright((\mathcal{M} \ a) \vee (\mathcal{M} \ b))$ 
    by (simp)
  have 2:  $\vdash \triangleright((\mathcal{M} \ a) \vee (\mathcal{M} \ b)) = ((\triangleright(\mathcal{M} \ a) \wedge (bs \ (\neg(\mathcal{M} \ b)))) \vee (\triangleright(\mathcal{M} \ b) \wedge (bs \ (\neg(\mathcal{M} \ a)))))$ 
    using FstWithOrEqv by blast
  have 3:  $\vdash ((\triangleright(\mathcal{M} \ a) \wedge (bs \ (\neg(\mathcal{M} \ b)))) \vee (\triangleright(\mathcal{M} \ b) \wedge (bs \ (\neg(\mathcal{M} \ a))))) =$ 
 $((\mathcal{M} \ a) \wedge ((\mathcal{M} \ b) \vee \neg(\mathcal{M} \ b)) \wedge (bs \ (\neg(\mathcal{M} \ b)))) \vee$ 
 $((\mathcal{M} \ b) \wedge ((\mathcal{M} \ a) \vee \neg(\mathcal{M} \ a)) \wedge (bs \ (\neg(\mathcal{M} \ a))))$ 
    using MFixFst by fastforce
  have 4:  $\vdash (((\mathcal{M} \ a) \wedge ((\mathcal{M} \ b) \vee \neg(\mathcal{M} \ b)) \wedge (bs \ (\neg(\mathcal{M} \ b)))) \vee$ 
 $((\mathcal{M} \ b) \wedge ((\mathcal{M} \ a) \vee \neg(\mathcal{M} \ a)) \wedge (bs \ (\neg(\mathcal{M} \ a))))) =$ 
 $((\mathcal{M} \ a) \wedge ((\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b))) \vee (\neg(\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b)))) \vee$ 
 $((\mathcal{M} \ b) \wedge ((\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a))) \vee (\neg(\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a))))$ 
    by auto
  have 5:  $\vdash (((\mathcal{M} \ a) \wedge ((\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b))) \vee (\neg(\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b)))) \vee$ 
 $((\mathcal{M} \ b) \wedge ((\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a))) \vee (\neg(\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a))))) =$ 
 $((\mathcal{M} \ a) \wedge ((\triangleright(\mathcal{M} \ b)) \vee (\neg(\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b)))) \vee$ 
 $((\mathcal{M} \ b) \wedge ((\triangleright(\mathcal{M} \ a)) \vee (\neg(\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a)))))$ 
    by (simp add: first-d-def)
  have 6:  $\vdash (((\mathcal{M} \ a) \wedge ((\triangleright(\mathcal{M} \ b)) \vee (\neg(\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b))))) \vee$ 
 $((\mathcal{M} \ b) \wedge ((\triangleright(\mathcal{M} \ a)) \vee (\neg(\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a))))) =$ 
 $((\mathcal{M} \ a) \wedge ((\mathcal{M} \ b) \vee (\neg(\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b))))) \vee$ 
 $((\mathcal{M} \ b) \wedge ((\mathcal{M} \ a) \vee (\neg(\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a)))))$ 
    using MFixFst by fastforce
  have 7:  $\vdash (\neg(\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b))) = (bi(\neg(\mathcal{M} \ b)) \wedge finite)$ 
    using AndBsEqvBi by blast
  have 8:  $\vdash (\neg(\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a))) = (bi(\neg(\mathcal{M} \ a)) \wedge finite)$ 
    using AndBsEqvBi by blast
  have 9:  $\vdash (((\mathcal{M} \ a) \wedge ((\mathcal{M} \ b) \vee ((\neg(\mathcal{M} \ b)) \wedge bs(\neg(\mathcal{M} \ b))))) \vee$ 
 $((\mathcal{M} \ b) \wedge ((\mathcal{M} \ a) \vee ((\neg(\mathcal{M} \ a)) \wedge bs(\neg(\mathcal{M} \ a))))) =$ 

```

$$(((\mathcal{M} a) \wedge ((\mathcal{M} b) \vee (bi(\neg(\mathcal{M} b)) \wedge finite))) \vee ((\mathcal{M} b) \wedge ((\mathcal{M} a) \vee (bi(\neg(\mathcal{M} a)) \wedge finite))))$$
using 7 8 **by** *fastforce*
have 10: $\vdash (((\mathcal{M} a) \wedge ((\mathcal{M} b) \vee (bi(\neg(\mathcal{M} b)) \wedge finite))) \vee ((\mathcal{M} b) \wedge ((\mathcal{M} a) \vee (bi(\neg(\mathcal{M} a)) \wedge finite)))) = (((\mathcal{M} a) \wedge (\mathcal{M} b)) \vee ((\mathcal{M} a) \wedge bi(\neg(\mathcal{M} b)) \wedge finite) \vee ((\mathcal{M} b) \wedge (\mathcal{M} a)) \vee ((\mathcal{M} b) \wedge bi(\neg(\mathcal{M} a)) \wedge finite)))$
by *auto*
have 11: $\vdash (((\mathcal{M} a) \wedge (\mathcal{M} b)) \vee ((\mathcal{M} a) \wedge bi(\neg(\mathcal{M} b)) \wedge finite) \vee ((\mathcal{M} b) \wedge (\mathcal{M} a)) \vee ((\mathcal{M} b) \wedge bi(\neg(\mathcal{M} a)) \wedge finite))) = (((\mathcal{M} a) \wedge bi(\neg(\mathcal{M} b)) \wedge finite) \vee ((\mathcal{M} b) \wedge bi(\neg(\mathcal{M} a)) \wedge finite) \vee ((\mathcal{M} a) \wedge (\mathcal{M} b)))$
by *auto*
show *?thesis*
by (*metis* 10 11 2 3 4 5 6 9 *MON.simps*(2) *int-eq*)
qed

lemma *MThruAlt*:

$\vdash \mathcal{M}(a \text{ THRU } b) = (((\mathcal{M} a) \wedge di(\mathcal{M} b)) \vee ((\mathcal{M} b) \wedge di(\mathcal{M} a)))$
proof –
have 1: $\vdash \mathcal{M}(a \text{ THRU } b) = \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))$
by (*simp*)
have 2: $\vdash \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b)) = ((\triangleright(\mathcal{M} a) \wedge di(\mathcal{M} b)) \vee (\triangleright(\mathcal{M} b) \wedge di(\mathcal{M} a)))$
using *FstDiAndDiEqv* **by** *auto*
have 3: $\vdash ((\triangleright(\mathcal{M} a) \wedge di(\mathcal{M} b)) \vee (\triangleright(\mathcal{M} b) \wedge di(\mathcal{M} a))) = (((\mathcal{M} a) \wedge di(\mathcal{M} b)) \vee ((\mathcal{M} b) \wedge di(\mathcal{M} a)))$
using *MFixFst* **by** *fastforce*
from 1 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *MHaltAlt*:

$\vdash \mathcal{M}(\text{HALT } w) = (\text{halt}(\text{init } w) \wedge finite)$
proof –
have 1: $\vdash \mathcal{M}(\text{HALT } w) = \mathcal{M}(\text{FIRST LIFT}(\text{fin}(\text{init } w)))$ **by** (*simp add: HALT-d-def*)
have 2: $\vdash \mathcal{M}(\text{FIRST LIFT}(\text{fin}(\text{init } w))) = \triangleright(\text{fin}(\text{init } w))$ **by** (*simp*)
have 3: $\vdash \triangleright(\text{fin}(\text{init } w)) = (\text{halt}(\text{init } w) \wedge finite)$ **using** *HaltStateEqvFstFinState* **by** *fastforce*
from 1 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *MFailUpto*:

$(\text{FAIL UPTO } a) \simeq (a)$
proof –
have 1: $\vdash \mathcal{M}(\text{FAIL UPTO } a) = \triangleright((\mathcal{M} \text{ FAIL}) \vee (\mathcal{M} a))$ **by** (*simp*)
have 2: $\vdash (\mathcal{M} \text{ FAIL} \vee \mathcal{M} a) = (\#False \vee \mathcal{M} a)$ **using** *MFailAlt* **by** *auto*
have 3: $\vdash \triangleright(\mathcal{M} \text{ FAIL} \vee (\mathcal{M} a)) = \triangleright(\#False \vee (\mathcal{M} a))$ **using** 2 *FstEqvRule* **by** *blast*
have 4: $\vdash (\#False \vee (\mathcal{M} a)) = \mathcal{M} a$ **by** *simp*
have 5: $\vdash \triangleright(\#False \vee (\mathcal{M} a)) = \triangleright(\mathcal{M} a)$ **using** 4 *FstEqvRule* **by** *blast*
have 6: $\vdash \triangleright(\mathcal{M} a) = \mathcal{M} a$ **using** *MFixFst* **by** *fastforce*
from 1 2 3 4 5 6 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MFailThru*:

$(\text{FAIL THRU } (a)) \simeq \text{FAIL}$

proof –

have 1: $\vdash \mathcal{M} (\text{FAIL THRU } (a)) = \triangleright (di(\mathcal{M} \text{ FAIL}) \wedge di(\mathcal{M} a))$
by (*simp*)

have 2: $\vdash \triangleright (di(\mathcal{M} \text{ FAIL}) \wedge di(\mathcal{M} a)) = \triangleright (di(\#False) \wedge di(\mathcal{M} a))$
using *MFailAlt* **by** (*metis 1 int-eq*)

have 3: $\vdash di \#False = \#False$
by (*simp add: di-defs Valid-def*)

hence 4: $\vdash \triangleright (di(\#False) \wedge di(\mathcal{M} a)) = \triangleright (\#False \wedge di(\mathcal{M} a))$
by (*metis 2 inteq-reflection*)

have 5: $\vdash \triangleright (\#False \wedge di(\mathcal{M} a)) = \triangleright \#False$
using *FstEqvRule* **by** *fastforce*

have 6: $\vdash \triangleright \#False = \#False$ **using** *FstFalse*
by *auto*

have 7: $\vdash \#False = \mathcal{M} \text{ FAIL}$
using *MFailAlt* **by** *auto*

from 1 2 4 5 6 7 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MFailAnd*:

$(\text{FAIL AND } a) \simeq \text{FAIL}$

proof –

have 1: $\vdash \mathcal{M} (\text{FAIL AND } a) = (\mathcal{M} \text{ FAIL} \wedge (\mathcal{M} a))$ **by** (*simp add: AND-d-def*)

have 2: $\vdash (\mathcal{M} \text{ FAIL} \wedge (\mathcal{M} a)) = (\#False \wedge (\mathcal{M} a))$ **using** *MFailAlt* **by** *fastforce*

have 3: $\vdash (\#False \wedge (\mathcal{M} a)) = \#False$ **by** *auto*

have 4: $\vdash \mathcal{M} (\text{FAIL AND } a) = \#False$ **using** 1 2 3 **by** *fastforce*

have 5: $\vdash \#False = \mathcal{M} \text{ FAIL}$ **using** *MFailAlt* **by** *auto*

from 1 2 3 4 5 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MThenFail*:

$(a \text{ THEN FAIL}) \simeq \text{FAIL}$

proof –

have 1: $\vdash \mathcal{M} (a \text{ THEN FAIL}) = (\mathcal{M} a) \frown (\mathcal{M} \text{ FAIL})$ **by** (*simp*)

have 2: $\vdash (\mathcal{M} a) \frown (\mathcal{M} \text{ FAIL}) = (\mathcal{M} a) \frown \#False$ **by** (*simp add: MFailAlt RightSChopEqvSChop*)

have 3: $\vdash (\mathcal{M} a) \frown \#False = \#False$ **by** (*simp add: SChopRightFalse*)

have 4: $\vdash \#False = \mathcal{M} \text{ FAIL}$ **using** *MFailAlt* **by** *auto*

from 1 2 3 4 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MFailThen*:

$(\text{FAIL THEN } a) \simeq \text{FAIL}$

proof –

have 1: $\vdash \mathcal{M} (\text{FAIL THEN } a) = (\mathcal{M} \text{ FAIL}) \frown (\mathcal{M} a)$ **by** (*simp*)

have 2: $\vdash (\mathcal{M} \text{ FAIL}) \frown (\mathcal{M} a) = \#False \frown (\mathcal{M} a)$ **using** *MFailAlt* **using** *LeftSChopEqvSChop* **by** *blast*

have 3: $\vdash \#False \frown (\mathcal{M} a) = \#False$ **by** (*simp add: schop-d-def chop-d-def Valid-def*)

have 4: $\vdash \#False = \mathcal{M} \text{ FAIL}$ **using** *MFailAlt* **by** *auto*

from 1 2 3 4 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MFailWith*:

$(\text{FAIL WITH } f) \simeq \text{FAIL}$

proof –

have 1: $\vdash \mathcal{M} (\text{FAIL WITH } f) = ((\mathcal{M} \text{ FAIL}) \wedge f)$ **by** (*simp*)

have 2: $\vdash ((\mathcal{M} \text{ FAIL}) \wedge f) = (\#False \wedge f)$ **using** *MFailAlt* **by** *auto*

have 3: $\vdash (\#False \wedge f) = \#False$ **by** *simp*

have 4: $\vdash \#False = \mathcal{M} \text{ FAIL}$ **using** *MFailAlt* **by** *auto*

from 1 2 3 4 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MWithFalse*:

$(a \text{ WITH } (\text{LIFT}(\#False))) \simeq \text{FAIL}$

proof –

have 1: $\vdash \mathcal{M} (a \text{ WITH } \text{LIFT}(\#False)) = ((\mathcal{M} a) \wedge \#False)$ **by** (*simp*)

have 2: $\vdash ((\mathcal{M} a) \wedge \#False) = \mathcal{M} \text{ FAIL}$ **using** *MFailAlt* **by** *auto*

from 1 2 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MWithTrue*:

$(a \text{ WITH } (\text{LIFT}(\#True))) \simeq a$

proof –

have 1: $\vdash \mathcal{M} (a \text{ WITH } \text{LIFT}(\#True)) = ((\mathcal{M} a) \wedge \#True)$ **by** (*simp*)

have 2: $\vdash ((\mathcal{M} a) \wedge \#True) = \mathcal{M} a$ **by** *simp*

from 1 2 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MEEmptyUpto*:

$(\text{EMPTY UPTO } a) \simeq \text{EMPTY}$

proof –

have 1: $\vdash \mathcal{M} (\text{EMPTY UPTO } a) = \triangleright(\mathcal{M} \text{ EMPTY} \vee (\mathcal{M} a))$ **by** (*simp*)

have 2: $\vdash \mathcal{M} \text{ EMPTY} = \text{empty}$ **using** *MEEmptyAlt* **by** *auto*

hence 3: $\vdash (\mathcal{M} \text{ EMPTY} \vee (\mathcal{M} a)) = (\text{empty} \vee (\mathcal{M} a))$ **by** *auto*

hence 4: $\vdash \triangleright(\mathcal{M} \text{ EMPTY} \vee \mathcal{M} a) = \triangleright(\text{empty} \vee \mathcal{M} a)$ **using** *FstEqvRule* **by** *blast*

have 5: $\vdash \triangleright(\text{empty} \vee \mathcal{M} a) = \text{empty}$ **using** *FstEmptyOrEqvEmpty* **by** *blast*

have 6: $\vdash \text{empty} = \mathcal{M} \text{ EMPTY}$ **using** *MEEmptyAlt* **by** *auto*

from 1 4 5 6 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MEEmptyThru*:

$(\text{EMPTY THRU } a) \simeq (a)$

proof –

have 1: $\vdash \mathcal{M}(\text{EMPTY THRU } a) = \triangleright(\text{di}(\mathcal{M} \text{ EMPTY}) \wedge \text{di}(\mathcal{M} a))$ **by** (*simp*)

have 2: $\vdash \text{di}(\mathcal{M} \text{ EMPTY}) = \text{di empty}$ **using** *MEEmptyAlt DiEqvDi* **by** *blast*

hence 3: $\vdash (\text{di}(\mathcal{M} \text{ EMPTY}) \wedge \text{di}(\mathcal{M} a)) = (\text{di empty} \wedge \text{di}(\mathcal{M} a))$ **by** *auto*

hence 4: $\vdash (\text{di empty} \wedge \text{di}(\mathcal{M} a)) = \text{di}(\mathcal{M} a)$ **using** *DiEmpty* **by** *auto*

have 5: $\vdash (\text{di}(\mathcal{M} \text{ EMPTY}) \wedge \text{di}(\mathcal{M} a)) = \text{di}(\mathcal{M} a)$ **using** 3 4 **by** *fastforce*

hence 6: $\vdash \triangleright(\text{di}(\mathcal{M} \text{ EMPTY}) \wedge \text{di}(\mathcal{M} a)) = \triangleright(\text{di}(\mathcal{M} a))$ **using** *FstEqvRule* **by** *blast*

have 7: $\vdash \triangleright(\text{di}(\mathcal{M} a)) = \triangleright(\mathcal{M} a)$ **using** *FstDiEqvFst* **by** *blast*

have 8: $\vdash \triangleright(\mathcal{M} a) = (\mathcal{M} a)$ **using** *MFixFst* **by** *fastforce*

from 1 6 7 8 show ?thesis using MonEq by (metis int-eq)
qed

lemma MThenEmpty:

(*a THEN EMPTY*) \simeq (*a WITH (LIFT finite)*)

proof –

have 1: $\vdash \mathcal{M}(\text{a THEN EMPTY}) = (\mathcal{M} \text{ a}) \frown (\mathcal{M} \text{ EMPTY})$ by (simp)

have 2: $\vdash (\mathcal{M} \text{ a}) \frown (\mathcal{M} \text{ EMPTY}) = (\mathcal{M} \text{ a}) \frown \text{empty}$ by (simp add: MEmptyAlt RightSChopEqvSChop)

have 3: $\vdash (\mathcal{M} \text{ a}) \frown \text{empty} = ((\mathcal{M} \text{ a}) \wedge \text{finite})$ by (simp add: ChopEmpty schop-d-def)

from 1 2 3 show ?thesis using MonEq by (metis MON.simps(5) inteq-reflection)

qed

lemma MEmptyThen:

(*EMPTY THEN a*) \simeq *a*

proof –

have 1: $\vdash \mathcal{M}(\text{EMPTY THEN a}) = (\mathcal{M} \text{ EMPTY}) \frown (\mathcal{M} \text{ a})$ by (simp)

have 2: $\vdash (\mathcal{M} \text{ EMPTY}) \frown (\mathcal{M} \text{ a}) = \text{empty} \frown (\mathcal{M} \text{ a})$ by (simp add: MEmptyAlt LeftSChopEqvSChop)

have 3: $\vdash \text{empty} \frown (\mathcal{M} \text{ a}) = (\mathcal{M} \text{ a})$ by (simp add: EmptySChop)

from 1 2 3 show ?thesis using MonEq by (metis int-eq)

qed

lemma MEmptyIterate:

(*EMPTY ITERATE b*) \simeq *EMPTY*

proof –

have 1: $\vdash \mathcal{M}(\text{EMPTY ITERATE b}) = \mathcal{M}(\text{EMPTY WITH LIFT (s chopstar } (\mathcal{M} \text{ b}))$

by (simp add: ITERATE-d-def)

have 2: $\vdash \mathcal{M}(\text{EMPTY WITH LIFT (s chopstar } (\mathcal{M} \text{ b}))) = (\mathcal{M} \text{ EMPTY} \wedge (\text{s chopstar } (\mathcal{M} \text{ b})))$

by (simp)

have 3: $\vdash (\mathcal{M} \text{ EMPTY} \wedge (\text{s chopstar } (\mathcal{M} \text{ b}))) = (\text{empty} \wedge (\text{s chopstar } (\mathcal{M} \text{ b})))$

using MEmptyAlt by auto

have 4: $\vdash (\text{empty} \wedge (\text{s chopstar } (\mathcal{M} \text{ b}))) = (\text{empty} \wedge (\text{empty} \vee (((\mathcal{M} \text{ b}) \wedge \text{more}) \frown (\text{s chopstar } (\mathcal{M} \text{ b}))))$

using SChopstarEqv by fastforce

have 5: $\vdash (\text{empty} \wedge (\text{empty} \vee (((\mathcal{M} \text{ b}) \wedge \text{more}) \frown (\text{s chopstar } (\mathcal{M} \text{ b})))) = \text{empty}$

by auto

have 6: $\vdash \mathcal{M}(\text{EMPTY ITERATE b}) = \mathcal{M} \text{ EMPTY}$

using 1 2 3 4 5 MEmptyAlt by fastforce

from 6 show ?thesis using MonEq by (metis int-eq)

qed

lemma MIterateIdemp:

(*a ITERATE a*) \simeq (*a*)

proof –

have 1: $\vdash \mathcal{M}(\text{a ITERATE a}) = \mathcal{M}(\text{a WITH LIFT (s chopstar } (\mathcal{M} \text{ a})))$ by (simp add: ITERATE-d-def)

have 2: $\vdash \mathcal{M}(\text{a WITH LIFT (s chopstar } (\mathcal{M} \text{ a}))) = ((\mathcal{M} \text{ a}) \wedge (\text{s chopstar } (\mathcal{M} \text{ a})))$ by (simp)

have 3: $\vdash ((\mathcal{M} \text{ a}) \wedge (\text{s chopstar } (\mathcal{M} \text{ a}))) = (\triangleright(\mathcal{M} \text{ a}) \wedge (\text{s chopstar } (\triangleright(\mathcal{M} \text{ a}))))$ using MFixFst

by (metis 2 inteq-reflection)

have 4: $\vdash (\triangleright(\mathcal{M} \text{ a}) \wedge (\text{s chopstar } (\triangleright(\mathcal{M} \text{ a})))) = \triangleright(\mathcal{M} \text{ a})$ using FstAndFstStarEqvFst by fastforce

have 5: $\vdash \triangleright(\mathcal{M} \text{ a}) = \mathcal{M} \text{ a}$ using MFixFst by fastforce

from 1 2 3 4 5 show ?thesis using MonEq by (metis int-eq)

qed

lemma *MUptoIdemp*:

$(a \text{ UPTO } a) \simeq (a)$

proof –

have 1: $\vdash \mathcal{M}(a \text{ UPTO } a) = \triangleright((\mathcal{M} a) \vee (\mathcal{M} a))$ **by** *auto*

have 2: $\vdash \triangleright((\mathcal{M} a) \vee (\mathcal{M} a)) = \triangleright(\mathcal{M} a)$ **using** *FstEqvRule* **by** *fastforce*

have 3: $\vdash \triangleright(\mathcal{M} a) = (\mathcal{M} a)$ **using** *MFixFst* **by** *fastforce*

from 1 2 3 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MThruIdemp*:

$(a \text{ THRU } a) \simeq (a)$

proof –

have 1: $\vdash \mathcal{M}(a \text{ THRU } a) = \triangleright(\text{di}(\mathcal{M} a) \wedge \text{di}(\mathcal{M} a))$ **by** *auto*

have 2: $\vdash \triangleright(\text{di}(\mathcal{M} a) \wedge \text{di}(\mathcal{M} a)) = \triangleright(\text{di}(\mathcal{M} a))$ **using** *FstEqvRule* **by** *fastforce*

have 3: $\vdash \triangleright(\text{di}(\mathcal{M} a)) = \triangleright(\mathcal{M} a)$ **using** *FstDiEqvFst* **by** *blast*

have 4: $\vdash \triangleright(\mathcal{M} a) = (\mathcal{M} a)$ **using** *MFixFst* **by** *fastforce*

from 1 2 3 4 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MAndIdemp*:

$(a \text{ AND } a) \simeq (a)$

proof –

have 1: $\vdash \mathcal{M}(a \text{ AND } a) = ((\mathcal{M} a) \wedge (\mathcal{M} a))$ **by** (*simp add: AND-d-def*)

have 2: $\vdash ((\mathcal{M} a) \wedge (\mathcal{M} a)) = (\mathcal{M} a)$ **by** *fastforce*

from 1 2 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MWithIdemp*:

$((a \text{ WITH } f) \text{ WITH } f) \simeq (a \text{ WITH } f)$

proof –

have 1: $\vdash \mathcal{M}((a \text{ WITH } f) \text{ WITH } f) = (((\mathcal{M} a) \wedge (f)) \wedge (f))$ **by** *auto*

have 2: $\vdash (((\mathcal{M} a) \wedge (f)) \wedge (f)) = ((\mathcal{M} a) \wedge (f))$ **by** *fastforce*

have 3: $\vdash ((\mathcal{M} a) \wedge (f)) = \mathcal{M}(a \text{ WITH } f)$ **by** *auto*

from 1 2 3 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MUptoCommut*:

$(a \text{ UPTO } b) \simeq (b \text{ UPTO } a)$

proof –

have 1: $\vdash \mathcal{M}(a \text{ UPTO } b) = \triangleright((\mathcal{M} a) \vee (\mathcal{M} b))$ **by** (*simp*)

have 2: $\vdash ((\mathcal{M} a) \vee (\mathcal{M} b)) = ((\mathcal{M} b) \vee (\mathcal{M} a))$ **by** *auto*

hence 3: $\vdash \triangleright((\mathcal{M} a) \vee (\mathcal{M} b)) = \triangleright((\mathcal{M} b) \vee (\mathcal{M} a))$ **using** *FstEqvRule* **by** *blast*

have 4: $\vdash \triangleright((\mathcal{M} b) \vee (\mathcal{M} a)) = \mathcal{M}(b \text{ UPTO } a)$ **by** *auto*

from 1 3 4 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MThruCommut*:

$(a \text{ THRU } b) \simeq (b \text{ THRU } a)$

proof –

have 1: $\vdash \mathcal{M}(a \text{ THRU } b) = \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))$ **by** (*simp*)
have 2: $\vdash (di(\mathcal{M} a) \wedge di(\mathcal{M} b)) = (di(\mathcal{M} b) \wedge di(\mathcal{M} a))$ **by** *auto*
hence 3: $\vdash \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b)) = \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} a))$ **using** *FstEqvRule* **by** *blast*
have 4: $\vdash \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} a)) = \mathcal{M}(b \text{ THRU } a)$ **by** *auto*
from 1 3 4 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MAndCommut*:

$(a \text{ AND } b) \simeq (b \text{ AND } a)$

proof –

have 1: $\vdash \mathcal{M}(a \text{ AND } b) = ((\mathcal{M} a) \wedge (\mathcal{M} b))$ **by** (*simp add: AND-d-def*)
have 2: $\vdash ((\mathcal{M} a) \wedge (\mathcal{M} b)) = ((\mathcal{M} b) \wedge (\mathcal{M} a))$ **by** *auto*
have 3: $\vdash ((\mathcal{M} b) \wedge (\mathcal{M} a)) = \mathcal{M}(b \text{ AND } a)$ **by** (*simp add: AND-d-def*)
from 1 2 3 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MWithCommut*:

$((a \text{ WITH } f) \text{ WITH } g) \simeq ((a \text{ WITH } g) \text{ WITH } f)$

proof –

have 1: $\vdash \mathcal{M}((a \text{ WITH } f) \text{ WITH } g) = (((\mathcal{M} a) \wedge (f)) \wedge (g))$ **by** *auto*
have 2: $\vdash (((\mathcal{M} a) \wedge (f)) \wedge (g)) = (((\mathcal{M} a) \wedge (g)) \wedge (f))$ **by** *auto*
have 3: $\vdash (((\mathcal{M} a) \wedge (g)) \wedge (f)) = \mathcal{M}((a \text{ WITH } g) \text{ WITH } f)$ **by** *auto*
from 1 2 3 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MWithAbsorp*:

$((a \text{ WITH } f) \text{ WITH } g) \simeq (a \text{ WITH } LIFT(f \wedge g))$

proof –

have 1: $\vdash \mathcal{M}((a \text{ WITH } f) \text{ WITH } g) = (((\mathcal{M} a) \wedge (f)) \wedge (g))$ **by** *auto*
have 2: $\vdash (((\mathcal{M} a) \wedge (f)) \wedge (g)) = ((\mathcal{M} a) \wedge (f \wedge g))$ **by** *auto*
from 1 2 **show** *?thesis* **by** (*simp add: MonEq*)
qed

lemma *MUptoAssoc*:

$((a \text{ UPTO } b) \text{ UPTO } c) \simeq (a \text{ UPTO } (b \text{ UPTO } c))$

proof –

have 1: $\vdash \mathcal{M}((a \text{ UPTO } b) \text{ UPTO } c) = \triangleright(\mathcal{M}(a \text{ UPTO } b) \vee (\mathcal{M} c))$
by (*simp*)
have 2: $\vdash \triangleright(\mathcal{M}(a \text{ UPTO } b) \vee (\mathcal{M} c)) = \triangleright(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b)) \vee (\mathcal{M} c))$
by *auto*
have 3: $\vdash \triangleright(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b)) \vee (\mathcal{M} c)) = \triangleright(((\mathcal{M} a) \vee (\mathcal{M} b)) \vee (\mathcal{M} c))$
using *FstFstOrEqvFstOrL* **by** *blast*
have 4: $\vdash (((\mathcal{M} a) \vee (\mathcal{M} b)) \vee (\mathcal{M} c)) = ((\mathcal{M} a) \vee ((\mathcal{M} b) \vee (\mathcal{M} c)))$
by *auto*
hence 5: $\vdash \triangleright(((\mathcal{M} a) \vee (\mathcal{M} b)) \vee (\mathcal{M} c)) = \triangleright((\mathcal{M} a) \vee ((\mathcal{M} b) \vee (\mathcal{M} c)))$
using *FstEqvRule* **by** *blast*
have 6: $\vdash \triangleright((\mathcal{M} a) \vee ((\mathcal{M} b) \vee (\mathcal{M} c))) = \triangleright((\mathcal{M} a) \vee \triangleright((\mathcal{M} b) \vee (\mathcal{M} c)))$
using *FstFstOrEqvFstOrR* **by** *fastforce*
have 7: $\vdash \triangleright((\mathcal{M} a) \vee \triangleright((\mathcal{M} b) \vee (\mathcal{M} c))) = \triangleright((\mathcal{M} a) \vee \mathcal{M}(b \text{ UPTO } c))$
by *auto*

have 8: $\vdash \triangleright((\mathcal{M} a) \vee \mathcal{M}(b \text{ UPTO } c)) = \mathcal{M}(a \text{ UPTO } (b \text{ UPTO } c))$
by *auto*
from 1 2 3 5 6 7 8 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *DiAndFiniteEqvDiFst*:
 $\vdash di (f \wedge finite) = di(\triangleright f)$
proof –
have 1: $\vdash (\triangleright f \wedge finite) = \triangleright f$
by (*meson FstImpFinite Prop10 Prop11*)
have 2: $\vdash (f \wedge finite); \# True = (\triangleright f \wedge finite); \# True$
using *DfEqvDfFst FstImpFinite unfolding df-d-def schop-d-def* **by** *auto*
show *?thesis*
by (*metis 1 2 di-d-def inteq-reflection*)
qed

lemma *FstEqvFstAndFinite*:
 $\vdash \triangleright f = \triangleright (f \wedge finite)$
by (*metis FstDfEqvFst FstDiEqvFst di-d-def inteq-reflection itl-def(15) schop-d-def*)

lemma *DiAndFiniteEqvDfAndFinite*:
 $\vdash (di f \wedge finite) = (df f \wedge finite)$
by (*auto simp add: Valid-def di-defs df-defs finite-defs*)

lemma *FstDiAndDiEqvFstDfAndDf*:
 $\vdash \triangleright(di f \wedge di g) = \triangleright(df f \wedge df g)$
proof –
have 1: $\vdash \triangleright(di f \wedge di g) = \triangleright((di f \wedge di g) \wedge finite)$
by (*simp add: FstEqvFstAndFinite*)
have 2: $\vdash ((di f \wedge di g) \wedge finite) = ((df f \wedge df g) \wedge finite)$
using *DiAndFiniteEqvDfAndFinite[of f] DiAndFiniteEqvDfAndFinite[of g]*
by *fastforce*
have 3: $\vdash \triangleright((di f \wedge di g) \wedge finite) = \triangleright((df f \wedge df g) \wedge finite)$
by (*simp add: 2 FstEqvRule*)
have 4: $\vdash \triangleright((df f \wedge df g) \wedge finite) = \triangleright(df f \wedge df g)$
by (*meson FstEqvFstAndFinite Prop11*)
show *?thesis*
by (*metis 1 3 4 inteq-reflection*)
qed

lemma *MThruAssoc*:
 $((a \text{ THRU } b) \text{ THRU } c) \simeq (a \text{ THRU } (b \text{ THRU } c))$
proof –
have 1: $\vdash \mathcal{M}((a \text{ THRU } b) \text{ THRU } c) = \triangleright(di(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) \wedge di(\mathcal{M} c))$
by *auto*
have 2: $\vdash di(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) = di((di(\mathcal{M} a) \wedge di(\mathcal{M} b)) \wedge finite)$
using *DiAndFiniteEqvDiFst* **by** *fastforce*
have 3: $\vdash di((di(\mathcal{M} a) \wedge di(\mathcal{M} b)) \wedge finite) = (di((\mathcal{M} a) \wedge finite) \wedge di((\mathcal{M} b) \wedge finite))$

by (metis DfDfAndEqvDf DiAndFiniteEqvDiFst MFixFst df-d-def di-d-def inteq-reflection schop-d-def)
 have 4: $\vdash di(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) = (di((\mathcal{M} a) \wedge finite) \wedge di((\mathcal{M} b) \wedge finite))$
 using 2 3 by fastforce
 hence 5: $\vdash (di(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) \wedge di((\mathcal{M} c) \wedge finite)) =$
 $(di((\mathcal{M} a) \wedge finite) \wedge (di((\mathcal{M} b) \wedge finite) \wedge di((\mathcal{M} c) \wedge finite)))$
 by auto
 have 6: $\vdash (di((\mathcal{M} b) \wedge finite) \wedge di((\mathcal{M} c) \wedge finite)) = di((di(\mathcal{M} b) \wedge di(\mathcal{M} c)) \wedge finite)$
 by (metis DfDfAndEqvDf DfEqvDiAndFinite DiAndFiniteEqvDiFst MFixFst inteq-reflection)
 have 7: $\vdash di((di(\mathcal{M} b) \wedge di(\mathcal{M} c)) \wedge finite) = di(\triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c)))$
 by (simp add: DiAndFiniteEqvDiFst)
 have 8: $\vdash (di((\mathcal{M} b) \wedge finite) \wedge di((\mathcal{M} c) \wedge finite)) =$
 $di(\triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c)))$
 using 6 7 by fastforce
 hence 9: $\vdash (di((\mathcal{M} a) \wedge finite) \wedge (di((\mathcal{M} b) \wedge finite) \wedge di((\mathcal{M} c) \wedge finite))) =$
 $(di((\mathcal{M} a) \wedge finite) \wedge di(\triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))))$
 by auto
 have 10: $\vdash (di(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) \wedge di((\mathcal{M} c) \wedge finite)) =$
 $(di((\mathcal{M} a) \wedge finite) \wedge di(\triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))))$
 using 5 9 by fastforce
 hence 11: $\vdash \triangleright(di(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) \wedge di((\mathcal{M} c) \wedge finite)) =$
 $\triangleright(di((\mathcal{M} a) \wedge finite) \wedge di(\triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))))$
 using FstEqvRule by fastforce
 have 12: $\vdash \triangleright(di((\mathcal{M} a) \wedge finite) \wedge di(\triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c)))) = \mathcal{M}(a \text{ THRU } (b \text{ THRU } c))$
 by (metis DiAndFiniteEqvDiFst MFixFst MON.simps(3) inteq-reflection)
 from 1 11 12 show ?thesis using MonEq
 by (metis DiAndFiniteEqvDiFst MFixFst inteq-reflection)
 qed

lemma MAndAssoc:

$$((a \text{ AND } b) \text{ AND } c) \simeq (a \text{ AND } (b \text{ AND } c))$$

proof –

have 1: $\vdash \mathcal{M}((a \text{ AND } b) \text{ AND } c) = ((\mathcal{M} a) \wedge (\mathcal{M} b) \wedge (\mathcal{M} c))$
 using AND-d-def by (metis MON.simps(5) MWithAbsorp eq-d-def)
 have 2: $\vdash ((\mathcal{M} a) \wedge (\mathcal{M} b) \wedge (\mathcal{M} c)) = \mathcal{M}(a \text{ AND } (b \text{ AND } c))$
 using AND-d-def by (simp add: AND-d-def)
 from 1 2 show ?thesis using MonEq by (metis int-eq)
 qed

lemma MThenAssoc:

$$((a \text{ THEN } b) \text{ THEN } c) \simeq (a \text{ THEN } (b \text{ THEN } c))$$

proof –

have 1: $\vdash \mathcal{M}((a \text{ THEN } b) \text{ THEN } c) = ((\mathcal{M} a) \frown (\mathcal{M} b)) \frown (\mathcal{M} c)$ by auto
 have 2: $\vdash ((\mathcal{M} a) \frown (\mathcal{M} b)) \frown (\mathcal{M} c) = (\mathcal{M} a) \frown ((\mathcal{M} b) \frown (\mathcal{M} c))$ by (meson Prop11 SChopAssoc)
 have 3: $\vdash (\mathcal{M} a) \frown ((\mathcal{M} b) \frown (\mathcal{M} c)) = \mathcal{M}(a \text{ THEN } (b \text{ THEN } c))$ by auto
 from 1 2 3 show ?thesis using MonEq by (metis int-eq)
 qed

lemma MUptoThruAbsorp:

$$(a \text{ UPTO } (a \text{ THRU } b)) \simeq a$$

proof –

have 1: $\vdash \mathcal{M}(a \text{ UPTO } (a \text{ THRU } b)) = \triangleright((\mathcal{M} a) \vee \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$
by *simp*
have 2: $\vdash \triangleright((\mathcal{M} a) \vee \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) =$
 $\triangleright((\mathcal{M} a) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$
using *FstFstOrEqvFstOrR* **by** *auto*
have 3: $\vdash ((\mathcal{M} a) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b))) =$
 $((\mathcal{M} a) \vee di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b))$
by *auto*
have 4: $\vdash (((\mathcal{M} a) \vee di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b))) =$
 $((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))$
using *OrDiEqvDi* **by** *fastforce*
have 5: $\vdash ((\mathcal{M} a) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b))) =$
 $((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))$
using *3 4* **by** *auto*
hence 6: $\vdash \triangleright((\mathcal{M} a) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b))) =$
 $\triangleright((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))$
using *FstEqvRule* **by** *blast*
have 7: $\vdash \triangleright((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b))) =$
 $((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$
 $bs(\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b))))$
by (*auto simp add: first-d-def*)
have 8: $\vdash ((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b))) =$
 $((di(\mathcal{M} a) \wedge (\mathcal{M} a)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$
by *auto*
hence 9: $\vdash (\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))) =$
 $(\neg((di(\mathcal{M} a) \wedge (\mathcal{M} a)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$
by *fastforce*
have 10: $\vdash (\neg((di(\mathcal{M} a) \wedge (\mathcal{M} a)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) =$
 $(\neg(((\mathcal{M} a)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$
using *AndDiEqv* **using** *5* **by** *auto*
have 11: $\vdash (\neg(((\mathcal{M} a)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) =$
 $(\neg(\mathcal{M} a) \wedge \neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$
by *auto*
have 12: $\vdash (\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))) =$
 $(\neg(\mathcal{M} a) \wedge \neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$
using *9 10 11* **by** *auto*
hence 13: $\vdash bs(\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))) =$
 $bs(\neg(\mathcal{M} a) \wedge \neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$
using *BsEqvRule* **by** *blast*
have 14: $\vdash bs(\neg(\mathcal{M} a) \wedge \neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) =$
 $(bs(\neg(\mathcal{M} a))) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$
using *BsAndEqv* **by** *fastforce*
have 141: $\vdash bs(\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))) =$
 $(bs(\neg(\mathcal{M} a))) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$
using *13 14* **by** *fastforce*
hence 15: $\vdash ((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$
 $bs(\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))) =$
 $((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$
 $bs(\neg(\mathcal{M} a))) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$
by *auto*

have 16: $\vdash ((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$
 $bs((\neg(\mathcal{M} a))) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) =$
 $((bs((\neg(\mathcal{M} a))) \wedge di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$
 $bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$
by *auto*
have 17: $\vdash ((bs((\neg(\mathcal{M} a))) \wedge di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$
 $bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) =$
 $((\triangleright(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$
 $bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$
using *FstEqvBsNotAndDi* **by** *fastforce*
have 18: $\vdash ((\triangleright(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$
 $bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) =$
 $((\mathcal{M} a) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$
 $bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$
using *MFixFst* **by** *fastforce*
have 19: $\vdash (((\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$
 $bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) =$
 $((\mathcal{M} a) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$
by *auto*
have 20: $\vdash (\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) = (\neg(di(\mathcal{M} a)) \vee \neg(di(\mathcal{M} b)))$
by *auto*
have 21: $\vdash (\neg(di(\mathcal{M} a)) \vee \neg(di(\mathcal{M} b))) = ((bi(\neg(\mathcal{M} a))) \vee (bi(\neg(\mathcal{M} b))))$
by (*simp add: bi-d-def*)
have 22: $\vdash (\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) = ((bi(\neg(\mathcal{M} a))) \vee (bi(\neg(\mathcal{M} b))))$
using 20 21 **by** *auto*
hence 23: $\vdash bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) = bs((bi(\neg(\mathcal{M} a))) \vee (bi(\neg(\mathcal{M} b))))$
using *BsEqvRule* **by** *blast*
have 24: $\vdash bs((bi(\neg(\mathcal{M} a))) \vee (bi(\neg(\mathcal{M} b)))) = bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b))$
using *BsOrBsEqvBsBiOrBi* **by** *fastforce*
have 25: $\vdash bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) = (bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b)))$
using 23 24 **using** *BsOrBsEqvBsBiOrBi* **by** *fastforce*
hence 26: $\vdash ((\mathcal{M} a) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) =$
 $((\mathcal{M} a) \wedge (bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b))))$
by *auto*
have 27: $\vdash ((\mathcal{M} a) \wedge (bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b)))) =$
 $(\triangleright(\mathcal{M} a) \wedge (bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b))))$
using *MFixFst* **by** *fastforce*
have 28: $\vdash (\triangleright(\mathcal{M} a) \wedge (bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b)))) =$
 $((\mathcal{M} a) \wedge bs(\neg(\mathcal{M} a)) \wedge (bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b))))$
by (*auto simp add: first-d-def*)
have 29: $\vdash ((\mathcal{M} a) \wedge bs(\neg(\mathcal{M} a)) \wedge (bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b)))) =$
 $((\mathcal{M} a) \wedge bs(\neg(\mathcal{M} a)))$
by *auto*
have 30: $\vdash ((\mathcal{M} a) \wedge bs(\neg(\mathcal{M} a))) = \triangleright(\mathcal{M} a)$
by (*simp add: first-d-def*)
have 31: $\vdash \triangleright(\mathcal{M} a) = (\mathcal{M} a)$
using *MFixFst* **by** *fastforce*
have 32: $\vdash \mathcal{M}(a \text{ UPTO } (a \text{ THRU } b)) =$
 $((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$
 $bs(\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b))))$

using 1 2 6 7 by fastforce
 have 33: $\vdash ((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$
 $bs(\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))) =$
 $((\mathcal{M} a) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$
 using 15 16 17 18 19 by (metis int-eq)
 have 34: $\vdash ((\mathcal{M} a) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) = (\mathcal{M} a)$
 using 26 27 28 29 30 31 by (metis int-eq)
 from 32 33 34 show ?thesis using MonEq by (metis int-eq)
 qed

lemma *MThruUptoAbsorp*:

$(a \text{ THRU } (a \text{ UPTO } b)) \simeq (a)$

proof –

have 1: $\vdash \mathcal{M}(a \text{ THRU } (a \text{ UPTO } b)) = \triangleright(di(\mathcal{M} a) \wedge di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))))$
 by simp
 have 2: $\vdash \triangleright(di(\mathcal{M} a) \wedge di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b)))) =$
 $\triangleright(di(\mathcal{M} a) \wedge di((\mathcal{M} a) \vee (\mathcal{M} b)))$
 by (metis DfEqvDfFst FstDiAndDiEqvFstDfAndDf inteq-reflection)
 have 3: $\vdash \triangleright(di(\mathcal{M} a) \wedge di((\mathcal{M} a) \vee (\mathcal{M} b))) =$
 $\triangleright(di(\mathcal{M} a) \wedge (di(\mathcal{M} a) \vee di(\mathcal{M} b)))$
 by (metis DiOrEqv FstEqvRule inteq-reflection lift-and-com)
 have 4: $\vdash (di(\mathcal{M} a) \wedge (di(\mathcal{M} a) \vee di(\mathcal{M} b))) = (di(\mathcal{M} a))$
 by auto
 hence 5: $\vdash \triangleright(di(\mathcal{M} a) \wedge (di(\mathcal{M} a) \vee di(\mathcal{M} b))) = \triangleright(di(\mathcal{M} a))$
 using FstEqvRule by blast
 have 6: $\vdash \triangleright(di(\mathcal{M} a)) = \triangleright(\mathcal{M} a)$
 using FstDiEqvFst by blast
 have 7: $\vdash \triangleright(\mathcal{M} a) = (\mathcal{M} a)$
 using MFixFst by fastforce
 from 1 2 3 5 6 7 show ?thesis using MonEq by (metis int-eq)
 qed

lemma *MUptoThruDistrib*:

$(a \text{ UPTO } (b \text{ THRU } c)) \simeq ((a \text{ UPTO } b) \text{ THRU } (a \text{ UPTO } c))$

proof –

have 1: $\vdash \mathcal{M}((a \text{ UPTO } b) \text{ THRU } (a \text{ UPTO } c)) =$
 $\triangleright(di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} c))))$
 by simp
 have 2: $\vdash (di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} c)))) =$
 $(di(((\mathcal{M} a) \vee (\mathcal{M} b)) \wedge finite) \wedge di(((\mathcal{M} a) \vee (\mathcal{M} c)) \wedge finite))$
 using DiAndFiniteEqvDiFst by fastforce
 have 3: $\vdash (di(((\mathcal{M} a) \vee (\mathcal{M} b)) \wedge finite) \wedge di(((\mathcal{M} a) \vee (\mathcal{M} c)) \wedge finite)) =$
 $((di((\mathcal{M} a) \wedge finite) \vee di((\mathcal{M} b) \wedge finite)) \wedge (di((\mathcal{M} a) \wedge finite) \vee di((\mathcal{M} c) \wedge finite)))$
 by (metis (no-types, lifting) DiDiAndEqvDi DiOrEqv FiniteOr inteq-reflection)
 have 4: $\vdash ((di((\mathcal{M} a) \wedge finite) \vee di((\mathcal{M} b) \wedge finite)) \wedge (di((\mathcal{M} a) \wedge finite) \vee di((\mathcal{M} c) \wedge finite))) =$
 $(di((\mathcal{M} a) \wedge finite) \vee (di((\mathcal{M} b) \wedge finite) \wedge di((\mathcal{M} c) \wedge finite)))$
 by auto
 have 5: $\vdash (di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} c)))) =$
 $(di((\mathcal{M} a) \wedge finite) \vee (di((\mathcal{M} b) \wedge finite) \wedge di((\mathcal{M} c) \wedge finite)))$
 using 2 3 4 by fastforce

hence 6: $\vdash \triangleright(di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} c)))) =$
 $\triangleright(di(\mathcal{M} a) \vee (di(\mathcal{M} b) \wedge di(\mathcal{M} c)))$
 by (metis DiAndFiniteEqvDiFst MFixFst int-simps(1) inteq-reflection)
 have 7: $\vdash \triangleright(di(\mathcal{M} a) \vee (di(\mathcal{M} b) \wedge di(\mathcal{M} c))) =$
 $\triangleright(\triangleright(di(\mathcal{M} a)) \vee \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c)))$
 using FstFstOrEqvFstOr by fastforce
 have 8: $\vdash \triangleright(di(\mathcal{M} a)) = \triangleright((\mathcal{M} a))$
 using FstDiEqvFst by blast
 have 9: $\vdash \triangleright((\mathcal{M} a)) = (\mathcal{M} a)$
 using MFixFst by fastforce
 have 10: $\vdash \triangleright(di(\mathcal{M} a)) = (\mathcal{M} a)$
 using 8 9 by fastforce
 hence 11: $\vdash (\triangleright(di(\mathcal{M} a)) \vee \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))) =$
 $((\mathcal{M} a) \vee \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c)))$
 by auto
 hence 12: $\vdash \triangleright(\triangleright(di(\mathcal{M} a)) \vee \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))) =$
 $\triangleright((\mathcal{M} a) \vee \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c)))$
 using FstEqvRule by blast
 have 13: $\vdash \triangleright((\mathcal{M} a) \vee \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))) = \mathcal{M}(a \text{ UPTO } (b \text{ THRU } c))$
 by simp
 from 1 6 7 12 13 show ?thesis using MonEq by (metis int-eq)
 qed

lemma MThruUptoDistrib:

$(a \text{ THRU } (b \text{ UPTO } c)) \simeq ((a \text{ THRU } b) \text{ UPTO } (a \text{ THRU } c))$

proof –

have 1: $\vdash \mathcal{M}((a \text{ THRU } b) \text{ UPTO } (a \text{ THRU } c)) =$
 $\triangleright(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b)) \vee \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} c)))$
 by simp
 have 2: $\vdash \triangleright(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b)) \vee \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} c))) =$
 $\triangleright((di(\mathcal{M} a) \wedge di(\mathcal{M} b)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} c)))$
 using FstFstOrEqvFstOr by auto
 have 3: $\vdash ((di((\mathcal{M} a) \wedge \text{finite}) \wedge di((\mathcal{M} b) \wedge \text{finite})) \vee (di((\mathcal{M} a) \wedge \text{finite}) \wedge di((\mathcal{M} c) \wedge \text{finite}))) =$
 $(di((\mathcal{M} a) \wedge \text{finite}) \wedge (di((\mathcal{M} b) \wedge \text{finite}) \vee di((\mathcal{M} c) \wedge \text{finite})))$ by auto
 have 4: $\vdash (di((\mathcal{M} a) \wedge \text{finite}) \wedge (di((\mathcal{M} b) \wedge \text{finite}) \vee di((\mathcal{M} c) \wedge \text{finite}))) =$
 $(di((\mathcal{M} a) \wedge \text{finite}) \wedge di((\mathcal{M} b) \vee (\mathcal{M} c)) \wedge \text{finite})$ using DiOrEqv
 by (metis 3 FiniteOr inteq-reflection)
 have 5: $\vdash (di((\mathcal{M} a) \wedge \text{finite}) \wedge di((\mathcal{M} b) \vee (\mathcal{M} c)) \wedge \text{finite}) =$
 $(di((\mathcal{M} a) \wedge \text{finite}) \wedge di(\triangleright((\mathcal{M} b) \vee (\mathcal{M} c))))$
 using DiAndFiniteEqvDiFst by fastforce
 have 6: $\vdash ((di((\mathcal{M} a) \wedge \text{finite}) \wedge di((\mathcal{M} b) \wedge \text{finite})) \vee (di((\mathcal{M} a) \wedge \text{finite}) \wedge di((\mathcal{M} c) \wedge \text{finite}))) =$
 $(di((\mathcal{M} a) \wedge \text{finite}) \wedge di(\triangleright((\mathcal{M} b) \vee (\mathcal{M} c))))$
 using 3 4 5 by fastforce
 hence 7: $\vdash \triangleright((di(\mathcal{M} a) \wedge di(\mathcal{M} b)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} c))) =$
 $\triangleright(di(\mathcal{M} a) \wedge di(\triangleright((\mathcal{M} b) \vee (\mathcal{M} c))))$
 by (metis DiAndFiniteEqvDiFst MFixFst int-simps(20) inteq-reflection)
 have 8: $\vdash \triangleright(di(\mathcal{M} a) \wedge di(\triangleright((\mathcal{M} b) \vee (\mathcal{M} c)))) =$
 $\mathcal{M}(a \text{ THRU } (b \text{ UPTO } c))$ by simp
 from 1 2 7 8 show ?thesis using MonEq by (metis int-eq)
 qed

lemma *MThruUptoRDistrib*:

$((a \text{ THRU } b) \text{ UPTO } c) \simeq ((a \text{ UPTO } c) \text{ THRU } (b \text{ UPTO } c))$

proof –

have 1: $((a \text{ THRU } b) \text{ UPTO } c) \simeq (c \text{ UPTO } (a \text{ THRU } b))$

using *MUptoCommut* **by** *auto*

have 2: $(c \text{ UPTO } (a \text{ THRU } b)) \simeq ((c \text{ UPTO } a) \text{ THRU } (c \text{ UPTO } b))$

using *MUptoThruDistrib* **by** *auto*

have 3: $(c \text{ UPTO } a) \simeq (a \text{ UPTO } c)$

using *MUptoCommut* **by** *auto*

have 4: $(c \text{ UPTO } b) \simeq (b \text{ UPTO } c)$

using *MUptoCommut* **by** *auto*

have 5: $((c \text{ UPTO } a) \text{ THRU } (c \text{ UPTO } b)) \simeq ((a \text{ UPTO } c) \text{ THRU } (c \text{ UPTO } b))$

using 3 **by** (*simp add: MonEqRefl MonEqSubstThru*)

have 6: $((a \text{ UPTO } c) \text{ THRU } (c \text{ UPTO } b)) \simeq ((c \text{ UPTO } b) \text{ THRU } (a \text{ UPTO } c))$

using *MThruCommut* **by** *auto*

have 7: $((c \text{ UPTO } b) \text{ THRU } (a \text{ UPTO } c)) \simeq ((b \text{ UPTO } c) \text{ THRU } (a \text{ UPTO } c))$

using 4 **by** (*simp add: MonEqRefl MonEqSubstThru*)

from 1 2 5 6 7 **show** *?thesis* **using** *MThruCommut MonEq* **by** (*metis int-eq*)

qed

lemma *MUptoThruRDistrib*:

$((a \text{ UPTO } b) \text{ THRU } c) \simeq ((a \text{ THRU } c) \text{ UPTO } (b \text{ THRU } c))$

proof –

have 1: $((a \text{ UPTO } b) \text{ THRU } c) \simeq (c \text{ THRU } (a \text{ UPTO } b))$

using *MThruCommut* **by** *auto*

have 2: $(c \text{ THRU } (a \text{ UPTO } b)) \simeq ((c \text{ THRU } a) \text{ UPTO } (c \text{ THRU } b))$

using *MThruUptoDistrib* **by** *auto*

have 3: $(c \text{ THRU } a) \simeq (a \text{ THRU } c)$

using *MThruCommut* **by** *auto*

have 4: $(c \text{ THRU } b) \simeq (b \text{ THRU } c)$

using *MThruCommut* **by** *auto*

have 5: $((c \text{ THRU } a) \text{ UPTO } (c \text{ THRU } b)) \simeq ((a \text{ THRU } c) \text{ UPTO } (c \text{ THRU } b))$

using 3 **by** (*simp add: MonEqRefl MonEqSubstUpto*)

have 6: $((a \text{ THRU } c) \text{ UPTO } (c \text{ THRU } b)) \simeq ((c \text{ THRU } b) \text{ UPTO } (a \text{ THRU } c))$

using *MUptoCommut* **by** *auto*

have 7: $((c \text{ THRU } b) \text{ UPTO } (a \text{ THRU } c)) \simeq ((b \text{ THRU } c) \text{ UPTO } (a \text{ THRU } c))$

using 4 **by** (*simp add: MonEqRefl MonEqSubstUpto*)

from 1 2 5 6 7 **show** *?thesis* **using** *MUptoCommut MonEq* **by** (*metis int-eq*)

qed

lemma *MWithAndDistrib*:

$((a \text{ AND } b) \text{ WITH } f) \simeq ((a \text{ WITH } f) \text{ AND } (b \text{ WITH } f))$

proof –

have 1: $\vdash \mathcal{M}((a \text{ AND } b) \text{ WITH } f) = (\mathcal{M}(a \text{ AND } b) \wedge f)$

by (*simp*)

have 2: $\vdash \mathcal{M}(a \text{ AND } b) = \mathcal{M}(a \text{ WITH LIFT}(\mathcal{M} \ b))$

by (*simp add: AND-d-def*)

have 3: $\vdash (\mathcal{M}(a \text{ AND } b) \wedge f) = (\mathcal{M}(a \text{ WITH LIFT}(\mathcal{M} \ b)) \wedge f)$

using 2 **by** *auto*

have 4: $\vdash \mathcal{M}(a \text{ WITH } (\text{LIFT}(\mathcal{M} b) \wedge f)) = (\mathcal{M}(a) \wedge \mathcal{M}(b) \wedge f)$
by *simp*
have 5: $\vdash (\mathcal{M}(a) \wedge \mathcal{M}(b) \wedge f) = ((\mathcal{M}(a) \wedge f) \wedge (\mathcal{M}(b) \wedge f))$
by *auto*
have 6: $\vdash ((\mathcal{M}(a) \wedge f) \wedge (\mathcal{M}(b) \wedge f)) = (\mathcal{M}(a \text{ WITH } f) \wedge \mathcal{M}(b \text{ WITH } f))$
by *simp*
have 7: $\vdash (\mathcal{M}(a \text{ WITH } f) \wedge \mathcal{M}(b \text{ WITH } f)) = \mathcal{M}((a \text{ WITH } f) \text{ WITH } \text{LIFT}(\mathcal{M}(b \text{ WITH } f)))$
by *simp*
have 8: $\vdash \mathcal{M}((a \text{ WITH } f) \text{ WITH } \text{LIFT}(\mathcal{M}(b \text{ WITH } f))) = \mathcal{M}((a \text{ WITH } f) \text{ AND } (b \text{ WITH } f))$
by (*simp add: AND-d-def*)
from 1 2 3 4 5 6 7 8 **show** ?thesis **using** MonEq **by** (*metis AND-d-def MWithAbsorp int-eq*)
qed

lemma *MHaltWithAndDistrib*:

$((\text{HALT } w) \text{ WITH } f) \text{ AND } ((\text{HALT } w) \text{ WITH } g) \simeq ((\text{HALT } w) \text{ WITH } \text{LIFT}(f \wedge g))$

proof –

have 1: $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ AND } ((\text{HALT } w) \text{ WITH } g)) =$
 $\mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ WITH } \text{LIFT}(\mathcal{M}((\text{HALT } w) \text{ WITH } g)))$
by (*simp add: AND-d-def*)
have 2: $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ WITH } \text{LIFT}(\mathcal{M}((\text{HALT } w) \text{ WITH } g))) =$
 $(\mathcal{M}(\text{HALT } w) \wedge f \wedge \mathcal{M}(\text{HALT } w) \wedge g)$
by *auto*
have 3: $\vdash (\mathcal{M}(\text{HALT } w) \wedge f \wedge \mathcal{M}(\text{HALT } w) \wedge g) = (\mathcal{M}(\text{HALT } w) \wedge f \wedge g)$
by *auto*
have 4: $\vdash (\mathcal{M}(\text{HALT } w) \wedge f \wedge g) = \mathcal{M}((\text{HALT } w) \text{ WITH } \text{LIFT}(f \wedge g))$
by *auto*
from 1 2 3 4 **show** ?thesis **using** MonEq **by** (*metis int-eq*)
qed

lemma *MHaltWithUptoHaltWithEqvHaltWithOr*:

$((\text{HALT } w) \text{ WITH } f) \text{ UPTO } ((\text{HALT } w) \text{ WITH } g) \simeq ((\text{HALT } w) \text{ WITH } \text{LIFT}(f \vee g))$

proof –

have 1: $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ UPTO } ((\text{HALT } w) \text{ WITH } g)) =$
 $\triangleright(\mathcal{M}((\text{HALT } w) \text{ WITH } f) \vee \mathcal{M}((\text{HALT } w) \text{ WITH } g))$
by (*simp*)
have 2: $\vdash \triangleright(\mathcal{M}((\text{HALT } w) \text{ WITH } f) \vee \mathcal{M}((\text{HALT } w) \text{ WITH } g)) =$
 $\triangleright((\mathcal{M}(\text{HALT } w) \wedge f) \vee (\mathcal{M}(\text{HALT } w) \wedge g))$
by *auto*
have 3: $\vdash ((\mathcal{M}(\text{HALT } w) \wedge f) \vee (\mathcal{M}(\text{HALT } w) \wedge g)) = (\mathcal{M}(\text{HALT } w) \wedge (f \vee g))$
by *auto*
have 4: $\vdash \triangleright((\mathcal{M}(\text{HALT } w) \wedge f) \vee (\mathcal{M}(\text{HALT } w) \wedge g)) = \triangleright(\mathcal{M}(\text{HALT } w) \wedge (f \vee g))$
using 3 *FstEqvRule* **by** *blast*
have 5: $\vdash \triangleright(\mathcal{M}(\text{HALT } w) \wedge (f \vee g)) = \triangleright(\mathcal{M}((\text{HALT } w) \text{ WITH } \text{LIFT}(f \vee g)))$
by *simp*
have 6: $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } \text{LIFT}(f \vee g))) = \triangleright(\mathcal{M}((\text{HALT } w) \text{ WITH } \text{LIFT}(f \vee g)))$
using *MFixFst* **by** *blast*
from 1 2 3 4 5 6 **show** ?thesis **using** MonEq **by** (*metis int-eq*)
qed

lemma *MHaltWithThruHaltWithEqvHaltWithAndHaltWith*:

$(((\text{HALT } w) \text{ WITH } f) \text{ THRU } ((\text{HALT } w) \text{ WITH } g)) \simeq (((\text{HALT } w) \text{ WITH } f) \text{ AND } ((\text{HALT } w) \text{ WITH } g))$

proof –

have 1: $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ THRU } ((\text{HALT } w) \text{ WITH } g)) =$
 $\triangleright (di(\mathcal{M}(\text{HALT } w) \wedge f) \wedge di(\mathcal{M}(\text{HALT } w) \wedge g))$
by *simp*
have 2: $\vdash (di(\mathcal{M}(\text{HALT } w) \wedge f) \wedge di(\mathcal{M}(\text{HALT } w) \wedge g)) =$
 $(di((\text{halt}(\text{init } w) \wedge \text{finite}) \wedge f) \wedge di((\text{halt}(\text{init } w) \wedge \text{finite}) \wedge g))$
using *MHaltAlt by (metis DiDiAndEqvDi inteq-reflection)*
have 3: $\vdash (di((\text{halt}(\text{init } w) \wedge \text{finite}) \wedge f) \wedge di((\text{halt}(\text{init } w) \wedge \text{finite}) \wedge g)) =$
 $di((\text{halt}(\text{init } w) \wedge \text{finite}) \wedge f \wedge g)$
by *(metis FstDiamondStateEqvHalt LFstAndDistrD inteq-reflection)*
have 4: $\vdash di((\text{halt}(\text{init } w) \wedge \text{finite}) \wedge f \wedge g) = di(\mathcal{M}(\text{HALT } w) \wedge f \wedge g)$
by *(metis 3 MHaltAlt inteq-reflection)*
have 5: $\vdash (di(\mathcal{M}(\text{HALT } w) \wedge f) \wedge di(\mathcal{M}(\text{HALT } w) \wedge g)) = di(\mathcal{M}(\text{HALT } w) \wedge f \wedge g)$
using *2 3 4 by fastforce*
have 6: $\vdash \triangleright (di(\mathcal{M}(\text{HALT } w) \wedge f) \wedge di(\mathcal{M}(\text{HALT } w) \wedge g)) = \triangleright (di(\mathcal{M}(\text{HALT } w) \wedge f \wedge g))$
using *5 FstEqvRule by blast*
have 7: $\vdash \triangleright (di(\mathcal{M}(\text{HALT } w) \wedge f \wedge g)) = \triangleright (\mathcal{M}(\text{HALT } w) \wedge f \wedge g)$
using *FstDiEqvFst by fastforce*
have 8: $\vdash \triangleright (\mathcal{M}(\text{HALT } w) \wedge f \wedge g) = \triangleright (\mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \wedge g)))$
by *simp*
have 9: $\vdash \mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \wedge g)) = \triangleright (\mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \wedge g)))$
using *MFixFst by blast*
have 10: $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ THRU } ((\text{HALT } w) \text{ WITH } g)) = \mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \wedge g))$
using *1 2 3 4 5 6 7 8 9 int-eq by metis*
have 11: $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ AND } ((\text{HALT } w) \text{ WITH } g)) = \mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \wedge g))$
using *MHaltWithAndDistrib using eq-d-def by blast*
have 12: $\vdash \mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \wedge g)) = \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ AND } ((\text{HALT } w) \text{ WITH } g))$
using *11 by fastforce*
from 10 12 show ?thesis using MonEq by (metis int-eq)
qed

lemma *MThenAndDistrib:*

$(a \text{ THEN } (b \text{ AND } c)) \simeq ((a \text{ THEN } b) \text{ AND } (a \text{ THEN } c))$

proof –

have 1: $\vdash \mathcal{M}(a \text{ THEN } (b \text{ AND } c)) = (\mathcal{M}(a)) \frown (\mathcal{M}(b \text{ AND } c))$
by *simp*
have 2: $\vdash (\mathcal{M}(a)) \frown (\mathcal{M}(b \text{ AND } c)) = (\mathcal{M}(a)) \frown (\mathcal{M}(b) \wedge \mathcal{M}(c))$
by *(simp add: AND-d-def)*
have 3: $\vdash (\mathcal{M}(a)) \frown (\mathcal{M}(b) \wedge \mathcal{M}(c)) = \triangleright (\mathcal{M}(a)) \frown (\mathcal{M}(b) \wedge \mathcal{M}(c))$
by *(simp add: LeftSChopEqvSChop MFixFst)*
have 4: $\vdash \triangleright (\mathcal{M}(a)) \frown (\mathcal{M}(b) \wedge \mathcal{M}(c)) = ((\triangleright (\mathcal{M}(a)) \frown (\mathcal{M}(b))) \wedge (\triangleright (\mathcal{M}(a)) \frown (\mathcal{M}(c))))$
by *(metis LFstAndDistrB Prop11 schop-d-def)*
have 5: $\vdash ((\triangleright (\mathcal{M}(a)) \frown (\mathcal{M}(b))) \wedge (\triangleright (\mathcal{M}(a)) \frown (\mathcal{M}(c)))) =$
 $((\mathcal{M}(a)) \frown (\mathcal{M}(b))) \wedge ((\mathcal{M}(a)) \frown (\mathcal{M}(c)))$ **using** *MFixFst*
by *(metis 4 inteq-reflection)*
have 6: $\vdash ((\mathcal{M}(a)) \frown (\mathcal{M}(b))) \wedge ((\mathcal{M}(a)) \frown (\mathcal{M}(c))) =$
 $(\mathcal{M}(a \text{ THEN } b) \wedge \mathcal{M}(a \text{ THEN } c))$

by *simp*
 have 7: $\vdash (\mathcal{M}(a \text{ THEN } b) \wedge \mathcal{M}(a \text{ THEN } c)) = \mathcal{M}((a \text{ THEN } b) \text{ AND } (a \text{ THEN } c))$
 by (*simp add: AND-d-def*)
 from 1 2 3 4 5 6 7 show ?thesis using *MonEq* by (*metis int-eq*)
 qed

lemma *MThenUptoDistrib*:

$(a \text{ THEN } (b \text{ UPTO } c)) \simeq ((a \text{ THEN } b) \text{ UPTO } (a \text{ THEN } c))$

proof –

have 1: $\vdash (\mathcal{M}(a \text{ THEN } (b \text{ UPTO } c))) = ((\mathcal{M} a) \frown (\triangleright((\mathcal{M} b) \vee (\mathcal{M} c))))$
 by *simp*
 have 2: $\vdash ((\mathcal{M} a) \frown (\triangleright((\mathcal{M} b) \vee (\mathcal{M} c)))) = (\triangleright(\mathcal{M} a) \frown (\triangleright((\mathcal{M} b) \vee (\mathcal{M} c))))$
 by (*simp add: MFixFst LeftSChopEqvSChop*)
 have 3: $\vdash (\triangleright(\mathcal{M} a) \frown (\triangleright((\mathcal{M} b) \vee (\mathcal{M} c)))) = ((\triangleright(\triangleright(\mathcal{M} a) \frown ((\mathcal{M} b) \vee (\mathcal{M} c))))$
 by (*metis FstFstChopEqvFstChopFst FstImpFinite Prop10 inteq-reflection schop-d-def*)
 have 4: $\vdash \triangleright(\mathcal{M} a) \frown ((\mathcal{M} b) \vee (\mathcal{M} c)) = (\mathcal{M} a) \frown ((\mathcal{M} b) \vee (\mathcal{M} c))$
 using *MFixFst* by (*metis LeftSChopEqvSChop inteq-reflection*)
 have 5: $\vdash (\mathcal{M} a) \frown ((\mathcal{M} b) \vee (\mathcal{M} c)) = ((\mathcal{M} a) \frown (\mathcal{M} b) \vee (\mathcal{M} a) \frown (\mathcal{M} c))$
 by (*simp add: SChopOrEqv*)
 have 6: $\vdash ((\mathcal{M} a) \frown (\mathcal{M} b) \vee (\mathcal{M} a) \frown (\mathcal{M} c)) = (\mathcal{M}(a \text{ THEN } b) \vee \mathcal{M}(a \text{ THEN } c))$
 by *simp*
 have 7: $\vdash \triangleright(\mathcal{M} a) \frown ((\mathcal{M} b) \vee (\mathcal{M} c)) = (\mathcal{M}(a \text{ THEN } b) \vee \mathcal{M}(a \text{ THEN } c))$
 using 6 5 4 by *fastforce*
 have 8: $\vdash \triangleright(\triangleright(\mathcal{M} a) \frown ((\mathcal{M} b) \vee (\mathcal{M} c))) = \triangleright(\mathcal{M}(a \text{ THEN } b) \vee \mathcal{M}(a \text{ THEN } c))$
 using 7 by (*simp add: FstEqvRule*)
 have 9: $\vdash \triangleright(\mathcal{M}(a \text{ THEN } b) \vee \mathcal{M}(a \text{ THEN } c)) = \mathcal{M}((a \text{ THEN } b) \text{ UPTO } (a \text{ THEN } c))$
 by *simp*
 from 9 7 1 2 3 show ?thesis by (*metis eq-d-def inteq-reflection*)
 qed

lemma *MThenthruDistrib*:

$(a \text{ THEN } (b \text{ THRU } c)) \simeq ((a \text{ THEN } b) \text{ THRU } (a \text{ THEN } c))$

proof –

have 1: $\vdash \mathcal{M}(a \text{ THEN } (b \text{ THRU } c)) = (\mathcal{M} a) \frown \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))$
 by *simp*
 have 2: $\vdash (\mathcal{M} a) \frown \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c)) = \triangleright(\mathcal{M} a) \frown \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))$
 by (*simp add: MFixFst LeftSChopEqvSChop*)
 have 3: $\vdash \triangleright(\mathcal{M} a) \frown \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c)) = \triangleright(\triangleright(\mathcal{M} a) \frown (di(\mathcal{M} b) \wedge di(\mathcal{M} c)))$
 by (*metis FstFstChopEqvFstChopFst FstImpFinite Prop10 inteq-reflection schop-d-def*)
 have 4: $\vdash \triangleright(\mathcal{M} a) \frown (di(\mathcal{M} b) \wedge di(\mathcal{M} c)) = (\triangleright(\mathcal{M} a) \frown di(\mathcal{M} b) \wedge \triangleright(\mathcal{M} a) \frown di(\mathcal{M} c))$
 by (*metis LFstAndDistrB inteq-reflection schop-d-def*)
 have 5: $\vdash (\triangleright(\mathcal{M} a) \frown di(\mathcal{M} b) \wedge \triangleright(\mathcal{M} a) \frown di(\mathcal{M} c)) = ((\mathcal{M} a) \frown di(\mathcal{M} b) \wedge (\mathcal{M} a) \frown di(\mathcal{M} c))$
 using *MFixFst* by (*metis 4 int-eq*)
 have 6: $\vdash (\mathcal{M} a) \frown di(\mathcal{M} b) = (\mathcal{M} a) \frown ((\mathcal{M} b) \frown \# \text{True})$
 by (*metis DiAndFiniteEqvDiFst MFixFst RightSChopEqvSChop di-d-def inteq-reflection schop-d-def*)
 have 7: $\vdash (\mathcal{M} a) \frown ((\mathcal{M} b) \frown \# \text{True}) = ((\mathcal{M} a) \frown (\mathcal{M} b)) \frown \# \text{True}$
 by (*simp add: SChopAssoc*)
 have 8: $\vdash ((\mathcal{M} a) \frown (\mathcal{M} b)) \frown \# \text{True} = di((\mathcal{M} a) \frown (\mathcal{M} b))$
 by (*metis DiAndFiniteEqvDiFst MFixFst MON.simps(4) di-d-def inteq-reflection schop-d-def*)
 have 9: $\vdash (\mathcal{M} a) \frown di(\mathcal{M} b) = di((\mathcal{M} a) \frown (\mathcal{M} b))$

```

    using 8 7 6 by fastforce
have 10:  $\vdash (\mathcal{M} a) \frown di(\mathcal{M} c) = (\mathcal{M} a) \frown ((\mathcal{M} c) \frown \#True)$ 
    by (metis DiAndFiniteEqvDiFst MFixFst di-d-def int-simps(20) inteq-reflection schop-d-def)
have 11:  $\vdash (\mathcal{M} a) \frown ((\mathcal{M} c) \frown \#True) = ((\mathcal{M} a) \frown (\mathcal{M} c)) \frown \#True$ 
    by (simp add: SChopAssoc)
have 12:  $\vdash ((\mathcal{M} a) \frown (\mathcal{M} c)) \frown \#True = di((\mathcal{M} a) \frown (\mathcal{M} c))$ 
    by (metis (no-types, lifting) 10 11 ChopAssoc di-d-def inteq-reflection schop-d-def)
have 13:  $\vdash (\mathcal{M} a) \frown di(\mathcal{M} c) = di((\mathcal{M} a) \frown (\mathcal{M} c))$ 
    using 12 11 10 by fastforce
have 14:  $\vdash ((\mathcal{M} a) \frown di(\mathcal{M} b) \wedge (\mathcal{M} a) \frown di(\mathcal{M} c)) = (di((\mathcal{M} a) \frown (\mathcal{M} b)) \wedge di((\mathcal{M} a) \frown (\mathcal{M} c)))$ 
    using 13 9 by fastforce
have 15:  $\vdash (di((\mathcal{M} a) \frown (\mathcal{M} b)) \wedge di((\mathcal{M} a) \frown (\mathcal{M} c))) = (di(\mathcal{M}(a \text{ THEN } b)) \wedge di(\mathcal{M}(a \text{ THEN } c)))$ 
    by simp
have 16:  $\vdash \triangleright (\mathcal{M} a) \frown (di(\mathcal{M} b) \wedge di(\mathcal{M} c)) = (di(\mathcal{M}(a \text{ THEN } b)) \wedge di(\mathcal{M}(a \text{ THEN } c)))$ 
    using 15 14 4 5 by fastforce
have 17:  $\vdash \triangleright (\triangleright (\mathcal{M} a) \frown (di(\mathcal{M} b) \wedge di(\mathcal{M} c))) = \triangleright (di(\mathcal{M}(a \text{ THEN } b)) \wedge di(\mathcal{M}(a \text{ THEN } c)))$ 
    using 16 by (simp add: FstEqvRule)
have 18:  $\vdash \triangleright (di(\mathcal{M}(a \text{ THEN } b)) \wedge di(\mathcal{M}(a \text{ THEN } c))) = \mathcal{M}((a \text{ THEN } b) \text{ THRU } (a \text{ THEN } c))$ 
    by simp
from 18 16 1 2 3 show ?thesis by (metis eq-d-def int-eq)
qed

```

end

2.14 Monitor Example

theory *MonitorExample*

imports

FOTheorems Monitor

begin

locale *Test* =

fixes $v :: \text{state} \Rightarrow \text{nat}$

fixes $y :: \text{state} \Rightarrow \text{bool}$

fixes $z :: \text{state} \Rightarrow \text{nat}$

fixes $F2 :: \text{nat} \text{ statefun} \Rightarrow \text{temporal}$

fixes $F3 :: \text{bool} \text{ statefun} \Rightarrow \text{temporal}$

fixes $F4 :: \text{nat} \text{ statefun} \Rightarrow \text{temporal}$

fixes $F5 :: \text{nat} \text{ statefun} \Rightarrow \text{temporal}$

fixes $Init2 :: \text{nat} \text{ statefun} \Rightarrow \text{temporal}$

fixes $Init3 :: \text{bool} \text{ statefun} \Rightarrow \text{temporal}$

fixes $Mon1 :: \text{state} \text{ monitor}$

fixes $Mon2 :: \text{state} \text{ monitor}$

fixes $Mon3 :: \text{state} \text{ monitor}$


```

fixes Mon4 :: state monitor
fixes Mon5 :: state monitor
fixes Mon6 :: state monitor
defines F2  $\equiv (\lambda v. TEMP \square (\#0 \leq \$v))$ 
defines F3  $\equiv (\lambda p. TEMP \square (\$p \vee \neg \$p))$ 
defines F4  $\equiv (\lambda z. TEMP \$z = \#0 \wedge z \text{ gets } \$z + \#1)$ 
defines F5  $\equiv (\lambda z. TEMP \text{ fin } (\$z = \#4))$ 
defines Init2  $\equiv (\lambda v. TEMP \$v = \#0)$ 
defines Init3  $\equiv (\lambda p. TEMP \$p)$ 
defines Mon1  $\equiv FIRST (F2 v)$ 
defines Mon2  $\equiv EMPTY \text{ UPTO } Mon1$ 
defines Mon3  $\equiv Mon1 \text{ WITH } (F2 v)$ 
defines Mon4  $\equiv Mon2 \text{ THEN } Mon1$ 
defines Mon5  $\equiv Mon3 \text{ THRU } Mon4$ 
defines Mon6  $\equiv (FIRST F4 z) \text{ WITH } (F5 z)$ 

```

lemma (**in** *Test*) *test*:

$\vdash \mathcal{M}(Mon1) = \text{empty}$

proof –

have 1: $\vdash \mathcal{M}(Mon1) = \triangleright(\square (\#0 \leq \$v))$

using *F2-def Mon1-def* **by** *fastforce*

have 2: $\vdash \square (\#0 \leq \$v)$

by (*simp add: Valid-def itl-defs*)

have 3: $\vdash \triangleright(\square (\#0 \leq \$v)) = \text{empty}$

using 2 **by** (*metis FstTrue int-eq int-eq-true*)

from 1 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma (**in** *Test*) *test1*:

$\vdash \mathcal{M}(Mon2) = \text{empty}$

proof –

have 1: $\vdash \mathcal{M}(Mon2) = \mathcal{M}(EMPTY \text{ UPTO } Mon1)$

using *Mon2-def* **by** *fastforce*

have 2: $\vdash \mathcal{M}(EMPTY \text{ UPTO } Mon1) = \triangleright(\mathcal{M}(EMPTY) \vee \mathcal{M}(Mon1))$

by *fastforce*

have 3: $\vdash \triangleright(\mathcal{M}(EMPTY) \vee \mathcal{M}(Mon1)) = \triangleright(\text{empty} \vee \text{empty})$

using *test* **by** (*metis 2 MEmptyAlt int-eq*)

have 4: $\vdash \triangleright(\text{empty} \vee \text{empty}) = \text{empty}$

using *FstEmptyOrEqvEmpty* **by** *blast*

from 1 2 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma (**in** *Test*) *test2*:

$\vdash \mathcal{M}(Mon3) = \text{empty}$

proof –

have 1: $\vdash \mathcal{M}(Mon3) = \mathcal{M}(Mon1 \text{ WITH } (F2 v))$ **using** *Mon3-def* **by** *fastforce*

have 2: $\vdash \mathcal{M}(Mon1 \text{ WITH } (F2 v)) = (\mathcal{M}(Mon1) \wedge (F2 v))$ **by** *fastforce*

have 3: $\vdash (\mathcal{M}(Mon1) \wedge (F2 v)) = (\text{empty} \wedge (F2 v))$ **using** *test* **by** *fastforce*

have 4: $\vdash (F2 v)$ **by** (*simp add: F2-def Valid-def itl-defs*)

have 5: $\vdash (\text{empty} \wedge (F2 v)) = \text{empty}$ **using** 4 **by** *fastforce*

from 1 2 3 5 show ?thesis by fastforce
qed

lemma (in Test) test3:

$\vdash \mathcal{M}(\text{Mon4}) = \text{empty}$

proof –

have 1: $\vdash \mathcal{M}(\text{Mon4}) = \mathcal{M}(\text{Mon2 THEN Mon1})$

using Mon4-def by fastforce

have 2: $\vdash \mathcal{M}(\text{Mon2 THEN Mon1}) = (\mathcal{M}(\text{Mon2}) \frown (\mathcal{M}(\text{Mon1})))$

by fastforce

have 3: $\vdash (\mathcal{M}(\text{Mon2}) \frown (\mathcal{M}(\text{Mon1}))) = \text{empty} \frown \text{empty}$

using test test1 using SChopEqvSChop by blast

have 4: $\vdash \text{empty} \frown \text{empty} = \text{empty}$

by (simp add: EmptySChop)

from 1 2 3 4 show ?thesis by fastforce

qed

lemma (in Test) test4:

$\vdash \mathcal{M}(\text{Mon5}) = \text{empty}$

proof –

have 1: $\vdash \mathcal{M}(\text{Mon5}) = \mathcal{M}(\text{Mon3 THRU Mon4})$

using Mon5-def by fastforce

have 2: $\vdash \mathcal{M}(\text{Mon3 THRU Mon4}) = \triangleright(\text{di}(\mathcal{M}(\text{Mon3})) \wedge \text{di}(\mathcal{M}(\text{Mon4})))$

by fastforce

have 3: $\vdash (\text{di}(\mathcal{M}(\text{Mon3})) \wedge \text{di}(\mathcal{M}(\text{Mon4}))) = (\text{di}(\text{empty}) \wedge \text{di}(\text{empty}))$

using test3 test2 by (metis inteq-reflection lift-and-com)

hence 4: $\vdash \triangleright(\text{di}(\mathcal{M}(\text{Mon3})) \wedge \text{di}(\mathcal{M}(\text{Mon4}))) = \triangleright(\text{di}(\text{empty}) \wedge \text{di}(\text{empty}))$

by (simp add: FstEqvRule)

have 5: $\vdash \triangleright(\text{di}(\text{empty}) \wedge \text{di}(\text{empty})) = \triangleright(\text{di}(\text{empty}))$

by simp

have 6: $\vdash \triangleright(\text{di}(\text{empty})) = \text{empty}$

using FstDiEqvFst FstEmpty by fastforce

from 6 5 4 2 1 show ?thesis by fastforce

qed

lemma (in Test) test5:

$\vdash \mathcal{M}(\text{Mon6}) = (\triangleright(\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))$

proof –

have 1: $\vdash \mathcal{M}(\text{Mon6}) = (\mathcal{M}(\text{FIRST } F4 \ z) \wedge (F5 \ z))$

using Mon6-def by fastforce

have 2: $\vdash (\mathcal{M}(\text{FIRST } F4 \ z) \wedge (F5 \ z)) = (\triangleright(F4 \ z) \wedge \text{fin}(\$z = \#4))$

using F5-def by fastforce

have 3: $\vdash (\triangleright(F4 \ z) \wedge \text{fin}(\$z = \#4)) = (\triangleright(\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))$

using F4-def by fastforce

from 1 2 3 show ?thesis by fastforce

qed

lemma (in Test) test5-1:

$\vdash \triangleright(\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4) \longrightarrow$

$\triangleright((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))$

using *FstWithAndImp* **by** *blast*

lemma (**in** *Test*) *test5-2*:

$(s \models (\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4) \wedge \text{finite}) =$
 $(\text{nfinite } s \wedge z (\text{nnth } s \ 0) = 0 \wedge (\forall i < \text{nlength } s. z (\text{nnth } s (\text{Suc } i)) = \text{Suc}(z (\text{nnth } s \ i))) \wedge$
 $z (\text{nnth } s (\text{the-enat } (\text{nlength } s))) = 4)$

apply (*simp add: itl-defs nsubn-def1*)

by (*metis ndropn-nfirst nfirst-eq-nnth-zero ntaken-nfirst*)

lemma (**in** *Test*) *test5-3*:

$(\text{nfinite } s \wedge z (\text{nnth } s \ 0) = 0 \wedge (\forall i < \text{nlength } s. z (\text{nnth } s (\text{Suc } i)) = \text{Suc}(z (\text{nnth } s \ i))) \wedge$
 $z (\text{nnth } s (\text{the-enat } (\text{nlength } s))) = 4)$

\implies

$(\text{nfinite } s \wedge z (\text{nnth } s \ 0) = 0 \wedge (\forall i \leq \text{nlength } s. z (\text{nnth } s \ i) = i)$
 $\wedge z (\text{nnth } s (\text{the-enat } (\text{nlength } s))) = 4)$

proof –

assume *a0*: $\text{nfinite } s \wedge (z (\text{nnth } s \ 0) = 0 \wedge (\forall i < \text{nlength } s. z (\text{nnth } s (\text{Suc } i)) = \text{Suc}(z (\text{nnth } s \ i))) \wedge$
 $z (\text{nnth } s (\text{the-enat } (\text{nlength } s))) = 4)$

show $\text{nfinite } s \wedge (z (\text{nnth } s \ 0) = 0 \wedge (\forall i \leq \text{nlength } s. z (\text{nnth } s \ i) = i)$

$\wedge z (\text{nnth } s (\text{the-enat } (\text{nlength } s))) = 4)$

proof –

have *0*: $\text{nfinite } s$ **using** *a0* **by** *auto*

have *1*: $z (\text{nnth } s \ 0) = 0$ **using** *a0* **by** *auto*

have *2*: $z (\text{nnth } s (\text{the-enat } (\text{nlength } s))) = 4$ **using** *a0* **by** *auto*

have *3*: $(\forall i \leq \text{nlength } s. z (\text{nnth } s \ i) = i)$

proof

fix *i*

show $i \leq \text{nlength } s \longrightarrow z (\text{nnth } s \ i) = i$

proof

(*induct i*)

case *0*

then show *?case* **by** (*simp add: 1*)

next

case (*Suc i*)

then show *?case* **by** (*simp add: Suc-ile-eq a0*)

qed

qed

from *0 1 2 3* **show** *?thesis* **by** *auto*

qed

qed

lemma (**in** *Test*) *test5-4*:

$(\text{nfinite } s \wedge z (\text{nnth } s \ 0) = 0 \wedge (\forall i \leq \text{nlength } s. z (\text{nnth } s \ i) = i)$

$\wedge z (\text{nnth } s (\text{the-enat } (\text{nlength } s))) = 4) \implies$

$(\text{nfinite } s \wedge z (\text{nnth } s \ 0) = 0 \wedge (\forall i < \text{nlength } s. z (\text{nnth } s (\text{Suc } i)) = \text{Suc}(z (\text{nnth } s \ i))) \wedge$
 $z (\text{nnth } s (\text{the-enat } (\text{nlength } s))) = 4)$

proof –

assume $a0$: $(nfinite\ s \wedge z\ (nnth\ s\ 0) = 0 \wedge (\forall\ i \leq nlength\ s. z\ (nnth\ s\ i) = i) \wedge z\ (nnth\ s\ (the-enat\ (nlength\ s))) = 4)$
show $(nfinite\ s \wedge z\ (nnth\ s\ 0) = 0 \wedge (\forall\ i < nlength\ s. z\ (nnth\ s\ (Suc\ i)) = Suc(z\ (nnth\ s\ i))) \wedge z\ (nnth\ s\ (the-enat\ (nlength\ s))) = 4)$

proof –

have 0 : $nfinite\ s$ **using** $a0$ **by** *auto*
have 1 : $z\ (nnth\ s\ 0) = 0$ **using** $a0$ **by** *auto*
have 2 : $z\ (nnth\ s\ (the-enat\ (nlength\ s))) = 4$ **using** $a0$ **by** *auto*
have 3 : $(\forall\ i < nlength\ s. z\ (nnth\ s\ (Suc\ i)) = Suc(z\ (nnth\ s\ i)))$ **by** (*simp add: Suc-ile-eq a0*)
from $0\ 1\ 2\ 3$ **show** *?thesis* **by** *auto*

qed

qed

lemma (**in** *Test*) *test5-5*:

$(nfinite\ s \wedge z\ (nnth\ s\ 0) = 0 \wedge (\forall\ i < nlength\ s. z\ (nnth\ s\ (Suc\ i)) = Suc(z\ (nnth\ s\ i))) \wedge z\ (nnth\ s\ (the-enat\ (nlength\ s))) = 4)$
 $=$
 $(nfinite\ s \wedge z\ (nnth\ s\ 0) = 0 \wedge (\forall\ i \leq nlength\ s. z\ (nnth\ s\ i) = i) \wedge z\ (nnth\ s\ (the-enat\ (nlength\ s))) = 4)$

using *test5-3 test5-4* **by** *blast*

lemma (**in** *Test*) *test5-6* :

$(nfinite\ s \wedge z\ (nnth\ s\ 0) = 0 \wedge (\forall\ i \leq nlength\ s. z\ (nnth\ s\ i) = i) \wedge z\ (nnth\ s\ (the-enat\ (nlength\ s))) = 4) =$
 $(nlength\ s = 4 \wedge (\forall\ i \leq nlength\ s. z\ (nnth\ s\ i) = i))$

by (*metis dual-order.refl nfinite-conv-nlength-enat numeral-eq-enat the-enat.simps zero-enat-def zero-le-numeral*)

lemma (**in** *Test*) *test5-7* :

$(s \models (\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4) \wedge finite) =$
 $(nlength\ s = 4 \wedge (\forall\ i \leq nlength\ s. z\ (nnth\ s\ i) = i))$
using *test5-6[of s] test5-5[of s] test5-2[of s]* **by** *presburger*

lemma (**in** *Test*) *test5-8-0*:

$(s \models \triangleright((\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4))) =$
 $(s \models \triangleright((\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4)) \wedge finite)$
using *FstEqvFstAndFinite[of TEMP (\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4)]*
unfolding *Valid-def* **by** *auto*

lemma (**in** *Test*) *test5-8-1*:

$(\vdash \triangleright((\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4)) \wedge finite) =$
 $(\vdash \triangleright((\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4) \wedge finite))$
proof –
have 1 : $(\vdash ((\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4)) \wedge finite) =$
 $((\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4) \wedge finite)$
by *auto*

show *?thesis* **by** (*metis 1 int-eq*)
qed

lemma (**in** *Test*) *test5-8-2*:

$(s \models \triangleright((\$z=\#0 \wedge z \text{ gets } \$z+\#1) \wedge \text{fin}(\$z=\#4)) \wedge \text{finite}) =$
 $(s \models \triangleright((\$z=\#0 \wedge z \text{ gets } \$z+\#1) \wedge \text{fin}(\$z=\#4) \wedge \text{finite}))$

using *test5-8-1 unfolding Valid-def*

by (*simp add: Fstsem*)

lemma (**in** *Test*) *test5-8* :

$(s \models \triangleright((\$z=\#0 \wedge z \text{ gets } \$z+\#1) \wedge \text{fin}(\$z=\#4)) \wedge \text{finite})) =$
 $((s \models ((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{finite}) \wedge \text{nlength } s = 0 \vee$
 $0 < \text{nlength } s \wedge$
 $\text{nfinite } s \wedge$
 $(s \models ((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{finite}) \wedge$
 $(\forall ia. \text{enat } ia < \text{nlength } s \longrightarrow (\text{ntaken } ia \ s \models \neg (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge$
 $\text{finite}))))))$

using *Fstsem[of TEMP ((\\$z=\#0 \wedge z gets \\$z+\#1) \wedge fin(\\$z=\#4)) \wedge finite]*

by *simp*

lemma (**in** *Test*) *test5-9* :

$\neg (s \models ((\$z=\#0 \wedge z \text{ gets } \$z+\#1) \wedge \text{fin}(\$z=\#4)) \wedge \text{finite}) \wedge \text{nlength } s = 0)$

using *test5-7*

by (*metis (mono-tags, lifting) unl-lift2 zero-neq-numeral*)

lemma (**in** *Test*) *test5-10*:

$(s \models ((\$z=\#0 \wedge z \text{ gets } \$z+\#1) \wedge \text{fin}(\$z=\#4)) \wedge \text{finite})$
 \implies
 $0 < \text{nlength } s \wedge$
 $\text{nfinite } s \wedge$
 $(s \models ((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{finite}) \wedge$
 $(\forall ia. \text{enat } ia < \text{nlength } s \longrightarrow (\text{ntaken } ia \ s \models \neg (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge$
 $\text{finite}))))$

proof –

assume *a0*: $s \models ((\$z=\#0 \wedge z \text{ gets } \$z+\#1) \wedge \text{fin}(\$z=\#4)) \wedge \text{finite}$

show $0 < \text{nlength } s \wedge$

$\text{nfinite } s \wedge$

$(s \models ((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{finite}) \wedge$

$(\forall ia. \text{enat } ia < \text{nlength } s \longrightarrow (\text{ntaken } ia \ s \models \neg (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge$
 $\text{finite}))))$

proof –

have *0*: $\text{nfinite } s$ **using** *a0* **by** (*simp add: finite-defs*)

have *1*: $0 < \text{nlength } s$ **using** *test5-7 a0 gr-zeroI test5-9* **by** *blast*

have *2*: $(\forall ia. \text{enat } ia < \text{nlength } s \longrightarrow (\text{ntaken } ia \ s \models \neg (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge$
 $\text{finite}))))$

```

proof
  fix ia
  show enat ia < nlength s  $\longrightarrow$ 
    (ntaken ia s  $\models \neg (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin } (\$z = \#4)) \wedge \text{finite}))$ )
  proof –
  have 1: (ntaken ia s  $\models \neg (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin } (\$z = \#4)) \wedge \text{finite})) =$ 
    ( $\neg(\text{ntaken ia s} \models (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin } (\$z = \#4)) \wedge \text{finite}))$ )
    by auto
  have 2: (ntaken ia s  $\models (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin } (\$z = \#4)) \wedge \text{finite}) =$ 
    (nlength (ntaken ia s) = 4  $\wedge (\forall i. \text{enat } i \leq \text{nlength (ntaken ia s)} \longrightarrow z (\text{nnth (ntaken ia s) } i) =$ 
i))
    using test5-7[of ntaken ia s] by auto
  have 3: enat ia < nlength s  $\longrightarrow$ 
    ( $\neg(\text{nlength (ntaken ia s)} = 4 \wedge (\forall i. \text{enat } i \leq \text{nlength (ntaken ia s)} \longrightarrow z (\text{nnth (ntaken ia s) } i) =$ 
i))
    using a0 using test5-7[of ntaken ia s]
    using not-less-iff-gr-or-eq test5-7 by fastforce
  from 1 2 3 show ?thesis by blast
  qed
  qed
  from 0 1 2 show ?thesis using a0 by blast
  qed
qed

lemma (in Test) test5-11 :
  (s  $\models \triangleright((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin } (\$z = \#4))$ ) =
  (s  $\models ((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin } (\$z = \#4)) \wedge \text{finite}$ )
by (meson test5-10 test5-8 test5-8-0)

lemma (in Test) test5-12 :
   $\vdash \triangleright((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin } (\$z = \#4)) = (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin } (\$z = \#4)) \wedge \text{finite})$ 
using test5-11 by (simp add: Valid-def)

end

```

2.15 Interval Temporal Algebra

We have given an algebraic axiom system for Interval Temporal Logic: Interval Temporal Algebra. The axiom system is a combination of a variant of Kleene algebra and Omega algebra plus axioms for linearity and confluence. Kleene algebra and Omega algebra have been defined by Alasdair Armstrong, Georg Struth and Tjark Weber in [1].

```

theory ITA
imports Semantics Chopstar Omega
begin

```

2.15.1 Definition of Set of intervals and Operations on them

type-synonym $'a\ iintervals = 'a\ nellist\ set$

definition $lan :: ('a :: world)\ formula \Rightarrow 'a\ iintervals$

where $lan\ f = \{ \sigma . (\sigma \models f) \}$

definition $fusion :: 'a\ iintervals \Rightarrow 'a\ iintervals \Rightarrow 'a\ iintervals\ (\mathbf{infixl}\ \cdot\ 70)$

where $X \cdot Y = \{ \sigma .$

$(\exists\ \sigma 1\ \sigma 2. \sigma = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge$
 $(\sigma 1 \in X) \wedge (\sigma 2 \in Y) \wedge (nlast\ \sigma 1 = nfirst\ \sigma 2))$
 $\vee (\neg\ nfinite\ \sigma \wedge \sigma \in X) \}$

definition $semt :: 'a\ iintervals\ (SEmpty)$

where

$SEmpty \equiv \{ \sigma .\ nlength\ \sigma = 0 \}$

definition $smore :: 'a\ iintervals\ (SMore)$

where

$SMore \equiv -\ SEmpty$

definition $sskip :: 'a\ iintervals\ (SSkip)$

where

$SSkip \equiv -(SEmpty \cup (SMore \cdot SMore))$

definition $sfalse :: 'a\ iintervals\ (SFalse)$

where

$SFalse \equiv \{\}$

definition $strue :: 'a\ iintervals\ (STrue)$

where

$STrue \equiv -\{\}$

definition $sinit :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SInit\ -)\ [85]\ 85)$

where

$SInit\ X \equiv (X \cap SEmpty) \cdot STrue$

definition $sinf :: 'a\ iintervals\ (SInf)$

where $SInf \equiv STrue \cdot SFalse$

definition $sfinite :: 'a\ iintervals\ (SFinite)$

where $SFinite \equiv -\ SInf$

definition $sfmore :: 'a\ iintervals\ (SFMore)$

where $SFMore \equiv SFinite \cap SMore$

definition $sfin :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SFin\ -)\ [85]\ 85)$

where

$SFin\ X \equiv SFinite \cdot (X \cap SEmpty)$

definition $ssometime :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SSometime\ -)\ [85]\ 85)$

where

$$SSometime\ X \equiv SFinite.X$$

definition $salways :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SAlways\ -)\ [85]\ 85)$

where

$$SAlways\ X \equiv -(SSometime\ (-X))$$

definition $sdi :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SDi\ -)\ [85]\ 85)$

where

$$SDi\ X \equiv X.STrue$$

definition $sbi :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SBi\ -)\ [85]\ 85)$

where

$$SBi\ X \equiv -(SDi\ (-X))$$

definition $sda :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SDa\ -)\ [85]\ 85)$

where

$$SDa\ X \equiv SFinite.X.STrue$$

definition $sba :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SBa\ -)\ [85]\ 85)$

where

$$SBa\ X \equiv -(SDa\ (-X))$$

definition $snext :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SNext\ -)\ [85]\ 85)$

where

$$SNext\ X \equiv SSkip.X$$

definition $swnext :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SWnext\ -)\ [85]\ 85)$

where

$$SWnext\ X \equiv -(SSkip.(-X))$$

definition $sprev :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SPrev\ -)\ [85]\ 85)$

where

$$SPrev\ X \equiv X.SSkip$$

definition $swprev :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SWprev\ -)\ [85]\ 85)$

where

$$SWprev\ X \equiv -((-X).SSkip)$$

primrec $spower :: 'a\ iintervals \Rightarrow nat \Rightarrow 'a\ iintervals\ ((SPower\ -\ -)\ [88,88]\ 87)$

where

$$\begin{aligned} pwr-0 &: SPower\ X\ 0 = SEmpty \\ | pwr-Suc: &SPower\ X\ (Suc\ n) = ((X \cap SFinite).(SPower\ X\ n)) \end{aligned}$$

definition $sfpowerstar :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SFPowerstar\ -)\ [85]\ 85)$

where

$$SFPowerstar\ X \equiv (\bigcup n. SPower\ X\ n)$$

definition $spowerstar :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SPowerstar\ -)\ [85]\ 85)$

where

$SPowerstar\ X \equiv (SFPowerstar\ X) \cdot (SEmpty \cup (X \cap SInf))$

definition $sstar :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SStar\ -)\ [85]\ 85)$

where

$SStar\ X \equiv SPowerstar(X \cap SMore)$

2.15.2 Simplification Lemmas

lemma *snot-elim* :

$x \in -X \longleftrightarrow x \notin X$

by *simp*

lemma *sor-elim* :

$x \in (X \cup Y) \longleftrightarrow (x \in X \vee x \in Y)$

by *simp*

lemma *sand-elim* :

$x \in (X \cap Y) \longleftrightarrow (x \in X \wedge x \in Y)$

by *simp*

lemma *sfalse-elim* :

$\sigma \notin SFalse$

by (*simp add: sfalse-def*)

lemma *strue-elim* :

$\sigma \in STrue$

by (*simp add: strue-def*)

lemma *sempty-elim* :

$\sigma \in SEmpty \longleftrightarrow (nlength\ \sigma = 0)$

by (*simp add: sempty-def*)

lemma *smore-elim* :

$\sigma \in SMore \longleftrightarrow$

$(nlength\ \sigma > 0)$

by (*simp add: sempty-elim smore-def*)

lemma *fusion-iff*:

$\sigma \in X \cdot Y \longleftrightarrow$

$($
 $(\exists\ \sigma1\ \sigma2. \sigma = nfuse\ \sigma1\ \sigma2 \wedge nfinite\ \sigma1 \wedge$
 $(\sigma1 \in X) \wedge (\sigma2 \in Y) \wedge (nlast\ \sigma1 = nfirst\ \sigma2))$
 $\vee (\neg\ nfinite\ \sigma \wedge \sigma \in X))$

by (*unfold fusion-def auto*)

lemma *fusion-iff-1*:

$\sigma \in X \cdot Y \longleftrightarrow$

$((\exists\ n \leq nlength\ \sigma. ((ntaken\ n\ \sigma) \in X) \wedge ((ndropn\ n\ \sigma) \in Y))$
 $\vee (\neg\ nfinite\ \sigma \wedge \sigma \in X))$

```

)
using fusion-iff[of  $\sigma$   $X$   $Y$ ] nfuse-ntaken-ndropn[of -  $\sigma$ ]
by (simp add: chop-nfuse-2)

lemma smore-fusion-smore :
   $\sigma \in (SMore \cdot SMore) \longleftrightarrow (enat\ 1) < nlength\ \sigma$ 
using fusion-iff-1[of  $\sigma$   $SMore$   $SMore$ ]
proof (auto simp add: smore-elim)
  show  $\bigwedge n. na.$ 
     $\sigma \in SMore \cdot SMore \implies$ 
     $enat\ n \leq nlength\ \sigma \implies$ 
     $enat\ n \neq 0 \implies$ 
     $nlength\ \sigma - enat\ n \neq 0 \implies enat\ na \leq nlength\ \sigma \implies enat\ na \neq 0 \implies$ 
     $nlength\ \sigma \neq 0 \implies nlength\ \sigma - enat\ na \neq 0 \implies$ 
     $enat\ (Suc\ 0) < nlength\ \sigma$ 
  by (metis One-nat-def antisym-conv3 enat-ord-code(4) idiff-self ileI1 le-less-trans less-le
    not-iless0 one-eSuc one-enat-def)
  show  $\bigwedge n. \sigma \in SMore \cdot SMore \implies$ 
     $enat\ n \leq nlength\ \sigma \implies$ 
     $enat\ n \neq 0 \implies nlength\ \sigma - enat\ n \neq 0 \implies$ 
     $\neg nfinite\ \sigma \implies$ 
     $nlength\ \sigma \neq 0 \implies$ 
     $enat\ (Suc\ 0) < nlength\ \sigma$ 
  by (metis Suc-ile-eq i0-less nlength-eq-enat-nfiniteD order.not-eq-order-implies-strict
    zero-enat-def)
  show  $\bigwedge n. \sigma \in SMore \cdot SMore \implies$ 
     $\neg nfinite\ \sigma \implies$ 
     $enat\ n \leq nlength\ \sigma \implies$ 
     $enat\ n \neq 0 \implies$ 
     $nlength\ \sigma \neq 0 \implies$ 
     $nlength\ \sigma - enat\ n \neq 0 \implies$ 
     $enat\ (Suc\ 0) < nlength\ \sigma$ 
  by (metis Suc-ile-eq i0-less nlength-eq-enat-nfiniteD order.not-eq-order-implies-strict zero-enat-def)
  show  $\sigma \in SMore \cdot SMore \implies \neg nfinite\ \sigma \implies nlength\ \sigma \neq 0 \implies enat\ (Suc\ 0) < nlength\ \sigma$ 
  by (metis One-nat-def ileI1 linorder-cases nlength-eq-enat-nfiniteD not-less-zero one-eSuc
    one-enat-def order.not-eq-order-implies-strict)
  show  $\sigma \notin SMore \cdot SMore \implies$ 
     $enat\ (Suc\ 0) < nlength\ \sigma \implies$ 
     $\forall n. enat\ n \leq nlength\ \sigma \longrightarrow enat\ n = 0 \vee nlength\ \sigma = 0 \vee nlength\ \sigma - enat\ n = 0 \implies$ 
     $nfinite\ \sigma \implies$ 
    False
proof -
  assume a1:  $enat\ (Suc\ 0) < nlength\ \sigma$ 
  assume nfinite  $\sigma$ 
  assume a2:  $\forall n. enat\ n \leq nlength\ \sigma \longrightarrow enat\ n = 0 \vee nlength\ \sigma = 0 \vee nlength\ \sigma - enat\ n = 0$ 
  have  $enat\ (Suc\ 0) = 1$ 
  using One-nat-def one-enat-def by presburger
  then have f3:  $enat\ 1 < nlength\ \sigma$ 
  using a1 one-enat-def by presburger
  have f4:  $enat\ 1 \leq enat\ 1$ 

```

```

  by blast
have enat 1  $\neq$   $\infty$ 
  by blast
then show False
  using f4 f3 a2
  by (metis (no-types) dual-order.strict-iff-order enat-diff-cancel-left
      gr-implies-not-zero idiff-self one-enat-def zero-neq-one)
qed
qed

lemma sskip-elim :
   $\sigma \in SSkip \longleftrightarrow$ 
  (nlength  $\sigma$  = 1)
using sskip-def smore-fusion-smore
by (metis One-nat-def Suc-ile-eq less-numeral-extra(4) one-enat-def order.not-eq-order-implies-strict
    sempty-elim smore-def smore-elim snot-elim sor-elim zero-enat-def zero-one-enat-neq(1))

lemma sinfinite-elim:
   $\sigma \in SInf \longleftrightarrow$ 
  ( $\neg$  nfinite  $\sigma$ )
by (simp add: fusion-iff-1 sfalse-elim sinf-def strue-elim)

lemma sfinite-elim:
   $\sigma \in SFinite \longleftrightarrow$ 
  (nfinite  $\sigma$ )
by (simp add: sfinite-def sinfinite-elim )

lemma ffusion-iff:
   $\sigma \in (X \cap SFinite) \cdot Y \longleftrightarrow$ 
  (
    ( $\exists \sigma 1 \sigma 2. \sigma = nfuse \sigma 1 \sigma 2 \wedge nfinite \sigma 1 \wedge$ 
      ( $\sigma 1 \in X$ )  $\wedge$  ( $\sigma 2 \in Y$ )  $\wedge$  ( $nlast \sigma 1 = nfirst \sigma 2$ ))
  )
unfolding fusion-def
by (simp add: sfinite-elim) blast

lemma ffusion-iff-1:
   $\sigma \in (X \cap SFinite) \cdot Y \longleftrightarrow$ 
  (( $(\exists n \leq nlength \sigma. (ntaken n \sigma) \in X) \wedge (ndropn n \sigma) \in Y$ )
  )
)
using fusion-iff-1[of  $\sigma$   $X \cap SFinite$   $Y$  ]
by (meson nfinite-ntaken sand-elim sfinite-elim)

lemma sfmore-elim:
   $\sigma \in SFMore \longleftrightarrow$ 
  (nfinite  $\sigma \wedge 0 < nlength \sigma$ )
by (simp add: sfinite-elim sfmore-def smore-elim )

lemma spower-elim-zero :
```

$\sigma \in \text{SPower } X \ 0 \longleftrightarrow \sigma \in \text{SEmpty}$
by *simp*

lemma *spower-elim-suc* :
 $\sigma \in \text{SPower } X \ (\text{Suc } n) \longleftrightarrow \sigma \in (X \cap \text{SFinite}) \cdot (\text{SPower } X \ n)$
by *simp*

lemma *spower-elim-suc-1* :
 $\sigma \in (X \cap \text{SFinite}) \cdot (\text{SPower } X \ n) \longleftrightarrow$
 $(\exists \sigma 1 \ \sigma 2. \sigma = \text{nfuse } \sigma 1 \ \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge$
 $\sigma 1 \in X \wedge \sigma 2 \in (\text{SPower } X \ n) \wedge$
 $\text{nlast } \sigma 1 = \text{nfirst } \sigma 2)$
by (*simp add: ffusion-iff*)

lemma *ffusionimp*:
assumes $Y0 \subseteq Y1$
shows $\sigma \in (X \cap \text{SFinite}) \cdot Y0 \longrightarrow \sigma \in (X \cap \text{SFinite}) \cdot Y1$
using *assms unfolding ffusion-iff-1* **by** (*auto*)

lemma *spower-finite*:
 $(\text{SPower } X \ n) \subseteq \text{SFinite}$
proof (*induct n*)
case 0
then show ?*case*
by (*metis nlength-eq-enat-nfiniteD empty-elim sfinite-elim spower.simps(1) subsetI zero-enat-def*)
next
case (*Suc n*)
then show ?*case*
proof –
have 1: $(\text{SPower } X \ (\text{Suc } n)) = (X \cap \text{SFinite}) \cdot (\text{SPower } X \ n)$
by *simp*
have 2: $\text{SPower } X \ n \subseteq \text{SFinite}$
using *Suc.hyps* **by** *blast*
have 3: $(X \cap \text{SFinite}) \cdot \text{SFinite} \subseteq \text{SFinite}$
using *ffusion-iff-1[of - X SFinite]*
by (*simp add: sfinite-elim*)
 $(\text{metis ComplI sfinite-def sinfinite-elim subsetI})$
have 4: $(X \cap \text{SFinite}) \cdot (\text{SPower } X \ n) \subseteq (X \cap \text{SFinite}) \cdot \text{SFinite}$
using *Suc.hyps ffusionimp* **by** *blast*
show ?*thesis*
using 3 4 **by** *auto*
qed
qed

lemma *sfpowerstar-elim*:
 $\sigma \in \text{SFPowerstar } X \longleftrightarrow (\exists n. \sigma \in \text{SPower } X \ n)$
by (*simp add: sfpowerstar-def*)

lemma *sfpowerstar-elim-1*:

$(\exists n. \sigma \in SPower\ X\ n) \longleftrightarrow (\sigma \in SPower\ X\ 0 \vee (\exists n. \sigma \in SPower\ X\ (Suc\ n)))$
by (*metis not0-implies-Suc*)

lemma *sfpowerstar-suc*:

$(\exists n. \sigma \in SPower\ X\ (Suc\ n)) \longleftrightarrow (\exists n. \sigma \in (X \cap SFinite) \cdot (SPower\ X\ n))$
by *simp*

lemma *sfpowerstar-suc-1*:

$(\exists n. \sigma \in (X \cap SFinite) \cdot (SPower\ X\ n)) \longleftrightarrow \sigma \in (X \cap SFinite) \cdot (SFPowerstar\ X)$
unfolding *fusion-iff*
by (*cases* σ) (*auto simp add: sfpowerstar-elim*)

lemma *sfpowerstar-equiv-sem*:

$\sigma \in SFPowerstar\ X \longleftrightarrow (\sigma \in SEmpty \vee \sigma \in (X \cap SFinite) \cdot (SFPowerstar\ X))$
by (*simp add: sfpowerstar-elim sfpowerstar-elim-1 sfpowerstar-suc-1*)

lemma *sfpowerstar-equiv*:

$SFPowerstar\ X = SEmpty \cup (X \cap SFinite) \cdot (SFPowerstar\ X)$
using *sfpowerstar-equiv-sem* **by** *blast*

lemma *sfpowerstar-equiv-1*:

$(\bigcup n. SPower\ X\ n) = SEmpty \cup (X \cap SFinite) \cdot (\bigcup n. SPower\ X\ n)$
using *sfpowerstar-equiv* **by** (*simp add: sfpowerstar-def*)

2.15.3 Algebraic Laws

Commutative Additive Monoid

lemma *UnionCommute*:

$(X::'a\ iintervals) \cup Y = Y \cup X$
by (*simp add: Un-commute*)

lemma *UnionSFalse*:

$X \cup SFalse = X$
by (*simp add: sfalse-def*)

lemma *UnionAssoc*:

$(X::'a\ iintervals) \cup (Y \cup Z) = (X \cup Y) \cup Z$
by (*simp add: sup-assoc*)

Boolean algebra

lemma *Huntington*:

$(X::'a\ iintervals) = \neg(\neg X \cup \neg Y) \cup \neg(\neg X \cup Y)$
by *auto*

lemma *Morgan*:

$(X::'a\ iintervals) \cap Y = \neg(\neg X \cup \neg Y)$
by *auto*

— identities

lemma *STrueTop*:

$$STrue = X \cup -X$$

by (*simp add: strue-def*)

lemma *SFalseBottom*:

$$SFalse = X \cap -X$$

by (*simp add: sfalse-def*)

multiplicative monoid

lemma *FusionSEmptyLsem*:

$$\sigma \in SEmpty \cdot X \longleftrightarrow \sigma \in X$$

using *fusion-iff-1*[*of* σ *SEmpty* *X*]

by (*auto simp add: fusion-iff-1 empty-elim nlength-eq-enat-nfiniteD zero-enat-def*)
(metis le-numeral-extra(3) min.bounded-iff min-enat-simps(3) ndropn-0 zero-enat-def,
metis le-numeral-extra(3) min.bounded-iff min-enat-simps(3) ndropn-0 zero-enat-def)

lemma *FusionSEmptyL* :

$$SEmpty \cdot X = X$$

using *set-eqI*[*of* *SEmpty*·*X* *X*] *FusionSEmptyLsem*

by *auto*

lemma *FusionSEmptyRsem* :

$$\sigma \in X \cdot SEmpty \longleftrightarrow \sigma \in X$$

using *fusion-iff-1*[*of* σ *X* *SEmpty*]

by (*auto simp add: fusion-iff-1 empty-elim*)
(metis is-NNil-ndropn le-numeral-extra(3) ndropn-0 ndropn-nlength ntaken-all zero-enat-def,
metis add.right-neutral le-iff-add ndropn-all ndropn-nlength nfinite-nlength-enat nlength-NNil
ntaken-all)

lemma *FusionSEmptyR* :

$$X \cdot SEmpty = X$$

using *set-eqI*[*of* *X*·*SEmpty* *X*] *FusionSEmptyRsem* **by** *auto*

lemma *FusionAssocA*:

assumes $x \in X \cdot (Y \cdot Z)$

shows $x \in (X \cdot Y) \cdot Z$

proof –

have 1: $(\exists \sigma 1 \sigma 2. x = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge \sigma 1 \in X \wedge \sigma 2 \in Y \cdot Z \wedge \text{nlast } \sigma 1 = \text{nfirst } \sigma 2)$
 $\vee (\neg \text{nfinite } x \wedge x \in X)$

using *assms fusion-iff*[*of* x *X* *Y* · *Z*] **by** *auto*

have 2: $\neg \text{nfinite } x \wedge x \in X \implies x \in (X \cdot Y) \cdot Z$

by (*simp add: fusion-iff*)

have 3: $(\exists \sigma 1 \sigma 2. x = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge \sigma 1 \in X \wedge \sigma 2 \in Y \cdot Z \wedge \text{nlast } \sigma 1 = \text{nfirst } \sigma 2) \implies$
 $x \in (X \cdot Y) \cdot Z$

proof –

assume *a0*: $(\exists \sigma 1 \sigma 2. x = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge \sigma 1 \in X \wedge \sigma 2 \in Y \cdot Z \wedge \text{nlast } \sigma 1 = \text{nfirst } \sigma 2)$

show $x \in (X \cdot Y) \cdot Z$

proof –

```

obtain  $\sigma 1 \sigma 2$  where 5:
   $x = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge \sigma 1 \in X \wedge \sigma 2 \in Y \cdot Z \wedge \text{nlast } \sigma 1 = \text{nfirst } \sigma 2$ 
  using a0 by auto
have 6:  $(\exists \sigma 3 \sigma 4. \sigma 2 = \text{nfuse } \sigma 3 \sigma 4 \wedge \text{nfinite } \sigma 3 \wedge \sigma 3 \in Y \wedge \sigma 4 \in Z \wedge \text{nlast } \sigma 3 = \text{nfirst } \sigma 4) \vee$ 
   $(\neg \text{nfinite } \sigma 2 \wedge \sigma 2 \in Y)$ 
  using fusion-iff[of  $\sigma 2 \ Y \ Z$ ] 5 by simp
have 7:  $(\neg \text{nfinite } \sigma 2 \wedge \sigma 2 \in Y) \implies x \in (X \cdot Y) \cdot Z$ 
  by (metis 5 fusion-iff ndropn-nfuse nfinite-ndropn)
have 8:  $(\exists \sigma 3 \sigma 4. \sigma 2 = \text{nfuse } \sigma 3 \sigma 4 \wedge \text{nfinite } \sigma 3 \wedge \sigma 3 \in Y \wedge \sigma 4 \in Z \wedge \text{nlast } \sigma 3 = \text{nfirst } \sigma 4) \implies$ 
   $x \in (X \cdot Y) \cdot Z$ 
proof –
assume a1:  $(\exists \sigma 3 \sigma 4. \sigma 2 = \text{nfuse } \sigma 3 \sigma 4 \wedge \text{nfinite } \sigma 3 \wedge \sigma 3 \in Y \wedge \sigma 4 \in Z \wedge \text{nlast } \sigma 3 = \text{nfirst } \sigma 4)$ 
show  $x \in (X \cdot Y) \cdot Z$ 
  proof –
  obtain  $\sigma 3 \sigma 4$  where 10:
     $\sigma 2 = \text{nfuse } \sigma 3 \sigma 4 \wedge \text{nfinite } \sigma 3 \wedge \sigma 3 \in Y \wedge \sigma 4 \in Z \wedge \text{nlast } \sigma 3 = \text{nfirst } \sigma 4$ 
    using a1 by auto
  have 11:  $x = \text{nfuse } \sigma 1 (\text{nfuse } \sigma 3 \sigma 4)$ 
    using 10 5 by blast
  have 12:  $x = \text{nfuse } (\text{nfuse } \sigma 1 \sigma 3) \sigma 4$ 
    by (simp add: 10 5 nfirst-nfuse nfuseassoc)
  have 13:  $(\text{nfuse } \sigma 1 \sigma 3) \in X \cdot Y$ 
    using fusion-iff[of  $(\text{nfuse } \sigma 1 \sigma 3) \ X \ Y$ ]
    using 10 5 nfirst-nfuse by fastforce
  show ?thesis using fusion-iff[of  $x \ X \cdot Y \ Z$ ]
    using 10 12 13 5
    by (metis nfuse-nlength nfinite-nlength-enat nfirst-nfuse nlength-eq-enat-nfiniteD
      plus-enat-simps(1) nlast-nfuse)
  qed
qed
show ?thesis
  using 6 7 8 by blast
qed
qed
show ?thesis
  using 1 2 3 by blast
qed

```

lemma *FusionAssocB*:

assumes $x \in (X \cdot Y) \cdot Z$

shows $x \in X \cdot (Y \cdot Z)$

proof –

have 1: $(\exists \sigma 1 \sigma 2. x = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge \sigma 1 \in X \cdot Y \wedge \sigma 2 \in Z \wedge \text{nlast } \sigma 1 = \text{nfirst } \sigma 2) \vee$
 $(\neg \text{nfinite } x \wedge x \in X \cdot Y)$

using *assms* *fusion-iff*[*of* $x \ X \cdot Y \ Z$] **by** *auto*

have 2: $(\neg \text{nfinite } x \wedge x \in X \cdot Y) \implies x \in X \cdot (Y \cdot Z)$

by (*metis* *fusion-iff-1* *nfinite-ndropn-b*)

have 3: $(\exists \sigma 1 \sigma 2. x = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge \sigma 1 \in X \cdot Y \wedge \sigma 2 \in Z \wedge \text{nlast } \sigma 1 = \text{nfirst } \sigma 2) \implies$
 $x \in X \cdot (Y \cdot Z)$

proof –

```

assume  $a0$ :  $(\exists \sigma 1 \sigma 2. x = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge \sigma 1 \in X \cdot Y \wedge \sigma 2 \in Z \wedge \text{nlast } \sigma 1 = \text{nfirst } \sigma 2)$ 
show  $x \in X \cdot (Y \cdot Z)$ 
proof –
  obtain  $\sigma 1 \sigma 2$  where 4:
     $x = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge \sigma 1 \in X \cdot Y \wedge \sigma 2 \in Z \wedge \text{nlast } \sigma 1 = \text{nfirst } \sigma 2$ 
    using  $a0$  by auto
  have 5:  $\exists \sigma 3 \sigma 4. \sigma 1 = \text{nfuse } \sigma 3 \sigma 4 \wedge \text{nfinite } \sigma 3 \wedge \sigma 3 \in X \wedge \sigma 4 \in Y \wedge \text{nlast } \sigma 3 = \text{nfirst } \sigma 4$ 
    using 4 fusion-iff[of  $\sigma 1 X Y$ ]
    using nfuse-nappend by fastforce
  obtain  $\sigma 3 \sigma 4$  where 6:
     $\sigma 1 = \text{nfuse } \sigma 3 \sigma 4 \wedge \text{nfinite } \sigma 3 \wedge \sigma 3 \in X \wedge \sigma 4 \in Y \wedge \text{nlast } \sigma 3 = \text{nfirst } \sigma 4$ 
    using 5 by auto
  have 7:  $x = \text{nfuse } (\text{nfuse } \sigma 3 \sigma 4) \sigma 2$ 
    by (simp add: 4 6)
  have 8:  $x = \text{nfuse } \sigma 3 (\text{nfuse } \sigma 4 \sigma 2)$ 
    using 4 6 nfuseassoc by metis
  have 9:  $(\text{nfuse } \sigma 4 \sigma 2) \in Y \cdot Z$ 
    using fusion-iff[of  $(\text{nfuse } \sigma 4 \sigma 2) Y Z$ ]
    by (metis 4 6 nfuse-nappend nlast-nfuse)
  have 10:  $\text{nlast } \sigma 3 = \text{nfirst } (\text{nfuse } \sigma 4 \sigma 2)$ 
    using 4 6 nfirst-nfuse nlast-nfuse by fastforce
show ?thesis
  using fusion-iff[of  $x X Y \cdot Z$ ]
  using 10 6 8 9 by auto
qed
qed
show ?thesis
using 1 2 3 by blast
qed

```

```

lemma FusionAssoc :
   $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$ 
using set-eqI[of  $X \cdot (Y \cdot Z) (X \cdot Y) \cdot Z$ ]
FusionAssocA FusionAssocB by blast

```

```

lemma FusionAssoc1:
   $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$ 
by (simp add: FusionAssoc)

```

— left and right distributivity

```

lemma FusionUnionDistLsem:
   $x \in (X \cup Y) \cdot Z \longleftrightarrow x \in (X \cdot Z) \cup (Y \cdot Z)$ 
by (auto simp add: fusion-iff)

```

```

lemma FusionUnionDistL:
   $(X \cup Y) \cdot Z = (X \cdot Z) \cup (Y \cdot Z)$ 
using FusionUnionDistLsem[of -  $X Y Z$ ] by fastforce

```


lemma *FusionUnionDistRsem*:

$$x \in X \cdot (Y \cup Z) \longleftrightarrow x \in (X \cdot Y) \cup (X \cdot Z)$$

by (*auto simp add: fusion-iff*)

lemma *FusionUnionDistR*:

$$X \cdot (Y \cup Z) = (X \cdot Y) \cup (X \cdot Z)$$

using *FusionUnionDistRsem*[*of - X Y Z*] **by** *fastforce*

— left and right annihilation

lemma *SFalseFusion*:

$$SFalse \cdot X = SFalse$$

by (*simp add: fusion-def sfalse-def*)

lemma *FusionSFalse*:

$$X \cdot SFalse = (X \cap SInf)$$

by (*auto simp add: fusion-def sfalse-def sinfinite-elim*)

— idempotency

lemma *UnionIdem*:

$$(X :: 'a \text{ intervals}) \cup X = X$$

by *simp*

Subsumption order

lemma *Subsumption*:

$$((X :: 'a \text{ intervals}) \subseteq Y) = (X \cup Y = Y)$$

by *auto*

Helper lemmas

lemma *FusionRuleR*:

assumes $X \subseteq Y$

shows $Z \cdot X \subseteq Z \cdot Y$

using *assms FusionUnionDistR* **by** (*metis Subsumption*)

lemma *FusionRuleL*:

assumes $X \subseteq Y$

shows $X \cdot Z \subseteq Y \cdot Z$

using *assms* **by** (*metis FusionUnionDistL subset-Un-eq*)

lemma *spower-commutes*:

$$(X \cap SFinite) \cdot (SPower X n) = (SPower X n) \cdot (X \cap SFinite)$$

proof (*induct n*)

case 0

then show *?case* **by** (*simp add: FusionSEmptyL FusionSEmptyR*)

next

case (*Suc n*)

then show *?case* **by** (*simp add: FusionAssoc*)

qed

```

lemma fusion-inductl:
  assumes  $Y \cup X \cdot Z \subseteq Z$ 
  shows  $(SPower\ X\ n) \cdot Y \subseteq Z$ 
  using assms
  proof (induct n)
  case 0
  then show ?case by (simp add: FusionSEmptyL)
  next
  case (Suc n)
  then show ?case
  proof -
    have f1:  $X \cdot (SPower\ X\ n \cdot Y) \subseteq Z$ 
      using FusionRuleR Suc.hyps assms by blast
    have  $X \cap SFinite \subseteq X$ 
      by blast
    then show ?thesis
      using f1 by (metis (no-types) FusionAssoc FusionRuleL order-trans pwr-Suc)
  qed
qed

```

```

lemma fusion-inductl-finite:
  assumes  $Y \cup (X \cap SFinite) \cdot Z \subseteq Z$ 
  shows  $(SPower\ X\ n) \cdot Y \subseteq Z$ 
  using assms
  proof (induct n)
  case 0
  then show ?case
  by (simp add: FusionSEmptyL)
  next
  case (Suc n)
  then show ?case
  proof -
    have f1:  $SPower\ X\ n \cdot Y \subseteq Z$ 
      using Suc.hyps assms by blast
    then have  $SPower\ X\ n \cdot Y \cup Z = Z$ 
      by blast
    then show ?thesis
      using f1
      by (metis FusionAssoc1 FusionUnionDistR assms le-supE pwr-Suc)
  qed
qed

```

```

lemma fusion-inductl-fmore:
  assumes  $Y \cup (X \cap SFMore) \cdot Z \subseteq Z$ 
  shows  $(SPower\ X\ n) \cdot Y \subseteq Z$ 
  using assms
  proof (induct n)
  case 0
  then show ?case

```

```

by (simp add: FusionSEmptyL)
next
case (Suc n)
then show ?case
proof -
  have 1:  $SPower\ X\ (Suc\ n) \cdot Y = ((X \cap SFinite) \cdot (SPower\ X\ n)) \cdot Y$ 
    by simp
  have 2:  $((X \cap SFinite) \cdot (SPower\ X\ n)) \cdot Y = (X \cap SFinite) \cdot ((SPower\ X\ n) \cdot Y)$ 
    by (simp add: FusionAssoc)
  have 3:  $(X \cap SFinite) \cdot ((SPower\ X\ n) \cdot Y) \subseteq (X \cap SFinite) \cdot Z$ 
    using FusionRuleR Suc.hyps assms by blast
  have 31:  $X \cap SFinite = ((X \cap SFMore) \cup ((X \cap SEmpty) \cap SFinite))$ 
    by (simp add: sfmore-def smore-def) blast
  have 4:  $(X \cap SFinite) \cdot Z = ((X \cap SFMore) \cdot Z \cup ((X \cap SEmpty) \cap SFinite) \cdot Z)$ 
    by (simp add: 31 FusionUnionDistL)
  have 5:  $(X \cap SFMore) \cdot Z \subseteq Z$ 
    using assms by auto
  have 6:  $((X \cap SEmpty) \cap SFinite) \cdot Z \subseteq Z$ 
    by (metis FusionRuleL FusionSEmptyL inf-assoc inf-commute inf-le1)
  show ?thesis
    using 1 2 3 4 5 6 by blast
qed
qed

lemma fusion-inductl-inf:
  assumes  $Y \cup X \cdot Z \subseteq Z$ 
  shows  $((SPower\ X\ n) \cdot (X \cap SInf)) \cdot Y \subseteq Z$ 
  using assms
  proof (induct n)
  case 0
  then show ?case
  by (metis FusionAssoc FusionRuleL FusionRuleR FusionSEmptyL FusionSFalse le-supE order-trans
    pwr-0 sfalse-elim subsetI)
  next
  case (Suc n)
  then show ?case
  proof -
    have f2:  $Z = Z \cup (Y \cup X \cdot Z)$ 
      using assms by blast
    have  $SPower\ X\ n \cdot (X \cap SInf) \cdot Y \subseteq Z$ 
      using Suc(1) assms by blast
    then have  $Z = Z \cup SPower\ X\ n \cdot (X \cap SInf) \cdot Y$ 
      by blast
    then have f3:  $(X \cup X \cap SFinite) \cdot (Z \cup SPower\ X\ n \cdot (X \cap SInf) \cdot Y) = X \cdot Z$ 
      by force
    have  $SPower\ X\ (Suc\ n) \cdot (X \cap SInf) \cdot Y = X \cap SFinite \cdot (SPower\ X\ n \cdot (X \cap SInf) \cdot Y)$ 
      by (simp add: FusionAssoc)
    then have  $SPower\ X\ (Suc\ n) \cdot (X \cap SInf) \cdot Y \cup X \cap SFinite \cdot (Z \cup SPower\ X\ n \cdot (X \cap SInf) \cdot Y) =$ 
       $X \cap SFinite \cdot (Z \cup SPower\ X\ n \cdot (X \cap SInf) \cdot Y)$ 
      using FusionUnionDistR by blast
  

```

then have $SPower\ X\ (Suc\ n) \cdot (X \cap SInf) \cdot Y \cup X \cdot Z = X \cdot Z$
using $f3\ FusionUnionDistL$ **by** $blast$
then show $?thesis$
using $f2$ **by** $blast$
qed
qed

lemma *fusion-inductl-infmore*:
assumes $Y \cup X \cdot Z \subseteq Z$
shows $((SPower\ X\ n) \cdot ((X \cap SMore) \cap SInf)) \cdot Y \subseteq Z$
using $assms$
proof (*induct n*)
case 0
then show $?case$
by (*metis (no-types, lifting) Diff-Compl Int-assoc Un-Diff-Int compl-inf double-compl fusion-inductl-inf inf.absorb-iff2 pwr-0 sfinite-def smore-def spower-finite sup-left-idem*)
next
case ($Suc\ n$)
then show $?case$
proof –
have $f1: \bigwedge n. SPower\ X\ n \cdot (X \cap SInf) \cdot Y \subseteq Z$
by (*meson assms fusion-inductl-inf*)
have $X \cap SMore \cap SInf \subseteq X \cap SInf$
by $blast$
then show $?thesis$
using $f1$ **by** (*metis (no-types) FusionRuleL FusionRuleR inf.orderE le-inf-iff*)
qed
qed

lemma *fusion-inductl-more*:
assumes $Y \cup X \cdot Z \subseteq Z$
shows $(SPower\ (X \cap SMore)\ n) \cdot Y \subseteq Z$
using $assms$
proof (*induct n*)
case 0
then show $?case$ **by** (*simp add: FusionSEmptyL*)
next
case ($Suc\ n$)
then show $?case$
by (*metis FusionUnionDistL fusion-inductl sup.boundedE sup.boundedI sup-inf-absorb*)
qed

lemma *fusion-inductr*:
assumes $Y \cup Z \cdot X \subseteq Z$
shows $Y \cdot (SPower\ X\ n) \subseteq Z$
using $assms$
proof (*induct n*)
case 0
then show $?case$ **by** (*simp add: FusionSEmptyR*)
next

```

case (Suc n)
then show ?case
proof –
have f1:  $Z \cdot X \subseteq Z$ 
  using assms by auto
have  $\forall S. Y \cdot (SPower\ X\ n \cdot S) \subseteq Z \cdot S$ 
  using FusionAssoc FusionRuleL Suc.hyps assms by blast
then show ?thesis
  using f1 by (metis (no-types) FusionRuleR Int-iff order-trans pwr-Suc spower-commutes subsetI)
qed
qed

```

```

lemma SInfSFinite:
 $X = (X \cap SFinite) \cup (X \cap SInf)$ 
using sfinite-def by auto

```

```

lemma fusion-inductr-inf:
assumes  $Y \cup Z \cdot X \subseteq Z$ 
shows  $Y \cdot ((SPower\ X\ n) \cdot (X \cap SInf)) \subseteq Z$ 
proof –
have 1:  $Y \cdot (SPower\ X\ n) \subseteq Z$ 
  using assms fusion-inductr by blast
show ?thesis
proof –
have f4:  $Y \cdot SPower\ X\ n = Y \cdot SPower\ X\ n \cap Z$ 
  using 1 by blast
have  $Y \cdot SPower\ X\ n \cup Z = Z$ 
  by (metis 1 subset-Un-eq)
then have f5:  $Z \cdot (X \cap SInf) = Z \cdot (X \cap SInf) \cup Y \cdot SPower\ X\ n \cap Z \cdot (X \cap SInf)$ 
  using f4 by (metis (no-types) FusionUnionDistL sup commute)
have  $Y \cdot SPower\ X\ n = Y \cdot SPower\ X\ n \cap Z$ 
  using f4 by auto
then have  $Y \cdot (SPower\ X\ n \cdot (X \cap SInf)) = Y \cdot SPower\ X\ n \cap Z \cdot (X \cap SInf)$ 
  by (simp add: FusionAssoc1)
then show ?thesis
  proof –
have f1:  $Y \cup Z \cdot X \cup Z = Z$ 
  by (meson Subsumption assms)
have f2:  $Y \cdot SPower\ X\ n \cap Z \cdot X \cup Z \cdot X = Z \cdot X$ 
  by (metis (no-types) FusionUnionDistL  $\langle Y \cdot SPower\ X\ n \cup Z = Z \rangle$  f4)
have  $Y \cdot SPower\ X\ n \cap Z \cdot X \cup Y \cdot SPower\ X\ n \cap Z \cdot (X \cap SInf) = Y \cdot SPower\ X\ n \cap Z \cdot X$ 
  by (metis (no-types) FusionUnionDistR sup-inf-absorb)
then show ?thesis
  using f2 f1  $\langle Y \cdot (SPower\ X\ n \cdot (X \cap SInf)) = Y \cdot SPower\ X\ n \cap Z \cdot (X \cap SInf) \rangle$  by auto
qed
qed
qed

```

```

lemma fusion-inductr-more:
assumes  $Y \cup Z \cdot X \subseteq Z$ 

```

shows $Y \cdot (SPower (X \cap SMore) \ n) \subseteq Z$
using *assms*
proof (*induct n*)
case 0
then show ?*case* **by** (*simp add: FusionSEmptyR*)
next
case (*Suc n*)
then show ?*case*
by (*metis FusionUnionDistR fusion-inductr le-supE sup.boundedI sup-inf-absorb*)
qed

lemma *FusionUnionSPowersem*:
 $\sigma \in Y \cdot (\bigcup n. (SPower X \ n) \cdot Z) \longleftrightarrow \sigma \in (\bigcup n. Y \cdot ((SPower X \ n) \cdot Z))$
by (*simp add: fusion-iff*) **blast**

lemma *FusionUnionSPower*:
 $Y \cdot (\bigcup n. (SPower X \ n) \cdot Z) = (\bigcup n. Y \cdot ((SPower X \ n) \cdot Z))$
using *FusionUnionSPowersem* **by** *blast*

lemma *FusionUnionSPower1*:
 $Y \cdot (\bigcup n. (SPower X \ n)) = (\bigcup n. Y \cdot ((SPower X \ n)))$
using *FusionUnionSPower[of Y X SEmpty]*
by (*metis (no-types, lifting) FusionSEmptyR Sup.SUP-cong*)

lemma *UnionFusionSPowersem*:
 $\sigma \in (\bigcup n. Z \cdot (SPower X \ n)) \cdot Y \longleftrightarrow \sigma \in (\bigcup n. (Z \cdot (SPower X \ n)) \cdot Y)$
by (*simp add: fusion-iff*) **blast**

lemma *UnionFusionSPower*:
 $(\bigcup n. Z \cdot (SPower X \ n)) \cdot Y = (\bigcup n. (Z \cdot (SPower X \ n)) \cdot Y)$
using *UnionFusionSPowersem* **by** *blast*

lemma *UnionFusionSPower1*:
 $(\bigcup n. (SPower X \ n)) \cdot Y = (\bigcup n. ((SPower X \ n)) \cdot Y)$
using *UnionFusionSPower[of SEmpty X Y]*
by (*metis (no-types, lifting) FusionSEmptyL Sup.SUP-cong*)

lemma *spowercommute*:
 $(X \cap SFinite) \cdot (\bigcup n. (SPower X \ n)) = (\bigcup n. (SPower X \ n) \cdot (X \cap SFinite))$
by (*metis (no-types, lifting) FusionUnionSPower1 Sup.SUP-cong spower-commutes*)

lemma *sstar-contl*:
 $Y \cdot (SStar X) = (\bigcup n. Y \cdot (SPower (X \cap SMore) \ n) \cdot (SEmpty \cup X \cap SMore \cap SInf))$
proof –
have 1: $Y \cdot (SStar X) = Y \cdot ((\bigcup n. (SPower (X \cap SMore) \ n))) \cdot (SEmpty \cup X \cap SMore \cap SInf)$
by (*simp add: sstar-def spowerstar-def sfpowerstar-def*)
have 2: $Y \cdot ((\bigcup n. (SPower (X \cap SMore) \ n))) \cdot (SEmpty \cup X \cap SMore \cap SInf) =$
 $(Y \cdot (\bigcup n. (SPower (X \cap SMore) \ n))) \cdot (SEmpty \cup X \cap SMore \cap SInf)$
using *FusionAssoc* **by** *blast*
have 3: $(Y \cdot (\bigcup n. (SPower (X \cap SMore) \ n))) = (\bigcup n. Y \cdot (SPower (X \cap SMore) \ n))$

using *FusionUnionSPower1* **by** *blast*
have 4: $(Y \cdot (\bigcup n. (SPower (X \cap SMore) n))) \cdot (SEmpty \cup X \cap SMore \cap SInf) =$
 $(\bigcup n. Y \cdot (SPower (X \cap SMore) n)) \cdot (SEmpty \cup X \cap SMore \cap SInf)$
by (*simp add: 3*)
have 5: $(\bigcup n. Y \cdot (SPower (X \cap SMore) n)) \cdot (SEmpty \cup X \cap SMore \cap SInf) =$
 $(\bigcup n. (Y \cdot (SPower (X \cap SMore) n)) \cdot (SEmpty \cup X \cap SMore \cap SInf))$
using *UnionFusionSPower*[*of Y X \cap SMore (SEmpty \cup X \cap SMore \cap SInf)*]
by *auto*
show ?thesis
by (*simp add: 1 2 4 5*)
qed

lemma *sstar-contr*:

$$(SStar X) \cdot Y = (\bigcup n. ((SPower (X \cap SMore) n) \cdot (SEmpty \cup X \cap SMore \cap SInf))) \cdot Y$$

proof –

have 1: $(SStar X) \cdot Y = ((\bigcup n. ((SPower (X \cap SMore) n))) \cdot (SEmpty \cup X \cap SMore \cap SInf)) \cdot Y$
by (*simp add: sstar-def spowerstar-def sfpowerstar-def*)
have 2: $((\bigcup n. ((SPower (X \cap SMore) n))) \cdot (SEmpty \cup X \cap SMore \cap SInf)) \cdot Y =$
 $(\bigcup n. ((SPower (X \cap SMore) n)) \cdot ((SEmpty \cup X \cap SMore \cap SInf) \cdot Y))$
using *FusionAssoc* **by** *blast*
have 3: $(\bigcup n. ((SPower (X \cap SMore) n)) \cdot ((SEmpty \cup X \cap SMore \cap SInf) \cdot Y)) =$
 $(\bigcup n. ((SPower (X \cap SMore) n)) \cdot ((SEmpty \cup X \cap SMore \cap SInf) \cdot Y))$
using *UnionFusionSPower1*[*of X \cap SMore ((SEmpty \cup X \cap SMore \cap SInf) \cdot Y)*]
by *auto*
have 4: $(\bigcup n. ((SPower (X \cap SMore) n)) \cdot ((SEmpty \cup X \cap SMore \cap SInf) \cdot Y)) =$
 $(\bigcup n. ((SPower (X \cap SMore) n) \cdot (SEmpty \cup X \cap SMore \cap SInf)) \cdot Y)$
using *FusionAssoc* **by** *blast*
show ?thesis
by (*simp add: 1 2 3 4*)
qed

lemma *FPowerstarInductL*:

assumes $Y \cup (X \cap SFinite) \cdot Z \subseteq Z$

shows $(SFPowerstar X) \cdot Y \subseteq Z$

proof –

have 1: $(SFPowerstar X) \cdot Y = (\bigcup n. (SPower X n)) \cdot Y$
by (*simp add: sfpowerstar-def*)
have 2: $(\bigcup n. (SPower X n)) \cdot Y = (\bigcup n. (SPower X n) \cdot Y)$
using *UnionFusionSPower1* **by** *fastforce*
have 3: $\bigwedge n. (SPower X n) \cdot Y \subseteq Z$
using *assms fusion-inductl-finite* **by** *blast*
have 4: $(\bigcup n. (SPower X n) \cdot Y) \subseteq Z$
using 3 **by** *blast*
show ?thesis
using 1 2 4 **by** *blast*
qed

lemma *FPowerstarInductMoreL*:

assumes $Y \cup ((X \cap SMore) \cap SFinite) \cdot Z \subseteq Z$

shows $(SFPowerstar X) \cdot Y \subseteq Z$

proof –

have 1: $(SPowerstar\ X) \cdot Y = (\bigcup n. (SPower\ X\ n)) \cdot Y$
 by *(simp add: sfpowerstar-def)*
have 2: $(\bigcup n. (SPower\ X\ n)) \cdot Y = (\bigcup n. (SPower\ X\ n) \cdot Y)$
 using *UnionFusionSPower1* by *fastforce*
have 3: $\bigwedge n. (SPower\ X\ n) \cdot Y \subseteq Z$
 by *(metis Int-assoc Int-commute assms fusion-inductl-fmore sfmore-def)*
have 4: $(\bigcup n. (SPower\ X\ n) \cdot Y) \subseteq Z$
 using 3 by *blast*
show *?thesis*
 using 1 2 4 by *blast*
qed

lemma *PowerstarInductL*:

assumes $Y \cup X \cdot Z \subseteq Z$
shows $(SPowerstar\ X) \cdot Y \subseteq Z$

proof –

have 1: $(SPowerstar\ X) \cdot Y = ((\bigcup n. (SPower\ X\ n)) \cdot (SEmpty \cup X \cap SInf)) \cdot Y$
 by *(simp add: spowerstar-def sfpowerstar-def)*
have 2: $((\bigcup n. (SPower\ X\ n)) \cdot (SEmpty \cup X \cap SInf)) \cdot Y =$
 $(\bigcup n. ((SPower\ X\ n)) \cdot ((SEmpty \cup X \cap SInf) \cdot Y))$
 using *FusionAssoc1*[of $(\bigcup n. ((SPower\ X\ n))) (SEmpty \cup X \cap SInf) Y]$
 by *blast*
have 3: $(\bigcup n. ((SPower\ X\ n)) \cdot ((SEmpty \cup X \cap SInf) \cdot Y)) =$
 $(\bigcup n. ((SPower\ X\ n)) \cdot ((SEmpty \cup X \cap SInf) \cdot Y))$
 using *UnionFusionSPower1*[of $X ((SEmpty \cup X \cap SInf) \cdot Y)$]
 by *auto*
have 4: $\bigwedge n. (SPower\ X\ n) \cdot Y \subseteq Z$
 using *assms fusion-inductl* by *blast*
have 5: $\bigwedge n. ((SPower\ X\ n) \cdot (X \cap SInf)) \cdot Y \subseteq Z$
 using *assms fusion-inductl-inf* by *blast*
have 6: $\bigwedge n. (SPower\ X\ n) \cdot Y \cup ((SPower\ X\ n) \cdot (X \cap SInf)) \cdot Y =$
 $((SPower\ X\ n) \cdot (SEmpty \cup (X \cap SInf))) \cdot Y$
 by *(simp add: FusionSEmptyR FusionUnionDistL FusionUnionDistR)*
have 7: $\bigwedge n. ((SPower\ X\ n) \cdot (SEmpty \cup (X \cap SInf))) \cdot Y \subseteq Z$
 using 4 5 6 by *blast*
have 8: $(\bigcup n. ((SPower\ X\ n) \cdot (SEmpty \cup (X \cap SInf)))) \cdot Y \subseteq Z$
 using 7 by *blast*
show *?thesis*
 by *(metis (no-types, lifting) 1 3 8 FusionAssoc Sup.SUP-cong)*
qed

lemma *SStarInductMoreL*:

assumes $Y \cup (X \cap SMore) \cdot Z \subseteq Z$
shows $(SStar\ X) \cdot Y \subseteq Z$

proof –

have 1: $(SStar\ X) \cdot Y = ((\bigcup n. (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup (X \cap SMore) \cap SInf)) \cdot Y$
 by *(simp add: sstar-def spowerstar-def sfpowerstar-def)*
have 2: $((\bigcup n. (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup (X \cap SMore) \cap SInf)) \cdot Y =$
 $(\bigcup n. ((SPower\ (X \cap SMore)\ n)) \cdot ((SEmpty \cup (X \cap SMore) \cap SInf) \cdot Y))$

using *FusionAssoc1*[of $(\bigcup n. ((SPower (X \cap SMore) n))) (SEmpty \cup (X \cap SMore) \cap SInf) \cdot Y]$
by *blast*
have 3: $(\bigcup n. ((SPower (X \cap SMore) n))) \cdot ((SEmpty \cup (X \cap SMore) \cap SInf) \cdot Y) =$
 $(\bigcup n. ((SPower (X \cap SMore) n)) \cdot ((SEmpty \cup (X \cap SMore) \cap SInf) \cdot Y))$
using *UnionFusionSPower1*[of $X \cap SMore ((SEmpty \cup (X \cap SMore) \cap SInf) \cdot Y)$]
by *auto*
have 4: $\bigwedge n. (SPower (X \cap SMore) n) \cdot Y \subseteq Z$
using *assms fusion-inductl* **by** *blast*
have 5: $\bigwedge n. ((SPower (X \cap SMore) n) \cdot ((X \cap SMore) \cap SInf)) \cdot Y \subseteq Z$
using *assms fusion-inductl-inf* **by** *blast*
have 6: $\bigwedge n. (SPower (X \cap SMore) n) \cdot Y \cup ((SPower (X \cap SMore) n) \cdot ((X \cap SMore) \cap SInf)) \cdot Y =$
 $((SPower (X \cap SMore) n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf))) \cdot Y$
by (*simp add: FusionSEmptyR FusionUnionDistL FusionUnionDistR*)
have 7: $\bigwedge n. ((SPower (X \cap SMore) n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf))) \cdot Y \subseteq Z$
using 4 5 6 **by** *blast*
have 8: $(\bigcup n. ((SPower (X \cap SMore) n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf)))) \cdot Y \subseteq Z$
using 7 **by** *blast*
show ?thesis
by (*metis (no-types, lifting) 1 3 8 FusionAssoc Sup.SUP-cong*)
qed

lemma *SFPowerstarInductR*:

assumes $Y \cup Z \cdot X \subseteq Z$
shows $Y \cdot (SFPowerstar X) \subseteq Z$
proof –
have 1: $Y \cdot (SFPowerstar X) = Y \cdot (\bigcup n. (SPower X n))$
by (*simp add: sfpowerstar-def*)
have 2: $Y \cdot (\bigcup n. (SPower X n)) = (\bigcup n. Y \cdot (SPower X n))$
using *FusionUnionSPower1* **by** *blast*
have 3: $\bigwedge n. Y \cdot (SPower X n) \subseteq Z$
using *assms fusion-inductr* **by** *blast*
have 4: $(\bigcup n. Y \cdot (SPower X n)) \subseteq Z$
using 3 **by** *blast*
show ?thesis
by (*simp add: 1 2 4*)
qed

lemma *SPowerstarInductR*:

assumes $Y \cup Z \cdot X \subseteq Z$
shows $Y \cdot (SPowerstar X) \subseteq Z$
proof –
have 1: $Y \cdot (SPowerstar X) = Y \cdot ((\bigcup n. (SPower X n)) \cdot (SEmpty \cup (X \cap SInf)))$
by (*simp add: spowerstar-def sfpowerstar-def*)
have 11: $Y \cdot ((\bigcup n. (SPower X n)) \cdot (SEmpty \cup (X \cap SInf))) =$
 $Y \cdot ((\bigcup n. (SPower X n) \cdot (SEmpty \cup (X \cap SInf))))$
by (*metis UnionFusionSPower1*)
have 12: $Y \cdot ((\bigcup n. (SPower X n) \cdot (SEmpty \cup (X \cap SInf)))) =$
 $((\bigcup n. Y \cdot (SPower X n) \cdot (SEmpty \cup (X \cap SInf))))$
using *FusionUnionSPower*[of $Y X (SEmpty \cup X \cap SInf)$]
by (*metis (no-types, lifting) FusionAssoc Sup.SUP-cong*)

have 3: $\bigwedge n. Y \cdot (SPower\ X\ n) \subseteq Z$
using *assms fusion-inductr* **by** *blast*
have 31: $\bigwedge n. Y \cdot (SPower\ X\ n) \cdot (X \cap SInf) \subseteq Z$
using *FusionAssoc assms fusion-inductr-inf* **by** *blast*
have 32: $\bigwedge n. Y \cdot (SPower\ X\ n) \cdot (SEmpty \cup (X \cap SInf)) \subseteq Z$
by (*simp add: 3 31 FusionSEmptyR FusionUnionDistR*)
have 4: $(\bigcup n. Y \cdot (SPower\ X\ n) \cdot (SEmpty \cup (X \cap SInf))) \subseteq Z$
using 32 **by** *blast*
show ?thesis
by (*simp add: 1 11 12 4*)
qed

lemma *SStarInductMoreR*:

assumes $Y \cup Z \cdot (X \cap SMore) \subseteq Z$

shows $Y \cdot (SStar\ X) \subseteq Z$

proof –

have 1: $Y \cdot (SStar\ X) = Y \cdot ((\bigcup n. (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf)))$

by (*simp add: spowerstar-def sfpowerstar-def sstar-def*)

have 11: $Y \cdot ((\bigcup n. (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf))) =$
 $Y \cdot ((\bigcup n. (SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf))))$

by (*metis UnionFusionSPower1*)

have 12: $Y \cdot ((\bigcup n. (SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf)))) =$
 $((\bigcup n. Y \cdot (SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf))))$

using *FusionUnionSPower[of Y (X ∩ SMore) (SEmpty ∪ ((X ∩ SMore) ∩ SInf))]*

by (*metis (no-types, lifting) FusionAssoc Sup.SUP-cong*)

have 3: $\bigwedge n. Y \cdot (SPower\ (X \cap SMore)\ n) \subseteq Z$

using *assms fusion-inductr* **by** *blast*

have 31: $\bigwedge n. Y \cdot (SPower\ (X \cap SMore)\ n) \cdot ((X \cap SMore) \cap SInf) \subseteq Z$

using *assms FusionAssoc fusion-inductr-inf* **by** *blast*

have 32: $\bigwedge n. Y \cdot (SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf)) \subseteq Z$

by (*simp add: 3 31 FusionSEmptyR FusionUnionDistR*)

have 4: $(\bigcup n. Y \cdot (SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf))) \subseteq Z$

using 32 **by** *blast*

show ?thesis

by (*simp add: 1 11 12 4*)

qed

lemma *Powerstarhelp2*:

$(X \cap SInf) = (X \cap SInf) \cdot Z$

by (*metis FusionAssoc FusionSFalse SFalseFusion*)

lemma *Powerstarhelp3*:

$((X \cap SInf) \cdot Y \cup (X \cap SFinite) \cdot Y) = X \cdot Y$

by (*metis Diff-Compl FusionUnionDistL Un-Diff-Int sfinite-def*)

lemma *Powerstareqv*:

$(SPowerstar\ X) = SEmpty \cup X \cdot (SPowerstar\ X)$

proof –

have 1: $(SPowerstar\ X) = (\bigcup n. (SPower\ X\ n)) \cdot (SEmpty \cup (X \cap SInf))$

by (*simp add: sfpowerstar-def spowerstar-def*)

have 2: $(\bigcup n. (SPower\ X\ n)) \cdot (SEmpty \cup (X \cap SInf)) =$
 $(SEmpty \cup (X \cap SFinite)) \cdot (\bigcup n. (SPower\ X\ n)) \cdot (SEmpty \cup (X \cap SInf))$
by (*metis sfpowerstar-equiv-1*)
have 3: $(SEmpty \cup (X \cap SFinite)) \cdot (\bigcup n. (SPower\ X\ n)) \cdot (SEmpty \cup (X \cap SInf)) =$
 $SEmpty \cdot (SEmpty \cup (X \cap SInf)) \cup (X \cap SFinite) \cdot (\bigcup n. (SPower\ X\ n)) \cdot (SEmpty \cup (X \cap SInf))$
by (*simp add: FusionUnionDistL*)
have 4: $SEmpty \cdot (SEmpty \cup (X \cap SInf)) \cup (X \cap SFinite) \cdot (\bigcup n. (SPower\ X\ n)) \cdot (SEmpty \cup (X \cap SInf))$
 $=$
 $SEmpty \cup (X \cap SInf) \cup (X \cap SFinite) \cdot (\bigcup n. (SPower\ X\ n)) \cdot (SEmpty \cup (X \cap SInf))$
by (*simp add: FusionSEmptyL*)
have 5: $SEmpty \cup (X \cap SInf) \cup (X \cap SFinite) \cdot (\bigcup n. (SPower\ X\ n)) \cdot (SEmpty \cup (X \cap SInf)) =$
 $SEmpty \cup (X \cap SInf) \cdot (\bigcup n. (SPower\ X\ n)) \cdot (SEmpty \cup (X \cap SInf)) \cup$
 $(X \cap SFinite) \cdot (\bigcup n. (SPower\ X\ n)) \cdot (SEmpty \cup (X \cap SInf))$
by (*metis Powerstarhelp2*)
have 6: $SEmpty \cup (X \cap SInf) \cdot (\bigcup n. (SPower\ X\ n)) \cdot (SEmpty \cup (X \cap SInf)) \cup$
 $(X \cap SFinite) \cdot (\bigcup n. (SPower\ X\ n)) \cdot (SEmpty \cup (X \cap SInf)) =$
 $SEmpty \cup X \cdot ((\bigcup n. (SPower\ X\ n)) \cdot (SEmpty \cup (X \cap SInf)))$
by (*metis FusionAssoc Powerstarhelp3 UnionAssoc*)
show ?thesis
using 1 2 3 4 5 6 **by** presburger
qed

lemma *SStareqv*:
 $SStar\ X = SEmpty \cup (X \cap SMore) \cdot (SStar\ X)$
by (*metis Powerstareqv sstar-def*)

lemma *SSkipSFinite*:
 $SSkip \cap SFinite = SSkip$
proof –
have 1: $SSkip \cap SFinite \subseteq SSkip$
by *simp*
have 2: $SInf \subseteq SEmpty \cup (-\ SEmpty \cdot -\ SEmpty)$
by (*metis Diff-Compl Diff-subset-conv Powerstarhelp2 Powerstarhelp3 double-compl inf-commute sup-ge1*)
have 3: $SSkip \subseteq SSkip \cap SFinite$
using 2 **by** (*simp add: sskip-def smore-def sfinite-def*) *blast*
show ?thesis
using 3 **by** *blast*
qed

lemma *spower-skip-elim* :
 $(\sigma \in SPower\ SSkip\ n) \longleftrightarrow$
 $(nlength\ \sigma = n)$
proof (*induct n arbitrary: σ*)
case 0
then show ?case
by (*simp add: empty-elim zero-enat-def*)
next
case (*Suc n*)
then show ?case

proof –

have 1: $(\sigma \in \text{SPower } \text{SSkip } (\text{Suc } n)) \longleftrightarrow \sigma \in (\text{SFinite} \cap \text{SSkip}) \cdot (\text{SPower } \text{SSkip } n)$
by (*simp add: inf-commute*)
have 2: $\sigma \in (\text{SFinite} \cap \text{SSkip}) \longleftrightarrow \sigma \in \text{SSkip}$
using *SSkipSFinite* **by** *auto*
have 3: $\sigma \in (\text{SFinite} \cap \text{SSkip}) \cdot (\text{SPower } \text{SSkip } n) \longleftrightarrow$
 $(\exists \sigma 1 \sigma 2. \sigma = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge \sigma 1 \in \text{SSkip} \wedge \sigma 2 \in \text{SPower } \text{SSkip } n \wedge \text{nlast } \sigma 1 = \text{nfirst } \sigma 2)$
using *ffusion-iff[of σ SSkip SPower SSkip n]* **by** (*simp add: inf-commute*)
have 4: $(\exists \sigma 1 \sigma 2. \sigma = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge \sigma 1 \in \text{SSkip} \wedge \sigma 2 \in \text{SPower } \text{SSkip } n \wedge \text{nlast } \sigma 1 = \text{nfirst } \sigma 2) \longleftrightarrow$
 $(\text{nlength } \sigma = \text{enat } (\text{Suc } n))$
by (*auto simp add: Suc.hyps nfuse-nlength one-enat-def sskip-elim*)
(metis One-nat-def add.commute eSuc-enat enat.simps(3) enat-add-sub-same enat-le-plus-same(2) min.orderE ndropn-nfirst ndropn-nlength nfinite-ntaken nfuse-ntaken-ndropn ntaken-nlast ntaken-nlength one-enat-def plus-1-eSuc(2))
show *?thesis*
using 1 3 4 **by** *presburger*
qed
qed

Kleene Algebra

lemma *UnfoldL*:

$$\text{SEmpty} \cup X \cdot (\text{SStar } X) = (\text{SStar } X)$$

proof –

have 1: $(\text{SStar } X) = \text{SEmpty} \cup (X \cap \text{SMore}) \cdot (\text{SStar } X)$
by (*meson Un-iff set-eqI SStareqv*)
have 2: $(X \cap \text{SMore}) \cdot (\text{SStar } X) \subseteq X \cdot (\text{SStar } X)$
by (*simp add: FusionRuleL*)
have 3: $(\text{SStar } X) \subseteq \text{SEmpty} \cup X \cdot (\text{SStar } X)$
using 1 2 **by** *blast*
have 4: $\text{SEmpty} \subseteq (\text{SStar } X)$
using 1 **by** *auto*
have 5: $X \subseteq \text{SEmpty} \cup (X \cap \text{SMore})$
by (*simp add: Un-Int-distrib smore-def*)
have 6: $X \cdot (\text{SStar } X) \subseteq (\text{SStar } X) \cup (X \cap \text{SMore}) \cdot (\text{SStar } X)$
using 5 **by** (*metis FusionRuleL FusionUnionDistL FusionSEmptyL*)
have 7: $(\text{SStar } X) \subseteq \text{SEmpty} \cup (X \cap \text{SMore}) \cdot (\text{SStar } X)$
using 1 **by** *auto*
have 8: $X \cdot (\text{SStar } X) \subseteq \text{SEmpty} \cup (X \cap \text{SMore}) \cdot (\text{SStar } X)$
using 6 7 **by** *blast*
hence 9: $X \cdot (\text{SStar } X) \subseteq (\text{SStar } X)$
using 1 **by** *auto*
have 10: $\text{SEmpty} \cup X \cdot (\text{SStar } X) \subseteq (\text{SStar } X)$
using 9 4 **by** *simp*
from 3 10 **show** *?thesis* **by** *auto*
qed

— Left induction

lemma *SStarInductL*:

assumes $Y \cup X \cdot Z \subseteq Z$

shows $(SStar\ X) \cdot Y \subseteq Z$

proof —

have 1: $(SStar\ X) \cdot Y = ((\bigcup n. (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup (X \cap SMore) \cap SInf)) \cdot Y$
by (*simp add: sstar-def spowerstar-def sfpowerstar-def*)

have 2: $((\bigcup n. (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup (X \cap SMore) \cap SInf)) \cdot Y =$
 $(\bigcup n. ((SPower\ (X \cap SMore)\ n)) \cdot ((SEmpty \cup (X \cap SMore) \cap SInf) \cdot Y))$
using *FusionAssoc1*[*of* $(\bigcup n. ((SPower\ (X \cap SMore)\ n)) (SEmpty \cup (X \cap SMore) \cap SInf)\ Y)$]
by *blast*

have 3: $(\bigcup n. ((SPower\ (X \cap SMore)\ n)) \cdot ((SEmpty \cup (X \cap SMore) \cap SInf) \cdot Y)) =$
 $(\bigcup n. ((SPower\ (X \cap SMore)\ n)) \cdot ((SEmpty \cup (X \cap SMore) \cap SInf) \cdot Y))$
using *UnionFusionSPower1*[*of* $X \cap SMore ((SEmpty \cup (X \cap SMore) \cap SInf) \cdot Y)$]
by *auto*

have 31: $Y \cup (X \cap SMore) \cdot Z \subseteq Z$
by (*meson FusionRuleL Un-mono assms dual-order.trans inf.cobounded1 subset-refl*)

have 4: $\bigwedge n. (SPower\ (X \cap SMore)\ n) \cdot Y \subseteq Z$
using *assms fusion-inductl-more* **by** *blast*

have 5: $\bigwedge n. ((SPower\ (X \cap SMore)\ n) \cdot ((X \cap SMore) \cap SInf)) \cdot Y \subseteq Z$
using 31 *fusion-inductl-inf* **by** *blast*

have 6: $\bigwedge n. (SPower\ (X \cap SMore)\ n) \cdot Y \cup ((SPower\ (X \cap SMore)\ n) \cdot ((X \cap SMore) \cap SInf)) \cdot Y =$
 $((SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf))) \cdot Y$
by (*simp add: FusionSEmptyR FusionUnionDistL FusionUnionDistR*)

have 7: $\bigwedge n. ((SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf))) \cdot Y \subseteq Z$
using 4 5 6 **by** *blast*

have 8: $(\bigcup n. ((SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf)))) \cdot Y \subseteq Z$
using 7 **by** *blast*

show *?thesis*

by (*metis (no-types, lifting) 1 3 8 FusionAssoc Sup.SUP-cong*)

qed

— Right induction

lemma *SStarInductR*:

assumes $Y \cup Z \cdot X \subseteq Z$

shows $Y \cdot (SStar\ X) \subseteq Z$

proof —

have 1: $Y \cdot (SStar\ X) = Y \cdot ((\bigcup n. (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf)))$
by (*simp add: spowerstar-def sfpowerstar-def sstar-def*)

have 11: $Y \cdot ((\bigcup n. (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf))) =$
 $Y \cdot ((\bigcup n. (SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf))))$
by (*metis UnionFusionSPower1*)

have 12: $Y \cdot ((\bigcup n. (SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf)))) =$
 $((\bigcup n. Y \cdot (SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf))))$
using *FusionUnionSPower*[*of* $Y (X \cap SMore) (SEmpty \cup ((X \cap SMore) \cap SInf))$]
by (*metis (no-types, lifting) FusionAssoc Sup.SUP-cong*)

have 21: $Y \cup Z \cdot (X \cap SMore) \subseteq Z$

by (*meson FusionRuleR Un-mono assms dual-order.trans inf.cobounded1 subset-refl*)

```

have 3:  $\bigwedge n. Y \cdot (SPower (X \cap SMore) n) \subseteq Z$ 
  using 21 fusion-inductr by blast
have 31:  $\bigwedge n. Y \cdot (SPower (X \cap SMore) n) \cdot ((X \cap SMore) \cap SInf) \subseteq Z$ 
  using 21 FusionAssoc fusion-inductr-inf by blast
have 32:  $\bigwedge n. Y \cdot (SPower (X \cap SMore) n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf)) \subseteq Z$ 
  by (simp add: 3 31 FusionSEmptyR FusionUnionDistR)
have 4:  $(\bigcup n. Y \cdot (SPower (X \cap SMore) n) \cdot (SEmpty \cup ((X \cap SMore) \cap SInf))) \subseteq Z$ 
  using 32 by blast
show ?thesis
  by (simp add: 1 11 12 4)
qed

```

Omega Algebra

lemma *SFMoresem* [mono]:

$x \in ((((F \cap SMore) \cap SFinite) \cdot G)) \iff$
 $((\exists \sigma 1 \sigma 2. x = nfuse \sigma 1 \sigma 2 \wedge nfinite \sigma 1 \wedge \sigma 1 \in F \wedge 0 < (nlength \sigma 1) \wedge \sigma 2 \in G \wedge nlast \sigma 1 = nfirst \sigma 2))$

by (simp add: fusion-iff sinfinite-elim smore-elim sfinite-elim) blast

lemma *monoomega*[mono]:

$mono (\lambda p x. x \in ((F \cap SMore) \cap SFinite \cdot p))$

by (auto simp add: mono-def sinfinite-elim fusion-iff sfmore-elim)

coinductive-set *somega* :: 'a iintervals \Rightarrow 'a iintervals

for *F* where

$((\exists \sigma 1 \sigma 2. x = nfuse \sigma 1 \sigma 2 \wedge nfinite \sigma 1 \wedge \sigma 1 \in F \wedge 0 < (nlength \sigma 1) \wedge$
 $\sigma 2 \in (somega F) \wedge nlast \sigma 1 = nfirst \sigma 2))$
 $\implies x \in (somega F)$

lemma *SOmegaIntros*:

$((((F \cap SMore) \cap SFinite) \cdot (somega F)) \subseteq (somega F))$

using *somega.intros*[of - *F*] *SFMoresem*[of - *F* (*somega F*)]

by (*meson subsetI*)

lemma *SOmegaCases*:

assumes $x \in somega F$

$x \in ((((F \cap SMore) \cap SFinite) \cdot (somega F))) \implies P$

shows *P*

using *assms somega.cases*[of *x F P*] *SFMoresem* **by** blast

lemma *SOmegaUnrollsem*:

$x \in (somega F) \iff x \in (((F \cap SMore) \cap SFinite) \cdot (somega F))$

proof auto

show $x \in somega F \implies x \in (F \cap SMore) \cap SFinite \cdot somega F$

using *SOmegaCases* **by** blast

show $x \in (F \cap SMore) \cap SFinite \cdot somega F \implies x \in somega F$

by (*metis* (*no-types*, *lifting*) *SOmegaIntros inf-commute subset-eq*)

qed

lemma *SOmegaUnroll*:

$(\text{somega } F) = (((F \cap \text{SMore}) \cap \text{SFinite}) \cdot (\text{somega } F))$

using *SOmegaUnrollsem* **by** *blast*

lemma *SOmegaCoinductsem*:

assumes $\bigwedge x. x \in X \implies x \in ((((F \cap \text{SMore}) \cap \text{SFinite}) \cdot (X \cup (\text{somega } F))))$

shows $x \in X \implies x \in (\text{somega } F)$

using *assms somega.coinduct[of $\lambda x. x \in X \ x \ F$] SFMoresem[of $- \ F \ (X \cup \text{somega } F)$]* **by** *auto*

lemma *SOmegaCoinduct*:

assumes $X \subseteq ((((F \cap \text{SMore}) \cap \text{SFinite}) \cdot (X \cup (\text{somega } F))))$

shows $X \subseteq (\text{somega } F)$

using *assms SOmegaCoinductsem[of $X \ F$]* **by** *blast*

lemma *SOmegaWeakCoinductsem*:

assumes $\bigwedge x. x \in X \implies x \in ((((F \cap \text{SMore}) \cap \text{SFinite}) \cdot X))$

shows $x \in X \implies x \in (\text{somega } F)$

using *assms somega.coinduct[of $\lambda x. x \in X \ x \ F$]*

by (*metis SFMoresem*)

lemma *SOmegaWeakCoinduct*:

assumes $X \subseteq ((((F \cap \text{SMore}) \cap \text{SFinite}) \cdot X))$

shows $X \subseteq (\text{somega } F)$

using *assms SOmegaWeakCoinductsem[of $X \ F$]* **by** *blast*

ITL specific Laws

lemma *PwrFusionInterLsem*:

$x \in (((((\text{SPower } \text{SSkip } n) \cap X) \cdot V) \cap (((\text{SPower } \text{SSkip } n) \cap Y) \cdot W))) \longleftrightarrow$

$x \in (((\text{SPower } \text{SSkip } n) \cap X \cap Y) \cdot (V \cap W))$

by (*auto simp add: spower-skip-elim fusion-iff-1 min-absorb1 nlength-eq-enat-nfiniteD*)

lemma *PwrFusionInterL*:

$(((((\text{SPower } \text{SSkip } n) \cap X) \cdot V) \cap (((\text{SPower } \text{SSkip } n) \cap Y) \cdot W))) =$

$((((\text{SPower } \text{SSkip } n) \cap X \cap Y) \cdot (V \cap W)))$

using *PwrFusionInterLsem* **by** *blast*

lemma *PwrFusionInterRsem* :

$x \in ((V \cdot ((\text{SPower } \text{SSkip } n) \cap X)) \cap ((W \cdot ((\text{SPower } \text{SSkip } n) \cap Y)))) \longleftrightarrow$

$x \in ((V \cap W) \cdot ((\text{SPower } \text{SSkip } n) \cap X \cap Y))$

by (*simp add: spower-skip-elim fusion-iff-1*)

(*rule,*

metis enat.simps(3) enat-add2-eq enat-add-sub-same le-iff-add ndropn-nlength nfinite-ndropn

nlength-eq-enat-nfiniteD the-enat.simps,

metis enat.simps(3) enat-add-sub-same le-iff-add ndropn-nlength nfinite-ndropn-b

nlength-eq-enat-nfiniteD)

lemma *PwrFusionInterR* :

$((V \cdot ((\text{SPower } \text{SSkip } n) \cap X)) \cap ((W \cdot ((\text{SPower } \text{SSkip } n) \cap Y)))) =$

$((V \cap W) \cdot ((\text{SPower } \text{SSkip } n) \cap X \cap Y))$

using *PwrFusionInterRsem* **by** *blast*

lemma *SSkipFusionImpSMore*:

$SSkip \cdot STrue \subseteq SMore$

using *subsetI*[of *SSkip*·*STrue* *SMore*]

by (*auto simp add: fusion-iff-1 sskip-elim smore-elim strue-elim*)

lemma *SMoreImpSSkipFusion*:

$SMore \subseteq SSkip \cdot STrue$

using *subsetI*[of *SMore* *SSkip*·*STrue*]

by (*auto simp add: fusion-iff-1 sskip-elim smore-elim strue-elim*)

(*metis i0-less ileI1 one-eSuc one-enat-def*,

metis i0-less ileI1 one-eSuc one-enat-def)

lemma *SSkipSMore*:

$SSkip \cap SMore = SSkip$

by (*simp add: inf.absorb1 smore-def sskip-def*)

lemma *SPowerSkip*:

$(\bigcup n. (SPower\ SSkip\ n)) = SFinite$

proof –

have 1: $(\bigcup n. (SPower\ SSkip\ n)) \subseteq SFinite$

by (*simp add: SUP-least spower-finite*)

have 2: $\bigwedge x. x \in SFinite \implies x \in (\bigcup n. (SPower\ SSkip\ n))$

by (*auto simp add: sfinite-elim spower-sskip-elim nfinite-nlength-enat*)

have 3: $SFinite \subseteq (\bigcup n. (SPower\ SSkip\ n))$

using 2 **by** *blast*

from 1 3 **show** ?thesis **by** *blast*

qed

lemma *SStarSkip*:

$(SStar\ SSkip) = SFinite$

by (*simp add: sstar-def SSkipSMore spowerstar-def sfpowerstar-def SPowerSkip*)

(*metis Compl-disjoint2 FusionSEmptyR SSkipSFinite UnionSFalse inf.assoc inf-bot-right*
sfalse-def sfinite-def)

2.15.4 Derived Laws

Helper Lemmas

lemma *B01*:

assumes $(X:: 'a\ iintervals) \subseteq Y$

shows $-Y \subseteq -X$

using *assms* **by** *auto*

lemma *B04*:

$((X:: 'a\ iintervals) = Y) \longleftrightarrow (X \subseteq Y) \wedge (Y \subseteq X)$

by *auto*

lemma *B09*:

assumes $-X \cup Y = STrue$

shows $(X:: 'a\ iintervals) \subseteq Y$
using *assms using strue-def by auto*

lemma B20:
 $(X:: 'a\ iintervals) \subseteq Y \cup Z \longleftrightarrow X \cap -Y \subseteq Z$
by *auto*

lemma B28:
 $((X:: 'a\ iintervals) \cap Y) \cup (X \cap Z) = X \cap (Y \cup Z)$
by *auto*

lemma CH01:
 $STrue \cdot STrue = STrue$
by (*metis FusionSEmptyR FusionUnionDistR Int-commute STrueTop inf-sup-absorb*)

lemma CH07:
 $((SSkip \cap X) \cdot V) \cap ((SSkip \cap Y) \cdot W) = ((SSkip \cap X \cap Y) \cdot (V \cap W))$
using *PwrFusionInterL[of 1 X V Y W]*
by (*simp add: FusionSEmptyR SSkipSFinite*)

lemma CH08:
 $((V \cdot (SSkip \cap X)) \cap ((W \cdot (SSkip \cap Y)))) = ((V \cap W) \cdot (SSkip \cap X \cap Y))$
using *PwrFusionInterR[of V 1 X W Y]*
by (*simp add: FusionSEmptyR SSkipSFinite*)

lemma CH09:
 $((X \cap SEmpty) \cdot V) \cap ((Y \cap SEmpty) \cdot W) = ((X \cap Y) \cap SEmpty) \cdot (V \cap W)$
using *PwrFusionInterL[of 0 X V Y W]*
by (*metis (no-types, lifting) inf-assoc inf-commute pwr-0*)

lemma CH10:
 $((V \cdot (X \cap SEmpty)) \cap ((W \cdot (Y \cap SEmpty)))) = ((V \cap W) \cdot ((X \cap Y) \cap SEmpty))$
using *PwrFusionInterR[of V 0 X W Y]*
by (*metis (no-types, lifting) inf-assoc inf-commute pwr-0*)

lemma ST13:
 $((X \cap SEmpty) \cdot Z) \cap ((Y \cap SEmpty) \cdot Z) = ((X \cap Y) \cap SEmpty) \cdot Z$
by (*simp add: CH09*)

lemma ST15:
 $(SStar (X \cap SEmpty)) = SEmpty$
using *SStareqv[of (X \cap SEmpty)]*
by (*metis Compl-disjoint2 Powerstarhelp2 SFalseBottom UnionSFalse inf-assoc inf-bot-right sfalse-def smore-def*)

lemma ST21:
 $((-X) \cap SEmpty) \cup (X \cap SEmpty) = SEmpty$
by *blast*

lemma ST24:

$(SInit\ X) \cap (SInit\ Y) = (SInit\ (X \cap Y))$
by (*simp add: ST13 sinit-def*)

lemma *ST25*:
 $(SInit\ STrue) = STrue$
by (*simp add: sinit-def strue-def FusionSEmptyL*)

lemma *ST26*:
 $(SInit\ (-X)) \cup (SInit\ X) = STrue$
by (*metis Compl-disjoint2 ST21 ST25 Un-Int-distrib compl-bot-eq FusionUnionDistL sinit-def strue-def sup-bot.right-neutral sup-top-right*)

lemma *ST28*:
 $(SDi\ (SInit\ X)) = (SInit\ X)$
by (*metis compl-bot-eq FusionAssoc FusionUnionDistR FusionSEmptyR sdi-def sinit-def strue-def sup-top-right UnionCommute*)

lemma *ST33*:
 $(STrue \cap SEmpty) \cdot SEmpty = SEmpty$
by (*simp add: strue-def FusionSEmptyL*)

lemma *ST36*:
 $(SInit\ (-X)) \subseteq -(SInit\ X)$
unfolding *sinit-def*
by (*metis SFalseBottom SFalseFusion ST24 disjoint-eq-subset-Compl inf commute inf-compl-bot-left2 sinit-def*)

lemma *ST37*:
 $-(SInit\ X) \subseteq (SInit\ (-X))$
using *B09 ST26* **by** *auto*

lemma *ST38*:
 $-(SInit\ X) = (SInit\ (-X))$
using *ST37 ST36* **by** *auto*

lemma *ST47*:
 $X \cup Y \cdot X = (SEmpty \cup Y) \cdot X$
by (*simp add: FusionUnionDistL FusionSEmptyL*)

lemma *SStar01*:
assumes $X \cdot (SStar\ Y) \cup SEmpty \subseteq (SStar\ Y)$
shows $(SStar\ X) \subseteq (SStar\ Y)$
using *assms*
by (*metis Un-commute FusionSEmptyR SStarInductL*)

lemma *SStar03*:
 $(SStar\ X) \cdot (SStar\ X) \subseteq (SStar\ X)$
by (*metis SStarInductL Un-absorb UnfoldL sup.absorb-iff1 sup.right-idem*)

lemma *SStar05*:

assumes $((SStar\ X) \cdot (SStar\ X)) \cup SEmpty \subseteq (SStar\ X)$
shows $(SStar\ (SStar\ X)) \subseteq (SStar\ X)$
using *assms*
by (*simp add: SStar01*)

lemma *SStar12*:
 $(SEmpty \cup (X \cdot (SStar\ X))) \subseteq (SStar\ X)$
using *UnfoldL* **by** *blast*

lemma *SStar06*:
 $((SStar\ X) \cdot (SStar\ X)) \cup SEmpty \subseteq (SStar\ X)$
using *SStar03 SStar12* **by** *force*

lemma *SStar07*:
 $(SStar\ X) \subseteq (SStar\ (SStar\ X))$
proof –
have $SStar\ X \cdot SEmpty \cup SStar\ X \cdot (SStar\ X \cdot SStar\ SStar\ X) = SStar\ X \cdot SStar\ SStar\ X$
by (*metis (full-types) FusionUnionDistR UnfoldL*)
then have $SStar\ X \cdot SEmpty \cup SStar\ X \cdot (SStar\ X \cdot SStar\ SStar\ X) \subseteq SStar\ SStar\ X$
using *UnfoldL* **by** *auto*
then show *?thesis*
by (*simp add: FusionSEmptyR*)
qed

lemma *SStar08*:
 $(SStar\ X) = (SStar\ (SStar\ X))$
by (*meson B04 SStar05 SStar06 SStar07*)

lemma *SStar15*:
 $SEmpty \subseteq (SStar\ SSkip)$
using *UnfoldL* **by** *fastforce*

lemma *SStar16*:
 $SSkip \subseteq (SStar\ SSkip)$
using *SSkipSFinite SStarSkip* **by** *blast*

lemma *SStar22*:
 $(SEmpty \cap X) \cdot (SStar\ (SEmpty \cap X)) = (SEmpty \cap X)$
by (*metis ST15 FusionSEmptyR inf-commute*)

lemma *SStar23*:
 $(SStar\ (SEmpty \cap X)) = SEmpty$
using *SStar22 UnfoldL* **by** *auto*

lemma *SStar25*:
 $(SStar\ STrue) = STrue$
by (*metis FusionRuleR FusionSEmptyR Un-absorb2 UnfoldL UnionCommute compl-bot-eq strue-def sup.right-idem sup-ge2 sup-top-right*)

lemma *SStar28*:

$(SStar\ X) \cdot X \subseteq X \cdot (SStar\ X)$
by (*metis B04 FusionUnionDistR FusionSEmptyR UnfoldL SStarInductL*)

lemma *SStar29*:
 $X \cdot (SStar\ X) \subseteq (SStar\ X) \cdot X$
by (*metis B04 SStar28 SStarInductR UnfoldL FusionRuleL ST47 sup.mono*)

lemma *SStar17*:
 $(SStar\ SSkip) \cdot SSkip \subseteq SSkip \cdot (SStar\ SSkip)$
by (*simp add: SStar28*)

lemma *SStar18*:
 $SSkip \cdot (SStar\ SSkip) \subseteq (SStar\ SSkip) \cdot SSkip$
by (*simp add: SStar29*)

lemma *SStar19*:
 $(SStar\ SSkip) \cdot SSkip = SSkip \cdot (SStar\ SSkip)$
using *SStar17 SStar18 by auto*

lemma *SStar30*:
 $X \cdot (SStar\ X) = (SStar\ X) \cdot X$
using *SStar28 SStar29 by auto*

lemma *SStar34*:
assumes $SEmpty \cup (X \cup Y) \cdot ((SStar\ X) \cdot (SStar\ (Y \cdot (SStar\ X)))) \subseteq (SStar\ X) \cdot (SStar\ (Y \cdot (SStar\ X)))$
shows $(SStar\ (X \cup Y)) \subseteq (SStar\ X) \cdot (SStar\ (Y \cdot (SStar\ X)))$
by (*metis assms FusionSEmptyR SStarInductL*)

lemma *SStar35*:
 $SEmpty \cup (X \cup Y) \cdot ((SStar\ X) \cdot (SStar\ (Y \cdot (SStar\ X)))) \subseteq (SStar\ X) \cdot (SStar\ (Y \cdot (SStar\ X)))$
by (*simp add: FusionAssoc FusionUnionDistL ST47 UnfoldL UnionAssoc UnionCommute*)

lemma *SStar36*:
 $(SStar\ (X \cup Y)) \subseteq (SStar\ X) \cdot (SStar\ (Y \cdot (SStar\ X)))$
using *SStar34 SStar35 by blast*

lemma *SStar46*:
 $(SStar\ X) \cdot (SStar\ (Y \cdot (SStar\ X))) \subseteq (SStar\ (X \cup Y))$
proof –
have $(SEmpty \cup SStar\ (X \cup Y) \cdot Y) \cdot SStar\ X \subseteq SStar\ (X \cup Y)$
by (*metis (no-types) FusionUnionDistR SStar12 SStar30 SStarInductR sup.bounded-iff*)
then show *?thesis* **by** (*simp add: SStarInductR ST47 FusionAssoc*)
qed

lemma *SStar47*:
 $(SStar\ Z) = (SStar\ (Z \cap SMore))$
proof –
have 1: $(SStar\ Z) = (SStar\ ((SEmpty \cap Z) \cup (SMore \cap Z)))$
by (*metis Int-Un-distrib2 compl-bot-eq inf-top.left-neutral smore-def strue-def STrueTop*)
have 2: $(SStar\ ((SEmpty \cap Z) \cup (SMore \cap Z))) =$

$(SStar (SEmpty \cap Z)) \cdot (SStar ((SMore \cap Z) \cdot (SStar (SEmpty \cap Z))))$
 by (simp add: SStar36 SStar46 subset-antisym)
 have 3: $(SStar (SEmpty \cap Z)) \cdot (SStar ((SMore \cap Z) \cdot (SStar (SEmpty \cap Z)))) =$
 $(SStar (Z \cap SMore))$
 by (simp add: FusionSEmptyL FusionSEmptyR SStar23 inf-commute)
 from 1 2 3 show ?thesis by auto
 qed

lemma SStar48:

$(SStar SMore) = STrue$
 by (metis Compl-Un Compl-disjoint2 SStar25 SStar47 ST21 ST33 FusionSEmptyR
 inf.right-idem smore-def strue-def)

lemma SStar50:

assumes $SSkip \cdot ((-X) \cup ((SStar SSkip) \cdot (X \cap (SSkip \cdot (-X))))) \cup (-X)$
 $\subseteq (-X) \cup (SStar SSkip) \cdot (X \cap (SSkip \cdot (-X)))$
 shows $((SStar SSkip) \cdot (-X)) \subseteq ((-X) \cup ((SStar SSkip) \cdot (X \cap (SSkip \cdot (-X)))))$
 using SStarInductL assms by blast

lemma SStar51:

$SSkip \cdot ((-X) \cup ((SStar SSkip) \cdot (X \cap (SSkip \cdot (-X))))) \cup (-X)$
 $\subseteq (-X) \cup (SStar SSkip) \cdot (X \cap (SSkip \cdot (-X)))$
 using FusionUnionDistR[of SSkip] UnfoldL[of SSkip]
 proof –
 have f1: $-X \cup (SEmpty \cup SSkip \cdot SStar SSkip) \cdot (X \cap (SSkip \cdot -X)) \subseteq$
 $-X \cup SStar SSkip \cdot (X \cap (SSkip \cdot -X))$
 by (simp add: $\langle SEmpty \cup SSkip \cdot SStar SSkip = SStar SSkip \rangle$)
 have f2: $SSkip \cdot -X \subseteq -X \cup SStar SSkip \cdot (X \cap (SSkip \cdot -X))$
 by (metis B20 Morgan ST47 UnionIdem $\langle SEmpty \cup SSkip \cdot SStar SSkip = SStar SSkip \rangle$
 inf-commute inf-idem sup.cobounded1)
 have $SSkip \cdot (SStar SSkip \cdot (X \cap (SSkip \cdot -X))) \subseteq -X \cup SStar SSkip \cdot (X \cap (SSkip \cdot -X))$
 using f1 by (metis (no-types) FusionAssoc ST47 Un-subset-iff)
 then show ?thesis
 using f2 by (simp add: $\langle \bigwedge Z Y. SSkip \cdot (Y \cup Z) = SSkip \cdot Y \cup SSkip \cdot Z \rangle$)
 qed

lemma SStar52:

$(SStar X) \subseteq SEmpty \cup (X \cap SMore) \cdot (SStar X)$
 by (metis B04 SStar47 UnfoldL)

lemma SStar53:

$SEmpty \cup (X \cap SMore) \cdot (SStar X) \subseteq (SStar X)$
 by (metis SStar12 SStar47)

lemma BD45:

$(SBI ((-X) \cup X1)) \cap (X \cdot Y) \subseteq X1 \cdot Y$
 proof –
 have 1: $(SBI ((-X) \cup X1)) = -(-((-X) \cup X1) \cdot (Y \cup (-Y)))$
 by (metis sbi-def sdi-def STrueTop)
 have 2: $-(-((-X) \cup X1) \cdot (Y \cup (-Y))) \cap (X \cdot Y) \subseteq$

$$\neg((\neg(-X) \cup X1) \cdot (Y)) \cap (X \cdot Y)$$
using *FusionUnionDistR* **by** *fastforce*
have 3: $\neg((\neg(-X) \cup X1) \cdot (Y)) \cap (X \cdot Y) \subseteq (((\neg(-X) \cup X1) \cap X) \cdot Y)$
by (*metis* (*no-types*, *opaque-lifting*) *B20 FusionRuleL FusionUnionDistL Morgan UnionCommute double-compl order-refl*)
have 4: $((\neg(-X) \cup X1) \cap X) \cdot Y \subseteq X1 \cdot Y$
by (*metis* *B20 double-compl FusionRuleL inf.right-idem inf-le1*)
from 1 2 3 4 **show** ?thesis **by** *blast*
qed

lemma *BD46*:

$(SAlways ((\neg Y) \cup Y1)) \cap (X1 \cdot Y) \subseteq (X1 \cdot Y1)$

proof –

have 1: $(SAlways ((\neg Y) \cup Y1)) = \neg((SFinite \cap (X1 \cup \neg X1)) \cdot (\neg((\neg Y) \cup Y1)))$

by (*simp add: salways-def sometime-def*)

have 2: $\neg((SFinite \cap (X1 \cup \neg X1)) \cdot (\neg((\neg Y) \cup Y1))) =$
 $\neg((SFinite \cap X1) \cup (SFinite \cap \neg X1)) \cdot (\neg((\neg Y) \cup Y1))$

by (*simp add: B28*)

have 3: $\neg((SFinite \cap X1) \cup (SFinite \cap \neg X1)) \cdot (\neg((\neg Y) \cup Y1)) =$
 $\neg((SFinite \cap X1) \cdot (\neg((\neg Y) \cup Y1)) \cup (SFinite \cap \neg X1) \cdot (\neg((\neg Y) \cup Y1)))$

by (*simp add: FusionUnionDistL*)

have 4: $\neg((SFinite \cap X1) \cdot (\neg((\neg Y) \cup Y1)) \cup (SFinite \cap \neg X1) \cdot (\neg((\neg Y) \cup Y1))) =$
 $\neg((SFinite \cap X1) \cdot (\neg((\neg Y) \cup Y1))) \cap \neg((SFinite \cap \neg X1) \cdot (\neg((\neg Y) \cup Y1)))$

by *blast*

have 5: $(X1 \cdot Y) = ((SFinite \cap X1) \cdot Y) \cup (SInf \cap X1)$

by (*metis* *Powerstarhelp2 Powerstarhelp3 UnionCommute inf-commute*)

have 6: $\neg((SFinite \cap X1) \cdot (\neg((\neg Y) \cup Y1))) =$
 $\neg((SFinite \cap X1) \cdot (Y \cap \neg Y1))$

by *simp*

have 7: $\neg((SFinite \cap X1) \cdot (\neg((\neg Y) \cup Y1))) \cap ((SFinite \cap X1) \cdot Y) \subseteq$
 $(SFinite \cap X1) \cdot ((\neg Y) \cup Y1) \cap Y$

by (*metis* (*no-types*) *B20 FusionRuleR FusionUnionDistR double-compl inf-sup-aci(1) order-refl*)

have 8: $(SFinite \cap X1) \cdot ((\neg Y) \cup Y1) \cap Y \subseteq (SFinite \cap X1) \cdot Y1$

by (*metis* *B20 FusionRuleR double-compl inf.cobounded1 inf.right-idem*)

have 9: $(SFinite \cap X1) \cdot Y1 \subseteq X1 \cdot Y1$

by (*simp add: FusionRuleL*)

have 10: $\neg((SFinite \cap X1) \cdot (\neg((\neg Y) \cup Y1))) \cap (SInf \cap X1) \subseteq ((SFinite \cap X1) \cdot Y1) \cup (SInf \cap X1)$

by *blast*

have 11: $((SFinite \cap X1) \cdot Y1) \cup (SInf \cap X1) \subseteq X1 \cdot Y1$

by (*metis* *B04 Powerstarhelp2 Powerstarhelp3 UnionCommute inf-commute*)

have 12: $\neg((SFinite \cap (X1 \cup \neg X1)) \cdot (\neg((\neg Y) \cup Y1))) \cap (X1 \cdot Y) =$
 $(\neg((SFinite \cap (X1 \cup \neg X1)) \cdot (\neg((\neg Y) \cup Y1)))) \cap ((SFinite \cap X1) \cdot Y) \cup$
 $(\neg((SFinite \cap (X1 \cup \neg X1)) \cdot (\neg((\neg Y) \cup Y1)))) \cap (SInf \cap X1)$

by (*simp add: 5 B28*)

have 13: $(\neg((SFinite \cap (X1 \cup \neg X1)) \cdot (\neg((\neg Y) \cup Y1)))) \cap ((SFinite \cap X1) \cdot Y) \subseteq X1 \cdot Y1$

using 7 8 9 2 3 **by** *blast*

have 14: $(\neg((SFinite \cap (X1 \cup \neg X1)) \cdot (\neg((\neg Y) \cup Y1)))) \cap (SInf \cap X1) \subseteq X1 \cdot Y1$

using 10 11 **by** *blast*

have 15: $(\neg((SFinite \cap (X1 \cup \neg X1)) \cdot (\neg((\neg Y) \cup Y1)))) \cap ((SFinite \cap X1) \cdot Y) \cup$
 $(\neg((SFinite \cap (X1 \cup \neg X1)) \cdot (\neg((\neg Y) \cup Y1)))) \cap (SInf \cap X1) \subseteq X1 \cdot Y1$

```

    using 13 14 by blast
show ?thesis
    using 1 12 15 by blast
qed

```

ITL Axioms derived

```

lemma SBoxGen:
  assumes  $X = STrue$ 
  shows  $(SAlways X) = STrue$ 
using assms
by (metis Compl-disjoint2 FusionSFalse double-compl salways-def sfalse-def sfinite-def
    ssometime-def strue-def)

```

```

lemma SBiGen:
  assumes  $X = STrue$ 
  shows  $(SBi X) = STrue$ 
using assms
by (metis double-compl sbi-def sdi-def sfalse-def SFalseFusion strue-def)

```

```

lemma SMP:
  assumes  $X \subseteq Y$ 
  assumes  $X = STrue$ 
  shows  $Y = STrue$ 
using assms using strue-def by blast

```

```

lemma SChopAssoc:
 $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$ 
by (simp add: FusionAssoc)

```

```

lemma SOrChopImp:
 $(X \cup Y) \cdot Z \subseteq (X \cdot Z) \cup (Y \cdot Z)$ 
by (simp add: FusionUnionDistL)

```

```

lemma SChopOrImp:
 $X \cdot (Y \cup Z) \subseteq (X \cdot Y) \cup (X \cdot Z)$ 
by (simp add: FusionUnionDistR)

```

```

lemma SEmptyChop:
 $SEmpty \cdot X = X$ 
by (simp add: FusionSEmptyL)

```

```

lemma SChopEmpty:
 $X \cdot SEmpty = X$ 
by (simp add: FusionSEmptyR)

```

```

lemma SStateImpBi:
 $(SInit X) \subseteq (SBi (SInit X))$ 
by (simp add: ST28 ST38 sbi-def)

```

lemma *SNextImpNotNextNot*:

$(SNext\ X) \subseteq \neg(SNext\ (\neg X))$

proof –

have 1: $((SNext\ X) \subseteq \neg(SNext\ (\neg X))) = (((SNext\ X) \cap (SNext\ (\neg X))) \subseteq SFalse)$

by (*simp add: disjoint-eq-subset-Compl sfalse-def*)

have 2: $((SNext\ X) \cap (SNext\ (\neg X))) = SSkip.(X \cap (\neg X))$

by (*metis CH07 SStar16 inf.orderE snext-def*)

have 3: $(SSkip).(X \cap (\neg X)) = SSkip.SFalse$

by (*simp add: sfalse-def*)

have 4: $SSkip.SFalse = SFalse$

using *FusionSFalse SStar16 SStarSkip sfalse-def sfinite-def* **by** *fastforce*

from 1 2 3 4 **show** *?thesis* **by** *auto*

qed

lemma *SBiBoxChopImpChop*:

$(SBi\ ((\neg X) \cup X1)) \cap (SAlways\ ((\neg Y) \cup Y1)) \cap (X.Y) \subseteq (X1.Y1)$

using *BD45 BD46* **by** *blast*

lemma *SBoxInduct*:

$(SAlways\ (\neg X \cup (SWnext\ X))) \cap X \subseteq (SAlways\ X)$

proof –

have 1: $((SAlways\ (\neg X \cup (SWnext\ X))) \cap X \subseteq (SAlways\ X)) =$

$((SSometime\ (\neg X)) \subseteq ((\neg X) \cup (SSometime\ (X \cap (SNext\ (\neg X))))))$

by (*simp add: salways-def snext-def swnext-def*)

blast

have 2: $((SSometime\ (\neg X)) \subseteq ((\neg X) \cup (SSometime\ (X \cap (SNext\ (\neg X)))))) =$

$(((SSStar\ SSkip).(\neg X) \subseteq ((\neg X) \cup ((SSStar\ SSkip).(X \cap (SSkip.(\neg X)))))))$

by (*simp add: SStarSkip snext-def ssometime-def*)

have 3: $(((SSStar\ SSkip).(\neg X) \subseteq ((\neg X) \cup ((SSStar\ SSkip).(X \cap (SSkip.(\neg X)))))))$

using *SStar51 SStar50* **by** *blast*

from 1 2 3 **show** *?thesis* **by** *auto*

qed

lemma *SChopstarEqv*:

$(SSStar\ X) = SEmpty \cup (X \cap SMore).(SSStar\ X)$

using *SStar52 SStar53* **by** *blast*

2.15.5 Extra Laws

Boolean Laws

lemma *B02*:

assumes $\neg Y \subseteq \neg X$

shows $(X:: 'a\ iintervals) \subseteq Y$

using *assms* **by** *auto*

lemma *B03*:

$((X:: 'a\ iintervals) = Y) \longleftrightarrow (\neg X = \neg Y)$

by *auto*

lemma *B05*:

assumes $(X:: 'a\ iintervals) \cup Y \subseteq Z$
shows $X \subseteq Z \wedge Y \subseteq Z$
using *assms* **by** *auto*

lemma *B06*:
assumes $X \subseteq Z \wedge Y \subseteq Z$
shows $(X:: 'a\ iintervals) \cup Y \subseteq Z$
using *assms* **by** *auto*

lemma *B07*:
 $(X:: 'a\ iintervals) \cup Y \subseteq Z \longleftrightarrow X \subseteq Z \wedge Y \subseteq Z$
by *auto*

lemma *B08*:
assumes $(X:: 'a\ iintervals) \subseteq Y$
shows $\neg X \cup Y = STrue$
using *assms*
using *strue-def* **by** *auto*

lemma *B10*:
 $(X:: 'a\ iintervals) \subseteq Y \longleftrightarrow \neg X \cup Y = STrue$
using *strue-def* **by** *auto*

lemma *B11*:
assumes $(X:: 'a\ iintervals) \subseteq Y$
shows $X \cap \neg Y = SFalse$
using *assms* *sfalse-def* **by** *auto*

lemma *B12*:
assumes $X \cap \neg Y = SFalse$
shows $(X:: 'a\ iintervals) \subseteq Y$
using *assms* *sfalse-def* **by** *auto*

lemma *B13*:
 $(X:: 'a\ iintervals) \subseteq Y \longleftrightarrow X \cap \neg Y = SFalse$
using *sfalse-def* **by** *auto*

lemma *B14*:
assumes $(X:: 'a\ iintervals) \subseteq Y$
shows $X \cap Y = X$
using *assms* **by** *auto*

lemma *B15*:
assumes $(X:: 'a\ iintervals) \subseteq Y \cap Z$
shows $X \subseteq Y \wedge X \subseteq Z$
using *assms* **by** *auto*

lemma *B16*:
assumes $X \subseteq Y \wedge X \subseteq Z$
shows $(X:: 'a\ iintervals) \subseteq Y \cap Z$

using *assms* **by** *auto*

lemma *B17*:

$(X:: 'a\ iintervals) \subseteq Y \cap Z \longleftrightarrow X \subseteq Y \wedge X \subseteq Z$

by *auto*

lemma *B18*:

assumes $(X:: 'a\ iintervals) \subseteq Y \cup Z$

shows $X \cap -Y \subseteq Z$

using *assms* **by** *auto*

lemma *B19*:

assumes $X \cap -Y \subseteq Z$

shows $(X:: 'a\ iintervals) \subseteq Y \cup Z$

using *assms* **by** *auto*

lemma *B21*:

$(X:: 'a\ iintervals) \cup (Y \cap Z) \subseteq (X \cup Y) \cap (X \cup Z) \longleftrightarrow$

$X \cup (Y \cap Z) \subseteq (X \cup Y) \wedge X \cup (Y \cap Z) \subseteq (X \cup Z)$

by *auto*

lemma *B22*:

$(X:: 'a\ iintervals) \cup (Y \cap Z) \subseteq X \cup Y$

by *auto*

lemma *B23*:

$(X:: 'a\ iintervals) \cup (Y \cap Z) \subseteq X \cup Z$

by *auto*

lemma *B24*:

$((X:: 'a\ iintervals) \cup Y) \cap (X \cup Z) \subseteq X \cup (Y \cap Z) \longleftrightarrow$

$((X \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y \cap Z$

by *auto*

lemma *B25*:

$((X:: 'a\ iintervals) \cup Y) \cap (X \cup Z) \cap -X \subseteq Y \cap Z \longleftrightarrow$

$((X \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y \wedge$

$((X \cup Y) \cap (X \cup Z)) \cap -X \subseteq Z$

by *auto*

lemma *B26*:

$((X:: 'a\ iintervals) \cup Y) \cap (X \cup Z) \cap -X \subseteq Y$

by *auto*

lemma *B27*:

$((X:: 'a\ iintervals) \cup Y) \cap (X \cup Z) \cap -X \subseteq Z$

by *auto*

lemma *B29*:

$(X:: 'a\ iintervals) \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$

by *auto*

Chop

lemma *CH02*:

$$X \cdot Y \cap -(X \cdot Z) \subseteq X \cdot (Y \cap -Z)$$

by (*metis B20 FusionRuleR FusionUnionDistR inf.right-idem inf-le1*)

lemma *CH03*:

$$X \cdot Y \cap -(Z \cdot Y) \subseteq (X \cap -Z) \cdot Y$$

by (*metis B20 FusionRuleL FusionUnionDistL inf.right-idem inf-le1*)

lemma *CH04*:

$$X \cdot Y \cap -(X \cdot -Z) \subseteq X \cdot (Y \cap Z)$$

using *CH02* by *fastforce*

lemma *CH05*:

$$X \cdot Y \cap -(-Z \cdot Y) \subseteq (X \cap Z) \cdot Y$$

using *CH03* by *fastforce*

lemma *CH06*:

assumes $X \subseteq X1$

$Y \subseteq Y1$

shows $X \cdot Y \subseteq X1 \cdot Y1$

using *assms*

by (*metis FusionRuleL FusionRuleR order-trans*)

lemma *CH11*:

$$((X \cap (SPower SSkip\ n)) \cdot STrue) \cap ((SPower SSkip\ n) \cdot Y) = (X \cap (SPower SSkip\ n)) \cdot Y$$

using *PwrFusionInterL[of n X STrue STrue Y]*

by (*simp add: inf-commute strue-def*)

lemma *CH12*:

$$(STrue \cdot (X \cap (SPower SSkip\ n))) \cap (Y \cdot (SPower SSkip\ n)) = (Y \cdot (X \cap (SPower SSkip\ n)))$$

using *PwrFusionInterR[of STrue n X Y STrue]*

by (*metis STrueTop inf-commute inf-sup-absorb*)

lemma *CH13*:

$$(SPower SSkip\ n) \cdot (SPower SSkip\ m) = (SPower SSkip\ (n+m))$$

proof

(*induct n arbitrary: m*)

case 0

then show *?case* by (*simp add: FusionSEmptyL*)

next

case (*Suc n*)

then show *?case*

by (*metis FusionAssoc add-Suc pwr-Suc*)

qed

Next

lemma *N01*:

$(SNext\ SEmpty) = SSkip$

by (*simp add: FusionSEmptyR snext-def*)

lemma *N02*:

$(SNext\ SFalse) = SFalse$

by (*metis FusionSFalse SStar16 SStarSkip disjoint-eq-subset-Compl sfalse-def sfinite-def snext-def*)

lemma *N03*:

$(SNext\ X) \cdot Y = (SNext\ (X \cdot Y))$

by (*simp add: snext-def FusionAssoc*)

lemma *N04*:

$(SNext\ (X \cup Y)) = (SNext\ X) \cup (SNext\ Y)$

by (*simp add: FusionUnionDistR snext-def*)

lemma *N05*:

$(SNext\ (X \cap Y)) = (SNext\ X) \cap (SNext\ Y)$

by (*metis CH07 SStar16 inf.orderE snext-def*)

lemma *N06*:

assumes $X \subseteq Y$

shows $(SNext\ X) \subseteq (SNext\ Y)$

using *assms*

by (*metis FusionUnionDistR Subsumption snext-def*)

lemma *N07*:

$(SNext\ ((-X) \cup Y)) = (SNext\ (-X)) \cup (SNext\ Y)$

by (*simp add: N04*)

lemma *N08*:

$SMore \subseteq SSkip \cdot STrue$

using *SMoreImpSSkipFusion* **by** *blast*

lemma *N23*:

$(SWprev\ X) \subseteq (SEmpty \cup (SPrev\ X))$

proof –

have 1: $(X \cap SFinite) \cdot SSkip \cup (-\ X \cap SFinite) \cdot SSkip = SStar\ SSkip \cdot SSkip$

by (*metis B28 FusionUnionDistL SStarSkip STrueTop boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def*)

have 2: $(SEmpty) \cup STrue \cdot SSkip = STrue$

by (*metis FusionSFalse FusionUnionDistL Powerstarhelp2 SStar19 SStarSkip UnfoldL UnionAssoc compl-bot-eq compl-sup-top sfinite-def sinf-def strue-def*)

have 3: $- \ SWprev\ X \cup (SEmpty \cup SPrev\ X) = STrue$

unfolding *supprev-def sprev-def*

by (*metis 2 FusionUnionDistL compl-bot-eq compl-sup-top double-compl inf-sup-aci(7) strue-def*)

then show *?thesis* **by** (*meson B09*)

qed

lemma N24:

$(SEmpty) \subseteq (SWprev X)$

proof –

have 1: $(- SEmpty \cap - (- SEmpty \cdot - SEmpty)) \subseteq SMore$

by (*simp add: smore-def*)

have 2: $-X \cdot SMore \subseteq SMore$

by (*metis CH01 FusionAssoc1 FusionUnionDistL SMoreImpSSkipFusion SSkipFusionImpSMore SStar30 SStar48 STrueTop subset-antisym sup.orderI sup.right-idem*)

have 3: $-X \cdot (- SEmpty \cap - (- SEmpty \cdot - SEmpty)) \subseteq SMore$

using 1 2 *FusionRuleR* **by** *blast*

show *?thesis*

by (*metis 3 compl-le-swap1 compl-sup smore-def sskip-def supprev-def*)

qed

lemma N25:

$(SPrev X) \subseteq (SWprev X)$

proof –

have 1: $((SPrev X) \subseteq (SWprev X)) = (((SPrev X) \cap (SPrev (-X))) \subseteq SFalse)$

by (*simp add: B10 sfalse-def sprevedef supprev-def*)

have 2: $((SPrev X) \cap (SPrev (-X))) = (X \cap (-X)) \cdot SSkip$

by (*metis CH08 SStar16 inf.orderE sprevedef*)

have 3: $(X \cap (-X)) \cdot SSkip = SFalse \cdot SSkip$

by (*simp add: sfalse-def*)

have 4: $SFalse \cdot SSkip = SFalse$

by (*simp add: SFalseFusion*)

from 1 2 3 4 **show** *?thesis* **by** *auto*

qed

lemma N26:

$(SWprev X) = (SEmpty \cup (SPrev X))$

using N23 N24 N25 **by** *blast*

lemma N09:

$SSkip \cup SMore \cdot SSkip \subseteq SMore$

proof –

have 1: $SSkip \subseteq SMore$

by (*simp add: smore-def sskip-def*)

have 2: $SMore \cdot SSkip \subseteq SMore$

by (*metis N26 UnionCommute compl-le-swap1 smore-def sup-ge2 supprev-def*)

from 1 2 **show** *?thesis* **by** *auto*

qed

lemma N10:

assumes $SSkip \cup SMore \cdot SSkip \subseteq SMore$

shows $SSkip \cdot (SStar SSkip) \subseteq SMore$

using *assms SStarInductR N09* **by** *blast*

lemma N11:

$SSkip \cdot STrue \subseteq SMore$

by (*simp add: SSkipFusionImpSMore*)

lemma N12:

$$(SNext\ X) = -(SWnext\ (-X))$$

by (*simp add: snext-def swnext-def*)

lemma N13:

$$SMore \cdot STrue = SMore$$

by (*metis CH01 FusionAssoc1 SMoreImpSSkipFusion SSkipFusionImpSMore subset-antisym*)

lemma N14:

$$STrue \cdot SSkip \subseteq SMore$$

by (*metis N09 ST47 STrueTop smore-def*)

lemma N15:

$$SMore \subseteq STrue \cdot SSkip$$

proof –

$$\text{have } 1: SMore \subseteq SSkip \cdot STrue$$

by (*simp add: SMoreImpSSkipFusion*)

$$\text{have } 2: SSkip \cdot STrue \subseteq STrue \cdot SSkip$$

proof –

$$\text{have } f3: SSkip \cdot SFinite \subseteq STrue \cdot SSkip$$

by (*metis B19 FusionRuleL SStar19 SStarSkip STrueTop inf-sup-ord(2)*)

$$\text{have } SSkip \cdot SInf \subseteq STrue \cdot SSkip$$

by (*metis (no-types, opaque-lifting) B04 Compl-disjoint2 FusionRuleR SChopAssoc SMoreImpSSkipFusion*

*SSkipFusionImpSMore ST47 boolean-algebra-cancel.inf0 compl-inf double-compl
inf-commute inf-le1 sfalse-def sinf-def strue-def*)

then show *?thesis*

using *f3* **by** (*metis (no-types) FusionUnionDistR STrueTop le-sup-iff sfinite-def*)

qed

show *?thesis*

using *2 SMoreImpSSkipFusion* **by** *blast*

qed

lemma N19:

$$(SWnext\ X) \subseteq (SEmpty \cup (SNext\ X))$$

proof –

$$\text{have } STrue \subseteq SSkip \cdot (-X) \cup (SEmpty \cup SSkip \cdot X)$$

by (*metis B04 FusionUnionDistR N13 SMoreImpSSkipFusion SSkipFusionImpSMore SStar48 UnfoldL
compl-bot-eq compl-sup-top inf-sup-aci(7) strue-def*)

then show *?thesis*

by (*simp add: B13 snext-def strue-def swnext-def*)

qed

lemma N20:

$$(SEmpty) \subseteq (SWnext\ X)$$

proof –

$$\text{have } 1: ((SEmpty) \subseteq (SWnext\ X)) = (-(SWnext\ X) \subseteq SMore)$$

by (*simp add: smore-def*)

$$\text{have } 2: (-(SWnext\ X) \subseteq SMore) = ((SNext\ (-X)) \subseteq SMore)$$

by (simp add: snext-def swnext-def)
 have 3: (SNext (−X)) ⊆ SSkip·STrue
 by (metis FusionUnionDistR STrueTop snext-def sup.orderI sup.right-idem)
 hence 4: (SNext (−X)) ⊆ SMore using SSkipFusionImpSMore by auto
 from 1 2 4 show ?thesis by auto
 qed

lemma N21:

(SEmpty ∪ (SNext X)) ⊆ (SWnext X)
 using N20
 by (metis B06 SNextImpNotNextNot snext-def swnext-def)

lemma N22:

(SWnext X) = (SEmpty ∪ (SNext X))
 using N21 N19 by blast

lemma N16:

(SNext X) = SMore ∩ (SWnext X)
 using N12 N22 smore-def by blast

lemma N17:

(SWnext (X ∩ Y)) = (SWnext X) ∩ (SWnext Y)
 by (simp add: N05 N22 Un-Int-distrib)

lemma N18:

(SWnext (X ∪ Y)) = (SWnext X) ∪ (SWnext Y)
 by (simp add: swnext-def)
 (metis CH07 SSkipSFinite compl-inf)

lemma N27:

(SNext ((−X) ∪ Y)) ⊆ (−(SNext X) ∪ (SNext Y))
 using N04 SNextImpNotNextNot by blast

lemma N28:

(SPrev ((−X) ∪ Y)) ⊆ (−(SPrev X) ∪ (SPrev Y))
 unfolding sprev-def
 proof −
 have ∧I. (SEmpty) ∪ I · SSkip = − (− I · SSkip)
 using N26 sprev-def swprev-def by blast
 then have (Y ∪ − X) · SSkip ⊆ Y · SSkip ∪ − (X · SSkip)
 using FusionUnionDistL by blast
 then show (− X ∪ Y) · SSkip ⊆ − (X · SSkip) ∪ Y · SSkip
 by (simp add: UnionCommute)
 qed

lemma N29:

(SPrev X) = −(SWprev (−X))
 by (simp add: sprev-def swprev-def)

SInit

lemma *ST01*:

$$(X \cap S\text{Empty}) \cdot Y \subseteq Y$$

by (*metis Int-commute FusionUnionDistL FusionSEmptyL le-iff-sup sup-inf-absorb UnionCommute*)

lemma *ST02*:

$$(X \cap S\text{Empty}) \cdot Y \subseteq (X \cap S\text{Empty}) \cdot S\text{True}$$

by (*simp add: FusionRuleR strue-def*)

lemma *ST03*:

$$(X \cap S\text{Empty}) \cdot (X \cap S\text{Empty}) \subseteq (X \cap S\text{Empty})$$

using *ST01* **by** *auto*

lemma *ST04*:

$$(X \cap S\text{Empty}) \subseteq (X \cap S\text{Empty}) \cdot (X \cap S\text{Empty})$$

proof –

$$\textbf{have } \forall S Sa Sb Sc. (Sc) \cdot (Sb \cap S\text{Empty}) \cap (Sa \cdot (S \cap S\text{Empty})) = Sc \cap Sa \cdot (Sb \cap (S \cap S\text{Empty}))$$

by (*simp add: CH10 inf-assoc*)

$$\textbf{then have } \forall S Sa. (Sa) \cdot (S \cap S\text{Empty}) \cdot (S \cap S\text{Empty}) = Sa \cdot (S \cap S\text{Empty})$$

by (*metis FusionSEmptyR inf.idem inf-commute*)

then show *?thesis*

by (*metis FusionSEmptyL subset-refl*)

qed

lemma *ST05*:

$$(X \cap S\text{Empty}) \subseteq -((-X) \cap S\text{Empty})$$

by *blast*

lemma *ST06*:

$$((-X) \cap S\text{Empty}) \subseteq -(X \cap S\text{Empty})$$

by *auto*

lemma *ST07*:

$$(X \cap S\text{Empty}) \cap (Y \cap S\text{Empty}) \subseteq (X \cap S\text{Empty}) \cdot S\text{True}$$

using *ST02 FusionSEmptyR* **by** *blast*

lemma *ST08*:

$$(X \cap S\text{Empty}) \cap (Y \cap S\text{Empty}) \subseteq (S\text{True} \cap S\text{Empty}) \cdot (Y \cap S\text{Empty})$$

by (*metis FusionSEmptyL FusionSEmptyR ST33 inf.cobounded2*)

lemma *ST09*:

$$((X \cap S\text{Empty}) \cdot S\text{True}) \cap (S\text{True} \cap S\text{Empty}) \cdot (Y \cap S\text{Empty}) \subseteq (X \cap S\text{Empty}) \cdot (Y \cap S\text{Empty})$$

by (*metis compl-bot-eq eq-refl FusionSEmptyR inf commute inf-top.left-neutral CH09 strue-def*)

lemma *ST10*:

$$(X \cap S\text{Empty}) \cdot (Y \cap S\text{Empty}) \subseteq (X \cap S\text{Empty})$$

by (*metis FusionRuleR FusionSEmptyR inf-le2*)

lemma *ST11*:

$$(X \cap S\text{Empty}) \cdot (Y \cap S\text{Empty}) \subseteq (Y \cap S\text{Empty})$$

using *ST01* **by** *blast*

lemma *ST12*:

$(X \cap SEmpty) \cap (Y \cap SEmpty) = (X \cap SEmpty) \cdot SEmpty \cap (Y \cap SEmpty) \cdot SEmpty$
by (*simp add: FusionSEmptyR*)

lemma *ST14*:

$((X \cap Y) \cap SEmpty) \cdot SEmpty = ((X \cap Y) \cap SEmpty)$
by (*simp add: FusionSEmptyR*)

lemma *ST16*:

$(X \cap SEmpty) \cap (Y \cap SEmpty) \subseteq SEmpty$
by (*simp add: le-infI2*)

lemma *ST17*:

$(X \cap SEmpty) \cdot (Y \cap SEmpty) \subseteq SEmpty$
using *ST10* **by** *auto*

lemma *ST18*:

$\neg((X \cap SEmpty) \cup (Y \cap SEmpty)) = \neg(X \cap SEmpty) \cap \neg(Y \cap SEmpty)$
by *auto*

lemma *ST19*:

$(X \cap SEmpty) \cdot (\neg X \cap SEmpty) \subseteq (X \cap SEmpty)$
using *ST10* **by** *blast*

lemma *ST20*:

$(X \cap SEmpty) \cdot (\neg X \cap SEmpty) \subseteq (\neg X \cap SEmpty)$
using *ST01* **by** *auto*

lemma *ST22*:

$((X \cap SEmpty) \cdot SSkip) \cdot (Y \cap SEmpty) \subseteq (X \cap SEmpty) \cdot SSkip$
using *FusionRuleR FusionSEmptyR* **by** *blast*

lemma *ST23*:

$((X \cap SEmpty) \cdot SSkip) \cdot (Y \cap SEmpty) \subseteq SSkip \cdot (Y \cap SEmpty)$
by (*simp add: ST01 FusionRuleL*)

lemma *ST27*:

$(SInit X) \cap (Y \cdot Z) \subseteq ((SInit X) \cap Y) \cdot Z$
by (*metis B04 compl-bot-eq FusionAssoc FusionSEmptyL inf-commute inf-top.left-neutral CH09 sinit-def strue-def*)

lemma *ST29*:

$(SInit X) \cdot Y \subseteq (SInit X)$
using *ST02 FusionAssoc sinit-def* **by** *fastforce*

lemma *ST30*:

$(SInit X) \cap (SDi Y) = (SDi ((SInit X) \cap Y))$
unfolding *sdi-def sinit-def strue-def*

by (metis CH09 FusionAssoc FusionSEmptyL compl-bot-eq inf-top.left-neutral)

lemma ST31:

$(X \cdot (STrue \cap SEmpty)) \cap (STrue \cdot (Y \cap SEmpty)) = X \cdot (Y \cap SEmpty)$

by (metis Int-commute compl-bot-eq inf-top.right-neutral CH10 strue-def)

lemma ST32:

$(STrue \cap SEmpty) \cdot SEmpty \cap (SInit X) = (X \cap SEmpty)$

proof –

have $\forall S Sa. (Sa) \cap (S \cap Sa) = S \cap Sa$

by *fastforce*

then have $f1: \forall S Sa. (Sa) \cap (S \cap SEmpty) \subseteq Sa \cap SEmpty \cdot STrue$

by (metis (no-types) ST07 inf-assoc)

have $f4: \forall S. - (- (S::'a iintervals)) = S$

by *blast*

have $f6: \forall S. (SEmpty) \cap (S \cap (S \cap SEmpty \cdot STrue)) = S \cap SEmpty$

using $f1$ **by** *auto*

have $f7: \forall S. (SEmpty) \cap (SEmpty \cap S \cdot STrue) \subseteq S$

using $f4$ **by** (metis CH11 FusionSEmptyR inf-aci(1) inf-le2 pwr-0)

have $\forall S. (SEmpty) \cap (S \cap (SEmpty \cap S \cdot STrue)) = SEmpty \cap S$

using $f6$ **by** (simp add: inf-commute)

then have $SEmpty \cap (SEmpty \cap X \cdot STrue) = SEmpty \cap X$

using $f7$ **by** *auto*

then show *?thesis*

by (metis (no-types) ST33 inf-commute sinit-def)

qed

lemma ST34:

$((X \cap SEmpty) \cdot Y) = (SInit X) \cap Y$

by (metis FusionSEmptyL Int-commute CH09 compl-bot-eq inf-top-right sinit-def strue-def)

lemma ST35:

$((SInit X) \cap Y) \cdot Z \subseteq (SInit X) \cap (Y \cdot Z)$

by (metis B04 ST34 FusionAssoc)

lemma ST39:

$SEmpty \cap (SInit X) \subseteq (X \cap SEmpty)$

using ST32 **by** *blast*

lemma ST40:

$(X \cap SEmpty) \subseteq SEmpty \cap (SInit X)$

using ST32 **by** *auto*

lemma ST41:

$SEmpty \cap (SInit X) = (X \cap SEmpty)$

using ST40 ST39 **by** *auto*

lemma ST42:

$(X \cap SEmpty) \subseteq ((X \cup Y) \cap SEmpty)$

by *blast*

lemma *ST43*:

$$(Y \cap SEmpty) \subseteq ((X \cup Y) \cap SEmpty)$$

by *blast*

lemma *ST44*:

$$(X \cap SEmpty) \cap ((-X) \cap SEmpty) = SFalse$$

by (*simp add: sfalse-def*)

lemma *ST45*:

$$((X \cup Y) \cap SEmpty) \subseteq (X \cap SEmpty) \cup (Y \cap SEmpty)$$

by *auto*

lemma *ST46*:

$$(SInit X) \cup (SInit Y) = (SInit (X \cup Y))$$

by (*simp add: Int-Un-distrib2 FusionUnionDistL sinit-def*)

lemma *ST48*:

$$-(STrue \cdot (X \cap SEmpty)) \subseteq STrue \cdot ((-X) \cap SEmpty)$$

by (*metis B09 FusionSEmptyR FusionUnionDistR ST21 double-compl*)

SStar

lemma *SStar02*:

assumes $X \subseteq Y$

shows $X \cdot (SStar Y) \cup SEmpty \subseteq (SStar Y)$

using *assms UnfoldL[of Y]*

by (*metis B05 B06 FusionRuleL Subsumption sup.cobounded2*)

lemma *SStar04*:

$$(SStar X) \subseteq (SStar X) \cdot (SStar X)$$

by (*metis Un-absorb UnfoldL UnionAssoc ST47 sup.absorb-iff2*)

lemma *SStar09*:

assumes $(X \cdot (SEmpty \cup (X \cdot (SStar X)))) \cup SEmpty \subseteq (SEmpty \cup (X \cdot (SStar X)))$

shows $(SStar X) \subseteq SEmpty \cup (X \cdot (SStar X))$

using *assms*

by (*simp add: UnfoldL*)

lemma *SStar10*:

$$(X \cdot (SEmpty \cup (X \cdot (SStar X)))) \subseteq (SEmpty \cup (X \cdot (SStar X)))$$

by (*metis UnfoldL sup-ge2*)

lemma *SStar11*:

$$SEmpty \subseteq (SEmpty \cup (X \cdot (SStar X)))$$

by *auto*

lemma *SStar13*:

$$(SStar SSkip) = SFinite$$

by (*simp add: SStarSkip*)

lemma *SStar14*:

$(SSometime\ X) = (SStar\ SSkip) \cdot X$

by (*simp add: SStarSkip ssometime-def*)

lemma *SStar20*:

$(SStar\ SEmpty) = SEmpty$

by (*metis FusionSEmptyR ST15 ST33*)

lemma *SStar21*:

$(SStar\ (SEmpty \cap X)) \cdot (SEmpty \cap X) = (SEmpty \cap X)$

by (*metis ST15 FusionSEmptyL inf-commute*)

lemma *SStar24*:

$(SStar\ SFalse) = SEmpty$

by (*metis SStar20 SStar47 inf-compl-bot sfalse-def smore-def*)

lemma *SStar26*:

$X \subseteq (SStar\ X)$

using *UnfoldL[of X]*

by (*metis B05 FusionSEmptyR FusionUnionDistR SStar10*)

lemma *SStar27*:

$SEmpty \subseteq (SStar\ X)$

using *UnfoldL* **by** *blast*

lemma *SStar31*:

assumes $X \cup (X \cdot Y) \cdot (X \cdot (SStar\ (Y \cdot X))) \subseteq X \cdot (SStar\ (Y \cdot X))$

shows $(SStar\ (X \cdot Y)) \cdot X \subseteq X \cdot (SStar\ (Y \cdot X))$

using *assms SStarInductL* **by** *blast*

lemma *SStar32*:

$X \cup (X \cdot Y) \cdot (X \cdot (SStar\ (Y \cdot X))) \subseteq X \cdot (SStar\ (Y \cdot X))$

by (*metis B06 SStar10 SStar11 FusionAssoc FusionRuleR FusionSEmptyR UnfoldL*)

lemma *SStar33*:

$(SStar\ (X \cdot Y)) \cdot X \subseteq X \cdot (SStar\ (Y \cdot X))$

by (*meson SStar32 SStarInductL*)

lemma *SStar37*:

assumes $X \cdot Z \subseteq Z \cdot Y$

shows $(SStar\ X) \cdot Z \subseteq Z \cdot (SStar\ Y)$

proof –

have $Z \cdot SStar\ Y = Z \cdot SEmpty \cup Z \cdot (Y \cdot SStar\ Y)$

by (*metis FusionUnionDistR UnfoldL*)

then have $Z \cdot SStar\ Y \cup (Z \cup X \cdot (Z \cdot SStar\ Y)) = Z \cup Z \cdot Y \cdot SStar\ Y \cup X \cdot Z \cdot SStar\ Y$

using *FusionAssoc FusionSEmptyR* **by** *blast*

then have $Z \cdot SStar\ Y \cup (Z \cup X \cdot (Z \cdot SStar\ Y)) = Z \cdot SStar\ Y$

by (*metis (no-types) FusionAssoc FusionSEmptyR FusionUnionDistL FusionUnionDistR UnfoldL UnionAssoc*)

```

    assms sup.absorb-iff1)
then show ?thesis
  by (meson SStarInductL sup.absorb-iff1)
qed

lemma SStar38:
  assumes  $Z \cdot X \subseteq Y \cdot Z$ 
  shows  $Z \cdot (SStar\ X) \subseteq (SStar\ Y) \cdot Z$ 
using assms
proof –
have f1:  $Z \cup SStar\ Y \cdot Y \cdot Z = SStar\ Y \cdot Z$ 
  by (metis (no-types) SStar30 ST47 UnfoldL)
have  $SStar\ Y \cdot Y \cdot Z = SStar\ Y \cdot Z \cdot X \cup SStar\ Y \cdot Y \cdot Z$ 
  by (metis FusionAssoc FusionUnionDistR assms subset-Un-eq)
then have  $Z \cup SStar\ Y \cdot Z \cdot X \subseteq SStar\ Y \cdot Z$ 
  using f1 by blast
then show ?thesis
  by (simp add: SStarInductR)
qed

lemma SStar39:
   $Y \cdot (SStar\ ((SStar\ X) \cdot Y)) \subseteq (SStar\ (Y \cdot (SStar\ X))) \cdot Y$ 
by (simp add: SStar38 FusionAssoc)

lemma SStar40:
   $(SStar\ (Y \cdot (SStar\ X))) \cdot Y \subseteq Y \cdot (SStar\ ((SStar\ X) \cdot Y))$ 
by (simp add: SStar33)

lemma SStar41:
   $Y \cdot (SStar\ ((SStar\ X) \cdot Y)) = (SStar\ (Y \cdot (SStar\ X))) \cdot Y$ 
using SStar39 SStar40 by blast

lemma SStar42:
   $Z \cdot (SStar\ (Y \cdot Z)) \subseteq (SStar\ (Z \cdot Y)) \cdot Z$ 
by (simp add: SStar38 FusionAssoc)

lemma SStar43:
   $(SStar\ (Z \cdot Y)) \cdot Z \subseteq Z \cdot (SStar\ (Y \cdot Z))$ 
by (simp add: SStar33)

lemma SStar44:
   $Z \cdot (SStar\ (Y \cdot Z)) = (SStar\ (Z \cdot Y)) \cdot Z$ 
using SStar42 SStar43 by blast

lemma SStar49:
   $(SStar\ X) = SEmpty \cup (SStar\ X) \cdot X$ 
using SStar30 UnfoldL by blast

```

Box and Diamond

lemma *BD01*:

$(SSometime\ SEmpty) = SFinite$

by (*simp add: ssometime-def FusionSEmptyR*)

lemma *BD02*:

$X \subseteq (SSometime\ X)$

unfolding *ssometime-def*

by (*metis SStar15 SStarSkip ST47 subset-Un-eq*)

lemma *BD03*:

$(SNext\ (SSometime\ X)) \subseteq (SSometime\ X)$

by (*metis FusionUnionDistL N03 SStar16 SStar03 SStar04 SStarSkip snext-def ssometime-def sup.absorb-iff2 sup.orderE*)

lemma *BD04*:

$(SSometime\ (SNext\ X)) \subseteq (SSometime\ X)$

by (*metis BD03 FusionAssoc SStar28 SStar29 SStarSkip inf-sup-aci(5) snext-def ssometime-def sup.orderE*)

lemma *BD05*:

$(SSometime\ X) \cup (SSometime\ Y) = (SSometime\ (X \cup Y))$

by (*simp add: FusionUnionDistR ssometime-def*)

lemma *BD06*:

$(SSometime\ STrue) = STrue$

using *BD02 SMP* **by** *blast*

lemma *BD07*:

$(SSometime\ (X \cap Y)) \subseteq (SSometime\ X) \cap (SSometime\ Y)$

by (*simp add: FusionRuleR ssometime-def*)

lemma *BD08*:

$(SAlways\ STrue) = STrue$

by (*simp add: SBoxGen*)

lemma *BD09*:

$\neg(SAlways\ X) = (SSometime\ (\neg X))$

by (*simp add: salways-def*)

lemma *BD10*:

$(SAlways\ X) \subseteq (SSometime\ X)$

by (*metis B02 BD02 BD09 set-rev-mp subsetI*)

lemma *BD11*:

$(SSometime\ (SSometime\ X)) = (SSometime\ X)$

by (*metis FusionAssoc SStar03 SStar04 SStarSkip ssometime-def subset-antisym*)

lemma *BD12*:

$(SAlways\ X) \subseteq X$

by (*simp add: B02 BD02 BD09*)

lemma *BD13*:

$(SDi\ STrue) = STrue$

by (*simp add: CH01 sdi-def*)

lemma *BD14*:

$(SDi\ SEmpty) = STrue$

by (*simp add: sdi-def FusionSEmptyL*)

lemma *BD15*:

$(SBi\ STrue) = STrue$

by (*simp add: SBiGen*)

lemma *BD16*:

$(SDi\ (X \cup Y)) = (SDi\ X) \cup (SDi\ Y)$

by (*simp add: FusionUnionDistL sdi-def*)

lemma *BD17*:

assumes $X \subseteq Y$

shows $(SDi\ X) \subseteq (SDi\ Y)$

using *assms*

by (*metis FusionUnionDistL Subsumption sdi-def*)

lemma *BD18*:

$(SDi\ (SDi\ X)) = (SDi\ X)$

by (*metis CH01 FusionAssoc sdi-def*)

lemma *BD19*:

$(SDa\ SEmpty) = STrue$

by (*metis BD01 BD06 sda-def ssometime-def*)

lemma *BD20*:

$(SDa\ STrue) = STrue$

by (*metis BD06 CH01 sda-def ssometime-def*)

lemma *BD21*:

$(SBa\ STrue) = STrue$

by (*metis BD15 BD08 BD09 sba-def sbi-def sda-def sdi-def ssometime-def*)

lemma *BD22*:

$(SDa\ (X \cup Y)) = (SDa\ X) \cup (SDa\ Y)$

by (*simp add: FusionUnionDistL FusionUnionDistR sda-def*)

lemma *BD23*:

assumes $X \subseteq Y$

shows $(SDa\ X) \subseteq (SDa\ Y)$

using *assms*

by (*metis BD22 Subsumption*)

lemma *BD24*:

assumes $X \subseteq Y$

shows $(SDa (-Y)) \subseteq (SDa (-X))$
using *assms*
by (*simp add: BD23*)

lemma *BD25*:
 $(SDi X) \subseteq (SDa X)$
by (*metis BD02 FusionAssoc sda-def sdi-def ssometime-def*)

lemma *BD26*:
 $(SSometime X) \subseteq (SDa X)$
by (*metis B08 B12 FusionRuleR FusionSEmptyR SFalseBottom double-compl inf.absorb-iff2 inf-le1 sda-def ssometime-def sup-inf-absorb*)

lemma *BD27*:
 $(SBa X) \subseteq (SBi X)$
by (*simp add: BD25 sba-def sbi-def*)

lemma *BD28*:
 $(SBa X) \subseteq (SAlways X)$
by (*simp add: B02 BD26 BD09 sba-def*)

lemma *BD29*:
 $(SAlways X) \cap (SAlways Y) = (SAlways (X \cap Y))$
proof –
have 1: $(SAlways X) \cap (SAlways Y) = - SSometime (-X) \cap - SSometime (-Y)$
by (*simp add: salways-def*)
have 2: $SSometime (-X) \cup SSometime (-Y) = SSometime(-X \cup -Y)$
using *BD05 by blast*
have 3: $- SSometime(-X \cup -Y) = (SAlways (X \cap Y))$
by (*simp add: salways-def*)
show ?thesis **using** 1 2 3 **by** *auto*
qed

lemma *BD30*:
 $(SAlways X) \cup (SAlways Y) \subseteq (SAlways (X \cup Y))$
using *BD07*
by (*metis B02 BD09 compl-sup*)

lemma *BD31*:
 $(SDi (X \cap Y)) \subseteq (SDi X) \cap (SDi Y)$
by (*simp add: BD17*)

lemma *BD32*:
 $(SBi X) \cup (SBi Y) \subseteq (SBi (X \cup Y))$
using *BD31*
by (*metis (mono-tags, lifting) B02 compl-sup double-compl sbi-def*)

lemma *BD33*:
 $(SDa (X \cap Y)) \subseteq (SDa X) \cap (SDa Y)$
by (*simp add: BD23*)

lemma *BD34*:

$(SBa\ X) \cup (SBa\ Y) \subseteq (SBa\ (X \cup Y))$

using *BD33*

by (*metis* (*mono-tags*, *lifting*) *B02 compl-sup double-compl sba-def*)

lemma *BD35*:

$(SAlways\ SEmpty) = SEmpty$

by (*metis* *B08 BD08 BD12 FusionSEmptyR N20 SBoxInduct ST33 sup.absorb-iff2 sup.orderE*)

lemma *BD36*:

$(SBi\ SEmpty) = SEmpty$

proof –

have 1: $-(- SEmpty \cdot STrue) \subseteq SEmpty$

by (*metis* *B04 N13 double-compl smore-def*)

have 2: $SEmpty \subseteq -(- SEmpty \cdot STrue)$

using *N13 smore-def* **by** *fastforce*

show *?thesis*

by (*simp add: 1 2 B04 sbi-def sdi-def*)

qed

lemma *BD37*:

$(SBa\ SEmpty) = SEmpty$

by (*metis* *BD09 BD35 BD36 sba-def sbi-def sda-def sdi-def ssometime-def*)

lemma *BD38*:

assumes $X \subseteq Y$

shows $(SAlways\ X) \subseteq (SAlways\ Y)$

using *assms*

by (*simp add: BD29 inf.absorb-iff2*)

lemma *BD39*:

assumes $X \subseteq Y$

shows $(SBi\ X) \subseteq (SBi\ Y)$

using *assms*

by (*simp add: BD17 sbi-def*)

lemma *BD40*:

assumes $X \subseteq Y$

shows $(SBa\ X) \subseteq (SBa\ Y)$

using *assms*

by (*simp add: BD24 sba-def*)

lemma *BD41*:

$(SBi\ (SBi\ X)) = (SBi\ X)$

by (*simp add: BD18 sbi-def*)

lemma *BD42*:

$(SAlways\ (SAlways\ X)) = (SAlways\ X)$

by (*simp add: BD11 salways-def*)

lemma *BD43*:

$$(SDa (SDa X)) = (SDa X)$$

by (*metis BD11 CH01 FusionAssoc sda-def ssometime-def*)

lemma *BD44*:

$$(SBa (SBa X)) = (SBa X)$$

by (*simp add: BD43 sba-def*)

lemma *BD47*:

$$(SAlways ((-X) \cup Y)) \subseteq ((- (SAlways X)) \cup (SAlways Y))$$

by (*metis B20 BD12 BD29 BD38 BD42 double-compl*)

lemma *BD48*:

$$(SAlways X) \subseteq X \cap (SNext (SAlways X))$$

by (*metis B02 B16 BD03 BD09 BD12 N12 salways-def*)

lemma *BD49*:

$$(SBi ((-X) \cup Y)) \subseteq ((- (SBi X)) \cup (SBi Y))$$

by (*metis B20 BD45 Un-commute double-complement sbi-def sdi-def*)

lemma *BD50*:

$$(SPrev (SDi X)) \subseteq (SDi X)$$

by (*simp add: FusionAssoc1 FusionRuleR sdi-def sprev-def strue-elim subsetI*)

lemma *BD51*:

$$-(SBi X) = (SDi (-X))$$

by (*simp add: sbi-def*)

lemma *BD52*:

$$X \subseteq (SDi X)$$

by (*metis FusionSEmptyR FusionUnionDistR ST33 Subsumption UnionCommute sdi-def sup-inf-absorb*)

lemma *BD53*:

$$(SBi X) \subseteq X$$

by (*simp add: B02 BD51 BD52*)

lemma *BD54*:

$$(SBi X) \subseteq X \cap (SNext (SBi X))$$

by (*metis B02 B16 BD50 BD51 BD53 N29 sbi-def*)

lemma *BD55*:

$$(SBi (SMore \cup X)) = (SInit X)$$

by (*metis (no-types, lifting) ST38 compl-sup double-complement inf-commute sbi-def sdi-def
sinit-def smore-def*)

lemma *BD56*:

$$(SAlways (SMore \cup X)) = STrue \cdot (X \cap SEmpty)$$

proof –

have 1: $((SFinite \cdot (X \cap SEmpty)) \cup SInf) = (STrue \cdot (X \cap SEmpty))$

by (metis Powerstarhelp2 Powerstarhelp3 UnionCommute boolean-algebra-cancel.inf0 compl-bot-eq
 inf-commute strue-def)
have 11: $SInf \cup SFinite \cdot (SEmpty \cap X \cup SEmpty \cap - X) = STrue$
 by (simp add: B28 FusionSEmptyR sfinite-def strue-def)
have 2: $(SAlways (SMore \cup X)) \cap SFinite \subseteq SFinite \cdot (X \cap SEmpty)$
 using 11
 by (simp add: B09 BD09 FusionUnionDistR UnionCommute inf-commute inf-sup-aci(7)
 smore-def ssometime-def sfinite-def)
have 3: $(SAlways (SMore \cup X)) \subseteq ((SFinite \cdot (X \cap SEmpty)) \cup SInf)$
 using 2 sfinite-def by fastforce
have 4: $(SAlways (SMore \cup X)) \subseteq STrue \cdot (X \cap SEmpty)$
 using 1 3 by blast
have 5: $SFinite \cdot (X \cap SEmpty) \subseteq (SAlways (SMore \cup X))$
 unfolding salways-def ssometime-def smore-def
 using CH10[of SFinite X SFinite] FusionSFalse
 by (metis (no-types, opaque-lifting) SFalseBottom compl-sup disjoint-eq-subset-Compl double-compl
 inf-commute inf-le2 sfinite-def)
have 6: $SInf \subseteq (SAlways (SMore \cup X))$
 by (metis B05 BD30 FusionSEmptyR double-compl salways-def sfinite-def smore-def ssometime-def)
show ?thesis
 using 1 3 5 6 by auto
qed

lemma BD57:

$(SAlways H) \subseteq SAlways (-G \cup ((SAlways H) \cap G))$
proof –
have 1: $(SAlways H) \subseteq (-G \cup ((SAlways H) \cap G))$
 by blast
have 2: $SAlways (SAlways H) \subseteq SAlways (-G \cup ((SAlways H) \cap G))$
 by (simp add: 1 BD38)
have 3: $SAlways (SAlways H) = (SAlways H)$
 by (simp add: BD42)
show ?thesis using 2 3 by blast
qed

lemma BD58:

$((SAlways H) \cap (F \cdot G)) \subseteq (F \cdot ((SAlways H) \cap G))$
proof –
have 1: $SAlways (-G \cup ((SAlways H) \cap G)) \cap (F \cdot G) \subseteq (F \cdot ((SAlways H) \cap G))$
 using BD46 by blast
show ?thesis using 1 BD57 by blast
qed

Finite and Infinite

lemma FI01:

$SFinite \cdot SFinite = SFinite$
 by (metis BD01 BD11 ssometime-def)

lemma FI02:

$(X \cap SFinite) \cdot (Y \cap SFinite) \subseteq (X \cdot Y) \cap SFinite$
by (*metis B16 CH06 FI01 inf.cobounded1 inf-le2*)

lemma FI03:

$(X \cdot Y) \cap SFinite \subseteq (X \cap SFinite) \cdot (Y \cap SFinite)$

proof –

have 1: $(X \cdot Y) = (X \cap SFinite) \cdot (Y \cap SFinite) \cup (X \cap SFinite) \cdot (Y \cap SInf)$
 $\cup (X \cap SInf) \cdot (Y \cap SFinite) \cup (X \cap SInf) \cdot (Y \cap SInf)$

by (*metis FusionUnionDistR Powerstarhelp2 Powerstarhelp3 SInfSFinite UnionCommute sup.right-idem*)

have 2: $((X \cap SFinite) \cdot (Y \cap SFinite)) \cap SFinite \subseteq (X \cap SFinite) \cdot (Y \cap SFinite)$

by *auto*

have 3: $(X \cap SFinite) \cdot (Y \cap SInf) \cap SFinite \subseteq (X \cap SFinite) \cdot (Y \cap SFinite)$

by (*metis (no-types, lifting) B20 FusionAssoc FusionSFalse inf-le2 sfinite-def subset-trans sup-ge1*)

have 4: $(X \cap SInf) \cdot (Y \cap SFinite) \cap SFinite \subseteq (X \cap SFinite) \cdot (Y \cap SFinite)$

using *Powerstarhelp2 sfinite-def* **by** *fastforce*

have 5: $(X \cap SInf) \cdot (Y \cap SInf) \cap SFinite \subseteq (X \cap SFinite) \cdot (Y \cap SFinite)$

by (*metis 4 Powerstarhelp2*)

have 6: $(X \cdot Y) \cap SFinite = ((X \cap SFinite) \cdot (Y \cap SFinite) \cap SFinite) \cup$
 $((X \cap SFinite) \cdot (Y \cap SInf) \cap SFinite) \cup$
 $((X \cap SInf) \cdot (Y \cap SFinite) \cap SFinite) \cup$
 $((X \cap SInf) \cdot (Y \cap SInf) \cap SFinite)$

using 1 **by** *auto*

from 6 2 3 4 5 **show** *?thesis* **by** *auto*

qed

lemma FI04:

$(X \cap SFinite) \cdot (Y \cap SFinite) = (X \cdot Y) \cap SFinite$

using *FI03 FI02* **by** *fastforce*

lemma FI05:

$SAlways SMore \subseteq SInf$

by (*metis B04 BD56 Compl-disjoint2 sfalse-def sinf-def smore-def sup.orderE*)

lemma FI06:

$SEmpty \subseteq SFinite$

using *BD01 BD02* **by** *auto*

lemma FI07:

$SSkip \cdot SFinite \subseteq SFinite$

using *SStarSkip UnfoldL* **by** *fastforce*

lemma FI08:

$SFinite \cdot SSkip \subseteq SFinite$

by (*metis FI07 SStar19 SStarSkip*)

lemma FI09:

$SFinite \cdot SSkip = SFinite \cap SMore$

proof –

have 1: $SFinite \cdot SSkip \subseteq SFinite \cap SMore$

by (*metis B16 FI07 N09 N10 SStar19 SStarSkip*)

have 2: $SFinite \cap SMore \subseteq SFinite.SSkip$
by (metis B20 FI01 FI02 SStar19 SStarSkip UnfoldL inf.idem smore-def)
show ?thesis **using** 1 2 **by** blast
qed

lemma FI10:
 $SFinite.SSkip = SSkip.SFinite$
by (metis SStar19 SStarSkip)

lemma FI11:
 $SFinite \cap SEmpty = SEmpty$
by (simp add: FI06 Int-absorb2 inf-sup-aci(1))

lemma FI12:
 $SInf.SInf = SInf$
by (metis FusionSFalse Powerstarhelp2 sinf-def)

lemma FI13:
 $SFinite.SInf = SInf$
by (metis BD06 FusionAssoc1 sinf-def ssometime-def)

lemma FI14:
 $SInf.SFinite = SInf$
by (metis FusionSFalse Powerstarhelp2 sinf-def)

lemma FI15:
 $SInf = - SFinite$
by (simp add: sfinite-def)

lemma FI16:
 $SFinite = - SInf$
by (simp add: sfinite-def)

lemma FI17:
 $((F.STrue) \cap SFinite) = (F \cap SFinite).SFinite$
by (metis FI04 boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def)

lemma FI18:
 $((STrue.F) \cap SFinite) = SFinite.(F \cap SFinite)$
by (metis BD19 FI01 FI04 FI17 FusionSEmptyR inf.idem sda-def)

lemma FI19:
 $SFinite.SMore = SMore$
by (metis BD06 FusionAssoc1 N14 N15 ssometime-def subset-antisym)

lemma FI20:
 $SInf \cup SFinite = STrue$
using STrueTop sfinite-def **by** auto

lemma FI21:

$SFMore = SSkip \cdot SFinite$
using *FI09 FI10 sfmore-def* **by** *auto*

lemma *FI22*:
 $(F \cap SFinite \subseteq G) = ((F \cap SFinite) \subseteq (G \cap SFinite))$
by *simp*

lemma *FI23*:
 $(F \cdot G) \cap SInf = F \cdot (G \cap SInf)$
by (*metis FusionAssoc FusionSFalse*)

lemma *FI24*:
 $((F \cdot G) \cap SInf) = ((F \cap SInf) \cup ((F \cap SFinite) \cdot (G \cap SInf)))$
using *FI23[of F G] Powerstarhelp2 Powerstarhelp3* **by** *fastforce*

lemma *FI25*:
 $SMore \cap SInf = SInf$
using *SStar15 SStarSkip sfinite-def smore-def* **by** *fastforce*

lemma *FI26*:
 $(F \cap SInf) \cdot (F \cap SInf) = (F \cap SInf)$
using *Powerstarhelp2* **by** *blast*

lemma *FI27*:
 $(F \cap SInf) \cdot G = (F \cap SInf)$
using *Powerstarhelp2* **by** *blast*

lemma *FI28*:
 $((F \cap SMore) \cap SInf) = (F \cap SInf)$
using *FI25* **by** *blast*

lemma *FI29*:
 $((F \cap SMore) \cap SFinite) = (F \cap SFMore)$
by (*metis inf-assoc inf-commute sfmore-def*)

lemma *FI30*:
 $(SEmpty \cap SFMore) = SFalse$
by (*simp add: sfalse-def sfmore-def smore-def*)

lemma *FI31*:
 $((F \cap SInf) \cap SFMore) = SFalse$
by (*simp add: sfalse-def sfinite-def sfmore-def*)

lemma *FI32*:
 $(SEmpty \cap SInf) = SFalse$
using *FI25 sfalse-def smore-def* **by** *auto*

lemma *FI33*:
 $(F \cap SInf) = F \cdot SFalse$
by (*simp add: FusionSFalse*)

lemma FI34:

$(F \cap SFinite) \cdot G \subseteq SSometime\ G$

by (*simp add: FusionRuleL ssometime-def*)

lemma FI35:

assumes $F \subseteq SNext\ F$

shows $SFinite \subseteq -F$

using *assms*

by (*metis B01 FusionSEmptyR N12 N22 SStarInductL SStarSkip double-compl snext-def*)

lemma FI36:

assumes $F \cap -G \subseteq SNext\ F$

shows $F \cap SFinite \subseteq SSometime\ G$

using *assms*

proof —

have 1: $F \cap -G \subseteq SNext\ F$

using *assms* **by** *auto*

have 2: $F \cap -G \cap SAlways\ (-G) \subseteq SNext\ F \cap SAlways\ (-G)$

using *assms* **by** *blast*

have 3: $SAlways\ (-G) \subseteq -G$

by (*simp add: BD12*)

have 4: $SAlways\ (-G) = SAlways\ (-G) \cap -G$

using 3 **by** *blast*

have 5: $F \cap SAlways\ (-G) \subseteq SNext\ F \cap SAlways\ (-G)$

using 2 4 **by** *blast*

have 51: $(SFinite \cdot G) = G \cup (SSkip \cdot (SFinite \cdot G))$

by (*metis FusionAssoc1 SStarSkip ST47 UnfoldL*)

have 6: $SAlways\ (-G) = -G \cap SWnext\ (SAlways\ (-G))$

using 51 **by** (*simp add: always-def ssometime-def swnext-def*) *blast*

have 7: $SNext\ F \cap SAlways\ (-G) \subseteq SNext\ F \cap SWnext\ (SAlways\ (-G))$

using 6 **by** *blast*

have 8: $F \cap SAlways\ (-G) \subseteq SNext\ F \cap SWnext\ (SAlways\ (-G))$

using 5 7 **by** *blast*

have 9: $F \cap SAlways\ (-G) \subseteq SMore \cap SWnext\ F \cap SWnext\ (SAlways\ (-G))$

using 8 N16 **by** *blast*

have 10: $F \cap SAlways\ (-G) \subseteq SWnext\ F \cap SWnext\ (SAlways\ (-G))$

using 8 N16 **by** *fastforce*

have 11: $F \cap SAlways\ (-G) \subseteq SWnext\ (F \cap SAlways\ (-G))$

using 10 N17 **by** *blast*

have 12: $SAlways\ (F \cap SAlways\ (-G)) \subseteq SWnext\ (F \cap SAlways\ (-G))$

by (*metis 10 B15 BD12 N17 inf.absorb-iff2*)

have 13: $SAlways\ (F \cap SAlways\ (-G)) \subseteq$

$SWnext\ (F \cap SAlways\ (-G)) \cap F \cap -(SAlways\ (-G)) \cup SAlways\ (F \cap SAlways\ (-G))$

using 12 BD12 **by** *auto*

have 14: $(F \cap SAlways\ (-G)) \subseteq SAlways\ (F \cap SAlways\ (-G))$

using 12 13

by (*metis 10 B08 BD08 N17 SBoxInduct boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def*)

have 15: $SAlways\ (F \cap SAlways\ (-G)) \subseteq (F \cap SAlways\ (-G))$

using BD12 **by** *blast*

have 16: $SAlways (F \cap SAlways (-G)) = (F \cap SAlways (-G))$
using 14 15 by *blast*
have 17: $(F \cap SAlways (-G)) \subseteq SMore$
using 9 by *blast*
have 18: $SAlways (F \cap SAlways (-G)) \subseteq SAlways SMore$
by (*simp add: 17 BD38*)
have 19: $SFinite = \neg(SAlways SMore)$
by (*simp add: BD01 salways-def smore-def*)
have 20: $SFinite \subseteq \neg(F \cap SAlways (-G))$
using 16 18 19 by *blast*
have 21: $SFinite \subseteq \neg F \cup \neg(SAlways (-G))$
using 20 by *blast*
have 22: $\neg(SAlways (-G)) = SSometime G$
by (*simp add: BD09*)
show ?thesis
using 20 22 by *blast*
qed

lemma FI37:
assumes $F \cap \neg G \subseteq SNext (F \cap \neg G)$
shows $F \cap SFinite \subseteq SSometime G$
using *assms*
by (*metis B15 FI36 N05*)

lemma FI38:
assumes $F \cap \neg G \subseteq (SNext F) \cap \neg(SNext G)$
shows $F \cap SFinite \subseteq G$
proof –
have 1: $F \cap \neg G \subseteq SNext((F \cap \neg G))$
using *N17[of F -G]*
by (*metis (no-types, lifting) N12 N16 assms double-compl inf.semigroup-axioms semigroup.assoc*)
have 2: $SFinite \subseteq \neg((F \cap \neg G))$
using 1 FI35 by *auto*
show ?thesis
using 2 by *blast*
qed

lemma FI39:
assumes $SWnext (SSometime F) \subseteq F$
shows $SFinite \subseteq F$
proof –
have 1: $\neg F \subseteq SNext (\neg F)$
by (*metis BD02 N12 N18 Subsumption assms compl-le-swap2 double-complement sup.coboundedI1*)
from 1 show ?thesis using FI35 by *blast*
qed

lemma FI40:
assumes $SEmpty \subseteq F$
 $SNext F \subseteq F$
shows $SFinite \subseteq F$

proof –
have 1: $-F \subseteq SNext (-F)$
using *N12 N19 assms(1) assms(2)* **by** *blast*
from 1 **show** *?thesis* **using** *FI35* **by** *blast*
qed

lemma *FI41*:
assumes $SEmpty \cap F \subseteq G$
 $SNext (-F \cup G) \cap F \subseteq G$
shows $F \cap SFinite \subseteq G$
proof –
have 1: $(F \cap -G) \subseteq SNext (F \cap -G)$
using *N19[of (-F \cup G)]*
using *assms(1) assms(2) snext-def swnext-def* **by** *fastforce*
have 2: $SFinite \subseteq -(F \cap -G)$
using 1 *FI35* **by** *auto*
from 2 **show** *?thesis* **by** *blast*
qed

lemma *FI42*:
assumes $F \subseteq SFMore \cdot F$
shows $SFinite \subseteq -F$
proof –
have 1: $SSometime F \subseteq SNext (SSometime F)$
by (*simp add: ssometime-def snext-def*)
(*metis FI21 SChopAssoc SStarSkip ST47 UnfoldL assms order-refl subset-Un-eq*)
have 2: $SFinite \subseteq -(SSometime F)$
by (*simp add: 1 FI35*)
show *?thesis*
by (*metis 2 B01 FI21 SStarSkip ST47 UnfoldL assms le-iff-sup ssometime-def subset-trans*)
qed

lemma *FI43*:
assumes $F \cap -G \subseteq SFMore \cdot (F \cap -G)$
shows $F \cap SFinite \subseteq G$
proof –
have 1: $SFinite \subseteq -(F \cap -G)$
using *FI42 assms* **by** *blast*
from 1 **show** *?thesis* **by** *blast*
qed

lemma *FI44*:
assumes $F \cap SFinite \subseteq SFMore \cdot F$
shows $SFinite \subseteq -F$
proof –
have 1: $F \cap SFinite \subseteq SFMore \cdot (F \cap SFinite)$
by (*metis FI04 FI21 FI22 SSkipSFinite assms inf.idem*)
show *?thesis*
by (*metis 1 B01 B11 B12 FI43 inf.right-idem sfinite-def*)
qed

lemma FI45:
assumes $F \cap SFinite \subseteq SMore \cdot F$
shows $SFinite \subseteq -F$
proof –
have 1: $F \cap SFinite \subseteq SMore \cdot (F \cap SFinite)$
by (*metis FI04 FI22 assms inf-commute sfmore-def*)
show ?thesis
by (*metis 1 B01 B11 B12 FI43 inf.right-idem sfinite-def*)
qed

lemma FI46:
assumes $F \subseteq SMore \cdot F$
shows $SFinite \subseteq -F$
proof –
have 1: $F \cap SFinite \subseteq SMore \cdot (F \cap SFinite)$
using B11 FI45 *assms* **by** *auto*
show ?thesis
by (*metis 1 B01 B11 B12 FI43 inf.right-idem sfinite-def*)
qed

lemma FI47:
assumes $(F \cap -G) \cap SFinite \subseteq SMore \cdot (F \cap -G)$
shows $F \cap SFinite \subseteq G$
proof –
have 1: $SFinite \subseteq -(F \cap -G)$
using FI44 *assms* **by** *blast*
from 1 **show** ?thesis **by** *blast*
qed

lemma FI48:
assumes $(F \cap -G) \subseteq SMore \cdot (F \cap -G)$
shows $F \cap SFinite \subseteq G$
proof –
have 1: $SFinite \subseteq -(F \cap -G)$
using FI45 *assms* **by** *blast*
from 1 **show** ?thesis **by** *blast*
qed

lemma FI49:
assumes $F \subseteq G \cdot F$
 $G \subseteq SMore$
shows $SFinite \subseteq -F$
proof –
have 1: $G \cdot F \subseteq SMore \cdot F$
by (*simp add: FusionRuleL assms(2)*)
have 2: $F \subseteq SMore \cdot F$
using 1 *assms*(1) **by** *auto*
from 2 **show** ?thesis
by (*simp add: FI42*)

qed

lemma FI50:

assumes $F \subseteq G \cdot F$

$G \subseteq SMore$

shows $SFinite \subseteq -F$

proof –

have 1: $G \cdot F \subseteq SMore \cdot F$

by (simp add: FusionRuleL assms(2))

have 2: $F \subseteq SMore \cdot F$

using 1 assms(1) by auto

from 2 show ?thesis

by (simp add: FI46)

qed

lemma FI51:

assumes $F \cap -G \subseteq (H \cdot F) \cap -(H \cdot G)$

$H \subseteq SFMore$

shows $F \cap SFinite \subseteq G$

proof –

have 1: $H \cdot (F \cap -G) \subseteq SFMore \cdot (F \cap -G)$

by (simp add: FusionRuleL assms(2))

have 2: $F \cap -G \subseteq SFMore \cdot (F \cap -G)$

using 1 CH02 assms(1) by blast

from 2 show ?thesis

by (simp add: FI43)

qed

lemma FI52:

assumes $F \cap -G \subseteq (H \cdot F) \cap -(H \cdot G)$

$H \subseteq SMore$

shows $F \cap SFinite \subseteq G$

proof –

have 1: $H \cdot (F \cap -G) \subseteq SMore \cdot (F \cap -G)$

by (simp add: FusionRuleL assms(2))

have 2: $F \cap -G \subseteq SMore \cdot (F \cap -G)$

using 1 CH02 assms(1) by blast

from 2 show ?thesis

by (simp add: FI48)

qed

lemma FI53:

$SAlways SInf = SInf$

by (metis BD01 BD35 BD42 FI15 salways-def)

Omega

lemma OA01:

$(somega SSkip) = SSkip \cdot (somega SSkip)$

by (metis SOmegaUnroll SSkipSFinite SSkipSMore)

lemma *OA02*:

$(\text{somega } S\text{Empty}) = S\text{False}$

by (*metis FI30 SFalseFusion SOmegaUnroll inf-assoc inf-commute sfmore-def*)

lemma *OA03*:

$(\text{somega } S\text{False}) = S\text{False}$

by (*metis Compl-disjoint2 Powerstarhelp2 SOmegaUnroll inf-assoc inf-compl-bot-right sfalse-def*)

lemma *OA04*:

$(\text{somega } S\text{Inf}) = S\text{False}$

by (*metis FI25 SFalseBottom SFalseFusion SOmegaUnroll inf-commute sfinite-def*)

lemma *BD59*:

$$S\text{Inf} \cap G \cap S\text{Always} (-G \cup (((F \cap S\text{More}) \cap S\text{Finite}) \cdot G)) \subseteq$$

$$((F \cap S\text{More}) \cap S\text{Finite}) \cdot (S\text{Inf} \cap G \cap S\text{Always} (-G \cup (((F \cap S\text{More}) \cap S\text{Finite}) \cdot G)))$$

proof –

have 1: $S\text{Inf} \cap S\text{Always} (-G \cup (((F \cap S\text{More}) \cap S\text{Finite}) \cdot G)) \subseteq (-G \cup (((F \cap S\text{More}) \cap S\text{Finite}) \cdot G))$
using *BD12* **by** *blast*

have 2: $S\text{Inf} \cap G \cap S\text{Always} (-G \cup (((F \cap S\text{More}) \cap S\text{Finite}) \cdot G)) \subseteq$
 $S\text{Inf} \cap (((F \cap S\text{More}) \cap S\text{Finite}) \cdot G) \cap S\text{Always} (-G \cup (((F \cap S\text{More}) \cap S\text{Finite}) \cdot G))$
using 1 **by** *blast*

have 3: $S\text{Inf} \cap (((F \cap S\text{More}) \cap S\text{Finite}) \cdot G) \cap S\text{Always} (-G \cup (((F \cap S\text{More}) \cap S\text{Finite}) \cdot G)) \subseteq$
 $((F \cap S\text{More}) \cap S\text{Finite}) \cdot (S\text{Inf} \cap G \cap S\text{Always} (-G \cup (((F \cap S\text{More}) \cap S\text{Finite}) \cdot G)))$
by (*metis BD58 FI23 inf-commute*)

show ?thesis **using** 2 3 **by** *blast*

qed

lemma *OA06*:

$S\text{Inf} \cap G \cap S\text{Always} (-G \cup (((F \cap S\text{More}) \cap S\text{Finite}) \cdot G)) \subseteq (\text{somega } F)$

by (*metis (no-types, lifting) BD59 SOmegaWeakCoinduct*)

lemma *FI54*:

$S\text{Always } S\text{FMore} \subseteq S\text{Inf}$

by (*metis BD38 BD56 Compl-disjoint2 SMoreImpSSkipFusion SSkipFusionImpSMore*
inf-le2 sfalse-def sfmore-def sinf-def smore-def subset-antisym sup.orderE)

lemma *FI55*:

$S\text{Always } S\text{Finite} = S\text{Finite}$

using *FI13 FI15 salways-def ssometime-def* **by** *fastforce*

lemma *FI56*:

$(S\text{Always } S\text{FMore}) = S\text{False}$

using *BD12 FI31 FI54* **by** *blast*

lemma *OA07*:

$(\text{somega } ((S\text{Power } S\text{Skip } (S\text{uc } n)))) \subseteq S\text{Inf}$

by (*metis FI15 FI42 FusionRuleL SOmegaUnroll boolean-algebra-cancel.inf0 compl-le-swap1 inf-commute*
inf-le2 inf-mono sfmore-def)

lemma *OA08*:

$SInf \subseteq (\text{somega } ((SPower SSkip (Suc n))))$

proof –

have 1: $SInf \cap STrue \cap SAlways (\neg STrue \cup (((SPower SSkip (Suc n)) \cap SMore) \cap SFinite) \cdot STrue)) \subseteq (\text{somega } (SPower SSkip (Suc n)))$

by (*simp add: OA06*)

have 2: $\neg STrue \cup (SPower SSkip (Suc n) \cap SMore) \cap SFinite \cdot STrue = (SPower SSkip (Suc n) \cap SMore) \cap SFinite \cdot STrue$

by (*simp add: strue-def*)

have 21: $((((SPower SSkip (Suc n)) \cap SMore) \cap SFinite) \cdot STrue) \subseteq SMore$

by (*metis N13 N14 N15 SStar25 SStarInductR inf-le2 le-sup-iff subset-antisym sup-inf-absorb*)

have 3: $SAlways ((((SPower SSkip (Suc n)) \cap SMore) \cap SFinite) \cdot STrue)) \subseteq SAlways (SMore)$

using *BD38 21* **by** *auto*

have 4: $SAlways (SMore) \subseteq SInf$

using *FI05* **by** *blast*

have 5: $SAlways (\neg STrue \cup (((SPower SSkip (Suc n)) \cap SMore) \cap SFinite) \cdot STrue)) \subseteq SInf$

by (*metis 2 3 FI05 subset-trans*)

have 6: $SInf \subseteq SAlways (SMore)$

by (*simp add: BD01 FI15 salways-def smore-def*)

have 7: $(SPower SSkip (Suc n)) \cap SMore \cap SFinite \subseteq SMore \cap SFinite$

using *FI07 FI21* **by** *blast*

have 8: $SMore \subseteq (SMore \cap SFinite) \cdot STrue$

by (*metis FI19 FI21 ITA.SChopAssoc SMoreImpSSkipFusion SSkipFusionImpSMore SStar19 SStarSkip inf-commute sfmore-def subset-antisym*)

have 9: $SInf \cap (((SPower SSkip (Suc n)) \cap SMore) \cap SFinite) \cdot STrue) =$

$((((SPower SSkip (Suc n)) \cap SMore) \cap SFinite) \cdot SInf)$

by (*metis FI23 boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def*)

have 10: $SInf \cap SAlways ((((SPower SSkip (Suc n)) \cap SMore) \cap SFinite) \cdot STrue)) =$

$SAlways ((((SPower SSkip (Suc n)) \cap SMore) \cap SFinite) \cdot SInf))$

by (*metis 9 BD29 FI53*)

have 11: $((((SPower SSkip (Suc n)) \cap SMore) \cap SFinite) \cdot SInf) = SInf$

proof (*induct n*)

case 0

then show *?case*

proof –

have *f1*: $(SFinite) \cap SSkip = SSkip$

using *SSkipSFinite* **by** *blast*

have *f2*: $\forall S. S \cdot SSkip \subseteq (SMore \cap SFinite) \vee \neg S \subseteq SFinite$

using *FI10 FI21 FusionRuleL* **by** (*metis inf-commute sfmore-def*)

have *f3*: $(SEmpty) \cdot SSkip = SSkip$

using *f1* **by** (*metis FusionSEmptyR inf-commute pwr-0 spower-commutes*)

have *f4*: $(SEmpty) \subseteq SFinite$

using *pwr-0 spower-finite* **by** *blast*

have *f5*: $(SSkip) \subseteq SFinite$

using *f1* **by** *blast*

have *f6*: $(SSkip) \cdot (STrue \cdot (SFalse \cdot SFalse)) = SInf$

by (*metis FI25 FusionAssoc1 FusionSFalse Powerstarhelp2 SMoreImpSSkipFusion SSkipFusion-*

ImpSMore

subset-antisym)

have *f7*: $(SSkip) \cap SMore \cap SFinite = SSkip$

```

    using f4 f3 f2 by blast
  have (SSkip) · SInf = SInf
    using f6 f5 f4 f3 f2 by (simp add: SFalseFusion sinf-def)
  then show ?thesis
    by (simp add: f7 FusionSEmptyR SSkipSFinite)
  qed
next
case (Suc n)
then show ?case
proof -
  have f1: SSkip · STrue = SMore
    using SMoreImpSSkipFusion SSkipFusionImpSMore by blast
  have f2:  $\forall S. STrue \cap S = S$ 
    by (simp add: strue-def)
  have f3:  $\forall S. SMore \cap (SSkip \cdot S) = SSkip \cdot S$ 
    using f1
    by (metis (no-types) B14 CH06 boolean-algebra-cancel.inf0 compl-bot-eq inf-aci(1) inf-le2 strue-def)
  then have f0:  $\forall S. SMore \cap SFinite \cap (SSkip \cdot S) = SFinite \cap (SSkip \cdot S)$ 
    by blast
  then show ?thesis
    using f3 f2 f1
  proof -
    have f4:  $SPower (SSkip) (Suc (Suc n)) \cap SMore \cap SFinite = SFinite \cap (SSkip \cap SFinite \cdot SPower SSkip (Suc n))$ 
      by (metis SSkipSFinite f3 inf-sup-aci(1) pwr-Suc)
    then have f5:  $SPower (SSkip) (Suc (Suc n)) \cap SMore \cap SFinite = SSkip \cap SFinite \cdot SPower SSkip (Suc n)$ 
      by (metis (no-types) B14 CH06 boolean-algebra-cancel.inf0 compl-bot-eq inf-aci(1) inf-le2 strue-def)
    then show ?thesis
      using f4 by blast
    qed
  show ?thesis
    by (metis B14 CH01 FI12 FI23 FI25 FusionAssoc1 Powerstarhelp3 SSkipSFinite Suc f1 f3
      inf-commute pwr-Suc sinf-def spower-finite sup-inf-absorb)
    qed
  qed
have 12:  $SInf \subseteq SAlways ( (((SPower SSkip (Suc n)) \cap SMore) \cap SFinite) \cdot SInf )$ 
  by (metis 11 B04 FI53)
show ?thesis
  using 11 SOmegaWeakCoinduct by (metis 12 FI53)
qed

lemma OA09:
  (somega ((SPower SSkip (Suc n)))) = SInf
  using OA07 OA08 by blast

lemma OA10:

```

(somega SSkip) = SInf
by (metis FusionSEmptyR OA09 SSkipSFinite pwr-0 spower.simps(2))

lemma OA11:

(somega STrue) \subseteq SInf
by (metis FI15 FI42 SOmegaUnroll boolean-algebra.compl-zero boolean-algebra-cancel.inf0
compl-le-swap1 dual-order.refl inf-commute sfmore-def strue-def)

lemma OA12:

SInf \subseteq (somega STrue)
proof –
have 1: (SAlways (SFalse \cup ((STrue \cap SMore) \cap SFinite) \cdot STrue)) \subseteq SInf
by (metis B04 BD01 BD09 BD35 FI10 FI15 FI19 FI21 ITA.SChopAssoc SMoreImpSSkipFusion
SSkipFusionImpSMore boolean-algebra.compl-zero boolean-algebra.de-Morgan-disj
boolean-algebra-cancel.inf0 inf-commute salways-def sfalse-def sfmore-def smore-def strue-def)
have 2: SInf \subseteq (SAlways (SFalse \cup ((STrue \cap SMore) \cap SFinite) \cdot STrue))
by (metis 1 B15 BD01 BD09 BD12 BD35 FI10 FI15 FI19 FI21 FI56 ITA.SChopAssoc SMoreImpSSkip-
Fusion
SSkipFusionImpSMore boolean-algebra.compl-zero boolean-algebra-cancel.inf0 inf.absorb-iff2
inf-commute salways-def sfmore-def smore-def strue-def sup.absorb2)
have 3: SInf \cap STrue \cap (SAlways (SFalse \cup ((STrue \cap SMore) \cap SFinite) \cdot STrue)) \subseteq (somega STrue)
by (metis OA06 compl-bot-eq compl-top-eq sfalse-def strue-def)
show ?thesis
using 2 3 strue-def **by** fastforce
qed

lemma OA13:

(somega STrue) = SInf
by (simp add: B04 OA11 OA12)

lemma OA14:

(somega SMore) \subseteq SInf
by (metis Int-absorb1 Int-lower2 OA13 SOmegaUnroll SOmegaWeakCoinduct compl-bot-eq inf-top-left strue-def)

lemma OA15:

SInf \subseteq (somega SMore)
proof –
have 1: (SAlways (SFalse \cup ((SMore \cap SMore) \cap SFinite) \cdot STrue)) \subseteq SInf
by (metis FI05 FI10 FI19 FI21 FusionSEmptyR ITA.SChopAssoc SMoreImpSSkipFusion SSkipFusion-
ImpSMore
ST33 boolean-algebra.compl-one boolean-algebra.compl-zero boolean-algebra.de-Morgan-conj
inf.idem inf-commute sfalse-def sfmore-def smore-def strue-def subset-antisym)
have 2: SInf \subseteq (SAlways (SFalse \cup ((SMore \cap SMore) \cap SFinite) \cdot STrue))
by (metis BD38 FI10 FI19 FI21 FI25 FI53 FusionSEmptyR ITA.SChopAssoc SMoreImpSSkipFusion
SSkipFusionImpSMore ST33 boolean-algebra.compl-one boolean-algebra.compl-zero
boolean-algebra.de-Morgan-conj inf.idem inf-commute inf-le1 sfalse-def sfmore-def
smore-def strue-def subset-antisym)
have 3: SInf \cap STrue \cap (SAlways (SFalse \cup ((SMore \cap SMore) \cap SFinite) \cdot STrue)) \subseteq (somega STrue)

```

    using OA13 by blast
show ?thesis
  by (metis 1 2 Int-absorb1 OA06 boolean-algebra-cancel.inf0 compl-bot-eq compl-top-eq sfalse-def
    strue-def subset-antisym)
qed

lemma OA16:
  (somega SMore) = SInf
  using OA14 OA15 by blast

lemma OA17:
  (somega SFinite)  $\subseteq$  SInf
  by (metis FI07 FI15 FI21 FI42 SOmegaUnroll compl-le-swap1 dual-order.refl inf.orderE sfmore-def)

lemma OA18:
  SInf  $\subseteq$  (somega SFinite)
proof -
  have 1: (SAlways (SFalse  $\cup$  ((SFinite  $\cap$  SMore) $\cap$ SFinite) $\cdot$ STrue))  $\subseteq$  SInf
  by (metis B14 FI05 FI07 FI10 FI19 FI21 FusionSEmptyR ITA.SChopAssoc SMoreImpSSkipFusion
    SSkipFusionImpSMore ST33 boolean-algebra.compl-one boolean-algebra.compl-zero
    boolean-algebra.de-Morgan-conj sfalse-def sfmore-def smore-def strue-def subset-antisym)
  have 2: SInf  $\subseteq$  (SAlways (SFalse  $\cup$  ((SFinite  $\cap$  SMore) $\cap$ SFinite) $\cdot$ STrue))
  by (metis B14 BD38 FI10 FI19 FI21 FI25 FI53 FusionSEmptyR ITA.SChopAssoc SMoreImpSSkipFusion
    SSkipFusionImpSMore ST33 boolean-algebra.compl-one boolean-algebra.compl-zero
    boolean-algebra.de-Morgan-conj inf-le1 sfalse-def sfmore-def smore-def strue-def subset-antisym)
  have 3: SInf  $\cap$  STrue  $\cap$  (SAlways (SFalse  $\cup$  ((SFinite  $\cap$  SMore) $\cap$ SFinite) $\cdot$ STrue))  $\subseteq$  (somega STrue)
  using OA13 by blast
  show ?thesis
  by (metis 1 2 Int-absorb1 OA06 boolean-algebra-cancel.inf0 compl-bot-eq compl-top-eq sfalse-def
    strue-def subset-antisym)
qed

lemma OA19:
  (somega SFinite) = SInf
  by (simp add: B04 OA17 OA18)

lemma OA20:
  assumes  $H \subseteq ((F \cap SMore) \cap SFinite) \cdot H$ 
  shows  $H \cap SInf \subseteq (somega F)$ 
  using assms
  using SOmegaWeakCoinduct by blast

lemma OA21:
  assumes  $F \subseteq G$ 
  shows  $(somega F) \cap SInf \subseteq (somega G)$ 
  using assms
  by (metis FusionRuleL OA20 SOmegaUnroll inf-mono subset-refl)

lemma OA22:
  assumes  $F = G$ 

```


shows $(\text{somega } F) = (\text{somega } G)$
using *assms* **by** *auto*

lemma *OA23*:
 $(\text{somega } (F \cap G)) \cap SInf \subseteq (\text{somega } F)$
by (*simp add: OA21*)

lemma *OA24*:
 $(\text{somega } (F \cap G)) \cap SInf \subseteq (\text{somega } G)$
by (*simp add: OA21*)

lemma *BD60*:
 $(SBI\ F) \cap (G0 \cdot G1) \subseteq (F \cap G0) \cdot G1$
proof –
have 1: $F \subseteq (-G0 \cup (F \cap G0))$
by *blast*
have 2: $SBI\ F \subseteq SBI\ (-G0 \cup (F \cap G0))$
by (*simp add: 1 BD39*)
have 3: $SBI\ (-G0 \cup (F \cap G0)) \cap (G0 \cdot G1) \subseteq (F \cap G0) \cdot G1$
by (*simp add: BD45*)
show *?thesis*
using 2 3 **by** *blast*
qed

lemma *BD61*:
 $(SAlways\ H) \cap (F \cdot G) \subseteq F \cdot (H \cap G)$
proof –
have 1: $H \subseteq (-G \cup (H \cap G))$
by *blast*
have 2: $SAlways\ H \subseteq SAlways\ (-G \cup (H \cap G))$
by (*simp add: 1 BD38*)
have 3: $SAlways\ (-G \cup (H \cap G)) \cap (F \cdot G) \subseteq F \cdot (H \cap G)$
using *BD46* **by** *blast*
show *?thesis*
using 2 3 **by** *blast*
qed

lemma *BD62*:
 $(SBA\ F) \cap (G0 \cdot G1) \subseteq (F \cap G0) \cdot (F \cap G1)$
proof –
have 1: $(SBA\ F) \subseteq (SBI\ F)$
by (*simp add: BD27*)
have 2: $(SBI\ F) \cap (G0 \cdot G1) \subseteq (F \cap G0) \cdot G1$
by (*simp add: BD60*)
have 3: $(SBA\ F) \subseteq (SAlways\ F)$
by (*simp add: BD28*)
have 4: $(SAlways\ F) \cap ((F \cap G0) \cdot G1) \subseteq (F \cap G0) \cdot (F \cap G1)$
using *BD61* **by** *blast*
show *?thesis*
using 1 2 3 4 **by** *blast*

qed

lemma BD63:

$$(SBa\ F) \cap (G \cdot G1) \subseteq (F \cap G) \cdot ((SBa\ F) \cap G1)$$

proof –

$$\text{have } 1: (SBa\ F) = (SBa\ (SBa\ F))$$

using BD44 by blast

$$\text{have } 2: (SBa\ (SBa\ F)) \cap (G \cdot G1) \subseteq G \cdot ((SBa\ F) \cap G1)$$

using BD28 BD61 by blast

$$\text{have } 3: (SBa\ F) \cap (G \cdot ((SBa\ F) \cap G1)) \subseteq (F \cap G) \cdot ((SBa\ F) \cap G1)$$

using BD62 FusionRuleR by blast

show ?thesis

using 1 2 3 by blast

qed

lemma OA25:

$$SBa\ (-F \cup G) \cap SInf \cap (somega\ F) \subseteq (somega\ G)$$

proof –

$$\text{have } 1: SBa\ (-F \cup G) \cap (((F \cap SMore) \cap SFinite) \cdot (somega\ F)) \subseteq ((-F \cup G) \cap ((F \cap SMore) \cap SFinite)) \cdot ((-F \cup G) \cap (somega\ F))$$

by (simp add: BD62)

$$\text{have } 2: (-F \cup G) \cap ((F \cap SMore) \cap SFinite) \subseteq ((G \cap SMore) \cap SFinite)$$

by auto

$$\text{have } 3: (-F \cup G) \cap (somega\ F) \subseteq (somega\ F)$$

by auto

$$\text{have } 4: SBa\ (-F \cup G) \cap (((F \cap SMore) \cap SFinite) \cdot (somega\ F)) \subseteq ((G \cap SMore) \cap SFinite) \cdot (somega\ F)$$

using 1 2 3 CH06 by blast

$$\text{have } 5: SBa\ (-F \cup G) \cap (((F \cap SMore) \cap SFinite) \cdot (somega\ F)) \subseteq ((-F \cup G) \cap ((F \cap SMore) \cap SFinite)) \cdot (SBa\ (-F \cup G) \cap (somega\ F))$$

using BD63 by blast

$$\text{have } 6: ((-F \cup G) \cap ((F \cap SMore) \cap SFinite)) \cdot (SBa\ (-F \cup G) \cap (somega\ F)) \subseteq ((G \cap SMore) \cap SFinite) \cdot (SBa\ (-F \cup G) \cap (somega\ F))$$

using 2 FusionRuleL by blast

$$\text{have } 7: (SBa\ (-F \cup G) \cap (somega\ F)) \subseteq ((G \cap SMore) \cap SFinite) \cdot (SBa\ (-F \cup G) \cap (somega\ F))$$

using 5 6 SOmegaUnroll by blast

$$\text{have } 8: (SBa\ (-F \cup G) \cap (somega\ F)) \cap SInf \subseteq (somega\ G)$$

by (simp add: 7 OA20)

show ?thesis

using 8 by auto

qed

lemma OA26:

$$SBa\ ((-F \cup G) \cap (-G \cup F)) \cap SInf \subseteq (-(somega\ G) \cup (somega\ F)) \cap (-(somega\ F) \cup (somega\ G))$$

proof –

$$\text{have } 1: SBa\ ((-F \cup G) \cap (-G \cup F)) = SBa\ (-F \cup G) \cap SBa\ (-G \cup F)$$

by (simp add: BD22 sba-def)

$$\text{have } 2: SBa\ (-F \cup G) \cap SInf \subseteq (-(somega\ F) \cup (somega\ G))$$

by (simp add: B19 OA25)

$$\text{have } 3: SBa\ (-G \cup F) \cap SInf \subseteq (-(somega\ G) \cup (somega\ F))$$

by (simp add: B19 OA25)

show *?thesis*
using 1 2 3 **by** *blast*
qed

lemma *OA27*:
 $SBa\ F \cap (somega\ G) \cap SInf \subseteq somega\ (F \cap G)$
by (*metis (no-types, lifting) BD63 OA20 SOmegaUnroll inf-assoc*)

lemma *FI57*:
 $SInf \cap (((F \cap SMore) \cap SFinite) \cdot G) = ((F \cap SMore) \cap SFinite) \cdot (G \cap SInf)$
using *FI23* **by** *blast*

lemma *FI58*:
 $SInf \cap (SAlways\ F) = SAlways\ (F \cap SInf)$
using *BD29 FI53* **by** *blast*

lemma *FI59*:
 $SInf \cap (SAlways\ (-F \cup G)) = SAlways\ ((-F \cap SInf) \cup (G \cap SInf))$
by (*simp add: FI58 inf-sup-distrib2*)

2.15.6 Link between Set of Intervals and ITL

lemma *interval-lan [simp]*:
 $\sigma \in (lan\ f) \longleftrightarrow (\sigma \models f)$
by (*simp add: lan-def*)

lemma *valid-lan-equiv* :
 $((lan\ f) = (lan\ g)) \longleftrightarrow (\vdash f = g)$
using *interval-lan lan-def Valid-def* **by** *fastforce*

lemma *valid-lan-imp* :
 $((lan\ f) \subseteq (lan\ g)) \longleftrightarrow (\vdash f \longrightarrow g)$
using *interval-lan lan-def Valid-def*
by (*simp add: Valid-def lan-def Collect-mono-iff*)

lemma *valid-strue* :
 $((lan\ f) = STrue) \longleftrightarrow (\vdash f)$
using *strue-def* **by** *fastforce*

lemma *strue-true*:
 $\sigma \in STrue \longleftrightarrow (\sigma \models \#True)$
by (*simp add: strue-elim*)

lemma *strue-true-1*:
 $STrue = (lan\ (LIFT\ \#True))$
using *lan-def strue-true* **by** *fastforce*

lemma *sfalse-false*:
 $\sigma \in SFalse \longleftrightarrow (\sigma \models \#False)$
by (*simp add: sfalse-def*)

lemma *sfalse-false-1*:

$SFalse = (lan\ (LIFT(\#False)))$

using *sfalse-false* **using** *lan-def* **by** *fastforce*

lemma *not-negation*:

$\sigma \in (-(lan\ f)) \longleftrightarrow (\sigma \models \neg f)$

by *simp*

lemma *not-negation-1*:

$-(lan\ f) = (lan\ (LIFT(\neg f)))$

using *interval-lan lan-def* **by** *fastforce*

lemma *inter-and*:

$(\sigma \in ((lan\ f) \cap (lan\ g))) \longleftrightarrow (\sigma \models f \wedge g)$

by (*simp add: lan-def*)

lemma *inter-and-1*:

$((lan\ f) \cap (lan\ g)) = (lan\ (LIFT(f \wedge g)))$

using *inter-and lan-def* **by** *fastforce*

lemma *union-or*:

$(\sigma \in ((lan\ f) \cup (lan\ g))) \longleftrightarrow (\sigma \models f \vee g)$

by (*simp add: lan-def*)

lemma *union-or-1*:

$((lan\ f) \cup (lan\ g)) = (lan\ (LIFT(f \vee g)))$

using *union-or lan-def* **by** *fastforce*

lemma *subset-impl*:

$(\sigma \in (-(lan\ f) \cup (lan\ g))) \longleftrightarrow (\sigma \models f \longrightarrow g)$

by *simp*

lemma *subset-impl-1*:

$(-(lan\ f) \cup (lan\ g)) = (lan\ (LIFT(f \longrightarrow g)))$

using *subset-impl lan-def* **by** *fastforce*

lemma *fusion-chop*:

$(\sigma \in ((lan\ f) \cdot (lan\ g))) \longleftrightarrow (\sigma \models f;g)$

by (*auto simp add: fusion-iff chop-nfuse*)

lemma *fusion-chop-1*:

$((lan\ f) \cdot (lan\ g)) = (lan\ (LIFT(f;g)))$

using *fusion-chop lan-def* **by** *blast*

lemma *empty-empty*:

$\sigma \in SEmpty \longleftrightarrow (\sigma \models empty)$

by (*simp add: itl-defs empty-elim zero-enat-def*)

lemma *empty-empty-1*:

$SEmpty = (\text{lan } (LIFT \text{ empty}))$
using *empty-empty lan-def* **by** *fastforce*

lemma *smore-more*:
 $\sigma \in SMore \longleftrightarrow (\sigma \models \text{more})$
using *zero-enat-def* **by** (*auto simp add: itl-defs smore-elim smore-def*)

lemma *smore-more-1*:
 $SMore = (\text{lan } (LIFT \text{ more}))$
using *smore-more lan-def* **by** *fastforce*

lemma *sskip-skip*:
 $\sigma \in SSkip = (\sigma \models \text{skip})$
by (*simp add: one-enat-def itl-defs sskip-elim*)

lemma *sstrip-skip-1*:
 $SSkip = (\text{lan } (LIFT \text{ skip}))$
using *sstrip-skip lan-def* **by** *fastforce*

lemma *snext-next*:
 $\sigma \in (SNext (\text{lan } f)) \longleftrightarrow (\sigma \models \bigcirc f)$
by (*metis snext-def fusion-chop next-d-def sstrip-skip-1*)

lemma *snext-next-1*:
 $(SNext (\text{lan } f)) = (\text{lan } (LIFT(\bigcirc f)))$
using *snext-next lan-def* **by** *fastforce*

lemma *swnext-wnext*:
 $\sigma \in (SWnext (\text{lan } f)) \longleftrightarrow (\sigma \models \text{wnext } f)$
by (*simp add: swnext-def fusion-chop-1 next-d-def not-negation-1 sstrip-skip-1 wnext-d-def*)

lemma *swnext-wnext-1*:
 $(SWnext (\text{lan } f)) = (\text{lan } (LIFT(\text{wnext } f)))$
using *swnext-wnext lan-def* **by** *fastforce*

lemma *sprev-prev*:
 $\sigma \in (SPrev (\text{lan } f)) \longleftrightarrow (\sigma \models \text{prev } f)$
by (*metis fusion-chop prev-d-def sprev-def sstrip-skip-1*)

lemma *sprev-prev-1*:
 $(SPrev (\text{lan } f)) = (\text{lan } (LIFT(\text{prev } f)))$
using *sprev-prev lan-def* **by** *fastforce*

lemma *suprev-wprev*:
 $\sigma \in (SWprev (\text{lan } f)) \longleftrightarrow (\sigma \models \text{wprev } f)$
by (*simp add: fusion-chop-1 not-negation-1 prev-d-def sstrip-skip-1 suprev-def wprev-d-def*)

lemma *suprev-wprev-1*:
 $(SWprev (\text{lan } f)) = (\text{lan } (LIFT(\text{wprev } f)))$
using *suprev-wprev lan-def* **by** *fastforce*

lemma *sinit-init*:

$\sigma \in SInit \text{ (lan } f) \longleftrightarrow (\sigma \models \text{init } f)$

by (*simp add: Int-commute fusion-chop-1 init-d-def inter-and-1 empty-empty-1 sinit-def strue-true-1*)

lemma *sinit-init-1*:

$SInit \text{ (lan } f) = (\text{lan } (LIFT(\text{init } f)))$

using *sinit-init lan-def* **by** *fastforce*

lemma *sfinite*:

$\sigma \in SFinite \longleftrightarrow (\sigma \models \text{finite})$

by (*simp add: sfinite-def sinf-def finite-d-def infinite-d-def fusion-chop sfalse-false-1 strue-true-1*)

lemma *sfinite-1*:

$SFinite = \text{lan}(LIFT(\text{finite}))$

using *sfinite lan-def* **by** *fastforce*

lemma *and-inter-finite*:

$\sigma \in (((\text{lan } f) \cap SFinite)) \longleftrightarrow (\sigma \models (f \wedge \text{finite}))$

using *sfinite inter-and* **by** *auto*

lemma *and-inter-finite-1*:

$(((\text{lan } f) \cap SFinite)) = \text{lan}(LIFT(f \wedge \text{finite}))$

by (*simp add: inter-and-1 sfinite-1*)

lemma *and-inter-more*:

$\sigma \in (((\text{lan } f) \cap SMore)) \longleftrightarrow (\sigma \models (f \wedge \text{more}))$

using *smore-more inter-and* **by** *auto*

lemma *and-inter-more-1*:

$\sigma \in (((\text{lan } f) \cap SMore)) \longleftrightarrow (\sigma \in (\text{lan } (LIFT(f \wedge \text{more}))))$

using *and-inter-more lan-def* **by** (*simp add: smore-more-1*)

lemma *and-inter-more-2*:

$((\text{lan } f) \cap SMore) = (\text{lan } (LIFT(f \wedge \text{more})))$

using *and-inter-more-1* **by** *blast*

lemma *and-chop*:

$\sigma \in (((\text{lan } f) \cap SMore) \cdot (\text{lan } g)) \longleftrightarrow (\sigma \models (f \wedge \text{more});g)$

by (*metis fusion-chop inter-and-1 smore-more-1*)

lemma *and-chop-1*:

$(((\text{lan } f) \cap SMore) \cdot (\text{lan } g)) = (\text{lan } (LIFT((f \wedge \text{more});g)))$

using *and-chop lan-def* **by** *blast*

lemma *spower-chop-power*:

$(SPower \text{ (lan } f) \text{ } n) = (\text{lan } (LIFT(\text{fpower } f \text{ } n)))$

proof (*induct n*)

case 0

then show ?case

```

by (simp add: empty-empty-1 fpower-d-def)
next
case (Suc n)
then show ?case
by (simp add: and-inter-finite-1 fusion-chop-1 fpower-d-def)
qed

```

lemma *powerstar*:

```

 $\sigma \in SPowerstar (lan f) \longleftrightarrow \sigma \in SFPowerstar (lan (f)) \cdot (SEmpty \cup ((lan f) \cap SInf))$ 
by (simp add: powerstar-def)

```

lemma *sstar-powerstar*:

```

 $\sigma \in SStar (lan f) \longleftrightarrow \sigma \in SPowerstar ((lan f) \cap SMore)$ 
by (simp add: sstar-def)

```

lemma *union-exists*:

```

 $\sigma \in (\bigcup n. SPower (lan f) n) \longleftrightarrow \sigma \in lan(LIFT(\exists n. fpower f n))$ 
by (simp add: power-chop-power)

```

lemma *union-exists-1*:

```

 $(\bigcup n. SPower (lan f) n) = lan(LIFT(\exists n. fpower f n))$ 

```

using *union-exists lan-def* **by** *blast*

lemma *sstar-chopstar*:

```

 $\sigma \in (SStar (lan f)) \longleftrightarrow \sigma \in (lan (LIFT(f^*)))$ 

```

proof –

```

have 1:  $\sigma \in (SStar (lan f)) \longleftrightarrow \sigma \in SPowerstar ((lan f) \cap SMore)$ 

```

```

using sstar-powerstar by blast

```

```

have 2:  $\sigma \in SPowerstar ((lan f) \cap SMore) \longleftrightarrow$ 

```

```

 $\sigma \in SFPowerstar (lan (f) \cap SMore) \cdot (SEmpty \cup (((lan f) \cap SMore) \cap SInf))$ 

```

```

by (simp add: powerstar-def)

```

```

have 3:  $SFPowerstar (lan (f) \cap SMore) = SFPowerstar(lan(LIFT(f \wedge more)))$ 

```

```

by (simp add: and-inter-more-2)

```

```

have 31:  $\bigwedge n. SPower (lan(LIFT(f \wedge more))) n = lan(LIFT(fpower (f \wedge more) n))$ 

```

```

using power-chop-power by blast

```

```

have 32:  $SFPowerstar(lan(LIFT(f \wedge more))) = lan(LIFT(fpowerstar (f \wedge more)))$ 

```

```

using union-exists-1 by (auto simp add: sfpowerstar-def fpowerstar-d-def)

```

```

have 4:  $(SEmpty \cup (((lan f) \cap SMore) \cap SInf)) =$ 

```

```

 $lan(LIFT(empty \vee ((f \wedge more) \wedge inf)))$ 

```

```

by (metis Morgan and-inter-more-2 finite-d-def inter-and-1 not-negation-1 empty-empty-1
sfinite-1 sfinite-def union-or-1)

```

```

have 5:  $SFPowerstar (lan (f) \cap SMore) \cdot (SEmpty \cup (((lan f) \cap SMore) \cap SInf)) =$ 

```

```

 $lan(LIFT(powerstar (f \wedge more)))$ 

```

```

by (simp add: powerstar-d-def 3 32 4 fpowerstar-d-def fusion-chop-1)

```

```

have 6:  $lan(LIFT(powerstar (f \wedge more))) =$ 

```

```

 $lan(LIFT(chopstar f))$ 

```

```

by (simp add: chopstar-d-def)

```

show *?thesis*

```

by (simp add: 1 2 5 6)

```

qed

lemma *chopstar-sstar-1*:

$(SStar\ (lan\ f)) = (lan\ (LIFT(f^*)))$

using *sstar-chopstar lan-def* **by** *blast*

lemma *chopstar-seqv*:

$\sigma \in (lan\ (LIFT(f^*))) \longleftrightarrow \sigma \in (lan\ (LIFT(empty \vee (f \wedge more); f^*)))$

by (*metis* (*no-types*, *lifting*) *SChopstarEqv chopstar-sstar-1 fusion-chop-1 inter-and-1 empty-empty-1 smore-more-1 union-or-1*)

lemma *chopstar-seqv-1*:

$(lan\ (LIFT(f^*))) = (lan\ (LIFT(empty \vee (f \wedge more); f^*)))$

using *chopstar-seqv lan-def* **by** *blast*

lemma *sinf*:

$\sigma \in SInf \longleftrightarrow (\sigma \models inf)$

by (*simp add: fusion-chop infinite-d-def sfalse-false-1 sinf-def strue-true-1*)

lemma *sinf-1*:

$SInf = lan(LIFT(inf))$

using *sinf* **by** *fastforce*

lemma *fmore*:

$\sigma \in SFMore \longleftrightarrow (\sigma \models fmore)$

by (*metis* *fmore-d-def inf-commute inter-and-1 interval-lan sfinite-1 sfmore-def smore-more-1*)

lemma *fmore-1*:

$SFMore = lan(LIFT(fmore))$

using *fmore* **by** *fastforce*

lemma *omega-induct-sem*:

$x \in - (lan\ g) \cup ((((lan\ f) \cap SMore) \cap SFinite) \cdot (lan\ g)) \longleftrightarrow$

$(x \models g \longrightarrow ((f \wedge more) \wedge finite); g)$

by (*simp add: fusion-chop-1 inter-and-1 sfinite-1 smore-more-1*)

lemma *omega-induct*:

$- (lan\ g) \cup ((((lan\ f) \cap SMore) \cap SFinite) \cdot (lan\ g)) =$

$lan(LIFT(g \longrightarrow ((f \wedge more) \wedge finite); g))$

using *omega-induct-sem[of - g f]* **by** *fastforce*

lemma *somega-omega-sem-1*:

assumes $x \models f^\omega$

shows $x \in somega\ (lan\ f)$

proof –

have 1: $x \models f^\omega \longrightarrow ((f \wedge more) \wedge finite); f^\omega$

by (*metis* *OmegaUnroll Valid-def int-iffD1 schop-d-def*)

have 2: $x \in - (lan\ (LIFT(f^\omega))) \cup ((((lan\ f) \cap SMore) \cap SFinite) \cdot (lan\ (LIFT(f^\omega))))$

using 1 *omega-induct-sem* **by** *blast*

have 3: $\bigwedge x . x \in (lan\ (LIFT(f^\omega))) \implies x \in ((((lan\ f) \cap SMore) \cap SFinite) \cdot (lan\ (LIFT(f^\omega))))$

by (*metis* (*mono-tags*, *opaque-lifting*) *OmegaUnroll and-inter-finite-1 and-inter-more-2 fusion-chop-1 int-eq*)

itl-def(9)

show *?thesis* **using** 3 *SOmegaWeakCoinductsem* *assms* *interval-lan* **by** *blast*
qed

lemma *somega-omega-sem-2*:

assumes $x \in \text{somega } (\text{lan } f)$

shows $x \models f^\omega$

proof –

have 1: $x \in - (\text{somega } (\text{lan } f)) \cup ((((\text{lan } f) \cap \text{SMore}) \cap \text{SFinite}) \cdot (\text{somega } (\text{lan } f))))$

using *SOmegaCases* **by** *blast*

have 2: $\bigwedge x. (\lambda x. x \in \text{somega } (\text{lan } f)) x \implies$

$x \in ((((\text{lan } f) \cap \text{SMore}) \cap \text{SFinite}) \cdot (\text{somega } (\text{lan } f))))$

using *SOmegaCases* **by** *blast*

have 3: $\bigwedge x. (\lambda x. x \in \text{somega } (\text{lan } f)) x \implies$

$((\exists n. f ((\text{ntaken } n \ x)) \wedge n > 0 \wedge$
 $(\lambda x. x \in \text{somega } (\text{lan } f)) ((\text{ndropn } n \ (x)))))$

using *interval-lan[of - f]* *somega.cases[of - (lan f)]*

by (*metis* *enat-ord-simps(2)* *ndropn-nfuse* *nfinite-nlength-enat* *ntaken-nfuse* *the-enat.simps*
zero-enat-def)

have 4: $\bigwedge x. (\lambda x. x \in \text{somega } (\text{lan } f)) x \implies$

$x \models ((f \wedge \text{more}) \wedge \text{finite}); (\lambda x. x \in \text{somega } (\text{lan } f))$

by (*metis* 3 *FMoreSem-var* *i0-less* *min-enat-simps(2)* *ntaken-nfuse* *ntaken-nlength* *somega.simps*)

have 5: $(\lambda x. x \in \text{somega } (\text{lan } f)) x$

by (*simp* *add: assms*)

show *?thesis* **using** 4 5 *OmegaWeakCoinductSem[of (\lambda x. x \in somega (lan f)) f]*

by (*simp* *add: chop-d-def*)

qed

lemma *somega-omega*:

$x \in \text{somega } (\text{lan } f) \longleftrightarrow (x \models f^\omega)$

using *somega-omega-sem-1* *somega-omega-sem-2* **by** *blast*

lemma *somega-omega-1*:

$\text{somega } (\text{lan } f) = \text{lan}(\text{LIFT}(f^\omega))$

using *somega-omega* **by** *fastforce*

end

Chapter 3

Deep embedding of PITL

```
theory PITL-Deep imports  
  NELList-Extras
```

```
begin
```

This theory is a deep embedding of Propositional Interval Temporal Logic (PITL). It provides the syntax and semantics.

3.1 Syntax

3.1.1 Primitive formulae

```
datatype pitl =  
  false-d                (ifalse)  
| atom-d nat            ((atom - ))  
| iimp-d pitl pitl      (( - iimp - ) [26,25] 25)  
| next-d pitl          ((next - ) [88] 87)  
| chop-d pitl pitl      (( - schop - ) [84,84] 83)  
| schopstar-d pitl      ((schopstar - ) [85] 85)  
| omega-d pitl         ((omega - ) [85] 85)
```

3.1.2 state formula

```
fun state-pitl :: pitl  $\Rightarrow$  bool  
where  
  state-pitl (ifalse) = True  
| state-pitl (atom p) = True  
| state-pitl (f iimp g) = ((state-pitl f)  $\wedge$  (state-pitl g))  
| state-pitl (next f) = False  
| state-pitl (f chop g) = False  
| state-pitl (schopstar f) = False  
| state-pitl (omega f) = False
```

3.1.3 Derived Boolean Operators

```
definition inot-d ((inot - ) [90] 90)
```

where

$inot\ f \equiv f\ iimp\ ifalse$

definition *itrue-d* (*itrue*)

where

$itrue \equiv inot\ ifalse$

definition *ior-d* ((- *ior* -) [31,30] 30)

where

$f\ ior\ g \equiv (inot\ f)\ iimp\ g$

definition *iland-d* ((- *iland* -) [36,35] 35)

where

$f\ iland\ g \equiv inot\ (inot\ f\ ior\ inot\ g)$

definition *ieqv-d* ((- *ieqv* -) [21,20] 20)

where

$f\ ieqv\ g \equiv ((f\ iimp\ g)\ iland\ (g\ iimp\ f))$

3.1.4 more, empty, skip and wnext

definition *more-d* (*more*)

where

$more \equiv next\ itrue$

definition *empty-d* (*empty*)

where

$empty \equiv inot\ more$

definition *skip-d* (*skip*)

where

$skip \equiv next\ empty$

definition *wnext-d* ((*wnext* -) [88] 87)

where

$wnext\ f \equiv inot\ (next\ (inot\ f))$

3.1.5 ifinite and inf

definition *finite-d* (*ifinite*)

where

$ifinite \equiv itrue\ schop\ empty$

definition *inf-d* (*inf*)

where

$inf \equiv inot\ ifinite$

3.1.6 weak chop

definition *chop-d* ((- ; -) [84,84] 83)

where $f ; g \equiv (f \text{ schop } g) \text{ ior } (f \text{ iand } \text{inf})$

3.1.7 Box and Diamond Operators

definition *sometimes-d* $((\text{diamond } -) [88] \ 87)$

where

$\text{diamond } f \equiv \text{itrue schop } f$

definition *always-d* $((\text{box } -) [88] \ 87)$

where

$\text{box } f \equiv \text{inot } (\text{diamond } (\text{inot } f))$

3.1.8 Initial and Final Operators

definition *init-d* $((\text{init } -) [88] \ 87)$

where

$\text{init } f \equiv ((\text{empty iand } f) \text{ schop } \text{itrue})$

definition *fin-d* $((\text{fin } -) [88] \ 87)$

where

$\text{fin } f \equiv \text{diamond } (\text{empty iand } f)$

3.1.9 Big operations

definition *big-ior* $:: \text{ pitl list} \Rightarrow \text{ pitl}$

where $\text{big-ior } F = \text{foldr } (\lambda x y. x \text{ ior } y) F \text{ ifalse}$

definition *big-iand* $:: \text{ pitl list} \Rightarrow \text{ pitl}$

where $\text{big-iand } F = \text{foldr } (\lambda x y. x \text{ iand } y) F \text{ itrue}$

definition *state-pitl-list* $:: \text{ pitl list} \Rightarrow \text{ bool}$

where $\text{state-pitl-list } L = \text{foldr } (\lambda x y. \text{state-pitl } x \wedge y) L \text{ True}$

3.2 Semantics

3.2.1 Intervals

type-synonym *interval* = *nat set nellist*

3.2.2 Semantics Primitive Operators

fun *semantics-pitl* $:: [\text{ interval}, \text{ pitl}] \Rightarrow \text{ bool } ((- \models -) [80,10] \ 10)$

where

$(\sigma \models \text{ifalse}) = \text{False}$

$| (\sigma \models \text{atom } p) = (p \in (\text{nfirst } \sigma))$

$| (\sigma \models f \text{ iimp } g) = ((\sigma \models f) \longrightarrow (\sigma \models g))$

$| (\sigma \models (\text{next } f)) = (1 \leq \text{nlength } \sigma \wedge ((\text{ndropn } 1 \ \sigma) \models f))$

$| (\sigma \models f \text{ schop } g) =$

$(\exists i. ((0 \leq i) \wedge (i \leq (\text{nlength } \sigma)) \wedge ((\text{ntaken } i \ \sigma) \models f) \wedge ((\text{ndropn } i \ \sigma) \models g)))$

| ($\sigma \models (\text{schopstar } f)$) =
 ($\exists (l :: \text{nat nellist}).$
 $(\text{nnth } l \ 0) = 0 \wedge \text{nfinite } l \wedge \text{nidx } l \wedge$
 $(\text{enat } (\text{nlast } l)) = (\text{nlength } \sigma) \wedge \text{nfinite } \sigma \wedge$
 $(\forall (i :: \text{nat}). (\text{enat } i) < (\text{nlength } l)) \longrightarrow$
 $((\text{nsubn } \sigma (\text{nnth } l \ i) (\text{nnth } l (\text{Suc } i))) \models f)$)
)

| ($\sigma \models (\text{omega } f)$) =
 ($\exists (l :: \text{nat nellist}).$
 $\neg \text{nfinite } l \wedge (\text{nnth } l \ 0) = 0 \wedge \text{nidx } l \wedge$
 $(\forall i. (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. ((\text{nsubn } \sigma (\text{nnth } l \ i) (\text{nnth } l (\text{Suc } i))) \models f))$)
)

lemma *inot-defs [simp]:*
 ($\sigma \models \text{inot } f$) = *Not* ($\sigma \models f$)
by (*simp add: inot-d-def*)

lemma *ior-defs [simp]:*
 ($\sigma \models f1 \text{ ior } f2$) = ($\sigma \models f1$) \vee ($\sigma \models f2$)
by (*metis inot-defs ior-d-def semantics-pitl.simps(3)*)

lemma *iand-defs [simp]:*
 ($\sigma \models f1 \text{ iand } f2$) = ($\sigma \models f1$) \wedge ($\sigma \models f2$)
by (*simp add: iand-d-def*)

lemma *ieqv-defs [simp]:*
 ($\sigma \models f1 \text{ ieqv } f2$) = ($\sigma \models f1$) = ($\sigma \models f2$)
by (*metis iand-defs ieqv-d-def semantics-pitl.simps(3)*)

lemma *itrue-defs [simp]:*
 ($\sigma \models \text{itrue}$)
by (*simp add: itrue-d-def*)

lemma *empty-defs [simp]:*
 ($\sigma \models \text{empty}$) = ($\text{nlength } \sigma = 0$)
by (*auto simp add: empty-d-def more-d-def*)
 (*metis co.enat.exhaust-sel enat-le-plus-same(2) plus-1-eSuc(2)*)

lemma *state-pitl-defs:*
assumes *state-pitl w*
shows ($\sigma \models w$) $\longleftrightarrow ((\text{NNil } (\text{nfirst } \sigma)) \models w)$
using *assms*
proof (*induct w*)
case *false-d*
then show ?*case* **by** *simp*
next
case (*atom-d x*)

```

then show ?case
by (metis nfirst-nfuse nfuse-leftneutral nlast-NNil semantics-pitl.simps(2))
next
case (iimp-d w1 w2)
then show ?case by simp
next
case (next-d w)
then show ?case by auto
next
case (chop-d w1 w2)
then show ?case by auto
next
case (schopstar-d w)
then show ?case by auto
next
case (omega-d w)
then show ?case by auto
qed

```

lemma *state-pitl-list-defs*:

```

state-pitl-list L  $\longleftrightarrow$  ( $\forall i < \text{length } L. \text{state-pitl } (L!i)$ )
proof (induct L)
case Nil
then show ?case unfolding state-pitl-list-def by simp
next
case (Cons a L)
then show ?case
proof –
have 1: state-pitl-list (a # L)  $\longleftrightarrow$  state-pitl a  $\wedge$  state-pitl-list L
unfolding state-pitl-list-def by simp
have 2: ( $\forall i < \text{length } (a \# L). \text{state-pitl } ((a \# L)!i)$ )  $\longleftrightarrow$ 
state-pitl ((a # L)!0)  $\wedge$ 
( $\forall i. 0 < i \wedge i < \text{length } (a \# L) \longrightarrow \text{state-pitl } ((a \# L)!i)$ )
by (metis length-greater-0-conv list.discI zero-less-iff-neq-zero)
have 3: state-pitl ((a # L)!0)  $\longleftrightarrow$  state-pitl a
by simp
have 4: ( $\forall i. 0 < i \wedge i < \text{length } (a \# L) \longrightarrow \text{state-pitl } ((a \# L)!i)$ )  $\longleftrightarrow$ 
( $\forall i < \text{length } (L). \text{state-pitl } (L!i)$ )
by auto
show ?thesis
using 1 2 3 4 local.Cons by presburger
qed
qed

```

lemma *big-ior-defs* :

```

( $\sigma \models (\text{big-ior } F)$ )  $\longleftrightarrow$  ( $\exists i < \text{length } F. (\sigma \models F ! i)$ )
proof (induct F)
case Nil

```

```

then show ?case unfolding big-ior-def by simp
next
case (Cons a F)
then show ?case
  proof -
    have 1:  $(\sigma \models (big-ior (a \# F))) \longleftrightarrow (\sigma \models a \text{ ior } (big-ior F))$ 
      unfolding big-ior-def by simp
    have 2:  $(\exists i < length (a \# F). \sigma \models (a \# F) ! i) \longleftrightarrow$ 
       $(\sigma \models (a \# F) ! 0) \vee$ 
       $(\exists i. 0 < i \wedge i < length (a \# F) \wedge (\sigma \models (a \# F) ! i))$ 
      by (metis length-greater-0-conv neq-Nil-conv zero-less-iff-neq-zero)
    have 3:  $(\sigma \models (a \# F) ! 0) = (\sigma \models a)$ 
      by auto
    have 4:  $(\exists i. 0 < i \wedge i < length (a \# F) \wedge (\sigma \models (a \# F) ! i)) \longleftrightarrow$ 
       $(\exists i < length (F). (\sigma \models (F) ! i))$ 
      by auto
      (metis Suc-le-eq diff-Suc-Suc diff-zero gr0-conv-Suc less-Suc-eq-le)
    show ?thesis
      using 1 2 4 local.Cons by force
  qed
qed

```

```

lemma big-iand-defs :
   $(\sigma \models (big-iand F)) \longleftrightarrow (\forall i < length F. (\sigma \models F ! i))$ 
proof (induct F)
case Nil
then show ?case unfolding big-iand-def by simp
next
case (Cons a F)
then show ?case
  proof -
    have 1:  $(\sigma \models (big-iand (a \# F))) \longleftrightarrow (\sigma \models a \text{ iand } (big-iand F))$ 
      unfolding big-iand-def by simp
    have 2:  $(\forall i < length (a \# F). \sigma \models (a \# F) ! i) \longleftrightarrow$ 
       $(\sigma \models (a \# F) ! 0) \wedge$ 
       $(\forall i. 0 < i \wedge i < length (a \# F) \longrightarrow (\sigma \models (a \# F) ! i))$ 
      by (metis length-greater-0-conv neq-Nil-conv zero-less-iff-neq-zero)
    have 3:  $(\sigma \models (a \# F) ! 0) = (\sigma \models a)$ 
      by auto
    have 4:  $(\forall i. 0 < i \wedge i < length (a \# F) \longrightarrow (\sigma \models (a \# F) ! i)) \longleftrightarrow$ 
       $(\forall i < length (F). (\sigma \models (F) ! i))$ 
      by auto
    show ?thesis
      using 1 2 4 local.Cons by force
  qed
qed

```

lemma big-ior-map-iand-zip :

```

assumes  $length\ F = length\ G$ 
shows  $(\sigma \models big\_ior\ (map2\ iand\_d\ F\ G)) \longleftrightarrow$ 
 $(\exists\ i < length\ F. (\sigma \models F\ !\ i\ iand\ G\ !\ i))$ 
using assms
proof (induct F arbitrary: G)
case Nil
then show ?case by (simp add: big-ior-defs)
next
case (Cons a F)
then show ?case
  proof (cases G=[])
  case True
  then show ?thesis using Cons.prems by auto
  next
  case False
  then show ?thesis using Cons by (auto simp add: big-ior-defs)
  qed
qed

```

```

lemma big-iand-map-iimp-zip :
assumes  $length\ F = length\ G$ 
shows  $(\sigma \models big\_iand\ (map2\ iimp\_d\ F\ G)) \longleftrightarrow$ 
 $(\forall\ i < length\ F. (\sigma \models F\ !\ i\ iimp\ G\ !\ i))$ 
using assms
proof (induct F arbitrary: G)
case Nil
then show ?case by (simp add: big-iand-defs)
next
case (Cons a F)
then show ?case
  proof (cases G=[])
  case True
  then show ?thesis using Cons.prems by auto
  next
  case False
  then show ?thesis using Cons by (auto simp add: big-iand-defs)
  qed
qed

```

```

lemma map2-iand-defs :
assumes  $length\ F = length\ G$ 
shows  $(\exists\ i < length\ G. \sigma \models map2\ iand\_d\ F\ G\ !\ i) \longleftrightarrow$ 
 $(\exists\ i < length\ G. (\sigma \models F\ !\ i) \wedge (\sigma \models G\ !\ i))$ 
using assms
proof (induct G arbitrary: F)
case Nil
then show ?case by simp
next
case (Cons a G)

```



```

then show ?case
  proof (cases F=[])
  case True
  then show ?thesis using Cons.prem by auto
  next
  case False
  then show ?thesis using Cons by auto
  qed
qed

lemma map2-iand-inot-defs :
  assumes length F = length G
  shows  $(\exists i < \text{length } G. \sigma \models \text{map2 } (\lambda x y. x \text{ iand inot } y) F G ! i) \longleftrightarrow$ 
     $(\exists i < \text{length } G. (\sigma \models F ! i) \wedge \neg(\sigma \models G ! i))$ 
  using assms
  proof (induct G arbitrary: F)
  case Nil
  then show ?case by simp
  next
  case (Cons a G)
  then show ?case
    proof (cases F=[])
    case True
    then show ?thesis using Cons.prem by auto
    next
    case False
    then show ?thesis using Cons by auto
    qed
  qed

```

3.2.3 Validity

definition *valid* :: *pitl* \Rightarrow *bool* ((\vdash -) 5)
 where $(\vdash f) = (\forall \sigma. (\sigma \models f))$

lemma *itl-valid* [*simp*] :
 $(\vdash f) = (\forall \sigma. (\sigma \models f))$
 by (*simp add: valid-def*)

lemma *itl-ieq*:
 $(\vdash f \text{ ieqv } g) = (\forall \sigma. (\sigma \models f) = (\sigma \models g))$
 by (*auto simp add: ieqv-d-def iand-d-def ior-d-def inot-d-def*)

lemma *big-ior-iimp-subset-eq*:
 assumes $\text{set } L1 \subseteq \text{set } L2$
 shows $\vdash \text{big-ior } L1 \text{ iimp } \text{big-ior } L2$
 using assms
 proof (induct rule: list-induct2')
 case 1
 then show ?case by simp

```

next
case (2 x xs)
then show ?case by simp
next
case (3 y ys)
then show ?case by (simp add: big-ior-defs)
next
case (4 x xs y ys)
then show ?case
  proof (cases x=y)
  case True
  then show ?thesis using 4 by (auto simp add: big-ior-defs)
  (metis diff-Suc-1 in-set-conv-nth length-Cons less-Suc-eq-0-disj list.simps(15) nth-Cons' subsetD)
  next
  case False
  then show ?thesis using 4 by (auto simp add: big-ior-defs)
  (metis 4.prem1 in-set-conv-nth length-Cons subsetD)
  qed
qed

```

```

lemma big-ior-ieqv-set-eq:
  assumes set L1 = set L2
  shows  $\vdash$  big-ior L1 ieqv big-ior L2
using assms
by (metis big-ior-iimp-subset-eq dual-order.refl iand-defs ieqv-d-def valid-def)

```

```

lemma big-ior-ieqv-remdups:
  shows  $\vdash$  big-ior L ieqv big-ior (remdups L)
by (metis big-ior-ieqv-set-eq set-remdups)

```

```

lemma big-iand-iimp-subset-eq:
  assumes set L1  $\subseteq$  set L2
  shows  $\vdash$  big-iand L2 iimp big-iand L1
using assms
proof (induct rule: list-induct2')
case 1
then show ?case by simp
next
case (2 x xs)
then show ?case by (simp add: big-iand-defs)
next
case (3 y ys)
then show ?case by simp
next
case (4 x xs y ys)
then show ?case
  proof (cases x=y)
  case True
  then show ?thesis using 4 by (auto simp add: big-iand-defs)

```

```

(metis (mono-tags, lifting) 4.premis all-nth-imp-all-set length-Cons list-ball-nth subsetD)
next
case False
then show ?thesis using 4 by (auto simp add: big-iand-defs)
(metis 4.premis add.right-neutral add-Suc-right in-set-conv-nth list.size(4) subsetD)
qed
qed

```

```

lemma big-iand-ieqv-set-eq:
  assumes set L1 = set L2
  shows  $\vdash \text{big-iand } L1 \text{ ieqv big-iand } L2$ 
using assms
by (metis big-iand-iimp-subset-eq dual-order.refl iand-defs ieqv-d-def valid-def)

```

```

lemma big-iand-ieqv-remdups:
   $\vdash \text{big-iand } L \text{ ieqv big-iand (remdups } L)$ 
by (metis big-iand-ieqv-set-eq set-remdups)

```

```

lemma big-ior-empty:
   $(\vdash \text{big-ior } []) \longleftrightarrow (\vdash \text{ifalse})$ 
by (auto simp add: big-ior-defs)

```

```

lemma big-ior-Cons:
   $(\vdash \text{big-ior } (f \# L)) \longleftrightarrow (\vdash f \text{ ior (big-ior } L))$ 
proof -
  have 1:  $(\vdash \text{big-ior } (f \# L)) \implies (\vdash f \text{ ior (big-ior } L))$ 
  by (auto simp add: big-ior-defs)
  (metis diff-Suc-1 gr0-conv-Suc less-Suc-eq less-imp-diff-less nth-Cons')
  have 2:  $(\vdash f \text{ ior (big-ior } L)) \implies (\vdash \text{big-ior } (f \# L))$ 
  by (auto simp add: big-ior-defs) fastforce
  show ?thesis using 1 2 by auto
qed

```

```

lemma big-ior-Cons-alt:
   $\vdash \text{big-ior } (f \# L) \text{ ieqv } f \text{ ior big-ior } L$ 
unfolding big-ior-def
by auto

```

```

lemma big-ior-1:
  assumes  $\bigwedge i. i < \text{length } F \implies (\vdash F ! i \text{ ieqv } G ! i)$ 
           $\text{length } F = \text{length } G$ 
  shows  $\vdash (\text{big-ior } F) \text{ ieqv } (\text{big-ior } G)$ 
using assms
by (auto simp add: big-ior-defs)

```

lemma *big-iand-1*:
assumes $\bigwedge i. i < \text{length } F \implies (\vdash F ! i \text{ ieqv } G ! i)$
 $\text{length } F = \text{length } G$
shows $\vdash (\text{big-iand } F) \text{ ieqv } (\text{big-iand } G)$
using *assms*
by (*auto simp add: big-iand-defs*)

lemma *big-ior-big-iand-1*:
 $\vdash (\text{inot } (\text{big-ior } F)) \text{ ieqv } (\text{big-iand } (\text{map } (\lambda x. \text{inot } x) F))$
by (*auto simp add: big-ior-defs big-iand-defs*)

lemma *big-ior-big-iand-2*:
 $\vdash (\text{inot } (\text{big-iand } F)) \text{ ieqv } (\text{big-ior } (\text{map } (\lambda x. \text{inot } x) F))$
by (*auto simp add: big-ior-defs big-iand-defs*)

lemma *big-ior-append*:
 $(\vdash (\text{big-ior } (L1 @ L2)) \text{ ieqv } (\text{big-ior } L1 \text{ ior } (\text{big-ior } L2)))$
proof (*induct L1 arbitrary: L2*)
case *Nil*
then show *?case* **by** (*simp add: big-ior-defs*)
next
case (*Cons a L1a*)
then show *?case*
proof –
have 1: $\vdash \text{big-ior } ((a \# L1a) @ L2) \text{ ieqv } a \text{ ior } ((\text{big-ior } (L1a @ L2)))$
using *big-ior-Cons-alt* **by** *auto*
have 2: $\vdash \text{big-ior } (L1a @ L2) \text{ ieqv } \text{big-ior } L1a \text{ ior } \text{big-ior } L2$
using *local.Cons* **by** *blast*
have 3: $\vdash a \text{ ior } ((\text{big-ior } (L1a @ L2))) \text{ ieqv } a \text{ ior } (\text{big-ior } L1a \text{ ior } \text{big-ior } L2)$
using 1 2 **by** *auto*
have 4: $\vdash \text{big-ior } (a \# L1a) \text{ ieqv } a \text{ ior } (\text{big-ior } L1a)$
using *big-ior-Cons-alt* **by** *blast*
have 5: $\vdash a \text{ ior } (\text{big-ior } L1a \text{ ior } \text{big-ior } L2) \text{ ieqv } \text{big-ior } (a \# L1a) \text{ ior } \text{big-ior } L2$
using 4 **by** *auto*
show *?thesis* **using** 1 3 5 **by** *force*
qed
qed

lemma *state-pitl-list-append*:
shows $\text{state-pitl-list } (L @ L1) \longleftrightarrow \text{state-pitl-list } L \wedge \text{state-pitl-list } L1$
proof (*induct L arbitrary: L1*)
case *Nil*
then show *?case*
by (*simp add: state-pitl-list-defs*)
next
case (*Cons a L*)
then show *?case*
proof –

```

have 1: state-pitl-list ((a # L) @ L1)  $\longleftrightarrow$  state-pitl a  $\wedge$  state-pitl-list (L @ L1)
  unfolding state-pitl-list-def by auto
have 2: state-pitl-list (L @ L1)  $\longleftrightarrow$  state-pitl-list L  $\wedge$  state-pitl-list L1
  by (simp add: local.Cons)
have 3: state-pitl a  $\wedge$  state-pitl-list L  $\wedge$  state-pitl-list L1  $\longleftrightarrow$ 
  (state-pitl-list (a # L)  $\wedge$  state-pitl-list L1)
  by (simp add: state-pitl-list-def)
show ?thesis using 1 3 local.Cons by auto
qed
qed

```

```

lemma map2-map:
assumes length L1 = length L2
shows map2 f (map g L1) (map h L2) = map2 ( $\lambda x y. f (g x) (h y)$ ) L1 L2
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by simp
next
case (Cons x xs y ys)
then show ?case by auto
qed

```

```

lemma map2-map-a:
assumes length L1 = length L2
shows (map2 ( $\lambda x y. x$  iand next y) (map ( $\lambda x1. x$  iand x1) L1) (map ( $\lambda y1. y$  ior y1) L2)) =
  (map2 ( $\lambda x1 y1. (x$  iand x1) iand next (y ior y1)) L1 L2)
using assms map2-map by blast

```

```

lemma map2-map-a-a:
assumes length L1 = length L2
shows (map2 ( $\lambda x y. x$  iand next y)
  (map ( $\lambda x1. x$  iand x1) L1)
  (map ( $\lambda y1. (if y = ifalse$  then y1 else y ior y1)) L2)) =
  (map2 ( $\lambda x1 y1. (x$  iand x1) iand next (if y = ifalse then y1 else y ior y1))) L1 L2)
using assms map2-map by blast

```

```

lemma map2-map-b:
assumes length L1 = length L2
shows (map2 ( $\lambda x y. x$  iand next y) (map ( $\lambda x1. (inot x)$  iand x1) L1) L2) =
  (map2 ( $\lambda x1 y1. ((inot x)$  iand x1) iand next y1) L1 L2)
using assms map2-map[of L1 L2 ( $\lambda x y. x$  iand next y) ( $\lambda x1. (inot x)$  iand x1) id]
by auto

```

```

lemma big-ior-big-ior-map2:
assumes length L1 = length L2

```

```

    ⊢ big-ior L1
shows ⊢ big-ior (map2 (λx1 y1. x1) L1 L2)
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by simp
next
case (Cons x xs y ys)
then show ?case
by (metis fst-def list.size(4) map-eq-conv map-fst-zip)
qed

```

lemma *big-ior-split-off*:

assumes $\text{length } L1 = \text{length } L2$

($\vdash \text{big-ior } L1$)

shows $\vdash (\text{big-ior } (\text{map2 } (\lambda x1 y1. (x \text{ iand } x1) \text{ iand } (\text{next } (y \text{ ior } y1))) L1 L2)) \text{ ior}$
 $(\text{big-ior } (\text{map2 } (\lambda x1 y1. (\text{inot } x \text{ iand } x1) \text{ iand } \text{next } y1) L1 L2)) \text{ ieqv}$
 $(x \text{ iand } \text{next } y) \text{ ior}$
 $(\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) L1 L2))$

proof –

have 1: $\vdash (\text{big-ior } (\text{map2 } (\lambda x1 y1. (x \text{ iand } x1) \text{ iand } (\text{next } (y \text{ ior } y1))) L1 L2)) \text{ ieqv}$
 $(x \text{ iand } (\text{big-ior } (\text{map2 } (\lambda x1 y1. x1 \text{ iand } (\text{next } (y \text{ ior } y1))) L1 L2)))$

by (auto simp add: big-ior-defs)

have 2: $\vdash (\text{big-ior } (\text{map2 } (\lambda x1 y1. x1 \text{ iand } (\text{next } (y \text{ ior } y1))) L1 L2)) \text{ ieqv}$
 $((\text{next } y \text{ iand } (\text{big-ior } (\text{map2 } (\lambda x1 y1. x1) L1 L2)))$
 ior
 $(\text{big-ior } (\text{map2 } (\lambda x1 y1. x1 \text{ iand } \text{next } (y1)) L1 L2)))$

by (auto simp add: big-ior-defs)

have 3: $\vdash (\text{big-ior } (\text{map2 } (\lambda x1 y1. (x \text{ iand } x1) \text{ iand } (\text{next } (y \text{ ior } y1))) L1 L2)) \text{ ieqv}$
 $(x \text{ iand } (\text{next } y \text{ iand } \text{big-ior } (\text{map2 } (\lambda x1 y1. x1) L1 L2)))$
 ior
 $(\text{big-ior } (\text{map2 } (\lambda x1 y1. x1 \text{ iand } \text{next } (y1)) L1 L2)))$

by (auto simp add: big-ior-defs)

have 4: $\vdash \text{big-ior } (\text{map2 } (\lambda x1 y1. x1) L1 L2)$

using assms *big-ior-big-ior-map2* **by** blast

have 5: $\vdash (x \text{ iand } (\text{next } y \text{ iand } \text{big-ior } (\text{map2 } (\lambda x1 y1. x1) L1 L2)))$
 ior
 $(\text{big-ior } (\text{map2 } (\lambda x1 y1. x1 \text{ iand } \text{next } (y1)) L1 L2))) \text{ ieqv}$
 $(x \text{ iand } \text{next } y) \text{ ior } (x \text{ iand } (\text{big-ior } (\text{map2 } (\lambda x1 y1. x1 \text{ iand } \text{next } (y1)) L1 L2)))$

using 4 **by** auto

have 6: $\vdash (\text{big-ior } (\text{map2 } (\lambda x1 y1. (\text{inot } x \text{ iand } x1) \text{ iand } \text{next } y1) L1 L2)) \text{ ieqv}$
 $(\text{inot } x \text{ iand } (\text{big-ior } (\text{map2 } (\lambda x1 y1. x1 \text{ iand } (\text{next } (y1))) L1 L2)))$

by (auto simp add: big-ior-defs)

have 7: $\vdash (x \text{ iand } (\text{big-ior } (\text{map2 } (\lambda x1 y1. x1 \text{ iand } \text{next } (y1)) L1 L2))) \text{ ior}$
 $(\text{inot } x \text{ iand } (\text{big-ior } (\text{map2 } (\lambda x1 y1. x1 \text{ iand } \text{next } (y1)) L1 L2))) \text{ ieqv}$
 $(\text{big-ior } (\text{map2 } (\lambda x1 y1. x1 \text{ iand } \text{next } (y1)) L1 L2))$

by (auto simp add: big-ior-defs)

have 8: $\vdash (\text{big-ior } (\text{map2 } (\lambda x1 y1. (x \text{ iand } x1) \text{ iand } (\text{next } (y \text{ ior } y1))) L1 L2)) \text{ ior}$
 $(\text{big-ior } (\text{map2 } (\lambda x1 y1. (\text{inot } x \text{ iand } x1) \text{ iand } \text{next } y1) L1 L2)) \text{ ieqv}$

```

      (x iand next y) ior
      (x iand (big-ior (map2 (λx1 y1. x1 iand next ( y1)) L1 L2 ))) ior
      (inot x iand (big-ior (map2 (λx1 y1. x1 iand next ( y1)) L1 L2 )))
    using 5 6 3 by auto
  show ?thesis using 7 8 by auto
qed

lemma big-ior-split-off-a:
assumes length L1 = length L2
  (⊢ big-ior L1)
shows ⊢ (big-ior (map2 (λx1 y1. (x iand x1) iand ( next ((if y = ifalse then y1 else y ior y1)))) L1 L2 ))
ior
  (big-ior (map2 (λx1 y1. (inot x iand x1) iand next y1)      L1 L2 )) ieqv
  (x iand next y) ior
  (big-ior (map2 (λx y. x iand next y) L1 L2 ))
proof -
  have 1: ⊢ (big-ior (map2 (λx1 y1. (x iand x1) iand ( next ((if y = ifalse then y1 else y ior y1)))) L1 L2
  )) ieqv
    (x iand (big-ior (map2 (λx1 y1. x1 iand ( next ((if y = ifalse then y1 else y ior y1)))) L1 L2 )))
  by (auto simp add: big-ior-defs)
  have 2: ⊢ (big-ior (map2 (λx1 y1. x1 iand ( next ((if y = ifalse then y1 else y ior y1)))) L1 L2 )) ieqv
    ((next y iand (big-ior (map2 (λx1 y1. x1 ) L1 L2 )))
    ior
    (big-ior (map2 (λx1 y1. x1 iand next ( y1)) L1 L2 )))
  proof (cases y=ifalse)
  case True
  then show ?thesis by simp
  next
  case False
  then show ?thesis by (auto simp add: big-ior-defs)
  qed
  have 3: ⊢ (big-ior (map2 (λx1 y1. (x iand x1) iand ( next (if y = ifalse then y1 else y ior y1))) L1 L2 ))
  ieqv
    (x iand (next y iand big-ior (map2 (λx1 y1. x1) L1 L2 )
    ior
    (big-ior (map2 (λx1 y1. x1 iand next ( y1)) L1 L2 ))))
  proof (cases y=ifalse)
  case True
  then show ?thesis by (auto simp add: big-ior-defs)
  next
  case False
  then show ?thesis by (auto simp add: big-ior-defs)
  qed
  have 4: ⊢ big-ior (map2 (λx1 y1. x1 ) L1 L2 )
    using assms big-ior-big-ior-map2 by blast
  have 5: ⊢ (x iand (next y iand big-ior (map2 (λx1 y1. x1 ) L1 L2 )
    ior
    (big-ior (map2 (λx1 y1. x1 iand next ( y1)) L1 L2 )))) ieqv
    (x iand next y) ior (x iand (big-ior (map2 (λx1 y1. x1 iand next ( y1)) L1 L2 )))
  using 4 by auto

```

```

have 6:  $\vdash$  (big-ior (map2 ( $\lambda x1\ y1.$  (inot x iand x1) iand next y1) L1 L2 )) ieqv
  (inot x iand (big-ior (map2 ( $\lambda x1\ y1.$  x1 iand (next (y1))) L1 L2 )))
  by (auto simp add: big-ior-defs)
have 7:  $\vdash$  (x iand (big-ior (map2 ( $\lambda x1\ y1.$  x1 iand next (y1)) L1 L2 ))) ior
  (inot x iand (big-ior (map2 ( $\lambda x1\ y1.$  x1 iand next (y1)) L1 L2 ))) ieqv
  (big-ior (map2 ( $\lambda x1\ y1.$  x1 iand next (y1)) L1 L2 ))
  by (auto simp add: big-ior-defs)
have 8:  $\vdash$  (big-ior (map2 ( $\lambda x1\ y1.$  (x iand x1) iand (next (if y = ifalse then y1 else y ior y1)))) L1 L2
  )) ior
  (big-ior (map2 ( $\lambda x1\ y1.$  (inot x iand x1) iand next y1) L1 L2 )) ieqv
  (x iand next y) ior
  (x iand (big-ior (map2 ( $\lambda x1\ y1.$  x1 iand next (y1)) L1 L2 ))) ior
  (inot x iand (big-ior (map2 ( $\lambda x1\ y1.$  x1 iand next (y1)) L1 L2 )))
  using 5 6 3 by auto
show ?thesis using 7 8 by auto
qed

```

```

lemma state-pitl-list-map:
assumes state-pitl f
  state-pitl-list L
shows state-pitl-list (map ( $\lambda x. f\ iand\ x$ ) L)
using assms
proof (induct L)
case Nil
then show ?case by simp
next
case (Cons a L)
then show ?case
  proof –
    have 1: state-pitl-list (map ( $\lambda x. f\ iand\ x$ ) (a # L))  $\longleftrightarrow$ 
      state-pitl (f iand a)  $\wedge$  state-pitl-list ((map ( $\lambda x. f\ iand\ x$ ) L))
    unfolding state-pitl-list-def by auto
    have 2: state-pitl-list (a # L)  $\longleftrightarrow$  state-pitl a  $\wedge$  state-pitl-list L
    unfolding state-pitl-list-def by auto
    have 3: state-pitl (f iand a)
      unfolding iand-d-def inot-d-def ior-d-def
      using 2 Cons.prems(2) assms(1) by auto
    show ?thesis
    using 1 2 3 Cons.hyps Cons.prems(2) assms(1) by blast
  qed
qed

```

```

lemma state-pitl-list-map-alt:
assumes state-pitl f
  state-pitl-list L
shows state-pitl-list (map ( $\lambda x. x\ iand\ f$ ) L)
using assms
proof (induct L)
case Nil

```



```

then show ?case by simp
next
case (Cons a L)
then show ?case
proof -
  have 1: state-pitl-list (map ( $\lambda x. x \text{ iand } f$ ) (a # L))  $\longleftrightarrow$ 
    state-pitl (a iand f)  $\wedge$  state-pitl-list ((map ( $\lambda x. x \text{ iand } f$ ) L))
    unfolding state-pitl-list-def by auto
  have 2: state-pitl-list (a # L)  $\longleftrightarrow$  state-pitl a  $\wedge$  state-pitl-list L
    unfolding state-pitl-list-def by auto
  have 3: state-pitl (a iand f)
    unfolding iand-d-def inot-d-def ior-d-def
    using 2 Cons.premis(2) assms(1) by auto
  show ?thesis
  using 1 2 3 Cons.hyps Cons.premis(2) assms(1) by blast
qed
qed

```

```

lemma map-fst-filter-sat:
assumes length L1 = length L2
shows map fst (filter ( $\lambda (x,y). (\exists \sigma. \sigma \models x)$ ) (zip L1 L2)) =
  filter ( $\lambda x. (\exists \sigma. \sigma \models x)$ ) L1
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by simp
next
case (Cons x xs y ys)
then show ?case
proof (cases xs = []  $\wedge$  ys = [])
case True
then show ?thesis by simp
next
case False
then show ?thesis
  using Cons by auto
qed
qed

```

```

lemma big-ior-ieqv-filter-sat:
 $\vdash \text{big-ior } L \text{ ieqv } \text{big-ior } ( \text{filter } (\lambda x. (\exists \sigma. \sigma \models x)) L )$ 
proof (induct L)
case Nil
then show ?case by simp
next
case (Cons a L)
then show ?case
  using big-ior-Cons-alt by auto
qed

```

```

lemma total-filter-sat:
  assumes  $\vdash \text{big-ior } L$ 
  shows  $\vdash \text{big-ior } (\text{filter } (\lambda x. (\exists \sigma. \sigma \models x)) L)$ 
using assms
using big-ior-ieqv-filter-sat by auto

lemma big-ior-filter-sat-ieqv:
assumes  $\text{length } L1 = \text{length } L2$ 
shows  $\vdash \text{big-ior } (\text{map } (\lambda (x,y). x \text{ iand } (\text{next } y))$ 
       $(\text{filter } (\lambda (x,y). (\exists \sigma. \sigma \models x)) (\text{zip } L1 L2)))$ 
      ieqv
       $\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) L1 L2)$ 
using assms
proof (induct  $L1 L2$  rule: list-induct2)
case Nil
then show ?case by (simp add: big-ior-def)
next
case (Cons  $x xs y ys$ )
then show ?case
  proof (cases  $xs = [] \wedge ys = []$ )
    case True
    then show ?thesis using Cons
      unfolding big-ior-def big-iand-def
      by simp
    next
    case False
    then show ?thesis
      proof –
        have 1: length xs = length ys
          by (simp add: Cons.hyps(1))
        have 2:  $\vdash \text{big-ior } (\text{map } (\lambda(x, y). x \text{ iand } \text{next } y)$ 
           $(\text{filter } (\lambda(x, y). \exists \sigma. \sigma \models x)$ 
           $(\text{zip } xs ys)))$ 
          ieqv big-ior (map2 (λx y. x iand next y) xs ys)
          using Cons.hyps(2) False by fastforce
        show ?thesis
          using 2 big-ior-Cons-alt by auto
      qed
    qed
  qed

```

```

lemma big-ior-rev-ieqv:
   $\vdash \text{big-ior } L \text{ ieqv } \text{big-ior } (\text{rev } L)$ 
by (auto simp add: big-ior-defs)
  (metis in-set-conv-nth length-rev set-rev,
   metis diff-less length-greater-0-conv less-nat-zero-code list.size(3) rev-nth zero-less-Suc)

```

```

lemma big-ior-map2-append:
assumes length L1= length N1
          length L2= length N2
shows  $\vdash (big-ior (map2 (\lambda x y . x \text{ iand } (next\ y)) (L1@L2) (N1@N2))) \text{ ieqv}$ 
         $(big-ior (map2 (\lambda x y . x \text{ iand } (next\ y)) L1\ N1)) \text{ ior}$ 
         $(big-ior (map2 (\lambda x y . x \text{ iand } (next\ y)) L2\ N2))$ 
using assms
proof (induct L1 N1 arbitrary: L2 N2 rule:list-induct2)
case Nil
then show ?case by (simp add: big-ior-defs)
next
case (Cons x xs y ys)
then show ?case
  by (metis Cons.hyps(1) big-ior-append length-Cons map-append zip-append)

qed

```

```

lemma map2-iand-next-defs :
assumes length F = length G
shows  $(\exists i < \text{length } G. \sigma \models \text{map2 } (\lambda x y . x \text{ iand } (next\ y))\ F\ G\ !\ i) \longleftrightarrow$ 
         $(\exists i < \text{length } G. (\sigma \models F\ !\ i) \wedge (\sigma \models (next\ (G\ !\ i))))$ 
using assms
proof (induct G arbitrary: F)
case Nil
then show ?case by simp
next
case (Cons a G)
then show ?case
  proof (cases F=[])
    case True
    then show ?thesis using Cons.prems by auto
    next
    case False
    then show ?thesis using Cons by auto
  qed
qed

```

```

lemma map2-iand-inot-next-defs :
assumes length F = length G
          0 < nlength  $\sigma$ 
shows  $(\exists i < \text{length } G. \sigma \models \text{map2 } (\lambda x y . x \text{ iand } (next\ (inot\ y)))\ F\ G\ !\ i) \longleftrightarrow$ 
         $(\exists i < \text{length } G. (\sigma \models F\ !\ i) \wedge \neg(\sigma \models (next\ (G\ !\ i))))$ 
using assms
proof (induct G arbitrary: F)
case Nil
then show ?case by simp
next
case (Cons a G)
then show ?case

```

```

proof (cases F=[] )
case True
then show ?thesis using Cons by simp
next
case False
then show ?thesis
  proof -
    obtain b F1 where 1: F = b#F1
    using False by (meson neq-Nil-conv)
    have 2: length F1 = length G
      using 1 Cons.prem1 by auto
    have 3: (∃ i < length G. σ ⊢ map2 (λx y. x iand next inot y) F1 G ! i) =
      (∃ i < length G. (σ ⊢ F1 ! i) ∧ ¬ (σ ⊢ next G ! i))
      using 2 Cons.hyps assms(2) by presburger
    have 4: (∃ i < length (a # G). σ ⊢ map2 (λx y. x iand next inot y) F (a # G) ! i) ⟷
      (σ ⊢ map2 (λx y. x iand next inot y) (b#F1) (a # G) ! 0) ∨
      (∃ i. 0 < i ∧ i < length (a # G) ∧ (σ ⊢ map2 (λx y. x iand next inot y) (b#F1) (a # G) ! i))
      using 1 by (metis gr-zeroI length-greater-0-conv list.discI)
    have 5: (σ ⊢ map2 (λx y. x iand next inot y) (b#F1) (a # G) ! 0) ⟷
      (σ ⊢ b iand next inot a)
      by auto
    have 6: (∃ i. 0 < i ∧ i < length (a # G) ∧ (σ ⊢ map2 (λx y. x iand next inot y) (b#F1) (a # G) !
i)) ⟷
      (∃ i < length (G). (σ ⊢ map2 (λx y. x iand next inot y) (F1) (G) ! i))
      by auto
      (metis Suc-less-eq Suc-pred)
    have 7: (∃ i < length (a # G). (σ ⊢ (b#F1) ! i) ∧ ¬ (σ ⊢ next (a # G) ! i)) ⟷
      ((σ ⊢ (b#F1) ! 0) ∧ ¬ (σ ⊢ next (a # G) ! 0)) ∨
      (∃ i. 0 < i ∧ i < length (a # G) ∧ (σ ⊢ (b#F1) ! i) ∧ ¬ (σ ⊢ next (a # G) ! i))
      by (metis gr-zeroI length-greater-0-conv list.discI)
    have 8: ((σ ⊢ (b#F1) ! 0) ∧ ¬ (σ ⊢ next (a # G) ! 0)) ⟷
      (σ ⊢ b iand next inot a)
      by simp
      (metis One-nat-def Suc-ile-eq assms(2) enat-defs(1) enat-defs(2))
    have 9: (∃ i. 0 < i ∧ i < length (a # G) ∧ (σ ⊢ (b#F1) ! i) ∧ ¬ (σ ⊢ next (a # G) ! i)) ⟷
      (∃ i < length (G). (σ ⊢ (F1) ! i) ∧ ¬ (σ ⊢ next (G) ! i))
      by auto
      (metis Suc-diff-Suc Suc-less-eq diff-zero, metis Suc-less-eq Suc-pred)
    show ?thesis
    using 1 3 4 5 6 7 8 9 by presburger
  qed
qed
qed

```

```

lemma iand-big-ior-dist:
assumes length G-now = length G-next
shows ⊢ F-e iand big-ior (map2 (λ x y. x iand next y) G-now G-next) ieqv
  big-ior (map2 (λ x y. x iand next y)
    (map (λx. x iand F-e) G-now) G-next)
using assms by (auto simp add: big-ior-defs)

```

```

lemma big-ior-map2-defs:
  assumes length L1 = length L2
  shows  $(\sigma \models (\text{big-ior } (\text{map2 } (\lambda x y. f x y) L1 L2))) \longleftrightarrow$ 
     $(\exists i < \text{length } L1. (\sigma \models f (L1!i) (L2!i)))$ 
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case unfolding big-ior-def by auto
next
case (Cons x xs y ys)
then show ?case unfolding big-ior-def using less-Suc-eq-0-disj by auto
qed

```

```

lemma big-ior-map2-schop-dis:
  assumes length L1 = length L2
  shows  $\vdash (\text{big-ior } (\text{map2 } (\lambda x y. f x y) L1 L2)) \text{schop } g \text{ieqv}$ 
     $(\text{big-ior } (\text{map2 } (\lambda x y. (f x y) \text{schop } g) L1 L2))$ 
using assms
by (auto simp add: big-ior-map2-defs)

```

end

```

theory PITL-Deep-GNF imports
  PITL-Deep

```

begin

This theory proves that all PITL formulae can be written in Guarded Normal Form (GNF) [4, 2].

3.3 Guarded Normal Form

3.3.1 Definitions

```

definition mutex
where mutex F =
   $(\forall k1 k2. 0 \leq k1 \wedge k1 < k2 \wedge k2 < \text{length } F \longrightarrow (\vdash \text{inot } ((F! k1) \text{iand } (F! k2))))$ 

```

```

definition mutex-single
where mutex-single f F =
   $(\forall k2. 0 \leq k2 \wedge k2 < \text{length } F \longrightarrow (\vdash \text{inot } (f \text{iand } (F! k2))))$ 

```

definition *add-to-now-single*

where

$$\text{add-to-now-single } f \ L = ((\text{map } (\lambda x. (f \text{ iand } x)) \ L) @ (\text{map } (\lambda x. (\text{inot } f \text{ iand } x)) \ L))$$

definition *add-to-next-single*

where

$$\text{add-to-next-single } f \ L = ((\text{map } (\lambda x. (f \text{ ior } x)) \ L) @ \ L)$$

definition *add-to-single*

where

$$\begin{aligned} \text{add-to-single } f1 \ f2 \ L1 \ L2 = \\ \text{filter } (\lambda (x,y). (\exists \ \sigma. \ \sigma \models x)) \ (\text{zip } (\text{add-to-now-single } f1 \ L1) \ (\text{add-to-next-single } f2 \ L2)) \end{aligned}$$

definition *add-to-next-single-a*

where

$$\text{add-to-next-single-a } f \ L = ((\text{map } (\lambda x. (\text{if } f = \text{ifalse then } x \text{ else } f \text{ ior } x)) \ L) @ \ L)$$

definition *add-to-now*

where *add-to-now* $L1 = \text{foldr } (\text{add-to-now-single}) \ L1 \ [\text{itrue}]$

definition *add-to-next*

where *add-to-next* $L1 = \text{foldr } (\text{add-to-next-single}) \ L1 \ [\text{ifalse}]$

definition *add-to-next-a*

where *add-to-next-a* $L1 = \text{foldr } (\text{add-to-next-single-a}) \ L1 \ [\text{ifalse}]$

definition *add-to*

where *add-to* $L1 \ L2 = \text{filter } (\lambda (x,y). (\exists \ \sigma. \ \sigma \models x)) \ (\text{zip } (\text{add-to-now } L1) \ (\text{add-to-next } L2))$

fun *GNF-empty* :: *pitl* \Rightarrow *pitl*

where

$$\begin{aligned} & \text{GNF-empty } (\text{ifalse}) &= \text{ifalse} \\ | & \text{GNF-empty } (\text{atom } p) &= (\text{atom } p) \\ | & \text{GNF-empty } (f \text{ iimp } g) &= ((\text{inot } (\text{GNF-empty } f)) \text{ ior } (\text{GNF-empty } g)) \\ | & \text{GNF-empty } (\text{next } f) &= (\text{ifalse}) \\ | & \text{GNF-empty } (f \text{ schop } g) &= ((\text{GNF-empty } f) \text{ iand } (\text{GNF-empty } g)) \\ | & \text{GNF-empty } (\text{schopstar } f) &= \text{itrue} \\ | & \text{GNF-empty } (\text{omega } f) &= \text{ifalse} \end{aligned}$$

fun *GNF-state* :: *pitl* \Rightarrow *pitl list*

where

$$\text{GNF-state } (\text{ifalse}) = [\text{itrue}]$$

```

| GNF-state (atom p)      = [(atom p)]
| GNF-state (f iimp g)    = ( ((add-to-now (GNF-state f)  )) @ (GNF-state g))
| GNF-state (next f)      = [itrue]
| GNF-state (f schop g)   = ( (GNF-state f) @ (map (λx. x iand (GNF-empty f)) (GNF-state g)))
| GNF-state (schopstar f) = (GNF-state f)
| GNF-state (omega f)     = (GNF-state f)

```

fun *GNF-next* :: *pitl* \Rightarrow *pitl list*

where

```

  GNF-next (ifalse)      = [ifalse]
| GNF-next (atom p)      = [itrue]
| GNF-next (f iimp g)    = ( (map (λx. (inot x)) (add-to-next (GNF-next f) )) @ (GNF-next g))
| GNF-next (next f)      = [f]
| GNF-next (f schop g)   = ( (map (λx. x schop g) (GNF-next f)) @ (GNF-next g))
| GNF-next (schopstar f) = (map (λx. x schop (schopstar f)) (GNF-next f))
| GNF-next (omega f)     = (map (λx. x schop (omega f)) (GNF-next f))

```

fun *GNF-next-a* :: *pitl* \Rightarrow *pitl list*

where

```

  GNF-next-a (ifalse)    = [ifalse]
| GNF-next-a (atom p)    = [itrue]
| GNF-next-a (f iimp g)  = ( (map (λx. (inot x)) (add-to-next-a (GNF-next-a f) )) @ (GNF-next-a g))
| GNF-next-a (next f)    = [f]
| GNF-next-a (f schop g) = ( (map (λx. x schop g) (GNF-next-a f)) @ (GNF-next-a g))
| GNF-next-a (schopstar f) = (map (λx. x schop (schopstar f)) (GNF-next-a f))
| GNF-next-a (omega f)   = (map (λx. x schop (omega f)) (GNF-next-a f))

```

definition *full-system*

where *full-system* *F* = ((\vdash *big-ior* *F*) \wedge *mutex* *F*)

definition *GNF-cons*

where *GNF-cons* *ae a A* =

((*ae* *iand* *empty*) *ior* (*big-ior* (*map2* (λ *x y*. *x iand* (*next* *y*)) *a A*)))

definition *GNF*

where *GNF* *f* =

(*GNF-cons* (*GNF-empty* *f*) (*GNF-state* *f*) (*GNF-next* *f*))

definition *GNF-prop*

where *GNF-prop* *X* =

(\exists *ae a A*.
 (*state-pitl* *ae*) \wedge (*state-pitl-list* *a*) \wedge
full-system *a* \wedge *length* *a* = *length* *A* \wedge
 \vdash *X* *ieqv* *GNF-cons* *ae a A*)

)

definition *GNF-det*

where *GNF-det* $f =$

(*GNF-cons* (*GNF-empty* f)
 (*foldr* (*add-to-now-single*) (*GNF-state* f) [*itrue*])
 (*foldr* (*add-to-next-single*) (*GNF-next* f) [*ifalse*])
)

3.3.2 Mutex lemmas

lemma *mutex-single-empty*:

shows *mutex-single* f []

unfolding *mutex-single-def* **by** *auto*

lemma *mutex-single-Cons*:

shows *mutex-single* f ($f1 \# F$) \longleftrightarrow

($\vdash \text{inot } (f \text{ iand } f1) \wedge \text{mutex-single } f F$)

unfolding *mutex-single-def* **using** *less-Suc-eq-0-disj* **by** *auto*

lemma *mutex-single-append*:

shows *mutex-single* f ($F1 @ F2$) $\longleftrightarrow \text{mutex-single } f F1 \wedge \text{mutex-single } f F2$

proof (*induct* $F1$ *arbitrary*: $F2$)

case *Nil*

then show ?*case* **unfolding** *mutex-single-def* **by** *simp*

next

case (*Cons* $a F1$)

then show ?*case*

proof –

have 1: *mutex-single* f ($(a \# F1) @ F2$) \longleftrightarrow
 ($\vdash \text{inot } (f \text{ iand } a) \wedge \text{mutex-single } f (F1 @ F2)$)

by (*simp* *add*: *mutex-single-Cons*)

have 2: *mutex-single* f ($F1 @ F2$) $\longleftrightarrow \text{mutex-single } f F1 \wedge \text{mutex-single } f F2$

using *Cons* **by** *blast*

have 3: ($\vdash \text{inot } (f \text{ iand } a) \wedge (\text{mutex-single } f F1 \wedge \text{mutex-single } f F2) \longleftrightarrow$
 (*mutex-single* f ($a \# F1$) $\wedge \text{mutex-single } f F2$)

using *mutex-single-Cons* **by** *blast*

show ?*thesis*

using 1 2 3 **by** *presburger*

qed

qed

lemma *mutex-empty*:

shows *mutex* []

unfolding *mutex-def* **by** *auto*


```

lemma mutex-Cons:
shows mutex (f # F1)  $\longleftrightarrow$  mutex-single f F1  $\wedge$  mutex F1
unfolding mutex-def mutex-single-def
using less-Suc-eq-0-disj
apply auto
apply (metis diff-Suc-1' nth-Cons')
by (metis diff-Suc-1' nth-Cons-Suc)

```

```

lemma mutex-append:
shows mutex (F1 @ F2)  $\longleftrightarrow$ 
  mutex F1  $\wedge$  foldr ( $\lambda x y. (\text{mutex-single } x \text{ } F2) \wedge y$ ) F1 True  $\wedge$  mutex F2
proof (induct F1 arbitrary: F2)
case Nil
then show ?case
by (simp add: mutex-empty)
next
case (Cons a F1)
then show ?case
  proof –
    have 1: mutex ((a # F1) @ F2)  $\longleftrightarrow$ 
      mutex-single a F1  $\wedge$  mutex-single a F2  $\wedge$  mutex (F1@F2)
      by (simp add: mutex-Cons mutex-single-append)
    have 2: foldr ( $\lambda x. (\wedge) (\text{mutex-single } x \text{ } F2)$ ) (a # F1) True  $\longleftrightarrow$ 
      mutex-single a F2  $\wedge$  foldr ( $\lambda x. (\wedge) (\text{mutex-single } x \text{ } F2)$ ) (F1) True
      by simp
    have 3: mutex (F1 @ F2)  $\longleftrightarrow$  (mutex F1  $\wedge$  foldr ( $\lambda x. (\wedge) (\text{mutex-single } x \text{ } F2)$ ) F1 True  $\wedge$  mutex F2)
      using Cons by blast
    have 4: mutex-single a F1  $\wedge$  mutex-single a F2  $\wedge$  mutex (F1@F2)  $\longleftrightarrow$ 
      mutex-single a F1  $\wedge$  mutex-single a F2  $\wedge$ 
      (mutex F1  $\wedge$  foldr ( $\lambda x. (\wedge) (\text{mutex-single } x \text{ } F2)$ ) F1 True  $\wedge$  mutex F2)
      using 1 3 by auto
    have 5: mutex-single a F1  $\wedge$  mutex-single a F2  $\wedge$ 
      (mutex F1  $\wedge$  foldr ( $\lambda x. (\wedge) (\text{mutex-single } x \text{ } F2)$ ) F1 True  $\wedge$  mutex F2)  $\longleftrightarrow$ 
      (mutex (a # F1)  $\wedge$  foldr ( $\lambda x. (\wedge) (\text{mutex-single } x \text{ } F2)$ ) (a # F1) True  $\wedge$  mutex F2)
      using 2 mutex-Cons by blast
    show ?thesis
    using 1 4 5 by presburger
  qed
qed

```

```

lemma unfold-mutex-single:
  foldr ( $\lambda x y. (\text{mutex-single } x \text{ } F2) \wedge y$ ) F1 True  $\longleftrightarrow$ 
  ( $\forall i. 0 \leq i \wedge i < \text{length } F1 \longrightarrow (\text{mutex-single } (F1 ! i) \text{ } F2)$ )
proof (induct F1 arbitrary: F2)
case Nil
then show ?case by simp
next
case (Cons a F1)
then show ?case

```

```

using less-Suc-eq-0-disj by auto
qed

```

3.3.3 Add to lemmas

```

lemma length-add-to-now-single:
shows length (add-to-now-single f L) = (if L=[] then 0 else 2*(length L))
proof (induct L)
case Nil
then show ?case unfolding add-to-now-single-def by simp
next
case (Cons a L)
then show ?case unfolding add-to-now-single-def by simp
qed

```

```

lemma length-add-to-next-single:
shows length (add-to-next-single f L) = (if L=[] then 0 else 2*(length L))
proof (induct L)
case Nil
then show ?case unfolding add-to-next-single-def by simp
next
case (Cons a L)
then show ?case unfolding add-to-next-single-def by simp
qed

```

```

lemma length-add-to-next-single-a:
shows length (add-to-next-single-a f L) = (if L=[] then 0 else 2*(length L))
proof (induct L)
case Nil
then show ?case unfolding add-to-next-single-a-def by simp
next
case (Cons a L)
then show ?case unfolding add-to-next-single-a-def by simp
qed

```

```

lemma length-add-to-now-next-gen:
assumes length xs = length ys
        length L1 = length L2
shows length (foldr add-to-now-single xs L1) =
        length (foldr add-to-next-single ys L2)
using assms
proof (induct xs ys rule: list-induct2)
case Nil
then show ?case by simp
next
case (Cons x xs y ys)
then show ?case
using Cons unfolding add-to-now-single-def add-to-next-single-def

```

by *simp*
qed

lemma *length-add-to-now-next-a-gen*:
assumes *length xs = length ys*
length L1 = length L2
shows *length (foldr add-to-now-single xs L1) =*
length (foldr add-to-next-single-a ys L2)
using *assms*
proof (*induct xs ys rule: list-induct2*)
case *Nil*
then show ?*case* **by** *simp*
next
case (*Cons x xs y ys*)
then show ?*case*
using Cons unfolding add-to-now-single-def add-to-next-single-a-def
by simp
qed

lemma *length-add-to-now-next*:
assumes *length xs = length ys*
shows *length (foldr add-to-now-single xs [itrue,ifalse]) =*
length (foldr add-to-next-single ys [ifalse,ifalse])
using *assms*
by (*simp add: length-add-to-now-next-gen*)

lemma *length-add-to-now-next-a*:
assumes *length xs = length ys*
shows *length (foldr add-to-now-single xs [itrue,ifalse]) =*
length (foldr add-to-next-single-a ys [ifalse,ifalse])
using *assms*
by (*simp add: length-add-to-now-next-a-gen*)

lemma *add-to-now-single-to-big-ior-inv*:
assumes $\vdash \text{big-ior } L$
shows $\vdash \text{big-ior } (\text{add-to-now-single } f \ L)$
unfolding *add-to-now-single-def* **using** *assms*
big-ior-append[of (map ($\lambda x. (f \ i \ \text{and } x)) \ L) (map (\lambda x. (i \ \text{not } f \ i \ \text{and } x)) \ L)]$
by (*auto simp add: big-ior-defs*)
(metis iand-defs nth-map)

lemma *big-ior-foldr-add-to-now-single-inv-a*:
shows $\vdash \text{big-ior } (\text{foldr add-to-now-single } xs \ [\text{itrue}])$
proof (*induct xs*)
case *Nil*
then show ?*case*
using *big-ior-defs* **by** *fastforce*
next

```

case (Cons a xs)
then show ?case
  using Cons add-to-now-single-to-big-ior-inv by auto
qed

```

```

lemma big-ior-foldr-add-to-now-single-inv:
shows  $\vdash$  big-ior (foldr add-to-now-single xs [itrue, ifalse])
proof (induct xs)
case Nil
then show ?case
using big-ior-defs by fastforce
next
case (Cons a xs)
then show ?case
  using Cons add-to-now-single-to-big-ior-inv by auto
qed

```

```

lemma add-to-now-single-to-mutex-inv:
assumes mutex L
shows mutex (add-to-now-single f L)
proof –
have 1: mutex (map ( $\lambda x. f \text{ iand } x$ ) L)
  using assms unfolding mutex-def by auto
have 2: mutex (map ( $\lambda x. \text{inot } f \text{ iand } x$ ) L)
  using assms unfolding mutex-def by auto
have 3: foldr ( $\lambda x. (\wedge) (\text{mutex-single } x (\text{map } (\lambda x. (\text{inot } f) \text{ iand } x) L))$ ) (map ( $\lambda x. f \text{ iand } x$ ) L) True
   $\longleftrightarrow (\forall i. 0 \leq i \wedge i < \text{length } (\text{map } (\lambda x. f \text{ iand } x) L) \longrightarrow$ 
     $\text{mutex-single } (\text{map } (\lambda x. f \text{ iand } x) L ! i) (\text{map } (\lambda x. (\text{inot } f) \text{ iand } x) L))$ 
  using unfold-mutex-single[of (map (lambda. (inot f) iand x) L) (map (lambda. f iand x) L)]
  by blast
have 4:  $(\forall i. 0 \leq i \wedge i < \text{length } (\text{map } (\lambda x. f \text{ iand } x) L) \longrightarrow$ 
   $\text{mutex-single } (\text{map } (\lambda x. f \text{ iand } x) L ! i) (\text{map } (\lambda x. (\text{inot } f) \text{ iand } x) L))$ 
  using assms unfolding mutex-single-def by auto
show ?thesis unfolding add-to-now-single-def using assms
  mutex-append[of (map (lambda. f iand x) L) (map (lambda. (inot f) iand x) L) ]
  using 1 2 3 4 by blast
qed

```

```

lemma mutex-foldr-add-to-now-single-inv-a:
shows mutex (foldr add-to-now-single xs [itrue])
proof (induct xs)
case Nil
then show ?case unfolding mutex-def by simp
next
case (Cons a xs)
then show ?case by (simp add: Cons.hyps add-to-now-single-to-mutex-inv)
qed

```

```

lemma mutex-foldr-add-to-now-single-inv:

```

```

shows mutex (foldr add-to-now-single xs [itrue, ifalse])
proof (induct xs)
case Nil
then show ?case unfolding mutex-def by simp
next
case (Cons a xs)
then show ?case by (simp add: Cons.hyps add-to-now-single-to-mutex-inv)
qed

```

```

lemma add-to-now-single-to-state-pitl-list:
assumes state-pitl f
         state-pitl-list L
shows state-pitl-list ( add-to-now-single f L )
using assms unfolding add-to-now-single-def
by (simp add: inot-d-def state-pitl-list-append state-pitl-list-map)

```

```

lemma state-pitl-foldr-add-to-now-single-inv-a:
assumes state-pitl-list xs
shows state-pitl-list (foldr add-to-now-single xs [itrue])
using assms
proof (induct xs)
case Nil
then show ?case using Nil
by (simp add: inot-d-def itrue-d-def state-pitl-list-def)
next
case (Cons a xs)
then show ?case
  by simp
  (metis add-to-now-single-to-state-pitl-list append-Cons append-Nil length-Cons nth-Cons-0
    state-pitl-list-append state-pitl-list-defs zero-less-Suc)
qed

```

```

lemma state-pitl-foldr-add-to-now-single-inv:
assumes state-pitl-list xs
shows state-pitl-list (foldr add-to-now-single xs [itrue, ifalse])
using assms
proof (induct xs)
case Nil
then show ?case using Nil
by (simp add: inot-d-def itrue-d-def state-pitl-list-def)
next
case (Cons a xs)
then show ?case
  by simp
  (metis add-to-now-single-to-state-pitl-list append-Cons append-Nil length-Cons nth-Cons-0
    state-pitl-list-append state-pitl-list-defs zero-less-Suc)
qed

```

```

lemma det-big-ior-ieqv-nondet-big-ior-a:
assumes length L1 = length L2
shows  $\vdash$  big-ior ( map2 ( $\lambda x y. x \text{ iand } (\text{next } y)$ )
    (foldr (add-to-now-single) L1 [itrue])
    (foldr (add-to-next-single) L2 [ifalse]))
    ieqv
    big-ior (map2 ( $\lambda x y. x \text{ iand } (\text{next } y)$ ) L1 L2)
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by (simp add: big-ior-def)
next
case (Cons x xs y ys)
then show ?case
  proof (cases xs = []  $\wedge$  ys = [] )
    case True
      then show ?thesis using Cons
        unfolding add-to-now-single-def add-to-next-single-def big-ior-def big-iand-def
          by simp
      next
      case False
      then show ?thesis
        proof –
          have 1: (foldr add-to-now-single (x # xs) [itrue]) =
            ( (map ( $\lambda x1. x \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue])) @
              (map ( $\lambda x1. (\text{inot } x) \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue])) )
            by (simp add: add-to-now-single-def)
          have 2: (foldr add-to-next-single (y # ys) [ifalse]) =
            ( (map ( $\lambda x1. y \text{ ior } x1$ ) (foldr add-to-next-single ys [ifalse])) @
              (foldr add-to-next-single ys [ifalse]) )
            by (simp add: add-to-next-single-def)
          have 3: length xs = length ys
            by (simp add: Cons.hyps(1))
          have 4:  $\vdash$  big-ior (map2 ( $\lambda x y. x \text{ iand } \text{next } y$ )
            (foldr add-to-now-single xs [itrue])
            (foldr add-to-next-single ys [ifalse])) ieqv
            big-ior (map2 ( $\lambda x y. x \text{ iand } \text{next } y$ ) xs ys)
            using 3 Cons.hyps(2) False by fastforce
          have 5: big-ior (map2 ( $\lambda x y. x \text{ iand } \text{next } y$ )
            (foldr add-to-now-single (x # xs) [itrue])
            (foldr add-to-next-single (y # ys) [ifalse])) =
            big-ior (map2 ( $\lambda x y. x \text{ iand } \text{next } y$ )
              ( (map ( $\lambda x1. x \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue])) @
                (map ( $\lambda x1. (\text{inot } x) \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue])) )
              ( (map ( $\lambda x1. y \text{ ior } x1$ ) (foldr add-to-next-single ys [ifalse])) @
                (foldr add-to-next-single ys [ifalse]) )
            )
            using 1 2 by presburger
          have 6: length (map ( $\lambda x1. x \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue])) =

```

```

    length (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse]))
  by (simp add: 3 length-add-to-now-next-gen)
have 7: length (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])) =
    length (foldr add-to-next-single ys [ifalse])
  using 6 by auto
have 75: length (foldr add-to-now-single xs [itrue]) =
    length (foldr add-to-next-single ys [ifalse])
  using 7 by force
have 8: ⊢ big-ior (map2 (λx y. x iand next y)
    ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])) )
    ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse])) @
      (foldr add-to-next-single ys [ifalse]) )
    ) ieqv
  big-ior ( (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
    (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse])) )
    @
    (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
    (foldr add-to-next-single ys [ifalse]) )
    )
  )
  using 6 by force
have 9: ⊢ big-ior ( (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
    (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse])) )
    @
    (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
    (foldr add-to-next-single ys [ifalse]) )
    )
  ) ieqv
  (big-ior (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
    (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse])) ) ) ior
  (big-ior (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
    (foldr add-to-next-single ys [ifalse]) ) )
  )
  using big-ior-append by blast
have 10: ⊢ (big-ior (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
    (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse])) ) ) ieqv
  (big-ior (map2 (λx1 y1. (x iand x1) iand next (y ior y1))
    ( (foldr add-to-now-single xs [itrue]))
    ( (foldr add-to-next-single ys [ifalse])) ) )
  )
  using map2-map-a[of (foldr add-to-now-single xs [itrue])
    (foldr add-to-next-single ys [ifalse]) x y ] using 7 by auto
have 11: ⊢ (big-ior (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
    (foldr add-to-next-single ys [ifalse]) ) ) ieqv
  (big-ior (map2 (λx1 y1. ((inot x) iand x1) iand next y1)

```

```

      ( (foldr add-to-now-single xs [itrue]))
      (foldr add-to-next-single ys [ifalse]) ) )
using map2-map-b[of (foldr add-to-now-single xs [itrue])
  (foldr add-to-next-single ys [ifalse]) x] 75
by auto
have 12:  $\vdash$  big-ior (foldr add-to-now-single xs [itrue])
  using big-ior-foldr-add-to-now-single-inv-a by metis
have 13:  $0 < \text{length}$  (foldr add-to-now-single xs [itrue])
  using 12 big-ior-empty by auto
have 14:  $\vdash$  (big-ior (map2 ( $\lambda x1\ y1.$  (x iand x1) iand next (y ior y1))
  ( (foldr add-to-now-single xs [itrue]))
  ( (foldr add-to-next-single ys [ifalse])))) ior
  (big-ior (map2 ( $\lambda x\ y.$  x iand next y)
  (map ( $\lambda x1.$  (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
  (foldr add-to-next-single ys [ifalse]) ) )

  ieqv
  (x iand next y) ior
  (big-ior (map2 ( $\lambda x1\ y1.$  x1 iand next y1)
  ( (foldr add-to-now-single xs [itrue]))
  ( (foldr add-to-next-single ys [ifalse]))))
  using big-ior-split-off[of ( (foldr add-to-now-single xs [itrue]))
  ( (foldr add-to-next-single ys [ifalse])) x y] 12 13
  by (simp add: 75 map2-map-b)
have 15:  $\vdash$  big-ior (map2 ( $\lambda x\ y.$  x iand next y) (x # xs) (y # ys)) ieqv
  (x iand next y) ior
  big-ior (map2 ( $\lambda x\ y.$  x iand next y) xs ys)
  using big-ior-Cons-alt by auto
have 16:  $\vdash$  (x iand next y) ior
  (big-ior (map2 ( $\lambda x1\ y1.$  x1 iand next y1)
  ( (foldr add-to-now-single xs [itrue]))
  ( (foldr add-to-next-single ys [ifalse])))) ieqv
  big-ior (map2 ( $\lambda x\ y.$  x iand next y) (x # xs) (y # ys))
  using 4 15 by auto
show ?thesis
using 10 14 16 5 6 9 by force
qed
qed
qed

```

lemma det-big-ior-ieqv-nondet-big-ior-a-a:

assumes $\text{length } L1 = \text{length } L2$

shows \vdash big-ior (map2 ($\lambda x\ y.$ x iand (next y))
 (foldr (add-to-now-single) L1 [itrue])
 (foldr (add-to-next-single-a) L2 [ifalse]))

ieqv

big-ior (map2 ($\lambda x\ y.$ x iand (next y)) L1 L2)

using assms

proof (induct L1 L2 rule: list-induct2)

case Nil

then show ?case **by** (simp add: big-ior-def)


```

next
case (Cons x xs y ys)
then show ?case
  proof (cases xs = [] ∧ ys = [] )
  case True
  then show ?thesis using Cons
    unfolding add-to-now-single-def add-to-next-single-a-def big-ior-def big-iand-def
    by simp
  next
  case False
  then show ?thesis
    proof -
    have 1: (foldr add-to-now-single (x # xs) [itrue]) =
      ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
        (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])))
      by (simp add: add-to-now-single-def)
    have 2: (foldr add-to-next-single-a (y # ys) [ifalse]) =
      ( (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))@
        (foldr add-to-next-single-a ys [ifalse]) )
      by (simp add: add-to-next-single-a-def)
    have 3: length xs = length ys
      by (simp add: Cons.hyps(1))
    have 4: ⊢ big-ior (map2 (λx y. x iand next y)
      (foldr add-to-now-single xs [itrue])
      (foldr add-to-next-single-a ys [ifalse])) ieqv
      big-ior (map2 (λx y. x iand next y) xs ys)
      using 3 Cons.hyps(2) False by fastforce
    have 5: big-ior (map2 (λx y. x iand next y)
      (foldr add-to-now-single (x # xs) [itrue])
      (foldr add-to-next-single-a (y # ys) [ifalse])) =
      big-ior (map2 (λx y. x iand next y)
      ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
        (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])))
      ( (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))@
        (foldr add-to-next-single-a ys [ifalse]) )
      )
      using 1 2 by presburger
    have 6: length (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) =
      length (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
      by (simp add: 3 length-add-to-now-next-a-gen)
    have 7: length (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])) =
      length (foldr add-to-next-single-a ys [ifalse])
      using 6 by auto
    have 75: length (foldr add-to-now-single xs [itrue]) =
      length (foldr add-to-next-single-a ys [ifalse])
      using 7 by force
    have 8: ⊢ big-ior (map2 (λx y. x iand next y)
      ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
        (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])))
      ( (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))@
        (foldr add-to-next-single-a ys [ifalse]))@

```

```

        (foldr add-to-next-single-a ys [ifalse]) )
    ) ieqv
big-ior ( (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
)
    @
    (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
    (foldr add-to-next-single-a ys [ifalse]) )
    )
    using 6 by force
have 9: ⊢ big-ior ( (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
)
    @
    (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
    (foldr add-to-next-single-a ys [ifalse]) )
    ) ieqv
    (big-ior (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
)
    ior
    (big-ior (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
    (foldr add-to-next-single-a ys [ifalse]) )
    )
    using big-ior-append by blast
have 10: ⊢ (big-ior (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
)
    ieqv
    (big-ior (map2 (λx1 y1. (x iand x1) iand next ((if y = ifalse then y1 else y ior y1)))
    ( (foldr add-to-now-single xs [itrue]))
    ( (foldr add-to-next-single-a ys [ifalse]))))
    using map2-map-a-a[of (foldr add-to-now-single xs [itrue])
    (foldr add-to-next-single-a ys [ifalse]) x y ] using 7 by auto
have 11: ⊢ (big-ior (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
    (foldr add-to-next-single-a ys [ifalse]) ) ) ieqv
    (big-ior (map2 (λx1 y1. ((inot x) iand x1) iand next y1)
    ( (foldr add-to-now-single xs [itrue]))
    (foldr add-to-next-single-a ys [ifalse]) ) )
    using map2-map-b[of (foldr add-to-now-single xs [itrue])
    (foldr add-to-next-single-a ys [ifalse]) x] 75
    by auto
have 12: ⊢ big-ior (foldr add-to-now-single xs [itrue])
    using big-ior-foldr-add-to-now-single-inv-a by metis
have 13: 0 < length (foldr add-to-now-single xs [itrue])

```

```

using 12 big-ior-empty by auto
have 14:  $\vdash$  (big-ior (map2 ( $\lambda x1\ y1.$  (x iand x1) iand next ((if y = ifalse then y1 else y ior y1)))
  ( (foldr add-to-now-single xs [itrue]))
  ( (foldr add-to-next-single-a ys [ifalse])))) ior
  (big-ior (map2 ( $\lambda x\ y.$  x iand next y)
    (map ( $\lambda x1.$  (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
    (foldr add-to-next-single-a ys [ifalse])) ))
  ieqv
  (x iand next y) ior
  (big-ior (map2 ( $\lambda x1\ y1.$  x1 iand next y1)
    ( (foldr add-to-now-single xs [itrue]))
    ( (foldr add-to-next-single-a ys [ifalse]))))
using big-ior-split-off-a[of ( (foldr add-to-now-single xs [itrue]))
  ( (foldr add-to-next-single-a ys [ifalse])) x y] 12 13
by (simp add: 75 map2-map-b)
have 15:  $\vdash$  big-ior (map2 ( $\lambda x\ y.$  x iand next y) (x # xs) (y # ys)) ieqv
  (x iand next y) ior
  big-ior (map2 ( $\lambda x\ y.$  x iand next y) xs ys)
using big-ior-Cons-alt by auto
have 16:  $\vdash$  (x iand next y) ior
  (big-ior (map2 ( $\lambda x1\ y1.$  x1 iand next y1)
    ( (foldr add-to-now-single xs [itrue]))
    ( (foldr add-to-next-single-a ys [ifalse])))) ieqv
  big-ior (map2 ( $\lambda x\ y.$  x iand next y) (x # xs) (y # ys))
using 4 15 by auto
show ?thesis
using 10 14 16 5 6 9 by force
qed
qed
qed

```

lemma *det-big-ior-ieqv-nondet-big-ior*:

assumes *length L1 = length L2*

shows \vdash *big-ior* (*map2* ($\lambda x\ y.$ *x iand (next y)*)
 (*foldr* (*add-to-now-single*) *L1 [itrue,ifalse]*)
 (*foldr* (*add-to-next-single*) *L2 [ifalse,ifalse]*))

ieqv

big-ior (*map2* ($\lambda x\ y.$ *x iand (next y)*) *L1 L2*)

using *assms*

proof (*induct L1 L2 rule: list-induct2*)

case *Nil*

then show ?case **by** (*simp add: big-ior-def*)

next

case (*Cons x xs y ys*)

then show ?case

proof (*cases xs = [] \wedge ys = []*)

case *True*

then show ?thesis **using** *Cons*

unfolding *add-to-now-single-def add-to-next-single-def big-ior-def big-iand-def*

by *simp*

```

next
case False
then show ?thesis
proof -
  have 1: (foldr add-to-now-single (x # xs) [itrue,ifalse]) =
    ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) @
      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])))
    by (simp add: add-to-now-single-def)
  have 2: (foldr add-to-next-single (y # ys) [ifalse,ifalse]) =
    ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse]))@
      (foldr add-to-next-single ys [ifalse,ifalse]) )
    by (simp add: add-to-next-single-def)
  have 3: length xs = length ys
    by (simp add: Cons.hyps(1))
  have 4: ⊢ big-ior (map2 (λx y. x iand next y)
    (foldr add-to-now-single xs [itrue,ifalse])
    (foldr add-to-next-single ys [ifalse,ifalse])) ieqv
    big-ior (map2 (λx y. x iand next y) xs ys)
    using 3 Cons.hyps(2) False by fastforce
  have 5: big-ior (map2 (λx y. x iand next y)
    (foldr add-to-now-single (x # xs) [itrue,ifalse])
    (foldr add-to-next-single (y # ys) [ifalse,ifalse])) =
    big-ior (map2 (λx y. x iand next y)
      ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) @
        (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])))
      ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse]))@
        (foldr add-to-next-single ys [ifalse,ifalse]) )
    )
    using 1 2 by presburger
  have 6: length (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) =
    length (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse]))
    by (simp add: 3 length-add-to-now-next)
  have 7: length (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])) =
    length (foldr add-to-next-single ys [ifalse,ifalse])
    using 6 by auto
  have 75: length (foldr add-to-now-single xs [itrue, ifalse]) =
    length (foldr add-to-next-single ys [ifalse, ifalse])
    using 7 by force
  have 8: ⊢ big-ior (map2 (λx y. x iand next y)
    ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) @
      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])))
    ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse]))@
      (foldr add-to-next-single ys [ifalse,ifalse]) )
    ) ieqv
    big-ior ( (map2 (λx y. x iand next y)
      (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
      (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse])) )
    @
      (map2 (λx y. x iand next y)
        (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
        (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse])) )
    )

```

```

      (foldr add-to-next-single ys [ifalse,ifalse]) )
    )
  using 6 by force
have 9: ⊢ big-ior ( (map2 (λx y. x iand next y)
  (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
  (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse])) )
  @
  (map2 (λx y. x iand next y)
  (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
  (foldr add-to-next-single ys [ifalse,ifalse]) )
  ) ieqv
  (big-ior (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
    (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse])) ) ) ior
  (big-ior (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
    (foldr add-to-next-single ys [ifalse,ifalse]) ) )
  )

  using big-ior-append by blast
have 10: ⊢ (big-ior (map2 (λx y. x iand next y)
  (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
  (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse])) ) ) ieqv
  (big-ior (map2 (λx1 y1. (x iand x1) iand next (y ior y1))
    ( (foldr add-to-now-single xs [itrue,ifalse]))
    ( (foldr add-to-next-single ys [ifalse,ifalse])) ) )
  )
  using map2-map-a[of (foldr add-to-now-single xs [itrue,ifalse])
  (foldr add-to-next-single ys [ifalse,ifalse]) x y ] using 7 by auto
have 11: ⊢ (big-ior (map2 (λx y. x iand next y)
  (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
  (foldr add-to-next-single ys [ifalse,ifalse]) ) ) ieqv
  (big-ior (map2 (λx1 y1. ((inot x) iand x1) iand next y1)
    ( (foldr add-to-now-single xs [itrue,ifalse]))
    (foldr add-to-next-single ys [ifalse,ifalse]) ) )
  )
  using map2-map-b[of (foldr add-to-now-single xs [itrue,ifalse])
  (foldr add-to-next-single ys [ifalse,ifalse]) x ] 75
  by auto
have 12: ⊢ big-ior (foldr add-to-now-single xs [itrue, ifalse])
  using big-ior-foldr-add-to-now-single-inv by metis
have 13: 0 < length (foldr add-to-now-single xs [itrue, ifalse])
  using 12 big-ior-empty by auto
have 14: ⊢ (big-ior (map2 (λx1 y1. (x iand x1) iand next (y ior y1))
  ( (foldr add-to-now-single xs [itrue,ifalse]))
  ( (foldr add-to-next-single ys [ifalse,ifalse])) ) ) ior
  (big-ior (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
    (foldr add-to-next-single ys [ifalse,ifalse]) ) )
  )
  ieqv
  (x iand next y) ior
  (big-ior (map2 (λx1 y1. x1 iand next y1)
    ( (foldr add-to-now-single xs [itrue,ifalse]))
    ( (foldr add-to-next-single ys [ifalse,ifalse])) ) )
  )

```

```

    using big-ior-split-off[of ( (foldr add-to-now-single xs [itrue,ifalse]))
      ( (foldr add-to-next-single ys [ifalse,ifalse])) x y] 12 13
  by (simp add: 75 map2-map-b)
have 15: ⊢ big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys)) ieqv
  (x iand next y) ior
  big-ior (map2 (λx y. x iand next y) xs ys)
  using big-ior-Cons-alt by auto
have 16: ⊢ (x iand next y) ior
  (big-ior (map2 (λx1 y1. x1 iand next y1)
    ( (foldr add-to-now-single xs [itrue,ifalse]))
    ( (foldr add-to-next-single ys [ifalse,ifalse])))) ieqv
  big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))
  using 4 15 by auto
show ?thesis
using 10 14 16 5 6 9 by force
qed
qed
qed

```

lemma *det-big-ior-ieqv-nondet-big-ior-b:*

assumes *length L1 = length L2*

shows ⊢ *big-ior* (*map2* (λ x y. x iand (next y))
 (foldr (add-to-now-single) L1 [itrue,ifalse])
 (foldr (add-to-next-single-a) L2 [ifalse,ifalse]))
 ieqv
big-ior (map2 (λx y. x iand (next y)) L1 L2)

using *assms*

proof (*induct L1 L2 rule: list-induct2*)

case *Nil*

then show ?*case* by (*simp add: big-ior-def*)

next

case (*Cons x xs y ys*)

then show ?*case*

proof (*cases xs = [] ∧ ys = []*)

case *True*

then show ?*thesis* using *Cons*

unfolding *add-to-now-single-def add-to-next-single-a-def big-ior-def big-iand-def*

by *simp*

next

case *False*

then show ?*thesis*

proof –

have 1: (foldr add-to-now-single (x # xs) [itrue,ifalse]) =
 ((map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) @
 (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])))

by (*simp add: add-to-now-single-def*)

have 2: (foldr add-to-next-single-a (y # ys) [ifalse,ifalse]) =
 ((map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse,ifalse]))@
 (foldr add-to-next-single-a ys [ifalse,ifalse]))

by (*simp add: add-to-next-single-a-def*)

```

have 3: length xs = length ys
  by (simp add: Cons.hyps(1))
have 4:  $\vdash$  big-ior (map2 ( $\lambda x y. x \text{ iand } next \ y$ )
  (foldr add-to-now-single xs [itrue,ifalse])
  (foldr add-to-next-single-a ys [ifalse,ifalse])) ieqv
  big-ior (map2 ( $\lambda x y. x \text{ iand } next \ y$ ) xs ys)
  using 3 Cons.hyps(2) False by fastforce
have 5: big-ior (map2 ( $\lambda x y. x \text{ iand } next \ y$ )
  (foldr add-to-now-single (x \# xs) [itrue,ifalse])
  (foldr add-to-next-single-a (y \# ys) [ifalse,ifalse])) =
  big-ior (map2 ( $\lambda x y. x \text{ iand } next \ y$ )
  ( (map ( $\lambda x1. x \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse])) @
  (map ( $\lambda x1. (inot \ x) \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse])))
  ( (map ( $\lambda x1. (if \ y = ifalse \text{ then } x1 \text{ else } y \text{ ior } x1$ )) (foldr add-to-next-single-a ys
  [ifalse,ifalse]))@
  (foldr add-to-next-single-a ys [ifalse,ifalse]) )
  )
  using 1 2 by presburger
have 6: length (map ( $\lambda x1. x \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse])) =
  length (map ( $\lambda x1. y \text{ ior } x1$ ) (foldr add-to-next-single-a ys [ifalse,ifalse]))
  by (simp add: 3 length-add-to-now-next-a)
have 7: length (map ( $\lambda x1. (inot \ x) \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse])) =
  length (foldr add-to-next-single-a ys [ifalse,ifalse])
  using 6 by auto
have 75: length (foldr add-to-now-single xs [itrue, ifalse]) =
  length (foldr add-to-next-single-a ys [ifalse, ifalse])
  using 7 by force
have 8:  $\vdash$  big-ior (map2 ( $\lambda x y. x \text{ iand } next \ y$ )
  ( (map ( $\lambda x1. x \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse])) @
  (map ( $\lambda x1. (inot \ x) \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse])))
  ( (map ( $\lambda x1. (if \ y = ifalse \text{ then } x1 \text{ else } y \text{ ior } x1$ )) (foldr add-to-next-single-a ys
  [ifalse,ifalse]))@
  (foldr add-to-next-single-a ys [ifalse,ifalse]) )
  ) ieqv
  big-ior ( (map2 ( $\lambda x y. x \text{ iand } next \ y$ )
  (map ( $\lambda x1. x \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse]))
  (map ( $\lambda x1. (if \ y = ifalse \text{ then } x1 \text{ else } y \text{ ior } x1$ )) (foldr add-to-next-single-a ys
  [ifalse,ifalse])) )
  )
  @
  (map2 ( $\lambda x y. x \text{ iand } next \ y$ )
  (map ( $\lambda x1. (inot \ x) \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse]))
  (foldr add-to-next-single-a ys [ifalse,ifalse]) )
  )
  using 6 by force
have 9:  $\vdash$  big-ior ( (map2 ( $\lambda x y. x \text{ iand } next \ y$ )
  (map ( $\lambda x1. x \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse]))
  (map ( $\lambda x1. (if \ y = ifalse \text{ then } x1 \text{ else } y \text{ ior } x1$ )) (foldr add-to-next-single-a ys
  [ifalse,ifalse])) )
  )
  @
  (map2 ( $\lambda x y. x \text{ iand } next \ y$ )

```

```

      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
      (foldr add-to-next-single-a ys [ifalse,ifalse]) )
    ) ieqv
  (big-ior (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys
[ifalse,ifalse])) )) ior
    (big-ior (map2 (λx y. x iand next y)
      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
      (foldr add-to-next-single-a ys [ifalse,ifalse]) ))
    using big-ior-append by blast
  have 10: ⊢ (big-ior (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys
[ifalse,ifalse])) )) ieqv
    (big-ior (map2 (λx1 y1. (x iand x1) iand next ((if y = ifalse then y1 else y ior y1)))
      ( (foldr add-to-now-single xs [itrue,ifalse]))
      ( (foldr add-to-next-single-a ys [ifalse,ifalse]))))
    using map2-map-a[of (foldr add-to-now-single xs [itrue,ifalse])
      (foldr add-to-next-single-a ys [ifalse,ifalse]) x y ] using 7 by auto
  have 11: ⊢ (big-ior (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
    (foldr add-to-next-single-a ys [ifalse,ifalse]) )) ieqv
    (big-ior (map2 (λx1 y1. ((inot x) iand x1) iand next y1)
      ( (foldr add-to-now-single xs [itrue,ifalse]))
      (foldr add-to-next-single-a ys [ifalse,ifalse]) ))
    using map2-map-b[of (foldr add-to-now-single xs [itrue,ifalse])
      (foldr add-to-next-single-a ys [ifalse,ifalse]) x ] 75
    by auto
  have 12: ⊢ big-ior (foldr add-to-now-single xs [itrue, ifalse])
    using big-ior-foldr-add-to-now-single-inv by metis
  have 13: 0 < length (foldr add-to-now-single xs [itrue, ifalse])
    using 12 big-ior-empty by auto
  have 14: ⊢ (big-ior (map2 (λx1 y1. (x iand x1) iand next (if y= ifalse then y1 else y ior y1))
    ( (foldr add-to-now-single xs [itrue,ifalse]))
    ( (foldr add-to-next-single-a ys [ifalse,ifalse])))) ior
    (big-ior (map2 (λx y. x iand next y)
      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
      (foldr add-to-next-single-a ys [ifalse,ifalse]) ))
    ieqv
    (x iand next y) ior
    (big-ior (map2 (λx1 y1. x1 iand next y1)
      ( (foldr add-to-now-single xs [itrue,ifalse]))
      ( (foldr add-to-next-single-a ys [ifalse,ifalse]))))
    using big-ior-split-off-a[of ( (foldr add-to-now-single xs [itrue,ifalse]))
      ( (foldr add-to-next-single-a ys [ifalse,ifalse])) x y ] 12 13
    by (simp add: 75 map2-map-b)
  have 15: ⊢ big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys)) ieqv
    (x iand next y) ior
    big-ior (map2 (λx y. x iand next y) xs ys)

```



```

      using big-ior-Cons-alt by auto
    have 16:  $\vdash (x \text{ iand next } y) \text{ ior}$ 
      (big-ior (map2 ( $\lambda x1 \ y1. x1 \text{ iand next } y1$ )
        ( (foldr add-to-now-single xs [itrue,ifalse]))
        ( (foldr add-to-next-single-a ys [ifalse,ifalse])))) ieqv
      big-ior (map2 ( $\lambda x \ y. x \text{ iand next } y$ ) (x # xs) (y # ys))
    using 4 15 by auto
  show ?thesis
  using 10 14 16 5 6 9 by force
qed
qed
qed

```

```

lemma mutex-single-filter-sat:
  assumes mutex-single f L1
  shows mutex-single f ( filter ( $\lambda x. (\exists \sigma. \sigma \models x)$ ) L1)
using assms
proof (induct L1)
case Nil
then show ?case by simp
next
case (Cons a L1)
then show ?case
by (simp add: mutex-single-Cons)
qed

```

```

lemma mutex-filter-sat:
  assumes mutex L1
  shows mutex ( filter ( $\lambda x. (\exists \sigma. \sigma \models x)$ ) L1)
using assms
proof (induct L1)
case Nil
then show ?case unfolding mutex-def by simp
next
case (Cons a L1)
then show ?case
by (simp add: mutex-Cons mutex-single-filter-sat)
qed

```

```

lemma state-pitl-list-filter-sat:
  assumes state-pitl-list L
  shows state-pitl-list ( filter ( $\lambda x. (\exists \sigma. \sigma \models x)$ ) L)
using assms
proof (induct L)
case Nil
then show ?case by simp
next
case (Cons a L)

```

then show *?case*
by (*simp add: state-pitl-list-def*)
qed

lemma *big-ior-itrue*:
 $\vdash \text{big-ior } [\text{itrue}]$
by (*simp add: big-ior-defs*)

lemma *mutex-itrue*:
 $\text{mutex } [\text{itrue}]$
by (*simp add: mutex-Cons mutex-empty mutex-single-empty*)

lemma *big-ior-mutex-start*:
 $(\vdash \text{big-ior } [\text{ifalse}, \text{itrue}]) \wedge \text{mutex } [\text{ifalse}, \text{itrue}]$
unfolding *big-ior-def mutex-def*
using *less-Suc-eq* **by** *force*

3.3.4 GNF lemmas

lemma *GNF-defs*:
assumes $\vdash F \text{ ieqv } ((F\text{-}e \text{ iand } \text{empty}) \text{ ior } (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) \text{ } F\text{-now } F\text{-next})))$
 $\text{state-pitl } F\text{-}e$
 $\text{state-pitl-list } F\text{-now}$
 $\text{length } F\text{-now} = \text{length } F\text{-next}$
shows $(\sigma \models F) \longleftrightarrow$
 $((\text{nlength } \sigma = 0 \wedge ((\text{NNil } (\text{nfirst } \sigma)) \models F\text{-}e)) \vee$
 $(\text{nlength } \sigma > 0 \wedge$
 $(\exists i < \text{length } F\text{-now}. ((\text{NNil } (\text{nfirst } \sigma)) \models F\text{-now } !i) \wedge ((\text{ndropn } 1 \sigma) \models F\text{-next } !i)))$
proof –
have 1: $(\sigma \models F) \longleftrightarrow$
 $(\sigma \models ((F\text{-}e \text{ iand } \text{empty}) \text{ ior } (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) \text{ } F\text{-now } F\text{-next}))))$

using *assms itl-ieq* **by** *blast*
have 2: $(\sigma \models (F\text{-}e \text{ iand } \text{empty})) \longleftrightarrow$
 $(\text{nlength } \sigma = 0 \wedge ((\text{NNil } (\text{nfirst } \sigma)) \models F\text{-}e))$
using *assms state-pitl-defs empty-defs iand-defs* **by** *blast*
have 3: $\text{length } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) \text{ } F\text{-now } F\text{-next}) = \text{length } F\text{-now}$
by (*simp add: assms(4)*)
have 4: $(\sigma \models (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) \text{ } F\text{-now } F\text{-next}))) \longleftrightarrow$
 $(\exists i < \text{length } F\text{-now}. (\sigma \models (\lambda x y. x \text{ iand } (\text{next } y)) (F\text{-now}!i) (F\text{-next}!i)))$
by (*simp add: assms(4) big-ior-map2-defs*)
have 5: $(\exists i < \text{length } F\text{-now}. (\sigma \models (\lambda x y. x \text{ iand } (\text{next } y)) (F\text{-now}!i) (F\text{-next}!i))) \longleftrightarrow$
 $(\text{nlength } \sigma > 0 \wedge$
 $(\exists i < \text{length } F\text{-now}. ((\text{NNil } (\text{nfirst } \sigma)) \models F\text{-now } !i) \wedge ((\text{ndropn } 1 \sigma) \models F\text{-next } !i)))$

```

by simp
  (metis assms(3) co.enat.exhaust-sel enat-le-plus-same(2) not-one-le-zero plus-1-eSuc(2)
    state-pitl-defs state-pitl-list-defs)
show ?thesis
using 1 2 4 5 by auto
qed

```

```

lemma big-ior-iand-next-big-iand-iimp:
  assumes (⊢ big-ior F)
    mutex F
    length F = length G
  shows ⊢ big-ior (map2 (λx y. x iand (next y)) F G) ieqv
    big-iand (map2 (λx y. x iimp (next y)) F G)
using assms unfolding valid-def mutex-def
apply (auto simp add: big-ior-defs big-iand-defs map2-iand-next-defs)
apply (metis antisym-conv3)
by meson

```

```

lemma inot-big-iand-next-iimp-big-iand-iimp-inot:
  assumes (⊢ big-ior F)
    mutex F
    length F = length G
  shows ⊢ more iimp (inot big-iand (map2 (λx y. x iimp (next y)) F G) ieqv
    big-iand (map2 (λ x y. x iimp (next (inot y))) F G))
using assms unfolding mutex-def
apply (auto simp add: big-iand-defs big-ior-defs more-d-def)
apply (metis linorder-less-linear)
by meson

```

```

lemma inot-big-ior-iand-big-ior-iand-inot:
  assumes (⊢ big-ior F)
    mutex F
    length F = length G
  shows ⊢ (inot (big-ior (map2 (λx y. x iand ( y)) F G)) ieqv
    big-ior (map2 (λ x y. x iand ( (inot y))) F G))
using assms unfolding mutex-def
apply (auto simp add: big-ior-defs big-iand-defs map2-iand-inot-defs)
apply meson
by (metis not-less-iff-gr-or-eq)

```

```

lemma inot-big-ior-iand-big-ior-iand-next-inot:
  assumes (⊢ big-ior F)
    mutex F
    length F = length G
  shows ⊢ more iimp (inot (big-ior (map2 (λx y. x iand (next y)) F G)) ieqv

```

```

      big-ior (map2 (λ x y. x iand (next (inot y))) F G))
using assms unfolding mutex-def
apply (auto simp add: big-ior-defs big-iand-defs map2-iand-inot-next-defs more-d-def)
apply (metis One-nat-def gr-zeroI map2-iand-inot-next-defs not-one-le-zero semantics-pitl.simps(4))
by (metis linorder-less-linear)

```

lemma *more-iimp-inot-big-ior-big-ior-map-inot*:

```

assumes (⊢ big-ior F)
          mutex F
          length F = length G
shows ⊢ more iimp
      ((inot (big-ior (map2 (λ x y. x iand (next y)) F G))) ieqv
       big-ior (map2 (λ x y. x iand (next y)) F (map (λ x. (inot x)) G)))
proof –
have 1: (map2 (λ x y. x iand (next y)) F (map (λ x. (inot x)) G)) =
      (map2 (λ x y. x iand (next (inot y))) F G)
using map2-map[of F G (λ x y. x iand (next y)) id (λ x. (inot x))]
      assms by auto
show ?thesis
using 1 assms inot-big-ior-iand-big-ior-iand-next-inot by auto
qed

```

lemma *GNF-iimp-step*:

```

assumes ⊢ F ieqv (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y)) F-now F-next)))
          state-pitl F-e
          state-pitl-list F-now
          length F-now = length F-next
          ⊢ G ieqv (( G-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y)) G-now G-next)))
          state-pitl G-e
          state-pitl-list G-now
          length G-now = length G-next
shows (σ ⊨ F iimp G) ⟷
      (σ ⊨ ( ( (inot F-e) ior G-e) iand empty) ior
        (big-ior
          (map2 (λ x y. x iand (next y))
            ((add-to-now F-now ) @ G-now)
            ((map (λ x. (inot x)) (add-to-next F-next )) @ G-next) )))
proof (cases nlength σ = 0)
case True
then show ?thesis
proof –
have 1: (σ ⊨ F iimp G) ⟷ (¬(σ ⊨ F) ∨ (σ ⊨ G))
      by auto
have 2: (¬(σ ⊨ F) ∨ (σ ⊨ G)) ⟷
      (¬((NNil (nfirst σ)) ⊨ F-e) ∨ ((NNil (nfirst σ)) ⊨ G-e))

```

```

using
  assms True
  by (metis (mono-tags, lifting) GNF-defs less-numeral-extra(3))
have 3: state-pitl (inot F-e ior G-e)
  by (simp add: assms(2) assms(6) inot-d-def ior-d-def)
have 35: state-pitl-list [ifalse, ittrue]
  unfolding state-pitl-list-def ittrue-d-def inot-d-def by auto
have 4: state-pitl-list ((add-to-now F-now ) @ G-now)
  by (simp add: add-to-now-def assms(3) assms(7) state-pitl-foldr-add-to-now-single-inv-a
    state-pitl-list-append)
have 5: length (add-to-now F-now @ G-now) =
  length (map inot-d (add-to-next F-next ) @ G-next)
  by (simp add: add-to-next-def add-to-now-def assms(4) assms(8) length-add-to-now-next-gen)
have 6:  $(\neg((NNil (nfirst \sigma)) \models F-e) \vee ((NNil (nfirst \sigma)) \models G-e)) \longleftrightarrow$ 
   $(\sigma \models ( ( (inot F-e) ior G-e) iand empty) ior$ 
     $(big-ior$ 
       $(map2 (\lambda x y. x iand (next y))$ 
         $((add-to-now F-now )@G-now)$ 
         $((map (\lambda x. (inot x)) (add-to-next F-next )) @ G-next) ) ) )$ 
using GNF-defs[of
   $( ( (inot F-e) ior G-e) iand empty) ior$ 
   $(big-ior$ 
     $(map2 (\lambda x y. x iand (next y))$ 
       $((add-to-now F-now )@G-now)$ 
       $((map (\lambda x. (inot x)) (add-to-next F-next )) @ G-next) ) )$ 
     $( (inot F-e) ior G-e)$ 
     $((add-to-now F-now )@G-now)$ 
     $((map (\lambda x. (inot x)) (add-to-next F-next )) @ G-next) \sigma ]$ 
using assms
using 3 4 5 True by force
show ?thesis
using 1 2 6 by blast
qed
next
case False
then show ?thesis
proof –
have 8:  $(\sigma \models F \text{ iimp } G) \longleftrightarrow (\sigma \models (inot F) ior G)$ 
  by auto
have 9:  $(\sigma \models (inot F)) \longleftrightarrow$ 
   $(\sigma \models (inot (big-ior (map2 (\lambda x y. x iand (next y)) F-now F-next))))$ 
  using False assms(1) by auto
have 10:  $\vdash (big-ior (map2 (\lambda x y. x iand (next y)) F-now F-next)) \text{ ieqv}$ 
   $big-ior ( map2 (\lambda x y. x iand (next y))$ 
     $(foldr (add-to-now-single) F-now [ittrue])$ 
     $(foldr (add-to-next-single) F-next [ifalse]))$ 
  using assms(4) det-big-ior-ieqv-nondet-big-ior-a by auto
have 101:  $(\sigma \models (inot (big-ior (map2 (\lambda x y. x iand (next y)) F-now F-next)))) \longleftrightarrow$ 
   $(\sigma \models (inot (big-ior ( map2 (\lambda x y. x iand (next y))$ 
     $(foldr (add-to-now-single) F-now [ittrue])$ 

```

```

      (foldr (add-to-next-single) F-next [ifalse])))
  using 10 by auto
have 11: ⊢ big-ior (foldr (add-to-now-single) F-now [itrue])
  using big-ior-foldr-add-to-now-single-inv-a by blast
have 12: mutex (foldr (add-to-now-single) F-now [itrue])
  using mutex-foldr-add-to-now-single-inv-a by blast
have 13: length (foldr (add-to-now-single) F-now [itrue]) =
  length (foldr (add-to-next-single) F-next [ifalse])
  by (simp add: assms(4) length-add-to-now-next-gen)
have 14: ⊢ more iimp
  ( (inot (big-ior ( map2 (λ x y. x iand (next y))
    (foldr (add-to-now-single) F-now [itrue])
    (foldr (add-to-next-single) F-next [ifalse]))) iequiv
    (big-ior
      (map2 (λ x y. x iand (next y))
        (foldr (add-to-now-single) F-now [itrue])
        ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse])))))
    using 11 12 13 more-iimp-inot-big-ior-big-ior-map-inot by blast
have 15: (σ ⊢ (inot (big-ior ( map2 (λ x y. x iand (next y))
  (foldr (add-to-now-single) F-now [itrue])
  (foldr (add-to-next-single) F-next [ifalse]))))) ↔
  (σ ⊢ (big-ior
    (map2 (λ x y. x iand (next y))
      (foldr (add-to-now-single) F-now [itrue])
      ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse])))))
    using 14 False by auto
    (metis (no-types, lifting) empty-d-def empty-defs inot-defs,
      metis (no-types, lifting) empty-d-def empty-defs inot-defs)
have 16: (σ ⊢ G) ↔
  (σ ⊢ (big-ior (map2 (λ x y. x iand (next y)) G-now G-next)))
  using False assms(5) by auto
have 17: (σ ⊢ (inot F) ior G) ↔
  (σ ⊢ (big-ior (map2 (λ x y. x iand (next y))
    (foldr (add-to-now-single) F-now [itrue])
    ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse]))))) ior
    (big-ior (map2 (λ x y. x iand (next y)) G-now G-next)))
    using 101 15 16 9 by auto
have 18: (σ ⊢ (big-ior (map2 (λ x y. x iand (next y))
  (foldr (add-to-now-single) F-now [itrue])
  ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse]))))) ior
    (big-ior (map2 (λ x y. x iand (next y)) G-now G-next))) ↔
  (σ ⊢ (big-ior (map2 (λ x y. x iand (next y))
    ((foldr (add-to-now-single) F-now [itrue]) @ G-now)
    ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse])) @ G-next) ) ) )
  using 13 big-ior-append by auto
show ?thesis unfolding add-to-now-def add-to-next-def
  using 17 18 False by auto
qed
qed

```

lemma *GNF-inot-step*:

assumes $\vdash F \text{ ieqv } ((F\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next y)) F\text{-}now F\text{-}next)))$
 $state\text{-}pitl F\text{-}e$
 $state\text{-}pitl\text{-}list F\text{-}now$
 $length F\text{-}now = length F\text{-}next$
shows $(\sigma \models (inot F)) \longleftrightarrow$
 $(\sigma \models ((inot F\text{-}e) \text{ iand empty}) \text{ ior } (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next y)) ((add\text{-}to\text{-}now F\text{-}now) @ [ifalse]) ((map (\lambda x. (inot x)) (add\text{-}to\text{-}next F\text{-}next)) @ [ifalse])))))$

proof –

have 1: $(\sigma \models (inot F)) \longleftrightarrow (\sigma \models F \text{ iimp ifalse})$
by *auto*
have 2: $\vdash ifalse \text{ ieqv } ifalse \text{ iand empty ior big-ior } (map2 (\lambda x y. x \text{ iand } next y) [ifalse] [ifalse])$
by *(auto simp add: big-ior-defs)*
have 3: $(\sigma \models F \text{ iimp ifalse}) \longleftrightarrow$
 $(\sigma \models ((inot F\text{-}e) \text{ ior ifalse}) \text{ iand empty}) \text{ ior } (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next y)) ((add\text{-}to\text{-}now F\text{-}now) @ [ifalse]) ((map (\lambda x. (inot x)) (add\text{-}to\text{-}next F\text{-}next)) @ [ifalse]))))$
using *GNF-iimp-step[of F F-e F-now F-next ifalse ifalse [ifalse] [ifalse] σ]*
assms
by *(metis 2 length-Cons less-Suc0 list.size(3) nth-Cons-0 state-pitl.simps(1) state-pitl-list-defs)*
show *?thesis*
using 3 **by** *auto*
qed

lemma *GNF-inot-step-valid*:

assumes $\vdash F \text{ ieqv } ((F\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next y)) F\text{-}now F\text{-}next)))$
 $state\text{-}pitl F\text{-}e$
 $state\text{-}pitl\text{-}list F\text{-}now$
 $length F\text{-}now = length F\text{-}next$
shows $\vdash (inot F) \text{ ieqv } ((inot F\text{-}e) \text{ iand empty}) \text{ ior } (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next y)) ((add\text{-}to\text{-}now F\text{-}now) @ [ifalse]) ((map (\lambda x. (inot x)) (add\text{-}to\text{-}next F\text{-}next)) @ [ifalse]))))$
using *assms GNF-inot-step[of F F-e F-now F-next]*
using *itl-ieq by blast*

lemma *GNF-inot-inot-step-valid*:

assumes $\vdash F \text{ ieqv } ((F\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior \text{ (map2 } (\lambda x y. x \text{ iand (next } y)) \text{ F-now F-next})))$
 $state\text{-}pitl \text{ F-}e$
 $state\text{-}pitl\text{-}list \text{ F-now}$
 $length \text{ F-now} = length \text{ F-next}$
shows $\vdash (inot (inot F)) \text{ ieqv}$
 $((F\text{-}e \text{ iand empty}) \text{ ior}$
 $(big\text{-}ior \text{ (map2 } (\lambda x y. x \text{ iand (next } y)) \text{ F-now F-next})))$
using *assms* **by** *simp*

lemma *GNF-ior-step*:

assumes $\vdash F \text{ ieqv } ((F\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior \text{ (map2 } (\lambda x y. x \text{ iand (next } y)) \text{ F-now F-next})))$
 $state\text{-}pitl \text{ F-}e$
 $state\text{-}pitl\text{-}list \text{ F-now}$
 $length \text{ F-now} = length \text{ F-next}$
 $\vdash G \text{ ieqv } ((G\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior \text{ (map2 } (\lambda x y. x \text{ iand (next } y)) \text{ G-now G-next})))$
 $state\text{-}pitl \text{ G-}e$
 $state\text{-}pitl\text{-}list \text{ G-now}$
 $length \text{ G-now} = length \text{ G-next}$
shows $(\sigma \models F \text{ ior } G) \longleftrightarrow$
 $(\sigma \models ((F\text{-}e \text{ ior } G\text{-}e) \text{ iand empty}) \text{ ior}$
 $(big\text{-}ior \text{ (map2 } (\lambda x y. x \text{ iand (next } y)) \text{ (F-now @ G-now) (F-next @ G-next)})))$
proof –
have 1: $(\sigma \models F \text{ ior } G) \longleftrightarrow$
 $(\sigma \models ((F\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior \text{ (map2 } (\lambda x y. x \text{ iand (next } y)) \text{ F-now F-next}))) \text{ ior}$
 $((G\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior \text{ (map2 } (\lambda x y. x \text{ iand (next } y)) \text{ G-now G-next}))))$
using *assms* **by** *auto*
have 2: $(\sigma \models ((F\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior \text{ (map2 } (\lambda x y. x \text{ iand (next } y)) \text{ F-now F-next}))) \text{ ior}$
 $((G\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior \text{ (map2 } (\lambda x y. x \text{ iand (next } y)) \text{ G-now G-next})))) \longleftrightarrow$
 $(\sigma \models ((F\text{-}e \text{ ior } G\text{-}e) \text{ iand empty}) \text{ ior}$
 $(big\text{-}ior \text{ (map2 } (\lambda x y. x \text{ iand (next } y)) \text{ (F-now @ G-now) (F-next @ G-next)})))$
using *assms* *big-ior-map2-append* **by** *fastforce*
show *?thesis*
using 1 2 **by** *blast*
qed

lemma *GNF-schop-step*:

assumes $\vdash F \text{ ieqv } ((F\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior \text{ (map2 } (\lambda x y. x \text{ iand (next } y)) \text{ F-now F-next})))$
 $state\text{-}pitl \text{ F-}e$
 $state\text{-}pitl\text{-}list \text{ F-now}$
 $length \text{ F-now} = length \text{ F-next}$
 $\vdash G \text{ ieqv } ((G\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior \text{ (map2 } (\lambda x y. x \text{ iand (next } y)) \text{ G-now G-next})))$
 $state\text{-}pitl \text{ G-}e$
 $state\text{-}pitl\text{-}list \text{ G-now}$
 $length \text{ G-now} = length \text{ G-next}$
shows $(\sigma \models F \text{ schop } G) \longleftrightarrow$
 $(\sigma \models (F\text{-}e \text{ iand } G\text{-}e) \text{ iand empty ior}$
 $(big\text{-}ior \text{ (map2 } (\lambda x y. x \text{ iand next } y))$


```

      (F-now @ (map (λx. x iand F-e) G-now))
      ((map (λx. x schop G) F-next) @ G-next))) )
proof (cases nlength σ = 0)
case True
then show ?thesis
proof -
  have 1: (σ ⊨ F schop G) ⟷ (σ ⊨ F) ∧ (σ ⊨ G)
    using True by (auto simp add: ntaken-all zero-enat-def)
  have 2: (σ ⊨ F) ∧ (σ ⊨ G) ⟷ ((NNil (nfirst σ)) ⊨ F-e) ∧ ((NNil (nfirst σ)) ⊨ G-e)
    using GNF-defs True assms
    by (metis less-numeral-extra(3))
  have 3: state-pitl (F-e iand G-e)
    by (simp add: assms(2) assms(6) iand-d-def inot-d-def ior-d-def)
  have 4: state-pitl-list (F-now @ map (λx. x iand F-e) G-now)
    using assms state-pitl-list-append state-pitl-list-map-alt by blast
  have 5: length (F-now @ map (λx. x iand F-e) G-now) =
    length (map (λx. x schop G) F-next @ G-next)
    by (simp add: assms(4) assms(8))
  have 6: ((NNil (nfirst σ)) ⊨ F-e) ∧ ((NNil (nfirst σ)) ⊨ G-e) ⟷
    (σ ⊨ (F-e iand G-e) iand empty ior
      (big-ior (map2 (λ x y. x iand next y)
        (F-now @ (map (λx. x iand F-e) G-now))
        ((map (λx. x schop G) F-next) @ G-next)))) )
    using GNF-defs[of (F-e iand G-e) iand empty ior
      (big-ior (map2 (λ x y. x iand next y)
        (F-now @ (map (λx. x iand F-e) G-now))
        ((map (λx. x schop G) F-next) @ G-next)))
      (F-e iand G-e) (F-now @ (map (λx. x iand F-e) G-now))
      ((map (λx. x schop G) F-next) @ G-next) σ]
    using assms
    by (simp add: 3 4 True)
  show ?thesis
  using 1 2 6 by presburger
qed
next
case False
assume a: nlength σ ≠ 0
show ?thesis (is ?lhs ⟷ ?rhs)
proof
  assume l: ?lhs
  show ?rhs
  proof -
    have 7: (∃ i. enat i ≤ nlength σ ∧ (ntaken i σ ⊨ F) ∧ (ndropn i σ ⊨ G))
      using l by simp
    obtain i where 8: enat i ≤ nlength σ ∧ (ntaken i σ ⊨ F) ∧ (ndropn i σ ⊨ G)
      using 7 by blast
    have 9: i=0 ⟹ ?rhs
      proof -
        assume b: i=0
        have 10: (ntaken 0 σ ⊨ F) ∧ (σ ⊨ G)

```

```

    using 8 b by auto
  have 11: (  $\sigma \models F-e$  )
    using assms using 8 by simp
    (metis (mono-tags, lifting) GNF-defs assms(1) b less-numeral-extra(3) min-enat-simps(3)
      ntaken-nfirst ntaken-nlength state-pitl-defs zero-enat-def)
  have 12:  $\sigma \models F-e$  iand big-ior (map2 ( $\lambda x y. x$  iand next y) G-now G-next)
    using 10 11 a assms(5) by force
  have 13:  $\sigma \models$  big-ior (map2 ( $\lambda x y. x$  iand next y)
    (map ( $\lambda x. x$  iand F-e) G-now) G-next)
    using iand-big-ior-dist 12 assms(8) itl-ieq by blast
  show ?thesis using assms(4) assms(8) big-ior-map2-append[of F-now map ( $\lambda x. x$  schop G) F-next
    map ( $\lambda x. x$  iand F-e) G-now G-next] by (simp add: 13)
qed
have 14:  $0 < i \wedge \text{enat } i \leq \text{nlength } \sigma \implies ?rhs$ 
proof -
  assume c:  $0 < i \wedge \text{enat } i \leq \text{nlength } \sigma$ 
  have 15: ( $\text{ntaken } i \sigma \models F$ )  $\wedge$  ( $\text{ndropn } i \sigma \models G$ )
    by (simp add: 8)
  have 16: ( $\text{ntaken } i \sigma \models$  (big-ior (map2 ( $\lambda x y. x$  iand next y) F-now F-next)))
    using 8 assms(1) c enat-0-iff(1) by auto
  have 17: (( $\text{ntaken } i \sigma \models$  (big-ior (map2 ( $\lambda x y. x$  iand next y) F-now F-next)))  $\longleftrightarrow$ 
    ( $\exists ia < \text{length } F\text{-now}. \text{ntaken } i \sigma \models F\text{-now} ! ia$  iand next F-next ! ia))
    using 16 big-ior-map2-defs[of F-now F-next ( $\text{ntaken } i \sigma$ ) ( $\lambda x y. x$  iand next y) ]
    assms by blast
  have 18:  $\text{length } F\text{-now} = \text{length } (\text{map } (\lambda x. x \text{ schop } G) F\text{-next})$ 
    using assms(4) by fastforce
  have 19: ( $\sigma \models$  (big-ior (map2 ( $\lambda x y. x$  iand next y)
    F-now (map ( $\lambda x. x$  schop G) F-next) )))  $\longleftrightarrow$ 
    ( $\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now} ! i$  iand next map ( $\lambda x. x$  schop G) F-next ! i)
    using 18 big-ior-map2-defs[of F-now (map ( $\lambda x. x$  schop G) F-next)  $\sigma$  ( $\lambda x y. x$  iand next y)]
    by blast
  obtain j where 20:  $j < \text{length } F\text{-now} \wedge (\text{ntaken } i \sigma \models F\text{-now} ! j$  iand next F-next ! j)
    using 16 17 by blast
  have 21:  $\text{enat } (i-1) \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0)$ 
    by (metis One-nat-def c enat-minus-mono1 idiff-enat-enat)
  have 22: ( $\text{ntaken } (i-1) (\text{ndropn } (\text{Suc } 0) \sigma) \models F\text{-next} ! j$ )
    by (metis 20 One-nat-def Suc-leI c diff-add iand-defs ntaken-ndropn-swap plus-enat-simps(1)
      semantics-pitl.simps(4))
  have 23: ( $\text{ndropn } (i-1) (\text{ndropn } (\text{Suc } 0) \sigma) \models G$ )
    using 8 by (simp add: c ndropn-ndropn)
  have 24: ( $\sigma \models F\text{-now} ! j$  iand next ((F-next ! j) schop G) )
    using 20 15 by simp
    (metis 21 22 23 assms(3) ntaken-nfirst state-pitl-defs state-pitl-list-defs)
  have 25: ( $\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now} ! i$  iand next map ( $\lambda x. x$  schop G) F-next ! i)
    using 20 24 assms(4) by fastforce
  show ?thesis using 18 19 25 assms(8) big-ior-map2-append[of F-now map ( $\lambda x. x$  schop G) F-next
    map ( $\lambda x. x$  iand F-e) G-now G-next] by simp
qed
show ?thesis
using 14 8 9 by blast

```

```

qed
next
assume r: ?rhs
show ?lhs
proof -
  have 26:  $\sigma \models \text{big-ior} (\text{map2 } (\lambda x y. x \text{ iand next } y) (F\text{-now} @ \text{map } (\lambda x. x \text{ iand } F\text{-e}) G\text{-now}) (\text{map } (\lambda x. x \text{ schop } G) F\text{-next} @ G\text{-next}))$ 
    using a r by auto
  have 27:  $\text{length } F\text{-now} = \text{length } (\text{map } (\lambda x. x \text{ schop } G) F\text{-next})$ 
    by (simp add: assms(4))
  have 28:  $\text{length } (\text{map } (\lambda x. x \text{ iand } F\text{-e}) G\text{-now}) = \text{length } (G\text{-next})$ 
    using assms(8) by fastforce
  have 29:  $(\sigma \models (\text{big-ior} (\text{map2 } (\lambda x y. x \text{ iand next } y) (F\text{-now}) (\text{map } (\lambda x. x \text{ schop } G) F\text{-next}))))$ 
     $\vee$ 
     $(\sigma \models ((\text{big-ior} (\text{map2 } (\lambda x y. x \text{ iand next } y) (\text{map } (\lambda x. x \text{ iand } F\text{-e}) G\text{-now}) (G\text{-next}) ))))$ 
    using 26 27 28 big-ior-map2-append itl-ieq ior-defs by blast
  have 30:  $(\sigma \models (\text{big-ior} (\text{map2 } (\lambda x y. x \text{ iand next } y) (F\text{-now}) (\text{map } (\lambda x. x \text{ schop } G) F\text{-next})))) \longleftrightarrow$ 
     $(\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now} ! i \text{ iand next map } (\lambda x. x \text{ schop } G) F\text{-next} ! i)$ 
    using big-ior-map2-defs[of F-now (map (λx. x schop G) F-next) σ (λx y. x iand next y)]
    using 27 by fastforce
  have 31:  $(\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now} ! i \text{ iand next map } (\lambda x. x \text{ schop } G) F\text{-next} ! i) \implies ?lhs$ 
    proof -
      assume d:  $(\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now} ! i \text{ iand next map } (\lambda x. x \text{ schop } G) F\text{-next} ! i)$ 
      obtain i where 32:  $i < \text{length } F\text{-now} \wedge (\sigma \models F\text{-now} ! i \text{ iand next map } (\lambda x. x \text{ schop } G) F\text{-next} ! i)$ 
        using d by blast
      have 33:  $(\sigma \models F\text{-now} ! i)$ 
        using 32 iand-defs by blast
      have 34:  $\text{map } (\lambda x. x \text{ schop } G) F\text{-next} ! i = (F\text{-next} ! i) \text{ schop } G$ 
        using 32 assms(4) by auto
      have 35:  $1 \leq \text{nlength } \sigma \wedge (\exists ia. \text{enat } ia \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge (\text{ntaken } ia (\text{ndropn } (\text{Suc } 0)) \sigma) \models F\text{-next} ! i \wedge (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0)) \sigma) \models G)$ 
        using 32 34 by simp
      obtain ia where 36:  $\text{enat } ia \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge (\text{ntaken } ia (\text{ndropn } (\text{Suc } 0)) \sigma) \models F\text{-next} ! i \wedge (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0)) \sigma) \models G$ 
        using 35 by blast
      have 37:  $(\text{ntaken } (ia+1) (\sigma) \models (\text{big-ior} (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) F\text{-now } F\text{-next}))) \longleftrightarrow$ 
         $(\exists i < \text{length } F\text{-now}. \text{ntaken } (ia+1) (\sigma) \models F\text{-now} ! i \text{ iand next } F\text{-next} ! i)$ 
        using assms big-ior-map2-defs[of F-now F-next (ntaken (ia+1) (σ)) (λ x y. x iand (next y))] by
      blast
      have 38:  $(\text{ntaken } (ia+1) \sigma \models F\text{-now} ! i \text{ iand next } F\text{-next} ! i)$ 
        by simp
      (metis 32 33 35 36 One-nat-def add commute assms(3) enat-min-eq enat-ord-simps(1)
        le-add1 ntaken-ndropn-swap ntaken-nfirst one-enat-def plus-1-eq-Suc state-pitl-defs
        state-pitl-list-defs)
      have 39:  $(\text{ntaken } (ia+1) \sigma \models F)$ 
        using 32 37 38 assms(1) by auto
      have 40:  $(\text{ndropn } (ia+1) \sigma) \models G$ 
        by (metis 36 One-nat-def add commute ndropn-ndropn)
      have 41:  $(ia+1) \leq \text{nlength } \sigma$ 
        by (metis 35 36 One-nat-def enat-min-eq one-enat-def plus-enat-simps(1))

```

```

    show ?thesis
    using 39 40 41 by auto
  qed
  have 42: ( $\sigma \models ((big\text{-}ior (map2 (\lambda x y. x \text{ iand } next y) (map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now) (G\text{-}next) ))))$ )
 $\longleftrightarrow$ 
    ( $\exists i < length (map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now). \sigma \models map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now ! i \text{ iand } next$ 
 $G\text{-}next ! i$ )
    using big-ior-map2-defs[of (map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now) G\text{-}next  $\sigma (\lambda x y. x \text{ iand } next y)$ ]
    using 28 by fastforce
  have 43: ( $\exists i < length (map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now). \sigma \models map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now ! i \text{ iand } next$ 
 $G\text{-}next ! i \implies ?lhs$ )
    proof -
      assume e: ( $\exists i < length (map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now). \sigma \models map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now ! i \text{ iand } next$ 
 $G\text{-}next ! i$ )
      obtain i where 44:  $i < length (map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now) \wedge (\sigma \models map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now ! i \text{ iand } next$ 
 $G\text{-}next ! i)$ 
      using e by blast
      have 45: ( $ntaken\ 0\ \sigma \models (G\text{-}now ! i) \text{ iand } F\text{-}e$ )
      using assms
      by (metis 28 44 iand-defs ntaken-0 nth-map state-pitl-defs state-pitl-list-defs)
      have 46: ( $ntaken\ 0\ \sigma \models F$ )
      using 45 assms(1) by fastforce
      have 47: ( $ndropn\ 0\ \sigma \models G\text{-}now ! i \text{ iand } next\ G\text{-}next ! i$ )
      using 44 by fastforce
      have 48: ( $(ndropn\ 0\ \sigma) \models (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next y)) G\text{-}now\ G\text{-}next))) \longleftrightarrow$ 
 $(\exists i < length\ G\text{-}now. ndropn\ 0\ \sigma \models G\text{-}now ! i \text{ iand } next\ G\text{-}next ! i)$ 
      using big-ior-map2-defs[of  $G\text{-}now\ G\text{-}next (ndropn\ 0\ \sigma) (\lambda x y. x \text{ iand } (next y))$ ]
      using assms(8) by fastforce
      have 49: ( $ndropn\ 0\ \sigma \models G$ )
      using 44 47 48 assms(5) by auto
      show ?thesis
      by (metis 46 49 semantics-pitl.simps(5) zero-enat-def zero-le)
    qed
  show ?thesis
  using 29 30 31 42 43 by fastforce
  qed
  qed
  qed

```

lemma *GNF-schopstar-step*:

```

  assumes  $\vdash F \text{ ieqv } ((F\text{-}e \text{ iand } empty) \text{ ior } (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next y)) F\text{-}now\ F\text{-}next)))$ 
    state-pitl  $F\text{-}e$ 
    state-pitl-list  $F\text{-}now$ 
    length  $F\text{-}now = length\ F\text{-}next$ 
  shows  $(\sigma \models chopstar\ F) \longleftrightarrow$ 
    ( $\sigma \models (itrue \text{ iand } empty) \text{ ior}$ 
      ( $big\text{-}ior (map2 (\lambda x y. x \text{ iand } next y) F\text{-}now (map (\lambda x. x \text{ chop } (chopstar\ F)) F\text{-}next)))$ 
      ( $is\ ?lhs \longleftrightarrow ?rhs$ ))
  proof -

```

have 1: $(\sigma \models \text{schopstar } F) \longleftrightarrow$
 $(\exists l. \text{nnth } l \ 0 = 0 \wedge \text{nfinite } l \wedge \text{nidx } l \wedge \text{enat } (\text{nlast } l) = \text{nlength } \sigma \wedge \text{nfinite } \sigma \wedge$
 $(\forall i. \text{enat } i < \text{nlength } l \longrightarrow (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l (\text{Suc } i)) \models F)))$
by *simp*
have 2: $?lhs \implies ?rhs$
proof –
assume $l: ?lhs$
obtain l **where** $\beta: \text{nnth } l \ 0 = 0 \wedge \text{nfinite } l \wedge \text{nidx } l \wedge \text{enat } (\text{nlast } l) = \text{nlength } \sigma \wedge \text{nfinite } \sigma \wedge$
 $(\forall i. \text{enat } i < \text{nlength } l \longrightarrow (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l (\text{Suc } i)) \models F))$
using $l \ 1$ **by** *blast*
have 4: $\text{nlength } l = 0 \implies ?rhs$
proof –
assume $n: \text{nlength } l = 0$
have 5: $\text{nlength } \sigma = 0$
by $(\text{metis } 3 \ n \ \text{nnth-nlast the-enat-0 zero-enat-def})$
show $?thesis$
by $(\text{simp add: } 5)$
qed
have 6: $0 < \text{nlength } l \implies ?rhs$
proof –
assume $g: 0 < \text{nlength } l$
have 7: $0 < \text{nlength } \sigma$
by $(\text{metis } 3 \ g \ \text{gr-zeroI less-numeral-extra}(3) \ \text{nfinite-conv-nlength-enat nidx-less-last-1}$
 $\text{nnth-nlast the-enat.simps the-enat-0})$
have 8: $(\text{nsbn } \sigma (\text{nnth } l \ 0) (\text{nnth } l (\text{Suc } 0)) \models F)$
using $3 \ g$ **by** $(\text{metis zero-enat-def})$
have 9: $(\text{nsbn } \sigma (\text{nnth } l \ 0) (\text{nnth } l (\text{Suc } 0))) = (\text{ntaken } (\text{nnth } l (\text{Suc } 0)) \ \sigma)$
using 3 **by** $(\text{simp add: nsbn-def1})$
have 10: $(\text{ntaken } (\text{nnth } l (\text{Suc } 0)) \ \sigma) \models F$
using $9 \ 8$ **by** *auto*
have 11: $(\text{ntaken } (\text{nnth } l (\text{Suc } 0)) \ \sigma) \models ((F\text{-e iand empty}) \ \text{ior } (\text{big-ior } (\text{map2 } (\lambda x \ y. x \ \text{iand } (\text{next } y)) \ F\text{-now } F\text{-next})))$
using $10 \ \text{assms}(1) \ \text{itl-ieq}$ **by** *blast*
have 12: $(\text{ntaken } (\text{nnth } l (\text{Suc } 0)) \ \sigma) \models (\text{big-ior } (\text{map2 } (\lambda x \ y. x \ \text{iand } (\text{next } y)) \ F\text{-now } F\text{-next}))$
by $(\text{metis } 11 \ 3 \ 7 \ 9 \ \text{One-nat-def eSuc-enat empty-defs enat-defs}(1) \ g \ \text{iand-defs ileI1}$
 $\text{ior-defs less-numeral-extra}(1) \ \text{linorder-le-cases nidx-gr-first nsbn-nlength-gr-one}$
 $\text{ntaken-all zero-less-iff-neq-zero})$
have 13: $0 < \text{nlength } (\text{ntaken } (\text{nnth } l (\text{Suc } 0)) \ \sigma)$
by $(\text{metis } 3 \ 7 \ 9 \ \text{One-nat-def eSuc-enat enat-defs}(1) \ g \ \text{ileI1 less-numeral-extra}(1)$
 $\text{linorder-le-cases nidx-gr-first nsbn-nlength-gr-one ntaken-all})$
have 14: $(\text{nlength } (\text{ntaken } (\text{nnth } l (\text{Suc } 0)) \ \sigma) > 0 \wedge$
 $(\exists i < \text{length } F\text{-now. } ((\text{NNil } (\text{nfirst } (\text{ntaken } (\text{nnth } l (\text{Suc } 0)) \ \sigma))) \models F\text{-now } !i) \wedge$
 $((\text{ndropn } 1 (\text{ntaken } (\text{nnth } l (\text{Suc } 0)) \ \sigma)) \models F\text{-next } !i)))$
using $\text{GNF-defs}[of \ F \ F\text{-e } F\text{-now } F\text{-next } (\text{ntaken } (\text{nnth } l (\text{Suc } 0)) \ \sigma)] \ 12 \ 13$
by $(\text{metis } 10 \ \text{assms}(1) \ \text{assms}(2) \ \text{assms}(3) \ \text{assms}(4) \ \text{order-less-irrefl})$
obtain i **where** $15: i < \text{length } F\text{-now} \wedge ((\text{NNil } (\text{nfirst } (\text{ntaken } (\text{nnth } l (\text{Suc } 0)) \ \sigma))) \models F\text{-now } !i) \wedge$
 $((\text{ndropn } 1 (\text{ntaken } (\text{nnth } l (\text{Suc } 0)) \ \sigma)) \models F\text{-next } !i)$
using 14 **by** *blast*
have 19: $((\text{ndropn } 1 \ \sigma) \models F\text{-next } !i \ \text{schop } (\text{schopstar } F)) \longleftrightarrow$
 $(\exists ia. \text{enat } ia \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge$

$(\text{ntaken } ia \text{ (ndropn (Suc 0) } \sigma) \models F\text{-next ! } i) \wedge$
 $(\exists l. \text{nnth } l \ 0 = 0 \wedge$
 $\quad \text{nfinite } l \wedge$
 $\quad \text{nidx } l \wedge$
 $\quad \text{enat (nlast } l) = \text{nlength } \sigma - \text{enat (Suc 0) - enat } ia \wedge \text{nfinite } \sigma \wedge$
 $\quad (\forall i. \text{enat } i < \text{nlength } l \longrightarrow$
 $\quad \quad (\text{nsbn (ndropn } ia \text{ (ndropn (Suc 0) } \sigma)) (\text{nnth } l \ i) (\text{nnth } l \text{ (Suc } i)) \models F)))$
by simp
have 191: $\text{enat } ((\text{nnth } l \text{ (Suc 0)}) - 1) \leq \text{nlength } \sigma - \text{enat (Suc 0)}$
by (*metis 3 One-nat-def eSuc-enat enat-minus-mono1 enat-ord-simps(1) g idiff-enat-enat ileI1 nidx-all-le-nlast zero-enat-def*)
have 192: $\text{ntaken } ((\text{nnth } l \text{ (Suc 0)}) - 1) \text{ (ndropn (Suc 0) } \sigma) \models F\text{-next ! } i$
by (*metis 15 191 3 7 One-nat-def Suc-leI diff-add eSuc-enat enat-min-eq g ileI1 less-numeral-extra(1) nidx-gr-first ntaken-ndropn-swap zero-enat-def*)
have 193: $\text{nnth (nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) \text{ (ndropn (Suc 0) } l)) \ 0 = 0$
using *enat-defs(1)* **by auto**
have 194: $\text{nfinite (nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) \text{ (ndropn (Suc 0) } l))$
by (*simp add: 3*)
have 195: $\text{nidx (ndropn 1 } l)$
by (*metis 3 g ileI1 nidx-ndropn one-eSuc one-enat-def*)
have 196: $\text{nnth (ndropn 1 } l) \ 0 = (\text{nnth } l \text{ (Suc 0)})$
by simp
have 197: $\text{nidx (nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) \text{ (ndropn (Suc 0) } l))$
using *nidx-shiftm[of (ndropn 1 l) (nnth l (Suc 0))]*
by (*metis 195 196 One-nat-def*)
have 198: $\text{nlast (ndropn (Suc 0) } l) = \text{nlast } l$
by (*metis Suc-ile-eq g nappend-NNil nappend-ntaken-ndropn nlast-NCons ntaken-0 zero-enat-def*)
have 199: $\text{nlast (nmap } (\lambda x. x - \text{nnth } l \text{ (Suc 0)}) \text{ (ndropn (Suc 0) } l)) =$
 $\quad \text{nlength } \sigma - \text{nnth } l \text{ (Suc 0)}$
using *nlast-nmap[of (ndropn (Suc 0) l) ($\lambda x. x - (\text{nnth } l \text{ (Suc 0)})$)]*
using 194 *nfinite-nmap*
by (*metis 198 3 idiff-enat-enat*)
have 200: $\text{nlength } \sigma - \text{nnth } l \text{ (Suc 0) = nlength } \sigma - \text{enat (Suc 0) - } ((\text{nnth } l \text{ (Suc 0)}) - 1)$
by auto
 $(\text{metis 3 One-nat-def Suc-pred g ileI1 less-numeral-extra(1) ndropn-ndropn}$
 $\quad \text{ndropn-nlength nidx-gr-first one-eSuc one-enat-def plus-1-eq-Suc})$
have 201: $\text{enat (nlast (nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) \text{ (ndropn (Suc 0) } l))) =$
 $\quad \text{nlength } \sigma - \text{enat (Suc 0) - } ((\text{nnth } l \text{ (Suc 0)}) - 1)$
using 199 200 **by presburger**
have 202: $\text{nlength (nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) \text{ (ndropn (Suc 0) } l)) = \text{nlength } l - (\text{Suc } 0)$
by force
have 203: $(\text{ndropn } ((\text{nnth } l \text{ (Suc 0)}) - 1) \text{ (ndropn (Suc 0) } \sigma)) = (\text{ndropn (nnth } l \text{ (Suc 0)) } \sigma)$
by (*metis 3 One-nat-def Suc-pred g ileI1 ndropn-ndropn nidx-gr-first one-eSuc one-enat-def plus-1-eq-Suc zero-less-one*)
have 204: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$
 $\quad \text{nnth (nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) \text{ (ndropn (Suc 0) } l)) \ j =$
 $\quad \text{nnth } l \text{ (j + (Suc 0)) - (nnth } l \text{ (Suc 0))}$
by simp
have 2041: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$
 $\quad \text{nnth (nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) \text{ (ndropn (Suc 0) } l)) \text{ (Suc } j) =$

$$\text{nnth } l ((\text{Suc } j) + (\text{Suc } 0)) - (\text{nnth } l (\text{Suc } 0))$$
by (*metis add.right-neutral add-Suc-right eSuc-enat ileI1 ndropn-nlength ndropn-nnth nnth-nmap plus-1-eq-Suc*)

have 205: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$

$$\text{nnth } l (j + (\text{Suc } 0)) - (\text{nnth } l (\text{Suc } 0)) < \text{nnth } l ((\text{Suc } j) + (\text{Suc } 0)) - (\text{nnth } l (\text{Suc } 0))$$
using 204 197 **unfolding** *nidx-expand* **by** *auto*
(metis 204 One-nat-def add commute eSuc-enat ileI1 plus-1-eq-Suc)

have 206: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$

$$\begin{aligned} & (\text{nsubn } (\text{ndropn } ((\text{nnth } l (\text{Suc } 0)) - 1)) (\text{ndropn } (\text{Suc } 0) \sigma)) \\ & (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l (\text{Suc } 0)))) (\text{ndropn } (\text{Suc } 0) l) j) \\ & (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l (\text{Suc } 0)))) (\text{ndropn } (\text{Suc } 0) l) (\text{Suc } j))) = \\ & (\text{nsubn } (\text{ndropn } (\text{nnth } l (\text{Suc } 0))) \sigma) \\ & (\text{nnth } l (j + (\text{Suc } 0)) - (\text{nnth } l (\text{Suc } 0))) \\ & (\text{nnth } l ((\text{Suc } j) + (\text{Suc } 0)) - (\text{nnth } l (\text{Suc } 0)))) \end{aligned}$$
using 203 204 2041 **by** *simp*

have 207: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$

$$\begin{aligned} & (\text{nsubn } (\text{ndropn } (\text{nnth } l (\text{Suc } 0)) \sigma) \\ & (\text{nnth } l (j + (\text{Suc } 0)) - (\text{nnth } l (\text{Suc } 0))) \\ & (\text{nnth } l ((\text{Suc } j) + (\text{Suc } 0)) - (\text{nnth } l (\text{Suc } 0)))) = \\ & (\text{nsubn } \sigma (\text{nnth } l (\text{Suc } j)) (\text{nnth } l (\text{Suc } (\text{Suc } j)))) \end{aligned}$$
using *nsubn-ndropn[of - - \sigma (\text{nnth } l (\text{Suc } 0))]* **by** *simp*
(metis (no-types, lifting) 197 202 204 205 add commute add.right-neutral bot-nat-0.not-eq-extremum diff-add less-nat-zero-code nidx-gr-first order-less-imp-le plus-1-eq-Suc zero-less-diff)

have 208: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$

$$(\text{nsubn } \sigma (\text{nnth } l (\text{Suc } j)) (\text{nnth } l (\text{Suc } (\text{Suc } j)))) \models F$$
using 3

by (*metis eSuc-enat enat-min g ileI1 one-eSuc plus-1-eSuc(2) zero-enat-def*)

have 209: $(\forall i. \text{enat } i < \text{nlength } (\text{nmap } (\lambda x. x - (\text{nnth } l (\text{Suc } 0)))) (\text{ndropn } (\text{Suc } 0) l)) \longrightarrow$

$$\begin{aligned} & (\text{nsubn } (\text{ndropn } ((\text{nnth } l (\text{Suc } 0)) - 1)) (\text{ndropn } (\text{Suc } 0) \sigma)) \\ & (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l (\text{Suc } 0)))) (\text{ndropn } (\text{Suc } 0) l) i) \\ & (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l (\text{Suc } 0)))) (\text{ndropn } (\text{Suc } 0) l) (\text{Suc } i)) \models F) \end{aligned}$$
using 202 206 207 208 **by** *presburger*

have 210: $(\text{enat } ((\text{nnth } l (\text{Suc } 0)) - 1) \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge$

$$\begin{aligned} & (\text{ntaken } ((\text{nnth } l (\text{Suc } 0)) - 1) (\text{ndropn } (\text{Suc } 0) \sigma) \models F\text{-next } ! i) \wedge \\ & (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l (\text{Suc } 0)))) (\text{ndropn } (\text{Suc } 0) l) 0 = 0 \wedge \\ & \text{nfinite } (\text{nmap } (\lambda x. x - (\text{nnth } l (\text{Suc } 0)))) (\text{ndropn } (\text{Suc } 0) l) \wedge \\ & \text{nidx } (\text{nmap } (\lambda x. x - (\text{nnth } l (\text{Suc } 0)))) (\text{ndropn } (\text{Suc } 0) l) \wedge \\ & \text{enat } (\text{nlast } (\text{nmap } (\lambda x. x - (\text{nnth } l (\text{Suc } 0)))) (\text{ndropn } (\text{Suc } 0) l))) = \\ & \text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ((\text{nnth } l (\text{Suc } 0)) - 1) \wedge \text{nfinite } \sigma \wedge \\ & (\forall i. \text{enat } i < \text{nlength } (\text{nmap } (\lambda x. x - (\text{nnth } l (\text{Suc } 0)))) (\text{ndropn } (\text{Suc } 0) l)) \longrightarrow \\ & (\text{nsubn } (\text{ndropn } ((\text{nnth } l (\text{Suc } 0)) - 1)) (\text{ndropn } (\text{Suc } 0) \sigma) \\ & (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l (\text{Suc } 0)))) (\text{ndropn } (\text{Suc } 0) l) i) \\ & (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l (\text{Suc } 0)))) (\text{ndropn } (\text{Suc } 0) l) (\text{Suc } i)) \models F))) \end{aligned}$$
using 191 192 193 194 197 201 209 3 **by** *blast*

have 20: $((\text{ndropn } 1 \sigma) \models F\text{-next } ! i \text{ schop } (\text{schoptstar } F))$
using 210 19 **by** *blast*

have 21: $\sigma \models F\text{-now } ! i$
by (*metis 15 assms(3) ntaken-nfirst state-pitl-defs state-pitl-list-defs*)

have 22: $\text{length } F\text{-now} = \text{length } (\text{map } (\lambda x. x \text{ schop } \text{schoptstar } F)) F\text{-next}$

```

    by (simp add: assms(4))
  have 23: ( $\sigma \models \text{big-ior} (\text{map2 } (\lambda x y. x \text{ iand next } y) F\text{-now } (\text{map } (\lambda x. x \text{ schop schopstar } F) ) F\text{-next}))$ )
 $\longleftrightarrow$ 
    ( $\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now} ! i \text{ iand next map } (\lambda x. x \text{ schop schopstar } F) ) F\text{-next} ! i$ )
  using big-ior-map2-defs[of  $F\text{-now } (\text{map } (\lambda x. x \text{ schop } (\text{s chopstar } F)) F\text{-next}) \sigma (\lambda x y. x \text{ iand next } y)$ ]]
  22 by auto
  have 24: ( $\sigma \models F\text{-now} ! i \text{ iand next map } (\lambda x. x \text{ schop schopstar } F) ) F\text{-next} ! i$ )
    by (metis 15 20 21 7 assms(4) iand-defs ileI1 nth-map one-eSuc semantics-pitl.simps(4))
  show ?thesis using 15 23 24 ior-defs by blast
qed
show ?thesis
using 4 6 zero-less-iff-neq-zero by blast
qed
have 1000: ?rhs  $\implies$  ?lhs
proof -
  assume r: ?rhs
  have 1001:  $n\text{length } \sigma = 0 \vee (\sigma \models \text{big-ior} (\text{map2 } (\lambda x y. x \text{ iand next } y) F\text{-now } (\text{map } (\lambda x. x \text{ schop schopstar } F) ) F\text{-next}))$ )
    using r by simp
  have 1002:  $n\text{length } \sigma = 0 \implies ?lhs$ 
  proof -
    assume n:  $n\text{length } \sigma = 0$ 
    have 1003:  $n\text{nth } (NNil\ 0)\ 0 = 0$ 
      by (simp add: nnth-NNil)
    have 1004:  $n\text{finite } (NNil\ 0)$ 
      by simp
    have 1005:  $n\text{idx } (NNil\ 0)$ 
      using Suc-ile-eq nidx-expand by auto
    have 1006:  $\text{enat } (n\text{last } (NNil\ 0)) = n\text{length } \sigma$ 
      by (simp add: n zero-enat-def)
    have 1007:  $n\text{finite } \sigma$ 
      using n
      by (simp add: nfinite-conv-nlength-enat zero-enat-def)
    have 1008:  $(\forall i. \text{enat } i < n\text{length } (NNil\ 0) \longrightarrow (\text{nsubn } \sigma (\text{nnth } (NNil\ 0)\ i) (\text{nnth } (NNil\ 0) (\text{Suc } i))$ 
 $\models F))$ 
      by auto
    show ?thesis apply simp
      using 1003 1005 1006 1007 1008 by blast
  qed
  have 1009: ( $\sigma \models \text{big-ior} (\text{map2 } (\lambda x y. x \text{ iand next } y) F\text{-now } (\text{map } (\lambda x. x \text{ schop schopstar } F) ) F\text{-next}))$ )
 $\implies$  ?lhs
  proof -
    assume b: ( $\sigma \models \text{big-ior} (\text{map2 } (\lambda x y. x \text{ iand next } y) F\text{-now } (\text{map } (\lambda x. x \text{ schop schopstar } F) ) F\text{-next}))$ )
    have 1010:  $\text{length } F\text{-now} = \text{length } (\text{map } (\lambda x. x \text{ schop schopstar } F) ) F\text{-next}$ 
      by (simp add: assms(4))
    have 1011: ( $\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now} ! i \text{ iand next map } (\lambda x. x \text{ schop schopstar } F) ) F\text{-next} ! i$ )
      using 1010 b big-ior-map2-defs[of  $F\text{-now } (\text{map } (\lambda x. x \text{ schop } (\text{s chopstar } F)) F\text{-next}) \sigma (\lambda x y. x \text{ iand next } y)$ ]]
      by blast
  
```


obtain i **where** 1012: $i < \text{length } F\text{-now} \wedge (\sigma \models F\text{-now} ! i \text{ and next map } (\lambda x. x \text{ schop schopstar } F) \text{ } F\text{-next} ! i)$
using 1011 **by** blast
have 1013: $(\sigma \models F\text{-now} ! i)$
using 1012 **by** auto
have 1014: $((\text{ndropn } 1 \sigma) \models F\text{-next} ! i \text{ schop } (\text{schoptstar } F))$
using 1012 *assms*(4) **by** fastforce
have 1015: $((\text{ndropn } 1 \sigma) \models F\text{-next} ! i \text{ schop } (\text{schoptstar } F)) \longleftrightarrow$
 $(\exists ia. \text{enat } ia \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge$
 $(\text{ntaken } ia (\text{ndropn } (\text{Suc } 0) \sigma) \models F\text{-next} ! i) \wedge$
 $(\exists l. \text{nnth } l \ 0 = 0 \wedge$
 $\text{nfinite } l \wedge$
 $\text{nidr } l \wedge$
 $\text{enat } (\text{nlast } l) = \text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia \wedge \text{nfinite } \sigma \wedge$
 $(\forall i. \text{enat } i < \text{nlength } l \longrightarrow (\text{subn } (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0) \sigma)) (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))$
 $\models F))$
by simp
obtain ia **where** 1016: $\text{enat } ia \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge$
 $(\text{ntaken } ia (\text{ndropn } (\text{Suc } 0) \sigma) \models F\text{-next} ! i) \wedge$
 $(\exists l. \text{nnth } l \ 0 = 0 \wedge$
 $\text{nfinite } l \wedge$
 $\text{nidr } l \wedge$
 $\text{enat } (\text{nlast } l) = \text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia \wedge \text{nfinite } \sigma \wedge$
 $(\forall i. \text{enat } i < \text{nlength } l \longrightarrow (\text{subn } (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0) \sigma)) (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))$
 $\models F))$
using 1014 1015 **by** blast
obtain l **where** 1017: $\text{nnth } l \ 0 = 0 \wedge$
 $\text{nfinite } l \wedge$
 $\text{nidr } l \wedge$
 $\text{enat } (\text{nlast } l) = \text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia \wedge \text{nfinite } \sigma \wedge$
 $(\forall i. \text{enat } i < \text{nlength } l \longrightarrow (\text{subn } (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0) \sigma)) (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))$
 $\models F))$
using 1016 **by** blast
have 1018: $0 < \text{nlength } \sigma$
using 1012 **by** force
have 10183: $\text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia = \text{nlength } \sigma - \text{Suc } ia$
by (metis One-nat-def ndropn-ndropn ndropn-nlength plus-1-eq-Suc)
have 10184: $\text{Suc } ia \leq \text{nlength } \sigma$
by (metis 1016 1018 One-nat-def eSuc-enat enat-min-eq ileI1 one-eSuc one-enat-def plus-1-eSuc(2))
have 10185: $\text{nlength } \sigma - \text{Suc } ia + \text{Suc } ia = \text{nlength } \sigma$
by (metis 10184 add commute enat.simps(3) enat-add-sub-same le-iff-add)
have 1019: $(\text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia) + \text{Suc } ia = \text{nlength } \sigma$
by (simp add: 10183 10185)
have 1020: $\text{nlast } (\text{nmap } (\lambda x. x + \text{Suc } ia) \ l) = \text{nlength } \sigma$
by (metis 1017 1019 nlast-nmap plus-enat-simps(1))
have 1021: $\text{nlast } (\text{NCons } 0 \ (\text{nmap } (\lambda x. x + \text{Suc } ia) \ l)) = \text{nlength } \sigma$
using 1020 **by** force
have 1022: $\text{nnth } (\text{NCons } 0 \ (\text{nmap } (\lambda x. x + \text{Suc } ia) \ l)) \ 0 = 0$
by simp
have 1023: $0 < \text{nnth } (\text{NCons } 0 \ (\text{nmap } (\lambda x. x + \text{Suc } ia) \ l)) \ 1$

```

using zero-enat-def by force
have 1024:  $nfinite\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))$ 
by (simp add: 1017)
have 1025:  $nidx\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))$ 
using 1017  $nidx$ -Ncons-shift by blast
have 1028:  $(nsubn\ \sigma\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))\ 0)\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))\ (Suc\ 0))) =$ 
 $(nsubn\ \sigma\ 0\ (Suc\ ia))$ 
using 1017 zero-enat-def by fastforce
have 1029:  $((nsubn\ \sigma\ 0\ (Suc\ ia)) \models (big\text{-}ior\ (map2\ (\lambda x\ y. x\ iand\ (next\ y))\ F\text{-}now\ F\text{-}next))) \longleftrightarrow$ 
 $(\exists i < length\ F\text{-}now. nsubn\ \sigma\ 0\ (Suc\ ia) \models F\text{-}now\ !\ i\ iand\ next\ F\text{-}next\ !\ i)$ 
using big-ior-map2-defs[of  $F\text{-}now\ F\text{-}next\ (nsubn\ \sigma\ 0\ (Suc\ ia))\ (\lambda x\ y. x\ iand\ (next\ y))$ ]
assms by blast
have 1030:  $nsubn\ \sigma\ 0\ (Suc\ ia) \models F\text{-}now\ !\ i$ 
using 1016
by (metis 1012 1013  $assms(3)\ ndropn\text{-}0\ nsubn\text{-}def1\ ntaken\text{-}nfirst\ state\text{-}pitl\text{-}defs\ state\text{-}pitl\text{-}list\text{-}defs$ )
have 1031:  $nsubn\ \sigma\ 0\ (Suc\ ia) \models next\ F\text{-}next\ !\ i$ 
using 1016  $ndropn\text{-}nsubn$ [of  $Suc\ ia\ \sigma\ Suc\ 0\ 0$ ]
by (metis 1019  $One\text{-}nat\text{-}def\ add\text{-}diff\text{-}cancel\text{-}left'\ enat\text{-}le\text{-}plus\text{-}same(2)\ ileI1\ le\text{-}add1$ 
 $nsubn\text{-}def1\ nsubn\text{-}nlength\text{-}gr\text{-}one\ one\text{-}eSuc\ plus\text{-}1\text{-}eq\text{-}Suc\ semantics\text{-}pitl.simps(4)\ zero\text{-}less\text{-}Suc$ )
have 1034:  $(nsubn\ \sigma\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))\ 0)\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))\ (Suc\ 0))) \models$ 
 $((F\text{-}e\ iand\ empty)\ ior\ (big\text{-}ior\ (map2\ (\lambda x\ y. x\ iand\ (next\ y))\ F\text{-}now\ F\text{-}next))))$ 
using 1012 1028 1029 1030 1031 by auto
have 1035:  $\bigwedge j. j < nlength\ l \implies$ 
 $(nnth\ ((nmap\ (\lambda x. x + Suc\ ia)\ l))\ j) = nnth\ l\ j + Suc\ ia$ 
by simp
have 1036:  $\bigwedge j. j < nlength\ l \implies$ 
 $(nnth\ ((nmap\ (\lambda x. x + Suc\ ia)\ l))\ (Suc\ j)) = nnth\ l\ (Suc\ j) + Suc\ ia$ 
using  $Suc\text{-}ile\text{-}eq$  by force
have 1037:  $\bigwedge j. j < nlength\ l \implies$ 
 $nnth\ l\ j + Suc\ ia < nnth\ l\ (Suc\ j) + Suc\ ia$ 
by (metis 1017  $add\text{-}strict\text{-}right\text{-}mono\ eSuc\text{-}enat\ ileI1\ nidx\text{-}expand$ )
have 1038:  $\bigwedge j. j < nlength\ l \implies$ 
 $enat\ (nnth\ l\ (Suc\ j) + Suc\ ia) \leq nlength\ \sigma$ 
using 1017  $nidx\text{-}expand$  by simp
 $(metis\ 1020\ 1025\ 1036\ add\text{-}Suc\text{-}right\ eSuc\text{-}enat\ enat\text{-}ord\text{-}simps(1)\ ileI1\ nfinite\text{-}nmap$ 
 $nidx\text{-}LCons\text{-}1\ nidx\text{-}all\text{-}le\text{-}nlast\ nlength\text{-}nmap)$ 
have 1039:  $\bigwedge j. j < nlength\ l \implies$ 
 $Suc\ 0 + (nnth\ l\ j + Suc\ ia) \leq nnth\ l\ (Suc\ j) + Suc\ ia$ 
using 1037 by fastforce
have 1040:  $\bigwedge j. j < nlength\ l \implies$ 
 $ndropn\ (Suc\ 0)\ (nsubn\ \sigma\ (nnth\ l\ j + Suc\ ia)\ (nnth\ l\ (Suc\ j) + Suc\ ia)) =$ 
 $nsubn\ \sigma\ (Suc\ 0 + (nnth\ l\ j + Suc\ ia))\ (nnth\ l\ (Suc\ j) + Suc\ ia)$ 
using  $ndropn\text{-}nsubn$ [of  $\sigma\ Suc\ 0$ ] using 1038 1039 by presburger
have 1041:  $\bigwedge j. j < nlength\ l \implies$ 
 $(nsubn\ (ndropn\ ia\ (ndropn\ (Suc\ 0)\ \sigma))\ (nnth\ l\ j)\ (nnth\ l\ (Suc\ j))) =$ 
 $nsubn\ \sigma\ ((nnth\ l\ j) + Suc\ ia)\ ((nnth\ l\ (Suc\ j)) + Suc\ ia)$ 
using  $nsubn\text{-}ndropn$ [of  $\sigma\ Suc\ ia$ ]
by (metis 1037  $One\text{-}nat\text{-}def\ add\text{-}less\text{-}cancel\text{-}right\ ndropn\text{-}ndropn\ plus\text{-}1\text{-}eq\text{-}Suc$ )

```

have 1280: $(\text{nsbn } \sigma \text{ (nnth (NCons 0 (nmap } (\lambda x. x + \text{Suc ia}) l)) 0) \text{ (nnth (NCons 0 (nmap } (\lambda x. x + \text{Suc ia}) l)) (Suc 0))} \models F)$
using 1034 *assms(1)* **by simp**
have 1281: $(\forall i. \text{enat } i < \text{nlength } (\text{nmap } (\lambda x. x + \text{Suc ia}) l)) \longrightarrow$
 $(\text{nsbn } \sigma \text{ (nnth } (\text{nmap } (\lambda x. x + \text{Suc ia}) l)) i) \text{ (nnth } (\text{nmap } (\lambda x. x + \text{Suc ia}) l)) (Suc i)) \models$
 $F))$
using 1017 1036 1041 **by auto**
have 1290: $(\forall i. \text{enat } i < \text{nlength } (\text{NCons 0 (nmap } (\lambda x. x + \text{Suc ia}) l)) \longrightarrow$
 $(\text{nsbn } \sigma \text{ (nnth (NCons 0 (nmap } (\lambda x. x + \text{Suc ia}) l)) i) \text{ (nnth (NCons 0 (nmap } (\lambda x. x + \text{Suc ia}) l))$
 $(Suc i)) \models F)) \longleftrightarrow$
 $(\text{nsbn } \sigma \text{ (nnth (NCons 0 (nmap } (\lambda x. x + \text{Suc ia}) l)) 0) \text{ (nnth (NCons 0 (nmap } (\lambda x. x + \text{Suc}$
 $\text{ia}) l)) (Suc 0))} \models F) \wedge$
 $(\forall i. \text{enat } i < \text{nlength } (\text{nmap } (\lambda x. x + \text{Suc ia}) l)) \longrightarrow$
 $(\text{nsbn } \sigma \text{ (nnth } (\text{nmap } (\lambda x. x + \text{Suc ia}) l)) i) \text{ (nnth } (\text{nmap } (\lambda x. x + \text{Suc ia}) l)) (Suc i)) \models$
 $F))$
using forall-ncons-split *[of (nmap } (\lambda x. x + \text{Suc ia}) l*
 $\lambda x y. (\text{nsbn } \sigma x y \models F)]$ **by (simp add: 1017)**
have 1300: $(\forall i. \text{enat } i < \text{nlength } (\text{NCons 0 (nmap } (\lambda x. x + \text{Suc ia}) l)) \longrightarrow$
 $(\text{nsbn } \sigma \text{ (nnth (NCons 0 (nmap } (\lambda x. x + \text{Suc ia}) l)) i) \text{ (nnth (NCons 0 (nmap } (\lambda x. x + \text{Suc ia}) l))$
 $(Suc i)) \models F))$
using 1280 1281 1290 **by blast**
have 1400: $\exists l. \text{nnth } l 0 = 0 \wedge \text{nfinite } l \wedge \text{nidx } l \wedge \text{enat } (\text{nlast } l) = \text{nlength } \sigma \wedge \text{nfinite } \sigma \wedge$
 $(\forall i. \text{enat } i < \text{nlength } l \longrightarrow (\text{nsbn } \sigma \text{ (nnth } l i) \text{ (nnth } l (Suc i))} \models F))$
by (meson 1017 1021 1022 1025 1300 nfinite-NConsI nfinite-nmap)
show ?thesis using 1400 **by simp**
qed
show ?thesis
using 1001 1002 1009 **by blast**
qed
show ?thesis
using 1000 2 **by blast**
qed

lemma GNF-omega-step:

assumes $\vdash F \text{ ieqv } ((F\text{-e iand empty}) \text{ ior } (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) F\text{-now } F\text{-next})))$
 $\text{state-pitl } F\text{-e}$
 $\text{state-pitl-list } F\text{-now}$
 $\text{length } F\text{-now} = \text{length } F\text{-next}$
shows $(\sigma \models \text{omega } F) \longleftrightarrow$
 $(\sigma \models (\text{ifalse iand empty}) \text{ ior}$
 $(\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) F\text{-now } (\text{map } (\lambda x. x \text{ schop } (\text{omega } F)) F\text{-next}))))$
 $(\text{is ?lhs} \longleftrightarrow \text{?rhs})$
proof –
have 1: $(\sigma \models \text{omega } F) \longleftrightarrow$
 $(\exists l. \neg \text{nfinite } l \wedge \text{nnth } l 0 = 0 \wedge \text{nidx } l \wedge$
 $(\forall i. (\text{nnth } l i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. (\text{nsbn } \sigma \text{ (nnth } l i) \text{ (nnth } l (Suc i))} \models F)))$
by simp

```

have 2: ?lhs  $\implies$  ?rhs
proof –
  assume l: ?lhs
  obtain l where 3:  $\neg \text{nfinite } l \wedge \text{nnth } l \ 0 = 0 \wedge \text{nidx } l \wedge$ 
     $(\forall i. (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$ 
     $(\forall i. (\text{nsubn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)) \models F))$ 
    using l 1 by blast
  have 6:  $\neg \text{nfinite } \sigma$ 
    using 3 infinite-nidx-imp-infinite-interval by blast
  have 7:  $0 < \text{nlength } \sigma$ 
    using 6 enat-defs(1) nfinite-conv-nlength-enat by fastforce
  have 8:  $(\text{nsubn } \sigma (\text{nnth } l \ 0) (\text{nnth } l \ (\text{Suc } 0)) \models F)$ 
    using 3 by blast
  have 9:  $(\text{nsubn } \sigma (\text{nnth } l \ 0) (\text{nnth } l \ (\text{Suc } 0))) = (\text{ntaken } (\text{nnth } l \ (\text{Suc } 0)) \ \sigma)$ 
    using 3 by (simp add: nsubn-def1)
  have 10:  $(\text{ntaken } (\text{nnth } l \ (\text{Suc } 0)) \ \sigma) \models F$ 
    using 9 8 by auto
  have 11:  $(\text{ntaken } (\text{nnth } l \ (\text{Suc } 0)) \ \sigma) \models ((F\text{-e iand empty}) \text{ ior } (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y))$ 
F-now F-next)))
    using 10 assms(1) itl-ieq by blast
  have 12:  $(\text{ntaken } (\text{nnth } l \ (\text{Suc } 0)) \ \sigma) \models (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) \text{ F-now F-next}))$ 
    by (metis 11 3 9 One-nat-def empty-defs enat-defs(2) iand-defs iless-eSuc0 ior-defs
less-numeral-extra(1) less-numeral-extra(3) nidx-gr-first nlength-eq-enat-nfiniteD
not-less nsubn-nlength-gr-one one-eSuc zero-enat-def)
  have 13:  $0 < \text{nlength } (\text{ntaken } (\text{nnth } l \ (\text{Suc } 0)) \ \sigma)$ 
    by (metis 3 9 One-nat-def iless-eSuc0 less-numeral-extra(1) linorder-not-less
nidx-gr-first nlength-eq-enat-nfiniteD nsubn-nlength-gr-one one-eSuc one-enat-def zero-enat-def)
  have 14:  $(\text{nlength } (\text{ntaken } (\text{nnth } l \ (\text{Suc } 0)) \ \sigma) > 0 \wedge$ 
     $(\exists i < \text{length } F\text{-now}. ((\text{NNil } (\text{nfirst } (\text{ntaken } (\text{nnth } l \ (\text{Suc } 0)) \ \sigma))) \models F\text{-now } !i) \wedge$ 
     $((\text{ndropn } 1 (\text{ntaken } (\text{nnth } l \ (\text{Suc } 0)) \ \sigma)) \models F\text{-next } !i)))$ 
    using GNF-defs[of F F-e F-now F-next (ntaken (nnth l (Suc 0)) sigma)] 12 13
    by (metis 10 assms(1) assms(2) assms(3) assms(4) order-less-irrefl)
  obtain i where 15:  $i < \text{length } F\text{-now} \wedge ((\text{NNil } (\text{nfirst } (\text{ntaken } (\text{nnth } l \ (\text{Suc } 0)) \ \sigma))) \models F\text{-now } !i) \wedge$ 
     $((\text{ndropn } 1 (\text{ntaken } (\text{nnth } l \ (\text{Suc } 0)) \ \sigma)) \models F\text{-next } !i)$ 
    using 14 by blast
  have 19:  $((\text{ndropn } 1 \ \sigma) \models F\text{-next } !i \text{ schop } (\text{omega } F)) \longleftrightarrow$ 
     $(\exists ia. \text{enat } ia \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge$ 
     $(\text{ntaken } ia \ (\text{ndropn } (\text{Suc } 0) \ \sigma) \models F\text{-next } !i) \wedge$ 
     $(\exists l. \neg \text{nfinite } l \wedge$ 
     $\text{nnth } l \ 0 = 0 \wedge \text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia) \wedge$ 
     $(\forall i. \text{nsubn } (\text{ndropn } ia \ (\text{ndropn } (\text{Suc } 0) \ \sigma)) (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)) \models F)))$ 
    by simp
  have 191:  $\text{enat } ((\text{nnth } l \ (\text{Suc } 0)) - 1) \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0)$ 
    by (metis 3 One-nat-def enat-minus-mono1 idiff-enat-enat)
  have 192:  $\text{ntaken } ((\text{nnth } l \ (\text{Suc } 0)) - 1) \ (\text{ndropn } (\text{Suc } 0) \ \sigma) \models F\text{-next } !i$ 
    using 13 15 191 3 7
    by (simp add: enat-0-iff(1) ntaken-ndropn-swap)
  have 193:  $\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l \ (\text{Suc } 0))) (\text{ndropn } (\text{Suc } 0) \ l)) \ 0 = 0$ 
    using enat-defs(1) by auto
  have 194:  $\neg \text{nfinite } (\text{nmap } (\lambda x. x - (\text{nnth } l \ (\text{Suc } 0))) (\text{ndropn } (\text{Suc } 0) \ l))$ 

```

by (simp add: 3)
 have 195: $\text{nidx} (\text{ndropn } 1 \ l)$
 by (metis 3 One-nat-def Suc-ile-eq dual-order.order-iff-strict enat-defs(1)
 nfinite-conv-nlength-enat nidx-ndropn zero-le)
 have 196: $\text{nnth} (\text{ndropn } 1 \ l) \ 0 = (\text{nnth } l \ (\text{Suc } 0))$
 by simp
 have 197: $\text{nidx} (\text{nmap } (\lambda x. x - (\text{nnth } l \ (\text{Suc } 0))) (\text{ndropn } (\text{Suc } 0) \ l))$
 using nidx-shiftm[of (ndropn 1 l) (nnth l (Suc 0))]
 by (metis 195 196 One-nat-def)
 have 198: $(\forall i. \text{enat} (\text{nnth} (\text{nmap } (\lambda x. x - (\text{nnth } l \ (\text{Suc } 0))) (\text{ndropn } (\text{Suc } 0) \ l)) \ i) \leq \text{nlength } \sigma - \text{enat}$
 $(\text{Suc } 0) - \text{enat} ((\text{nnth } l \ (\text{Suc } 0)) - 1))$
 by (metis 6 ndropn-nlength nfinite-ndropn nfinite-ntaken nle-le ntaken-all)
 have 202: $\text{nlength} (\text{nmap } (\lambda x. x - (\text{nnth } l \ (\text{Suc } 0))) (\text{ndropn } (\text{Suc } 0) \ l)) = \text{nlength } l - (\text{Suc } 0)$
 by force
 have 203: $(\text{ndropn} ((\text{nnth } l \ (\text{Suc } 0)) - 1) (\text{ndropn } (\text{Suc } 0) \ \sigma)) = (\text{ndropn} (\text{nnth } l \ (\text{Suc } 0)) \ \sigma)$
 by (metis 3 One-nat-def Suc-pred ileI1 less-numeral-extra(1) ndropn-ndropn nidx-gr-first
 nlength-eq-enat-nfiniteD one-eSuc one-enat-def plus-1-eq-Suc zero-enat-def zero-less-iff-neq-zero)
 have 204: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$
 $\text{nnth} (\text{nmap } (\lambda x. x - (\text{nnth } l \ (\text{Suc } 0))) (\text{ndropn } (\text{Suc } 0) \ l)) \ j =$
 $\text{nnth } l \ (j + (\text{Suc } 0)) - (\text{nnth } l \ (\text{Suc } 0))$
 by simp
 have 2041: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$
 $\text{nnth} (\text{nmap } (\lambda x. x - (\text{nnth } l \ (\text{Suc } 0))) (\text{ndropn } (\text{Suc } 0) \ l)) \ (\text{Suc } j) =$
 $\text{nnth } l \ ((\text{Suc } j) + (\text{Suc } 0)) - (\text{nnth } l \ (\text{Suc } 0))$
 by (metis add.right-neutral add-Suc-right eSuc-enat ileI1 ndropn-nlength ndropn-nnth nnth-nmap
 plus-1-eq-Suc)
 have 205: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$
 $\text{nnth } l \ (j + (\text{Suc } 0)) - (\text{nnth } l \ (\text{Suc } 0)) < \text{nnth } l \ ((\text{Suc } j) + (\text{Suc } 0)) - (\text{nnth } l \ (\text{Suc } 0))$
 using 204 197 unfolding nidx-expand apply auto
 by (metis 204 One-nat-def add commute eSuc-enat ileI1 plus-1-eq-Suc)
 have 206: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$
 $(\text{nsubn} (\text{ndropn} ((\text{nnth } l \ (\text{Suc } 0)) - 1) (\text{ndropn } (\text{Suc } 0) \ \sigma))$
 $(\text{nnth} (\text{nmap } (\lambda x. x - (\text{nnth } l \ (\text{Suc } 0))) (\text{ndropn } (\text{Suc } 0) \ l)) \ j)$
 $(\text{nnth} (\text{nmap } (\lambda x. x - (\text{nnth } l \ (\text{Suc } 0))) (\text{ndropn } (\text{Suc } 0) \ l)) \ (\text{Suc } j))) =$
 $(\text{nsubn} (\text{ndropn} (\text{nnth } l \ (\text{Suc } 0)) \ \sigma)$
 $(\text{nnth } l \ (j + (\text{Suc } 0)) - (\text{nnth } l \ (\text{Suc } 0)))$
 $(\text{nnth } l \ ((\text{Suc } j) + (\text{Suc } 0)) - (\text{nnth } l \ (\text{Suc } 0))))$
 using 203 204 2041 by simp
 have 207: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$
 $(\text{nsubn} (\text{ndropn} (\text{nnth } l \ (\text{Suc } 0)) \ \sigma)$
 $(\text{nnth } l \ (j + (\text{Suc } 0)) - (\text{nnth } l \ (\text{Suc } 0)))$
 $(\text{nnth } l \ ((\text{Suc } j) + (\text{Suc } 0)) - (\text{nnth } l \ (\text{Suc } 0)))) =$
 $(\text{nsubn } \sigma \ (\text{nnth } l \ (\text{Suc } j)) \ (\text{nnth } l \ (\text{Suc } (\text{Suc } j))))$
 using nsubn-ndropn[of - - σ (nnth l (Suc 0))] by simp
 (metis (no-types, lifting) 197 202 204 205 add commute add.right-neutral
 bot-nat-0.not-eq-extremum diff-add less-nat-zero-code nidx-gr-first
 order-less-imp-le plus-1-eq-Suc zero-less-diff)
 have 208: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$
 $(\text{nsubn } \sigma \ (\text{nnth } l \ (\text{Suc } j)) \ (\text{nnth } l \ (\text{Suc } (\text{Suc } j)))) \models F$
 using 3 by blast

```

have 209: (∀ i. enat i < nlength (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) →
  (nsubn (ndropn ((nnth l (Suc 0)) - 1) (ndropn (Suc 0) σ))
    (nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) i)
    (nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) (Suc i)) ⊨ F))
  using 202 206 207 208 by presburger
have 2090: nlength (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) = ∞
  by (metis 194 enat2-cases nlength-eq-enat-nfiniteD)
have 2091: (∀ i.
  (nsubn (ndropn ((nnth l (Suc 0)) - 1) (ndropn (Suc 0) σ))
    (nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) i)
    (nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) (Suc i)) ⊨ F))
  using 2090 209
  using enat-ord-code(4) by presburger
let ?ia = ((nnth l (Suc 0)) - 1)
let ?l = (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l))
have 210: ( enat ?ia ≤ nlength σ - enat (Suc 0) ∧
  (ntaken ?ia (ndropn (Suc 0) σ) ⊨ F-next ! i) ∧
  (¬nfinite ?l ∧
    nnth ?l 0 = 0 ∧
    nidx ?l ∧
    (∀ i. enat (nnth ?l i) ≤ nlength σ - enat (Suc 0) - enat ?ia) ∧
    (∀ i.
      (nsubn (ndropn ?ia (ndropn (Suc 0) σ))
        (nnth ?l i)
        (nnth ?l (Suc i)) ⊨ F))
    )
  )
  using 191 192 193 194 197 198 2091 by blast
have 20: ((ndropn 1 σ) ⊨ F-next ! i schop (omega F))
  using 210 19 by blast
have 21: σ ⊨ F-now ! i
  by (metis 15 assms(3) ntaken-nfirst state-pitl-defs state-pitl-list-defs)
have 22: length F-now = length (map (λx. x schop omega F) F-next)
  by (simp add: assms(4))
have 23: (σ ⊨ big-ior (map2 (λx y. x iand next y) F-now (map (λx. x schop omega F) F-next))) ↔
  (∃ i < length F-now. σ ⊨ F-now ! i iand next map (λx. x schop omega F) F-next ! i)
  using big-ior-map2-defs[of F-now (map (λx. x schop (omega F)) F-next) σ (λ x y. x iand next y)]

  22 by auto
have 24: (σ ⊨ F-now ! i iand next map (λx. x schop omega F) F-next ! i)
  by (metis 15 20 21 7 assms(4) iand-defs ileI1 nth-map one-eSuc semantics-pitl.simps(4))
show ?thesis using 15 23 24 ior-defs by blast
qed
have 1000: ?rhs ⇒ ?lhs
proof -
assume r: ?rhs
have 1001: (σ ⊨ big-ior (map2 (λx y. x iand next y) F-now (map (λx. x schop omega F) F-next)))
  using r by simp
have 1010: length F-now = length (map (λx. x schop omega F) F-next)
  by (simp add: assms(4))

```

```

have 1011: ( $\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now} ! i$  iand  $\text{next map } (\lambda x. x \text{ schop } \omega F) F\text{-next} ! i$ )
  using 1001 1010 big-ior-map2-defs[of  $F\text{-now}$  ( $\text{map } (\lambda x. x \text{ schop } (\omega F)) F\text{-next}$ )  $\sigma$  ( $\lambda x y. x$ 
iand  $\text{next } y$ )]
  by blast
obtain  $i$  where 1012:  $i < \text{length } F\text{-now} \wedge (\sigma \models F\text{-now} ! i$  iand  $\text{next map } (\lambda x. x \text{ schop } \omega F) F\text{-next}$ 
 $! i)$ 
  using 1011 by blast
have 1013: ( $\sigma \models F\text{-now} ! i$ )
  using 1012 by auto
have 1014: ( $(\text{ndropn } 1 \sigma) \models F\text{-next} ! i \text{ schop } (\omega F)$ )
  using 1012 assms(4) by fastforce
have 1015: ( $(\text{ndropn } 1 \sigma) \models F\text{-next} ! i \text{ schop } (\omega F) \longleftrightarrow$ 
 $(\exists ia. \text{enat } ia \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge$ 
 $(\text{ntaken } ia (\text{ndropn } (\text{Suc } 0) \sigma) \models F\text{-next} ! i) \wedge$ 
 $(\exists l. \neg \text{nfinite } l \wedge$ 
 $\text{nnth } l 0 = 0 \wedge \text{nidx } l \wedge$ 
 $(\forall i. \text{enat } (\text{nnth } l i) \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia) \wedge$ 
 $(\forall i. \text{nsubn } (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0) \sigma)) (\text{nnth } l i) (\text{nnth } l (\text{Suc } i)) \models F)))$ 
  by simp
obtain  $ia$  where 1016:  $\text{enat } ia \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge$ 
 $(\text{ntaken } ia (\text{ndropn } (\text{Suc } 0) \sigma) \models F\text{-next} ! i) \wedge$ 
 $(\exists l. \neg \text{nfinite } l \wedge$ 
 $\text{nnth } l 0 = 0 \wedge \text{nidx } l \wedge$ 
 $(\forall i. \text{enat } (\text{nnth } l i) \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia) \wedge$ 
 $(\forall i. \text{nsubn } (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0) \sigma)) (\text{nnth } l i) (\text{nnth } l (\text{Suc } i)) \models F))$ 
  using 1014 1015 by blast
obtain  $l$  where 1017:  $\neg \text{nfinite } l \wedge$ 
 $\text{nnth } l 0 = 0 \wedge \text{nidx } l \wedge$ 
 $(\forall i. \text{enat } (\text{nnth } l i) \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia) \wedge$ 
 $(\forall i. \text{nsubn } (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0) \sigma)) (\text{nnth } l i) (\text{nnth } l (\text{Suc } i)) \models F)$ 
  using 1016 by blast
have 1018:  $\neg \text{nfinite } \sigma$ 
  by (metis 1016 infinite-nidx-imp-infinite-interval ndropn-nlength nfinite-ndropn-a)
have 1022:  $\text{nnth } (N\text{Cons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) 0 = 0$ 
  by simp
have 1023:  $0 < \text{nnth } (N\text{Cons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) 1$ 
  using zero-enat-def by force
have 1024:  $\neg \text{nfinite } (N\text{Cons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l))$ 
  by (simp add: 1017)
have 1025:  $\text{nidx } (N\text{Cons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l))$ 
  using 1017 nidx-Ncons-shift-alt by blast
have 1026:  $(\forall i. (\text{nnth } (N\text{Cons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) i) \leq \text{nlength } \sigma)$ 
  by (meson 1018 enat-ile nle-le nlength-eq-enat-nfiniteD)
have 1028:  $(\text{nsubn } \sigma (\text{nnth } (N\text{Cons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) 0) (\text{nnth } (N\text{Cons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) (\text{Suc } 0))) =$ 
 $(\text{nsubn } \sigma 0 (\text{Suc } ia))$ 
  using 1017 zero-enat-def by fastforce
have 1029:  $((\text{nsubn } \sigma 0 (\text{Suc } ia)) \models (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) F\text{-now } F\text{-next}))) \longleftrightarrow$ 
 $(\exists i < \text{length } F\text{-now}. \text{nsubn } \sigma 0 (\text{Suc } ia) \models F\text{-now} ! i \text{ iand } \text{next } F\text{-next} ! i)$ 
  using big-ior-map2-defs[of  $F\text{-now } F\text{-next}$  ( $\text{nsubn } \sigma 0 (\text{Suc } ia)$ ) ( $\lambda x y. x \text{ iand } (\text{next } y)$ )]

```

```

    assms by blast
have 1030: nsubn  $\sigma$  0 (Suc ia)  $\models$  F-now ! i
  using 1016
  by (metis 1012 1013 assms(3) ndropn-0 nsubn-def1 ntaken-nfirst state-pitl-defs state-pitl-list-defs)
have 1031: nsubn  $\sigma$  0 (Suc ia)  $\models$  next F-next ! i
  using 1016 ndropn-nsubn[of Suc ia  $\sigma$  Suc 0 0]
  by (metis 1022 1023 1026 1028 One-nat-def add commute diff-zero enat-min-eq enat-ord-simps(1)
    ileI1 leD ndropn-0 nsubn-def1 nsubn-nlength-gr-one ntaken-ndropn-swap one-eSuc one-enat-def
    plus-1-eq-Suc semantics-pitl.simps(4) zero-less-iff-neq-zero)
have 1034: (nsubn  $\sigma$  (nnth (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) 0) (nnth (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) (Suc 0)))  $\models$ 
  (( F-e iand empty ) ior (big-ior (map2 ( $\lambda x y. x \text{ iand } (next y)$ ) F-now F-next))))
  using 1012 1028 1029 1030 1031 by auto
have 1035:  $\bigwedge j. j < nlength\ l \implies$ 
  (nnth ( (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) j) = nnth l j + Suc ia
  by simp
have 1036:  $\bigwedge j. j < nlength\ l \implies$ 
  (nnth ( (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) (Suc j)) = nnth l (Suc j) + Suc ia
  using Suc-ile-eq by force
have 1037:  $\bigwedge j. j < nlength\ l \implies$ 
  nnth l j + Suc ia < nnth l (Suc j) + Suc ia
  by (metis 1017 add-strict-right-mono eSuc-enat ileI1 nidx-expand)
have 1038:  $\bigwedge j. j < nlength\ l \implies$ 
  enat ( nnth l (Suc j) + Suc ia)  $\leq$  nlength  $\sigma$ 
  using 1017 nidx-expand by simp
  (meson 1018 enat-ile nle-le nlength-eq-enat-nfiniteD)
have 1039:  $\bigwedge j. j < nlength\ l \implies$ 
  Suc 0 + (nnth l j + Suc ia)  $\leq$  nnth l (Suc j) + Suc ia
  using 1037 by fastforce
have 1040:  $\bigwedge j. j < nlength\ l \implies$ 
  ndropn (Suc 0) (nsubn  $\sigma$  (nnth l j + Suc ia) (nnth l (Suc j) + Suc ia)) =
  nsubn  $\sigma$  (Suc 0 + (nnth l j + Suc ia)) (nnth l (Suc j) + Suc ia)
  using ndropn-nsubn[of -  $\sigma$  Suc 0 ] using 1038 1039 by presburger
have 1041:  $\bigwedge j. j < nlength\ l \implies$ 
  (nsubn (ndropn ia (ndropn (Suc 0)  $\sigma$ )) (nnth l j) (nnth l (Suc j))) =
  nsubn  $\sigma$  ((nnth l j) + Suc ia) ((nnth l (Suc j)) + Suc ia)
  using nsubn-ndropn[of -  $\sigma$  Suc ia ]
  by (metis 1037 One-nat-def add-less-cancel-right ndropn-ndropn plus-1-eq-Suc)
have 1280: (nsubn  $\sigma$  (nnth (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) 0) (nnth (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) (Suc 0)))  $\models$  F)
  using 1034 assms(1) by simp
have 1281: ( $\forall i. \text{enat } i < nlength\ ( (nmap (\lambda x. x + \text{Suc ia}) l) ) \longrightarrow$ 
  (nsubn  $\sigma$  (nnth ( (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) i) (nnth ( (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) (Suc i))  $\models$ 
F))
  using 1017 1036 1041
  by (metis 1035 nlength-nmap)
have 1290: ( $\forall i. \text{enat } i < nlength\ (NCons\ 0\ (nmap\ (\lambda x. x + \text{Suc ia})\ l)) \longrightarrow$ 
  (nsubn  $\sigma$  (nnth (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) i) (nnth (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) (Suc i))  $\models$  F))  $\longleftrightarrow$ 
  (nsubn  $\sigma$  (nnth (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) 0) (nnth (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) (Suc i))  $\models$  F))

```


$ia) l)) (Suc\ 0)) \models F) \wedge$
 $(\forall i. enat\ i < nlength\ ((nmap\ (\lambda x. x + Suc\ ia)\ l)) \longrightarrow$
 $(nsubn\ \sigma\ (nnth\ ((nmap\ (\lambda x. x + Suc\ ia)\ l))\ i)\ (nnth\ ((nmap\ (\lambda x. x + Suc\ ia)\ l))\ (Suc\ i)) \models$
 $F))$
using *forall-ncons-split-alt*[*of* $(nmap\ (\lambda x. x + Suc\ ia)\ l)$
 $\lambda\ x\ y. (nsubn\ \sigma\ x\ y \models F)$]
by (*simp add: 1017*)
have 1300: $(\forall i. enat\ i < nlength\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l)) \longrightarrow$
 $(nsubn\ \sigma\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))\ i)\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))$
 $(Suc\ i)) \models F))$
using 1280 1281 1290 **by** *blast*
have 1340: $nlength\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l)) = \infty$
by (*metis 1024 enat2-cases nlength-eq-enat-nfiniteD*)
have 1350: $(\forall i.$
 $(nsubn\ \sigma\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))\ i)\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))$
 $(Suc\ i)) \models F))$
using 1300 1340 *enat-ord-code(4)* **by** *presburger*
have 1400: $(\exists (l:: nat\ nellist).$
 $\neg nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nid\ x\ l \wedge$
 $(\forall i. (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
 $(\forall i. ((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models F))$
 $)$
by (*meson 1017 1022 1025 1026 1350 nfinite-NCons nfinite-nmap*)
show ?thesis **using** 1400 **by** *simp*
qed
show ?thesis
using 1000 2 **by** *blast*
qed

lemma *GNF-atom-base:*

$\vdash (atom\ p) \text{ieqv}\ (GNF\ (atom\ p))$
unfolding *GNF-def GNF-cons-def big-ior-def*
by (*auto*)
 $(metis\ add-0\ ileI1\ linorder-not-less\ not-less-zero\ order-neq-le-trans\ plus-1-eSuc(2))$

lemma *GNF-ifalse-base:*

$\vdash ifalse\ \text{ieqv}\ (GNF\ ifalse)$
unfolding *GNF-def GNF-cons-def big-ior-def*
by *auto*

lemma *GNF-next-step:*

$\vdash (next\ f) \text{ieqv}\ (GNF\ (next\ f))$
unfolding *GNF-def GNF-cons-def big-ior-def*
by (*auto*)

lemma *exists-GNF-nondet:*

```

∃ ae a A. (state-pitl ae) ∧ (state-pitl-list a) ∧ length a = length A ∧
  (⊢ f ieqv (ae iand empty) ior (big-ior (map2 (λ x y. x iand next y) a A)))
proof (induction f)
case false-d
then show ?case
  proof –
    have f1: ⊢ ifalse ieqv (GNF ifalse)
      using GNF-ifalse-base by auto
    have f2: state-pitl-list (GNF-state ifalse)
      unfolding state-pitl-list-def by (simp add: inot-d-def ittrue-d-def)
    have f3: length (GNF-state ifalse) = length (GNF-next ifalse)
      by simp
    show ?thesis
      by (metis f1 f2 f3 GNF-cons-def GNF-def GNF-empty.simps(1) state-pitl.simps(1))
    qed
next
case (atom-d x)
then show ?case
  proof –
    have a1: ⊢(atom x) ieqv (GNF (atom x))
      using GNF-atom-base by blast
    have a2: state-pitl-list (GNF-state (atom x))
      unfolding state-pitl-list-def by simp
    have a3: length (GNF-state (atom x)) = length (GNF-next (atom x))
      by simp
    show ?thesis
      by (metis GNF-cons-def GNF-def GNF-empty.simps(2) a1 a2 a3 state-pitl.simps(2))
    qed
next
case (iimp-d f1 f2)
then show ?case
  proof –
    obtain ae1 a1 A1 where i1: state-pitl ae1 ∧ state-pitl-list a1 ∧ length a1 = length A1 ∧
      (⊢ f1 ieqv ae1 iand empty ior big-ior (map2 (λ x y. x iand next y) a1 A1))
      using iimp-d.IH(1) by blast
    obtain ae2 a2 A2 where i2: state-pitl ae2 ∧ state-pitl-list a2 ∧ length a2 = length A2 ∧
      (⊢ f2 ieqv ae2 iand empty ior big-ior (map2 (λ x y. x iand next y) a2 A2))
      using iimp-d.IH(2) by blast
    have i3: ⊢ (f1 iimp f2) ieqv ( ( (inot ae1) ior ae2) iand empty) ior
      (big-ior
        (map2 (λ x y. x iand (next y))
          ((add-to-now a1) @ a2)
          ((map (λ x. (inot x)) (add-to-next A1)) @ A2) ))
      using GNF-iimp-step[of f1 ae1 a1 A1 f2 ae2 a2 A2] i1 i2 itl-ieq by blast
    have i4: length ((add-to-now a1) @ a2) = length ((map (λ x. (inot x)) (add-to-next A1)) @ A2)
      by (simp add: add-to-next-def add-to-now-def i1 i2 length-add-to-now-next-gen)
    have i5: state-pitl-list ((add-to-now a1) @ a2)
      by (simp add: add-to-now-def i1 i2 state-pitl-foldr-add-to-now-single-inv-a state-pitl-list-append)
    show ?thesis
      by (metis i1 i2 i3 i4 i5 inot-d-def ior-d-def state-pitl.simps(1) state-pitl.simps(3))
  
```

```

qed
next
case (next-d f)
then show ?case
proof -
  obtain ae a A where n1: state-pitl ae  $\wedge$  state-pitl-list a  $\wedge$  length a = length A  $\wedge$ 
    ( $\vdash$  f ieqv ae iand empty ior big-ior (map2 ( $\lambda$  x y. x iand next y) a A) )
    using next-d.IH by blast
  have n2:  $\vdash$  (next f) ieqv (GNF (next f))
    using GNF-next-step by blast
  have n3: state-pitl-list (GNF-state (next f))
    by (simp add: inot-d-def itrue-d-def state-pitl-list-defs)
  have n4: length (GNF-state (next f)) = length (GNF-next (next f))
    by simp
  show ?thesis
    by (metis GNF-cons-def GNF-def GNF-empty.simps(4) n2 n3 n4 state-pitl.simps(1))
qed
next
case (chop-d f1 f2)
then show ?case
proof -
  obtain ae1 a1 A1 where c1: state-pitl ae1  $\wedge$  state-pitl-list a1  $\wedge$  length a1 = length A1  $\wedge$ 
    ( $\vdash$  f1 ieqv ae1 iand empty ior big-ior (map2 ( $\lambda$  x y. x iand next y) a1 A1) )
    using chop-d.IH(1) by blast
  obtain ae2 a2 A2 where c2: state-pitl ae2  $\wedge$  state-pitl-list a2  $\wedge$  length a2 = length A2  $\wedge$ 
    ( $\vdash$  f2 ieqv ae2 iand empty ior big-ior (map2 ( $\lambda$  x y. x iand next y) a2 A2) )
    using chop-d.IH(2) by blast
  have c3:  $\vdash$  f1 schop f2 ieqv (ae1 iand ae2) iand empty ior
    (big-ior (map2 ( $\lambda$  x y. x iand next y)
      (a1 @ (map ( $\lambda$  x. x iand ae1) a2))
      ((map ( $\lambda$  x. x schop f2) A1) @ A2)))
    using GNF-schop-step[of f1 ae1 a1 A1 f2 ae2 a2 A2] c1 c2 itl-ieq by blast
  have c4: length (a1 @ (map ( $\lambda$  x. x iand ae1) a2)) = length ((map ( $\lambda$  x. x schop f2) A1) @ A2)
    by (simp add: c1 c2)
  have c5: state-pitl-list (a1 @ (map ( $\lambda$  x. x iand ae1) a2))
    by (simp add: c1 c2 state-pitl-list-append state-pitl-list-map-alt)
  have c6: state-pitl (ae1 iand ae2)
    by (simp add: c1 c2 iand-d-def inot-d-def ior-d-def)
  show ?thesis using c1 c2 c3 c4 c5 c6 by blast
qed
next
case (schopstar-d f)
then show ?case
proof -
  obtain ae a A where s1: state-pitl ae  $\wedge$  state-pitl-list a  $\wedge$  length a = length A  $\wedge$ 
    ( $\vdash$  f ieqv ae iand empty ior big-ior (map2 ( $\lambda$  x y. x iand next y) a A) )
    using chopstar-d.IH by blast
  have s2:  $\vdash$  chopstar f ieqv (itrue iand empty) ior
    (big-ior (map2 ( $\lambda$  x y. x iand next y) a (map ( $\lambda$  x. x schop (schopstar f)) A)))
    using GNF-schopstar-step[of f ae a A] s1 using itl-ieq by blast

```

```

have s3: length a = length (map (λx. x schop (schopstar f)) A)
  by (simp add: s1)
have s4: state-pitl itrue
  by (simp add: inot-d-def itrue-d-def)
show ?thesis
  by (metis s1 s2 s3 s4)
qed
next
case (omega-d f)
then show ?case
  proof -
    obtain ae a A where o1: state-pitl ae ∧ state-pitl-list a ∧ length a = length A ∧
      (⊢ f ieqv ae iand empty ior big-ior (map2 (λx y. x iand next y) a A) )
    using omega-d.IH by blast
    have o2: ⊢ omega f ieqv (ifalse iand empty) ior
      (big-ior (map2 (λ x y. x iand next y) a (map (λx. x schop (omega f)) A)))
    using GNF-omega-step[of f ae a A] o1 using itl-ieq by blast
    have o3: length a = length (map (λx. x schop (omega f)) A)
      by (simp add: o1)
    show ?thesis
      using o1 o2 o3 state-pitl.simps(1) by blast
  qed
qed

```

```

lemma state-pitl-GNF-empty:
  state-pitl (GNF-empty f)
proof (induction f)
case false-d
then show ?case by simp
next
case (atom-d x)
then show ?case by simp
next
case (iimp-d f1 f2)
then show ?case apply simp unfolding inot-d-def ior-d-def by simp
next
case (next-d f)
then show ?case by simp
next
case (chop-d f1 f2)
then show ?case apply simp unfolding iand-d-def inot-d-def ior-d-def by simp
next
case (schopstar-d f)
then show ?case apply simp unfolding itrue-d-def inot-d-def by simp
next
case (omega-d f)
then show ?case by simp
qed

```

```

lemma state-pitl-list-GNF-state:

```

```

  state-pitl-list (GNF-state f)
proof (induction f)
case false-d
then show ?case unfolding state-pitl-list-def apply simp unfolding itrue-d-def inot-d-def by simp
next
case (atom-d x)
then show ?case unfolding state-pitl-list-def by simp
next
case (iimp-d f1 f2)
then show ?case unfolding state-pitl-list-def apply simp
by (metis add-to-now-def state-pitl-foldr-add-to-now-single-inv-a state-pitl-list-def)
next
case (next-d f)
then show ?case unfolding state-pitl-list-def apply simp unfolding itrue-d-def inot-d-def by simp
next
case (chop-d f1 f2)
then show ?case unfolding state-pitl-list-def apply simp
by (metis (full-types) state-pitl-GNF-empty state-pitl-list-def state-pitl-list-map-alt)
next
case (schopstar-d f)
then show ?case unfolding state-pitl-list-def by simp
next
case (omega-d f)
then show ?case unfolding state-pitl-list-def by simp
qed

```

lemma *length-GNF-state-eq-length-GNF-next:*

```

  length (GNF-state f) = length (GNF-next f)
proof (induction f)
case false-d
then show ?case by simp
next
case (atom-d x)
then show ?case by simp
next
case (iimp-d f1 f2)
then show ?case
by (simp add: add-to-next-def add-to-now-def length-add-to-now-next-gen)
next
case (next-d f)
then show ?case by simp
next
case (chop-d f1 f2)
then show ?case by simp
next
case (schopstar-d f)
then show ?case by simp
next
case (omega-d f)
then show ?case by simp

```

qed

lemma *ieqv-GNF*:

$\vdash f \text{ieqv GNF } f$

proof (*induction f*)

case *false-d*

then show *?case*

using *GNF-ifalse-base* **by** *auto*

next

case (*atom-d x*)

then show *?case* **using** *GNF-atom-base* **by** *blast*

next

case (*iimp-d f1 f2*)

then show *?case*

proof –

have *i1*: $\vdash f1 \text{ieqv GNF-empty } f1 \text{ iand empty ior big-ior (map2 } (\lambda x y. x \text{ iand next } y) \text{ (GNF-state } f1) \text{ (GNF-next } f1))$

using *iimp-d* **by** (*simp add: GNF-cons-def GNF-def*)

have *i2*: $\vdash f2 \text{ieqv GNF-empty } f2 \text{ iand empty ior big-ior (map2 } (\lambda x y. x \text{ iand next } y) \text{ (GNF-state } f2) \text{ (GNF-next } f2))$

using *iimp-d* **by** (*simp add: GNF-cons-def GNF-def*)

have *i3*: *state-pitl* (*GNF-empty f1*)

by (*simp add: state-pitl-GNF-empty*)

have *i4*: *state-pitl-list* (*GNF-state f1*)

by (*simp add: state-pitl-list-GNF-state*)

have *i5*: *state-pitl* (*GNF-empty f2*)

by (*simp add: state-pitl-GNF-empty*)

have *i6*: *state-pitl-list* (*GNF-state f2*)

by (*simp add: state-pitl-list-GNF-state*)

have *i7*: *length* (*GNF-state f1*) = *length* (*GNF-next f1*)

by (*simp add: length-GNF-state-eq-length-GNF-next*)

have *i8*: *length* (*GNF-state f2*) = *length* (*GNF-next f2*)

by (*simp add: length-GNF-state-eq-length-GNF-next*)

show *?thesis* **using**

GNF-iimp-step[of f1 GNF-empty f1 (GNF-state f1) (GNF-next f1) f2 GNF-empty f2 (GNF-state f2) (GNF-next f2)]

using *GNF-cons-def GNF-def i1 i2 i3 i4 i5 i6 i7 i8*

by (*metis GNF-empty.simps(3) GNF-next.simps(3) GNF-state.simps(3) itl-ieq*)

qed

next

case (*next-d f*)

then show *?case* **using** *GNF-next-step* **by** *blast*

next

case (*chop-d f1 f2*)

then show *?case*

proof –

have *c1*: $\vdash f1 \text{ieqv GNF-empty } f1 \text{ iand empty ior big-ior (map2 } (\lambda x y. x \text{ iand next } y) \text{ (GNF-state } f1) \text{ (GNF-next } f1))$

using *chop-d* **by** (*simp add: GNF-cons-def GNF-def*)

have *c2*: $\vdash f2 \text{ieqv GNF-empty } f2 \text{ iand empty ior big-ior (map2 } (\lambda x y. x \text{ iand next } y) \text{ (GNF-state } f2) \text{ (GNF-next } f2))$

```

(GNF-next f2))
  using chop-d by (simp add: GNF-cons-def GNF-def)
  have c3: state-pitl (GNF-empty f1)
    by (simp add: state-pitl-GNF-empty)
  have c4: state-pitl-list (GNF-state f1)
    by (simp add: state-pitl-list-GNF-state)
  have c5: state-pitl (GNF-empty f2)
    by (simp add: state-pitl-GNF-empty)
  have c6: state-pitl-list (GNF-state f2)
    by (simp add: state-pitl-list-GNF-state)
  have c7: length (GNF-state f1) = length (GNF-next f1)
    by (simp add: length-GNF-state-eq-length-GNF-next)
  have c8: length (GNF-state f2) = length (GNF-next f2)
    by (simp add: length-GNF-state-eq-length-GNF-next)
  show ?thesis using
    GNF-schop-step[of f1 GNF-empty f1 (GNF-state f1) (GNF-next f1)
      f2 GNF-empty f2 (GNF-state f2) (GNF-next f2)]
    unfolding GNF-def GNF-cons-def using c1 c2 c3 c4 c5 c6 c7 c8 by fastforce
qed
next
case (schopstar-d f)
then show ?case
  proof –
    have s1:  $\vdash f \text{ ieqv } \text{GNF-empty } f \text{ iand empty ior big-ior } (\text{map2 } (\lambda x y. x \text{ iand next } y) (\text{GNF-state } f) (\text{GNF-next } f))$ 
      using schopstar-d by (simp add: GNF-cons-def GNF-def)
    have s2: state-pitl (GNF-empty f)
      by (simp add: state-pitl-GNF-empty)
    have s3: state-pitl-list (GNF-state f)
      by (simp add: state-pitl-list-GNF-state)
    have s4: length (GNF-state f) = length (GNF-next f)
      by (simp add: length-GNF-state-eq-length-GNF-next)
    show ?thesis using
      GNF-schopstar-step[of f GNF-empty f (GNF-state f) (GNF-next f)]
      unfolding GNF-def GNF-cons-def using s1 s2 s3 s4 by fastforce
  qed
next
case (omega-d f)
then show ?case
  proof –
    have o1:  $\vdash f \text{ ieqv } \text{GNF-empty } f \text{ iand empty ior big-ior } (\text{map2 } (\lambda x y. x \text{ iand next } y) (\text{GNF-state } f) (\text{GNF-next } f))$ 
      using omega-d by (simp add: GNF-cons-def GNF-def)
    have o2: state-pitl (GNF-empty f)
      by (simp add: state-pitl-GNF-empty)
    have o3: state-pitl-list (GNF-state f)
      by (simp add: state-pitl-list-GNF-state)
    have o4: length (GNF-state f) = length (GNF-next f)
      by (simp add: length-GNF-state-eq-length-GNF-next)
    show ?thesis using

```

```

GNF-omega-step[of f GNF-empty f (GNF-state f) (GNF-next f)]
unfolding GNF-def GNF-cons-def using o1 o2 o3 o4 by fastforce
qed
qed

```

lemma *exist-GNF-det*:

```

GNF-prop f
proof –
have 1:  $\exists$  ae a A. (state-pitl ae)  $\wedge$  (state-pitl-list a)  $\wedge$  length a = length A  $\wedge$ 
  ( $\vdash$  f ieqv (ae iand empty) ior (big-ior (map2 ( $\lambda$  x y. x iand next y) a A)))
using exists-GNF-nondet by blast
obtain ae a A where 2: (state-pitl ae)  $\wedge$  (state-pitl-list a)  $\wedge$  length a = length A  $\wedge$ 
  ( $\vdash$  f ieqv (ae iand empty) ior (big-ior (map2 ( $\lambda$  x y. x iand next y) a A)))
using 1 by blast
have 3:  $\vdash$  (big-ior (map2 ( $\lambda$  x y. x iand next y) a A)) ieqv
  big-ior (map2 ( $\lambda$  x y. x iand (next y))
    (foldr (add-to-now-single) a [itrue])
    (foldr (add-to-next-single) A [ifalse]))
using 2 det-big-ior-ieqv-nondet-big-ior-a by auto
have 4:  $\vdash$  f ieqv (ae iand empty) ior
  big-ior ( map2 ( $\lambda$  x y. x iand (next y))
    (foldr (add-to-now-single) a [itrue])
    (foldr (add-to-next-single) A [ifalse]))
using 2 3 by fastforce
have 5: state-pitl-list (foldr (add-to-now-single) a [itrue])
by (simp add: 2 state-pitl-foldr-add-to-now-single-inv-a)
have 6: length (foldr (add-to-now-single) a [itrue]) =
  length (foldr (add-to-next-single) A [ifalse])
by (simp add: 2 length-add-to-now-next-gen)
have 7: full-system (foldr (add-to-now-single) a [itrue])
using big-ior-foldr-add-to-now-single-inv-a full-system-def mutex-foldr-add-to-now-single-inv-a by blast
show ?thesis unfolding GNF-prop-def GNF-cons-def
using 2 4 5 6 7 by blast
qed

```

lemma *state-pitl-list-GNF-det-state*:

```

state-pitl-list (foldr (add-to-now-single) (GNF-state f) [itrue])
by (simp add: state-pitl-foldr-add-to-now-single-inv-a state-pitl-list-GNF-state)

```

lemma *full-system-GNF-det-state*:

```

full-system (foldr (add-to-now-single) (GNF-state f) [itrue])
using big-ior-foldr-add-to-now-single-inv-a full-system-def
mutex-foldr-add-to-now-single-inv-a by blast

```

lemma *ieqv-GNF-det*:

```

 $\vdash$  f ieqv GNF-det f

```



```

using ieqv-GNF[of f]
unfolding GNF-def GNF-det-def GNF-cons-def
using det-big-ior-ieqv-nondet-big-ior-a length-GNF-state-eq-length-GNF-next by auto

end

```

Chapter 4

Deep embedding of QPITL

```
theory QPITL-Deep imports
  NELList-Extras
```

```
begin
```

This theory is a deep embedding of Quantified Propositional Interval Temporal Logic (QPITL). It provides the syntax and semantics.

4.1 Syntax

4.1.1 Primitive formulae

```
datatype qpitl =
  false-d          (ifalse)
| atom-d nat       (($-) )
| iimp-d qpitl qpitl (( - iimp - ) [26,25] 25)
| next-d qpitl      ((next - ) [88] 87)
| chop-d qpitl qpitl (( - chop - ) [84,84] 83)
| schopstar-d qpitl ((s chopstar - ) [85] 85)
| omega-d qpitl      ((omega - ) [85] 85)
| exists-d nat qpitl ( Ex -. - 10)
```

4.1.2 state formula

```
fun state-qpitl :: qpitl ⇒ bool
where
  state-qpitl (ifalse)      = True
| state-qpitl ($n)          = True
| state-qpitl (f iimp g)    = ((state-qpitl f) ∧ (state-qpitl g))
| state-qpitl (next f)      = False
| state-qpitl (f chop g)    = False
| state-qpitl (s chopstar f) = False
| state-qpitl (omega f)     = False
| state-qpitl (Ex n. f)     = state-qpitl f
```

4.1.3 Derived Boolean Operators

definition *inot-d* ((*inot* -) [90] 90)

where

$inot\ f \equiv f\ iimp\ ifalse$

definition *itrue-d* (*itrue*)

where

$itrue \equiv inot\ ifalse$

definition *ior-d* ((- *ior* -) [31,30] 30)

where

$f\ ior\ g \equiv (inot\ f)\ iimp\ g$

definition *iand-d* ((- *iand* -) [36,35] 35)

where

$f\ iand\ g \equiv inot\ (inot\ f\ ior\ inot\ g)$

definition *ieqv-d* ((- *ieqv* -) [21,20] 20)

where

$f\ ieqv\ g \equiv ((f\ iimp\ g)\ iand\ (g\ iimp\ f))$

4.1.4 more, empty, skip and wnext

definition *more-d* (*more*)

where

$more \equiv next\ itrue$

definition *empty-d* (*empty*)

where

$empty \equiv inot\ more$

definition *skip-d* (*skip*)

where

$skip \equiv next\ empty$

definition *wnext-d* ((*wnext* -) [88] 87)

where

$wnext\ f \equiv inot\ (next\ (inot\ f))$

4.1.5 ifinite and inf

definition *finite-d* (*ifinite*)

where

$ifinite \equiv itrue\ schop\ empty$

definition *inf-d* (*inf*)

where

$inf \equiv inot\ ifinite$

4.1.6 weak chop

definition *chop-d* $((- ; -) [84,84] 83)$
where $f ; g \equiv (f \text{ chop } g) \text{ ior } (f \text{ iand } \text{inf})$

4.1.7 Box and Diamond Operators

definition *sometimes-d* $((\text{diamond } -) [88] 87)$
where
 $\text{diamond } f \equiv \text{itrue chop } f$

definition *always-d* $((\text{box } -) [88] 87)$
where
 $\text{box } f \equiv \text{inot } (\text{diamond } (\text{inot } f))$

definition *di-d* $((\text{di } -) [88] 87)$
where
 $\text{di } f \equiv f ; \text{itrue}$

definition *bi-d* $((\text{bi } -) [88] 87)$
where
 $\text{bi } f \equiv \text{inot } (\text{di } (\text{inot } f))$

definition *df-d* $((\text{df } -) [88] 87)$
where
 $\text{df } f \equiv f \text{ chop } \text{itrue}$

definition *bf-d* $((\text{bf } -) [88] 87)$
where
 $\text{bf } f \equiv \text{inot } (\text{df } (\text{inot } f))$

definition *da-d* $((\text{da } -) [88] 87)$
where
 $\text{da } f \equiv \text{itrue chop } (f ; \text{itrue})$

definition *ba-d* $((\text{ba } -) [88] 87)$
where
 $\text{ba } f \equiv \text{inot } (\text{da } (\text{inot } f))$

definition *sda-d* $((\text{sda } -) [88] 87)$
where
 $\text{sda } f \equiv \text{itrue chop } (f \text{ chop } \text{itrue})$

definition *sba-d* $((\text{sba } -) [88] 87)$
where
 $\text{sba } f \equiv \text{inot } (\text{sda } (\text{inot } f))$

4.1.8 Initial and Final Operators

definition *init-d* $((\text{init } -) [88] 87)$
where

$init\ f \equiv ((empty\ iand\ f)\ schop\ itrue)$

definition $fin-d\ ((fin\ -)\ [88]\ 87)$

where

$fin\ f \equiv diamond\ (empty\ iand\ f)$

4.1.9 Forall

definition $forall-d\ (Fa\ -. -\ 10)$

where

$Fa\ v.\ f \equiv inot\ (Ex\ v.\ inot\ f)$

4.1.10 Big operations

definition $big-ior :: qpitl\ list \Rightarrow qpitl$

where $big-ior\ F = foldr\ (\lambda\ x\ y.\ x\ ior\ y)\ F\ ifalse$

definition $big-iand :: qpitl\ list \Rightarrow qpitl$

where $big-iand\ F = foldr\ (\lambda\ x\ y.\ x\ iand\ y)\ F\ itrue$

definition $state-qpitl-list :: qpitl\ list \Rightarrow bool$

where $state-qpitl-list\ L = foldr\ (\lambda\ x\ y.\ state-qpitl\ x \wedge y)\ L\ True$

4.2 Semantics

4.2.1 Intervals

type-synonym $interval = nat\ set\ nellist$

4.2.2 Semantics Primitive Operators

definition $similar :: interval \Rightarrow nat \Rightarrow interval \Rightarrow bool\ ((- \sim -))$

where $similar\ \sigma 0\ n\ \sigma 1 = (nlength\ \sigma 0 = nlength\ \sigma 1 \wedge$

$(\forall\ k\ p.\ k \leq nlength\ \sigma 0 \longrightarrow$

$(p \in (nnth\ \sigma 0\ k) \wedge p \neq n) \longleftrightarrow (p \in (nnth\ \sigma 1\ k) \wedge p \neq n))\)$

definition $similar :: interval \Rightarrow nat\ set \Rightarrow interval \Rightarrow bool\ ((- \approx -))$

where $similar\ \sigma 0\ A\ \sigma 1 = (nlength\ \sigma 0 = nlength\ \sigma 1 \wedge$

$(\forall\ k\ p.\ k \leq nlength\ \sigma 0 \longrightarrow p \notin A \longrightarrow (p \in (nnth\ \sigma 0\ k) \longleftrightarrow p \in (nnth\ \sigma 1\ k))))$

primrec $fvars :: qpitl \Rightarrow nat\ set$

where

$fvars\ ifalse = \{\}$

$| fvars\ (\$n) = \{n\}$

$| fvars\ (f\ iimp\ g) = (fvars\ f) \cup (fvars\ g)$

$| fvars\ (next\ f) = fvars\ f$

$| fvars\ (f\ schop\ g) = (fvars\ f) \cup (fvars\ g)$

$| fvars\ (schopstar\ f) = fvars\ f$

$| fvars\ (omega\ f) = fvars\ f$

| $fvars (Ex\ n.\ f) = (fvars\ f) - \{n\}$

primrec $bvars :: qpitl \Rightarrow nat\ set$

where

$bvars\ ifalse = \{\}$
 | $bvars\ (\$n) = \{\}$
 | $bvars\ (f\ iimp\ g) = (bvars\ f) \cup (bvars\ g)$
 | $bvars\ (next\ f) = bvars\ f$
 | $bvars\ (f\ schop\ g) = (bvars\ f) \cup (bvars\ g)$
 | $bvars\ (schopstar\ f) = bvars\ f$
 | $bvars\ (omega\ f) = bvars\ f$
 | $bvars\ (Ex\ n.\ f) = (insert\ n\ (bvars\ f))$

primrec $suform :: qpitl \Rightarrow nat \Rightarrow qpitl \Rightarrow qpitl$

where

$suform\ ifalse\ n\ h = ifalse$
 | $suform\ (\$k)\ n\ h = (if\ k = n\ then\ h\ else\ (\$k))$
 | $suform\ (f\ iimp\ g)\ n\ h = ((suform\ f\ n\ h)\ iimp\ (suform\ g\ n\ h))$
 | $suform\ (next\ f)\ n\ h = (next\ (suform\ f\ n\ h))$
 | $suform\ (f\ schop\ g)\ n\ h = ((suform\ f\ n\ h)\ schop\ (suform\ g\ n\ h))$
 | $suform\ (schopstar\ f)\ n\ h = (schopstar\ (suform\ f\ n\ h))$
 | $suform\ (omega\ f)\ n\ h = (omega\ (suform\ f\ n\ h))$
 | $suform\ (Ex\ k.\ f)\ n\ h = (Ex\ k.\ (suform\ f\ n\ h))$

primrec $rename :: qpitl \Rightarrow nat \Rightarrow nat \Rightarrow qpitl$

where

$rename\ ifalse\ n\ k = ifalse$
 | $rename\ (\$m)\ n\ k = (if\ m = n\ then\ \$k\ else\ \$m)$
 | $rename\ (f\ iimp\ g)\ n\ k = ((rename\ f\ n\ k)\ iimp\ (rename\ g\ n\ k))$
 | $rename\ (next\ f)\ n\ k = (next\ (rename\ f\ n\ k))$
 | $rename\ (f\ schop\ g)\ n\ k = ((rename\ f\ n\ k)\ schop\ (rename\ g\ n\ k))$
 | $rename\ (schopstar\ f)\ n\ k = (schopstar\ (rename\ f\ n\ k))$
 | $rename\ (omega\ f)\ n\ k = (omega\ (rename\ f\ n\ k))$
 | $rename\ (Ex\ m.\ f)\ n\ k = (Ex\ (if\ m = n\ then\ k\ else\ m).\ (rename\ f\ n\ k))$

primrec $max-var :: qpitl \Rightarrow nat \Rightarrow nat$

where

$max-var\ ifalse\ m = m$
 | $max-var\ (\$k)\ m = (if\ m < k\ then\ k\ else\ m)$
 | $max-var\ (f\ iimp\ g)\ m = (if\ (max-var\ f\ m) < (max-var\ g\ m)\ then\ (max-var\ g\ m)\ else\ (max-var\ f\ m))$
 | $max-var\ (next\ f)\ m = max-var\ f\ m$
 | $max-var\ (f\ schop\ g)\ m = (if\ (max-var\ f\ m) < (max-var\ g\ m)\ then\ (max-var\ g\ m)\ else\ (max-var\ f\ m))$
 | $max-var\ (schopstar\ f)\ m = max-var\ f\ m$
 | $max-var\ (omega\ f)\ m = max-var\ f\ m$
 | $max-var\ (Ex\ k.\ f)\ m = (if\ (max-var\ f\ m) < k\ then\ k\ else\ (max-var\ f\ m))$

primcorec $upd :: interval \Rightarrow nat \Rightarrow (bool\ nellist) \Rightarrow interval$

where

```

upd  $\sigma$   $n$   $l$  = (case  $\sigma$  of (NNil  $b$ )  $\Rightarrow$ 
  (NNil (if (nfirst  $l$ ) then insert  $n$   $b$  else  $b - \{n\}$ )) |
  (NCons  $x$   $\sigma'$ )  $\Rightarrow$ 
    (case  $l$  of (NNil  $a$ )  $\Rightarrow$ 
      (NNil (if  $a$  then insert  $n$   $x$  else  $x - \{n\}$ )) |
      (NCons  $y$   $l'$ )  $\Rightarrow$ 
        (NCons (if  $y$  then (insert  $n$   $x$ ) else ( $x - \{n\}$ ))
          (upd  $\sigma'$   $n$   $l'$ ))))))

```

simps-of-case *upd-simps* [*simp*, *code*, *nitpick-simp*]: *upd.code*

```

fun semantics-qpitl :: [interval, qpitl]  $\Rightarrow$  bool ((-  $\models$  -) [80,10] 10)
where
  ( $\sigma \models$  ifalse) = False
  | ( $\sigma \models$  $ $p$ ) = ( $p \in$  (nfirst  $\sigma$ ))
  | ( $\sigma \models$  f iimp  $g$ ) = (( $\sigma \models$   $f$ )  $\longrightarrow$  ( $\sigma \models$   $g$ ))
  | ( $\sigma \models$  (next  $f$ )) = ( $1 \leq$  nlength  $\sigma \wedge$  ((ndropn 1  $\sigma$ )  $\models$   $f$ ))
  | ( $\sigma \models$  f schop  $g$ ) =
    ( $\exists i.$  (( $0 \leq i$ )  $\wedge$  ( $i \leq$  (nlength  $\sigma$ ))  $\wedge$  ((ntaken  $i$   $\sigma$ )  $\models$   $f$ )  $\wedge$  ((ndropn  $i$   $\sigma$ )  $\models$   $g$ )) )
  | ( $\sigma \models$  (schopstar  $f$ )) =
    ( $\exists (l:: \text{nat nellist}).$ 
      (nnth  $l$  0) = 0  $\wedge$  nfinite  $l \wedge$  nid $x$   $l \wedge$ 
      (enat (nlast  $l$ )) = (nlength  $\sigma$ )  $\wedge$  nfinite  $\sigma \wedge$ 
      ( $\forall (i::\text{nat}). ( \text{enat } i < \text{nlength } l ) \longrightarrow$ 
        ((nsubn  $\sigma$  (nnth  $l$   $i$ ) (nnth  $l$  (Suc  $i$ )) )  $\models$   $f$  ) )
    )
  | ( $\sigma \models$  (omega  $f$ )) =
    ( $\exists (l:: \text{nat nellist}).$ 
       $\neg$ nfinite  $l \wedge$  (nnth  $l$  0) = 0  $\wedge$  nid $x$   $l \wedge$ 
      ( $\forall i.$  (nnth  $l$   $i$ )  $\leq$  nlength  $\sigma$ )  $\wedge$ 
      ( $\forall i.$  ( (nsubn  $\sigma$  (nnth  $l$   $i$ ) (nnth  $l$  (Suc  $i$ )))  $\models$   $f$  ) )
    )
  | ( $\sigma \models$  (Ex  $n.$   $f$ )) = ( $\exists l.$  nlength  $\sigma$  = nlength  $l \wedge$  ((upd  $\sigma$   $n$   $l$ )  $\models$   $f$ ))

```

lemma similar-defs [*simp*]:

```

  similar  $\sigma 0$   $n$   $\sigma 1 \longleftrightarrow$  (nlength  $\sigma 0$  = nlength  $\sigma 1 \wedge$ 
    ( $\forall k$   $p.$   $k \leq$  nlength  $\sigma 0 \longrightarrow (p \in$  (nnth  $\sigma 0$   $k$ )  $\wedge$   $p \neq n$ )  $\longleftrightarrow$  ( $p \in$  (nnth  $\sigma 1$   $k$ )  $\wedge$   $p \neq$ 
     $n$ ))) )

```

unfolding similar-def **by** auto

lemma similar-defs-alt:

```

  (nlength  $\sigma 0$  = nlength  $\sigma 1 \wedge$  ( $\forall k.$   $k \leq$  nlength  $\sigma 0 \longrightarrow$  (nnth  $\sigma 0$   $k$ ) - { $n$ } = (nnth  $\sigma 1$   $k$ ) - { $n$ }))  $\longleftrightarrow$ 
  similar  $\sigma 0$   $n$   $\sigma 1$ 

```

unfolding similar-def **by** auto

lemma *similar-defs* [simp]:

$similar\ \sigma\ 0\ A\ \sigma\ 1 \iff (nlength\ \sigma\ 0 = nlength\ \sigma\ 1 \wedge$
 $(\forall\ k\ p.\ k \leq nlength\ \sigma\ 0 \longrightarrow p \notin A \longrightarrow (p \in (nnth\ \sigma\ 0\ k) \iff p \in (nnth\ \sigma\ 1\ k))))$

unfolding *similar-def* **by** *auto*

lemma *similar-defs-alt*:

$similar\ \sigma\ 0\ A\ \sigma\ 1 \iff (nlength\ \sigma\ 0 = nlength\ \sigma\ 1 \wedge (\forall\ k.\ k \leq nlength\ \sigma\ 0 \longrightarrow (nnth\ \sigma\ 0\ k) - A = (nnth\ \sigma\ 1\ k) - A))$

unfolding *similar-def* **by** *auto*

lemma *similar-similar*:

$similar\ \sigma\ 0\ n\ \sigma\ 1 = similar\ \sigma\ 0\ \{n\}\ \sigma\ 1$

using *similar-defs-alt* *similar-defs-alt* **by** *auto*

lemma *similar-mono*:

assumes $A \subseteq B$

$similar\ \sigma\ 0\ A\ \sigma\ 1$

shows $similar\ \sigma\ 0\ B\ \sigma\ 1$

using *assms*

by *fastforce*

lemma *similar-refl*:

$similar\ \sigma\ n\ \sigma$

by *simp*

lemma *similar-commute*:

$similar\ \sigma\ n\ \sigma' = similar\ \sigma'\ n\ \sigma$

by *fastforce*

lemma *similar-trans*:

assumes $similar\ \sigma\ n\ \sigma'$

$similar\ \sigma'\ n\ \sigma''$

shows $similar\ \sigma\ n\ \sigma''$

using *assms* **by** *simp*

lemma *similar-refl*:

$similar\ \sigma\ A\ \sigma$

by *simp*

lemma *similar-trans*:

assumes $similar\ \sigma\ A\ \sigma'$

$similar\ \sigma'\ A\ \sigma''$

shows $similar\ \sigma\ A\ \sigma''$

using *assms* **by** *simp*

lemma *upd-NCons*:

$upd\ (NCons\ x\ xs)\ n\ (NCons\ y\ ys) =$
 $(if\ y\ then\ NCons\ (insert\ n\ x)\ (upd\ xs\ n\ ys)\ else\ NCons\ (x - \{n\})\ (upd\ xs\ n\ ys))$
by *simp*

lemma *upd-NNil*:

$upd\ (NNil\ a)\ n\ (NNil\ b) =$
 $(if\ b\ then\ (NNil\ (insert\ n\ a))\ else\ (NNil\ (a - \{n\})))$
by (*simp add: nfirst-eq-nnth-zero nnth-NNil*)

lemma *nnth-upd*:

assumes $nlength\ \sigma = nlength\ l$
 $(enat\ k) \leq nlength\ \sigma$
shows $(nnth\ (upd\ \sigma\ n\ l)\ k) = (if\ (nnth\ l\ k)\ then\ (insert\ n\ (nnth\ \sigma\ k))\ else\ (nnth\ \sigma\ k) - \{n\})$
using *assms*
proof (*induction k arbitrary: $\sigma\ l$*)
case 0
then show ?*case*
proof (*cases σ*)
case (*NNil x1*)
then show ?*thesis*
by (*simp add: nfirst-eq-nnth-zero nnth-NNil*)
next
case (*NCons x21 x22*)
then show ?*thesis*
proof –
have 1: $\exists\ b\ bs.\ l = (NCons\ b\ bs)$
using 0
by (*metis NCons nellist-split-2-first nlength-NCons order-le-less zero-enat-def zero-ne-eSuc*)
obtain *b bs* **where** 2: $l = (NCons\ b\ bs)$
using 1 **by** *blast*
show ?*thesis* **by** (*simp add: 2 NCons*)
qed
qed
next
case (*Suc k*)
then show ?*case*
proof (*cases σ*)
case (*NNil x1*)
then show ?*thesis* **using** *Suc enat-0-iff(1)* **by** *auto*
next
case (*NCons x21 x22*)
then show ?*thesis*
proof –
have 1: $\exists\ b\ bs.\ l = (NCons\ b\ bs)$
using *Suc*
by (*metis enat-0-iff(1) linorder-not-less nat.simps(3) nellist-split-2-first not-gr-zero*)
obtain *b bs* **where** 2: $l = (NCons\ b\ bs)$
using 1 **by** *blast*
show ?*thesis* **using** 2 *Suc*

```

    by (metis 2 NCons eSuc-enat eSuc-ile-mono eSuc-inject nlength-NCons nnth-Suc-NCons upd-simps(3))

qed
qed
qed

lemma nlength-upd:
  assumes nlength  $\sigma$  = nlength  $l$ 
  shows nlength ((upd  $\sigma$   $n$   $l$ )) = nlength  $\sigma$ 
  using assms
  proof (coinduction arbitrary:  $\sigma$   $l$  rule: enat-coinduct)
  case Eq-enat
  then show ?case
    proof (cases nlength  $\sigma$  = 0)
    case True
    then show ?thesis
      by (metis True nlength-NNil ntaken-0 ntaken-all order-le-less upd-simps(1) zero-enat-def)

    next
    case False
    then show ?thesis
      proof -
        have 2: nlength  $l$  > 0
          using False by (simp add: Eq-enat)
        show ?thesis using False 2 by simp
          (metis 2 Eq-enat co.enat.sel(2) nellist-split-2-first nlength-NCons upd-simps(3) zero-ne-eSuc)
      qed
    qed
  qed

lemma upd-nmap-nzip:
  assumes nlength  $\sigma$  = nlength  $l$ 
  shows upd  $\sigma$   $n$   $l$  = nmap ( $\lambda$  ( $x,y$ ). (if  $y$  then insert  $n$   $x$  else  $x - \{n\}$ )) (nzip  $\sigma$   $l$ )
  proof -
    have 1: nlength (upd  $\sigma$   $n$   $l$ ) = nlength (nmap ( $\lambda$  ( $x,y$ ). (if  $y$  then insert  $n$   $x$  else  $x - \{n\}$ )) (nzip  $\sigma$   $l$ ))
    using assms nlength-upd by auto
    have 2:  $\bigwedge i. i \leq \text{nlength (upd } \sigma \text{ } n \text{ } l) \implies$ 
      (nnth (upd  $\sigma$   $n$   $l$ )  $i$ ) =
      (nnth (nmap ( $\lambda$  ( $x,y$ ). (if  $y$  then insert  $n$   $x$  else  $x - \{n\}$ )) (nzip  $\sigma$   $l$ ))  $i$ )
      by (simp add: assms nlength-upd nnth-nzip nnth-upd)
    show ?thesis
      using 1 2 nellist-eq-nnth-eq by blast
  qed

lemma upd-swap:
  assumes nlength  $\sigma$  = nlength  $l1$ 
    nlength  $\sigma$  = nlength  $l2$ 
     $x1 \neq x2$ 

```

shows $\text{upd} (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2 = (\text{upd} (\text{upd } \sigma \ x2 \ l2) \ x1 \ l1)$
proof –
have 1: $\text{nlength} (\text{upd} (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) = \text{nlength} (\text{upd} (\text{upd } \sigma \ x2 \ l2) \ x1 \ l1)$
using *assms nlength-upd* **by** *auto*
have 2: $\bigwedge i. i \leq \text{nlength} (\text{upd} (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \implies$
 $(\text{nnth} (\text{upd} (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \ i) =$
 $(\text{if } (\text{nnth} \ l2 \ i) \text{ then insert } x2 \ (\text{nnth} (\text{upd } \sigma \ x1 \ l1) \ i) \text{ else } (\text{nnth} (\text{upd } \sigma \ x1 \ l1) \ i) - \{x2\})$
using *assms* **by** (*simp add: nlength-upd nnth-upd*)
have 3: $\bigwedge i. i \leq \text{nlength} (\text{upd} (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \implies$
 $(\text{nnth} (\text{upd} (\text{upd } \sigma \ x2 \ l2) \ x1 \ l1) \ i) =$
 $(\text{if } (\text{nnth} \ l1 \ i) \text{ then insert } x1 \ (\text{nnth} (\text{upd } \sigma \ x2 \ l2) \ i) \text{ else } (\text{nnth} (\text{upd } \sigma \ x2 \ l2) \ i) - \{x1\})$
using *assms* **by** (*simp add: nlength-upd nnth-upd*)
have 4: $\bigwedge i. i \leq \text{nlength} (\text{upd} (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \implies$
 $(\text{if } (\text{nnth} \ l2 \ i) \text{ then insert } x2 \ (\text{nnth} (\text{upd } \sigma \ x1 \ l1) \ i) \text{ else } (\text{nnth} (\text{upd } \sigma \ x1 \ l1) \ i) - \{x2\}) =$
 $(\text{if } (\text{nnth} \ l1 \ i) \text{ then insert } x1 \ (\text{nnth} (\text{upd } \sigma \ x2 \ l2) \ i) \text{ else } (\text{nnth} (\text{upd } \sigma \ x2 \ l2) \ i) - \{x1\})$
using *assms nnth-upd nlength-upd* **by** *auto*
show *?thesis*
by (*simp add: 1 2 3 4 nllist-eq-nnth-eq*)
qed

lemma *upd-absorb*:

assumes $\text{nlength } \sigma = \text{nlength } l1$
 $\text{nlength } \sigma = \text{nlength } l2$
 $x1 = x2$

shows $\text{upd} (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2 = (\text{upd } \sigma \ x2 \ l2)$

proof –
have 1: $\text{nlength} (\text{upd} (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) = \text{nlength} (\text{upd } \sigma \ x2 \ l2)$
using *assms nlength-upd* **by** *auto*
have 2: $\bigwedge i. i \leq \text{nlength} (\text{upd} (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \implies$
 $(\text{nnth} (\text{upd} (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \ i) =$
 $(\text{if } (\text{nnth} \ l2 \ i) \text{ then insert } x2 \ (\text{nnth} (\text{upd } \sigma \ x1 \ l1) \ i) \text{ else } (\text{nnth} (\text{upd } \sigma \ x1 \ l1) \ i) - \{x2\})$
using *assms* **by** (*simp add: nlength-upd nnth-upd*)
have 3: $\bigwedge i. i \leq \text{nlength} (\text{upd} (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \implies$
 $(\text{nnth} (\text{upd } \sigma \ x2 \ l2) \ i) =$
 $(\text{if } (\text{nnth} \ l2 \ i) \text{ then insert } x2 \ (\text{nnth } \sigma \ i) \text{ else } (\text{nnth } \sigma \ i) - \{x2\})$
using *assms* **by** (*simp add: nlength-upd nnth-upd*)
have 4: $\bigwedge i. i \leq \text{nlength} (\text{upd} (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \implies$
 $(\text{if } (\text{nnth} \ l2 \ i) \text{ then insert } x2 \ (\text{nnth} (\text{upd } \sigma \ x1 \ l1) \ i) \text{ else } (\text{nnth} (\text{upd } \sigma \ x1 \ l1) \ i) - \{x2\}) =$
 $(\text{if } (\text{nnth} \ l2 \ i) \text{ then insert } x2 \ (\text{nnth } \sigma \ i) \text{ else } (\text{nnth } \sigma \ i) - \{x2\})$
using *assms nnth-upd nlength-upd* **by** *auto*
show *?thesis*
by (*simp add: 1 2 3 4 nllist-eq-nnth-eq*)
qed

lemma *upd-nappend*:

assumes $\text{nlength } \sigma1 = \text{nlength } l1$
 $\text{nlength } \sigma2 = \text{nlength } l2$

shows $\text{upd} (\text{nappend } \sigma1 \ \sigma2) \ x \ (\text{nappend } l1 \ l2) =$
 $(\text{nappend} (\text{upd } \sigma1 \ x \ l1) (\text{upd } \sigma2 \ x \ l2))$

using *assms*

by (simp add: nmap-nappend-distrib nzip-nappend upd-nmap-nzip)

lemma *ntaken-upd*:

assumes $nlength\ \sigma = nlength\ l$

$j \leq nlength\ \sigma$

shows $ntaken\ j\ (upd\ \sigma\ x\ l) = (upd\ (ntaken\ j\ \sigma)\ x\ (ntaken\ j\ l))$

using *assms*

by (simp add: ntaken-nzip upd-nmap-nzip)

lemma *ndropn-upd*:

assumes $nlength\ \sigma = nlength\ l$

$j \leq nlength\ \sigma$

shows $ndropn\ j\ (upd\ \sigma\ x\ l) = (upd\ (ndropn\ j\ \sigma)\ x\ (ndropn\ j\ l))$

using *assms* *ndropn-nzip* *upd-nmap-nzip* **by** (simp add: ndropn-nmap)

lemma *nsubn-upd*:

assumes $nlength\ \sigma = nlength\ l$

$(enat\ j) \leq enat\ k$

$(enat\ k) \leq nlength\ \sigma$

shows $nsubn\ (upd\ \sigma\ x\ l)\ j\ k = (upd\ (nsubn\ \sigma\ j\ k)\ x\ (nsubn\ l\ j\ k))$

proof –

have 1: $nsubn\ (upd\ \sigma\ x\ l)\ j\ k = ntaken\ (k - j)\ (ndropn\ j\ (upd\ \sigma\ x\ l))$

using *assms* **unfolding** *nsubn-def1* **by** *simp*

have 2: $(ndropn\ j\ (upd\ \sigma\ x\ l)) = (upd\ (ndropn\ j\ \sigma)\ x\ (ndropn\ j\ l))$

using *assms* **by** (*meson* *dual-order.trans* *ndropn-upd*)

have 3: $ntaken\ (k - j)\ (upd\ (ndropn\ j\ \sigma)\ x\ (ndropn\ j\ l)) =$
 $(upd\ (ntaken\ (k - j)\ (ndropn\ j\ \sigma))\ x\ (ntaken\ (k - j)\ (ndropn\ j\ l)))$

using *assms* *ntaken-upd*[*of* $(ndropn\ j\ \sigma)\ (ndropn\ j\ l)\ (k - j)\ x]$

by (*metis* *ndropn-nlength* *nle-le* *nlength-upd* *ntaken-all*)

have 4: $(upd\ (ntaken\ (k - j)\ (ndropn\ j\ \sigma))\ x\ (ntaken\ (k - j)\ (ndropn\ j\ l))) =$
 $(upd\ (nsubn\ \sigma\ j\ k)\ x\ (nsubn\ l\ j\ k))$

unfolding *nsubn-def1* **by** *simp*

show *?thesis* **using** 1 2 3 4 **by** *presburger*

qed

lemma *nfirst-upd*:

assumes $nlength\ \sigma = nlength\ l$

shows $nfirst\ (upd\ \sigma\ x\ l) = (if\ nfirst\ l\ then\ insert\ x\ (nfirst\ \sigma)\ else\ (nfirst\ \sigma) - \{x\})$

by (*metis* *assms* *nfirst-eq-nnth-zero* *nnth-upd* *zero-enat-def* *zero-le*)

lemma *nlast-upd*:

assumes $nlength\ \sigma = nlength\ l$

$nfinite\ \sigma$

shows $nlast\ (upd\ \sigma\ x\ l) = (if\ nlast\ l\ then\ insert\ x\ (nlast\ \sigma)\ else\ (nlast\ \sigma) - \{x\})$

using *assms*

by (*metis* *dual-order.order-iff-strict* *nfinite-conv-nlength-enat* *nlength-upd* *nnth-nlast* *nnth-upd* *the-enat.simps*)

lemma *upd-nfuse*:

assumes $nlength\ \sigma 1 = nlength\ l1$

$nlength\ \sigma 2 = nlength\ l2$

$nfirst\ \sigma 2 = nlast\ \sigma 1$

$nfinite\ \sigma 1$

$nfirst\ l2 = nlast\ l1$

shows $upd\ (nfuse\ \sigma 1\ \sigma 2)\ x\ (nfuse\ l1\ l2) =$
 $(nfuse\ (upd\ \sigma 1\ x\ l1)\ (upd\ \sigma 2\ x\ l2))$

using *assms* **by** (*simp add: nfuse-nlength nmap-nfuse nzip-nfuse upd-nmap-nzip*)

lemma *similar-upd*:

assumes $nlength\ \sigma = nlength\ l$

shows $similar\ \sigma\ n\ (upd\ \sigma\ n\ l)$

using *assms*

using *nlength-upd nnth-upd* **by** *auto*

lemma *exist-sigma-exist-value*:

$(\exists\ \sigma'. (\sigma' \models f) \wedge (similar\ \sigma\ n\ \sigma')) \longleftrightarrow (\exists\ l. nlength\ \sigma = nlength\ l \wedge ((upd\ \sigma\ n\ l) \models f))$

proof –

have 2: $(\exists\ \sigma'. (\sigma' \models f) \wedge (similar\ \sigma\ n\ \sigma')) \implies$
 $(\exists\ l. nlength\ \sigma = nlength\ l \wedge ((upd\ \sigma\ n\ l) \models f))$

proof –

assume *a0*: $(\exists\ \sigma'. (\sigma' \models f) \wedge (similar\ \sigma\ n\ \sigma'))$

show $(\exists\ l. nlength\ \sigma = nlength\ l \wedge ((upd\ \sigma\ n\ l) \models f))$

proof –

obtain σ' **where** 3: $(\sigma' \models f) \wedge (similar\ \sigma\ n\ \sigma')$

using *a0* **by** *blast*

have 30: $nlength\ \sigma' = nlength\ (upd\ \sigma'\ n\ (nmap\ (\lambda x. n \in x)\ \sigma'))$

by (*simp add: nlength-upd*)

have 31: $\bigwedge k. k \leq nlength\ \sigma' \implies nnth\ \sigma'\ k = (nnth\ (upd\ \sigma'\ n\ (nmap\ (\lambda x. n \in x)\ \sigma'))\ k)$

by (*simp add: insert-absorb nnth-upd*)

have 4: $\sigma' = (upd\ \sigma'\ n\ (nmap\ (\lambda x. n \in x)\ \sigma'))$

by (*simp add: 30 31 nelist-eq-nnth-eq*)

have 40: $nlength\ \sigma = nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda x. n \in x)\ \sigma))$

by (*simp add: nlength-upd*)

have 41: $\bigwedge k. k \leq nlength\ \sigma \implies nnth\ \sigma\ k = (nnth\ (upd\ \sigma\ n\ (nmap\ (\lambda x. n \in x)\ \sigma))\ k)$

by (*simp add: insert-absorb nnth-upd*)

have 8: $\sigma = (upd\ \sigma\ n\ (nmap\ (\lambda x. n \in x)\ \sigma))$

by (*simp add: 40 41 nelist-eq-nnth-eq*)

have 12: $(similar\ \sigma\ n\ \sigma')$

using 3 **by** *auto*

have 13: $nlength\ \sigma = nlength\ \sigma'$

using 12 **by** *auto*

have 14: $\bigwedge k. k \leq nlength\ \sigma \implies (nnth\ \sigma'\ k) = (nnth\ (upd\ \sigma\ n\ (nmap\ (\lambda x. n \in x)\ \sigma'))\ k)$

using 13 *nnth-upd* 12 **by** *auto*

```

have 15: nlength  $\sigma' = \text{nlength } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda x. n \in x) \ \sigma'))$ 
  by (simp add: 13 nlength-upd)
have 16:  $\sigma' = (\text{upd } \sigma \ n \ (\text{nmap } (\lambda x. n \in x) \ \sigma'))$ 
  using nellist-eq-nnth-eq[of  $\sigma' (\text{upd } \sigma \ n \ (\text{nmap } (\lambda x. n \in x) \ \sigma'))$ ]
  using 13 14 15 by presburger
show ?thesis
  by (metis 13 16 3 nlength-nmap)
qed
qed
have 17:  $(\exists l. \text{nlength } \sigma = \text{nlength } l \wedge ((\text{upd } \sigma \ n \ l) \models f)) \implies$ 
   $(\exists \sigma'. (\sigma' \models f) \wedge (\text{similar } \sigma \ n \ \sigma'))$ 
proof -
  assume a1:  $(\exists l. \text{nlength } \sigma = \text{nlength } l \wedge ((\text{upd } \sigma \ n \ l) \models f))$ 
  show  $(\exists \sigma'. (\sigma' \models f) \wedge (\text{similar } \sigma \ n \ \sigma'))$ 
  proof -
    obtain l where 18:  $\text{nlength } \sigma = \text{nlength } l \wedge ((\text{upd } \sigma \ n \ l) \models f)$ 
    using a1 by blast
    have 19:  $\text{similar } \sigma \ n \ (\text{upd } \sigma \ n \ l)$ 
    using 18 nlength-upd nnth-upd by auto
    show ?thesis
    using 18 19 by blast
  qed
qed
show ?thesis
using 17 2 by blast
qed

```

lemma *not-fvar-upd*:

```

assumes  $n \notin \text{fvars } f$ 
   $\text{nlength } l = \text{nlength } \sigma$ 
shows  $(\sigma \models f) = ((\text{upd } \sigma \ n \ l) \models f)$ 
using assms
proof (induction f arbitrary:  $\sigma \ l$ )
case false-d
then show ?case
using semantics-qpitl.simps(1) by presburger
next
case (atom-d x)
then show ?case by simp
  (metis Diff-iff empty-iff insert-iff nfirst-eq-nnth-zero nnth-upd zero-enat-def zero-le)
next
case (iimp-d f1 f2)
then show ?case by simp
next
case (next-d f)
then show ?case using nlength-upd by simp
  (metis One-nat-def ndropn-nlength ndropn-upd one-enat-def)
next
case (chop-d f1 f2)

```

```

then show ?case
proof -
  have c1:  $(\sigma \models f1 \text{ schop } f2) =$ 
     $(\exists i. \text{enat } i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \sigma \models f1) \wedge (\text{ndropn } i \sigma \models f2))$  by simp
  have c2:  $(\text{upd } \sigma \text{ } n \text{ } l \models f1 \text{ schop } f2) =$ 
     $(\exists i. \text{enat } i \leq \text{nlength } (\text{upd } \sigma \text{ } n \text{ } l) \wedge (\text{ntaken } i (\text{upd } \sigma \text{ } n \text{ } l) \models f1) \wedge$ 
     $(\text{ndropn } i (\text{upd } \sigma \text{ } n \text{ } l) \models f2))$ 
    by simp
  have c3:  $(\exists i. \text{enat } i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \sigma \models f1) \wedge (\text{ndropn } i \sigma \models f2)) \implies$ 
     $(\exists i. \text{enat } i \leq \text{nlength } (\text{upd } \sigma \text{ } n \text{ } l) \wedge (\text{ntaken } i (\text{upd } \sigma \text{ } n \text{ } l) \models f1) \wedge$ 
     $(\text{ndropn } i (\text{upd } \sigma \text{ } n \text{ } l) \models f2))$ 
  proof -
    assume a0:  $(\exists i. \text{enat } i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \sigma \models f1) \wedge (\text{ndropn } i \sigma \models f2))$ 
    show  $(\exists i. \text{enat } i \leq \text{nlength } (\text{upd } \sigma \text{ } n \text{ } l) \wedge (\text{ntaken } i (\text{upd } \sigma \text{ } n \text{ } l) \models f1) \wedge$ 
     $(\text{ndropn } i (\text{upd } \sigma \text{ } n \text{ } l) \models f2))$ 
    proof -
      obtain i where c31:  $\text{enat } i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \sigma \models f1) \wedge (\text{ndropn } i \sigma \models f2)$ 
      using a0 by blast
      have c32:  $n \notin \text{fvars } f1$ 
      using chop-d.premis(1) by auto
      have c33:  $\text{nlength } (\text{ntaken } i \text{ } l) = \text{nlength } (\text{ntaken } i \sigma)$ 
      by (simp add: chop-d.premis(2))
      have c34:  $(\text{upd } (\text{ntaken } i \sigma) \text{ } n (\text{ntaken } i \text{ } l) \models f1)$ 
      using chop-d.IH(1)[of  $(\text{ntaken } i \text{ } l) (\text{ntaken } i \sigma)$ ] c31 c32 c33 by blast
      have c35:  $n \notin \text{fvars } f2$ 
      using chop-d.premis(1) by auto
      have c36:  $\text{nlength } (\text{ndropn } i \text{ } l) = \text{nlength } (\text{ndropn } i \sigma)$ 
      by (simp add: chop-d.premis(2))
      have c37:  $(\text{upd } (\text{ndropn } i \sigma) \text{ } n (\text{ndropn } i \text{ } l) \models f2)$ 
      using chop-d.IH(2)[of  $(\text{ndropn } i \text{ } l) (\text{ndropn } i \sigma)$ ] c31 c35 c36 by blast
      have c38:  $(\text{upd } (\text{ntaken } i \sigma) \text{ } n (\text{ntaken } i \text{ } l)) = \text{ntaken } i (\text{upd } \sigma \text{ } n \text{ } l)$ 
      by (simp add: c31 chop-d.premis(2) ntaken-upd)
      have c39:  $(\text{upd } (\text{ndropn } i \sigma) \text{ } n (\text{ndropn } i \text{ } l)) = \text{ndropn } i (\text{upd } \sigma \text{ } n \text{ } l)$ 
      by (simp add: c31 chop-d.premis(2) ndropn-upd)
      have c40:  $\text{enat } i \leq \text{nlength } (\text{upd } \sigma \text{ } n \text{ } l)$ 
      by (simp add: c31 chop-d.premis(2) nlength-upd)
      have c41:  $(\text{ntaken } i (\text{upd } \sigma \text{ } n \text{ } l) \models f1)$ 
      using c34 c38 by auto
      have c42:  $(\text{ndropn } i (\text{upd } \sigma \text{ } n \text{ } l) \models f2)$ 
      using c37 c39 by auto
      show ?thesis
      using c40 c41 c42 by auto
    qed
  qed
  have c5:  $(\exists i. \text{enat } i \leq \text{nlength } (\text{upd } \sigma \text{ } n \text{ } l) \wedge (\text{ntaken } i (\text{upd } \sigma \text{ } n \text{ } l) \models f1) \wedge$ 
     $(\text{ndropn } i (\text{upd } \sigma \text{ } n \text{ } l) \models f2)) \implies$ 
     $(\exists i. \text{enat } i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \sigma \models f1) \wedge (\text{ndropn } i \sigma \models f2))$ 
  proof -
    assume a1:  $(\exists i. \text{enat } i \leq \text{nlength } (\text{upd } \sigma \text{ } n \text{ } l) \wedge (\text{ntaken } i (\text{upd } \sigma \text{ } n \text{ } l) \models f1) \wedge$ 
     $(\text{ndropn } i (\text{upd } \sigma \text{ } n \text{ } l) \models f2))$ 

```

```

show ( $\exists i. \text{enat } i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \sigma \models f1) \wedge (\text{ndropn } i \sigma \models f2)$ )
proof –
  obtain  $i$  where  $c51: \text{enat } i \leq \text{nlength } (\text{upd } \sigma \text{ } n \text{ } l) \wedge (\text{ntaken } i (\text{upd } \sigma \text{ } n \text{ } l) \models f1) \wedge$ 
     $(\text{ndropn } i (\text{upd } \sigma \text{ } n \text{ } l) \models f2)$ 
  using  $a1$  by blast
  have  $c52: \text{nlength } (\text{upd } \sigma \text{ } n \text{ } l) = \text{nlength } \sigma$ 
  by (simp add: chop-d.premis(2) nlength-upd)
  have  $c53: (\text{ntaken } i (\text{upd } \sigma \text{ } n \text{ } l)) = \text{upd } (\text{ntaken } i \sigma) \text{ } n (\text{ntaken } i l)$ 
  using  $c51 \text{ } c52 \text{ } \text{chop-d.premis(2)} \text{ } \text{ntaken-upd}$  by auto
  have  $c54: n \notin \text{fvars } f1$ 
  using  $\text{chop-d.premis(1)}$  by auto
  have  $c55: (\text{ntaken } i \sigma \models f1)$ 
  by (metis c51 c53 c54 chop-d.IH(1) chop-d.premis(2) ntaken-nlength)
  have  $c56: n \notin \text{fvars } f2$ 
  using  $\text{chop-d.premis(1)}$  by auto
  have  $c57: (\text{ndropn } i (\text{upd } \sigma \text{ } n \text{ } l)) = \text{upd } (\text{ndropn } i \sigma) \text{ } n (\text{ndropn } i l)$ 
  using  $c51 \text{ } c52 \text{ } \text{chop-d.premis(2)} \text{ } \text{ndropn-upd}$  by auto
  have  $c58: (\text{ndropn } i \sigma \models f2)$ 
  by (metis c51 c56 c57 chop-d.IH(2) chop-d.premis(2) ndropn-nlength)
  show ?thesis
  using  $c51 \text{ } c52 \text{ } c55 \text{ } c58$  by auto
qed
qed
show ?thesis
using  $c1 \text{ } c2 \text{ } c3 \text{ } c5$  by blast
qed
next
case (schopstar-d f)
then show ?case
proof –
  have  $s1: (\sigma \models \text{schopstar } f) = (\exists l1. \text{nnth } l1 \text{ } 0 = 0 \wedge$ 
     $\text{nfinite } l1 \wedge$ 
     $\text{nidx } l1 \wedge$ 
     $\text{enat } (\text{nlast } l1) = \text{nlength } \sigma \wedge$ 
     $\text{nfinite } \sigma \wedge$ 
     $(\forall i. \text{enat } i < \text{nlength } l1 \longrightarrow$ 
       $(\text{nsbn } \sigma (\text{nnth } l1 \text{ } i)$ 
       $(\text{nnth } l1 (\text{Suc } i)) \models f)))$  by simp
  have  $s2: ((\text{upd } \sigma \text{ } n \text{ } l) \models \text{schopstar } f) = (\exists l1. \text{nnth } l1 \text{ } 0 = 0 \wedge$ 
     $\text{nfinite } l1 \wedge$ 
     $\text{nidx } l1 \wedge$ 
     $\text{enat } (\text{nlast } l1) = \text{nlength } (\text{upd } \sigma \text{ } n \text{ } l) \wedge$ 
     $\text{nfinite } (\text{upd } \sigma \text{ } n \text{ } l) \wedge$ 
     $(\forall i. \text{enat } i < \text{nlength } l1 \longrightarrow$ 
       $(\text{nsbn } (\text{upd } \sigma \text{ } n \text{ } l) (\text{nnth } l1 \text{ } i)$ 
       $(\text{nnth } l1 (\text{Suc } i)) \models f)))$  by simp
  have  $s3: (\exists l1. \text{nnth } l1 \text{ } 0 = 0 \wedge$ 
     $\text{nfinite } l1 \wedge$ 
     $\text{nidx } l1 \wedge$ 
     $\text{enat } (\text{nlast } l1) = \text{nlength } \sigma \wedge$ 

```


$nfinite\ \sigma \wedge$
 $(\forall i. enat\ i < nlength\ l1 \longrightarrow$
 $\quad (nsubn\ \sigma\ (nnth\ l1\ i)$
 $\quad\quad (nnth\ l1\ (Suc\ i)) \models f))) \implies$
 $(\exists l1. nnth\ l1\ 0 = 0 \wedge$
 $nfinite\ l1 \wedge$
 $nidx\ l1 \wedge$
 $enat\ (nlast\ l1) = nlength\ (upd\ \sigma\ n\ l) \wedge$
 $nfinite\ (upd\ \sigma\ n\ l) \wedge$
 $(\forall i. enat\ i < nlength\ l1 \longrightarrow$
 $\quad (nsubn\ (upd\ \sigma\ n\ l)\ (nnth\ l1\ i)$
 $\quad\quad (nnth\ l1\ (Suc\ i)) \models f)))$

proof –

assume $a2$: $(\exists l1. nnth\ l1\ 0 = 0 \wedge$
 $nfinite\ l1 \wedge$
 $nidx\ l1 \wedge$
 $enat\ (nlast\ l1) = nlength\ \sigma \wedge$
 $nfinite\ \sigma \wedge$
 $(\forall i. enat\ i < nlength\ l1 \longrightarrow$
 $\quad (nsubn\ \sigma\ (nnth\ l1\ i)$
 $\quad\quad (nnth\ l1\ (Suc\ i)) \models f)))$

show $(\exists l1. nnth\ l1\ 0 = 0 \wedge$
 $nfinite\ l1 \wedge$
 $nidx\ l1 \wedge$
 $enat\ (nlast\ l1) = nlength\ (upd\ \sigma\ n\ l) \wedge$
 $nfinite\ (upd\ \sigma\ n\ l) \wedge$
 $(\forall i. enat\ i < nlength\ l1 \longrightarrow$
 $\quad (nsubn\ (upd\ \sigma\ n\ l)\ (nnth\ l1\ i)$
 $\quad\quad (nnth\ l1\ (Suc\ i)) \models f)))$

proof –

obtain $l1$ **where** $s31$: $nnth\ l1\ 0 = 0 \wedge$
 $nfinite\ l1 \wedge$
 $nidx\ l1 \wedge$
 $enat\ (nlast\ l1) = nlength\ \sigma \wedge$
 $nfinite\ \sigma \wedge$
 $(\forall i. enat\ i < nlength\ l1 \longrightarrow$
 $\quad (nsubn\ \sigma\ (nnth\ l1\ i)$
 $\quad\quad (nnth\ l1\ (Suc\ i)) \models f))$

using $a2$ **by** $blast$

have $s32$: $nlength\ (upd\ \sigma\ n\ l) = nlength\ \sigma$

by $(simp\ add: nlength-upd\ schopstar-d.prem(2))$

have $s33$: $\bigwedge i. enat\ i < nlength\ l1 \longrightarrow$
 $nlength\ (nsubn\ l\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i))) =$
 $nlength(nsubn\ \sigma\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i)))$
by $(simp\ add: nsubn-nlength\ schopstar-d.prem(2))$

have $s33$: $\bigwedge i. enat\ i < nlength\ l1 \longrightarrow$
 $(nsubn\ \sigma\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i)) \models f) =$
 $(upd\ (nsubn\ \sigma\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i)))\ n$
 $\quad (nsubn\ l\ (nnth\ l1\ i)\ (nnth\ l1\ (Suc\ i))) \models f)$

using $fvars.simp(6)$ $s33$ $s chopstar-d.IH$ $s chopstar-d.prem(1)$ **by** $blast$

```

have s34:  $\bigwedge i. \text{enat } i < \text{nlength } l1 \longrightarrow$ 
  ( $\text{upd } (\text{nsubn } \sigma (\text{nnth } l1 \ i) (\text{nnth } l1 (\text{Suc } i))) \ n$ 
    ( $\text{nsubn } l (\text{nnth } l1 \ i) (\text{nnth } l1 (\text{Suc } i)))$ ) =
    ( $\text{nsubn } (\text{upd } \sigma \ n \ l) (\text{nnth } l1 \ i) (\text{nnth } l1 (\text{Suc } i)))$ )
by (metis eSuc-enat enat-ord-simps(1) ileI1 nidx-all-le-nlast nidx-expand nsubn-upd
  order-less-imp-le s31 schopstar-d.premis(2))
have s35:  $(\forall i. \text{enat } i < \text{nlength } l1 \longrightarrow$ 
  ( $\text{nsubn } (\text{upd } \sigma \ n \ l) (\text{nnth } l1 \ i) (\text{nnth } l1 (\text{Suc } i)) \models f$ ))
using s31 s33 s34 by auto
have s36: nfinite ( $\text{upd } \sigma \ n \ l$ )
by (metis nlength-eq-enat-nfiniteD s31 s32)
show ?thesis
using s31 s32 s35 s36 by auto
qed
qed
have s4:  $(\exists l1. \text{nnth } l1 \ 0 = 0 \wedge$ 
  nfinite  $l1 \wedge$ 
  nidx  $l1 \wedge$ 
  enat ( $\text{nlast } l1$ ) =  $\text{nlength } (\text{upd } \sigma \ n \ l) \wedge$ 
  nfinite ( $\text{upd } \sigma \ n \ l$ )  $\wedge$ 
   $(\forall i. \text{enat } i < \text{nlength } l1 \longrightarrow$ 
    ( $\text{nsubn } (\text{upd } \sigma \ n \ l) (\text{nnth } l1 \ i)$ 
      ( $\text{nnth } l1 (\text{Suc } i)) \models f$ )))  $\implies$ 
   $(\exists l1. \text{nnth } l1 \ 0 = 0 \wedge$ 
    nfinite  $l1 \wedge$ 
    nidx  $l1 \wedge$ 
    enat ( $\text{nlast } l1$ ) =  $\text{nlength } \sigma \wedge$ 
    nfinite  $\sigma \wedge$ 
     $(\forall i. \text{enat } i < \text{nlength } l1 \longrightarrow$ 
      ( $\text{nsubn } \sigma (\text{nnth } l1 \ i)$ 
        ( $\text{nnth } l1 (\text{Suc } i)) \models f$ )))
```

proof –

```

assume a3:  $(\exists l1. \text{nnth } l1 \ 0 = 0 \wedge$ 
  nfinite  $l1 \wedge$ 
  nidx  $l1 \wedge$ 
  enat ( $\text{nlast } l1$ ) =  $\text{nlength } (\text{upd } \sigma \ n \ l) \wedge$ 
  nfinite ( $\text{upd } \sigma \ n \ l$ )  $\wedge$ 
   $(\forall i. \text{enat } i < \text{nlength } l1 \longrightarrow$ 
    ( $\text{nsubn } (\text{upd } \sigma \ n \ l) (\text{nnth } l1 \ i)$ 
      ( $\text{nnth } l1 (\text{Suc } i)) \models f$ )))
```

show $(\exists l1. \text{nnth } l1 \ 0 = 0 \wedge$
 nfinite $l1 \wedge$
 nidx $l1 \wedge$
 enat ($\text{nlast } l1$) = $\text{nlength } \sigma \wedge$
 nfinite $\sigma \wedge$
 $(\forall i. \text{enat } i < \text{nlength } l1 \longrightarrow$
 ($\text{nsubn } \sigma (\text{nnth } l1 \ i)$
 ($\text{nnth } l1 (\text{Suc } i)) \models f$)))

proof –

```

obtain l1 where s41:  $\text{nnth } l1 \ 0 = 0 \wedge$ 
```

```

  nfinite l1 ∧
  nidx l1 ∧
  enat (nlast l1) = nlength (upd σ n l) ∧
  nfinite (upd σ n l) ∧
  (∀ i. enat i < nlength l1 →
    (nsubn (upd σ n l) (nnth l1 i)
      (nnth l1 (Suc i)) ⊨ f))
  using a3 by blast
have s42: nlength (upd σ n l) = nlength σ
by (simp add: nlength-upd schopstar-d.prem(2))
have s43: ∧i. enat i < nlength l1 →
  nlength (nsubn l (nnth l1 i) (nnth l1 (Suc i))) =
  nlength(nsubn σ (nnth l1 i) (nnth l1 (Suc i)))
by (simp add: nsubn-nlength schopstar-d.prem(2))
have s44: ∧i. enat i < nlength l1 →
  (nsubn (upd σ n l) (nnth l1 i) (nnth l1 (Suc i))) =
  upd (nsubn σ (nnth l1 i) (nnth l1 (Suc i))) n
  (nsubn l (nnth l1 i) (nnth l1 (Suc i)))
by (metis eSuc-enat enat-ord-simps(1) ileI1 nidx-all-le-nlast nidx-expand nsubn-upd
  order-less-imp-le s41 s42 schopstar-d.prem(2))
have s45: ∧i. enat i < nlength l1 →
  (upd (nsubn σ (nnth l1 i) (nnth l1 (Suc i))) n
    (nsubn l (nnth l1 i) (nnth l1 (Suc i))) ⊨ f) =
  (nsubn σ (nnth l1 i) (nnth l1 (Suc i)) ⊨ f)
using fvars.simps(6) s43 schopstar-d.IH schopstar-d.prem(1) by blast
have s46: (∀ i. enat i < nlength l1 →
  (nsubn σ (nnth l1 i) (nnth l1 (Suc i)) ⊨ f))
  using s41 s44 s45 by auto
have s47: nfinite σ
by (metis nlength-eq-enat-nfiniteD s41 s42)
show ?thesis
using s41 s42 s46 s47 by auto
qed
qed
show ?thesis
using s4 s1 s2 s3 by blast
qed
next
case (omega-d f)
then show ?case
proof -
  have o1: (σ ⊨ omega f) = (∃ l1. ¬ nfinite l1 ∧
    nnth l1 0 = 0 ∧
    nidx l1 ∧
    (∀ i. enat (nnth l1 i) ≤ nlength σ) ∧
    (∀ i. nsubn σ (nnth l1 i) (nnth l1 (Suc i)) ⊨ f))
  by simp
  have o2: ((upd σ n l) ⊨ omega f) = (∃ l1. ¬ nfinite l1 ∧
    nnth l1 0 = 0 ∧
    nidx l1 ∧

```

$(\forall i. \text{enat } (\text{nnth } l1 \ i) \leq \text{nlength } (\text{upd } \sigma \ n \ l)) \wedge$
 $(\forall i. (\text{nsubn } (\text{upd } \sigma \ n \ l) (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)) \models f)))$
by *simp*
have *o3*: $(\exists l1. \neg \text{nfinite } l1 \wedge$
 $\text{nnth } l1 \ 0 = 0 \wedge$
 $\text{nidx } l1 \wedge$
 $(\forall i. \text{enat } (\text{nnth } l1 \ i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. \text{nsubn } \sigma (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)) \models f)) \implies$
 $(\exists l1. \neg \text{nfinite } l1 \wedge$
 $\text{nnth } l1 \ 0 = 0 \wedge$
 $\text{nidx } l1 \wedge$
 $(\forall i. \text{enat } (\text{nnth } l1 \ i) \leq \text{nlength } (\text{upd } \sigma \ n \ l)) \wedge$
 $(\forall i. (\text{nsubn } (\text{upd } \sigma \ n \ l) (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)) \models f)))$
proof –
assume *a5*: $(\exists l1. \neg \text{nfinite } l1 \wedge$
 $\text{nnth } l1 \ 0 = 0 \wedge$
 $\text{nidx } l1 \wedge$
 $(\forall i. \text{enat } (\text{nnth } l1 \ i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. \text{nsubn } \sigma (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)) \models f))$
show $(\exists l1. \neg \text{nfinite } l1 \wedge$
 $\text{nnth } l1 \ 0 = 0 \wedge$
 $\text{nidx } l1 \wedge$
 $(\forall i. \text{enat } (\text{nnth } l1 \ i) \leq \text{nlength } (\text{upd } \sigma \ n \ l)) \wedge$
 $(\forall i. (\text{nsubn } (\text{upd } \sigma \ n \ l) (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)) \models f)))$
proof –
obtain *l1* **where** *o31*: $\neg \text{nfinite } l1 \wedge$
 $\text{nnth } l1 \ 0 = 0 \wedge$
 $\text{nidx } l1 \wedge$
 $(\forall i. \text{enat } (\text{nnth } l1 \ i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. \text{nsubn } \sigma (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)) \models f)$
using *a5* **by** *blast*
have *o32*: $(\forall i. \text{enat } (\text{nnth } l1 \ i) \leq \text{nlength } (\text{upd } \sigma \ n \ l))$
by (*simp add: nlength-upd o31 omega-d.premis(2)*)
have *o33*: $\bigwedge i.$
 $\text{nlength } (\text{nsubn } l (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i))) =$
 $\text{nlength}(\text{nsubn } \sigma (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)))$
by (*simp add: nsubn-nlength omega-d.premis(2)*)
have *o33*: $\bigwedge i.$
 $(\text{nsubn } \sigma (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)) \models f) =$
 $(\text{upd } (\text{nsubn } \sigma (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i))) \ n$
 $(\text{nsubn } l (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i))) \models f)$
using *fvars.simps(7) omega-d.IH omega-d.premis(1) o33* **by** *blast*
have *o34*: $\bigwedge i.$
 $(\text{upd } (\text{nsubn } \sigma (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i))) \ n$
 $(\text{nsubn } l (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)))) =$
 $(\text{nsubn } (\text{upd } \sigma \ n \ l) (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)))$
by (*metis Suc-n-not-le-n enat-ord-simps(1) linorder-le-cases nfinite-ntaken nidx-less-eq*
 $\text{nsubn-upd ntaken-all o31 omega-d.premis(2)}$)
have *o35*: $(\forall i.$
 $(\text{nsubn } (\text{upd } \sigma \ n \ l) (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)) \models f))$

```

    using o31 o33 o34 by auto
    show ?thesis
    using o31 o32 o35 by blast
qed
qed
have o4: ( $\exists l1. \neg \text{nfinite } l1 \wedge$ 
     $\text{nnth } l1 \ 0 = 0 \wedge$ 
     $\text{nidx } l1 \wedge$ 
     $(\forall i. \text{enat } (\text{nnth } l1 \ i) \leq \text{nlength } (\text{upd } \sigma \ n \ l)) \wedge$ 
     $(\forall i. (\text{nsbn } (\text{upd } \sigma \ n \ l) (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)) \models f))) \implies$ 
    ( $\exists l1. \neg \text{nfinite } l1 \wedge$ 
     $\text{nnth } l1 \ 0 = 0 \wedge$ 
     $\text{nidx } l1 \wedge$ 
     $(\forall i. \text{enat } (\text{nnth } l1 \ i) \leq \text{nlength } \sigma) \wedge$ 
     $(\forall i. \text{nsbn } \sigma (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)) \models f))$ )
proof -
    assume a6: ( $\exists l1. \neg \text{nfinite } l1 \wedge$ 
     $\text{nnth } l1 \ 0 = 0 \wedge$ 
     $\text{nidx } l1 \wedge$ 
     $(\forall i. \text{enat } (\text{nnth } l1 \ i) \leq \text{nlength } (\text{upd } \sigma \ n \ l)) \wedge$ 
     $(\forall i. (\text{nsbn } (\text{upd } \sigma \ n \ l) (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)) \models f)))$ )
    show ( $\exists l1. \neg \text{nfinite } l1 \wedge$ 
     $\text{nnth } l1 \ 0 = 0 \wedge$ 
     $\text{nidx } l1 \wedge$ 
     $(\forall i. \text{enat } (\text{nnth } l1 \ i) \leq \text{nlength } \sigma) \wedge$ 
     $(\forall i. \text{nsbn } \sigma (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)) \models f))$ )
    proof -
        obtain l1 where o41:  $\neg \text{nfinite } l1 \wedge$ 
         $\text{nnth } l1 \ 0 = 0 \wedge$ 
         $\text{nidx } l1 \wedge$ 
         $(\forall i. \text{enat } (\text{nnth } l1 \ i) \leq \text{nlength } (\text{upd } \sigma \ n \ l)) \wedge$ 
         $(\forall i. (\text{nsbn } (\text{upd } \sigma \ n \ l) (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)) \models f))$ 
        using a6 by blast
    have o42:  $(\forall i. \text{enat } (\text{nnth } l1 \ i) \leq \text{nlength } \sigma)$ 
    using nlength-upd o41 omega-d.premis(2) by auto
    have o43:  $\bigwedge i. \text{nlength } (\text{nsbn } l (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i))) =$ 
     $\text{nlength}(\text{nsbn } \sigma (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)))$ 
    by (simp add: nsbn-nlength omega-d.premis(2))
    have o44:  $\bigwedge i. (\text{nsbn } (\text{upd } \sigma \ n \ l) (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i))) =$ 
     $\text{upd } (\text{nsbn } \sigma (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i))) \ n$ 
     $(\text{nsbn } l (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)))$ 
    by (metis enat-ord-simps(1) less-Suc-eq less-le-not-le linorder-not-less nfinite-ntaken
    nidx-less-eq nsbn-upd ntaken-all o41 o42 omega-d.premis(2))
    have o45:  $\bigwedge i. (\text{upd } (\text{nsbn } \sigma (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i))) \ n$ 
     $(\text{nsbn } l (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i))) \models f) =$ 
     $(\text{nsbn } \sigma (\text{nnth } l1 \ i) (\text{nnth } l1 \ (\text{Suc } i)) \models f)$ 
    using fvars.simps(7) o43 omega-d.IH omega-d.premis(1) by blast

```

```

have o46: (∀ i.
  (nsubn σ (nnth l1 i) (nnth l1 (Suc i)) ⊨ f))
using o41 o44 o45 by auto
show ?thesis
using o41 o42 o46 by blast
qed
qed
show ?thesis
using o1 o2 o3 o4 by blast
qed
next
case (exists-d x1 f)
then show ?case
proof -
have e1: (σ ⊨ Ex x1. f) = (∃ l. nlength σ = nlength l ∧ ((upd σ x1 l) ⊨ f) )
  by simp
have e2: (upd σ n l ⊨ Ex x1. f) =
  (∃ l1. nlength (upd σ n l) = nlength l1 ∧ ((upd (upd σ n l) x1 l1) ⊨ f) )
  by simp
have e3: (∃ l. nlength σ = nlength l ∧ ((upd σ x1 l) ⊨ f) ) ⇒
  (∃ l1. nlength (upd σ n l) = nlength l1 ∧ ((upd (upd σ n l) x1 l1) ⊨ f) )
  by (metis Diff-iff empty-iff exists-d.IH exists-d.prem1 exists-d.prem2
    fvars.simps(8) insert-iff nlength-upd upd-absorb upd-swap)
have e4: (∃ l1. nlength (upd σ n l) = nlength l1 ∧ ((upd (upd σ n l) x1 l1) ⊨ f) ) ⇒
  (∃ l. nlength σ = nlength l ∧ ((upd σ x1 l) ⊨ f) )
  by (metis Diff-iff empty-iff exists-d.IH exists-d.prem1 exists-d.prem2 fvars.simps(8)
    insert-iff nlength-upd upd-absorb upd-swap)
show ?thesis
using e1 e2 e3 e4 by blast
qed
qed

lemma inot-defs [simp]:
  (σ ⊨ inot f) = Not (σ ⊨ f)
by (simp add: inot-d-def)

lemma ior-defs [simp]:
  (σ ⊨ f1 ior f2) = (σ ⊨ f1) ∨ (σ ⊨ f2)
by (metis inot-defs ior-d-def semantics-qpitl.simps(3))

lemma iand-defs [simp]:
  (σ ⊨ f1 iand f2) = (σ ⊨ f1) ∧ (σ ⊨ f2)
by (simp add: iand-d-def)

lemma ieqv-defs [simp]:
  (σ ⊨ f1 ieqv f2) = (σ ⊨ f1) = (σ ⊨ f2)
by (metis iand-defs ieqv-d-def semantics-qpitl.simps(3))

lemma itrue-defs [simp]:
  (σ ⊨ itrue)

```

by (*simp add: itrue-d-def*)

lemma *empty-defs* [*simp*]:

$(\sigma \models \text{empty}) = (\text{nlength } \sigma = 0)$

by (*auto simp add: empty-d-def more-d-def*)

$(\text{metis co.enat.exhaust-sel enat-le-plus-same}(2) \text{ plus-1-eSuc}(2))$

lemma *forall-defs* [*simp*]:

$(\sigma \models \text{Fa } v. f) = (\forall l. \text{nlength } l = \text{nlength } \sigma \longrightarrow (\text{upd } \sigma \ v \ l \models f))$

using *forall-d-def* **by** *auto*

lemma *sometimes-defs*:

$(\sigma \models \text{diamond } f) = (\exists i. i \leq \text{nlength } \sigma \wedge ((\text{ndropn } i \ \sigma) \models f))$

by (*auto simp add: sometimes-d-def*)

lemma *always-defs*:

$(\sigma \models \text{box } f) = (\forall i. i \leq \text{nlength } \sigma \longrightarrow ((\text{ndropn } i \ \sigma) \models f))$

by (*auto simp add: always-d-def sometimes-defs*)

lemma *fin-defs*:

$(\sigma \models (\text{fin } f)) \longleftrightarrow \text{nfinite } \sigma \wedge ((\text{ndropn } (\text{the-enat}(\text{nlength } \sigma)) \ \sigma) \models f)$

apply (*auto simp add: sometimes-defs fin-d-def*)

apply (*metis ndropn-nlength nfinite-ndropn nlength-eq-enat-nfiniteD zero-enat-def*)

apply (*metis enat-diff-cancel-left idiff-self not-enat-eq order-refl the-enat.simps*)

by (*metis enat-the-enat idiff-self infinity-ileE is-NNil-def is-NNil-ndropn ndropn-nlast*)

lemma *ifinite-defs*:

$(\sigma \models \text{ifinite}) \longleftrightarrow \text{nfinite } \sigma$

apply (*auto simp add: finite-d-def*)

using *nfinite-conv-nlength-enat* **apply** *fastforce*

by (*metis enat-ord-simps*(3) *enat-the-enat ndropn-nlast ndropn-nlength nlength-NNil order-refl*)

lemma *inf-defs*:

$(\sigma \models \text{inf}) \longleftrightarrow \neg \text{nfinite } \sigma$

by (*auto simp add: inf-d-def ifinite-defs*)

lemma *chop-defs*:

$(\sigma \models f;g) \longleftrightarrow (\exists i. i \leq \text{nlength } \sigma \wedge ((\text{ntaken } i \ \sigma) \models f) \wedge ((\text{ndropn } i \ \sigma) \models g)) \vee (\neg \text{nfinite } \sigma \wedge (\sigma \models f))$

by (*auto simp add: chop-d-def inf-defs*)

lemma *di-defs*:

$(\sigma \models \text{di } f) \longleftrightarrow (\exists i. i \leq \text{nlength } \sigma \wedge ((\text{ntaken } i \ \sigma) \models f)) \vee (\neg \text{nfinite } \sigma \wedge (\sigma \models f))$

by (*auto simp add: di-d-def chop-defs*)

lemma *bi-defs*:

$(\sigma \models \text{bi } f) \longleftrightarrow (\forall i. i \leq \text{nlength } \sigma \longrightarrow ((\text{ntaken } i \ \sigma) \models f)) \wedge (\neg \text{nfinite } \sigma \longrightarrow (\sigma \models f))$

by (*auto simp add: di-defs bi-d-def*)

lemma *df-defs*:

$(\sigma \models df\ f) \longleftrightarrow (\exists i. i \leq nlength\ \sigma \wedge ((ntaken\ i\ \sigma) \models f))$

by (*auto simp add: df-d-def*)

lemma *bf-defs*:

$(\sigma \models bf\ f) \longleftrightarrow (\forall i. i \leq nlength\ \sigma \longrightarrow ((ntaken\ i\ \sigma) \models f))$

by (*auto simp add: df-defs bf-d-def*)

lemma *da-defs* :

$(\sigma \models da\ f) =$
 $(\exists n\ k. (n \leq k \wedge k \leq nlength\ \sigma \wedge ((nsubn\ \sigma\ n\ k) \models f))) \vee$
 $(\neg nfinite\ \sigma \wedge (\exists n. ((ndropn\ n\ \sigma) \models f)))$
)

apply (*auto simp add: da-d-def chop-defs*)

apply (*metis enat-min-eq le-add2 ntaken-ndropn plus-enat-simps(1)*)

apply (*metis enat-le-plus-same(2) enat-min-eq enat-ord-simps(1) ntaken-ndropn*
plus-enat-simps(1))

apply (*metis enat-minus-mono1 enat-ord-simps(1) idiff-enat-enat nsubn-def1 order-subst2*)

using *is-NNil-imp-nfinite is-NNil-ndropn nfinite-ndropn nle-le* **by** *blast*

lemma *ba-defs*:

$(\sigma \models ba\ f) \longleftrightarrow ((\forall n\ k. (n \leq k \wedge k \leq nlength\ \sigma \longrightarrow ((nsubn\ \sigma\ n\ k) \models f)) \wedge (\neg nfinite\ \sigma \longrightarrow ((ndropn\ n\ \sigma) \models f)))$
)

by (*auto simp add: ba-d-def da-defs*)

lemma *sda-defs* :

$(\sigma \models sda\ f) =$
 $(\exists n\ k. (n \leq k \wedge k \leq nlength\ \sigma \wedge ((nsubn\ \sigma\ n\ k) \models f)))$
)

apply (*auto simp add: sda-d-def chop-defs*)

apply (*metis add-diff-cancel-right' enat-min-eq le-add2 nsubn-def1 plus-enat-simps(1)*)

by (*metis enat-minus-mono1 enat-ord-simps(1) idiff-enat-enat nsubn-def1 order.trans*)

lemma *sba-defs*:

$(\sigma \models sba\ f) \longleftrightarrow ((\forall n\ k. (n \leq k \wedge k \leq nlength\ \sigma \longrightarrow ((nsubn\ \sigma\ n\ k) \models f)))$
)

by (*auto simp add: sba-d-def sda-defs*)

lemma *state-qpitl-defs*:

assumes *state-qpitl w*

shows $(\sigma \models w) \longleftrightarrow ((NNil\ (nfirst\ \sigma)) \models w)$

using *assms*

proof (*induct w arbitrary: σ*)

case *false-d*

then show *?case* **by** *simp*

next


```

case (atom-d x)
then show ?case
by (metis nfirst-nfuse nfuse-leftneutral nlast-NNil semantics-qpitl.simps(2))
next
case (iimp-d w1 w2)
then show ?case by simp
next
case (next-d w)
then show ?case by auto
next
case (chop-d w1 w2)
then show ?case by auto
next
case (schopstar-d w)
then show ?case by auto
next
case (omega-d w)
then show ?case by auto
next
case (exists-d x1 w)
then show ?case
  proof –
    have 1:  $(\sigma \models \text{Ex } x1. w) \longleftrightarrow (\exists \sigma''. (\sigma'' \models w) \wedge \text{similar } \sigma \ x1 \ \sigma'')$ 
    using exist-sigma-exist-value semantics-qpitl.simps(8) by presburger
    have 2:  $(\text{NNil } (nfirst \ \sigma) \models \text{Ex } x1. w) \longleftrightarrow (\exists \sigma''. (\sigma'' \models w) \wedge \text{similar } (\text{NNil } (nfirst \ \sigma)) \ x1 \ \sigma'')$ 
    using exist-sigma-exist-value semantics-qpitl.simps(8) by presburger
    have 3:  $(\exists \sigma''. (\sigma'' \models w) \wedge \text{similar } \sigma \ x1 \ \sigma'') \implies$ 
       $(\exists \sigma''. (\sigma'' \models w) \wedge \text{similar } (\text{NNil } (nfirst \ \sigma)) \ x1 \ \sigma'')$ 
    proof –
      assume a0:  $(\exists \sigma''. (\sigma'' \models w) \wedge \text{similar } \sigma \ x1 \ \sigma'')$ 
      show  $(\exists \sigma''. (\sigma'' \models w) \wedge \text{similar } (\text{NNil } (nfirst \ \sigma)) \ x1 \ \sigma'')$ 
      proof –
        obtain  $\sigma''$  where 4:  $(\sigma'' \models w) \wedge \text{similar } \sigma \ x1 \ \sigma''$ 
        using a0 by blast
        have 5: state-qpitl w
        using exists-d.prems by auto
        have 6:  $(\text{NNil } (nfirst \ \sigma'') \models w)$ 
        using 4 5 exists-d.hyps by blast
        have 7:  $\text{similar } (\text{NNil } (nfirst \ \sigma)) \ x1 \ (\text{NNil } (nfirst \ \sigma''))$ 
        by (metis (no-types, lifting) 4 enat-defs(1) nfirst-eq-nnth-zero nlength-NNil nnth-NNil
          similar-defs-alt zero-order(1))
        show ?thesis
        using 6 7 by blast
      qed
    qed
  have 8:  $(\exists \sigma''. (\sigma'' \models w) \wedge \text{similar } (\text{NNil } (nfirst \ \sigma)) \ x1 \ \sigma'') \implies$ 
     $(\exists \sigma''. (\sigma'' \models w) \wedge \text{similar } \sigma \ x1 \ \sigma'')$ 
  proof –
    assume a1:  $(\exists \sigma''. (\sigma'' \models w) \wedge \text{similar } (\text{NNil } (nfirst \ \sigma)) \ x1 \ \sigma'')$ 
    show  $(\exists \sigma''. (\sigma'' \models w) \wedge \text{similar } \sigma \ x1 \ \sigma'')$ 

```

proof –
obtain σ'' **where** $9: (\sigma'' \models w) \wedge \text{similar } (NNil \ (nfirst \ \sigma)) \ x1 \ \sigma''$
using $a1$ **by** *blast*
have $10: \text{state-qpitl } w$
using *exists-d.premis* **by** *auto*
have $11: (NNil \ (nfirst \ \sigma'') \models w)$
using $9 \ 10$ *exists-d.hyps* **by** *blast*
have $12: \text{nlength } \sigma = 0 \implies \text{similar } \sigma \ x1 \ \sigma''$
by (*metis* 9 *ndropn-0 ndropn-nlast nlength-eq-enat-nfiniteD ntake-0 ntake-NNil*
the-enat.simps zero-enat-def)
have $13: \text{nlength } \sigma > 0 \implies \text{similar } \sigma \ x1 \ (\text{nappend } (NNil \ (nfirst \ \sigma)) \ (\text{ndropn } 1 \ \sigma))$
by (*simp* *add: enat-defs(1) ndropn-Suc-conv-ndropn nfirst-eq-nnth-zero*)
have $14: \text{nlength } \sigma = 0 \implies (\sigma'' \models w)$
by (*simp* *add: 9*)
have $15: \text{nlength } \sigma > 0 \implies (NNil \ (nfirst \ (\text{nappend } (NNil \ (nfirst \ \sigma)) \ (\text{ndropn } 1 \ \sigma)))) = (NNil \ (nfirst \ \sigma))$
by (*simp* *add: nfirst-eq-nnth-zero*)
have $16: \text{nlength } \sigma > 0 \implies \text{similar } (NNil \ (nfirst \ \sigma)) \ x1 \ (NNil \ (nfirst \ \sigma''))$
by (*metis* 9 *le-numeral-extra(3) nlength-NNil ntaken-0 ntaken-all similar-defs-alt zero-enat-def*)
have $17: \text{nlength } \sigma > 0 \implies \sigma = \text{nappend } (NNil \ (nfirst \ \sigma)) \ (\text{ndropn } 1 \ \sigma)$
by (*simp* *add: ndropn-Suc-conv-ndropn nfirst-eq-nnth-zero zero-enat-def*)
have $18: \text{nlength } \sigma > 0 \implies \text{similar } (\text{nappend } (NNil \ (nfirst \ \sigma)) \ (\text{ndropn } 1 \ \sigma)) \ x1 \ (\text{nappend } (NNil \ (nfirst \ \sigma'')) \ (\text{ndropn } 1 \ \sigma))$
by *simp*
(metis (no-types, lifting) 9 le-zero-eq nappend-NNil nfirst-eq-nnth-zero nlength-NNil nnth-0
nnth-nappend similar-defs the-enat.simps)
have $19: \text{nlength } \sigma > 0 \implies (\text{nappend } (NNil \ (nfirst \ \sigma'')) \ (\text{ndropn } 1 \ \sigma)) \models w$
by (*metis* $10 \ 9$ *exists-d.hyps nfirst-eq-nnth-zero nnth-nappend zero-enat-def zero-le*)
show *?thesis*
by (*metis* $12 \ 17 \ 18 \ 19 \ 9$ *i0-less*)
qed
qed
show *?thesis*
using $1 \ 2 \ 3 \ 8$ **by** *blast*
qed
qed

lemma *not-fvar-upd-state:*

assumes $n \notin \text{fvars } w$
 $\text{nlength } l = \text{nlength } \sigma$
 $\text{state-qpitl } w$
shows $(NNil \ (\text{nnth } \sigma \ 0) \models w) = (NNil \ (\text{nnth } (\text{upd } \sigma \ n \ l) \ 0) \models w)$
using *assms* **by** (*metis* *ndropn-0 ndropn-nfirst not-fvar-upd state-qpitl-defs*)

lemma *not-fvar-upd-nmap:*

assumes $n \notin \text{fvars } w$
 $\text{nlength } l = \text{nlength } \sigma$
 $\text{state-qpitl } w$
shows $(\text{nmap } (\lambda c. NNil \ c \models w) \ (\text{upd } \sigma \ n \ l)) = (\text{nmap } (\lambda c. NNil \ c \models w) \ \sigma)$
proof –

```

have 1:  $nlength\ (nmap\ (\lambda c. NNil\ c \models w)\ (upd\ \sigma\ n\ l)) = nlength\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma)$ 
by (simp add: assms(2) nlength-upd)
have 2:  $\bigwedge i. enat\ i \leq nlength\ (nmap\ (\lambda c. NNil\ c \models w)\ (upd\ \sigma\ n\ l)) \implies$ 
 $(nnth\ (nmap\ (\lambda c. NNil\ c \models w)\ (upd\ \sigma\ n\ l))\ i) =$ 
 $(nnth\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma)\ i)$ 
using assms nnth-nmap[of - (upd  $\sigma$   $n$   $l$ ) ( $\lambda c. NNil\ c \models w$ ) ]
by (metis (mono-tags, lifting) 1 ndropn-nfirst ndropn-nlength ndropn-upd nlength-nmap
 $nnth-nmap\ not-fvar-upd\ state-qpitl-defs$ )
show ?thesis
using 1 2 nellist-eq-nnth-eq by blast
qed

```

lemma *subst-suform*:

assumes $n \notin bvars\ f$

$\bigwedge z. z \in fvars\ w \implies z \notin bvars\ f$
state-qpitl w

shows $(\sigma \models suform\ f\ n\ w) = ((upd\ \sigma\ n\ (nmap\ (\lambda c. (NNil\ c) \models w)\ \sigma)) \models f)$

using *assms*

proof (*induction f arbitrary: $\sigma\ w\ n$*)

case *false-d*

then show ?case **by** *simp*

next

case (*atom-d x*)

then show ?case

proof –

have *a1*: $(\sigma \models (suform\ (\$x)\ n\ w)) = (if\ x = n\ then\ (\sigma \models w)\ else\ (x \in nfirst\ \sigma))$

by *simp*

have *a2*: $nlength\ \sigma = nlength\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma)$

by *auto*

have *a3*: $(upd\ \sigma\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma) \models \$x) = (if\ x = n\ then\ (\sigma \models w)\ else\ (x \in nfirst\ \sigma))$

using *a2 nfirst-upd* **by** *simp*

(*metis atom-d.premis(3) ndropn-0 ndropn-nfirst nnth-nmap state-qpitl-defs zero-enat-def zero-le*)

show ?thesis

using *a1 a3* **by** *blast*

qed

next

case (*iimp-d f1 f2*)

then show ?case **by** *auto*

next

case (*next-d f*)

then show ?case **by** (*auto simp add: nlength-upd ndropn-nmap ndropn-upd one-enat-def*)

next

case (*chop-d f1 f2*)

then show ?case

proof –

have *c1*: $(\sigma \models suform\ (f1\ schop\ f2)\ n\ w) =$

$(\exists i. enat\ i \leq nlength\ \sigma \wedge (ntaken\ i\ \sigma \models suform\ f1\ n\ w) \wedge (ndropn\ i\ \sigma \models suform\ f2\ n\ w))$

by *simp*

have *c2*: $(upd\ \sigma\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma) \models f1\ schop\ f2) =$

$(\exists i. \text{enat } i \leq \text{nlength } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \wedge$
 $(\text{ntaken } i \text{ } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \models f1) \wedge$
 $(\text{ndropn } i \text{ } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \models f2))$
by *simp*
have *c3*: $(\exists i. \text{enat } i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \text{ } \sigma \models \text{suform } f1 \text{ } n \text{ } w) \wedge (\text{ndropn } i \text{ } \sigma \models \text{suform } f2 \text{ } n \text{ } w))$
 \implies
 $(\exists i. \text{enat } i \leq \text{nlength } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \wedge$
 $(\text{ntaken } i \text{ } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \models f1) \wedge$
 $(\text{ndropn } i \text{ } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \models f2))$
proof –
assume *c4*: $(\exists i. \text{enat } i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \text{ } \sigma \models \text{suform } f1 \text{ } n \text{ } w) \wedge (\text{ndropn } i \text{ } \sigma \models \text{suform } f2 \text{ } n \text{ } w))$
show $(\exists i. \text{enat } i \leq \text{nlength } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \wedge$
 $(\text{ntaken } i \text{ } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \models f1) \wedge$
 $(\text{ndropn } i \text{ } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \models f2))$
proof –
obtain *i* **where** *c5*: $\text{enat } i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \text{ } \sigma \models \text{suform } f1 \text{ } n \text{ } w) \wedge (\text{ndropn } i \text{ } \sigma \models \text{suform } f2 \text{ } n \text{ } w)$
using *c4* **by** *blast*
have *c6*: $\text{nlength } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) = \text{nlength } \sigma$
by (*simp add: nlength-upd*)
have *c7*: $n \notin \text{bvars } f1$
using *chop-d.premis(1)* **by** *auto*
have *c8*: $(\text{ntaken } i \text{ } \sigma \models \text{suform } f1 \text{ } n \text{ } w) = (\text{ntaken } i \text{ } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \models f1)$
using *chop-d.IH(1)[of n w (ntaken i σ)]*
using *c5 c7 chop-d.premis(2) chop-d.premis(3) ntaken-upd* **by** *auto*
have *c9*: $n \notin \text{bvars } f2$
using *chop-d.premis(1)* **by** *auto*
have *c10*: $(\text{ndropn } i \text{ } \sigma \models \text{suform } f2 \text{ } n \text{ } w) = (\text{ndropn } i \text{ } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \models f2)$
using *chop-d.IH(2)[of n w (ndropn i σ)]*
by (*metis Un-iff bvars.simps(5) c5 c9 chop-d.premis(2) chop-d.premis(3) ndropn-nmap ndropn-upd*
nlength-nmap)
show *?thesis*
using *c10 c5 c6 c8* **by** *auto*
qed
qed
have *c11*: $(\exists i. \text{enat } i \leq \text{nlength } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \wedge$
 $(\text{ntaken } i \text{ } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \models f1) \wedge$
 $(\text{ndropn } i \text{ } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \models f2))$
 \implies
 $(\exists i. \text{enat } i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \text{ } \sigma \models \text{suform } f1 \text{ } n \text{ } w) \wedge (\text{ndropn } i \text{ } \sigma \models \text{suform } f2 \text{ } n \text{ } w))$
proof –
assume *c12*: $(\exists i. \text{enat } i \leq \text{nlength } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \wedge$
 $(\text{ntaken } i \text{ } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \models f1) \wedge$
 $(\text{ndropn } i \text{ } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \models f2))$
show $(\exists i. \text{enat } i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \text{ } \sigma \models \text{suform } f1 \text{ } n \text{ } w) \wedge (\text{ndropn } i \text{ } \sigma \models \text{suform } f2 \text{ } n \text{ } w))$
proof –
obtain *i* **where** *c13*: $\text{enat } i \leq \text{nlength } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \wedge$
 $(\text{ntaken } i \text{ } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \models f1) \wedge$
 $(\text{ndropn } i \text{ } (\text{upd } \sigma \text{ } n \text{ } (\text{nmap } (\lambda c. \text{NNil } c \models w) \sigma)) \models f2)$
using *c12* **by** *blast*

```

have c14: nlength (upd  $\sigma$  n (nmap ( $\lambda c$ . NNil c  $\models$  w)  $\sigma$ )) = nlength  $\sigma$ 
by (simp add: nlength-upd)
have c15: n  $\notin$  bvars f1
  using chop-d.premis(1) by auto
have c16: (ntaken i  $\sigma \models$  suform f1 n w) = (ntaken i (upd  $\sigma$  n (nmap ( $\lambda c$ . NNil c  $\models$  w)  $\sigma$ ))  $\models$  f1)
  using chop-d.IH(1)[of n w (ntaken i  $\sigma$ )]
  using c13 c14 c15 chop-d.premis(2) chop-d.premis(3) ntaken-upd by auto
have c17: n  $\notin$  bvars f2
  using chop-d.premis(1) by auto
have c18: (ndropn i  $\sigma \models$  suform f2 n w) = (ndropn i (upd  $\sigma$  n (nmap ( $\lambda c$ . NNil c  $\models$  w)  $\sigma$ ))  $\models$  f2)
  using chop-d.IH(2)[of n w (ndropn i  $\sigma$ )]
  by (metis Un-iff bvars.simps(5) c13 c14 c17 chop-d.premis(2) chop-d.premis(3) ndropn-nmap ndropn-upd
nlength-nmap)
show ?thesis
using c13 c14 c16 c18 by auto
qed
qed
show ?thesis
using c1 c11 c2 c3 by blast
qed
next
case (schopstar-d f)
then show ?case
proof –
have s1: ( $\sigma \models$  suform (schopstar f) n w) =
  ( $\exists l$ . nnth l 0 = 0  $\wedge$ 
    nfinite l  $\wedge$ 
    nidx l  $\wedge$ 
    enat (nlast l) = nlength  $\sigma \wedge$  nfinite  $\sigma \wedge$ 
    ( $\forall i$ . enat i < nlength l  $\longrightarrow$  (nsubn  $\sigma$  (nnth l i) (nnth l (Suc i))  $\models$  suform f n w)))
by simp
have s2: (upd  $\sigma$  n (nmap ( $\lambda c$ . NNil c  $\models$  w)  $\sigma$ )  $\models$  chopstar f) =
  ( $\exists l$ . nnth l 0 = 0  $\wedge$ 
    nfinite l  $\wedge$ 
    nidx l  $\wedge$ 
    enat (nlast l) = nlength (upd  $\sigma$  n (nmap ( $\lambda c$ . NNil c  $\models$  w)  $\sigma$ ))  $\wedge$ 
    nfinite (upd  $\sigma$  n (nmap ( $\lambda c$ . NNil c  $\models$  w)  $\sigma$ ))  $\wedge$ 
    ( $\forall i$ . enat i < nlength l  $\longrightarrow$  (nsubn (upd  $\sigma$  n (nmap ( $\lambda c$ . NNil c  $\models$  w)  $\sigma$ )) (nnth l i) (nnth l (Suc
i))  $\models$  f)))
by simp
have s3: ( $\exists l$ . nnth l 0 = 0  $\wedge$ 
  nfinite l  $\wedge$ 
  nidx l  $\wedge$ 
  enat (nlast l) = nlength  $\sigma \wedge$  nfinite  $\sigma \wedge$ 
  ( $\forall i$ . enat i < nlength l  $\longrightarrow$  (nsubn  $\sigma$  (nnth l i) (nnth l (Suc i))  $\models$  suform f n w)))  $\implies$ 
  ( $\exists l$ . nnth l 0 = 0  $\wedge$ 
    nfinite l  $\wedge$ 
    nidx l  $\wedge$ 
    enat (nlast l) = nlength (upd  $\sigma$  n (nmap ( $\lambda c$ . NNil c  $\models$  w)  $\sigma$ ))  $\wedge$ 
    nfinite (upd  $\sigma$  n (nmap ( $\lambda c$ . NNil c  $\models$  w)  $\sigma$ ))  $\wedge$ 
    ( $\forall i$ . enat i < nlength l  $\longrightarrow$  (nsubn (upd  $\sigma$  n (nmap ( $\lambda c$ . NNil c  $\models$  w)  $\sigma$ )) (nnth l i) (nnth l (Suc
i))  $\models$  f)))

```

$(\forall i. \text{enat } i < \text{nlength } l \longrightarrow (\text{nsbn } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)) \models f)))$
proof –
assume $\text{sa0}: (\exists l. \text{nnth } l \ 0 = 0 \wedge$
 $\text{nfinite } l \wedge$
 $\text{nidx } l \wedge$
 $\text{enat } (\text{nlast } l) = \text{nlength } \sigma \wedge \text{nfinite } \sigma \wedge$
 $(\forall i. \text{enat } i < \text{nlength } l \longrightarrow (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)) \models \text{suform } f \ n \ w)))$
show $(\exists l. \text{nnth } l \ 0 = 0 \wedge$
 $\text{nfinite } l \wedge$
 $\text{nidx } l \wedge$
 $\text{enat } (\text{nlast } l) = \text{nlength } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) \wedge$
 $\text{nfinite } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) \wedge$
 $(\forall i. \text{enat } i < \text{nlength } l \longrightarrow (\text{nsbn } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)) \models f)))$
proof –
obtain l **where** $s4: \text{nnth } l \ 0 = 0 \wedge$
 $\text{nfinite } l \wedge$
 $\text{nidx } l \wedge$
 $\text{enat } (\text{nlast } l) = \text{nlength } \sigma \wedge \text{nfinite } \sigma \wedge$
 $(\forall i. \text{enat } i < \text{nlength } l \longrightarrow (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)) \models \text{suform } f \ n \ w))$
using sa0 **by** blast
have $s5: \text{enat } (\text{nlast } l) = \text{nlength } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma))$
by $(\text{simp add: nlength-upd } s4)$
have $s6: \text{nfinite } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma))$
by $(\text{metis nlength-eq-enat-nfiniteD } s5)$
have $s7: \bigwedge i. \text{enat } i < \text{nlength } l \implies (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)) \models \text{suform } f \ n \ w)$
using $s4$ **by** blast
have $s8: \bigwedge i. \text{enat } i < \text{nlength } l \implies$
 $(\text{nsbn } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)) \models f)$
proof –
fix i
assume $\text{sa1}: \text{enat } i < \text{nlength } l$
show $(\text{nsbn } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)) \models f)$
proof –
have $s9: (\text{upd } (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)))) \models f)$
using $\text{schopstar-d.IH}[of \ n \ w \ (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))]$
using $\text{bvars.simps}(6) \ s4 \ \text{sa1} \ \text{schopstar-d.prem}(1) \ \text{schopstar-d.prem}(2) \ \text{schopstar-d.prem}(3)$ **by** blast

have $s10: (\text{nsbn } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) =$
 $\text{upd } (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) \ n \ (\text{nsbn } (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma) \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)))$
using $\text{nsbn-upd}[of \ \sigma \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma) \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)) \ n]$
by $(\text{metis Suc-n-not-le-n eSuc-enat enat-ord-simps}(1) \ \text{ileI1} \ \text{nidx-all-le-nlast} \ \text{nidx-less-eq} \ \text{nle-le-nlength-nmap } s4 \ \text{sa1})$
have $s11: (\text{nsbn } (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma) \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) =$
 $(\text{nmap } (\lambda c. \text{NNil } c \models w) \ (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))))$
by $(\text{simp add: ndropn-nmap nsbn-def1})$
have $s12: (\text{upd } (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))))$

```

(nnth l (Suc i)))) =
  (nsubn (upd σ n (nmap (λc. NNil c ⊨ w) σ)) (nnth l i) (nnth l (Suc i)))
  by (simp add: s10 s11)
  show ?thesis
  using s12 s9 by auto
qed
qed
show ?thesis
using s4 s5 s6 s8 by blast
qed
qed
have s13: (∃ l. nnth l 0 = 0 ∧
  nfinite l ∧
  nidx l ∧
  enat (nlast l) = nlength (upd σ n (nmap (λc. NNil c ⊨ w) σ)) ∧
  nfinite (upd σ n (nmap (λc. NNil c ⊨ w) σ)) ∧
  (∀ i. enat i < nlength l ⟶ (nsubn (upd σ n (nmap (λc. NNil c ⊨ w) σ)) (nnth l i) (nnth l (Suc
i)) ⊨ f))) ⟹
  (∃ l. nnth l 0 = 0 ∧
  nfinite l ∧
  nidx l ∧
  enat (nlast l) = nlength σ ∧ nfinite σ ∧
  (∀ i. enat i < nlength l ⟶ (nsubn σ (nnth l i) (nnth l (Suc i)) ⊨ suform f n w)))
proof -
  assume sa2: (∃ l. nnth l 0 = 0 ∧
  nfinite l ∧
  nidx l ∧
  enat (nlast l) = nlength (upd σ n (nmap (λc. NNil c ⊨ w) σ)) ∧
  nfinite (upd σ n (nmap (λc. NNil c ⊨ w) σ)) ∧
  (∀ i. enat i < nlength l ⟶ (nsubn (upd σ n (nmap (λc. NNil c ⊨ w) σ)) (nnth l i) (nnth l (Suc
i)) ⊨ f)))
  show (∃ l. nnth l 0 = 0 ∧
  nfinite l ∧
  nidx l ∧
  enat (nlast l) = nlength σ ∧ nfinite σ ∧
  (∀ i. enat i < nlength l ⟶ (nsubn σ (nnth l i) (nnth l (Suc i)) ⊨ suform f n w)))
  proof -
    obtain l where s14: nnth l 0 = 0 ∧
      nfinite l ∧
      nidx l ∧
      enat (nlast l) = nlength (upd σ n (nmap (λc. NNil c ⊨ w) σ)) ∧
      nfinite (upd σ n (nmap (λc. NNil c ⊨ w) σ)) ∧
      (∀ i. enat i < nlength l ⟶ (nsubn (upd σ n (nmap (λc. NNil c ⊨ w) σ)) (nnth l i) (nnth l (Suc
i)) ⊨ f))
    using sa2 by blast
    have s15: enat (nlast l) = nlength σ
    by (simp add: nlength-upd s14)
    have s16: nfinite σ
    by (metis nlength-eq-enat-nfiniteD s15)
    have s17: ⋀ i. enat i < nlength l ⟹

```

```

      (nsubn (upd σ n (nmap (λc. NNil c ⊨ w) σ)) (nnth l i) (nnth l (Suc i)) ⊨ f)
using s14 by blast
have s18: ∧i. enat i < nlength l ⇒ (nsubn σ (nnth l i) (nnth l (Suc i)) ⊨ suform f n w)
proof -
  fix i
  assume sa3: enat i < nlength l
  show (nsubn σ (nnth l i) (nnth l (Suc i)) ⊨ suform f n w)
  proof -
    have s19: (nsubn (nmap (λc. NNil c ⊨ w) σ) (nnth l i) (nnth l (Suc i))) =
      (nmap (λc. NNil c ⊨ w) (nsubn σ (nnth l i) (nnth l (Suc i))))
    by (simp add: ndropn-nmap nsubn-def1)
    have s20: (nsubn (upd σ n (nmap (λc. NNil c ⊨ w) σ)) (nnth l i) (nnth l (Suc i))) =
      upd (nsubn σ (nnth l i) (nnth l (Suc i))) n (nsubn (nmap (λc. NNil c ⊨ w) σ) (nnth l i)
      (nnth l (Suc i)))
    using nsubn-upd[of σ (nmap (λc. NNil c ⊨ w) σ) (nnth l i) (nnth l (Suc i)) n]
    by (metis Suc-n-not-le-n eSuc-enat enat-ord-simps(1) ileI1 linorder-le-cases nidx-all-le-nlast
      nidx-less-eq nlength-nmap s14 s15 sa3)
    have s21: (upd (nsubn σ (nnth l i) (nnth l (Suc i))) n (nmap (λc. NNil c ⊨ w) (nsubn σ (nnth l i)
      (nnth l (Suc i))))) =
      (nsubn (upd σ n (nmap (λc. NNil c ⊨ w) σ)) (nnth l i) (nnth l (Suc i)))
    using s19 s20 by presburger
    have s22: (nsubn (upd σ n (nmap (λc. NNil c ⊨ w) σ)) (nnth l i) (nnth l (Suc i)) ⊨ f)
    using s14 sa3 by blast
    have s23: (nsubn σ (nnth l i) (nnth l (Suc i)) ⊨ suform f n w)
    using bvars.simps(6) s21 s22 schopstar-d.IH schopstar-d.premis(1) schopstar-d.premis(2)
      schopstar-d.premis(3) by presburger
    show ?thesis
    by (simp add: s23)
  qed
qed
show ?thesis
using s14 s15 s16 s18 by blast
qed
qed
show ?thesis
using s1 s13 s2 s3 by blast
qed
next
case (omega-d f)
then show ?case
proof -
  have s1: (σ ⊨ suform (omega f) n w) =
    (∃ l. ¬ nfinite l ∧
      nnth l 0 = 0 ∧
      nidx l ∧
      (∀ i. enat (nnth l i) ≤ nlength σ) ∧
      (∀ i. nsubn σ (nnth l i) (nnth l (Suc i)) ⊨ suform f n w))
  by simp
  have s2: (upd σ n (nmap (λc. NNil c ⊨ w) σ) ⊨ omega f) =
    (∃ l. ¬ nfinite l ∧

```


$nnth\ l\ 0 = 0 \wedge$
 $nidx\ l \wedge$
 $(\forall i. enat\ (nnth\ l\ i) \leq nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma))) \wedge$
 $(\forall i. nsubn\ (upd\ \sigma\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models f))$
by *simp*
have *s3*: $(\exists l. \neg nfinite\ l \wedge$
 $nnth\ l\ 0 = 0 \wedge$
 $nidx\ l \wedge$
 $(\forall i. enat\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
 $(\forall i. nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models suform\ f\ n\ w)) \implies$
 $(\exists l. \neg nfinite\ l \wedge$
 $nnth\ l\ 0 = 0 \wedge$
 $nidx\ l \wedge$
 $(\forall i. enat\ (nnth\ l\ i) \leq nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma))) \wedge$
 $(\forall i. nsubn\ (upd\ \sigma\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models f))$
proof –
assume *sa0*: $(\exists l. \neg nfinite\ l \wedge$
 $nnth\ l\ 0 = 0 \wedge$
 $nidx\ l \wedge$
 $(\forall i. enat\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
 $(\forall i. nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models suform\ f\ n\ w))$
show $(\exists l. \neg nfinite\ l \wedge$
 $nnth\ l\ 0 = 0 \wedge$
 $nidx\ l \wedge$
 $(\forall i. enat\ (nnth\ l\ i) \leq nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma))) \wedge$
 $(\forall i. nsubn\ (upd\ \sigma\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models f))$
proof –
obtain *l* **where** *s4*: $\neg nfinite\ l \wedge$
 $nnth\ l\ 0 = 0 \wedge$
 $nidx\ l \wedge$
 $(\forall i. enat\ (nnth\ l\ i) \leq nlength\ \sigma) \wedge$
 $(\forall i. nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models suform\ f\ n\ w)$
using *sa0* **by** *blast*
have *s5*: $(\forall i. enat\ (nnth\ l\ i) \leq nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma)))$
by (*simp add: nlength-upd s4*)
have *s7*: $\bigwedge i. enat\ i < nlength\ l \implies (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models suform\ f\ n\ w)$
using *s4* **by** *blast*
have *s8*: $\bigwedge i. enat\ i < nlength\ l \implies$
 $(nsubn\ (upd\ \sigma\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models f)$
proof –
fix *i*
assume *sa1*: $enat\ i < nlength\ l$
show $(nsubn\ (upd\ \sigma\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models f)$
proof –
have *s9*: $(upd\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ (nsubn\ \sigma\ (nnth\ l\ i)$
 $(nnth\ l\ (Suc\ i)))) \models f)$
using *omega-d.IH*[*of* *n w (nsubn σ (nnth l i) (nnth l (Suc i)))*]
using *omega-d.prem1* *omega-d.prem2* *omega-d.prem3* *s7 sa1* **by** *force*
have *s10*: $(nsubn\ (upd\ \sigma\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma))\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) =$
 $upd\ (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))\ n\ (nsubn\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma)\ (nnth\ l\ i))$

```

(nnth l (Suc i)))
  using nsubn-upd[of  $\sigma$  (nmap ( $\lambda c. \text{NNil } c \models w$ )  $\sigma$ ) (nnth l i) (nnth l (Suc i)) n]
  by (metis Suc-n-not-le-n eSuc-enat enat-ord-simps(1) ileI1 nidx-less-eq nle-le nlength-nmap s4 sa1)
  have s11: (nsubn (nmap ( $\lambda c. \text{NNil } c \models w$ )  $\sigma$ ) (nnth l i) (nnth l (Suc i))) =
    (nmap ( $\lambda c. \text{NNil } c \models w$ ) (nsubn  $\sigma$  (nnth l i) (nnth l (Suc i))))
  by (simp add: ndropn-nmap nsubn-def1)
  have s12: (upd (nsubn  $\sigma$  (nnth l i) (nnth l (Suc i))) n (nmap ( $\lambda c. \text{NNil } c \models w$ ) (nsubn  $\sigma$  (nnth l i)
(nnth l (Suc i)))) =
    (nsubn (upd  $\sigma$  n (nmap ( $\lambda c. \text{NNil } c \models w$ )  $\sigma$ )) (nnth l i) (nnth l (Suc i)))
  by (simp add: s10 s11)
  show ?thesis
  using s12 s9 by auto
qed
qed
show ?thesis
by (metis enat-ile linorder-not-less nlength-eq-enat-nfiniteD s4 s5 s8)
qed
qed
have s13: ( $\exists l. \neg \text{nfinite } l \wedge$ 
  nnth l 0 = 0  $\wedge$ 
  nidx l  $\wedge$ 
  ( $\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma))) \wedge$ 
  ( $\forall i. \text{nsubn } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)) \models f$ )  $\implies$ 
  ( $\exists l. \neg \text{nfinite } l \wedge$ 
  nnth l 0 = 0  $\wedge$ 
  nidx l  $\wedge$ 
  ( $\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma$ )  $\wedge$ 
  ( $\forall i. \text{nsubn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)) \models \text{suform } f \ n \ w$ ))
proof -
  assume sa2: ( $\exists l. \neg \text{nfinite } l \wedge$ 
    nnth l 0 = 0  $\wedge$ 
    nidx l  $\wedge$ 
    ( $\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma))) \wedge$ 
    ( $\forall i. \text{nsubn } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)) \models f$ )
  show ( $\exists l. \neg \text{nfinite } l \wedge$ 
    nnth l 0 = 0  $\wedge$ 
    nidx l  $\wedge$ 
    ( $\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma$ )  $\wedge$ 
    ( $\forall i. \text{nsubn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)) \models \text{suform } f \ n \ w$ )
  proof -
    obtain l where s14:  $\neg \text{nfinite } l \wedge$ 
      nnth l 0 = 0  $\wedge$ 
      nidx l  $\wedge$ 
      ( $\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma))) \wedge$ 
      ( $\forall i. \text{nsubn } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)) \models f$ )
    using sa2 by blast
    have s15: ( $\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma$ )
    using nlength-upd s14 by auto
    have s17:  $\bigwedge i. (\text{nsubn } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)) \models f)$ 
    using s14 by blast

```

```

have s18:  $\bigwedge i. (nsubn \sigma (nnth\ l\ i) (nnth\ l\ (Suc\ i)) \models suform\ f\ n\ w)$ 
proof -
  fix i
  show  $(nsubn \sigma (nnth\ l\ i) (nnth\ l\ (Suc\ i)) \models suform\ f\ n\ w)$ 
proof -
  have s19:  $(nsubn (nmap (\lambda c. NNil\ c \models w) \sigma) (nnth\ l\ i) (nnth\ l\ (Suc\ i))) =$ 
     $(nmap (\lambda c. NNil\ c \models w) (nsubn \sigma (nnth\ l\ i) (nnth\ l\ (Suc\ i))))$ 
  by (simp add: ndropn-nmap nsubn-def1)
  have s20:  $(nsubn (upd\ \sigma\ n\ (nmap (\lambda c. NNil\ c \models w) \sigma)) (nnth\ l\ i) (nnth\ l\ (Suc\ i))) =$ 
     $upd\ (nsubn \sigma (nnth\ l\ i) (nnth\ l\ (Suc\ i)))\ n\ (nsubn (nmap (\lambda c. NNil\ c \models w) \sigma) (nnth\ l\ i)$ 
 $(nnth\ l\ (Suc\ i)))$ 
  using nsubn-upd[of  $\sigma\ (nmap (\lambda c. NNil\ c \models w) \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))\ n]$ 
  unfolding nsubn-def1
  by (metis Suc-ile-eq enat-ord-simps(1) le-SucI linorder-not-le nfinite-ntaken nidx-less-eq nlength-nmap
ntaken-all s14 s15)
  have s21:  $(upd\ (nsubn \sigma (nnth\ l\ i) (nnth\ l\ (Suc\ i)))\ n\ (nmap (\lambda c. NNil\ c \models w) (nsubn \sigma (nnth\ l\ i)$ 
 $(nnth\ l\ (Suc\ i)))) =$ 
     $(nsubn (upd\ \sigma\ n\ (nmap (\lambda c. NNil\ c \models w) \sigma)) (nnth\ l\ i) (nnth\ l\ (Suc\ i)))$ 
  using s19 s20 by presburger
  have s22:  $(nsubn (upd\ \sigma\ n\ (nmap (\lambda c. NNil\ c \models w) \sigma)) (nnth\ l\ i) (nnth\ l\ (Suc\ i)) \models f)$ 
  by (simp add: s17)
  have s23:  $(nsubn \sigma (nnth\ l\ i) (nnth\ l\ (Suc\ i)) \models suform\ f\ n\ w)$ 
  using bvars.simps(7) omega-d.IH omega-d.premis(1) omega-d.premis(2) omega-d.premis(3) s21 s22
by presburger
  show ?thesis
  by (simp add: s23)
qed
qed
show ?thesis
using s14 s15 s18 by blast
qed
qed
show ?thesis
using s1 s13 s2 s3 by blast
qed
next
case (exists-d x1 f)
then show ?case
proof -
  have e1:  $(\sigma \models suform\ (Ex\ x1.\ f)\ n\ w) = (\sigma \models Ex\ x1.\ suform\ f\ n\ w)$ 
  using suform.simps(8) by presburger
  have e2:  $n = x1 \implies ?thesis$ 
  using bvars.simps(8) exists-d.premis(1) by blast

  have e3:  $n \neq x1 \implies ?thesis$ 
  proof -
    assume ae0:  $n \neq x1$ 
    show ?thesis
    proof -
      have e4:  $n \notin bvars\ f$ 

```

```

using exists-d.premis(1) by auto
have e5: ( $\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma) \models \text{Ex } x1. f$ ) =
  ( $\exists \ l. \text{nlength } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) = \text{nlength } l \wedge$ 
    ( $\text{upd } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) \ x1 \ l \models f$ ))
by simp
have e6: ( $\bigwedge z. z \in \text{fvars } w \implies z \notin \text{bvars } f$ )
using exists-d.premis(2) by auto
have e60:  $x1 \notin \text{fvars } w$ 
using exists-d.premis(2) by auto
have e61:  $\bigwedge \sigma \ l. \text{nlength } l = \text{nlength } \sigma \implies (\sigma \models w) = ((\text{upd } \sigma \ x1 \ l) \models w)$ 
using e60 not-fvar-upd by blast
have e7: ( $\sigma \models \text{Ex } x1. \text{suform } f \ n \ w$ ) = ( $\exists \ l. \text{nlength } \sigma = \text{nlength } l \wedge (\text{upd } \sigma \ x1 \ l \models \text{suform } f \ n \ w)$ )
by simp
have e8: ( $\exists \ l. \text{nlength } \sigma = \text{nlength } l \wedge (\text{upd } \sigma \ x1 \ l \models \text{suform } f \ n \ w)$ )  $\implies$ 
  ( $\exists \ l. \text{nlength } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) = \text{nlength } l \wedge$ 
    ( $\text{upd } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) \ x1 \ l \models f$ ))
proof –
  assume ae1: ( $\exists \ l. \text{nlength } \sigma = \text{nlength } l \wedge (\text{upd } \sigma \ x1 \ l \models \text{suform } f \ n \ w)$ )
  show ( $\exists \ l. \text{nlength } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) = \text{nlength } l \wedge$ 
    ( $\text{upd } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) \ x1 \ l \models f$ ))
  proof –
    obtain l where e9:  $\text{nlength } \sigma = \text{nlength } l \wedge (\text{upd } \sigma \ x1 \ l \models \text{suform } f \ n \ w)$ 
    using ae1 by blast
    have e10: ( $\text{upd } (\text{upd } \sigma \ x1 \ l) \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ (\text{upd } \sigma \ x1 \ l)) \models f$ )
    using e4 e6 e9 exists-d.IH exists-d.premis(3) by auto
    have e11: ( $\text{upd } (\text{upd } \sigma \ x1 \ l) \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ (\text{upd } \sigma \ x1 \ l))) =$ 
      ( $\text{upd } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ (\text{upd } \sigma \ x1 \ l))) \ x1 \ l$ )
      by (simp add: ae0 e9 nlength-upd upd-swap)
    have e12:  $\text{nlength } \sigma = \text{nlength } (\text{nmap } (\lambda c. \text{NNil } c \models w) \ (\text{upd } \sigma \ x1 \ l))$ 
    by (simp add: e9 nlength-upd)
    have e120:  $\text{nlength } (\text{nmap } (\lambda c. \text{NNil } c \models w) \ (\text{upd } \sigma \ x1 \ l)) = \text{nlength } (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)$ 
    by (simp add: e12)
    have e121:  $\bigwedge i. i \leq \text{nlength } (\text{nmap } (\lambda c. \text{NNil } c \models w) \ (\text{upd } \sigma \ x1 \ l)) \implies$ 
      ( $\text{nnth } (\text{nmap } (\lambda c. \text{NNil } c \models w) \ (\text{upd } \sigma \ x1 \ l)) \ i =$ 
        ( $\text{nnth } (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma) \ i$ )
      )
    by (simp add: e60 e9 exists-d.premis(3) not-fvar-upd-nmap)
    have e122: ( $\text{nmap } (\lambda c. \text{NNil } c \models w) \ (\text{upd } \sigma \ x1 \ l) = \text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma$ )
    using e120 e121 nellist-eq-nnth-eq by blast
    show ?thesis
    using e10 e11 e122 e9 nlength-upd by auto
  qed
qed
have e13: ( $\exists \ l. \text{nlength } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) = \text{nlength } l \wedge$ 
  ( $\text{upd } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) \ x1 \ l \models f$ )  $\implies$ 
  ( $\exists \ l. \text{nlength } \sigma = \text{nlength } l \wedge (\text{upd } \sigma \ x1 \ l \models \text{suform } f \ n \ w)$ )
proof –
  assume ae1: ( $\exists \ l. \text{nlength } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) = \text{nlength } l \wedge$ 
    ( $\text{upd } (\text{upd } \sigma \ n \ (\text{nmap } (\lambda c. \text{NNil } c \models w) \ \sigma)) \ x1 \ l \models f$ )
  )
  show ( $\exists \ l. \text{nlength } \sigma = \text{nlength } l \wedge (\text{upd } \sigma \ x1 \ l \models \text{suform } f \ n \ w)$ )
proof –

```

obtain l **where** $e14$: $nlength\ (upd\ \sigma\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma)) = nlength\ l \wedge$
 $(upd\ (upd\ \sigma\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma))\ x1\ l \models f)$
using $ae1$ **by** *blast*
have $e15$: $nlength\ (nmap\ (\lambda c. NNil\ c \models w)\ (upd\ \sigma\ x1\ l)) = nlength\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma)$
using $e14$ *nlength-upd* **by** *auto*
have $e16$: $\bigwedge i. i \leq nlength\ (nmap\ (\lambda c. NNil\ c \models w)\ (upd\ \sigma\ x1\ l)) \implies$
 $(nnth\ (nmap\ (\lambda c. NNil\ c \models w)\ (upd\ \sigma\ x1\ l))\ i) =$
 $(nnth\ (nmap\ (\lambda c. NNil\ c \models w)\ \sigma)\ i)$
using $e14$ $e60$ *exists-d.premis(3)* *nlength-upd not-fvar-upd-nmap* **by** *force*
have $e17$: $(nmap\ (\lambda c. NNil\ c \models w)\ (upd\ \sigma\ x1\ l)) = (nmap\ (\lambda c. NNil\ c \models w)\ \sigma)$
using $e15$ $e16$ *nellist-eq-nnth-eq* **by** *blast*
have $e18$: $(upd\ (upd\ \sigma\ x1\ l)\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ (upd\ \sigma\ x1\ l))) =$
 $(upd\ (upd\ \sigma\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ (upd\ \sigma\ x1\ l)))\ x1\ l)$
using $ae0$ $e14$ *nlength-upd upd-swap* **by** *auto*
have $e19$: $(upd\ (upd\ \sigma\ x1\ l)\ n\ (nmap\ (\lambda c. NNil\ c \models w)\ (upd\ \sigma\ x1\ l))) \models f$
using $e14$ $e17$ $e18$ **by** *presburger*
have $e20$: $(upd\ \sigma\ x1\ l \models suform\ f\ n\ w)$
by (*simp add: e19 e4 e6 exists-d.IH exists-d.premis(3)*)
show *?thesis*
using $e14$ $e20$ *nlength-upd* **by** *auto*
qed
qed
show *?thesis*
using $e1$ $e13$ $e5$ $e7$ $e8$ **by** *blast*
qed
qed
show *?thesis*
using $e2$ $e3$ **by** *blast*
qed
qed

lemma *rename-qpitl*:

assumes $v1 \notin bvars\ f$

$v2 \notin bvars\ f$

$v1 \neq v2$

$v2 \notin fvars\ f$

shows $(\sigma \models Ex\ v1. f) = (\sigma \models Ex\ v2. suform\ f\ v1\ (\$v2))$

proof –

have 1 : $(\sigma \models Ex\ v1. f) = (\exists\ l. nlength\ l = nlength\ \sigma \wedge ((upd\ \sigma\ v1\ l) \models f))$

by *force*

have 2 : $(\sigma \models Ex\ v2. suform\ f\ v1\ (\$v2)) =$

$(\exists\ l. nlength\ l = nlength\ \sigma \wedge ((upd\ \sigma\ v2\ l) \models suform\ f\ v1\ (\$v2)))$

by *force*

have 3 : $(\exists\ l. nlength\ l = nlength\ \sigma \wedge ((upd\ \sigma\ v2\ l) \models suform\ f\ v1\ (\$v2))) \implies$
 $(\exists\ l. nlength\ l = nlength\ \sigma \wedge ((upd\ \sigma\ v1\ l) \models f))$

proof –

assume $a0$: $(\exists\ l. nlength\ l = nlength\ \sigma \wedge ((upd\ \sigma\ v2\ l) \models suform\ f\ v1\ (\$v2)))$

show $(\exists\ l. nlength\ l = nlength\ \sigma \wedge ((upd\ \sigma\ v1\ l) \models f))$

proof –

obtain l **where** 4 : $nlength\ l = nlength\ \sigma \wedge ((upd\ \sigma\ v2\ l) \models suform\ f\ v1\ (\$v2))$

```

using a0 by blast
have 5: ((upd  $\sigma$  v2 l)  $\models$  suform f v1 ($v2)) =
  (upd (upd  $\sigma$  v2 l) v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l))  $\models$  f)
using subst-suform[of v1 f $v2 (upd  $\sigma$  v2 l)]
by (simp add: assms(1) assms(2))
have 6: (upd (upd  $\sigma$  v2 l) v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l))) =
  (upd (upd  $\sigma$  v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l))) v2 l)
by (simp add: 4 assms(3) nlength-upd upd-swap)
have 8: ((upd (upd  $\sigma$  v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l))) v2 l)  $\models$  f) =
  ((upd  $\sigma$  v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l)))  $\models$  f)
by (metis 4 assms(4) nlength-nmap nlength-upd not-fvar-upd)
show ?thesis
by (metis 4 5 6 8 nlength-nmap nlength-upd)
qed
qed
have 9: ( $\exists$  l. nlength l = nlength  $\sigma$   $\wedge$  ((upd  $\sigma$  v1 l)  $\models$  f) )  $\implies$ 
  ( $\exists$  l. nlength l = nlength  $\sigma$   $\wedge$  ((upd  $\sigma$  v2 l)  $\models$  suform f v1 ($v2)))
proof -
assume a1: ( $\exists$  l. nlength l = nlength  $\sigma$   $\wedge$  ((upd  $\sigma$  v1 l)  $\models$  f) )
show ( $\exists$  l. nlength l = nlength  $\sigma$   $\wedge$  ((upd  $\sigma$  v2 l)  $\models$  suform f v1 ($v2)))
proof -
obtain l where 10: nlength l = nlength  $\sigma$   $\wedge$  ((upd  $\sigma$  v1 l)  $\models$  f)
using a1 by blast
have 11: nlength l = nlength (upd  $\sigma$  v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l)))
by (simp add: 10 nlength-upd)
have 12: v2  $\notin$  fvars f
by (simp add: assms(4))
have 13: (upd  $\sigma$  v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l))  $\models$  f) =
  ((upd (upd  $\sigma$  v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l))) v2 l)  $\models$  f)
using not-fvar-upd[of v2 f l (upd  $\sigma$  v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l))) ]
11 12 by blast
have 14: nlength (upd  $\sigma$  v1 l) =
  nlength (upd  $\sigma$  v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l)) )
by (simp add: 10 nlength-upd)
have 15:  $\bigwedge i . i \leq \text{nlength (upd } \sigma \text{ v1 l)} \implies$ 
  (nnth (upd  $\sigma$  v1 l) i) =
  (nnth (upd  $\sigma$  v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l)) ) i)
using nnth-upd[of  $\sigma$  (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l)) - v1] by auto
  (metis 10 Diff-iff insertCI nfirst-eq-nnth-zero nlength-upd nnth-NNil nnth-upd)+
have 16: (upd  $\sigma$  v1 l) =
  (upd  $\sigma$  v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l)) )
by (simp add: 14 15 nellist-eq-nnth-eq)
have 17: ((upd  $\sigma$  v2 l)  $\models$  suform f v1 ($v2)) =
  (upd (upd  $\sigma$  v2 l) v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l))  $\models$  f)
using assms(1) assms(2) subst-suform by fastforce
have 18: (upd (upd  $\sigma$  v2 l) v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l))) =
  (upd (upd  $\sigma$  v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l))) v2 l)
by (simp add: 10 assms(3) nlength-upd upd-swap)
have 19: ((upd (upd  $\sigma$  v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l))) v2 l)  $\models$  f) =
  ((upd  $\sigma$  v1 (nmap ( $\lambda c$ . NNil c  $\models$  $v2) (upd  $\sigma$  v2 l)))  $\models$  f)

```

```

    using 13 by auto
    show ?thesis
    using 10 13 16 17 18 by auto
  qed
qed
show ?thesis
using 1 2 3 9 by blast
qed

```

```

lemma state-qpitl-list-defs:
  state-qpitl-list L  $\longleftrightarrow$  ( $\forall i < \text{length } L. \text{state-qpitl } (L!i)$ )
proof (induct L)
case Nil
then show ?case unfolding state-qpitl-list-def by simp
next
case (Cons a L)
then show ?case
  proof -
    have 1: state-qpitl-list (a # L)  $\longleftrightarrow$  state-qpitl a  $\wedge$  state-qpitl-list L
      unfolding state-qpitl-list-def by simp
    have 2: ( $\forall i < \text{length } (a \# L). \text{state-qpitl } ((a \# L)!i)$ )  $\longleftrightarrow$ 
      state-qpitl ((a # L)!0)  $\wedge$ 
      ( $\forall i. 0 < i \wedge i < \text{length } (a \# L) \longrightarrow \text{state-qpitl } ((a \# L)!i)$ )
      by (metis length-greater-0-conv list.discI zero-less-iff-neq-zero)
    have 3: state-qpitl ((a # L)!0)  $\longleftrightarrow$  state-qpitl a
      by simp
    have 4: ( $\forall i. 0 < i \wedge i < \text{length } (a \# L) \longrightarrow \text{state-qpitl } ((a \# L)!i)$ )  $\longleftrightarrow$ 
      ( $\forall i < \text{length } (L). \text{state-qpitl } (L!i)$ )
      by auto
    show ?thesis
      using 1 2 3 4 local.Cons by presburger
  qed
qed

```

```

lemma big-ior-defs :
  ( $\sigma \models (\text{big-ior } F)$ )  $\longleftrightarrow$  ( $\exists i < \text{length } F. (\sigma \models F ! i)$ )
proof (induct F)
case Nil
then show ?case unfolding big-ior-def by simp
next
case (Cons a F)
then show ?case
  proof -
    have 1: ( $\sigma \models (\text{big-ior } (a \# F))$ )  $\longleftrightarrow$  ( $\sigma \models a \text{ ior } (\text{big-ior } F)$ )
      unfolding big-ior-def by simp
    have 2: ( $\exists i < \text{length } (a \# F). \sigma \models (a \# F) ! i$ )  $\longleftrightarrow$ 
      ( $\sigma \models (a \# F) ! 0$ )  $\vee$ 

```

```

      (∃ i. 0 < i ∧ i < length (a # F) ∧ (σ ⊨ (a # F) ! i))
    by (metis length-greater-0-conv neq-Nil-conv zero-less-iff-neq-zero)
  have 3: (σ ⊨ (a # F) ! 0) = (σ ⊨ a)
    by auto
  have 4: (∃ i. 0 < i ∧ i < length (a # F) ∧ (σ ⊨ (a # F) ! i)) ⟷
    (∃ i < length (F). (σ ⊨ (F) ! i))
    by auto
    (metis Suc-le-eq diff-Suc-Suc diff-zero gr0-conv-Suc less-Suc-eq-le)
  show ?thesis
  using 1 2 4 local.Cons by force
qed
qed

```

```

lemma big-iand-defs :
  (σ ⊨ (big-iand F)) ⟷ (∀ i < length F. (σ ⊨ F ! i))
proof (induct F)
case Nil
then show ?case unfolding big-iand-def by simp
next
case (Cons a F)
then show ?case
proof -
  have 1: (σ ⊨ (big-iand (a # F))) ⟷ (σ ⊨ a iand (big-iand F))
    unfolding big-iand-def by simp
  have 2: (∀ i < length (a # F). σ ⊨ (a # F) ! i) ⟷
    (σ ⊨ (a # F) ! 0) ∧
    (∀ i. 0 < i ∧ i < length (a # F) ⟶ (σ ⊨ (a # F) ! i))
    by (metis length-greater-0-conv neq-Nil-conv zero-less-iff-neq-zero)
  have 3: (σ ⊨ (a # F) ! 0) = (σ ⊨ a)
    by auto
  have 4: (∀ i. 0 < i ∧ i < length (a # F) ⟶ (σ ⊨ (a # F) ! i)) ⟷
    (∀ i < length (F). (σ ⊨ (F) ! i))
    by auto
  show ?thesis
  using 1 2 4 local.Cons by force
qed
qed

```

```

lemma big-ior-map-iand-zip :
  assumes length F = length G
  shows (σ ⊨ big-ior (map2 iand-d F G)) ⟷
    (∃ i < length F. (σ ⊨ F ! i iand G ! i))
  using assms
  proof (induct F arbitrary: G)
  case Nil
  then show ?case by (simp add: big-ior-defs)
  next
  case (Cons a F)

```



```

then show ?case
  proof (cases G=[])
  case True
  then show ?thesis using Cons.prem by auto
  next
  case False
  then show ?thesis using Cons by (auto simp add: big-ior-defs)
  qed
qed

```

```

lemma big-iand-map-iimp-zip :
assumes length F = length G
shows  $(\sigma \models \text{big-iand} (\text{map2 iimp-d } F \ G)) \longleftrightarrow$ 
 $(\forall i < \text{length } F. (\sigma \models F ! i \text{ iimp } G ! i))$ 
using assms
proof (induct F arbitrary: G)
case Nil
then show ?case by (simp add: big-iand-defs)
next
case (Cons a F)
then show ?case
  proof (cases G=[])
  case True
  then show ?thesis using Cons.prem by auto
  next
  case False
  then show ?thesis using Cons by (auto simp add: big-iand-defs)
  qed
qed

```

```

lemma map2-iand-defs :
assumes length F = length G
shows  $(\exists i < \text{length } G. \sigma \models \text{map2 iand-d } F \ G ! i) \longleftrightarrow$ 
 $(\exists i < \text{length } G. (\sigma \models F ! i) \wedge (\sigma \models G ! i))$ 
using assms
proof (induct G arbitrary: F)
case Nil
then show ?case by simp
next
case (Cons a G)
then show ?case
  proof (cases F=[])
  case True
  then show ?thesis using Cons.prem by auto
  next
  case False
  then show ?thesis using Cons by auto
  qed
qed

```

```

lemma map2-iand-inot-defs :
  assumes length F = length G
  shows  $(\exists i < \text{length } G. \sigma \models \text{map2 } (\lambda x y. x \text{ iand inot } y) F G ! i) \longleftrightarrow$ 
     $(\exists i < \text{length } G. (\sigma \models F ! i) \wedge \neg(\sigma \models G ! i))$ 
using assms
proof (induct G arbitrary: F)
case Nil
then show ?case by simp
next
case (Cons a G)
then show ?case
  proof (cases F=[])
  case True
  then show ?thesis using Cons.prems by auto
  next
  case False
  then show ?thesis using Cons by auto
qed
qed

```

4.2.3 Validity

```

definition valid :: qpitl  $\Rightarrow$  bool (( $\vdash$  -) 5)
where  $(\vdash f) = (\forall \sigma. (\sigma \models f))$ 

```

```

lemma itl-valid [simp] :
   $(\vdash f) = (\forall \sigma. (\sigma \models f))$ 
by (simp add: valid-def)

```

```

lemma itl-ieq:
   $(\vdash f \text{ ieqv } g) = (\forall \sigma. (\sigma \models f) = (\sigma \models g))$ 
by (auto simp add: ieqv-d-def iand-d-def ior-d-def inot-d-def)

```

```

lemma big-ior-imp-subset-eq:
  assumes set L1  $\subseteq$  set L2
  shows  $\vdash \text{big-ior } L1 \text{ imp big-ior } L2$ 
using assms
proof (induct rule: list-induct2')
case 1
then show ?case by simp
next
case (2 x xs)
then show ?case by simp
next
case (3 y ys)
then show ?case by (simp add: big-ior-defs)
next
case (4 x xs y ys)
then show ?case

```

```

proof (cases x=y)
case True
then show ?thesis using 4 by (auto simp add: big-ior-defs)
(metis diff-Suc-1 in-set-conv-nth length-Cons less-Suc-eq-0-disj list.simps(15) nth-Cons' subsetD)
next
case False
then show ?thesis using 4 by (auto simp add: big-ior-defs)
(metis 4.prem1 in-set-conv-nth length-Cons subsetD)
qed
qed

```

```

lemma big-ior-ieqv-set-eq:
assumes set L1 = set L2
shows  $\vdash$  big-ior L1 ieqv big-ior L2
using assms
by (metis big-ior-iimp-subset-eq dual-order.refl iand-defs ieqv-d-def valid-def)

```

```

lemma big-ior-ieqv-remdups:
shows  $\vdash$  big-ior L ieqv big-ior (remdups L)
by (metis big-ior-ieqv-set-eq set-remdups)

```

```

lemma big-iand-iimp-subset-eq:
assumes set L1  $\subseteq$  set L2
shows  $\vdash$  big-iand L2 iimp big-iand L1
using assms
proof (induct rule: list-induct2')
case 1
then show ?case by simp
next
case (2 x xs)
then show ?case by (simp add: big-iand-defs)
next
case (3 y ys)
then show ?case by simp
next
case (4 x xs y ys)
then show ?case
proof (cases x=y)
case True
then show ?thesis using 4 by (auto simp add: big-iand-defs)
(metis (mono-tags, lifting) 4.prem1 all-nth-imp-all-set length-Cons list-ball-nth subsetD)
next
case False
then show ?thesis using 4 by (auto simp add: big-iand-defs)
(metis (no-types, opaque-lifting) 4.prem1 all-nth-imp-all-set insert-absorb insert-subset length-Cons nth-mem)
qed
qed

```

```

lemma big-iand-ieqv-set-eq:

```

assumes $set\ L1 = set\ L2$
shows $\vdash big_iand\ L1\ ieqv\ big_iand\ L2$
using *assms*
by (*metis big-iand-iimp-subset-eq dual-order.refl iand-defs ieqv-d-def valid-def*)

lemma *big-iand-ieqv-remdups*:
 $\vdash big_iand\ L\ ieqv\ big_iand\ (remdups\ L)$
by (*metis big-iand-ieqv-set-eq set-remdups*)

lemma *big-ior-empty*:
 $(\vdash big_ior\ []) \longleftrightarrow (\vdash ifalse)$
by (*auto simp add: big-ior-defs*)

lemma *big-ior-Cons*:
 $(\vdash big_ior\ (f\ \# \ L)) \longleftrightarrow (\vdash f\ ior\ (big_ior\ L))$
proof –
have 1: $(\vdash big_ior\ (f\ \# \ L)) \implies (\vdash f\ ior\ (big_ior\ L))$
by (*auto simp add: big-ior-defs*)
(metis diff-Suc-1 gr0-conv-Suc less-Suc-eq less-imp-diff-less nth-Cons')
have 2: $(\vdash f\ ior\ (big_ior\ L)) \implies (\vdash big_ior\ (f\ \# \ L))$
by (*auto simp add: big-ior-defs fastforce*)
show ?thesis **using** 1 2 **by** *auto*
qed

lemma *big-ior-Cons-alt*:
 $\vdash big_ior\ (f\ \# \ L)\ ieqv\ f\ ior\ big_ior\ L$
unfolding *big-ior-def*
by *auto*

lemma *big-ior-1*:
assumes $\bigwedge i. i < length\ F \implies (\vdash F\ !\ i\ ieqv\ G\ !\ i)$
 $length\ F = length\ G$
shows $\vdash (big_ior\ F)\ ieqv\ (big_ior\ G)$
using *assms*
by (*auto simp add: big-ior-defs*)

lemma *big-iand-1*:
assumes $\bigwedge i. i < length\ F \implies (\vdash F\ !\ i\ ieqv\ G\ !\ i)$
 $length\ F = length\ G$
shows $\vdash (big_iand\ F)\ ieqv\ (big_iand\ G)$
using *assms*
by (*auto simp add: big-iand-defs*)

lemma *big-ior-big-iand-1*:

$\vdash (\text{inot } (\text{big-ior } F)) \text{ ieqv } (\text{big-iand } (\text{map } (\lambda x. \text{inot } x) F))$
by (*auto simp add: big-ior-defs big-iand-defs*)

lemma *big-ior-big-iand-2*:

$\vdash (\text{inot } (\text{big-iand } F)) \text{ ieqv } (\text{big-ior } (\text{map } (\lambda x. \text{inot } x) F))$
by (*auto simp add: big-ior-defs big-iand-defs*)

lemma *big-ior-append*:

$(\vdash (\text{big-ior } (L1 @ L2)) \text{ ieqv } (\text{big-ior } L1 \text{ ior } (\text{big-ior } L2)))$
proof (*induct L1 arbitrary: L2*)
case Nil
then show *?case* **by** (*simp add: big-ior-defs*)
next
case (*Cons a L1a*)
then show *?case*
 proof –
 have 1: $\vdash \text{big-ior } ((a \# L1a) @ L2) \text{ ieqv } a \text{ ior } ((\text{big-ior } (L1a @ L2)))$
 using *big-ior-Cons-alt* **by** *auto*
 have 2: $\vdash \text{big-ior } (L1a @ L2) \text{ ieqv } \text{big-ior } L1a \text{ ior } \text{big-ior } L2$
 using *local.Cons* **by** *blast*
 have 3: $\vdash a \text{ ior } ((\text{big-ior } (L1a @ L2))) \text{ ieqv } a \text{ ior } (\text{big-ior } L1a \text{ ior } \text{big-ior } L2)$
 using 1 2 **by** *auto*
 have 4: $\vdash \text{big-ior } (a \# L1a) \text{ ieqv } a \text{ ior } (\text{big-ior } L1a)$
 using *big-ior-Cons-alt* **by** *blast*
 have 5: $\vdash a \text{ ior } (\text{big-ior } L1a \text{ ior } \text{big-ior } L2) \text{ ieqv } \text{big-ior } (a \# L1a) \text{ ior } \text{big-ior } L2$
 using 4 **by** *auto*
 show *?thesis* **using** 1 3 5 **by** *force*
 qed
qed

lemma *state-qpitl-list-append*:

shows $\text{state-qpitl-list } (L @ L1) \longleftrightarrow \text{state-qpitl-list } L \wedge \text{state-qpitl-list } L1$
proof (*induct L arbitrary: L1*)
case Nil
then show *?case*
by (*simp add: state-qpitl-list-defs*)
next
case (*Cons a L*)
then show *?case*
 proof –
 have 1: $\text{state-qpitl-list } ((a \# L) @ L1) \longleftrightarrow \text{state-qpitl } a \wedge \text{state-qpitl-list } (L @ L1)$
 unfolding *state-qpitl-list-def* **by** *auto*
 have 2: $\text{state-qpitl-list } (L @ L1) \longleftrightarrow \text{state-qpitl-list } L \wedge \text{state-qpitl-list } L1$
 by (*simp add: local.Cons*)
 have 3: $\text{state-qpitl } a \wedge \text{state-qpitl-list } L \wedge \text{state-qpitl-list } L1 \longleftrightarrow$
 $(\text{state-qpitl-list } (a \# L) \wedge \text{state-qpitl-list } L1)$
 by (*simp add: state-qpitl-list-def*)
 show *?thesis* **using** 1 3 *local.Cons* **by** *auto*
 qed

qed

lemma *map2-map*:

assumes $\text{length } L1 = \text{length } L2$

shows $\text{map2 } f \ (\text{map } g \ L1) \ (\text{map } h \ L2) = \text{map2 } (\lambda x \ y. f \ (g \ x) \ (h \ y)) \ L1 \ L2$

using *assms*

proof (*induct L1 L2 rule:list-induct2*)

case *Nil*

then show *?case* **by** *simp*

next

case (*Cons x xs y ys*)

then show *?case* **by** *auto*

qed

lemma *map2-map-a*:

assumes $\text{length } L1 = \text{length } L2$

shows $(\text{map2 } (\lambda x \ y. x \ \text{iand} \ \text{next } y) \ (\text{map } (\lambda x1. x \ \text{iand} \ x1) \ L1) \ (\text{map } (\lambda y1. y \ \text{ior} \ y1) \ L2)) =$
 $(\text{map2 } (\lambda x1 \ y1. (x \ \text{iand} \ x1) \ \text{iand} \ \text{next } (y \ \text{ior} \ y1)) \ L1 \ L2)$

using *assms map2-map* **by** *blast*

lemma *map2-map-a-a*:

assumes $\text{length } L1 = \text{length } L2$

shows $(\text{map2 } (\lambda x \ y. x \ \text{iand} \ \text{next } y) \ (\text{map } (\lambda x1. x \ \text{iand} \ x1) \ L1) \ (\text{map } (\lambda y1. (\text{if } y = \text{ifalse then } y1 \ \text{else } y \ \text{ior} \ y1)) \ L2)) =$

$(\text{map2 } (\lambda x1 \ y1. (x \ \text{iand} \ x1) \ \text{iand} \ \text{next } ((\text{if } y = \text{ifalse then } y1 \ \text{else } y \ \text{ior} \ y1))) \ L1 \ L2)$

using *assms map2-map* **by** *blast*

lemma *map2-map-b*:

assumes $\text{length } L1 = \text{length } L2$

shows $(\text{map2 } (\lambda x \ y. x \ \text{iand} \ \text{next } y) \ (\text{map } (\lambda x1. (\text{inot } x) \ \text{iand} \ x1) \ L1) \ L2) =$
 $(\text{map2 } (\lambda x1 \ y1. ((\text{inot } x) \ \text{iand} \ x1) \ \text{iand} \ \text{next } y1) \ L1 \ L2)$

using *assms map2-map*[*of L1 L2* ($\lambda x \ y. x \ \text{iand} \ \text{next } y$) ($\lambda x1. (\text{inot } x) \ \text{iand} \ x1$) *id*]

by *auto*

lemma *big-ior-big-ior-map2*:

assumes $\text{length } L1 = \text{length } L2$

$\vdash \text{big-ior } L1$

shows $\vdash \text{big-ior } (\text{map2 } (\lambda x1 \ y1. x1) \ L1 \ L2)$

using *assms*

proof (*induct L1 L2 rule:list-induct2*)

case *Nil*

then show *?case* **by** *simp*

next

case (*Cons x xs y ys*)

then show *?case*

by (*metis fst-def list.size(4) map-eq-conv map-fst-zip*)

qed

lemma *big-ior-split-off*:

assumes *length L1 = length L2*

(\vdash *big-ior L1*)

shows \vdash (*big-ior* (*map2* ($\lambda x1\ y1.$ (*x iand x1*) *iand* (*next* (*y ior y1*)))) *L1 L2*)) *ior*
(*big-ior* (*map2* ($\lambda x1\ y1.$ (*inot x iand x1*) *iand next y1*) *L1 L2*)) *ieqv*
(*x iand next y*) *ior*
(*big-ior* (*map2* ($\lambda x\ y.$ *x iand next y*) *L1 L2*))

proof –

have 1: \vdash (*big-ior* (*map2* ($\lambda x1\ y1.$ (*x iand x1*) *iand* (*next* (*y ior y1*)))) *L1 L2*)) *ieqv*
(*x iand* (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand* (*next* (*y ior y1*)))) *L1 L2*))

by (*auto simp add: big-ior-defs*)

have 2: \vdash (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand* (*next* (*y ior y1*)))) *L1 L2*)) *ieqv*
((*next y iand* (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1*) *L1 L2*)))
ior
(*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand next* (*y1*)) *L1 L2*)))

by (*auto simp add: big-ior-defs*)

have 3: \vdash (*big-ior* (*map2* ($\lambda x1\ y1.$ (*x iand x1*) *iand* (*next* (*y ior y1*)))) *L1 L2*)) *ieqv*
(*x iand* (*next y iand* *big-ior* (*map2* ($\lambda x1\ y1.$ *x1*) *L1 L2*))
ior
(*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand next* (*y1*)) *L1 L2*))))

by (*auto simp add: big-ior-defs*)

have 4: \vdash *big-ior* (*map2* ($\lambda x1\ y1.$ *x1*) *L1 L2*)

using *assms big-ior-big-ior-map2* **by** *blast*

have 5: \vdash (*x iand* (*next y iand* *big-ior* (*map2* ($\lambda x1\ y1.$ *x1*) *L1 L2*))
ior

(*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand next* (*y1*)) *L1 L2*))) *ieqv*

(*x iand next y*) *ior* (*x iand* (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand next* (*y1*)) *L1 L2*)))

using 4 **by** *auto*

have 6: \vdash (*big-ior* (*map2* ($\lambda x1\ y1.$ (*inot x iand x1*) *iand next y1*) *L1 L2*)) *ieqv*
(*inot x iand* (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand* (*next* (*y1*))) *L1 L2*)))

by (*auto simp add: big-ior-defs*)

have 7: \vdash (*x iand* (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand next* (*y1*)) *L1 L2*))) *ior*
(*inot x iand* (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand next* (*y1*)) *L1 L2*))) *ieqv*
(*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand next* (*y1*)) *L1 L2*)))

by (*auto simp add: big-ior-defs*)

have 8 : \vdash (*big-ior* (*map2* ($\lambda x1\ y1.$ (*x iand x1*) *iand* (*next* (*y ior y1*)))) *L1 L2*)) *ior*
(*big-ior* (*map2* ($\lambda x1\ y1.$ (*inot x iand x1*) *iand next y1*) *L1 L2*)) *ieqv*
(*x iand next y*) *ior*
(*x iand* (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand next* (*y1*)) *L1 L2*))) *ior*
(*inot x iand* (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand next* (*y1*)) *L1 L2*)))

using 5 6 3 **by** *auto*

show *?thesis* **using** 7 8 **by** *auto*

qed

lemma *big-ior-split-off-a*:

assumes *length L1 = length L2*

(\vdash *big-ior L1*)

shows \vdash (*big-ior* (*map2* ($\lambda x1\ y1.$ (*x iand* *x1*) *iand* (*next* ((*if* *y = ifalse* *then* *y1* *else* *y ior* *y1*)))) *L1 L2*))
ior
(*big-ior* (*map2* ($\lambda x1\ y1.$ (*inot* *x iand* *x1*) *iand* *next* *y1*) *L1 L2*)) *ieqv*
(*x iand* *next* *y*) *ior*
(*big-ior* (*map2* ($\lambda x\ y.$ *x iand* *next* *y*) *L1 L2*))
proof –
have 1: \vdash (*big-ior* (*map2* ($\lambda x1\ y1.$ (*x iand* *x1*) *iand* (*next* ((*if* *y = ifalse* *then* *y1* *else* *y ior* *y1*)))) *L1 L2*))
ieqv
(*x iand* (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand* (*next* ((*if* *y = ifalse* *then* *y1* *else* *y ior* *y1*)))) *L1 L2*)))
by (*auto simp add: big-ior-defs*)
have 2: \vdash (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand* (*next* ((*if* *y = ifalse* *then* *y1* *else* *y ior* *y1*)))) *L1 L2*)) *ieqv*
((*next* *y iand* (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1*) *L1 L2*)))
ior
(*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand* *next* (*y1*)) *L1 L2*)))
proof (*cases y=ifalse*)
case *True*
then show ?thesis **by** *simp*
next
case *False*
then show ?thesis **by** (*auto simp add: big-ior-defs*)
qed
have 3: \vdash (*big-ior* (*map2* ($\lambda x1\ y1.$ (*x iand* *x1*) *iand* (*next* (*if* *y = ifalse* *then* *y1* *else* *y ior* *y1*)))) *L1 L2*))
ieqv
(*x iand* (*next* *y iand* *big-ior* (*map2* ($\lambda x1\ y1.$ *x1*) *L1 L2*))
ior
(*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand* *next* (*y1*)) *L1 L2*))))
proof (*cases y=ifalse*)
case *True*
then show ?thesis **by** (*auto simp add: big-ior-defs*)
next
case *False*
then show ?thesis **by** (*auto simp add: big-ior-defs*)
qed
have 4: \vdash *big-ior* (*map2* ($\lambda x1\ y1.$ *x1*) *L1 L2*)
using *assms big-ior-big-ior-map2* **by** *blast*
have 5: \vdash (*x iand* (*next* *y iand* *big-ior* (*map2* ($\lambda x1\ y1.$ *x1*) *L1 L2*))
ior
(*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand* *next* (*y1*)) *L1 L2*)))) *ieqv*
(*x iand* *next* *y*) *ior* (*x iand* (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand* *next* (*y1*)) *L1 L2*)))
using 4 **by** *auto*
have 6: \vdash (*big-ior* (*map2* ($\lambda x1\ y1.$ (*inot* *x iand* *x1*) *iand* *next* *y1*) *L1 L2*)) *ieqv*
(*inot* *x iand* (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand* (*next* (*y1*)) *L1 L2*)))
by (*auto simp add: big-ior-defs*)
have 7: \vdash (*x iand* (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand* *next* (*y1*)) *L1 L2*))) *ior*
(*inot* *x iand* (*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand* *next* (*y1*)) *L1 L2*))) *ieqv*
(*big-ior* (*map2* ($\lambda x1\ y1.$ *x1 iand* *next* (*y1*)) *L1 L2*))
by (*auto simp add: big-ior-defs*)
have 8: \vdash (*big-ior* (*map2* ($\lambda x1\ y1.$ (*x iand* *x1*) *iand* (*next* (*if* *y = ifalse* *then* *y1* *else* *y ior* *y1*)))) *L1 L2*))
ior
(*big-ior* (*map2* ($\lambda x1\ y1.$ (*inot* *x iand* *x1*) *iand* *next* *y1*) *L1 L2*)) *ieqv*


```

      (x iand next y) ior
      (x iand (big-ior (map2 (λx1 y1. x1 iand next ( y1)) L1 L2 ))) ior
      (inot x iand (big-ior (map2 (λx1 y1. x1 iand next ( y1)) L1 L2 )))
    using 5 6 3 by auto
  show ?thesis using 7 8 by auto
qed

```

```

lemma state-qpitl-list-map:
  assumes state-qpitl f
    state-qpitl-list L
  shows state-qpitl-list (map (λ x. f iand x) L)
using assms
proof (induct L)
case Nil
then show ?case by simp
next
case (Cons a L)
then show ?case
  proof -
    have 1: state-qpitl-list (map (λx. f iand x) (a # L)) ⟷
      state-qpitl ( f iand a ) ∧ state-qpitl-list ((map (λx. f iand x) L))
    unfolding state-qpitl-list-def by auto
    have 2: state-qpitl-list (a # L) ⟷ state-qpitl a ∧ state-qpitl-list L
    unfolding state-qpitl-list-def by auto
    have 3: state-qpitl ( f iand a )
    unfolding iand-d-def inot-d-def ior-d-def
    using 2 Cons.prem1(2) assms(1) by auto
    show ?thesis
    using 1 2 3 Cons.hyps Cons.prem1(2) assms(1) by blast
  qed
qed

```

```

lemma state-qpitl-list-map-alt:
  assumes state-qpitl f
    state-qpitl-list L
  shows state-qpitl-list (map (λ x. x iand f) L)
using assms
proof (induct L)
case Nil
then show ?case by simp
next
case (Cons a L)
then show ?case
  proof -
    have 1: state-qpitl-list (map (λx. x iand f) (a # L)) ⟷
      state-qpitl ( a iand f ) ∧ state-qpitl-list ((map (λx. x iand f) L))
    unfolding state-qpitl-list-def by auto
    have 2: state-qpitl-list (a # L) ⟷ state-qpitl a ∧ state-qpitl-list L
    unfolding state-qpitl-list-def by auto
  qed

```

```

have 3: state-qpitl ( a iand f)
  unfolding iand-d-def inot-d-def ior-d-def
  using 2 Cons.premis(2) assms(1) by auto
show ?thesis
using 1 2 3 Cons.hyps Cons.premis(2) assms(1) by blast
qed
qed

```

```

lemma map-fst-filter-sat:
assumes length L1 = length L2
shows map fst (filter (λ (x,y). (∃ σ. σ ⊨ x)) (zip L1 L2)) =
  filter (λx. (∃ σ. σ ⊨ x)) L1
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by simp
next
case (Cons x xs y ys)
then show ?case
  proof (cases xs = [] ∧ ys = [] )
  case True
  then show ?thesis by simp
  next
  case False
  then show ?thesis
    using Cons by auto
  qed
qed
qed

```

```

lemma big-ior-ieqv-filter-sat:
⊢ big-ior L ieqv big-ior ( filter (λx. (∃ σ. σ ⊨ x)) L)
proof (induct L)
case Nil
then show ?case by simp
next
case (Cons a L)
then show ?case
using big-ior-Cons-alt by auto
qed

```

```

lemma total-filter-sat:
assumes ⊢ big-ior L
shows ⊢ big-ior ( filter (λx. (∃ σ. σ ⊨ x)) L)
using assms
using big-ior-ieqv-filter-sat by auto

```

```

lemma big-ior-filter-sat-equiv:

```

```

assumes  $length\ L1 = length\ L2$ 
shows  $\vdash big\_ior\ (\ map\ (\lambda\ (x,y)\ .\ x\ iand\ (next\ y))$ 
       $(filter\ (\lambda\ (x,y)\ .\ (\exists\ \sigma.\ \sigma \models x))\ (zip\ L1\ L2)))$ 
       $ieqv$ 
       $big\_ior\ (map2\ (\lambda\ x\ y.\ x\ iand\ (next\ y))\ L1\ L2)$ 
using assms
proof (induct  $L1\ L2$  rule: list-induct2)
case Nil
then show ?case by (simp add: big-ior-def)
next
case (Cons  $x\ xs\ y\ ys$ )
then show ?case
  proof (cases  $xs = [] \wedge ys = []$ )
  case True
  then show ?thesis using Cons
    unfolding big-ior-def big-iand-def
    by simp
  next
  case False
  then show ?thesis
    proof –
    have  $1: length\ xs = length\ ys$ 
      by (simp add: Cons.hyps(1))
    have  $2: \vdash big\_ior\ (map\ (\lambda(x, y).\ x\ iand\ next\ y)$ 
       $(filter\ (\lambda(x, y).\ \exists\sigma.\ \sigma \models x)$ 
       $(zip\ xs\ ys)))$ 
       $ieqv\ big\_ior\ (map2\ (\lambda x\ y.\ x\ iand\ next\ y)\ xs\ ys)$ 
      using Cons.hyps(2) False by fastforce
    show ?thesis
      using  $2\ big\_ior\ Cons\text{-}alt$  by auto
    qed
  qed
qed

```

```

lemma big-ior-rev-ieqv:
   $\vdash big\_ior\ L\ ieqv\ big\_ior\ (rev\ L)$ 
by (auto simp add: big-ior-defs)
  (metis in-set-conv-nth length-rev set-rev,
   metis diff-less length-greater-0-conv less-nat-zero-code list.size(3) rev-nth zero-less-Suc)

```

```

lemma big-ior-map2-append:
assumes  $length\ L1 = length\ N1$ 
       $length\ L2 = length\ N2$ 
shows  $\vdash (big\_ior\ (map2\ (\lambda\ x\ y.\ x\ iand\ (next\ y))\ (L1@L2)\ (N1@N2)))\ ieqv$ 
       $(big\_ior\ (map2\ (\lambda\ x\ y.\ x\ iand\ (next\ y))\ L1\ N1))\ ior$ 
       $(big\_ior\ (map2\ (\lambda\ x\ y.\ x\ iand\ (next\ y))\ L2\ N2))$ 
using assms
proof (induct  $L1\ N1$  arbitrary: L2 N2 rule: list-induct2)
case Nil

```

```

then show ?case by (simp add: big-ior-defs)
next
case (Cons x xs y ys)
then show ?case
  by (metis Cons.hyps(1) big-ior-append length-Cons map-append zip-append)

```

qed

```

lemma map2-iand-next-defs :
  assumes length F = length G
  shows  $(\exists i < \text{length } G. \sigma \models \text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) F G ! i) \longleftrightarrow$ 
     $(\exists i < \text{length } G. (\sigma \models F ! i) \wedge (\sigma \models (\text{next } (G ! i))))$ 
using assms
proof (induct G arbitrary: F)
case Nil
then show ?case by simp
next
case (Cons a G)
then show ?case
  proof (cases F=[])
  case True
  then show ?thesis using Cons.prem by auto
  next
  case False
  then show ?thesis using Cons by auto
  qed
qed

```

```

lemma map2-iand-inot-next-defs :
  assumes length F = length G
    0 < nlength  $\sigma$ 
  shows  $(\exists i < \text{length } G. \sigma \models \text{map2 } (\lambda x y. x \text{ iand } (\text{next } (\text{inot } y))) F G ! i) \longleftrightarrow$ 
     $(\exists i < \text{length } G. (\sigma \models F ! i) \wedge \neg(\sigma \models (\text{next } (G ! i))))$ 
using assms
proof (induct G arbitrary: F)
case Nil
then show ?case by simp
next
case (Cons a G)
then show ?case
  proof (cases F=[])
  case True
  then show ?thesis using Cons by simp
  next
  case False
  then show ?thesis
    proof -
      obtain b F1 where 1:  $F = b \# F1$ 
      using False by (meson neq-Nil-conv)
      have 2:  $\text{length } F1 = \text{length } G$ 

```

```

    using 1 Cons.prem by auto
  have 3: ( $\exists i < \text{length } G. \sigma \models \text{map2 } (\lambda x y. x \text{ iand next inot } y) F1 G ! i =$ 
    ( $\exists i < \text{length } G. (\sigma \models F1 ! i) \wedge \neg (\sigma \models \text{next } G ! i)$ ))
    using 2 Cons.hyps assms(2) by presburger
  have 4: ( $\exists i < \text{length } (a \# G). \sigma \models \text{map2 } (\lambda x y. x \text{ iand next inot } y) F (a \# G) ! i \longleftrightarrow$ 
    ( $\sigma \models \text{map2 } (\lambda x y. x \text{ iand next inot } y) (b \# F1) (a \# G) ! 0$ )  $\vee$ 
    ( $\exists i. 0 < i \wedge i < \text{length } (a \# G) \wedge (\sigma \models \text{map2 } (\lambda x y. x \text{ iand next inot } y) (b \# F1) (a \# G) ! i)$ ))
    using 1 by (metis gr-zeroI length-greater-0-conv list.discI)
  have 5: ( $\sigma \models \text{map2 } (\lambda x y. x \text{ iand next inot } y) (b \# F1) (a \# G) ! 0 \longleftrightarrow$ 
    ( $\sigma \models b \text{ iand next inot } a$ ))
    by auto
  have 6: ( $\exists i. 0 < i \wedge i < \text{length } (a \# G) \wedge (\sigma \models \text{map2 } (\lambda x y. x \text{ iand next inot } y) (b \# F1) (a \# G) !$ 
     $i) \longleftrightarrow$ 
    ( $\exists i < \text{length } (G). (\sigma \models \text{map2 } (\lambda x y. x \text{ iand next inot } y) (F1) (G) ! i)$ ))
    by auto
    (metis Suc-less-eq Suc-pred)
  have 7: ( $\exists i < \text{length } (a \# G). (\sigma \models (b \# F1) ! i) \wedge \neg (\sigma \models \text{next } (a \# G) ! i) \longleftrightarrow$ 
    ( $(\sigma \models (b \# F1) ! 0) \wedge \neg (\sigma \models \text{next } (a \# G) ! 0)$ )  $\vee$ 
    ( $\exists i. 0 < i \wedge i < \text{length } (a \# G) \wedge (\sigma \models (b \# F1) ! i) \wedge \neg (\sigma \models \text{next } (a \# G) ! i)$ ))
    by (metis gr-zeroI length-greater-0-conv list.discI)
  have 8: ( $(\sigma \models (b \# F1) ! 0) \wedge \neg (\sigma \models \text{next } (a \# G) ! 0) \longleftrightarrow$ 
    ( $\sigma \models b \text{ iand next inot } a$ ))
    by simp
    (metis One-nat-def Suc-ile-eq assms(2) enat-defs(1) enat-defs(2))
  have 9: ( $\exists i. 0 < i \wedge i < \text{length } (a \# G) \wedge (\sigma \models (b \# F1) ! i) \wedge \neg (\sigma \models \text{next } (a \# G) ! i) \longleftrightarrow$ 
    ( $\exists i < \text{length } (G). (\sigma \models (F1) ! i) \wedge \neg (\sigma \models \text{next } (G) ! i)$ ))
    by auto
    (metis Suc-diff-Suc Suc-less-eq diff-zero, metis Suc-less-eq Suc-pred)
  show ?thesis
    using 1 3 4 5 6 7 8 9 by presburger
  qed
  qed
  qed

```

lemma *iand-big-ior-dist*:

assumes *length G-now = length G-next*

shows $\vdash F\text{-e iand big-ior } (\text{map2 } (\lambda x y. x \text{ iand next } y) G\text{-now } G\text{-next}) \text{ ieqv}$

$\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand next } y)$

$(\text{map } (\lambda x. x \text{ iand } F\text{-e}) G\text{-now}) G\text{-next})$

using *assms* **by** (*auto simp add: big-ior-defs*)

lemma *big-ior-map2-defs*:

assumes *length L1 = length L2*

shows $(\sigma \models (\text{big-ior } (\text{map2 } (\lambda x y. f x y) L1 L2))) \longleftrightarrow$

$(\exists i < \text{length } L1. (\sigma \models f (L1!i) (L2!i)))$

using *assms*

proof (*induct L1 L2 rule: list-induct2*)

case *Nil*

then show *?case* **unfolding** *big-ior-def* **by** *auto*

```

next
case (Cons x xs y ys)
then show ?case using big-ior-def less-Suc-eq-0-disj by simp auto
qed

```

```

lemma big-ior-map2-schop-dis:
  assumes length L1 = length L2
  shows  $\vdash (big-ior (map2 (\lambda x y. f x y) L1 L2)) \text{schop } g \text{ieqv}$ 
     $(big-ior (map2 (\lambda x y. (f x y) \text{schop } g) L1 L2))$ 
using assms
by (auto simp add: big-ior-map2-defs)

```

4.2.4 Quantifier lemmas

```

lemma ExistSubst:
  assumes  $n \notin \text{bvars } f$ 
     $\bigwedge z. z \in \text{fvars } w \implies z \notin \text{bvars } f$ 
    state-qpitl w
  shows  $\vdash \text{suform } f \text{ } n \text{ } w \text{ } iimp (Ex \text{ } n. f)$ 
  using assms subst-suform valid-def by force

```

```

lemma ForallSubst:
  assumes  $n \notin \text{bvars } f$ 
     $\bigwedge z. z \in \text{fvars } w \implies z \notin \text{bvars } f$ 
    state-qpitl w
  shows  $\vdash (Fa \text{ } n. f) \text{ } iimp \text{suform } f \text{ } n \text{ } w$ 
  using assms by (simp add: subst-suform)

```

```

lemma ExistsImp:
  assumes  $v1 \notin \text{fvars } f1$ 
  shows  $\vdash (Ex \text{ } v1. (f1 \text{ } iimp \text{ } f2)) \text{ieqv } (f1 \text{ } iimp (Ex \text{ } v1. f2))$ 
proof -
  have 1:  $\bigwedge \sigma. (\sigma \models (Ex \text{ } v1. (f1 \text{ } iimp \text{ } f2))) = (\sigma \models (f1 \text{ } iimp (Ex \text{ } v1. f2)))$ 
    by simp
    (metis assms nlength-nmap not-fvar-upd)
  show ?thesis
  using 1 itl-ieq by blast
qed

```

```

lemma ExistsSChopRight:
  assumes  $v1 \notin \text{fvars } f2$ 
  shows  $\vdash (Ex \text{ } v1. f1 \text{schop } f2) \text{ } iimp (Ex \text{ } v1. f1) \text{schop } f2$ 
proof -
  have 1:  $\bigwedge \sigma. (\sigma \models (Ex \text{ } v1. f1 \text{schop } f2) \text{ } iimp (Ex \text{ } v1. f1) \text{schop } f2)$ 
  proof -
    fix  $\sigma$ 
    show  $(\sigma \models (Ex \text{ } v1. f1 \text{schop } f2) \text{ } iimp (Ex \text{ } v1. f1) \text{schop } f2)$ 

```

proof –
have 2: $(\sigma \models (Ex\ v1.\ f1\ schop\ f2)) =$
 $(\exists\ l\ k.\ nlength\ \sigma = nlength\ l \wedge k \leq nlength\ \sigma \wedge$
 $(ntaken\ k\ (upd\ \sigma\ v1\ l) \models f1) \wedge$
 $(ndropn\ k\ (upd\ \sigma\ v1\ l) \models f2))$
by *simp* (*metis nlength-upd*)
have 3: $(\sigma \models (Ex\ v1.\ f1)\ schop\ f2) =$
 $(\exists\ k\ l.\ k \leq nlength\ \sigma \wedge nlength\ l = nlength\ (ntaken\ k\ \sigma)$
 $\wedge (upd\ (ntaken\ k\ \sigma)\ v1\ l \models f1) \wedge (ndropn\ k\ \sigma \models f2))$
by (*metis less-eq-nat.simps(1) semantics-qpitl.simps(5) semantics-qpitl.simps(8)*)
have 4: $(\exists\ l\ k.\ nlength\ \sigma = nlength\ l \wedge k \leq nlength\ \sigma \wedge$
 $(ntaken\ k\ (upd\ \sigma\ v1\ l) \models f1) \wedge$
 $(ndropn\ k\ (upd\ \sigma\ v1\ l) \models f2)) \implies$
 $(\exists\ k\ l.\ k \leq nlength\ \sigma \wedge nlength\ l = nlength\ (ntaken\ k\ \sigma)$
 $\wedge (upd\ (ntaken\ k\ \sigma)\ v1\ l \models f1) \wedge (ndropn\ k\ \sigma \models f2))$
by (*metis assms ndropn-nlength ndropn-upd not-fvar-upd ntaken-nlength ntaken-upd*)
show *?thesis*
using 2 3 4 *semantics-qpitl.simps(3)* **by** *presburger*
qed
qed
show *?thesis*
using 1 *valid-def* **by** *blast*
qed

lemma *ExistsSChopLeft*:

assumes $v1 \notin fvars\ f1$
shows $\vdash (Ex\ v1.\ f1\ schop\ f2) \text{ iimp } f1\ schop\ (Ex\ v1.\ f2)$
proof –
have 1: $\bigwedge\ \sigma.\ \sigma \models (Ex\ v1.\ f1\ schop\ f2) \text{ iimp } f1\ schop\ (Ex\ v1.\ f2)$
proof –
fix σ
show $\sigma \models (Ex\ v1.\ f1\ schop\ f2) \text{ iimp } f1\ schop\ (Ex\ v1.\ f2)$
proof –
have 2: $(\sigma \models (Ex\ v1.\ f1\ schop\ f2)) =$
 $(\exists\ l\ k.\ nlength\ \sigma = nlength\ l \wedge k \leq nlength\ \sigma \wedge$
 $(ntaken\ k\ (upd\ \sigma\ v1\ l) \models f1) \wedge$
 $(ndropn\ k\ (upd\ \sigma\ v1\ l) \models f2))$
by *simp* (*metis nlength-upd*)
have 3: $(\sigma \models f1\ schop\ (Ex\ v1.\ f2)) =$
 $(\exists\ k\ l.\ k \leq nlength\ \sigma \wedge nlength\ l = nlength\ (ndropn\ k\ \sigma) \wedge$
 $(ntaken\ k\ \sigma \models f1) \wedge (upd\ (ndropn\ k\ \sigma)\ v1\ l \models f2))$
by *force*
have 4: $(\exists\ l\ k.\ nlength\ \sigma = nlength\ l \wedge k \leq nlength\ \sigma \wedge$
 $(ntaken\ k\ (upd\ \sigma\ v1\ l) \models f1) \wedge$
 $(ndropn\ k\ (upd\ \sigma\ v1\ l) \models f2)) \implies$
 $(\exists\ k\ l.\ k \leq nlength\ \sigma \wedge nlength\ l = nlength\ (ndropn\ k\ \sigma) \wedge$
 $(ntaken\ k\ \sigma \models f1) \wedge (upd\ (ndropn\ k\ \sigma)\ v1\ l \models f2))$
by (*metis assms ndropn-nlength ndropn-upd not-fvar-upd ntaken-nlength ntaken-upd*)
show *?thesis*
using 2 3 4 *semantics-qpitl.simps(3)* **by** *presburger*

```

qed
qed
show ?thesis
using 1 valid-def by blast
qed

```

```

lemma ForallGen:
assumes  $\vdash f$ 
shows  $\vdash \text{Fa } v. f$ 
using assms by simp

```

```

lemma ExistsI:
assumes  $(\sigma \models f)$ 
shows  $(\sigma \models \text{Ex } v. f)$ 
using assms exist-sigma-exist-value semantics-qpitl.simps(8) similar-refl by blast

```

```

lemma ExistsNoDep:
assumes  $v \notin \text{fvars } f$ 
shows  $\vdash (\text{Ex } v. f) \text{ ieqv } f$ 
using ExistsI assms not-fvar-upd by auto

```

```

lemma ForAllNoDep:
assumes  $v \notin \text{fvars } f$ 
shows  $\vdash (\text{Fa } v. f) \text{ ieqv } f$ 
using assms
using nlength-nmap not-fvar-upd
by (metis (mono-tags, lifting) forall-defs itl-ieq)

```

```

lemma ExistsEqvRule:
assumes  $\vdash f \text{ ieqv } g$ 
shows  $\vdash (\text{Ex } v. f) \text{ ieqv } (\text{Ex } v. g)$ 
using assms by auto

```

```

lemma ForallImpExists:
 $\vdash (\text{Fa } v. f) \text{ iimp } (\text{Ex } v. f)$ 
by simp
(metis exist-sigma-exist-value ittrue-defs similar-refl)

```

```

lemma ExistsImpRuleDist:
assumes  $\vdash f \text{ iimp } g$ 
shows  $\vdash (\text{Fa } v. f) \text{ iimp } (\text{Ex } v. g)$ 
using assms by simp
(metis nlength-nmap)

```

```

lemma ExistsImpNoDepDist:
assumes  $v \notin \text{fvars } f$ 
 $\vdash f \text{ iimp } g$ 
shows  $\vdash f \text{ iimp } (\text{Ex } v. g)$ 
using assms

```


using *ExistsI* **by** *force*

lemma *ExistsOrDist-1*:

$\vdash (Ex\ v.\ h)\ iimp\ (Ex\ v.\ f\ ior\ h)$

by *auto*

lemma *ExistsOrDist-2*:

$\vdash (Ex\ v.\ h)\ iimp\ (Ex\ v.\ h\ ior\ f)$

by *auto*

lemma *ExistsOrDist-3*:

$\vdash (Ex\ v.\ f)\ ior\ (Ex\ v.\ h)\ iimp\ (Ex\ v.\ f\ ior\ h)$

by *auto*

lemma *ExistsOrDist*:

$\vdash (Ex\ v.\ f)\ ior\ (Ex\ v.\ h)\ ieqv\ (Ex\ v.\ f\ ior\ h)$

by *auto*

lemma *ExistsOrImport-1*:

assumes $v \notin fvars\ g$

shows $\vdash g\ iimp\ (Ex\ v.\ g\ ior\ f)$

using *assms*

using *ExistsI* *ior-defs* *semantics-qpitl.simps*(*3*) *valid-def* **by** *presburger*

lemma *ExistsOrImport-2*:

assumes $v \notin fvars\ g$

shows $\vdash (Ex\ v.\ f)\ iimp\ (Ex\ v.\ g\ ior\ f)$

using *assms*

using *ExistsOrDist-1* **by** *blast*

lemma *ExistsOrImport-3*:

assumes $v \notin fvars\ g$

shows $\vdash g\ ior\ (Ex\ v.\ f)\ iimp\ (Ex\ v.\ g\ ior\ f)$

using *assms*

using *ExistsI* **by** *fastforce*

lemma *ExistsOrImport-4*:

assumes $v \notin fvars\ g$

shows $\vdash (Ex\ v.\ g\ ior\ f)\ iimp\ g\ ior\ (Ex\ v.\ f)$

using *assms*

using *ExistsNoDep* **by** *auto*

lemma *ExistsOrImport*:

assumes $v \notin fvars\ g$

shows $\vdash g\ ior\ (Ex\ v.\ g\ ior\ f)\ ieqv\ (Ex\ v.\ g\ ior\ f)$

using *assms*

using *ExistsI* **by** *fastforce*

lemma *ExistsAndImport-1*:

assumes $v \notin fvars\ g$

shows $\vdash g \text{ iand } (Ex\ v.\ f) \text{ iimp } (Ex\ v.\ g \text{ iand } f)$
using *assms*
using *not-fvar-upd* **by** *auto*

lemma *ExistsAndImport-2*:
assumes $v \notin fvars\ g$
shows $\vdash (Ex\ v.\ g \text{ iand } f) \text{ iimp } g \text{ iand } (Ex\ v.\ f)$
using *assms*
using *not-fvar-upd* **by** *auto*

lemma *ExistsAndImport*:
assumes $v \notin fvars\ g$
shows $\vdash g \text{ iand } (Ex\ v.\ f) \text{ ieqv } (Ex\ v.\ g \text{ iand } f)$
using *assms*
using *not-fvar-upd* **by** *auto*

lemma *ExistsAndDist*:
assumes $\vdash f \text{ iand } g$
shows $\vdash (Ex\ v.\ f) \text{ iand } (Ex\ v.\ g)$
using *assms*
using *ExistsI* **by** *fastforce*

lemma *ExistsAndNoDepDist*:
assumes $\vdash f \text{ iand } g$
 $v \notin fvars\ f$
shows $\vdash f \text{ iand } (Ex\ v.\ g)$
using *assms*
by (*meson* *ExistsI* *iand-defs* *valid-def*)

lemma *ForallE*:
assumes $\vdash Fa\ v.\ f$
 $\vdash f \text{ iimp } g$
shows $\vdash g$
using *assms*
by (*metis* *ExistsI* *forall-d-def* *inot-defs* *semantics-qpitl.simps*(3) *valid-def*)

lemma *ForallEqvRule*:
assumes $\vdash f \text{ ieqv } g$
shows $\vdash (Fa\ v.\ f) \text{ ieqv } (Fa\ v.\ g)$
using *assms* **by** *auto*

lemma *ForallAndDist*:
 $\vdash (Fa\ v.\ f \text{ iand } g) \text{ ieqv } (Fa\ v.\ f) \text{ iand } (Fa\ v.\ g)$
by *auto*

lemma *ForallAndImport*:
assumes $v \notin fvars\ g$
shows $\vdash g \text{ iand } (Fa\ v.\ f) \text{ ieqv } (Fa\ v.\ g \text{ iand } f)$
using *assms*
using *ForaAllNoDep* *ForallAndDist* **by** *force*

lemma *ForallOrImport:*

assumes $v \notin \text{fvars } g$

shows $\vdash g \text{ ior } (Fa \ v. \ f) \text{ ieqv } (Fa \ v. \ g \text{ ior } f)$

using *assms*

using *not-fvar-upd* **by** *auto*

lemma *RightChopImpChop:*

assumes $\vdash f \text{ iimp } g$

shows $\vdash h \text{ schop } f \text{ iimp } h \text{ schop } g$

using *assms* **by** *auto*

lemma *ExistsImpChopRule:*

assumes $\vdash f \text{ iimp } g$

$v \notin \text{fvars } h$

shows $\vdash (Ex \ v. \ (h \text{ schop } f) \text{ iimp } (h \text{ schop } g))$

using *assms*

by (*meson* *ExistsI* *RightChopImpChop* *valid-def*)

lemma *SChop-nfuse:*

shows $(\sigma \models f \text{ schop } g) =$

$(\exists \ \sigma 1 \ \sigma 2. \ \sigma = \text{nfuse } \sigma 1 \ \sigma 2 \wedge \text{nlast } \sigma 1 = \text{nfirst } \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge$
 $(\sigma 1 \models f) \wedge (\sigma 2 \models g))$

by *auto*

(*metis* *ndropn-nfirst* *nfinite-ntaken* *nfuse-ntaken-ndropn* *ntaken-nlast*,

metis *linorder-le-cases* *ndropn-nfuse* *nfinite-conv-nlength-enat* *ntaken-all* *ntaken-nfuse* *the-enat.simps*)

lemma *ExistsSChopRightNoDep:*

assumes $v \notin \text{fvars } g$

shows $\vdash (Ex \ v. \ f \text{ schop } g) \text{ ieqv } (Ex \ v. \ f) \text{ schop } g$

proof –

have 1: $\vdash (Ex \ v. \ f \text{ schop } g) \text{ iimp } (Ex \ v. \ f) \text{ schop } g$

using *ExistsSChopRight* *assms* **by** *blast*

have 2: $\vdash (Ex \ v. \ f) \text{ schop } g \text{ iimp } (Ex \ v. \ f \text{ schop } g)$

proof –

have 3: $\bigwedge \sigma. (\sigma \models (Ex \ v. \ f) \text{ schop } g) \implies (\sigma \models (Ex \ v. \ f \text{ schop } g))$

proof –

fix σ

assume $a0: (\sigma \models (Ex \ v. \ f) \text{ schop } g)$

show $(\sigma \models (Ex \ v. \ f \text{ schop } g))$

proof –

have 4: $(\exists \ k \ l. \ k \leq \text{nlength } \sigma \wedge \text{nlength } l = \text{nlength}(\text{ntaken } k \ \sigma)$
 $\wedge ((\text{upd } (\text{ntaken } k \ \sigma) \ v \ l) \models f) \wedge (\text{ndropn } k \ \sigma \models g))$

using $a0$ **by** *fastforce*

obtain $k \ l$ **where** 5: $k \leq \text{nlength } \sigma \wedge \text{nlength } l = \text{nlength}(\text{ntaken } k \ \sigma)$
 $\wedge ((\text{upd } (\text{ntaken } k \ \sigma) \ v \ l) \models f) \wedge (\text{ndropn } k \ \sigma \models g)$

using 4 **by** *blast*

have 6: $\bigwedge l1. \text{nlength } l1 = \text{nlength } (\text{ndropn } k \ \sigma) \implies$

```

      ( (upd (ndropn k  $\sigma$ ) v l1)  $\models$  g)
using 5 assms not-fvar-upd by blast
obtain l1 where 7: nlength l1 = nlength (ndropn k  $\sigma$ )  $\wedge$ 
      nfirst l1 = nlast l  $\wedge$ 
      ( (upd (ndropn k  $\sigma$ ) v l1)  $\models$  g)
using 6 nlength-nmap
by (metis ndropn-0 ndropn-nfirst nellist.exhaust nlength-NCons nlength-NNil nnth-0 nnth-NNil)
have 70: enat k  $\leq$  nlength (nfuse l l1)
by (simp add: 5 nfuse-nlength)
have 8: ntaken k (nfuse l l1) = l
by (metis 5 enat-le-plus-same(2) gen-nlength-def min.absorb1 nfuse-def1 nlength-code
      ntaken-all ntaken-nappend1 ntaken-nlength)
have 9: ndropn k (nfuse l l1) = l1
      using ndropn-nfuse[of l l1]
      by (metis 7 70 8 min.absorb1 nfinite-ntaken ntaken-nlength the-enat.simps)
have 10: (ntaken k (upd  $\sigma$  v (nfuse l l1))  $\models$  f)
by (metis 5 7 8 nfuse-nlength nfuse-ntaken-ndropn ntaken-upd)
have 11: (ndropn k (upd  $\sigma$  v (nfuse l l1))  $\models$  g)
by (metis 5 7 9 ndropn-upd nfuse-nlength nfuse-ntaken-ndropn)
have 12: nlength  $\sigma$  = nlength (nfuse l l1)
by (metis 5 7 nfuse-nlength nfuse-ntaken-ndropn-nlength)
show ?thesis using 70 10 11 12
using nlength-upd by fastforce
qed
qed
show ?thesis
using 3 semantics-qpitl.simps(3) valid-def by presburger
qed
show ?thesis
using 1 2 ieqv-d-def by auto
qed

```

lemma *ExistsSChopLeftNoDep:*

```

assumes v  $\notin$  fvars g
shows  $\vdash$  (Ex v. g schop f) ieqv g schop (Ex v. f)
proof –
have 1:  $\vdash$  (Ex v. g schop f) iimp g schop (Ex v. f)
using ExistsSChopLeft assms by blast
have 2:  $\vdash$  g schop (Ex v. f) iimp (Ex v. g schop f)
proof –
have 3:  $\bigwedge \sigma. (\sigma \models g \text{ schop } (Ex v. f)) \implies (\sigma \models (Ex v. g \text{ schop } f))$ 
proof –
fix  $\sigma$ 
assume a0: ( $\sigma \models g \text{ schop } (Ex v. f)$ )
show  $(\sigma \models (Ex v. g \text{ schop } f))$ 
proof –
have 4:  $(\exists k l. k \leq nlength \sigma \wedge nlength l = nlength (ndropn k \sigma) \wedge$ 
       $(ntaken k \sigma \models g) \wedge (upd (ndropn k \sigma) v l \models f))$ 
using a0 by fastforce
obtain k l where 5:  $k \leq nlength \sigma \wedge nlength l = nlength (ndropn k \sigma) \wedge$ 

```

$(\text{ntaken } k \ \sigma \models g) \wedge (\text{upd } (\text{ndropn } k \ \sigma) \ v \ l \models f)$
using 4 **by** *blast*
have 6: $\bigwedge l1. \text{nlength } l1 = \text{nlength } (\text{ntaken } k \ \sigma) \implies$
 $(\text{upd } (\text{ntaken } k \ \sigma) \ v \ l1 \models g)$
using 5 *assms nfinite-conv-nlength-enat not-fvar-upd* **by** *auto*

have 60: $\text{nlength } (\text{ntaken } k \ \sigma) = k$
by (*simp add: 5*)
have 61: $\text{nlength } \sigma = 0 \implies \exists l1. \text{nlength } l1 = \text{nlength } (\text{ntaken } k \ \sigma) \wedge (\text{upd } (\text{ntaken } k \ \sigma) \ v \ l1 \models g) \wedge$
nfinite l1
 $\wedge \text{nlast } l1 = \text{nfirst } l$
by (*metis 5 6 le-zero-eq ndropn-0 nfirst-eq-nnth-zero nlength-eq-enat-nfiniteD nnth-nlast*
ntaken-all the-enat.simps the-enat-0)
have 62: $\text{nlength } \sigma > 0 \wedge k > 0 \implies \exists l1. \text{nlength } (\text{nappend } l1 \ (\text{NNil } (\text{nnth } l \ 0))) = \text{nlength } (\text{ntaken } k \ \sigma) \wedge$
 $(\text{upd } (\text{ntaken } k \ \sigma) \ v \ (\text{nappend } l1 \ (\text{NNil } (\text{nnth } l \ 0)))) \models g) \wedge \text{nfinite } l1$
using 6 60 5 **by** *auto*
(metis One-nat-def Suc-pred add.right-neutral add-Suc-right idiff-enat-enat min.orderE
ndropn-nlength nfinite-ntaken nle-le nlength-NNil nlength-nappend nlength-nmap ntaken-all
ntaken-nlength one-enat-def plus-enat-simps(1))
have 63: $\text{nlength } \sigma > 0 \wedge k = 0 \implies \exists l1. \text{nlength } l1 = \text{nlength } (\text{ntaken } k \ \sigma) \wedge (\text{upd } (\text{ntaken } k \ \sigma) \ v$
 $l1 \models g) \wedge \text{nfinite } l1$
 $\wedge \text{nlast } l1 = \text{nfirst } l$
by (*metis 6 nfinite-NNil nlast-NNil nlength-NNil ntaken-0*)
have 64: $\text{nlength } \sigma = 0 \implies ?thesis$
by (*metis 5 6 ndropn-0 nle-le nlength-upd ntaken-all semantics-qpitl.simps(5)*
semantics-qpitl.simps(8) the-enat.simps zero-enat-def zero-le)
have 65: $\text{nlength } \sigma > 0 \wedge k = 0 \implies ?thesis$
by (*metis 5 6 ndropn-0 nlength-upd ntaken-nlength ntaken-upd semantics-qpitl.simps(5)*
semantics-qpitl.simps(8) zero-le)
have 66: $\text{nlength } \sigma > 0 \wedge k > 0 \implies ?thesis$
proof –
assume a1: $\text{nlength } \sigma > 0 \wedge k > 0$
show *?thesis*
proof –
have 70: $\exists l1. \text{nlength } (\text{nappend } l1 \ (\text{NNil } (\text{nnth } l \ 0))) = \text{nlength } (\text{ntaken } k \ \sigma) \wedge$
 $(\text{upd } (\text{ntaken } k \ \sigma) \ v \ (\text{nappend } l1 \ (\text{NNil } (\text{nnth } l \ 0)))) \models g) \wedge \text{nfinite } l1$
using a1 62 **by** *auto*
obtain l1 **where** 71: $\text{nlength } (\text{nappend } l1 \ (\text{NNil } (\text{nnth } l \ 0))) = \text{nlength } (\text{ntaken } k \ \sigma) \wedge$
 $(\text{upd } (\text{ntaken } k \ \sigma) \ v \ (\text{nappend } l1 \ (\text{NNil } (\text{nnth } l \ 0)))) \models g) \wedge \text{nfinite } l1$
using 70 **by** *blast*
have 72: $\text{nlast } (\text{nappend } l1 \ (\text{NNil } (\text{nnth } l \ 0))) = \text{nfirst } l$
by (*simp add: 71 nfirst-eq-nnth-zero nlast-nappend*)
have 73: $\text{enat } k \leq \text{nlength } (\text{nfuse } (\text{nappend } l1 \ (\text{NNil } (\text{nnth } l \ 0)))) \ l)$
by (*metis 60 71 enat-le-plus-same(1) nfuse-nlength*)
have 74: $\text{ntaken } k \ (\text{nfuse } (\text{nappend } l1 \ (\text{NNil } (\text{nnth } l \ 0)))) \ l = (\text{nappend } l1 \ (\text{NNil } (\text{nnth } l \ 0)))$
using 60 71 72 *ntaken-nfuse[of (nappend l1 (NNil (nnth l 0))) l]*
nlength-eq-enat-nfiniteD **by** (*metis the-enat.simps*)
have 75: $\text{ndropn } k \ (\text{nfuse } (\text{nappend } l1 \ (\text{NNil } (\text{nnth } l \ 0)))) \ l = l$
by (*metis 60 71 74 ndropn-nfuse nfinite-ntaken nfirst-eq-nnth-zero nlast-NNil*)

```

      nlast-nappend the-enat.simps)
have 76: (ntaken k (upd  $\sigma$  v (nfuse (nappend l1 (NNil (nnth l 0))) l))  $\models$  g)
  by (metis 5 71 74 nfuse-nlength nfuse-ntaken-ndropn ntaken-upd)
have 77: (ndropn k (upd  $\sigma$  v (nfuse (nappend l1 (NNil (nnth l 0))) l))  $\models$  f)
  by (metis 5 71 75 ndropn-upd nfuse-nlength nfuse-ntaken-ndropn)
have 78: nlength  $\sigma$  = nlength (nfuse (nappend l1 (NNil (nnth l 0))) l)
  by (metis 5 71 nfuse-nlength nfuse-ntaken-ndropn-nlength)
show ?thesis by simp
  (metis 5 76 77 78 nlength-upd)
qed
qed
show ?thesis
  by (meson 64 65 66 i0-less less-nat-zero-code linorder-neqE-nat)
qed
qed
show ?thesis
using 3 semantics-qpitl.simps(3) valid-def by presburger
qed
show ?thesis
using 1 2 ieqv-d-def by auto
qed

```

```

lemma ExistsExistsSChop:
assumes v  $\notin$  fvars g
      y  $\notin$  fvars f
shows  $\vdash$  (Ex v. (Ex y. f schop g)) ieqv (Ex v. f) schop (Ex y. g)
proof -
  have 1:  $\vdash$  ( (Ex y. f schop g)) ieqv ( f schop (Ex y. g) )
    using assms ExistsSChopLeftNoDep[of y f g] by blast
  have 2:  $\vdash$  (Ex v. (Ex y. f schop g)) ieqv (Ex v. f schop (Ex y. g) )
    using 1 ExistsEqvRule by blast
  have 3:  $\vdash$  (Ex v. f schop (Ex y. g) ) ieqv (Ex v. f) schop (Ex y. g)
    by (metis Diff-iff ExistsSChopRightNoDep assms(1) fvars.simps(8))
  show ?thesis
  by (meson 2 3 itl-ieq)
qed

```

```

lemma exists-elim-state-qpitl:
assumes state-qpitl w
      n  $\notin$  bvars w
shows  $\vdash$  (Ex n. w) ieqv (suform w n ittrue) ior (suform w n ifalse)
proof -
  have 1: state-qpitl ittrue
    by (simp add: inot-d-def ittrue-d-def)
  have 2: ( $\bigwedge z. z \in$  fvars ittrue  $\implies z \notin$  bvars w)
    by (simp add: inot-d-def ittrue-d-def)
  have 3:  $\bigwedge \sigma. (\sigma \models$  (suform w n ittrue)) = ((upd  $\sigma$  n (nmap ( $\lambda c. (NNil c) \models$  ittrue)  $\sigma$ ))  $\models$  w)

```

```

    using subst-suform[of n w ittrue ] 1 2 assms(2) by blast
have 4: state-qpitl ifalse
  by simp
have 5:  $(\bigwedge z. z \in fvars\ ifalse \implies z \notin bvars\ w)$ 
  by simp
have 6:  $\bigwedge \sigma. (\sigma \models (suform\ w\ n\ ifalse)) = ((upd\ \sigma\ n\ (nmap\ (\lambda c. (NNil\ c) \models ifalse)\ \sigma)) \models w)$ 
  using subst-suform[of n w ifalse ] 4 5 assms(2) by blast
have 7:  $\bigwedge \sigma. (\sigma \models (Ex\ n. w)) = (\exists l. nlength\ \sigma = nlength\ l \wedge (upd\ \sigma\ n\ l \models w))$ 
  by simp
have 8:  $\bigwedge \sigma. (\exists l. nlength\ \sigma = nlength\ l \wedge (upd\ \sigma\ n\ l \models w)) =$ 
 $(\exists l. nlength\ \sigma = nlength\ l \wedge ((NNil\ (nfirst\ (upd\ \sigma\ n\ l))) \models w))$ 
  using assms(1) state-qpitl-defs by blast
have 9:  $\bigwedge \sigma. ((upd\ \sigma\ n\ (nmap\ (\lambda c. (NNil\ c) \models ittrue)\ \sigma)) \models w) =$ 
 $(NNil\ (nfirst\ (upd\ \sigma\ n\ (nmap\ (\lambda c. (NNil\ c) \models ittrue)\ \sigma))) \models w)$ 
  using assms(1) state-qpitl-defs by blast
have 10:  $\bigwedge \sigma. (nfirst\ (upd\ \sigma\ n\ (nmap\ (\lambda c. (NNil\ c) \models ittrue)\ \sigma))) =$ 
 $insert\ n\ (nnth\ \sigma\ 0)$ 
  by (metis ittrue-defs nfirst-eq-nnth-zero nfirst-upd nlength-nmap nnth-nmap zero-enat-def zero-le)
have 11:  $\bigwedge \sigma. ((upd\ \sigma\ n\ (nmap\ (\lambda c. (NNil\ c) \models ifalse)\ \sigma)) \models w) =$ 
 $(NNil\ (nfirst\ (upd\ \sigma\ n\ (nmap\ (\lambda c. (NNil\ c) \models ifalse)\ \sigma))) \models w)$ 
  using assms(1) state-qpitl-defs by blast
have 12:  $\bigwedge \sigma. (nfirst\ (upd\ \sigma\ n\ (nmap\ (\lambda c. (NNil\ c) \models ifalse)\ \sigma))) =$ 
 $(nnth\ \sigma\ 0) - \{n\}$ 
  by (metis i0-less nfirst-eq-nnth-zero nfirst-upd nle-le nlength-nmap nnth-nmap order-less-imp-le
    semantics-qpitl.simps(1) zero-enat-def)
have 13:  $\bigwedge \sigma\ l. nlength\ \sigma = nlength\ l \implies (nfirst\ (upd\ \sigma\ n\ l)) = (if\ (nnth\ l\ 0)\ then\ insert\ n\ (nnth\ \sigma\ 0)$ 
 $else\ (nnth\ \sigma\ 0) - \{n\})$ 
  by (metis ndropn-0 ndropn-nfirst nfirst-upd)
have 14:  $\bigwedge \sigma. (\exists l. nlength\ \sigma = nlength\ l \wedge ((NNil\ (nfirst\ (upd\ \sigma\ n\ l))) \models w)) \longleftrightarrow$ 
 $(NNil\ (insert\ n\ (nnth\ \sigma\ 0)) \models w) \vee (NNil\ ((nnth\ \sigma\ 0) - \{n\}) \models w)$ 
  by (metis 10 12 ndropn-0 ndropn-nfirst nfirst-upd nlength-nmap)
have 15:  $\bigwedge \sigma. (\sigma \models (Ex\ n. w)) \longleftrightarrow (\sigma \models (suform\ w\ n\ ittrue)) \vee (\sigma \models (suform\ w\ n\ ifalse))$ 
  by (metis 10 12 14 3 6 7 assms(1) state-qpitl-defs)
show ?thesis
using 15 ior-defs itl-ieq by presburger
qed

```

end

theory QPITL-Deep-GNF imports
 QPITL-Deep

begin

This theory proves that all QPITL formulae can be written in Guarded Normal Form (GNF) [4, 2].

4.3 Guarded Normal Form

4.3.1 Definitions

definition *mutex*

where *mutex* $F =$

$$(\forall k1\ k2. 0 \leq k1 \wedge k1 < k2 \wedge k2 < \text{length } F \longrightarrow (\vdash \text{inot } ((F! k1) \text{ iand } (F! k2))))$$

definition *mutex-single*

where *mutex-single* $f\ F =$

$$(\forall k2. 0 \leq k2 \wedge k2 < \text{length } F \longrightarrow (\vdash \text{inot } (f \text{ iand } (F! k2))))$$

definition *add-to-now-single*

where

$$\begin{aligned} \text{add-to-now-single } f\ L &= ((\text{map } (\lambda x. (f \text{ iand } x))\ L) @ \\ &\quad (\text{map } (\lambda x. (\text{inot } f \text{ iand } x))\ L)) \end{aligned}$$

definition *add-to-next-single*

where

$$\text{add-to-next-single } f\ L = ((\text{map } (\lambda x. (f \text{ ior } x))\ L) @ L)$$

definition *add-to-single*

where

$$\begin{aligned} \text{add-to-single } f1\ f2\ L1\ L2 &= \\ \text{filter } (\lambda (x,y). (\exists \sigma. \sigma \models x)) &(\text{zip } (\text{add-to-now-single } f1\ L1) (\text{add-to-next-single } f2\ L2)) \end{aligned}$$

definition *add-to-next-single-a*

where

$$\text{add-to-next-single-a } f\ L = ((\text{map } (\lambda x. ((\text{if } f = \text{ifalse then } x \text{ else } f \text{ ior } x)))\ L) @ L)$$

definition *add-to-now*

where *add-to-now* $L1 = \text{foldr } (\text{add-to-now-single})\ L1\ [\text{itrue}]$

definition *add-to-next*

where *add-to-next* $L1 = \text{foldr } (\text{add-to-next-single})\ L1\ [\text{ifalse}]$

definition *add-to-next-a*

where *add-to-next-a* $L1 = \text{foldr } (\text{add-to-next-single-a})\ L1\ [\text{ifalse}]$

definition *add-to*

where *add-to* $L1\ L2 = \text{filter } (\lambda (x,y). (\exists \sigma. \sigma \models x))\ (\text{zip } (\text{add-to-now } L1) (\text{add-to-next } L2))$

fun *GNF-empty* :: *qpitl* \Rightarrow *qpitl*

where

GNF-empty (*ifalse*) = *ifalse*
 | *GNF-empty* (*\$n*) = (*\$n*)
 | *GNF-empty* (*f iimp g*) = ((*inot* (*GNF-empty f*)) *ior* (*GNF-empty g*))
 | *GNF-empty* (*next f*) = (*ifalse*)
 | *GNF-empty* (*f schop g*) = ((*GNF-empty f*) *iand* (*GNF-empty g*))
 | *GNF-empty* (*s chopstar f*) = *itrue*
 | *GNF-empty* (*omega f*) = *ifalse*
 | *GNF-empty* (*Ex n. f*) = (*Ex n. (GNF-empty f)*)

fun *GNF-state* :: *qpitl* \Rightarrow *qpitl list*

where

GNF-state (*ifalse*) = [*itrue*]
 | *GNF-state* (*\$n*) = [*\$n*]
 | *GNF-state* (*f iimp g*) = ((*add-to-now* (*GNF-state f*))) @ (*GNF-state g*)
 | *GNF-state* (*next f*) = [*itrue*]
 | *GNF-state* (*f schop g*) = ((*GNF-state f*) @ (*map* ($\lambda x. x$ *iand* (*GNF-empty f*)) (*GNF-state g*)))
 | *GNF-state* (*s chopstar f*) = (*GNF-state f*)
 | *GNF-state* (*omega f*) = (*GNF-state f*)
 | *GNF-state* (*Ex n. f*) = (*map* ($\lambda x. (Ex n. x)$) (*GNF-state f*))

fun *GNF-next* :: *qpitl* \Rightarrow *qpitl list*

where

GNF-next (*ifalse*) = [*ifalse*]
 | *GNF-next* (*\$n*) = [*itrue*]
 | *GNF-next* (*f iimp g*) = ((*map* ($\lambda x. (inot x)$) (*add-to-next* (*GNF-next f*)))) @ (*GNF-next g*)
 | *GNF-next* (*next f*) = [*f*]
 | *GNF-next* (*f schop g*) = ((*map* ($\lambda x. x$ *s chop g*) (*GNF-next f*))) @ (*GNF-next g*)
 | *GNF-next* (*s chopstar f*) = (*map* ($\lambda x. x$ *s chop* (*s chopstar f*)) (*GNF-next f*))
 | *GNF-next* (*omega f*) = (*map* ($\lambda x. x$ *s chop* (*omega f*)) (*GNF-next f*))
 | *GNF-next* (*Ex n. f*) = (*map* ($\lambda x. (Ex n. x)$) (*GNF-next f*))

fun *GNF-next-a* :: *qpitl* \Rightarrow *qpitl list*

where

GNF-next-a (*ifalse*) = [*ifalse*]
 | *GNF-next-a* (*\$n*) = [*itrue*]
 | *GNF-next-a* (*f iimp g*) = ((*map* ($\lambda x. (inot x)$) (*add-to-next-a* (*GNF-next-a f*)))) @ (*GNF-next-a g*)
 | *GNF-next-a* (*next f*) = [*f*]
 | *GNF-next-a* (*f schop g*) = ((*map* ($\lambda x. x$ *s chop g*) (*GNF-next-a f*))) @ (*GNF-next-a g*)
 | *GNF-next-a* (*s chopstar f*) = (*map* ($\lambda x. x$ *s chop* (*s chopstar f*)) (*GNF-next-a f*))
 | *GNF-next-a* (*omega f*) = (*map* ($\lambda x. x$ *s chop* (*omega f*)) (*GNF-next-a f*))
 | *GNF-next-a* (*Ex n. f*) = (*map* ($\lambda x. (Ex n. f)$) (*GNF-next-a f*))

definition *full-system*

where *full-system* *F* = ((\vdash *big-ior F*) \wedge *mutex F*)

definition *GNF-cons*

where *GNF-cons* *ae a A* =
 $((ae \text{ iand } empty) \text{ ior } (big\text{-ior } (map2 (\lambda x y. x \text{ iand } (next y)) a A)))$

definition *GNF*

where *GNF f* =
 $(GNF\text{-cons } (GNF\text{-empty } f) (GNF\text{-state } f) (GNF\text{-next } f))$

definition *GNF-prop*

where *GNF-prop X* =
 $(\exists ae a A.$
 $(state\text{-qpitl } ae) \wedge (state\text{-qpitl-list } a) \wedge$
 $full\text{-system } a \wedge length\ a = length\ A \wedge$
 $(\vdash X \text{ ieqv } GNF\text{-cons } ae\ a\ A)$
 $)$

definition *GNF-det*

where *GNF-det f* =
 $(GNF\text{-cons } (GNF\text{-empty } f)$
 $(foldr (add\text{-to-now-single }) (GNF\text{-state } f) [itrue])$
 $(foldr (add\text{-to-next-single}) (GNF\text{-next } f) [ifalse])$
 $)$

4.3.2 Mutex lemmas

lemma *mutex-single-empty:*

shows *mutex-single f []*

unfolding *mutex-single-def* **by** *auto*

lemma *mutex-single-Cons:*

shows *mutex-single f (f1 # F) \longleftrightarrow*
 $(\vdash \text{ inot } (f \text{ iand } f1)) \wedge \text{mutex-single } f\ F$

unfolding *mutex-single-def* **using** *less-Suc-eq-0-disj* **by** *auto*

lemma *mutex-single-append:*

shows *mutex-single f (F1 @ F2) \longleftrightarrow mutex-single f F1 \wedge mutex-single f F2*

proof (*induct F1 arbitrary: F2*)

case *Nil*

then show *?case* **unfolding** *mutex-single-def* **by** *simp*

next

case (*Cons a F1*)

then show *?case*

proof –

have 1: *mutex-single f ((a # F1) @ F2) \longleftrightarrow*
 $(\vdash \text{ inot } (f \text{ iand } a)) \wedge \text{mutex-single } f\ (F1 @ F2)$

by (*simp add: mutex-single-Cons*)

have 2: *mutex-single f (F1 @ F2) \longleftrightarrow mutex-single f F1 \wedge mutex-single f F2*

```

using Cons by blast
have 3:  $(\vdash \text{inot } (f \text{ iand } a)) \wedge (\text{mutex-single } f \ F1 \wedge \text{mutex-single } f \ F2) \longleftrightarrow$ 
 $(\text{mutex-single } f \ (a \ \# \ F1) \wedge \text{mutex-single } f \ F2)$ 
using mutex-single-Cons by blast
show ?thesis
using 1 2 3 by presburger
qed
qed

```

```

lemma mutex-empty:
shows mutex []
unfolding mutex-def by auto

```

```

lemma mutex-Cons:
shows  $\text{mutex } (f \ \# \ F1) \longleftrightarrow \text{mutex-single } f \ F1 \wedge \text{mutex } F1$ 
unfolding mutex-def mutex-single-def
using less-Suc-eq-0-disj
apply auto
apply (metis diff-Suc-1' nth-Cons')
by (metis diff-Suc-1' nth-Cons-Suc)

```

```

lemma mutex-append:
shows  $\text{mutex } (F1 \ @ \ F2) \longleftrightarrow$ 
 $\text{mutex } F1 \wedge \text{foldr } (\lambda x \ y. (\text{mutex-single } x \ F2) \wedge y) \ F1 \ \text{True} \wedge \text{mutex } F2$ 
proof (induct F1 arbitrary: F2)
case Nil
then show ?case
by (simp add: mutex-empty)
next
case (Cons a F1)
then show ?case
proof –
have 1:  $\text{mutex } ((a \ \# \ F1) \ @ \ F2) \longleftrightarrow$ 
 $\text{mutex-single } a \ F1 \wedge \text{mutex-single } a \ F2 \wedge \text{mutex } (F1 \ @ \ F2)$ 
by (simp add: mutex-Cons mutex-single-append)
have 2:  $\text{foldr } (\lambda x. (\wedge) (\text{mutex-single } x \ F2)) (a \ \# \ F1) \ \text{True} \longleftrightarrow$ 
 $\text{mutex-single } a \ F2 \wedge \text{foldr } (\lambda x. (\wedge) (\text{mutex-single } x \ F2)) (F1) \ \text{True}$ 
by simp
have 3:  $\text{mutex } (F1 \ @ \ F2) \longleftrightarrow (\text{mutex } F1 \wedge \text{foldr } (\lambda x. (\wedge) (\text{mutex-single } x \ F2)) \ F1 \ \text{True} \wedge \text{mutex } F2)$ 
using Cons by blast
have 4:  $\text{mutex-single } a \ F1 \wedge \text{mutex-single } a \ F2 \wedge \text{mutex } (F1 \ @ \ F2) \longleftrightarrow$ 
 $\text{mutex-single } a \ F1 \wedge \text{mutex-single } a \ F2 \wedge$ 
 $(\text{mutex } F1 \wedge \text{foldr } (\lambda x. (\wedge) (\text{mutex-single } x \ F2)) \ F1 \ \text{True} \wedge \text{mutex } F2)$ 
using 1 3 by auto
have 5:  $\text{mutex-single } a \ F1 \wedge \text{mutex-single } a \ F2 \wedge$ 
 $(\text{mutex } F1 \wedge \text{foldr } (\lambda x. (\wedge) (\text{mutex-single } x \ F2)) \ F1 \ \text{True} \wedge \text{mutex } F2) \longleftrightarrow$ 

```

```

      (mutex (a # F1) ∧ foldr (λx. (∧) (mutex-single x F2)) (a # F1) True ∧ mutex F2)
    using 2 mutex-Cons by blast
  show ?thesis
    using 1 4 5 by presburger
qed
qed

```

```

lemma unfold-mutex-single:
  foldr (λx y. (mutex-single x F2) ∧ y) F1 True ⟷
    (∀ i. 0 ≤ i ∧ i < length F1 ⟶ (mutex-single (F1 ! i) F2))
proof (induct F1 arbitrary: F2)
case Nil
then show ?case by simp
next
case (Cons a F1)
then show ?case
  using less-Suc-eq-0-disj by auto
qed

```

4.3.3 Add to lemmas

```

lemma length-add-to-now-single:
shows length (add-to-now-single f L) = (if L=[] then 0 else 2*(length L))
proof (induct L)
case Nil
then show ?case unfolding add-to-now-single-def by simp
next
case (Cons a L)
then show ?case unfolding add-to-now-single-def by simp
qed

```

```

lemma length-add-to-next-single:
shows length (add-to-next-single f L) = (if L=[] then 0 else 2*(length L))
proof (induct L)
case Nil
then show ?case unfolding add-to-next-single-def by simp
next
case (Cons a L)
then show ?case unfolding add-to-next-single-def by simp
qed

```

```

lemma length-add-to-next-single-a:
shows length (add-to-next-single-a f L) = (if L=[] then 0 else 2*(length L))
proof (induct L)
case Nil
then show ?case unfolding add-to-next-single-a-def by simp
next
case (Cons a L)

```

then show *?case* **unfolding** *add-to-next-single-a-def* **by** *simp*
qed

lemma *length-add-to-now-next-gen*:
assumes *length xs = length ys*
 length L1 = length L2
shows *length (foldr add-to-now-single xs L1) =*
 length (foldr add-to-next-single ys L2)
using *assms*
proof (*induct xs ys rule: list-induct2*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons x xs y ys*)
then show *?case*
 using *Cons* **unfolding** *add-to-now-single-def add-to-next-single-def*
 by *simp*
qed

lemma *length-add-to-now-next-a-gen*:
assumes *length xs = length ys*
 length L1 = length L2
shows *length (foldr add-to-now-single xs L1) =*
 length (foldr add-to-next-single-a ys L2)
using *assms*
proof (*induct xs ys rule: list-induct2*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons x xs y ys*)
then show *?case*
 using *Cons* **unfolding** *add-to-now-single-def add-to-next-single-a-def*
 by *simp*
qed

lemma *length-add-to-now-next*:
assumes *length xs = length ys*
shows *length (foldr add-to-now-single xs [itrue,ifalse]) =*
 length (foldr add-to-next-single ys [ifalse,ifalse])
using *assms*
by (*simp add: length-add-to-now-next-gen*)

lemma *length-add-to-now-next-a*:
assumes *length xs = length ys*
shows *length (foldr add-to-now-single xs [itrue,ifalse]) =*
 length (foldr add-to-next-single-a ys [ifalse,ifalse])
using *assms*
by (*simp add: length-add-to-now-next-a-gen*)

lemma *add-to-now-single-to-big-ior-inv*:
assumes $\vdash \text{big-ior } L$
shows $\vdash \text{big-ior } (\text{add-to-now-single } f \ L)$
unfolding *add-to-now-single-def* **using** *assms*
 $\text{big-ior-append}[\text{of } (\text{map } (\lambda x. (f \ i \ \text{and } x)) \ L) \ (\text{map } (\lambda x. (i \ \text{not } f \ i \ \text{and } x)) \ L)]$
by (*auto simp add: big-ior-defs*)
(metis iand-defs nth-map)

lemma *big-ior-foldr-add-to-now-single-inv-a*:
shows $\vdash \text{big-ior } (\text{foldr } \text{add-to-now-single } xs \ [\text{itrue}])$
proof (*induct xs*)
case *Nil*
then show *?case*
using *big-ior-defs* **by** *fastforce*
next
case (*Cons a xs*)
then show *?case*
using *Cons add-to-now-single-to-big-ior-inv* **by** *auto*
qed

lemma *big-ior-foldr-add-to-now-single-inv*:
shows $\vdash \text{big-ior } (\text{foldr } \text{add-to-now-single } xs \ [\text{itrue}, \text{ifalse}])$
proof (*induct xs*)
case *Nil*
then show *?case*
using *big-ior-defs* **by** *fastforce*
next
case (*Cons a xs*)
then show *?case*
using *Cons add-to-now-single-to-big-ior-inv* **by** *auto*
qed

lemma *add-to-now-single-to-mutex-inv*:
assumes *mutex L*
shows $\text{mutex } (\text{add-to-now-single } f \ L)$
proof –
have 1: $\text{mutex } (\text{map } (\lambda x. f \ i \ \text{and } x) \ L)$
using *assms* **unfolding** *mutex-def* **by** *auto*
have 2: $\text{mutex } (\text{map } (\lambda x. i \ \text{not } f \ i \ \text{and } x) \ L)$
using *assms* **unfolding** *mutex-def* **by** *auto*
have 3: $\text{foldr } (\lambda x. (\wedge) (\text{mutex-single } x \ (\text{map } (\lambda x. (i \ \text{not } f) \ i \ \text{and } x) \ L))) \ (\text{map } (\lambda x. f \ i \ \text{and } x) \ L) \ \text{True}$
 $\iff (\forall i. 0 \leq i \wedge i < \text{length } (\text{map } (\lambda x. f \ i \ \text{and } x) \ L) \longrightarrow \text{mutex-single } (\text{map } (\lambda x. f \ i \ \text{and } x) \ L \ ! \ i) \ (\text{map } (\lambda x. (i \ \text{not } f) \ i \ \text{and } x) \ L))$
using *unfold-mutex-single* [*of* ($\text{map } (\lambda x. (i \ \text{not } f) \ i \ \text{and } x) \ L$) ($\text{map } (\lambda x. f \ i \ \text{and } x) \ L$)]
by *blast*
have 4: $(\forall i. 0 \leq i \wedge i < \text{length } (\text{map } (\lambda x. f \ i \ \text{and } x) \ L) \longrightarrow \text{mutex-single } (\text{map } (\lambda x. f \ i \ \text{and } x) \ L \ ! \ i) \ (\text{map } (\lambda x. (i \ \text{not } f) \ i \ \text{and } x) \ L))$
using *assms* **unfolding** *mutex-single-def* **by** *auto*

```

show ?thesis unfolding add-to-now-single-def using assms
  mutex-append[of (map (λx. f iand x ) L) (map (λx. (inot f) iand x ) L) ]
using 1 2 3 4 by blast
qed

```

```

lemma mutex-foldr-add-to-now-single-inv-a:
shows mutex (foldr add-to-now-single xs [itrue])
proof (induct xs)
case Nil
then show ?case unfolding mutex-def by simp
next
case (Cons a xs)
then show ?case by (simp add: Cons.hyps add-to-now-single-to-mutex-inv)
qed

```

```

lemma mutex-foldr-add-to-now-single-inv:
shows mutex (foldr add-to-now-single xs [itrue, ifalse])
proof (induct xs)
case Nil
then show ?case unfolding mutex-def by simp
next
case (Cons a xs)
then show ?case by (simp add: Cons.hyps add-to-now-single-to-mutex-inv)
qed

```

```

lemma add-to-now-single-to-state-qpitl-list:
assumes state-qpitl f
  state-qpitl-list L
shows state-qpitl-list ( add-to-now-single f L )
using assms unfolding add-to-now-single-def
by (simp add: inot-d-def state-qpitl-list-append state-qpitl-list-map)

```

```

lemma state-qpitl-foldr-add-to-now-single-inv-a:
assumes state-qpitl-list xs
shows state-qpitl-list (foldr add-to-now-single xs [itrue])
using assms
proof (induct xs)
case Nil
then show ?case using Nil
by (simp add: inot-d-def itrue-d-def state-qpitl-list-def)
next
case (Cons a xs)
then show ?case
  by simp
  (metis add-to-now-single-to-state-qpitl-list append-Cons append-Nil length-Cons nth-Cons-0 state-qpitl-list-append
state-qpitl-list-defs zero-less-Suc)
qed

```

```

lemma state-qpitl-foldr-add-to-now-single-inv:

```

```

assumes state-qpitl-list xs
shows state-qpitl-list (foldr add-to-now-single xs [itrue, ifalse])
using assms
proof (induct xs)
case Nil
then show ?case using Nil
by (simp add: inot-d-def itrue-d-def state-qpitl-list-def)
next
case (Cons a xs)
then show ?case
  by simp
  (metis add-to-now-single-to-state-qpitl-list append-Cons append-Nil length-Cons nth-Cons-0 state-qpitl-list-append
state-qpitl-list-defs zero-less-Suc)
qed

```

```

lemma det-big-ior-ieqv-nondet-big-ior-a:
assumes length L1 = length L2
shows  $\vdash$  big-ior ( map2 ( $\lambda x y. x \text{ iand } (\text{next } y)$ )
  (foldr (add-to-now-single) L1 [itrue])
  (foldr (add-to-next-single) L2 [ifalse]))
  ieqv
  big-ior (map2 ( $\lambda x y. x \text{ iand } (\text{next } y)$ ) L1 L2)
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by (simp add: big-ior-def)
next
case (Cons x xs y ys)
then show ?case
  proof (cases xs = []  $\wedge$  ys = [])
  case True
  then show ?thesis using Cons
  unfolding add-to-now-single-def add-to-next-single-def big-ior-def big-iand-def
  by simp
  next
  case False
  then show ?thesis
  proof –
  have 1: (foldr add-to-now-single (x # xs) [itrue]) =
    ((map ( $\lambda x1. x \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue])) @
    (map ( $\lambda x1. (\text{inot } x) \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue])))
    by (simp add: add-to-now-single-def)
  have 2: (foldr add-to-next-single (y # ys) [ifalse]) =
    ((map ( $\lambda x1. y \text{ ior } x1$ ) (foldr add-to-next-single ys [ifalse]))@
    (foldr add-to-next-single ys [ifalse]) )
    by (simp add: add-to-next-single-def)
  have 3: length xs = length ys
    by (simp add: Cons.hyps(1))
  have 4:  $\vdash$  big-ior (map2 ( $\lambda x y. x \text{ iand } \text{next } y$ )
    (foldr add-to-now-single xs [itrue])

```



```

      (foldr add-to-next-single ys [ifalse])) ieqv
    big-ior (map2 (λx y. x iand next y) xs ys)
  using 3 Cons.hyps(2) False by fastforce
have 5: big-ior (map2 (λx y. x iand next y)
  (foldr add-to-now-single (x # xs) [itrue]))
  (foldr add-to-next-single (y # ys) [ifalse])) =
  big-ior (map2 (λx y. x iand next y)
    ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))))
    ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse])) @
      (foldr add-to-next-single ys [ifalse]) )
    )
  using 1 2 by presburger
have 6: length (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) =
  length (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse]))
  by (simp add: 3 length-add-to-now-next-gen)
have 7: length (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])) =
  length (foldr add-to-next-single ys [ifalse])
  using 6 by auto
have 75: length (foldr add-to-now-single xs [itrue]) =
  length (foldr add-to-next-single ys [ifalse])
  using 7 by force
have 8: ⊢ big-ior (map2 (λx y. x iand next y)
  ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))))
  ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse])) @
    (foldr add-to-next-single ys [ifalse]) )
  ) ieqv
  big-ior ( (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
    (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse])) )
    @
    (map2 (λx y. x iand next y)
      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
      (foldr add-to-next-single ys [ifalse]) )
    )
  )
  using 6 by force
have 9: ⊢ big-ior ( (map2 (λx y. x iand next y)
  (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
  (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse])) )
  @
  (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
    (foldr add-to-next-single ys [ifalse]) )
  ) ieqv
  (big-ior (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
    (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse])) ) ) ior
  (big-ior (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))

```

```

                                (foldr add-to-next-single ys [ifalse]) ))
  using big-ior-append by blast
have 10: ⊢ (big-ior (map2 (λx y. x iand next y)
                        (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
                        (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse])))) ieqv
  (big-ior (map2 (λx1 y1. (x iand x1) iand next (y ior y1))
                ( (foldr add-to-now-single xs [itrue]))
                ( (foldr add-to-next-single ys [ifalse])))))
  using map2-map-a[of (foldr add-to-now-single xs [itrue])
                    (foldr add-to-next-single ys [ifalse]) x y] using 7 by auto
have 11: ⊢ (big-ior (map2 (λx y. x iand next y)
                        (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
                        (foldr add-to-next-single ys [ifalse])))) ieqv
  (big-ior (map2 (λx1 y1. ((inot x) iand x1) iand next y1)
                ( (foldr add-to-now-single xs [itrue]))
                (foldr add-to-next-single ys [ifalse]))))
  using map2-map-b[of (foldr add-to-now-single xs [itrue])
                    (foldr add-to-next-single ys [ifalse]) x] 75
  by auto
have 12: ⊢ big-ior (foldr add-to-now-single xs [itrue])
  using big-ior-foldr-add-to-now-single-inv-a by metis
have 13: 0 < length (foldr add-to-now-single xs [itrue])
  using 12 big-ior-empty by auto
have 14: ⊢ (big-ior (map2 (λx1 y1. (x iand x1) iand next (y ior y1))
                        ( (foldr add-to-now-single xs [itrue]))
                        ( (foldr add-to-next-single ys [ifalse])))) ior
  (big-ior (map2 (λx y. x iand next y)
                (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
                (foldr add-to-next-single ys [ifalse]))))
  ieqv
  (x iand next y) ior
  (big-ior (map2 (λx1 y1. x1 iand next y1)
                ( (foldr add-to-now-single xs [itrue]))
                ( (foldr add-to-next-single ys [ifalse]))))
  using big-ior-split-off[of ( (foldr add-to-now-single xs [itrue]))
                           ( (foldr add-to-next-single ys [ifalse])) x y] 12 13
  by (simp add: 75 map2-map-b)
have 15: ⊢ big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys)) ieqv
  (x iand next y) ior
  big-ior (map2 (λx y. x iand next y) xs ys)
  using big-ior-Cons-alt by auto
have 16: ⊢ (x iand next y) ior
  (big-ior (map2 (λx1 y1. x1 iand next y1)
                ( (foldr add-to-now-single xs [itrue]))
                ( (foldr add-to-next-single ys [ifalse])))) ieqv
  big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))
  using 4 15 by auto
show ?thesis
  using 10 14 16 5 6 9 by force
qed

```

qed
qed

lemma *det-big-ior-ieqv-nondet-big-ior-a-a:*

assumes *length L1 = length L2*

shows $\vdash \text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y))$
 $(\text{foldr } (\text{add-to-now-single}) L1 [\text{itrue}])$
 $(\text{foldr } (\text{add-to-next-single-a}) L2 [\text{ifalse}]))$
 ieqv
 $\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) L1 L2)$

using *assms*

proof (*induct L1 L2 rule: list-induct2*)

case *Nil*

then show *?case* **by** (*simp add: big-ior-def*)

next

case (*Cons x xs y ys*)

then show *?case*

proof (*cases xs = [] \wedge ys = []*)

case *True*

then show *?thesis* **using** *Cons*

unfolding *add-to-now-single-def add-to-next-single-a-def big-ior-def big-iand-def*
by *simp*

next

case *False*

then show *?thesis*

proof –

have 1: (*foldr add-to-now-single (x # xs) [itrue]*) =
 $(\text{map } (\lambda x1. x \text{ iand } x1) (\text{foldr add-to-now-single } xs [\text{itrue}])) @$
 $(\text{map } (\lambda x1. (\text{inot } x) \text{ iand } x1) (\text{foldr add-to-now-single } xs [\text{itrue}]))$
by (*simp add: add-to-now-single-def*)

have 2: (*foldr add-to-next-single-a (y # ys) [ifalse]*) =
 $(\text{map } (\lambda x1. (\text{if } y = \text{ifalse} \text{ then } x1 \text{ else } y \text{ ior } x1)) (\text{foldr add-to-next-single-a } ys [\text{ifalse}])) @$
 $(\text{foldr add-to-next-single-a } ys [\text{ifalse}])$
by (*simp add: add-to-next-single-a-def*)

have 3: *length xs = length ys*

by (*simp add: Cons.hyps(1)*)

have 4: $\vdash \text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y)$
 $(\text{foldr add-to-now-single } xs [\text{itrue}])$
 $(\text{foldr add-to-next-single-a } ys [\text{ifalse}])) \text{ ieqv}$
 $\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) xs ys)$

using 3 *Cons.hyps(2) False* **by** *fastforce*

have 5: *big-ior* (*map2* ($\lambda x y. x \text{ iand } \text{next } y$)
 $(\text{foldr add-to-now-single } (x \# xs) [\text{itrue}])$
 $(\text{foldr add-to-next-single-a } (y \# ys) [\text{ifalse}])) =$
 $\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y)$
 $(\text{map } (\lambda x1. x \text{ iand } x1) (\text{foldr add-to-now-single } xs [\text{itrue}])) @$
 $(\text{map } (\lambda x1. (\text{inot } x) \text{ iand } x1) (\text{foldr add-to-now-single } xs [\text{itrue}]))$
 $(\text{map } (\lambda x1. (\text{if } y = \text{ifalse} \text{ then } x1 \text{ else } y \text{ ior } x1)) (\text{foldr add-to-next-single-a } ys [\text{ifalse}])) @$
 $(\text{foldr add-to-next-single-a } ys [\text{ifalse}])$
 $)$

```

    using 1 2 by presburger
  have 6: length (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) =
    length (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
  by (simp add: 3 length-add-to-now-next-a-gen)
  have 7: length (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])) =
    length (foldr add-to-next-single-a ys [ifalse])
  using 6 by auto
  have 75: length (foldr add-to-now-single xs [itrue]) =
    length (foldr add-to-next-single-a ys [ifalse])
  using 7 by force
  have 8: ⊢ big-ior (map2 (λx y. x iand next y)
    ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue])) @
      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue])) )
    ( (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse])) @
      (foldr add-to-next-single-a ys [ifalse]) )
  ) ieqv
  big-ior ( (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
  )
    @
    (map2 (λx y. x iand next y)
      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
      (foldr add-to-next-single-a ys [ifalse]) )
    )
  using 6 by force
  have 9: ⊢ big-ior ( (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
  )
    @
    (map2 (λx y. x iand next y)
      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
      (foldr add-to-next-single-a ys [ifalse]) )
    ) ieqv
  (big-ior (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
  )
    @
    (big-ior (map2 (λx y. x iand next y)
      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
      (foldr add-to-next-single-a ys [ifalse]) )
    )
  ) ior
  (big-ior (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
    (foldr add-to-next-single-a ys [ifalse]) )
  )
  using big-ior-append by blast
  have 10: ⊢ (big-ior (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue]))
    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys [ifalse]))
  )
    ieqv
    (big-ior (map2 (λx1 y1. (x iand x1) iand next ((if y = ifalse then y1 else y ior y1)))
      ( (foldr add-to-now-single xs [itrue]))
      ( (foldr add-to-next-single-a ys [ifalse])) )
  )

```

```

    using map2-map-a-a[of (foldr add-to-now-single xs [itrue])
      (foldr add-to-next-single-a ys [ifalse]) x y ] using 7 by auto
  have 11: ⊢ (big-ior (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
    (foldr add-to-next-single-a ys [ifalse]) )) ieqv
    (big-ior (map2 (λx1 y1. ((inot x) iand x1) iand next y1)
      ( (foldr add-to-now-single xs [itrue]))
      (foldr add-to-next-single-a ys [ifalse]) ))
    using map2-map-b[of (foldr add-to-now-single xs [itrue])
      (foldr add-to-next-single-a ys [ifalse]) x] 75
  by auto
  have 12: ⊢ big-ior (foldr add-to-now-single xs [itrue])
    using big-ior-foldr-add-to-now-single-inv-a by metis
  have 13: 0 < length (foldr add-to-now-single xs [itrue])
    using 12 big-ior-empty by auto
  have 14: ⊢ (big-ior (map2 (λx1 y1. (x iand x1) iand next ((if y = ifalse then y1 else y ior y1)))
    ( (foldr add-to-now-single xs [itrue]))
    ( (foldr add-to-next-single-a ys [ifalse]) ))) ior
    (big-ior (map2 (λx y. x iand next y)
      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue]))
      (foldr add-to-next-single-a ys [ifalse]) ))
    ieqv
    (x iand next y) ior
    (big-ior (map2 (λx1 y1. x1 iand next y1)
      ( (foldr add-to-now-single xs [itrue]))
      ( (foldr add-to-next-single-a ys [ifalse]) )))
    using big-ior-split-off-a[of ( (foldr add-to-now-single xs [itrue]))
      ( (foldr add-to-next-single-a ys [ifalse]) ) x y] 12 13
  by (simp add: 75 map2-map-b)
  have 15: ⊢ big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys)) ieqv
    (x iand next y) ior
    big-ior (map2 (λx y. x iand next y) xs ys)
    using big-ior-Cons-alt by auto
  have 16: ⊢ (x iand next y) ior
    (big-ior (map2 (λx1 y1. x1 iand next y1)
      ( (foldr add-to-now-single xs [itrue]))
      ( (foldr add-to-next-single-a ys [ifalse]) ))) ieqv
    big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))
    using 4 15 by auto
  show ?thesis
  using 10 14 16 5 6 9 by force
qed
qed
qed

```

lemma *det-big-ior-ieqv-nondet-big-ior*:

assumes *length L1 = length L2*

shows ⊢ *big-ior (map2 (λ x y. x iand (next y))*
(foldr (add-to-now-single) L1 [itrue,ifalse])
(foldr (add-to-next-single) L2 [ifalse,ifalse])

```

    ieqv
    big-ior (map2 (λx y. x iand (next y)) L1 L2)
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by (simp add: big-ior-def)
next
case (Cons x xs y ys)
then show ?case
  proof (cases xs = [] ∧ ys = [] )
  case True
  then show ?thesis using Cons
    unfolding add-to-now-single-def add-to-next-single-def big-ior-def big-iand-def
    by simp
  next
  case False
  then show ?thesis
    proof -
    have 1: (foldr add-to-now-single (x # xs) [itrue,ifalse]) =
      ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) @
        (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])))
      by (simp add: add-to-now-single-def)
    have 2: (foldr add-to-next-single (y # ys) [ifalse,ifalse]) =
      ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse]))@
        (foldr add-to-next-single ys [ifalse,ifalse]) )
      by (simp add: add-to-next-single-def)
    have 3: length xs = length ys
      by (simp add: Cons.hyps(1))
    have 4: ⊢ big-ior (map2 (λx y. x iand next y)
      (foldr add-to-now-single xs [itrue,ifalse])
      (foldr add-to-next-single ys [ifalse,ifalse])) ieqv
      big-ior (map2 (λx y. x iand next y) xs ys)
      using 3 Cons.hyps(2) False by fastforce
    have 5: big-ior (map2 (λx y. x iand next y)
      (foldr add-to-now-single (x # xs) [itrue,ifalse])
      (foldr add-to-next-single (y # ys) [ifalse,ifalse])) =
      big-ior (map2 (λx y. x iand next y)
      ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) @
        (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])))
      ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse]))@
        (foldr add-to-next-single ys [ifalse,ifalse]) )
      )
      using 1 2 by presburger
    have 6: length (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) =
      length (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse]))
      by (simp add: 3 length-add-to-now-next)
    have 7: length (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])) =
      length (foldr add-to-next-single ys [ifalse,ifalse])
      using 6 by auto
    have 75: length (foldr add-to-now-single xs [itrue, ifalse]) =

```

```

length (foldr add-to-next-single ys [ifalse, ifalse])
using 7 by force
have 8: ⊢ big-ior (map2 (λx y. x iand next y)
  ( (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) @
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])) )
  ( (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse])) @
    (foldr add-to-next-single ys [ifalse,ifalse]) )
  ) ieqv
big-ior ( (map2 (λx y. x iand next y)
  (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
  (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse])) )
  @
  (map2 (λx y. x iand next y)
  (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
  (foldr add-to-next-single ys [ifalse,ifalse]) )
  )
  )
using 6 by force
have 9: ⊢ big-ior ( (map2 (λx y. x iand next y)
  (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
  (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse])) )
  @
  (map2 (λx y. x iand next y)
  (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
  (foldr add-to-next-single ys [ifalse,ifalse]) )
  ) ieqv
  )
  (big-ior (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
    (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse])) ) ) ior
  (big-ior (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
    (foldr add-to-next-single ys [ifalse,ifalse]) ) )
  )
  )
using big-ior-append by blast
have 10: ⊢ (big-ior (map2 (λx y. x iand next y)
  (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
  (map (λx1. y ior x1) (foldr add-to-next-single ys [ifalse,ifalse])) ) ) ieqv
  (big-ior (map2 (λx1 y1. (x iand x1) iand next (y ior y1))
    ( (foldr add-to-now-single xs [itrue,ifalse]))
    ( (foldr add-to-next-single ys [ifalse,ifalse])) ) )
  )
  )
  using map2-map-a[of (foldr add-to-now-single xs [itrue,ifalse])
    (foldr add-to-next-single ys [ifalse,ifalse]) x y ] using 7 by auto
have 11: ⊢ (big-ior (map2 (λx y. x iand next y)
  (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
  (foldr add-to-next-single ys [ifalse,ifalse]) ) ) ieqv
  (big-ior (map2 (λx1 y1. ((inot x) iand x1) iand next y1)
    ( (foldr add-to-now-single xs [itrue,ifalse]))
    (foldr add-to-next-single ys [ifalse,ifalse]) ) )
  )
  )
  using map2-map-b[of (foldr add-to-now-single xs [itrue,ifalse])
    (foldr add-to-next-single ys [ifalse,ifalse]) x ] 75
  by auto
have 12: ⊢ big-ior (foldr add-to-now-single xs [itrue, ifalse])

```

```

using big-ior-foldr-add-to-now-single-inv by metis
have 13:  $0 < \text{length} (\text{foldr } \text{add-to-now-single } xs \ [ittrue, ifalse])$ 
using 12 big-ior-empty by auto
have 14:  $\vdash (\text{big-ior } (\text{map2 } (\lambda x1 \ y1. (x \text{ iand } x1) \text{ iand } \text{next } (y \text{ ior } y1))$ 
 $\quad ( (\text{foldr } \text{add-to-now-single } xs \ [ittrue, ifalse]))$ 
 $\quad ( (\text{foldr } \text{add-to-next-single } ys \ [ifalse, ifalse])))) \text{ ior}$ 
 $\quad (\text{big-ior } (\text{map2 } (\lambda x \ y. x \text{ iand } \text{next } y)$ 
 $\quad (\text{map } (\lambda x1. (\text{inot } x) \text{ iand } x1) (\text{foldr } \text{add-to-now-single } xs \ [ittrue, ifalse]))$ 
 $\quad (\text{foldr } \text{add-to-next-single } ys \ [ifalse, ifalse])) \ ))$ 
 $\text{ieqv}$ 
 $(x \text{ iand } \text{next } y) \text{ ior}$ 
 $(\text{big-ior } (\text{map2 } (\lambda x1 \ y1. x1 \text{ iand } \text{next } y1)$ 
 $\quad ( (\text{foldr } \text{add-to-now-single } xs \ [ittrue, ifalse]))$ 
 $\quad ( (\text{foldr } \text{add-to-next-single } ys \ [ifalse, ifalse]))))$ 
using big-ior-split-off [of  $( (\text{foldr } \text{add-to-now-single } xs \ [ittrue, ifalse]))$ 
 $( (\text{foldr } \text{add-to-next-single } ys \ [ifalse, ifalse])) \ x \ y]$  12 13
by (simp add: 75 map2-map-b)
have 15:  $\vdash \text{big-ior } (\text{map2 } (\lambda x \ y. x \text{ iand } \text{next } y) (x \# xs) (y \# ys)) \text{ ieqv}$ 
 $(x \text{ iand } \text{next } y) \text{ ior}$ 
 $\text{big-ior } (\text{map2 } (\lambda x \ y. x \text{ iand } \text{next } y) \ xs \ ys)$ 
using big-ior-Cons-alt by auto
have 16:  $\vdash (x \text{ iand } \text{next } y) \text{ ior}$ 
 $(\text{big-ior } (\text{map2 } (\lambda x1 \ y1. x1 \text{ iand } \text{next } y1)$ 
 $\quad ( (\text{foldr } \text{add-to-now-single } xs \ [ittrue, ifalse]))$ 
 $\quad ( (\text{foldr } \text{add-to-next-single } ys \ [ifalse, ifalse])))) \text{ ieqv}$ 
 $\text{big-ior } (\text{map2 } (\lambda x \ y. x \text{ iand } \text{next } y) (x \# xs) (y \# ys))$ 
using 4 15 by auto
show ?thesis
using 10 14 16 5 6 9 by force
qed
qed
qed

```

```

lemma det-big-ior-ieqv-nondet-big-ior-b:
assumes  $\text{length } L1 = \text{length } L2$ 
shows  $\vdash \text{big-ior } (\text{map2 } (\lambda x \ y. x \text{ iand } (\text{next } y))$ 
 $\quad (\text{foldr } (\text{add-to-now-single}) \ L1 \ [ittrue, ifalse])$ 
 $\quad (\text{foldr } (\text{add-to-next-single-a}) \ L2 \ [ifalse, ifalse]))$ 
 $\text{ieqv}$ 
 $\text{big-ior } (\text{map2 } (\lambda x \ y. x \text{ iand } (\text{next } y)) \ L1 \ L2)$ 
using assms
proof (induct L1 L2 rule: list-induct2)
case Nil
then show ?case by (simp add: big-ior-def)
next
case (Cons x xs y ys)
then show ?case
proof (cases xs = []  $\wedge$  ys = [])
case True
then show ?thesis using Cons

```



```

unfolding add-to-now-single-def add-to-next-single-a-def big-ior-def big-iand-def
  by simp
next
case False
then show ?thesis
proof –
  have 1: (foldr add-to-now-single (x # xs) [itrue,ifalse]) =
    ( (map ( $\lambda x1. x \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse])) @
      (map ( $\lambda x1. (\text{inot } x) \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse])))
    by (simp add: add-to-now-single-def)
  have 2: (foldr add-to-next-single-a (y # ys) [ifalse,ifalse]) =
    ( (map ( $\lambda x1. (\text{if } y = \text{ifalse then } x1 \text{ else } y \text{ ior } x1)$ ) (foldr add-to-next-single-a ys [ifalse,ifalse])) @
      (foldr add-to-next-single-a ys [ifalse,ifalse]) )
    by (simp add: add-to-next-single-a-def)
  have 3: length xs = length ys
    by (simp add: Cons.hyphs(1))
  have 4:  $\vdash \text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand next } y)
    (\text{foldr add-to-now-single xs [itrue,ifalse]})
    (\text{foldr add-to-next-single-a ys [ifalse,ifalse]})) \text{ ieqv}$ 
    big-ior (map2 (λx y. x iand next y) xs ys)
    using 3 Cons.hyphs(2) False by fastforce
  have 5: big-ior (map2 (λx y. x iand next y)
    (foldr add-to-now-single (x # xs) [itrue,ifalse])
    (foldr add-to-next-single-a (y # ys) [ifalse,ifalse])) =
    big-ior (map2 (λx y. x iand next y)
    ( (map ( $\lambda x1. x \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse])) @
      (map ( $\lambda x1. (\text{inot } x) \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse])))
    ( (map ( $\lambda x1. (\text{if } y = \text{ifalse then } x1 \text{ else } y \text{ ior } x1)$ ) (foldr add-to-next-single-a ys
[ifalse,ifalse])) @
      (foldr add-to-next-single-a ys [ifalse,ifalse]) )
    )
    using 1 2 by presburger
  have 6: length (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse])) =
    length (map (λx1. y ior x1) (foldr add-to-next-single-a ys [ifalse,ifalse]))
    by (simp add: 3 length-add-to-now-next-a)
  have 7: length (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse])) =
    length (foldr add-to-next-single-a ys [ifalse,ifalse])
    using 6 by auto
  have 75: length (foldr add-to-now-single xs [itrue, ifalse]) =
    length (foldr add-to-next-single-a ys [ifalse, ifalse])
    using 7 by force
  have 8:  $\vdash \text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand next } y)
    ( (map ( $\lambda x1. x \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse])) @
      (map ( $\lambda x1. (\text{inot } x) \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse])))
    ( (map ( $\lambda x1. (\text{if } y = \text{ifalse then } x1 \text{ else } y \text{ ior } x1)$ ) (foldr add-to-next-single-a ys
[ifalse,ifalse])) @
      (foldr add-to-next-single-a ys [ifalse,ifalse]) )
    ) \text{ ieqv}$ 
    big-ior ( (map2 (λx y. x iand next y)
    (map ( $\lambda x1. x \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse]))
    (map ( $\lambda x1. (\text{inot } x) \text{ iand } x1$ ) (foldr add-to-now-single xs [itrue,ifalse])))
    (map ( $\lambda x1. (\text{if } y = \text{ifalse then } x1 \text{ else } y \text{ ior } x1)$ ) (foldr add-to-next-single-a ys
[ifalse,ifalse])) @
      (foldr add-to-next-single-a ys [ifalse,ifalse]) )
    ) \text{ ieqv}

```

```

    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys
[ifalse,ifalse])) )
    @
    (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
    (foldr add-to-next-single-a ys [ifalse,ifalse])) )
  )
  using 6 by force
  have 9: ⊢ big-ior ( (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys
[ifalse,ifalse])) ) )
    @
    (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
    (foldr add-to-next-single-a ys [ifalse,ifalse])) )
  ) ieqv
  (big-ior (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys
[ifalse,ifalse])) ) ) ior
    (big-ior (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
    (foldr add-to-next-single-a ys [ifalse,ifalse])) ) )
  )
  using big-ior-append by blast
  have 10: ⊢ (big-ior (map2 (λx y. x iand next y)
    (map (λx1. x iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
    (map (λx1. (if y = ifalse then x1 else y ior x1)) (foldr add-to-next-single-a ys
[ifalse,ifalse])) ) ) ieqv
    (big-ior (map2 (λx1 y1. (x iand x1) iand next ((if y = ifalse then y1 else y ior y1)))
    ( (foldr add-to-now-single xs [itrue,ifalse]))
    ( (foldr add-to-next-single-a ys [ifalse,ifalse])))))
  )
  using map2-map-a-a[of (foldr add-to-now-single xs [itrue,ifalse])
(foldr add-to-next-single-a ys [ifalse,ifalse]) x y ] using 7 by auto
  have 11: ⊢ (big-ior (map2 (λx y. x iand next y)
    (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
    (foldr add-to-next-single-a ys [ifalse,ifalse])) ) ) ieqv
    (big-ior (map2 (λx1 y1. ((inot x) iand x1) iand next y1)
    ( (foldr add-to-now-single xs [itrue,ifalse]))
    (foldr add-to-next-single-a ys [ifalse,ifalse])) ) )
  )
  using map2-map-b[of (foldr add-to-now-single xs [itrue,ifalse])
(foldr add-to-next-single-a ys [ifalse,ifalse]) x] 75
  by auto
  have 12: ⊢ big-ior (foldr add-to-now-single xs [itrue, ifalse])
  using big-ior-foldr-add-to-now-single-inv by metis
  have 13: 0 < length (foldr add-to-now-single xs [itrue, ifalse])
  using 12 big-ior-empty by auto
  have 14: ⊢ (big-ior (map2 (λx1 y1. (x iand x1) iand next (if y= ifalse then y1 else y ior y1))
    ( (foldr add-to-now-single xs [itrue,ifalse]))
    ( (foldr add-to-next-single-a ys [ifalse,ifalse])))) ior

```

```

    (big-ior (map2 (λx y. x iand next y)
      (map (λx1. (inot x) iand x1) (foldr add-to-now-single xs [itrue,ifalse]))
      (foldr add-to-next-single-a ys [ifalse,ifalse] ) )
    ieqv
    (x iand next y) ior
    (big-ior (map2 (λx1 y1. x1 iand next y1)
      ( (foldr add-to-now-single xs [itrue,ifalse]))
      ( (foldr add-to-next-single-a ys [ifalse,ifalse]))))
    using big-ior-split-off-a[of ( (foldr add-to-now-single xs [itrue,ifalse]))
      ( (foldr add-to-next-single-a ys [ifalse,ifalse])) x y] 12 13
    by (simp add: 75 map2-map-b)
  have 15: ⊢ big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys)) ieqv
    (x iand next y) ior
    big-ior (map2 (λx y. x iand next y) xs ys)
    using big-ior-Cons-alt by auto
  have 16: ⊢ (x iand next y) ior
    (big-ior (map2 (λx1 y1. x1 iand next y1)
      ( (foldr add-to-now-single xs [itrue,ifalse]))
      ( (foldr add-to-next-single-a ys [ifalse,ifalse])))) ieqv
    big-ior (map2 (λx y. x iand next y) (x # xs) (y # ys))
    using 4 15 by auto
  show ?thesis
  using 10 14 16 5 6 9 by force
qed
qed
qed

```

```

lemma mutex-single-filter-sat:
  assumes mutex-single f L1
  shows mutex-single f ( filter (λx. (∃ σ. σ ⊨ x)) L1)
using assms
proof (induct L1)
case Nil
then show ?case by simp
next
case (Cons a L1)
then show ?case
by (simp add: mutex-single-Cons)
qed

```

```

lemma mutex-filter-sat:
  assumes mutex L1
  shows mutex ( filter (λx. (∃ σ. σ ⊨ x)) L1)
using assms
proof (induct L1)
case Nil
then show ?case unfolding mutex-def by simp
next
case (Cons a L1)

```

then show ?case
by (simp add: mutex-Cons mutex-single-filter-sat)
qed

lemma state-qpitl-list-filter-sat:
assumes state-qpitl-list L
shows state-qpitl-list (filter ($\lambda x. (\exists \sigma. \sigma \models x)$) L)
using assms
proof (induct L)
case Nil
then show ?case **by** simp
next
case (Cons $a L$)
then show ?case
by (simp add: state-qpitl-list-def)
qed

lemma big-ior-itrue:
 \vdash big-ior [itrue]
by (simp add: big-ior-defs)

lemma mutex-itrue:
 mutex [itrue]
by (simp add: mutex-Cons mutex-empty mutex-single-empty)

lemma big-ior-mutex-start:
 $(\vdash \text{big-ior} [\text{ifalse}, \text{itrue}]) \wedge \text{mutex} [\text{ifalse}, \text{itrue}]$
unfolding big-ior-def mutex-def
using less-Suc-eq **by** force

4.3.4 GNF lemmas

lemma GNF-defs:
assumes $\vdash F \text{ ieqv } ((F\text{-e iand empty}) \text{ ior } (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) \ F\text{-now } F\text{-next})))$
 state-qpitl $F\text{-e}$
 state-qpitl-list $F\text{-now}$
 length $F\text{-now} = \text{length } F\text{-next}$
shows $(\sigma \models F) \longleftrightarrow$
 $((\text{nlength } \sigma = 0 \wedge ((\text{NNil } (\text{nfirst } \sigma)) \models F\text{-e})) \vee$
 $(\text{nlength } \sigma > 0 \wedge$
 $(\exists i < \text{length } F\text{-now}. ((\text{NNil } (\text{nfirst } \sigma)) \models F\text{-now } !i) \wedge ((\text{ndropn } 1 \ \sigma) \models F\text{-next } !i)))$
proof –
have 1: $(\sigma \models F) \longleftrightarrow$
 $(\sigma \models ((F\text{-e iand empty}) \text{ ior } (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) \ F\text{-now } F\text{-next}))))$

using assms itl-ieq **by** blast
have 2: $(\sigma \models (F\text{-e iand empty})) \longleftrightarrow$

```

  (nlength  $\sigma = 0 \wedge ((NNil (nfirst \sigma)) \models F-e))$ 
  using assms state-qpitl-defs empty-defs iand-defs by blast
have 3:  $length (map2 (\lambda x y. x \text{ iand } next y) F\text{-now } F\text{-next}) = length F\text{-now}$ 
  by (simp add: assms(4))
have 4:
  ( $\sigma \models (big\text{-ior } (map2 (\lambda x y. x \text{ iand } (next y)) F\text{-now } F\text{-next})) \longleftrightarrow$ 
    ( $\exists i < length F\text{-now}. (\sigma \models (\lambda x y. x \text{ iand } (next y)) (F\text{-now}!i) (F\text{-next}!i))$ )
  by (simp add: assms(4) big-ior-map2-defs)
have 5: ( $\exists i < length F\text{-now}. (\sigma \models (\lambda x y. x \text{ iand } (next y)) (F\text{-now}!i) (F\text{-next}!i)) \longleftrightarrow$ 
  ( $nlength \sigma > 0 \wedge$ 
    ( $\exists i < length F\text{-now}. ((NNil (nfirst \sigma)) \models F\text{-now} !i) \wedge ((ndropn 1 \sigma) \models F\text{-next} !i))$  )
  by simp
  (metis assms(3) co.enat.exhaust-sel enat-le-plus-same(2) not-one-le-zero plus-1-eSuc(2)
    state-qpitl-defs state-qpitl-list-defs)
show ?thesis
using 1 2 4 5 by auto
qed

```

```

lemma big-ior-iand-next-big-iand-iimp:
  assumes ( $\vdash big\text{-ior } F$ )
    mutex F
     $length F = length G$ 
  shows  $\vdash big\text{-ior } (map2 (\lambda x y. x \text{ iand } (next y)) F G) \text{ ieqv}$ 
     $big\text{-iand } (map2 (\lambda x y. x \text{ iimp } (next y)) F G)$ 
using assms unfolding valid-def mutex-def
apply (auto simp add: big-ior-defs big-iand-defs map2-iand-next-defs)
apply (metis not-less-iff-gr-or-eq)
by meson

```

```

lemma inot-big-iand-next-iimp-big-iand-iimp-inot:
  assumes ( $\vdash big\text{-ior } F$ )
    mutex F
     $length F = length G$ 
  shows  $\vdash more \text{ iimp } (inot \text{ big-iand } (map2 (\lambda x y. x \text{ iimp } (next y)) F G) \text{ ieqv}$ 
     $big\text{-iand } (map2 (\lambda x y. x \text{ iimp } (next (inot y))) F G))$ 
using assms unfolding mutex-def
apply (auto simp add: big-iand-defs big-ior-defs more-d-def)
apply (metis linorder-less-linear)
by meson

```

```

lemma inot-big-ior-iand-big-ior-iand-inot:
  assumes ( $\vdash big\text{-ior } F$ )
    mutex F
     $length F = length G$ 
  shows  $\vdash (inot (big\text{-ior } (map2 (\lambda x y. x \text{ iand } (next y)) F G)) \text{ ieqv}$ 
     $big\text{-ior } (map2 (\lambda x y. x \text{ iand } (next (inot y))) F G))$ 
using assms unfolding mutex-def

```

```

apply (auto simp add: big-ior-defs big-iand-defs map2-iand-inot-defs)
apply meson
by (metis not-less-iff-gr-or-eq)

lemma inot-big-ior-iand-big-ior-iand-next-inot:
  assumes ( $\vdash$  big-ior  $F$ )
    mutex  $F$ 
    length  $F$  = length  $G$ 
  shows  $\vdash$  more iimp (inot (big-ior (map2 ( $\lambda x y. x$  iand (next  $y$ ))  $F$   $G$ )) ieqv
    big-ior (map2 ( $\lambda x y. x$  iand (next (inot  $y$ )))  $F$   $G$ ))
using assms unfolding mutex-def
apply (auto simp add: big-ior-defs big-iand-defs map2-iand-inot-next-defs more-d-def)
apply (metis One-nat-def gr-zeroI map2-iand-inot-next-defs not-one-le-zero semantics-qpitl.simps(4))
by (metis linorder-less-linear)

```

```

lemma more-iimp-inot-big-ior-big-ior-map-inot:
  assumes ( $\vdash$  big-ior  $F$ )
    mutex  $F$ 
    length  $F$  = length  $G$ 
shows  $\vdash$  more iimp
  ((inot (big-ior (map2 ( $\lambda x y. x$  iand (next  $y$ ))  $F$   $G$ ))) ieqv
    big-ior (map2 ( $\lambda x y. x$  iand (next  $y$ ))  $F$  (map ( $\lambda x. (inot x)$ )  $G$ )))
proof –
  have 1: (map2 ( $\lambda x y. x$  iand (next  $y$ ))  $F$  (map ( $\lambda x. (inot x)$ )  $G$ )) =
    (map2 ( $\lambda x y. x$  iand (next (inot  $y$ )))  $F$   $G$ )
    using map2-map[of  $F$   $G$  ( $\lambda x y. x$  iand (next  $y$ )) id ( $\lambda x. (inot x)$ )]
    assms by auto
  show ?thesis
  using 1 assms inot-big-ior-iand-big-ior-iand-next-inot by auto
qed

```

```

lemma GNF-iimp-step:
  assumes  $\vdash F$  ieqv (( $F$ -e iand empty) ior (big-ior (map2 ( $\lambda x y. x$  iand (next  $y$ ))  $F$ -now  $F$ -next)))
    state-qpitl  $F$ -e
    state-qpitl-list  $F$ -now
    length  $F$ -now = length  $F$ -next
   $\vdash G$  ieqv (( $G$ -e iand empty) ior (big-ior (map2 ( $\lambda x y. x$  iand (next  $y$ ))  $G$ -now  $G$ -next)))
    state-qpitl  $G$ -e
    state-qpitl-list  $G$ -now
    length  $G$ -now = length  $G$ -next
shows ( $\sigma \models F$  iimp  $G$ )  $\longleftrightarrow$ 
  ( $\sigma \models$  ( ( (inot  $F$ -e) ior  $G$ -e) iand empty) ior
    (big-ior
      (map2 ( $\lambda x y. x$  iand (next  $y$ ))
        ((add-to-now  $F$ -now) @  $G$ -now)

```

```

      ((map (λx. (inot x)) (add-to-next F-next )) @ G-next) )))
proof (cases nlength σ = 0)
case True
then show ?thesis
proof -
  have 1: (σ ⊨ F iimp G) ⟷ (¬(σ ⊨ F) ∨ (σ ⊨ G))
    by auto
  have 2: (¬(σ ⊨ F) ∨ (σ ⊨ G)) ⟷
    (¬((NNil (nfirst σ)) ⊨ F-e) ∨ ((NNil (nfirst σ)) ⊨ G-e))
  using
    assms True
    by (metis (mono-tags, lifting) GNF-defs less-numeral-extra(3))
  have 3: state-qpitl (inot F-e ior G-e)
    by (simp add: assms(2) assms(6) inot-d-def ior-d-def)
  have 35: state-qpitl-list [ifalse, ittrue]
    unfolding state-qpitl-list-def ittrue-d-def inot-d-def by auto
  have 4: state-qpitl-list ((add-to-now F-now ) @ G-now)
    by (simp add: add-to-now-def assms(3) assms(7) state-qpitl-foldr-add-to-now-single-inv-a
      state-qpitl-list-append)
  have 5: length (add-to-now F-now @ G-now) =
    length (map inot-d (add-to-next F-next ) @ G-next)
    by (simp add: add-to-next-def add-to-now-def assms(4) assms(8) length-add-to-now-next-gen)
  have 6: (¬((NNil (nfirst σ)) ⊨ F-e) ∨ ((NNil (nfirst σ)) ⊨ G-e)) ⟷
    (σ ⊨ ( ( (inot F-e) ior G-e) iand empty) ior
      (big-ior
        (map2 (λ x y. x iand (next y))
          ((add-to-now F-now )@G-now)
          ((map (λx. (inot x)) (add-to-next F-next )) @ G-next) ) ) )
  using GNF-defs[of
    ( ( (inot F-e) ior G-e) iand empty) ior
    (big-ior
      (map2 (λ x y. x iand (next y))
        ((add-to-now F-now )@G-now)
        ((map (λx. (inot x)) (add-to-next F-next )) @ G-next) ) )
    ( (inot F-e) ior G-e)
    ((add-to-now F-now )@G-now)
    ((map (λx. (inot x)) (add-to-next F-next )) @ G-next) σ ]
  using assms
  using 3 4 5 True by force
show ?thesis
using 1 2 6 by blast
qed
next
case False
then show ?thesis
proof -
  have 8: (σ ⊨ F iimp G) ⟷ (σ ⊨ (inot F) ior G)
    by auto
  have 9: (σ ⊨ (inot F)) ⟷
    (σ ⊨ (inot (big-ior (map2 (λ x y. x iand (next y)) F-now F-next))))

```

```

using False assms(1) by auto
have 10:  $\vdash$  (big-ior (map2 ( $\lambda x y. x \text{ iand } (\text{next } y)$ ) F-now F-next)) ieqv
  (big-ior (map2 ( $\lambda x y. x \text{ iand } (\text{next } y)$ )
    (foldr (add-to-now-single) F-now [itrue])
    (foldr (add-to-next-single) F-next [ifalse]))))
using assms(4) det-big-ior-ieqv-nondet-big-ior-a by auto
have 101: ( $\sigma \models (\text{inot } (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) \text{ F-now F-next})))) \longleftrightarrow$ 
  ( $\sigma \models (\text{inot } (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y))
    (\text{foldr } (\text{add-to-now-single}) \text{ F-now } [\text{itrue}])
    (\text{foldr } (\text{add-to-next-single}) \text{ F-next } [\text{ifalse}])))$ ))
using 10 by auto
have 11:  $\vdash$  big-ior (foldr (add-to-now-single) F-now [itrue])
using big-ior-foldr-add-to-now-single-inv-a by blast
have 12: mutex (foldr (add-to-now-single) F-now [itrue])
using mutex-foldr-add-to-now-single-inv-a by blast
have 13: length (foldr (add-to-now-single) F-now [itrue]) =
  length (foldr (add-to-next-single) F-next [ifalse])
by (simp add: assms(4) length-add-to-now-next-gen)
have 14:  $\vdash$  more iimp
  ( $(\text{inot } (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y))
    (\text{foldr } (\text{add-to-now-single}) \text{ F-now } [\text{itrue}])
    (\text{foldr } (\text{add-to-next-single}) \text{ F-next } [\text{ifalse}]))) \text{ieqv}$ 
    (big-ior
      (map2 ( $\lambda x y. x \text{ iand } (\text{next } y)$ )
        (foldr (add-to-now-single) F-now [itrue])
        ((map ( $\lambda x. (\text{inot } x)$ ) (foldr (add-to-next-single) F-next [ifalse])))))))
using 11 12 13 more-iimp-inot-big-ior-big-ior-map-inot by blast
have 15: ( $\sigma \models (\text{inot } (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y))
  (\text{foldr } (\text{add-to-now-single}) \text{ F-now } [\text{itrue}])
  (\text{foldr } (\text{add-to-next-single}) \text{ F-next } [\text{ifalse}])))$ ))  $\longleftrightarrow$ 
  ( $\sigma \models (\text{big-ior}$ 
    (map2 ( $\lambda x y. x \text{ iand } (\text{next } y)$ )
      (foldr (add-to-now-single) F-now [itrue])
      ((map ( $\lambda x. (\text{inot } x)$ ) (foldr (add-to-next-single) F-next [ifalse])))))))
using 14 False by auto
  (metis (no-types, lifting) empty-d-def empty-defs inot-defs,
    metis (no-types, lifting) empty-d-def empty-defs inot-defs)
have 16: ( $\sigma \models G$ )  $\longleftrightarrow$ 
  ( $\sigma \models (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) \text{ G-now G-next}))$ )
using False assms(5) by auto
have 17: ( $\sigma \models (\text{inot } F) \text{ ior } G$ )  $\longleftrightarrow$ 
  ( $\sigma \models (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y))
    (\text{foldr } (\text{add-to-now-single}) \text{ F-now } [\text{itrue}])
    ((\text{map } (\lambda x. (\text{inot } x)) (\text{foldr } (\text{add-to-next-single}) \text{ F-next } [\text{ifalse}])))$ )) ior
    (big-ior (map2 ( $\lambda x y. x \text{ iand } (\text{next } y)$ ) G-now G-next)))
using 101 15 16 9 by auto
have 18: ( $\sigma \models (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y))
  (\text{foldr } (\text{add-to-now-single}) \text{ F-now } [\text{itrue}])
  ((\text{map } (\lambda x. (\text{inot } x)) (\text{foldr } (\text{add-to-next-single}) \text{ F-next } [\text{ifalse}])))$ )) ior
    (big-ior (map2 ( $\lambda x y. x \text{ iand } (\text{next } y)$ ) G-now G-next)))  $\longleftrightarrow$ 

```



```

      (σ ⊨ (big-ior (map2 (λ x y. x iand (next y))
        ((foldr (add-to-now-single) F-now [itrue]) @ G-now)
        ((map (λx. (inot x)) (foldr (add-to-next-single) F-next [ifalse])) @ G-next) ) ))
    using 13 big-ior-append by auto
    show ?thesis unfolding add-to-now-def add-to-next-def
    using 17 18 False by auto
qed
qed

```

lemma *GNF-inot-step*:

```

assumes ⊢ F ieqv (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y)) F-now F-next)))
  state-qpitl F-e
  state-qpitl-list F-now
  length F-now = length F-next
shows (σ ⊨ (inot F) ) ⟷
  (σ ⊨ ( (inot F-e) iand empty) ior
    (big-ior
      (map2 (λ x y. x iand (next y))
        ((add-to-now F-now ) @[ifalse])
        ((map (λx. (inot x)) (add-to-next F-next )) @[ifalse]) )))
proof –
have 1: (σ ⊨ (inot F) ) ⟷ (σ ⊨ F iimp ifalse)
  by auto
have 2: ⊢ ifalse ieqv ifalse iand empty ior big-ior (map2 (λx y. x iand next y) [ifalse] [ifalse])
  by (auto simp add: big-ior-defs)
have 3: (σ ⊨ F iimp ifalse) ⟷
  (σ ⊨ ( (inot F-e) ior ifalse) iand empty) ior
    (big-ior
      (map2 (λ x y. x iand (next y))
        ((add-to-now F-now ) @[ifalse])
        ((map (λx. (inot x)) (add-to-next F-next )) @[ifalse]) )))
using GNF-iimp-step[of F F-e F-now F-next ifalse ifalse [ifalse] [ifalse] σ]
  asms
by (metis 2 length-Cons less-Suc0 list.size(3) nth-Cons-0 state-qpitl.simps(1)
  state-qpitl-list-defs)
show ?thesis
using 3 by auto
qed

```

lemma *GNF-inot-step-valid*:

```

assumes ⊢ F ieqv (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y)) F-now F-next)))
  state-qpitl F-e
  state-qpitl-list F-now
  length F-now = length F-next
shows ⊢ (inot F) ieqv
  ( (inot F-e) iand empty) ior
    (big-ior

```

```

      (map2 (λ x y. x iand (next y))
        ((add-to-now F-now ) @[ifalse])
        ((map (λx. (inot x)) (add-to-next F-next )) @[ifalse]) )))
using assms GNF-inot-step[of F F-e F-now F-next ]
using itl-ieq by blast

```

lemma *GNF-inot-inot-step-valid:*

```

assumes ⊢ F ieqv (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y)) F-now F-next)))
      state-qpitl F-e
      state-qpitl-list F-now
      length F-now = length F-next
shows ⊢ (inot (inot F)) ieqv
      ( ( F-e iand empty) ior
        (big-ior (map2 (λ x y. x iand (next y)) F-now F-next )))
using assms by simp

```

lemma *GNF-ior-step:*

```

assumes ⊢ F ieqv (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y)) F-now F-next)))
      state-qpitl F-e
      state-qpitl-list F-now
      length F-now = length F-next
      ⊢ G ieqv (( G-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y)) G-now G-next)))
      state-qpitl G-e
      state-qpitl-list G-now
      length G-now = length G-next
shows (σ ⊨ F ior G) ⟷
      (σ ⊨ ( ( F-e ior G-e) iand empty) ior
        (big-ior (map2 (λ x y. x iand (next y)) (F-now @ G-now) (F-next @ G-next) )))

```

proof –

```

have 1: (σ ⊨ F ior G) ⟷
      (σ ⊨ (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y)) F-now F-next))) ior
        (( G-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y)) G-now G-next))))
      using assms by auto
have 2: (σ ⊨ (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y)) F-now F-next))) ior
      (( G-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y)) G-now G-next)))) ⟷
      (σ ⊨ ( ( F-e ior G-e) iand empty) ior
        (big-ior (map2 (λ x y. x iand (next y)) (F-now @ G-now) (F-next @ G-next) )))
      using assms big-ior-map2-append by fastforce
show ?thesis
using 1 2 by blast
qed

```

lemma *GNF-schop-step:*

```

assumes ⊢ F ieqv (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y)) F-now F-next)))
      state-qpitl F-e

```

```

state-qpitl-list F-now
length F-now = length F-next
⊢ G ieqv (( G-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y)) G-now G-next)))
state-qpitl G-e
state-qpitl-list G-now
length G-now = length G-next
shows (σ ⊨ F schop G) ⟷
  (σ ⊨ (F-e iand G-e) iand empty ior
    (big-ior (map2 (λ x y. x iand next y)
      (F-now @ (map (λx. x iand F-e) G-now))
      ((map (λx. x schop G) F-next) @ G-next)))) )
proof (cases nlength σ = 0)
case True
then show ?thesis
proof -
  have 1: (σ ⊨ F schop G) ⟷ (σ ⊨ F) ∧ (σ ⊨ G)
    using True by (auto simp add: ntaken-all zero-enat-def)
  have 2: (σ ⊨ F) ∧ (σ ⊨ G) ⟷ ((NNil (nfirst σ)) ⊨ F-e) ∧ ((NNil (nfirst σ)) ⊨ G-e)
    using GNF-defs True assms
    by (metis less-numeral-extra(3))
  have 3: state-qpitl (F-e iand G-e)
    by (simp add: assms(2) assms(6) iand-d-def inot-d-def ior-d-def)
  have 4: state-qpitl-list (F-now @ map (λx. x iand F-e) G-now)
    using assms state-qpitl-list-append state-qpitl-list-map-alt by blast
  have 5: length (F-now @ map (λx. x iand F-e) G-now) =
    length (map (λx. x schop G) F-next @ G-next)
    by (simp add: assms(4) assms(8))
  have 6: ((NNil (nfirst σ)) ⊨ F-e) ∧ ((NNil (nfirst σ)) ⊨ G-e) ⟷
    (σ ⊨ (F-e iand G-e) iand empty ior
      (big-ior (map2 (λ x y. x iand next y)
        (F-now @ (map (λx. x iand F-e) G-now))
        ((map (λx. x schop G) F-next) @ G-next)))) )
    using GNF-defs[of (F-e iand G-e) iand empty ior
      (big-ior (map2 (λ x y. x iand next y)
        (F-now @ (map (λx. x iand F-e) G-now))
        ((map (λx. x schop G) F-next) @ G-next)))
      (F-e iand G-e) (F-now @ (map (λx. x iand F-e) G-now))
      ((map (λx. x schop G) F-next) @ G-next) σ]
    using assms
    by (simp add: 3 4 True)
  show ?thesis
  using 1 2 6 by presburger
qed
next
case False
  assume a: nlength σ ≠ 0
  show ?thesis (is ?lhs ⟷ ?rhs)
  proof
    assume l: ?lhs
    show ?rhs

```

```

proof –
  have 7:  $(\exists i. \text{enat } i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \ \sigma \models F) \wedge (\text{ndropn } i \ \sigma \models G))$ 
    using l by simp
  obtain i where 8:  $\text{enat } i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \ \sigma \models F) \wedge (\text{ndropn } i \ \sigma \models G)$ 
    using 7 by blast
  have 9:  $i=0 \implies ?rhs$ 
    proof –
      assume b:i=0
      have 10:  $(\text{ntaken } 0 \ \sigma \models F) \wedge (\sigma \models G)$ 
        using 8 b by auto
      have 11:  $(\sigma \models F-e)$ 
        using assms
        by (metis (mono-tags, lifting) 8 GNF-defs b less-numeral-extra(3) min-def ntaken-nfirst
          ntaken-nlength state-qpitl-defs zero-enat-def)
      have 12:  $\sigma \models F-e \text{ iand } \text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) \ G\text{-now } G\text{-next})$ 
        using 10 11 a assms(5) by force
      have 13:  $\sigma \models \text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) \ (\text{map } (\lambda x. x \text{ iand } F-e) \ G\text{-now}) \ G\text{-next})$ 
        using iand-big-ior-dist 12 assms(8) itl-ieq by blast
      show ?thesis
      using 13 assms(4) assms(8) big-ior-map2-append by force
    qed
have 14:  $0 < i \wedge \text{enat } i \leq \text{nlength } \sigma \implies ?rhs$ 
  proof –
    assume c: 0 < i ∧ enat i ≤ nlength σ
    have 15:  $(\text{ntaken } i \ \sigma \models F) \wedge (\text{ndropn } i \ \sigma \models G)$ 
      by (simp add: 8)
    have 16:  $(\text{ntaken } i \ \sigma \models (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) \ F\text{-now } F\text{-next})))$ 
      using 8 assms(1) c enat-0-iff(1) by auto
    have 17:  $((\text{ntaken } i \ \sigma) \models (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) \ F\text{-now } F\text{-next}))) \longleftrightarrow$ 
       $(\exists ia < \text{length } F\text{-now}. \text{ntaken } i \ \sigma \models F\text{-now} ! ia \text{ iand } \text{next } F\text{-next} ! ia)$ 
      using 16 big-ior-map2-defs[of F-now F-next (ntaken i σ) ( $\lambda x y. x \text{ iand } \text{next } y$ )]
      assms by blast
    have 18:  $\text{length } F\text{-now} = \text{length } (\text{map } (\lambda x. x \text{ schop } G) \ F\text{-next})$ 
      using assms(4) by fastforce
    have 19:  $(\sigma \models (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) \ F\text{-now } (\text{map } (\lambda x. x \text{ schop } G) \ F\text{-next})))) \longleftrightarrow$ 
       $(\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now} ! i \text{ iand } \text{next } \text{map } (\lambda x. x \text{ schop } G) \ F\text{-next} ! i)$ 
      using 18 big-ior-map2-defs[of F-now ( $\text{map } (\lambda x. x \text{ schop } G) \ F\text{-next}$ )  $\sigma$  ( $\lambda x y. x \text{ iand } \text{next } y$ )]
      by blast
    obtain j where 20:  $j < \text{length } F\text{-now} \wedge (\text{ntaken } i \ \sigma \models F\text{-now} ! j \text{ iand } \text{next } F\text{-next} ! j)$ 
      using 16 17 by blast
    have 21:  $\text{enat } (i-1) \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0)$ 
      by (metis One-nat-def c enat-minus-mono1 idiff-enat-enat)
    have 22:  $(\text{ntaken } (i-1) \ (\text{ndropn } (\text{Suc } 0) \ \sigma) \models F\text{-next} ! j)$ 
      by (metis 20 One-nat-def Suc-leI c diff-add iand-defs ntaken-ndropn-swap plus-enat-simps(1)
        semantics-qpitl.simps(4))
    have 23:  $(\text{ndropn } (i-1) \ (\text{ndropn } (\text{Suc } 0) \ \sigma) \models G)$ 
      using 8 by (simp add: c ndropn-ndropn)
    have 24:  $(\sigma \models F\text{-now} ! j \text{ iand } \text{next } ((F\text{-next} ! j) \text{ schop } G))$ 

```

```

    using 20 15 by simp
    (metis 21 22 23 assms(3) ntaken-nfirst state-qpitl-defs state-qpitl-list-defs)
  have 25:  $(\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now} ! i \text{ iand next map } (\lambda x. x \text{ schop } G) F\text{-next} ! i)$ 
    using 20 24 assms(4) by fastforce
  show ?thesis
    using 18 19 25 assms(8) big-ior-map2-append by auto
qed
show ?thesis
using 14 8 9 by blast
qed
next
assume r: ?rhs
show ?lhs
proof -
  have 26:  $\sigma \models \text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand next } y) (F\text{-now} @ \text{map } (\lambda x. x \text{ iand } F\text{-e}) G\text{-now}) (\text{map } (\lambda x. x \text{ schop } G) F\text{-next} @ G\text{-next}))$ 
    using a r by auto
  have 27:  $\text{length } F\text{-now} = \text{length } (\text{map } (\lambda x. x \text{ schop } G) F\text{-next})$ 
    by (simp add: assms(4))
  have 28:  $\text{length } (\text{map } (\lambda x. x \text{ iand } F\text{-e}) G\text{-now}) = \text{length } (G\text{-next})$ 
    using assms(8) by fastforce
  have 29:  $(\sigma \models (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand next } y) (F\text{-now}) (\text{map } (\lambda x. x \text{ schop } G) F\text{-next}))))$ 
     $\vee$ 
     $(\sigma \models ((\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand next } y) (\text{map } (\lambda x. x \text{ iand } F\text{-e}) G\text{-now}) (G\text{-next}))))$ 
    using 26 27 28 big-ior-map2-append itl-ieq ior-defs by blast
  have 30:  $(\sigma \models (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand next } y) (F\text{-now}) (\text{map } (\lambda x. x \text{ schop } G) F\text{-next})))) \longleftrightarrow$ 
     $(\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now} ! i \text{ iand next map } (\lambda x. x \text{ schop } G) F\text{-next} ! i)$ 
    using big-ior-map2-defs[of F-now (map (λx. x schop G) F-next) σ (λx y. x iand next y)]
    using 27 by fastforce
  have 31:  $(\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now} ! i \text{ iand next map } (\lambda x. x \text{ schop } G) F\text{-next} ! i) \implies ?lhs$ 
  proof -
    assume d:  $(\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now} ! i \text{ iand next map } (\lambda x. x \text{ schop } G) F\text{-next} ! i)$ 
    obtain i where 32:  $i < \text{length } F\text{-now} \wedge (\sigma \models F\text{-now} ! i \text{ iand next map } (\lambda x. x \text{ schop } G) F\text{-next} ! i)$ 
    using d by blast
    have 33:  $(\sigma \models F\text{-now} ! i)$ 
    using 32 iand-defs by blast
    have 34:  $\text{map } (\lambda x. x \text{ schop } G) F\text{-next} ! i = (F\text{-next} ! i) \text{ schop } G$ 
    using 32 assms(4) by auto
    have 35:  $1 \leq \text{nlength } \sigma \wedge (\exists ia. \text{enat } ia \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge (\text{ntaken } ia (\text{ndropn } (\text{Suc } 0) \sigma) \models F\text{-next} ! i) \wedge (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0) \sigma) \models G))$ 
    using 32 34 by simp
    obtain ia where 36:  $\text{enat } ia \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge (\text{ntaken } ia (\text{ndropn } (\text{Suc } 0) \sigma) \models F\text{-next} ! i) \wedge (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0) \sigma) \models G)$ 
    using 35 by blast
    have 37:  $(\text{ntaken } (ia+1) (\sigma) \models (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) F\text{-now } F\text{-next}))) \longleftrightarrow$ 
     $(\exists i < \text{length } F\text{-now}. \text{ntaken } (ia+1) (\sigma) \models F\text{-now} ! i \text{ iand next } F\text{-next} ! i)$ 
    using assms big-ior-map2-defs[of F-now F-next (ntaken (ia+1) (σ)) (λ x y. x iand (next y))] by
blast
  have 38:  $(\text{ntaken } (ia+1) \sigma \models F\text{-now} ! i \text{ iand next } F\text{-next} ! i)$ 
    by simp

```

```

    (metis 32 33 35 36 One-nat-def add commute assms(3) enat-min-eq enat-ord-simps(1)
      le-add1 ntaken-ndropn-swap ntaken-nfirst one-enat-def plus-1-eq-Suc state-qpitl-defs
      state-qpitl-list-defs)
  have 39: (ntaken (ia+1)  $\sigma \models F$ )
    using 32 37 38 assms(1) by auto
  have 40: (ndropn (ia+1)  $\sigma \models G$ )
    by (metis 36 One-nat-def add commute ndropn-ndropn)
  have 41: (ia+1)  $\leq$  nlength  $\sigma$ 
    by (metis 35 36 One-nat-def enat-min-eq one-enat-def plus-enat-simps(1))
  show ?thesis
    using 39 40 41 by auto
qed
have 42: ( $\sigma \models ((big\text{-}ior (map2 (\lambda x y. x \text{ iand } next\ y) (map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now) (G\text{-}next) ))$ 
 $\longleftrightarrow$ 
  ( $\exists i < \text{length } (map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now). \sigma \models map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now ! i \text{ iand } next$ 
 $G\text{-}next ! i$ )
    using big-ior-map2-defs[of (map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now) G\text{-}next  $\sigma (\lambda x y. x \text{ iand } next\ y)$ ]
    using 28 by fastforce
  have 43: ( $\exists i < \text{length } (map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now). \sigma \models map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now ! i \text{ iand } next$ 
 $G\text{-}next ! i \implies ?lhs$ )
    proof -
      assume e: ( $\exists i < \text{length } (map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now). \sigma \models map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now ! i \text{ iand } next$ 
 $G\text{-}next ! i$ )
      obtain i where 44:  $i < \text{length } (map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now) \wedge (\sigma \models map (\lambda x. x \text{ iand } F\text{-}e) G\text{-}now ! i \text{ iand } next$ 
 $G\text{-}next ! i)$ 
      using e by blast
      have 45: (ntaken 0  $\sigma \models (G\text{-}now ! i) \text{ iand } F\text{-}e$ )
        using assms
        by (metis 28 44 iand-defs ntaken-0 nth-map state-qpitl-defs state-qpitl-list-defs)
      have 46: (ntaken 0  $\sigma \models F$ )
        using 45 assms(1) by fastforce
      have 47: (ndropn 0  $\sigma \models G\text{-}now ! i \text{ iand } next\ G\text{-}next ! i$ )
        using 44 by fastforce
      have 48: ((ndropn 0  $\sigma \models (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next\ y)) G\text{-}now\ G\text{-}next))) \longleftrightarrow$ 
 $(\exists i < \text{length } G\text{-}now. ndropn\ 0\ \sigma \models G\text{-}now ! i \text{ iand } next\ G\text{-}next ! i)$ 
        using big-ior-map2-defs[of G\text{-}now G\text{-}next (ndropn 0  $\sigma$ ) ( $\lambda x y. x \text{ iand } (next\ y)$ )]
        using assms(8) by fastforce
      have 49: (ndropn 0  $\sigma \models G$ )
        using 44 47 48 assms(5) by auto
      show ?thesis
        by (metis 46 49 semantics-qpitl.simps(5) zero-enat-def zero-le)
    qed
  show ?thesis
    using 29 30 31 42 43 by fastforce
qed
qed
qed

```

lemma GNF-schopstar-step:

assumes $\vdash F \text{ ieqv } ((F\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next y)) F\text{-}now F\text{-}next)))$
 $state\text{-}qpitl\ F\text{-}e$
 $state\text{-}qpitl\text{-}list\ F\text{-}now$
 $length\ F\text{-}now = length\ F\text{-}next$
shows $(\sigma \models schopstar\ F) \longleftrightarrow$
 $(\sigma \models (itrue \text{ iand empty}) \text{ ior}$
 $(big\text{-}ior (map2 (\lambda x y. x \text{ iand } next\ y)\ F\text{-}now (map (\lambda x. x\ schop\ (s chopstar\ F))\ F\text{-}next))))$
 $(is\ ?lhs \longleftrightarrow ?rhs)$
proof –
have 1: $(\sigma \models schopstar\ F) \longleftrightarrow$
 $(\exists l. nnth\ l\ 0 = 0 \wedge nfinite\ l \wedge nidx\ l \wedge enat\ (nlast\ l) = nlength\ \sigma \wedge nfinite\ \sigma \wedge$
 $(\forall i. enat\ i < nlength\ l \longrightarrow (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models F)))$
by *simp*
have 2: $?lhs \implies ?rhs$
proof –
assume $l: ?lhs$
obtain l **where** 3: $nnth\ l\ 0 = 0 \wedge nfinite\ l \wedge nidx\ l \wedge enat\ (nlast\ l) = nlength\ \sigma \wedge nfinite\ \sigma \wedge$
 $(\forall i. enat\ i < nlength\ l \longrightarrow (nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)) \models F))$
using $l\ 1$ **by** *blast*
have 4: $nlength\ l = 0 \implies ?rhs$
proof –
assume $n: nlength\ l = 0$
have 5: $nlength\ \sigma = 0$
by $(metis\ 3\ n\ nnth\text{-}nlast\ the\text{-}enat\text{-}0\ zero\text{-}enat\text{-}def)$
show *?thesis*
by $(simp\ add: 5)$
qed
have 6: $0 < nlength\ l \implies ?rhs$
proof –
assume $g: 0 < nlength\ l$
have 7: $0 < nlength\ \sigma$
by $(metis\ 3\ g\ gr\text{-}zeroI\ less\text{-}numeral\text{-}extra(3)\ nfinite\text{-}conv\text{-}nlength\text{-}enat\ nidx\text{-}less\text{-}last\text{-}1$
 $nnth\text{-}nlast\ the\text{-}enat.\text{simps}\ the\text{-}enat\text{-}0)$
have 8: $(nsubn\ \sigma\ (nnth\ l\ 0)\ (nnth\ l\ (Suc\ 0))) \models F$
using $3\ g$ **by** $(metis\ zero\text{-}enat\text{-}def)$
have 9: $(nsubn\ \sigma\ (nnth\ l\ 0)\ (nnth\ l\ (Suc\ 0))) = (ntaken\ (nnth\ l\ (Suc\ 0))\ \sigma)$
using 3 **by** $(simp\ add: nsubn\text{-}def1)$
have 10: $(ntaken\ (nnth\ l\ (Suc\ 0))\ \sigma) \models F$
using $9\ 8$ **by** *auto*
have 11: $(ntaken\ (nnth\ l\ (Suc\ 0))\ \sigma) \models ((F\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next\ y))\ F\text{-}now\ F\text{-}next)))$
using $10\ asms(1)\ itl\text{-}ieq$ **by** *blast*
have 12: $(ntaken\ (nnth\ l\ (Suc\ 0))\ \sigma) \models (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next\ y))\ F\text{-}now\ F\text{-}next))$
by $(metis\ 11\ 3\ 7\ 9\ One\text{-}nat\text{-}def\ eSuc\text{-}enat\ empty\text{-}defs\ enat\text{-}defs(1)\ g\ iand\text{-}defs\ ileI1$
 $ior\text{-}defs\ less\text{-}numeral\text{-}extra(1)\ linorder\text{-}le\text{-}cases\ nidx\text{-}gr\text{-}first\ nsubn\text{-}nlength\text{-}gr\text{-}one$
 $ntaken\text{-}all\ zero\text{-}less\text{-}iff\text{-}neq\text{-}zero)$
have 13: $0 < nlength\ (ntaken\ (nnth\ l\ (Suc\ 0))\ \sigma)$
by $(metis\ 3\ 7\ 9\ One\text{-}nat\text{-}def\ eSuc\text{-}enat\ enat\text{-}defs(1)\ g\ ileI1\ less\text{-}numeral\text{-}extra(1)$
 $linorder\text{-}le\text{-}cases\ nidx\text{-}gr\text{-}first\ nsubn\text{-}nlength\text{-}gr\text{-}one\ ntaken\text{-}all)$
have 14: $(nlength\ (ntaken\ (nnth\ l\ (Suc\ 0))\ \sigma) > 0 \wedge$

$(\exists i < \text{length } F\text{-now}. ((NNil (nfirst (ntaken (nnth l (Suc 0)) \sigma))) \models F\text{-now } !i) \wedge$
 $((ndropn 1 (ntaken (nnth l (Suc 0)) \sigma)) \models F\text{-next } !i)))$
using *GNF-defs*[of *F F-e F-now F-next (ntaken (nnth l (Suc 0)) \sigma)*] 12 13
by (*metis 10 assms(1) assms(2) assms(3) assms(4) order-less-irrefl*)
obtain *i where 15: $i < \text{length } F\text{-now} \wedge ((NNil (nfirst (ntaken (nnth l (Suc 0)) \sigma))) \models F\text{-now } !i) \wedge$*
 $((ndropn 1 (ntaken (nnth l (Suc 0)) \sigma)) \models F\text{-next } !i)$
using 14 **by** *blast*
have 19: $((ndropn 1 \sigma) \models F\text{-next } !i \text{ schop } (\text{schoptstar } F)) \longleftrightarrow$
 $(\exists ia. \text{enat } ia \leq \text{nlength } \sigma - \text{enat } (Suc 0) \wedge$
 $(ntaken ia (ndropn (Suc 0) \sigma) \models F\text{-next } !i) \wedge$
 $(\exists l. \text{nnth } l 0 = 0 \wedge$
 $\text{nfinite } l \wedge$
 $\text{nidx } l \wedge$
 $\text{enat } (\text{nlast } l) = \text{nlength } \sigma - \text{enat } (Suc 0) - \text{enat } ia \wedge \text{nfinite } \sigma \wedge$
 $(\forall i. \text{enat } i < \text{nlength } l \longrightarrow$
 $(\text{nsubn } (ndropn ia (ndropn (Suc 0) \sigma)) (\text{nnth } l i) (\text{nnth } l (Suc i)) \models F))))$
by *simp*
have 191: $\text{enat } ((\text{nnth } l (Suc 0)) - 1) \leq \text{nlength } \sigma - \text{enat } (Suc 0)$
by (*metis 3 One-nat-def eSuc-enat enat-minus-mono1 enat-ord-simps(1) g idiff-enat-enat*
ileI1 nidx-all-le-nlast zero-enat-def)
have 192: $\text{ntaken } ((\text{nnth } l (Suc 0)) - 1) (ndropn (Suc 0) \sigma) \models F\text{-next } !i$
by (*metis 15 191 3 7 One-nat-def Suc-leI diff-add eSuc-enat enat-min-eq g ileI1*
less-numeral-extra(1) nidx-gr-first ntaken-ndropn-swap zero-enat-def)
have 193: $\text{nnth } (nmap (\lambda x. x - (\text{nnth } l (Suc 0))) (ndropn (Suc 0) l)) 0 = 0$
using *enat-defs(1)* **by** *auto*
have 194: $\text{nfinite } (nmap (\lambda x. x - (\text{nnth } l (Suc 0))) (ndropn (Suc 0) l))$
by (*simp add: 3*)
have 195: $\text{nidx } (ndropn 1 l)$
by (*metis 3 g ileI1 nidx-ndropn one-eSuc one-enat-def*)
have 196: $\text{nnth } (ndropn 1 l) 0 = (\text{nnth } l (Suc 0))$
by *simp*
have 197: $\text{nidx } (nmap (\lambda x. x - (\text{nnth } l (Suc 0))) (ndropn (Suc 0) l))$
using *nidx-shiftm*[of $(ndropn 1 l) (\text{nnth } l (Suc 0))$]
by (*metis 195 196 One-nat-def*)
have 198: $\text{nlast } (ndropn (Suc 0) l) = \text{nlast } l$
by (*metis Suc-ile-eq g nappend-NNil nappend-ntaken-ndropn nlast-NCons ntaken-0 zero-enat-def*)
have 199: $\text{nlast } (nmap (\lambda x. x - \text{nnth } l (Suc 0)) (ndropn (Suc 0) l)) =$
 $\text{nlength } \sigma - \text{nnth } l (Suc 0)$
using *nlast-nmap*[of $(ndropn (Suc 0) l) (\lambda x. x - (\text{nnth } l (Suc 0)))$]
using 194 *nfinite-nmap*
by (*metis 198 3 idiff-enat-enat*)
have 200: $\text{nlength } \sigma - \text{nnth } l (Suc 0) = \text{nlength } \sigma - \text{enat } (Suc 0) - ((\text{nnth } l (Suc 0)) - 1)$
by *auto*
 $(\text{metis 3 One-nat-def Suc-pred } g \text{ ileI1 less-numeral-extra(1) ndropn-ndropn}$
 $\text{ndropn-nlength nidx-gr-first one-eSuc one-enat-def plus-1-eq-Suc})$
have 201: $\text{enat } (\text{nlast } (nmap (\lambda x. x - (\text{nnth } l (Suc 0))) (ndropn (Suc 0) l))) =$
 $\text{nlength } \sigma - \text{enat } (Suc 0) - ((\text{nnth } l (Suc 0)) - 1)$
using 199 200 **by** *presburger*
have 202: $\text{nlength } (nmap (\lambda x. x - (\text{nnth } l (Suc 0))) (ndropn (Suc 0) l)) = \text{nlength } l - (\text{Suc } 0)$
by *force*

have 203: $(\text{ndropn } ((\text{nnth } l \text{ (Suc 0)}) - 1) (\text{ndropn } (\text{Suc } 0) \sigma)) = (\text{ndropn } (\text{nnth } l \text{ (Suc 0)}) \sigma)$
by (metis 3 One-nat-def Suc-pred g ileI1 ndropn-ndropn nidx-gr-first one-eSuc
one-enat-def plus-1-eq-Suc zero-less-one)
have 204: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$
 $\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) (\text{ndropn } (\text{Suc } 0) l)) j =$
 $\text{nnth } l (j + (\text{Suc } 0)) - (\text{nnth } l \text{ (Suc 0)})$
by simp
have 2041: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$
 $\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) (\text{ndropn } (\text{Suc } 0) l)) (\text{Suc } j) =$
 $\text{nnth } l ((\text{Suc } j) + (\text{Suc } 0)) - (\text{nnth } l \text{ (Suc 0)})$
by (metis add.right-neutral add-Suc-right eSuc-enat ileI1 ndropn-nlength ndropn-nnth
nnth-nmap plus-1-eq-Suc)
have 205: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$
 $\text{nnth } l (j + (\text{Suc } 0)) - (\text{nnth } l \text{ (Suc 0)}) < \text{nnth } l ((\text{Suc } j) + (\text{Suc } 0)) - (\text{nnth } l \text{ (Suc 0)})$
using 204 197 **unfolding** nidx-expand **by** auto
(metis 204 One-nat-def add commute eSuc-enat ileI1 plus-1-eq-Suc)
have 206: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$
 $(\text{nsbn } (\text{ndropn } ((\text{nnth } l \text{ (Suc 0)}) - 1) (\text{ndropn } (\text{Suc } 0) \sigma))$
 $(\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) (\text{ndropn } (\text{Suc } 0) l)) j)$
 $(\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) (\text{ndropn } (\text{Suc } 0) l)) (\text{Suc } j))) =$
 $(\text{nsbn } (\text{ndropn } (\text{nnth } l \text{ (Suc 0)})) \sigma)$
 $(\text{nnth } l (j + (\text{Suc } 0)) - (\text{nnth } l \text{ (Suc 0)}))$
 $(\text{nnth } l ((\text{Suc } j) + (\text{Suc } 0)) - (\text{nnth } l \text{ (Suc 0)}))$
using 203 204 2041 **by** simp
have 207: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$
 $(\text{nsbn } (\text{ndropn } (\text{nnth } l \text{ (Suc 0)})) \sigma)$
 $(\text{nnth } l (j + (\text{Suc } 0)) - (\text{nnth } l \text{ (Suc 0)}))$
 $(\text{nnth } l ((\text{Suc } j) + (\text{Suc } 0)) - (\text{nnth } l \text{ (Suc 0)})) =$
 $(\text{nsbn } \sigma (\text{nnth } l \text{ (Suc } j)) (\text{nnth } l \text{ (Suc (Suc } j)))))$
using nsbn-ndropn[of - - σ ($\text{nnth } l \text{ (Suc 0)}$)] **by** simp
(metis (no-types, lifting) 197 202 204 205 add commute add.right-neutral
bot-nat-0.not-eq-extremum diff-add less-nat-zero-code nidx-gr-first order-less-imp-le
plus-1-eq-Suc zero-less-diff)
have 208: $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$
 $(\text{nsbn } \sigma (\text{nnth } l \text{ (Suc } j)) (\text{nnth } l \text{ (Suc (Suc } j))))) \models F$
using 3
by (metis eSuc-enat enat-min g ileI1 one-eSuc plus-1-eSuc(2) zero-enat-def)
have 209: $(\forall i. \text{enat } i < \text{nlength } (\text{nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) (\text{ndropn } (\text{Suc } 0) l)) \longrightarrow$
 $(\text{nsbn } (\text{ndropn } ((\text{nnth } l \text{ (Suc 0)}) - 1) (\text{ndropn } (\text{Suc } 0) \sigma))$
 $(\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) (\text{ndropn } (\text{Suc } 0) l)) i)$
 $(\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) (\text{ndropn } (\text{Suc } 0) l)) (\text{Suc } i)) \models F))$
using 202 206 207 208 **by** presburger
have 210: $(\text{enat } ((\text{nnth } l \text{ (Suc 0)}) - 1) \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge$
 $(\text{ntaken } ((\text{nnth } l \text{ (Suc 0)}) - 1) (\text{ndropn } (\text{Suc } 0) \sigma) \models F\text{-next } ! i) \wedge$
 $(\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) (\text{ndropn } (\text{Suc } 0) l)) 0 = 0 \wedge$
 $\text{nfinite } (\text{nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) (\text{ndropn } (\text{Suc } 0) l)) \wedge$
 $\text{nidx } (\text{nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) (\text{ndropn } (\text{Suc } 0) l)) \wedge$
 $\text{enat } (\text{nlast } (\text{nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) (\text{ndropn } (\text{Suc } 0) l))) =$
 $\text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ((\text{nnth } l \text{ (Suc 0)}) - 1) \wedge \text{nfinite } \sigma \wedge$
 $(\forall i. \text{enat } i < \text{nlength } (\text{nmap } (\lambda x. x - (\text{nnth } l \text{ (Suc 0)})) (\text{ndropn } (\text{Suc } 0) l)) \longrightarrow$

```

      (nsubn (ndropn ((nnth l (Suc 0)) - 1) (ndropn (Suc 0) σ))
        (nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) i)
        (nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) (Suc i)) ⊨ F))))
    using 191 192 193 194 197 201 209 3 by blast
  have 20: ((ndropn 1 σ) ⊨ F-next ! i schop (schoptar F))
    using 210 19 by blast
  have 21: σ ⊨ F-now ! i
    by (metis 15 assms(3) ntaken-nfirst state-qpitl-defs state-qpitl-list-defs)
  have 22: length F-now = length (map (λx. x schop schoptar F ) F-next)
    by (simp add: assms(4))
  have 23: (σ ⊨ big-ior (map2 (λx y. x iand next y) F-now (map (λx. x schop schoptar F ) F-next)))
    <=>
      (∃ i < length F-now. σ ⊨ F-now ! i iand next map (λx. x schop schoptar F ) F-next ! i)
    using big-ior-map2-defs[of F-now (map (λx. x schop (schoptar F)) F-next) σ (λ x y. x iand next
y)]
    22 by auto
  have 24: (σ ⊨ F-now ! i iand next map (λx. x schop schoptar F ) F-next ! i)
    by (metis 15 20 21 7 assms(4) iand-defs ileI1 nth-map one-eSuc semantics-qpitl.simps(4))
  show ?thesis using 15 23 24 ior-defs by blast
qed
show ?thesis
using 4 6 zero-less-iff-neq-zero by blast
qed
have 1000: ?rhs ==> ?lhs
proof -
  assume r: ?rhs
  have 1001: nlength σ = 0 ∨ (σ ⊨ big-ior (map2 (λx y. x iand next y) F-now (map (λx. x schop
schoptar F ) F-next)))
    using r by simp
  have 1002: nlength σ = 0 ==> ?lhs
  proof -
    assume n: nlength σ = 0
    have 1003: nnth (NNil 0) 0 = 0
      by (simp add: nnth-NNil)
    have 1004: nfinite (NNil 0)
      by simp
    have 1005: nidx (NNil 0)
      using Suc-ile-eq nidx-expand by auto
    have 1006: enat (nlast (NNil 0)) = nlength σ
      by (simp add: n zero-enat-def)
    have 1007: nfinite σ
      using n
      by (simp add: nfinite-conv-nlength-enat zero-enat-def)
    have 1008: (∀ i. enat i < nlength (NNil 0) ==> (nsubn σ (nnth (NNil 0) i) (nnth (NNil 0) (Suc i))
⊨ F))
      by auto
    show ?thesis apply simp
      using 1003 1005 1006 1007 1008 by blast
  qed
  have 1009: (σ ⊨ big-ior (map2 (λx y. x iand next y) F-now (map (λx. x schop schoptar F ) F-next)))

```

$\implies ?lhs$

proof –

assume b : $(\sigma \models \text{big-ior} (\text{map2 } (\lambda x y. x \text{ iand next } y) F\text{-now } (\text{map } (\lambda x. x \text{ schop schopstar } F)) F\text{-next})))$

have 1010: $\text{length } F\text{-now} = \text{length } (\text{map } (\lambda x. x \text{ schop schopstar } F)) F\text{-next})$

by (*simp add: assms(4)*)

have 1011: $(\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now} ! i \text{ iand next map } (\lambda x. x \text{ schop schopstar } F)) F\text{-next} ! i)$

using 1010 $b \text{ big-ior-map2-defs[of } F\text{-now } (\text{map } (\lambda x. x \text{ schop } (\text{s chopstar } F)) F\text{-next}) \sigma (\lambda x y. x \text{ iand next } y)]$

by *blast*

obtain i **where** 1012: $i < \text{length } F\text{-now} \wedge (\sigma \models F\text{-now} ! i \text{ iand next map } (\lambda x. x \text{ schop schopstar } F)) F\text{-next} ! i)$

using 1011 **by** *blast*

have 1013: $(\sigma \models F\text{-now} ! i)$

using 1012 **by** *auto*

have 1014: $((\text{ndropn } 1 \sigma) \models F\text{-next} ! i \text{ schop } (\text{s chopstar } F))$

using 1012 *assms(4)* **by** *fastforce*

have 1015: $((\text{ndropn } 1 \sigma) \models F\text{-next} ! i \text{ schop } (\text{s chopstar } F)) \longleftrightarrow$

$(\exists ia. \text{enat } ia \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge$

$(\text{ntaken } ia (\text{ndropn } (\text{Suc } 0) \sigma) \models F\text{-next} ! i) \wedge$

$(\exists l. \text{nnth } l \ 0 = 0 \wedge$

$\text{nfinite } l \wedge$

$\text{nidx } l \wedge$

$\text{enat } (\text{nlast } l) = \text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia \wedge \text{nfinite } \sigma \wedge$

$(\forall i. \text{enat } i < \text{nlength } l \longrightarrow (\text{nsubn } (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0) \sigma)) (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))$

$\models F)))$

by *simp*

obtain ia **where** 1016: $\text{enat } ia \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge$

$(\text{ntaken } ia (\text{ndropn } (\text{Suc } 0) \sigma) \models F\text{-next} ! i) \wedge$

$(\exists l. \text{nnth } l \ 0 = 0 \wedge$

$\text{nfinite } l \wedge$

$\text{nidx } l \wedge$

$\text{enat } (\text{nlast } l) = \text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia \wedge \text{nfinite } \sigma \wedge$

$(\forall i. \text{enat } i < \text{nlength } l \longrightarrow (\text{nsubn } (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0) \sigma)) (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))$

$\models F)))$

using 1014 1015 **by** *blast*

obtain l **where** 1017: $\text{nnth } l \ 0 = 0 \wedge$

$\text{nfinite } l \wedge$

$\text{nidx } l \wedge$

$\text{enat } (\text{nlast } l) = \text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia \wedge \text{nfinite } \sigma \wedge$

$(\forall i. \text{enat } i < \text{nlength } l \longrightarrow (\text{nsubn } (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0) \sigma)) (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))$

$\models F))$

using 1016 **by** *blast*

have 1018: $0 < \text{nlength } \sigma$

using 1012 **by** *force*

have 10183: $\text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia = \text{nlength } \sigma - \text{Suc } ia$

by (*metis One-nat-def ndropn-ndropn ndropn-nlength plus-1-eq-Suc*)

have 10184: $\text{Suc } ia \leq \text{nlength } \sigma$

by (*metis 1016 1018 One-nat-def eSuc-enat enat-min-eq ileI1 one-eSuc one-enat-def plus-1-eSuc(2)*)

have 10185: $\text{nlength } \sigma - \text{Suc } ia + \text{Suc } ia = \text{nlength } \sigma$

by (*metis 10184 add commute enat.simps(3) enat-add-sub-same le-iff-add*)

```

have 1019: (nlength  $\sigma$  - enat (Suc 0) - enat ia) + Suc ia = nlength  $\sigma$ 
  by (simp add: 10183 10185)
have 1020: nlast (nmap ( $\lambda x. x + \text{Suc ia}$ ) l) = nlength  $\sigma$ 
  by (metis 1017 1019 nlast-nmap plus-enat-simps(1))
have 1021: nlast (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) = nlength  $\sigma$ 
  using 1020 by force
have 1022: nnth (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) 0 = 0
  by simp
have 1023: 0 < nnth (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) 1
  using zero-enat-def by force
have 1024: nfinite (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l))
  by (simp add: 1017)
have 1025: nidx (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l))
  using 1017 nidx-Ncons-shift by blast
have 1028: (nsbn  $\sigma$  (nnth (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) 0) (nnth (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) (Suc 0))) =
  (nsbn  $\sigma$  0 (Suc ia))
  using 1017 zero-enat-def by fastforce
have 1029: ((nsbn  $\sigma$  0 (Suc ia))  $\models$  (big-ior (map2 ( $\lambda x y. x \text{ iand } (\text{next } y)$ ) F-now F-next)))  $\longleftrightarrow$ 
  ( $\exists i < \text{length } F\text{-now. nsbn } \sigma \text{ } 0 \text{ (Suc ia)} \models F\text{-now } ! i \text{ iand next } F\text{-next } ! i$ )
  using big-ior-map2-defs[of F-now F-next (nsbn  $\sigma$  0 (Suc ia)) ( $\lambda x y. x \text{ iand } (\text{next } y)$ )]
  assms by blast
have 1030: nsbn  $\sigma$  0 (Suc ia)  $\models F\text{-now } ! i$ 
  using 1016
  by (metis 1012 1013 assms(3) ndropn-0 nsbn-def1 ntaken-nfirst state-qpitl-defs state-qpitl-list-defs)
have 1031: nsbn  $\sigma$  0 (Suc ia)  $\models \text{next } F\text{-next } ! i$ 
  using 1016 ndropn-nsbn[of Suc ia  $\sigma$  Suc 0 0]
  by (metis 1019 One-nat-def add-diff-cancel-left' enat-le-plus-same(2) ileI1 le-add1
    nsbn-def1 nsbn-nlength-gr-one one-eSuc plus-1-eq-Suc semantics-qpitl.simps(4) zero-less-Suc)
have 1034: (nsbn  $\sigma$  (nnth (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) 0) (nnth (NCons 0 (nmap ( $\lambda x. x + \text{Suc ia}$ ) l)) (Suc 0)))  $\models$ 
  ((F-e iand empty) ior (big-ior (map2 ( $\lambda x y. x \text{ iand } (\text{next } y)$ ) F-now F-next)))
  using 1012 1028 1029 1030 1031 by auto
have 1035:  $\bigwedge j. j < \text{nlength } l \implies$ 
  (nnth ( (nmap ( $\lambda x. x + \text{Suc ia}$ ) l) ) j) = nnth l j + Suc ia
  by simp
have 1036:  $\bigwedge j. j < \text{nlength } l \implies$ 
  (nnth ( (nmap ( $\lambda x. x + \text{Suc ia}$ ) l) ) (Suc j)) = nnth l (Suc j) + Suc ia
  using Suc-ile-eq by force
have 1037:  $\bigwedge j. j < \text{nlength } l \implies$ 
  nnth l j + Suc ia < nnth l (Suc j) + Suc ia
  by (metis 1017 add-strict-right-mono eSuc-enat ileI1 nidx-expand)
have 1038:  $\bigwedge j. j < \text{nlength } l \implies$ 
  enat ( nnth l (Suc j) + Suc ia)  $\leq$  nlength  $\sigma$ 
  using 1017 nidx-expand by simp
  (metis 1020 1025 1036 add-Suc-right eSuc-enat enat-ord-simps(1) ileI1 nfinite-nmap
    nidx-LCons-1 nidx-all-le-nlast nlength-nmap)
have 1039:  $\bigwedge j. j < \text{nlength } l \implies$ 
  Suc 0 + (nnth l j + Suc ia)  $\leq$  nnth l (Suc j) + Suc ia
  using 1037 by fastforce

```

have 1040: $\bigwedge j. j < \text{nlength } l \implies$
 $\text{ndropn } (\text{Suc } 0) (\text{nsubn } \sigma (\text{nnth } l j + \text{Suc } ia) (\text{nnth } l (\text{Suc } j) + \text{Suc } ia)) =$
 $\text{nsubn } \sigma (\text{Suc } 0 + (\text{nnth } l j + \text{Suc } ia)) (\text{nnth } l (\text{Suc } j) + \text{Suc } ia)$
using *ndropn-nsubn[of - σ Suc 0]* **using** 1038 1039 **by** *presburger*
have 1041: $\bigwedge j. j < \text{nlength } l \implies$
 $(\text{nsubn } (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0) \sigma)) (\text{nnth } l j) (\text{nnth } l (\text{Suc } j))) =$
 $\text{nsubn } \sigma ((\text{nnth } l j) + \text{Suc } ia) ((\text{nnth } l (\text{Suc } j)) + \text{Suc } ia)$
using *nsubn-ndropn[of - - σ Suc ia]*
by (*metis* 1037 *One-nat-def add-less-cancel-right ndropn-ndropn plus-1-eq-Suc*)
have 1280: $(\text{nsubn } \sigma (\text{nnth } (\text{NCons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) 0) (\text{nnth } (\text{NCons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) (\text{Suc } 0))) \models F)$
using 1034 *assms(1)* **by** *simp*
have 1281: $(\forall i. \text{enat } i < \text{nlength } (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) \longrightarrow$
 $(\text{nsubn } \sigma (\text{nnth } (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) i) (\text{nnth } (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) (\text{Suc } i)) \models$
 $F))$
using 1017 1036 1041 **by** *auto*
have 1290: $(\forall i. \text{enat } i < \text{nlength } (\text{NCons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) \longrightarrow$
 $(\text{nsubn } \sigma (\text{nnth } (\text{NCons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) i) (\text{nnth } (\text{NCons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l))$
 $(\text{Suc } i)) \models F)) \longleftrightarrow$
 $(\text{nsubn } \sigma (\text{nnth } (\text{NCons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) 0) (\text{nnth } (\text{NCons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) (\text{Suc } 0))) \models F) \wedge$
 $(\forall i. \text{enat } i < \text{nlength } (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) \longrightarrow$
 $(\text{nsubn } \sigma (\text{nnth } (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) i) (\text{nnth } (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) (\text{Suc } i)) \models$
 $F))$
using *forall-ncons-split[of (nmap (λx. x + Suc ia) l)*
 $\lambda x y. (\text{nsubn } \sigma x y \models F)]$ **by** (*simp add: 1017*)
have 1300: $(\forall i. \text{enat } i < \text{nlength } (\text{NCons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) \longrightarrow$
 $(\text{nsubn } \sigma (\text{nnth } (\text{NCons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) i) (\text{nnth } (\text{NCons } 0 (\text{nmap } (\lambda x. x + \text{Suc } ia) l))$
 $(\text{Suc } i)) \models F))$
using 1280 1281 1290 **by** *blast*
have 1400: $\exists l. \text{nnth } l 0 = 0 \wedge \text{nfinite } l \wedge \text{nidx } l \wedge \text{enat } (\text{nlast } l) = \text{nlength } \sigma \wedge \text{nfinite } \sigma \wedge$
 $(\forall i. \text{enat } i < \text{nlength } l \longrightarrow (\text{nsubn } \sigma (\text{nnth } l i) (\text{nnth } l (\text{Suc } i)) \models F))$
by (*meson* 1017 1021 1022 1025 1300 *nfinite-NConsI nfinite-nmap*)
show ?thesis **using** 1400 **by** *simp*
qed
show ?thesis
using 1001 1002 1009 **by** *blast*
qed
show ?thesis
using 1000 2 **by** *blast*
qed

lemma *GNF-omega-step:*

assumes $\vdash F \text{ ieqv } ((F\text{-e iand empty }) \text{ ior } (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) F\text{-now } F\text{-next})))$
 $\text{state-qpitl } F\text{-e}$
 $\text{state-qpitl-list } F\text{-now}$
 $\text{length } F\text{-now} = \text{length } F\text{-next}$
shows $(\sigma \models \text{omega } F) \longleftrightarrow$

```

    (σ ⊨ (ifalse iand empty) ior
      (big-ior (map2 (λ x y. x iand next y) F-now (map (λ x. x schop (omega F)) F-next))))
    (is ?lhs ←→ ?rhs)
  proof -
  have 1: (σ ⊨ omega F) ←→
    (∃ l. ¬nfinite l ∧ nnth l 0 = 0 ∧ nidx l ∧
      (∀ i. (nnth l i) ≤ nlength σ) ∧
      (∀ i. (nsubn σ (nnth l i) (nnth l (Suc i)) ⊨ F)))
  by simp
  have 2: ?lhs ⇒ ?rhs
  proof -
  assume l: ?lhs
  obtain l where 3: ¬nfinite l ∧ nnth l 0 = 0 ∧ nidx l ∧
    (∀ i. (nnth l i) ≤ nlength σ) ∧
    (∀ i. (nsubn σ (nnth l i) (nnth l (Suc i)) ⊨ F))
  using l 1 by blast
  have 6: ¬nfinite σ
  using 3 infinite-nidx-imp-infinite-interval by blast
  have 7: 0 < nlength σ
  using 6 enat-defs(1) nfinite-conv-nlength-enat by fastforce
  have 8: (nsubn σ (nnth l 0) (nnth l (Suc 0)) ⊨ F)
  using 3 by blast
  have 9: (nsubn σ (nnth l 0) (nnth l (Suc 0))) = (ntaken (nnth l (Suc 0)) σ)
  using 3 by (simp add: nsubn-def1)
  have 10: (ntaken (nnth l (Suc 0)) σ) ⊨ F
  using 9 8 by auto
  have 11: (ntaken (nnth l (Suc 0)) σ) ⊨ (( F-e iand empty ) ior (big-ior (map2 (λ x y. x iand (next y))
    F-now F-next)))
  using 10 assms(1) itl-ieq by blast
  have 12: (ntaken (nnth l (Suc 0)) σ) ⊨ (big-ior (map2 (λ x y. x iand (next y)) F-now F-next))
  by (metis 11 3 9 One-nat-def empty-defs enat-defs(2) iand-defs iless-eSuc0 ior-defs
    less-numeral-extra(1) less-numeral-extra(3) nidx-gr-first nlength-eq-enat-nfiniteD
    not-less nsubn-nlength-gr-one one-eSuc zero-enat-def)
  have 13: 0 < nlength (ntaken (nnth l (Suc 0)) σ)
  by (metis 3 9 One-nat-def iless-eSuc0 less-numeral-extra(1) linorder-not-less
    nidx-gr-first nlength-eq-enat-nfiniteD nsubn-nlength-gr-one one-eSuc one-enat-def zero-enat-def)
  have 14: (nlength (ntaken (nnth l (Suc 0)) σ) > 0 ∧
    (∃ i < length F-now. ((NNil (nfirst (ntaken (nnth l (Suc 0)) σ))) ⊨ F-now !i) ∧
      ((ndropn 1 (ntaken (nnth l (Suc 0)) σ)) ⊨ F-next !i)) )
  using GNF-defs[of F F-e F-now F-next (ntaken (nnth l (Suc 0)) σ)] 12 13
  by (metis 10 assms(1) assms(2) assms(3) assms(4) order-less-irrefl)
  obtain i where 15: i < length F-now ∧ ((NNil (nfirst (ntaken (nnth l (Suc 0)) σ))) ⊨ F-now !i) ∧
    ((ndropn 1 (ntaken (nnth l (Suc 0)) σ)) ⊨ F-next !i)
  using 14 by blast
  have 19: ((ndropn 1 σ) ⊨ F-next !i schop (omega F)) ←→
    (∃ ia. enat ia ≤ nlength σ - enat (Suc 0) ∧
      (ntaken ia (ndropn (Suc 0) σ) ⊨ F-next !i) ∧
      (∃ l. ¬ nfinite l ∧
        nnth l 0 = 0 ∧ nidx l ∧ (∀ i. enat (nnth l i) ≤ nlength σ - enat (Suc 0) - enat ia) ∧
        (∀ i. nsubn (ndropn ia (ndropn (Suc 0) σ)) (nnth l i) (nnth l (Suc i)) ⊨ F)))

```

```

  by simp
have 191: enat ((nnth l (Suc 0)) - 1) ≤ nlength σ - enat (Suc 0)
  by (metis 3 One-nat-def enat-minus-mono1 idiff-enat-enat)
have 192: ntaken ((nnth l (Suc 0)) - 1) (ndropn (Suc 0) σ) = F-next ! i
  using 13 15 191 3 7
  by (simp add: enat-0-iff(1) ntaken-ndropn-swap)
have 193: nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) 0 = 0
  using enat-defs(1) by auto
have 194: ¬nfinite (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l))
  by (simp add: 3)
have 195: nidx (ndropn 1 l)
  by (metis 3 One-nat-def Suc-ile-eq dual-order.order-iff-strict enat-defs(1)
    nfinite-conv-nlength-enat nidx-ndropn zero-le)
have 196: nnth (ndropn 1 l) 0 = (nnth l (Suc 0))
  by simp
have 197: nidx (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l))
  using nidx-shiftm[of (ndropn 1 l) (nnth l (Suc 0)) ]
  by (metis 195 196 One-nat-def)
have 198: (∀ i. enat (nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) i) ≤ nlength σ - enat
  (Suc 0) - enat ((nnth l (Suc 0)) - 1))
  by (metis 6 ndropn-nlength nfinite-ndropn nfinite-ntaken nle-le ntaken-all)
have 202: nlength (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) = nlength l - (Suc 0)
  by force
have 203: (ndropn ((nnth l (Suc 0)) - 1) (ndropn (Suc 0) σ)) = (ndropn (nnth l (Suc 0)) σ)
  by (metis 3 One-nat-def Suc-pred ileI1 less-numeral-extra(1) ndropn-ndropn nidx-gr-first
    nlength-eq-enat-nfiniteD one-eSuc one-enat-def plus-1-eq-Suc zero-enat-def zero-less-iff-neq-zero)
have 204: ∧j. j < nlength l - (Suc 0) ⇒
  nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) j =
  nnth l (j + (Suc 0)) - (nnth l (Suc 0))
  by simp
have 2041: ∧j. j < nlength l - (Suc 0) ⇒
  nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) (Suc j) =
  nnth l ((Suc j) + (Suc 0)) - (nnth l (Suc 0))
  by (metis add.right-neutral add-Suc-right eSuc-enat ileI1 ndropn-nlength ndropn-nnth nnth-nmap
    plus-1-eq-Suc)
have 205: ∧j. j < nlength l - (Suc 0) ⇒
  nnth l (j + (Suc 0)) - (nnth l (Suc 0)) < nnth l ((Suc j) + (Suc 0)) - (nnth l (Suc 0))
  using 204 197 unfolding nidx-expand apply auto
  by (metis 204 One-nat-def add commute eSuc-enat ileI1 plus-1-eq-Suc)
have 206: ∧j. j < nlength l - (Suc 0) ⇒
  (nsubn (ndropn ((nnth l (Suc 0)) - 1) (ndropn (Suc 0) σ))
    (nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) j)
    (nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) (Suc j))) =
  (nsubn (ndropn (nnth l (Suc 0)) σ)
    (nnth l (j + (Suc 0)) - (nnth l (Suc 0)))
    (nnth l ((Suc j) + (Suc 0)) - (nnth l (Suc 0))))
  using 203 204 2041 by simp
have 207: ∧j. j < nlength l - (Suc 0) ⇒
  (nsubn (ndropn (nnth l (Suc 0)) σ)
    (nnth l (j + (Suc 0)) - (nnth l (Suc 0)))

```

```

      (nnth l ((Suc j)+ (Suc 0)) - (nnth l (Suc 0))) =
      (nsbn σ (nnth l (Suc j)) (nnth l (Suc (Suc j))))
using nsbn-ndropn[of - σ (nnth l (Suc 0))] by simp
      (metis (no-types, lifting) 197 202 204 205 add.commute add.right-neutral
      bot-nat-0.not-eq-extremum diff-add less-nat-zero-code nidx-gr-first
      order-less-imp-le plus-1-eq-Suc zero-less-diff)
have 208:  $\bigwedge j. j < \text{nlength } l - (\text{Suc } 0) \implies$ 
      (nsbn σ (nnth l (Suc j)) (nnth l (Suc (Suc j))))  $\models F$ 
using 3 by blast
have 209:  $(\forall i. \text{enat } i < \text{nlength } (\text{nmap } (\lambda x. x - (\text{nnth } l (\text{Suc } 0))) (\text{ndropn } (\text{Suc } 0) l)) \longrightarrow$ 
      (nsbn (ndropn ((nnth l (Suc 0))-1) (ndropn (Suc 0) σ))
      (nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) i)
      (nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) (Suc i))  $\models F$ ))
using 202 206 207 208 by presburger
have 2090:  $\text{nlength } (\text{nmap } (\lambda x. x - (\text{nnth } l (\text{Suc } 0))) (\text{ndropn } (\text{Suc } 0) l)) = \infty$ 
by (metis 194 enat2-cases nlength-eq-enat-nfiniteD)
have 2091:  $(\forall i.$ 
      (nsbn (ndropn ((nnth l (Suc 0))-1) (ndropn (Suc 0) σ))
      (nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) i)
      (nnth (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l)) (Suc i))  $\models F$ ))
using 2090 209
using enat-ord-code(4) by presburger
let ?ia = ((nnth l (Suc 0))-1)
let ?l = (nmap (λx. x - (nnth l (Suc 0))) (ndropn (Suc 0) l))
have 210:  $(\text{enat } ?ia \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge$ 
      (ntaken ?ia (ndropn (Suc 0) σ)  $\models F\text{-next } ! i) \wedge$ 
       $(\neg \text{nfinite } ?l \wedge$ 
       $\text{nnth } ?l 0 = 0 \wedge$ 
       $\text{nidx } ?l \wedge$ 
       $(\forall i. \text{enat } (\text{nnth } ?l i) \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ?ia) \wedge$ 
       $(\forall i.$ 
      (nsbn (ndropn ?ia (ndropn (Suc 0) σ))
      (nnth ?l i)
      (nnth ?l (Suc i))  $\models F$ ))
      )
      )
using 191 192 193 194 197 198 2091 by blast
have 20:  $((\text{ndropn } 1 \sigma) \models F\text{-next } ! i \text{ schop } (\text{omega } F))$ 
using 210 19 by blast
have 21:  $\sigma \models F\text{-now } ! i$ 
by (metis 15 assms(3) ntaken-nfirst state-qpitl-defs state-qpitl-list-defs)
have 22:  $\text{length } F\text{-now} = \text{length } (\text{map } (\lambda x. x \text{ schop } \text{omega } F) F\text{-next})$ 
by (simp add: assms(4))
have 23:  $(\sigma \models \text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand next } y) F\text{-now } (\text{map } (\lambda x. x \text{ schop } \text{omega } F) F\text{-next}))) \longleftrightarrow$ 
       $(\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now } ! i \text{ iand next map } (\lambda x. x \text{ schop } \text{omega } F) F\text{-next } ! i)$ 
using big-ior-map2-defs[of F-now (map (λx. x schop (omega F)) F-next) σ (λ x y. x iand next y)]

22 by auto
have 24:  $(\sigma \models F\text{-now } ! i \text{ iand next map } (\lambda x. x \text{ schop } \text{omega } F) F\text{-next } ! i)$ 
by (metis 15 20 21 7 assms(4) iand-defs ileI1 nth-map one-eSuc semantics-qpitl.simps(4))

```



```

  show ?thesis using 15 23 24 ior-defs by blast
qed
have 1000: ?rhs  $\implies$  ?lhs
proof -
  assume r: ?rhs
  have 1001:  $(\sigma \models \text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand next } y) F\text{-now } (\text{map } (\lambda x. x \text{ schop omega } F) ) F\text{-next})))$ 
    using r by simp
  have 1010:  $\text{length } F\text{-now} = \text{length } (\text{map } (\lambda x. x \text{ schop omega } F) ) F\text{-next}$ 
    by (simp add: assms(4))
  have 1011:  $(\exists i < \text{length } F\text{-now}. \sigma \models F\text{-now } ! i \text{ iand next map } (\lambda x. x \text{ schop omega } F) ) F\text{-next } ! i)$ 
    using 1001 1010 big-ior-map2-defs[of F-now (map (lambda x. x schop (omega F)) F-next) sigma (lambda x y. x
iand next y)]
    by blast
  obtain i where 1012:  $i < \text{length } F\text{-now} \wedge (\sigma \models F\text{-now } ! i \text{ iand next map } (\lambda x. x \text{ schop omega } F) ) F\text{-next}
! i)$ 
    using 1011 by blast
  have 1013:  $(\sigma \models F\text{-now } ! i)$ 
    using 1012 by auto
  have 1014:  $((\text{ndropn } 1 \sigma) \models F\text{-next } ! i \text{ schop } (\text{omega } F))$ 
    using 1012 assms(4) by fastforce
  have 1015:  $((\text{ndropn } 1 \sigma) \models F\text{-next } ! i \text{ schop } (\text{omega } F)) \longleftrightarrow$ 
 $(\exists ia. \text{enat } ia \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge$ 
 $(\text{ntaken } ia (\text{ndropn } (\text{Suc } 0) \sigma) \models F\text{-next } ! i) \wedge$ 
 $(\exists l. \neg \text{nfinite } l \wedge$ 
 $\text{nnth } l 0 = 0 \wedge \text{nidx } l \wedge$ 
 $(\forall i. \text{enat } (\text{nnth } l i) \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia) \wedge$ 
 $(\forall i. \text{nsubn } (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0) \sigma)) (\text{nnth } l i) (\text{nnth } l (\text{Suc } i)) \models F)))$ 
    by simp
  obtain ia where 1016:  $\text{enat } ia \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) \wedge$ 
 $(\text{ntaken } ia (\text{ndropn } (\text{Suc } 0) \sigma) \models F\text{-next } ! i) \wedge$ 
 $(\exists l. \neg \text{nfinite } l \wedge$ 
 $\text{nnth } l 0 = 0 \wedge \text{nidx } l \wedge$ 
 $(\forall i. \text{enat } (\text{nnth } l i) \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia) \wedge$ 
 $(\forall i. \text{nsubn } (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0) \sigma)) (\text{nnth } l i) (\text{nnth } l (\text{Suc } i)) \models F))$ 
    using 1014 1015 by blast
  obtain l where 1017:  $\neg \text{nfinite } l \wedge$ 
 $\text{nnth } l 0 = 0 \wedge \text{nidx } l \wedge$ 
 $(\forall i. \text{enat } (\text{nnth } l i) \leq \text{nlength } \sigma - \text{enat } (\text{Suc } 0) - \text{enat } ia) \wedge$ 
 $(\forall i. \text{nsubn } (\text{ndropn } ia (\text{ndropn } (\text{Suc } 0) \sigma)) (\text{nnth } l i) (\text{nnth } l (\text{Suc } i)) \models F)$ 
    using 1016 by blast
  have 1018:  $\neg \text{nfinite } \sigma$ 
    by (metis 1016 infinite-nidx-imp-infinite-interval ndropn-nlength nfinite-ndropn-a)
  have 1022:  $\text{nnth } (NCons 0 (nmap (\lambda x. x + \text{Suc } ia) l)) 0 = 0$ 
    by simp
  have 1023:  $0 < \text{nnth } (NCons 0 (nmap (\lambda x. x + \text{Suc } ia) l)) 1$ 
    using zero-enat-def by force
  have 1024:  $\neg \text{nfinite } (NCons 0 (nmap (\lambda x. x + \text{Suc } ia) l))$ 
    by (simp add: 1017)
  have 1025:  $\text{nidx } (NCons 0 (nmap (\lambda x. x + \text{Suc } ia) l))$ 
    using 1017 nidx-Ncons-shift-alt by blast

```

have 1026: $(\forall i. (nnth (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))\ i) \leq nlength\ \sigma)$
by (*meson* 1018 *enat-ile nle-le nlength-eq-enat-nfiniteD*)
have 1028: $(nsubn\ \sigma\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))\ 0)\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))\ (Suc\ 0))) =$
 $(nsubn\ \sigma\ 0\ (Suc\ ia))$
using 1017 *zero-enat-def* **by** *fastforce*
have 1029: $((nsubn\ \sigma\ 0\ (Suc\ ia)) \models (big\text{-}ior\ (map2\ (\lambda\ x\ y. x\ iand\ (next\ y))\ F\text{-}now\ F\text{-}next))) \longleftrightarrow$
 $(\exists i < length\ F\text{-}now. nsubn\ \sigma\ 0\ (Suc\ ia) \models F\text{-}now\ !\ i\ iand\ next\ F\text{-}next\ !\ i)$
using *big-ior-map2-defs*[*of* *F-now F-next* (*nsubn* σ 0 (*Suc ia*)) ($\lambda\ x\ y. x\ iand\ (next\ y)$)]
assms **by** *blast*
have 1030: $nsubn\ \sigma\ 0\ (Suc\ ia) \models F\text{-}now\ !\ i$
using 1016
by (*metis* 1012 1013 *assms*(3) *ndropn-0 nsubn-def1 ntaken-nfirst state-qpittl-defs state-qpittl-list-defs*)
have 1031: $nsubn\ \sigma\ 0\ (Suc\ ia) \models next\ F\text{-}next\ !\ i$
using 1016 *ndropn-nsubn*[*of* *Suc ia* σ *Suc 0*]
by (*metis* 1022 1023 1026 1028 *One-nat-def add commute diff-zero enat-min-eq enat-ord-simps*(1) *ileI1 leD ndropn-0 nsubn-def1 nsubn-nlength-gr-one ntaken-ndropn-swap one-eSuc one-enat-def plus-1-eq-Suc semantics-qpittl.simps*(4) *zero-less-iff-neq-zero*)
have 1034: $(nsubn\ \sigma\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))\ 0)\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))\ (Suc\ 0))) \models$
 $((F\text{-}e\ iand\ empty)\ ior\ (big\text{-}ior\ (map2\ (\lambda\ x\ y. x\ iand\ (next\ y))\ F\text{-}now\ F\text{-}next)))$
using 1012 1028 1029 1030 1031 **by** *auto*
have 1035: $\bigwedge j. j < nlength\ l \implies$
 $(nnth\ ((nmap\ (\lambda x. x + Suc\ ia)\ l))\ j) = nnth\ l\ j + Suc\ ia$
by *simp*
have 1036: $\bigwedge j. j < nlength\ l \implies$
 $(nnth\ ((nmap\ (\lambda x. x + Suc\ ia)\ l))\ (Suc\ j)) = nnth\ l\ (Suc\ j) + Suc\ ia$
using *Suc-ile-eq* **by** *force*
have 1037: $\bigwedge j. j < nlength\ l \implies$
 $nnth\ l\ j + Suc\ ia < nnth\ l\ (Suc\ j) + Suc\ ia$
by (*metis* 1017 *add-strict-right-mono eSuc-enat ileI1 nidx-expand*)
have 1038: $\bigwedge j. j < nlength\ l \implies$
 $enat\ (nnth\ l\ (Suc\ j) + Suc\ ia) \leq nlength\ \sigma$
using 1017 *nidx-expand* **by** *simp*
 $(meson\ 1018\ enat-ile\ nle-le\ nlength-eq-enat-nfiniteD)$
have 1039: $\bigwedge j. j < nlength\ l \implies$
 $Suc\ 0 + (nnth\ l\ j + Suc\ ia) \leq nnth\ l\ (Suc\ j) + Suc\ ia$
using 1037 **by** *fastforce*
have 1040: $\bigwedge j. j < nlength\ l \implies$
 $ndropn\ (Suc\ 0)\ (nsubn\ \sigma\ (nnth\ l\ j + Suc\ ia)\ (nnth\ l\ (Suc\ j) + Suc\ ia)) =$
 $nsubn\ \sigma\ (Suc\ 0 + (nnth\ l\ j + Suc\ ia))\ (nnth\ l\ (Suc\ j) + Suc\ ia)$
using *ndropn-nsubn*[*of* - σ *Suc 0*] **using** 1038 1039 **by** *presburger*
have 1041: $\bigwedge j. j < nlength\ l \implies$
 $(nsubn\ (ndropn\ ia\ (ndropn\ (Suc\ 0)\ \sigma))\ (nnth\ l\ j)\ (nnth\ l\ (Suc\ j))) =$
 $nsubn\ \sigma\ ((nnth\ l\ j) + Suc\ ia)\ ((nnth\ l\ (Suc\ j)) + Suc\ ia)$
using *nsubn-ndropn*[*of* - - σ *Suc ia*]
by (*metis* 1037 *One-nat-def add-less-cancel-right ndropn-ndropn plus-1-eq-Suc*)
have 1280: $(nsubn\ \sigma\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))\ 0)\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x. x + Suc\ ia)\ l))\ (Suc\ 0))) \models F$
using 1034 *assms*(1) **by** *simp*

have 1281: $(\forall i. \text{enat } i < \text{nlength } (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) \longrightarrow$
 $(\text{nsubn } \sigma (\text{nnth } (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) i) (\text{nnth } (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) (\text{Suc } i)) \models$
 $F))$
using 1017 1036 1041
by (metis 1035 nlength-nmap)
have 1290: $(\forall i. \text{enat } i < \text{nlength } (NCons\ 0\ (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) \longrightarrow$
 $(\text{nsubn } \sigma (\text{nnth } (NCons\ 0\ (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) i) (\text{nnth } (NCons\ 0\ (\text{nmap } (\lambda x. x + \text{Suc } ia) l))$
 $(\text{Suc } i)) \models F)) \longleftrightarrow$
 $(\text{nsubn } \sigma (\text{nnth } (NCons\ 0\ (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) 0) (\text{nnth } (NCons\ 0\ (\text{nmap } (\lambda x. x + \text{Suc } ia) l))$
 $(\text{Suc } 0)) \models F) \wedge$
 $(\forall i. \text{enat } i < \text{nlength } (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) \longrightarrow$
 $(\text{nsubn } \sigma (\text{nnth } (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) i) (\text{nnth } (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) (\text{Suc } i)) \models$
 $F))$
using forall-ncons-split-alt[of $(\text{nmap } (\lambda x. x + \text{Suc } ia) l)$
 $\lambda x y. (\text{nsubn } \sigma\ x\ y \models F)$]
by (simp add: 1017)
have 1300: $(\forall i. \text{enat } i < \text{nlength } (NCons\ 0\ (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) \longrightarrow$
 $(\text{nsubn } \sigma (\text{nnth } (NCons\ 0\ (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) i) (\text{nnth } (NCons\ 0\ (\text{nmap } (\lambda x. x + \text{Suc } ia) l))$
 $(\text{Suc } i)) \models F))$
using 1280 1281 1290 **by** blast
have 1340: $\text{nlength } (NCons\ 0\ (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) = \infty$
by (metis 1024 enat2-cases nlength-eq-enat-nfiniteD)
have 1350: $(\forall i.$
 $(\text{nsubn } \sigma (\text{nnth } (NCons\ 0\ (\text{nmap } (\lambda x. x + \text{Suc } ia) l)) i) (\text{nnth } (NCons\ 0\ (\text{nmap } (\lambda x. x + \text{Suc } ia) l))$
 $(\text{Suc } i)) \models F))$
using 1300 1340 enat-ord-code(4) **by** presburger
have 1400: $(\exists (l:: \text{nat nellist}).$
 $\neg \text{nfinite } l \wedge (\text{nnth } l\ 0) = 0 \wedge \text{nidx } l \wedge$
 $(\forall i. (\text{nnth } l\ i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. ((\text{nsubn } \sigma (\text{nnth } l\ i) (\text{nnth } l\ (\text{Suc } i))) \models F))$
 $)$
by (meson 1017 1022 1025 1026 1350 nfinite-NCons nfinite-nmap)
show ?thesis **using** 1400 **by** simp
qed
show ?thesis
using 1000 2 **by** blast
qed

lemma big-ior-dist-exists:

$\vdash (\text{Ex } n. \text{big-ior } L) \text{ieqv } \text{big-ior } (\text{map } (\lambda x. (\text{Ex } n. x)) L)$
apply (auto simp add: big-ior-defs)
apply fastforce
by blast

lemma more-defs [simp]:

$(\sigma \models \text{more}) = (\text{nlength } \sigma > 0)$
by (auto simp add: empty-d-def more-d-def ileI1 one-eSuc)

lemma *state-qpitl-exists-dist-iand*:

assumes *state-qpitl w*

shows $\vdash ((Ex\ n.\ w\ \text{iand}\ next\ f)\ \text{ieqv}\ (Ex\ n.\ w)\ \text{iand}\ next\ (Ex\ n.\ f))$

proof –

have 1: $\bigwedge \sigma.$

$(\sigma \models (Ex\ n.\ w\ \text{iand}\ next\ f)) \implies (\sigma \models (Ex\ n.\ w)\ \text{iand}\ next\ (Ex\ n.\ f))$

using *assms* **by** (*auto simp add: nlength-upd*)

(*metis One-nat-def ndropn-nlength ndropn-upd one-enat-def*)

have 2: $\bigwedge \sigma.\ (\sigma \models (Ex\ n.\ w)\ \text{iand}\ next\ (Ex\ n.\ f)) \implies (\sigma \models (Ex\ n.\ w\ \text{iand}\ next\ f))$

proof –

fix σ

assume *a*: $(\sigma \models (Ex\ n.\ w)\ \text{iand}\ next\ (Ex\ n.\ f))$

show $(\sigma \models (Ex\ n.\ w\ \text{iand}\ next\ f))$

proof –

have 3: $(\sigma \models (Ex\ n.\ w))$

using *a* **by** *auto*

obtain *l* **where** 4: $nlength\ l = nlength\ \sigma \wedge (upd\ \sigma\ n\ l \models w)$

using 3 **by** *auto*

have 5: $nlength\ \sigma > 0 \wedge (ndropn\ 1\ \sigma \models (Ex\ n.\ f))$

using *a* **by** *auto*

obtain *l1* **where** 6: $nlength\ l1 = nlength(ndropn\ 1\ \sigma) \wedge (upd\ (ndropn\ 1\ \sigma)\ n\ l1 \models f)$

using 5 **by** *auto*

have 7: $(NNil\ (nfirst\ (upd\ \sigma\ n\ l)) \models w)$

using *assms* 4 *state-qpitl-defs* **by** *blast*

have 8: $(NNil\ (nfirst\ (upd\ \sigma\ n\ (nappend\ (NNil\ (nfirst\ l))\ l1)))) = (NNil\ (nfirst\ (upd\ \sigma\ n\ l)))$

by (*metis* 4 5 6 *One-nat-def nappend-NNil ndropn-0 ndropn-Suc-conv-ndropn nfirst-eq-nnth-zero nfirst-upd nlength-NCons nnth-0 zero-enat-def*)

have 9: $(NNil\ (nfirst\ (upd\ \sigma\ n\ (nappend\ (NNil\ (nfirst\ l))\ l1)))) \models w$

using 7 8 **by** *fastforce*

have 10: $(ndropn\ 1\ (upd\ \sigma\ n\ (nappend\ (NNil\ (nfirst\ l))\ l1)) \models f)$

by (*metis* 5 6 *One-nat-def ileI1 nappend-NNil ndropn-0 ndropn-Suc-NCons ndropn-Suc-conv-ndropn ndropn-upd nlength-NCons one-eSuc one-enat-def zero-enat-def*)

have 11: $nlength\ (nappend\ (NNil\ (nfirst\ l))\ l1) = nlength\ \sigma$

by (*metis* 5 6 *One-nat-def nappend-NNil ndropn-0 ndropn-Suc-conv-ndropn nlength-NCons zero-enat-def*)

have 12: $(upd\ \sigma\ n\ (nappend\ (NNil\ (nfirst\ l))\ l1) \models w\ \text{iand}\ next\ f)$

by (*metis* 10 11 9 *a* *assms* *iand-defs nlength-upd semantics-qpitl.simps(4) state-qpitl-defs*)

have 13: $(\sigma \models (Ex\ n.\ w\ \text{iand}\ next\ f)) \longleftrightarrow$

$(\exists\ l.\ nlength\ l = nlength\ \sigma \wedge (upd\ \sigma\ n\ l \models w\ \text{iand}\ next\ f))$

by (*metis semantics-qpitl.simps(8)*)

show *?thesis*

using 11 12 13 **by** *blast*

qed

qed

show *?thesis* **unfolding** *valid-def*

by (*metis* 1 2 *ieqv-defs*)

qed

lemma *big-iior-state-qpitl-exists-dist-iand*:

```

assumes state-qpitl-list F-now
          length F-now = length F-next
shows  $\vdash ((\text{Ex } n. (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) \text{ F-now F-next})))$ 
      ieqv
       $(\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y))$ 
         $(\text{map } (\lambda x. (\text{Ex } n. x)) \text{ F-now})$ 
         $(\text{map } (\lambda x. (\text{Ex } n. x)) \text{ F-next}))))$ 

proof –
have 1:  $\vdash (\text{Ex } n. (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) \text{ F-now F-next})))$  ieqv
       $(\text{big-ior } (\text{map } (\lambda x. (\text{Ex } n. x)) (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) \text{ F-now F-next}))))$ 
using big-ior-dist-exists by auto
have 2:  $\bigwedge \sigma. (\sigma \models ((\text{big-ior } (\text{map } (\lambda x. (\text{Ex } n. x)) (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y)) \text{ F-now F-next})))) =$ 
       $(\sigma \models (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y))$ 
         $(\text{map } (\lambda x. (\text{Ex } n. x)) \text{ F-now})$ 
         $(\text{map } (\lambda x. (\text{Ex } n. x)) \text{ F-next}))))$ 

proof –
fix  $\sigma$ 
show  $(\sigma \models \text{big-ior } (\text{map } (\text{exists-d } n) (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) \text{ F-now F-next}))) =$ 
       $(\sigma \models \text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) (\text{map } (\text{exists-d } n) \text{ F-now}) (\text{map } (\text{exists-d } n) \text{ F-next}))))$ 
proof –
have 3:  $(\sigma \models \text{big-ior } (\text{map } (\text{exists-d } n) (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) \text{ F-now F-next}))) =$ 
       $(\exists i < \text{length } (\text{map } (\text{exists-d } n) (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) \text{ F-now F-next})).$ 
       $\sigma \models \text{map } (\text{exists-d } n) (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) \text{ F-now F-next}) ! i)$ 
unfolding big-ior-defs by simp
have 4:  $\bigwedge i. i < \text{length } \text{F-now} \implies$ 
       $(\sigma \models \text{map } (\text{exists-d } n) (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) \text{ F-now F-next}) ! i) =$ 
       $(\sigma \models (\text{Ex } n. \text{F-now} ! i \text{ iand } \text{next } \text{F-next} ! i))$ 
by (simp add: assms(2))
have 5:  $\bigwedge i. i < \text{length } \text{F-now} \implies$ 
       $(\sigma \models (\text{Ex } n. \text{F-now} ! i \text{ iand } \text{next } \text{F-next} ! i)) =$ 
       $(\sigma \models (\text{Ex } n. \text{F-now} ! i) \text{ iand } \text{next } (\text{Ex } n. \text{F-next} ! i))$ 
using assms(1) itl-ieq state-qpitl-exists-dist-iand state-qpitl-list-defs by blast
have 6:  $(\sigma \models (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y))$ 
       $(\text{map } (\lambda x. (\text{Ex } n. x)) \text{ F-now})$ 
       $(\text{map } (\lambda x. (\text{Ex } n. x)) \text{ F-next})))) =$ 
       $(\exists i < \text{length } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) (\text{map } (\text{exists-d } n) \text{ F-now}) (\text{map } (\text{exists-d } n) \text{ F-next})).$ 
       $\sigma \models \text{map2 } (\lambda x y. x \text{ iand } \text{next } y) (\text{map } (\text{exists-d } n) \text{ F-now}) (\text{map } (\text{exists-d } n) \text{ F-next}) ! i)$ 
unfolding big-ior-defs by simp
have 7:  $\bigwedge i. i < \text{length } \text{F-now} \implies$ 
       $(\sigma \models \text{map2 } (\lambda x y. x \text{ iand } \text{next } y) (\text{map } (\text{exists-d } n) \text{ F-now}) (\text{map } (\text{exists-d } n) \text{ F-next}) ! i) =$ 
       $(\sigma \models ((\text{map } (\text{exists-d } n) \text{ F-now}) ! i) \text{ iand } \text{next } ((\text{map } (\text{exists-d } n) \text{ F-next}) ! i))$ 
by (simp add: assms(2))
have 8:  $\bigwedge i. i < \text{length } \text{F-now} \implies$ 
       $(\sigma \models ((\text{map } (\text{exists-d } n) \text{ F-now}) ! i) \text{ iand } \text{next } ((\text{map } (\text{exists-d } n) \text{ F-next}) ! i)) =$ 
       $(\sigma \models (\text{Ex } n. \text{F-now} ! i) \text{ iand } \text{next } (\text{Ex } n. \text{F-next} ! i))$ 
using assms(2) by auto
show ?thesis
using 3 5 6 assms(2) by auto
qed
qed

```

```

show ?thesis
using 1 2 by auto
qed

```

lemma *GNF-exists-step*:

```

assumes  $\vdash F \text{ ieqv } ((F\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next\ y))\ F\text{-}now\ F\text{-}next)))$ 
  state-qpitl  $F\text{-}e$ 
  state-qpitl-list  $F\text{-}now$ 
   $length\ F\text{-}now = length\ F\text{-}next$ 

```

```

shows  $(\sigma \models (Ex\ n. F)) \longleftrightarrow$ 
   $(\sigma \models ((Ex\ n. F\text{-}e) \text{ iand empty}) \text{ ior } (big\text{-}ior (map2 (\lambda x y. x \text{ iand } next\ y)$ 
     $(map (\lambda x. (Ex\ n. x))\ F\text{-}now)$ 
     $(map (\lambda x. (Ex\ n. x))\ F\text{-}next))))$ 

```

proof –

```

have 1:  $(\sigma \models (Ex\ n. F)) =$ 
   $(\sigma \models (Ex\ n. ((F\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next\ y))\ F\text{-}now\ F\text{-}next))))))$ 
using assms by simp
have 2:  $(\sigma \models (Ex\ n. ((F\text{-}e \text{ iand empty}) \text{ ior } (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next\ y))\ F\text{-}now\ F\text{-}next))))))$ 
=

```

```

   $((\sigma \models (Ex\ n. (F\text{-}e \text{ iand empty}))) \vee (\sigma \models (Ex\ n. (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next\ y))\ F\text{-}now\ F\text{-}next))))))$ 

```

by *auto*

```

have 3:  $(\sigma \models (Ex\ n. (F\text{-}e \text{ iand empty}))) = (\sigma \models (Ex\ n. F\text{-}e) \text{ iand empty})$ 

```

using *nlength-upd* **by** *auto*

```

have 4:  $(\sigma \models (Ex\ n. (big\text{-}ior (map2 (\lambda x y. x \text{ iand } (next\ y))\ F\text{-}now\ F\text{-}next)))) =$ 
   $(\sigma \models (big\text{-}ior (map2 (\lambda x y. x \text{ iand } next\ y)$ 
     $(map (\lambda x. (Ex\ n. x))\ F\text{-}now)$ 
     $(map (\lambda x. (Ex\ n. x))\ F\text{-}next))))$ 

```

using *assms* *big-ior-state-qpitl-exists-dist-iand itl-ieq* **by** *blast*

```

have 5:  $(\sigma \models ((Ex\ n. F\text{-}e) \text{ iand empty}) \text{ ior } (big\text{-}ior (map2 (\lambda x y. x \text{ iand } next\ y)$ 
   $(map (\lambda x. (Ex\ n. x))\ F\text{-}now)$ 
   $(map (\lambda x. (Ex\ n. x))\ F\text{-}next)))) =$ 
   $((\sigma \models (Ex\ n. F\text{-}e) \text{ iand empty}) \vee$ 
   $(\sigma \models (big\text{-}ior (map2 (\lambda x y. x \text{ iand } next\ y)$ 
     $(map (\lambda x. (Ex\ n. x))\ F\text{-}now)$ 
     $(map (\lambda x. (Ex\ n. x))\ F\text{-}next))))))$ 

```

by *auto*

show ?thesis

using 1 2 3 4 5 **by** *blast*

qed

lemma *GNF-atom-base*:

```

 $\vdash (\$n) \text{ ieqv } (GNF\ \$n)$ 

```

unfolding *GNF-def* *GNF-cons-def* *big-ior-def*

by (*auto*)

(metis add-0 ileI1 linorder-not-less not-less-zero order-neq-le-trans plus-1-eSuc(2))

lemma *GNF-ifalse-base*:

⊢ ifalse ieqv (GNF ifalse)

unfolding *GNF-def GNF-cons-def big-ior-def*

by *auto*

lemma *GNF-next-step*:

⊢ (next f) ieqv (GNF (next f))

unfolding *GNF-def GNF-cons-def big-ior-def*

by (*auto*)

lemma *exists-GNF-nondet*:

∃ ae a A. (state-qpitl ae) ∧ (state-qpitl-list a) ∧ length a = length A ∧
 (⊢ f ieqv (ae iand empty) ior (big-ior (map2 (λ x y. x iand next y) a A)))

proof (*induction f*)

case *false-d*

then show ?case

proof –

have *f1*: ⊢ ifalse ieqv (GNF ifalse)

using *GNF-ifalse-base* **by** *auto*

have *f2*: state-qpitl-list (GNF-state ifalse)

unfolding *state-qpitl-list-def* **by** (*simp add: inot-d-def ittrue-d-def*)

have *f3*: length (GNF-state ifalse) = length (GNF-next ifalse)

by *simp*

show ?thesis

by (*metis f1 f2 f3 GNF-cons-def GNF-def GNF-empty.simps(1) state-qpitl.simps(1)*)

qed

next

case (*atom-d x*)

then show ?case

proof –

have *a1*: ⊢ (\$x) ieqv (GNF (\$x))

using *GNF-atom-base* **by** *blast*

have *a2*: state-qpitl-list (GNF-state (\$x))

unfolding *state-qpitl-list-def* **by** *simp*

have *a3*: length (GNF-state (\$x)) = length (GNF-next (\$x))

by *simp*

show ?thesis

by (*metis GNF-cons-def GNF-def GNF-empty.simps(2) a1 a2 a3 state-qpitl.simps(2)*)

qed

next

case (*iimp-d f1 f2*)

then show ?case

proof –

obtain *ae1 a1 A1* **where** *i1*: state-qpitl ae1 ∧ state-qpitl-list a1 ∧ length a1 = length A1 ∧

(⊢ f1 ieqv ae1 iand empty ior big-ior (map2 (λ x y. x iand next y) a1 A1))

using *iimp-d.IH(1)* **by** *blast*

obtain $ae2\ a2\ A2$ **where** $i2$: $state\text{-}qpitl\ ae2 \wedge state\text{-}qpitl\text{-}list\ a2 \wedge length\ a2 = length\ A2 \wedge$
 $(\vdash f2\ ieqv\ ae2\ iand\ empty\ ior\ big\text{-}ior\ (map2\ (\lambda x\ y.\ x\ iand\ next\ y)\ a2\ A2))$
using $iimp\text{-}d.IH(2)$ **by** $blast$
have $i3$: $\vdash (f1\ iimp\ f2)\ ieqv\ (((inot\ ae1)\ ior\ ae2)\ iand\ empty)\ ior$
 $(big\text{-}ior$
 $(map2\ (\lambda x\ y.\ x\ iand\ (next\ y))$
 $((add\text{-}to\text{-}now\ a1)\ @\ a2)$
 $((map\ (\lambda x.\ (inot\ x))\ (add\text{-}to\text{-}next\ A1))\ @\ A2))$
using $GNF\text{-}iimp\text{-}step\ i1\ i2\ itl\text{-}ieq$ **by** $force$
have $i4$: $length\ ((add\text{-}to\text{-}now\ a1)\ @\ a2) = length\ ((map\ (\lambda x.\ (inot\ x))\ (add\text{-}to\text{-}next\ A1))\ @\ A2)$
by $(simp\ add:\ add\text{-}to\text{-}next\text{-}def\ add\text{-}to\text{-}now\text{-}def\ i1\ i2\ length\text{-}add\text{-}to\text{-}now\text{-}next\text{-}gen)$
have $i5$: $state\text{-}qpitl\text{-}list\ ((add\text{-}to\text{-}now\ a1)\ @\ a2)$
by $(simp\ add:\ add\text{-}to\text{-}now\text{-}def\ i1\ i2\ state\text{-}qpitl\text{-}foldr\text{-}add\text{-}to\text{-}now\text{-}single\text{-}inv\text{-}a\ state\text{-}qpitl\text{-}list\text{-}append)$
show $?thesis$
by $(metis\ i1\ i2\ i3\ i4\ i5\ inot\text{-}d\text{-}def\ ior\text{-}d\text{-}def\ state\text{-}qpitl.simps(1)\ state\text{-}qpitl.simps(3))$
qed
next
case $(next\text{-}d\ f)$
then show $?case$
proof –
obtain $ae\ a\ A$ **where** $n1$: $state\text{-}qpitl\ ae \wedge state\text{-}qpitl\text{-}list\ a \wedge length\ a = length\ A \wedge$
 $(\vdash f\ ieqv\ ae\ iand\ empty\ ior\ big\text{-}ior\ (map2\ (\lambda x\ y.\ x\ iand\ next\ y)\ a\ A))$
using $next\text{-}d.IH$ **by** $blast$
have $n2$: $\vdash (next\ f)\ ieqv\ (GNF\ (next\ f))$
using $GNF\text{-}next\text{-}step$ **by** $blast$
have $n3$: $state\text{-}qpitl\text{-}list\ (GNF\text{-}state\ (next\ f))$
by $(simp\ add:\ inot\text{-}d\text{-}def\ itrue\text{-}d\text{-}def\ state\text{-}qpitl\text{-}list\text{-}defs)$
have $n4$: $length\ (GNF\text{-}state\ (next\ f)) = length\ (GNF\text{-}next\ (next\ f))$
by $simp$
show $?thesis$
by $(metis\ GNF\text{-}cons\text{-}def\ GNF\text{-}def\ GNF\text{-}empty.simps(4)\ n2\ n3\ n4\ state\text{-}qpitl.simps(1))$
qed
next
case $(chop\text{-}d\ f1\ f2)$
then show $?case$
proof –
obtain $ae1\ a1\ A1$ **where** $c1$: $state\text{-}qpitl\ ae1 \wedge state\text{-}qpitl\text{-}list\ a1 \wedge length\ a1 = length\ A1 \wedge$
 $(\vdash f1\ ieqv\ ae1\ iand\ empty\ ior\ big\text{-}ior\ (map2\ (\lambda x\ y.\ x\ iand\ next\ y)\ a1\ A1))$
using $chop\text{-}d.IH(1)$ **by** $blast$
obtain $ae2\ a2\ A2$ **where** $c2$: $state\text{-}qpitl\ ae2 \wedge state\text{-}qpitl\text{-}list\ a2 \wedge length\ a2 = length\ A2 \wedge$
 $(\vdash f2\ ieqv\ ae2\ iand\ empty\ ior\ big\text{-}ior\ (map2\ (\lambda x\ y.\ x\ iand\ next\ y)\ a2\ A2))$
using $chop\text{-}d.IH(2)$ **by** $blast$
have $c3$: $\vdash f1\ schop\ f2\ ieqv\ (ae1\ iand\ ae2)\ iand\ empty\ ior$
 $(big\text{-}ior\ (map2\ (\lambda x\ y.\ x\ iand\ next\ y)$
 $(a1\ @\ (map\ (\lambda x.\ x\ iand\ ae1)\ a2))$
 $((map\ (\lambda x.\ x\ schop\ f2)\ A1)\ @\ A2)))$
using $GNF\text{-}schop\text{-}step[of\ f1\ ae1\ a1\ A1\ f2\ ae2\ a2\ A2]\ c1\ c2\ itl\text{-}ieq$ **by** $blast$
have $c4$: $length\ (a1\ @\ (map\ (\lambda x.\ x\ iand\ ae1)\ a2)) = length\ ((map\ (\lambda x.\ x\ schop\ f2)\ A1)\ @\ A2)$
by $(simp\ add:\ c1\ c2)$
have $c5$: $state\text{-}qpitl\text{-}list\ (a1\ @\ (map\ (\lambda x.\ x\ iand\ ae1)\ a2))$


```

  by (simp add: c1 c2 state-qpitl-list-append state-qpitl-list-map-alt)
have c6: state-qpitl (ae1 iand ae2)
  by (simp add: c1 c2 iand-d-def inot-d-def ior-d-def)
show ?thesis using c1 c2 c3 c4 c5 c6 by blast
qed
next
case (schopstar-d f)
then show ?case
proof -
  obtain ae a A where s1: state-qpitl ae  $\wedge$  state-qpitl-list a  $\wedge$  length a = length A  $\wedge$ 
    ( $\vdash$  f ieqv ae iand empty ior big-ior (map2 ( $\lambda$  x y. x iand next y) a A) )
  using schopstar-d.IH by blast
  have s2:  $\vdash$  schopstar f ieqv (itrue iand empty) ior
    (big-ior (map2 ( $\lambda$  x y. x iand next y) a (map ( $\lambda$  x. x schop (schopstar f)) A)))
  using GNF-schopstar-step[of f ae a A] s1 using itl-ieq by blast
  have s3: length a = length (map ( $\lambda$  x. x schop (schopstar f)) A)
  by (simp add: s1)
  have s4: state-qpitl itrue
  by (simp add: inot-d-def itrue-d-def)
  show ?thesis
  by (metis s1 s2 s3 s4)
qed
next
case (omega-d f)
then show ?case
proof -
  obtain ae a A where o1: state-qpitl ae  $\wedge$  state-qpitl-list a  $\wedge$  length a = length A  $\wedge$ 
    ( $\vdash$  f ieqv ae iand empty ior big-ior (map2 ( $\lambda$  x y. x iand next y) a A) )
  using omega-d.IH by blast
  have o2:  $\vdash$  omega f ieqv (ifalse iand empty) ior
    (big-ior (map2 ( $\lambda$  x y. x iand next y) a (map ( $\lambda$  x. x schop (omega f)) A)))
  using GNF-omega-step[of f ae a A] o1 using itl-ieq by blast
  have o3: length a = length (map ( $\lambda$  x. x schop (omega f)) A)
  by (simp add: o1)
  show ?thesis
  using o1 o2 o3 state-qpitl.simps(1) by blast
qed
next
case (exists-d n f)
then show ?case
proof -
  obtain ae a A where e1: state-qpitl ae  $\wedge$  state-qpitl-list a  $\wedge$  length a = length A  $\wedge$ 
    ( $\vdash$  f ieqv ae iand empty ior big-ior (map2 ( $\lambda$  x y. x iand next y) a A) )
  using exists-d.IH by blast
  have e2:  $\vdash$  (Ex n. f) ieqv
    ((Ex n. ae) iand empty) ior (big-ior (map2 ( $\lambda$  x y. x iand next y)
      (map ( $\lambda$  x. (Ex n. x)) a) (map ( $\lambda$  x. (Ex n. x)) A)))
  using GNF-exists-step[of f ae a A - n] e1 using itl-ieq by blast
  have e3: length (map ( $\lambda$  x. (Ex n. x)) a) = length (map ( $\lambda$  x. (Ex n. x)) A)
  using e1 by simp

```

```

have e4: state-qpitl (Ex n. ae)
  by (simp add: e1)
have e5: state-qpitl-list (map (λ x. (Ex n. x)) a)
  using e1 state-qpitl-list-defs by auto
show ?thesis
  using e1 e2 e3 e4 e5 by blast
qed
qed

```

```

lemma state-qpitl-GNF-empty:
  state-qpitl (GNF-empty f)
proof (induction f)
case false-d
then show ?case by simp
next
case (atom-d x)
then show ?case by simp
next
case (iimp-d f1 f2)
then show ?case apply simp unfolding inot-d-def ior-d-def by simp
next
case (next-d f)
then show ?case by simp
next
case (chop-d f1 f2)
then show ?case apply simp unfolding iand-d-def inot-d-def ior-d-def by simp
next
case (schopstar-d f)
then show ?case apply simp unfolding ittrue-d-def inot-d-def by simp
next
case (omega-d f)
then show ?case by simp
next
case (exists-d x1 f)
then show ?case by simp
qed

```

```

lemma state-qpitl-map-exists:
  assumes state-qpitl-list xs
  shows state-qpitl-list (map (λ x. Ex n. x) xs)
using assms unfolding state-qpitl-list-defs by simp

```

```

lemma state-qpitl-list-GNF-state:
  state-qpitl-list (GNF-state f)
proof (induction f)
case false-d
then show ?case unfolding state-qpitl-list-def apply simp unfolding ittrue-d-def inot-d-def by simp
next

```

```

case (atom-d x)
then show ?case unfolding state-qpitl-list-def by simp
next
case (iimp-d f1 f2)
then show ?case unfolding state-qpitl-list-def apply simp
using add-to-now-def state-qpitl-foldr-add-to-now-single-inv-a state-qpitl-list-def by auto
next
case (next-d f)
then show ?case unfolding state-qpitl-list-def apply simp unfolding itrue-d-def inot-d-def by simp
next
case (chop-d f1 f2)
then show ?case unfolding state-qpitl-list-def apply simp
using state-qpitl-GNF-empty state-qpitl-list-def state-qpitl-list-map-alt by force
next
case (schopstar-d f)
then show ?case unfolding state-qpitl-list-def by simp
next
case (omega-d f)
then show ?case unfolding state-qpitl-list-def by simp
next
case (exists-d x1 f)
then show ?case unfolding state-qpitl-list-def apply simp
using state-qpitl-list-def state-qpitl-map-exists by blast
qed

```

```

lemma length-GNF-state-eq-length-GNF-next:
  length (GNF-state f) = length (GNF-next f)
proof (induction f)
case false-d
then show ?case by simp
next
case (atom-d x)
then show ?case by simp
next
case (iimp-d f1 f2)
then show ?case
by (simp add: add-to-next-def add-to-now-def length-add-to-now-next-gen)
next
case (next-d f)
then show ?case by simp
next
case (chop-d f1 f2)
then show ?case by simp
next
case (schopstar-d f)
then show ?case by simp
next
case (omega-d f)
then show ?case by simp
next

```

```

case (exists-d x1 f)
then show ?case by simp
qed

```

lemma *ieqv-GNF*:

```

  ⊢ f ieqv GNF f
proof (induction f)
case false-d
then show ?case
using GNF-ifalse-base by auto
next
case (atom-d x)
then show ?case using GNF-atom-base by blast
next
case (iimp-d f1 f2)
then show ?case
  proof –
    have i1: ⊢ f1 ieqv GNF-empty f1 iand empty ior big-ior (map2 (λx y. x iand next y) (GNF-state f1)
      (GNF-next f1))
      using iimp-d by (simp add: GNF-cons-def GNF-def)
    have i2: ⊢ f2 ieqv GNF-empty f2 iand empty ior big-ior (map2 (λx y. x iand next y) (GNF-state f2)
      (GNF-next f2))
      using iimp-d by (simp add: GNF-cons-def GNF-def)
    have i3: state-qpitl (GNF-empty f1)
      by (simp add: state-qpitl-GNF-empty)
    have i4: state-qpitl-list (GNF-state f1)
      by (simp add: state-qpitl-list-GNF-state)
    have i5: state-qpitl (GNF-empty f2)
      by (simp add: state-qpitl-GNF-empty)
    have i6: state-qpitl-list (GNF-state f2)
      by (simp add: state-qpitl-list-GNF-state)
    have i7: length (GNF-state f1) = length (GNF-next f1)
      by (simp add: length-GNF-state-eq-length-GNF-next)
    have i8: length (GNF-state f2) = length (GNF-next f2)
      by (simp add: length-GNF-state-eq-length-GNF-next)
    show ?thesis using
      GNF-iimp-step[of f1 GNF-empty f1 (GNF-state f1) (GNF-next f1)
        f2 GNF-empty f2 (GNF-state f2) (GNF-next f2)]
    using GNF-cons-def GNF-def i1 i2 i3 i4 i5 i6 i7 i8 by fastforce
  qed
next
case (next-d f)
then show ?case using GNF-next-step by blast
next
case (chop-d f1 f2)
then show ?case
  proof –
    have c1: ⊢ f1 ieqv GNF-empty f1 iand empty ior big-ior (map2 (λx y. x iand next y) (GNF-state f1)

```

```

(GNF-next f1))
  using chop-d by (simp add: GNF-cons-def GNF-def)
  have c2:  $\vdash f2 \text{ ieqv } \text{GNF-empty } f2 \text{ iand empty ior big-ior } (\text{map2 } (\lambda x y. x \text{ iand next } y) (\text{GNF-state } f2))$ 
(GNF-next f2))
  using chop-d by (simp add: GNF-cons-def GNF-def)
  have c3: state-qpitl (GNF-empty f1)
    by (simp add: state-qpitl-GNF-empty)
  have c4: state-qpitl-list (GNF-state f1)
    by (simp add: state-qpitl-list-GNF-state)
  have c5: state-qpitl (GNF-empty f2)
    by (simp add: state-qpitl-GNF-empty)
  have c6: state-qpitl-list (GNF-state f2)
    by (simp add: state-qpitl-list-GNF-state)
  have c7: length (GNF-state f1) = length (GNF-next f1)
    by (simp add: length-GNF-state-eq-length-GNF-next)
  have c8: length (GNF-state f2) = length (GNF-next f2)
    by (simp add: length-GNF-state-eq-length-GNF-next)
  show ?thesis using
    GNF-schop-step[of f1 GNF-empty f1 (GNF-state f1) (GNF-next f1)
      f2 GNF-empty f2 (GNF-state f2) (GNF-next f2)]
    unfolding GNF-def GNF-cons-def using c1 c2 c3 c4 c5 c6 c7 c8 by fastforce
qed
next
case (schopstar-d f)
then show ?case
  proof –
    have s1:  $\vdash f \text{ ieqv } \text{GNF-empty } f \text{ iand empty ior big-ior } (\text{map2 } (\lambda x y. x \text{ iand next } y) (\text{GNF-state } f))$ 
(GNF-next f))
    using chopstar-d by (simp add: GNF-cons-def GNF-def)
    have s2: state-qpitl (GNF-empty f)
      by (simp add: state-qpitl-GNF-empty)
    have s3: state-qpitl-list (GNF-state f)
      by (simp add: state-qpitl-list-GNF-state)
    have s4: length (GNF-state f) = length (GNF-next f)
      by (simp add: length-GNF-state-eq-length-GNF-next)
    show ?thesis using
      GNF-schopstar-step[of f GNF-empty f (GNF-state f) (GNF-next f)]
      unfolding GNF-def GNF-cons-def using s1 s2 s3 s4 by fastforce
    qed
  next
  case (omega-d f)
  then show ?case
    proof –
      have o1:  $\vdash f \text{ ieqv } \text{GNF-empty } f \text{ iand empty ior big-ior } (\text{map2 } (\lambda x y. x \text{ iand next } y) (\text{GNF-state } f))$ 
(GNF-next f))
      using omega-d by (simp add: GNF-cons-def GNF-def)
      have o2: state-qpitl (GNF-empty f)
        by (simp add: state-qpitl-GNF-empty)
      have o3: state-qpitl-list (GNF-state f)
        by (simp add: state-qpitl-list-GNF-state)

```

```

have o4: length (GNF-state f) = length (GNF-next f)
by (simp add: length-GNF-state-eq-length-GNF-next)
show ?thesis using
  GNF-omega-step[of f GNF-empty f (GNF-state f) (GNF-next f)]
  unfolding GNF-def GNF-cons-def using o1 o2 o3 o4 by fastforce
qed
next
case (exists-d x1 f)
then show ?case
proof -
  have e1:  $\vdash f \text{ ieqv } \text{GNF-empty } f \text{ iand } \text{empty ior } \text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) (\text{GNF-state } f) (\text{GNF-next } f))$ 
  using exists-d by (simp add: GNF-cons-def GNF-def)
  have e2: state-qpitl (GNF-empty f)
  by (simp add: state-qpitl-GNF-empty)
  have e3: state-qpitl-list (GNF-state f)
  by (simp add: state-qpitl-list-GNF-state)
  have e4: length (GNF-state f) = length (GNF-next f)
  by (simp add: length-GNF-state-eq-length-GNF-next)
  show ?thesis using
    GNF-exists-step[of f GNF-empty f (GNF-state f) (GNF-next f)]
    unfolding GNF-def GNF-cons-def using e1 e2 e3 e4 by fastforce
  qed
qed

```

lemma *exist-GNF-det*:

GNF-prop f

proof –

```

have 1:  $\exists ae a A. (\text{state-qpitl } ae) \wedge (\text{state-qpitl-list } a) \wedge \text{length } a = \text{length } A \wedge$ 
  ( $\vdash f \text{ ieqv } (ae \text{ iand } \text{empty}) \text{ ior } (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) a A))$ ))
using exists-GNF-nondet by blast
obtain ae a A where 2:  $(\text{state-qpitl } ae) \wedge (\text{state-qpitl-list } a) \wedge \text{length } a = \text{length } A \wedge$ 
  ( $\vdash f \text{ ieqv } (ae \text{ iand } \text{empty}) \text{ ior } (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) a A))$ ))
using 1 by blast
have 3:  $\vdash (\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } \text{next } y) a A)) \text{ ieqv }$ 
   $\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y))$ 
     $(\text{foldr } (\text{add-to-now-single}) a [\text{itrue}])$ 
     $(\text{foldr } (\text{add-to-next-single}) A [\text{ifalse}])))$ 
using 2 det-big-ior-ieqv-nondet-big-ior-a by auto
have 4:  $\vdash f \text{ ieqv } (ae \text{ iand } \text{empty}) \text{ ior }$ 
   $\text{big-ior } (\text{map2 } (\lambda x y. x \text{ iand } (\text{next } y))$ 
     $(\text{foldr } (\text{add-to-now-single}) a [\text{itrue}])$ 
     $(\text{foldr } (\text{add-to-next-single}) A [\text{ifalse}])))$ 
using 2 3 by fastforce
have 5: state-qpitl-list (foldr (add-to-now-single) a [itrue])
  by (simp add: 2 state-qpitl-foldr-add-to-now-single-inv-a)
have 6: length (foldr (add-to-now-single) a [itrue]) =
  length (foldr (add-to-next-single) A [ifalse])
  by (simp add: 2 length-add-to-now-next-gen)

```

```

have 7: full-system (foldr (add-to-now-single) a [itrue])
  using big-ior-foldr-add-to-now-single-inv-a full-system-def mutex-foldr-add-to-now-single-inv-a by blast
show ?thesis unfolding GNF-prop-def GNF-cons-def
using 2 4 5 6 7 by blast
qed

```

```

lemma state-qpitl-list-GNF-det-state:
  state-qpitl-list (foldr (add-to-now-single) (GNF-state f) [itrue])
by (simp add: state-qpitl-foldr-add-to-now-single-inv-a state-qpitl-list-GNF-state)

```

```

lemma full-system-GNF-det-state:
  full-system (foldr (add-to-now-single) (GNF-state f) [itrue])
using big-ior-foldr-add-to-now-single-inv-a full-system-def
mutex-foldr-add-to-now-single-inv-a by auto

```

```

lemma ieqv-GNF-det:
   $\vdash f \text{ieqv } \text{GNF-det } f$ 
using ieqv-GNF[of f]
unfolding GNF-def GNF-det-def GNF-cons-def
using det-big-ior-ieqv-nondet-big-ior-a length-GNF-state-eq-length-GNF-next by auto

```

```

end

```

Chapter 5

CDT-PITL

theory *CDT-PITL-Deep*

imports

NELList-Extras

begin

This theory contains a deep embedding of PITL with the CDT operators of Yde Venema[14] without chop-star and chop-omega.

5.1 Syntax

5.1.1 Primitive formulae

datatype *cdt* =
 false-d (*ifalse*)
 | *atom-d* *nat* ((*\$-*) [100] 99)
 | *iimp-d* *cdt cdt* ((*- iimp -*) [26,25] 25)
 | *empty-d* (*(empty)*)
 | *c-d* *cdt cdt* ((*- cee -*) [84,84] 83)
 | *d-d* *cdt cdt* ((*- dee -*) [84,84] 83)
 | *t-d* *cdt cdt* ((*- tee -*) [84,84] 83)

5.1.2 Derived Boolean Operators

definition *inot-d* ((*inot -*) [90] 90)

where

inot f \equiv *f iimp ifalse*

definition *itrue-d* (*itrue*)

where

itrue \equiv *inot ifalse*

definition *ior-d* ((*- ior -*) [31,30] 30)

where

f ior g \equiv (*inot f*) *iimp g*

definition *iand-d* ((- *iand* -) [36,35] 35)

where

$f \text{ iand } g \equiv \text{inot } (\text{inot } f \text{ ior } \text{inot } g)$

definition *ieqv-d* ((- *ieqv* -) [21,20] 20)

where

$f \text{ ieqv } g \equiv ((f \text{ iimp } g) \text{ iand } (g \text{ iimp } f))$

5.1.3 ifinite and inf

definition *finite-d* (*ifinite*)

where

$\text{ifinite} \equiv \text{itrue cee empty}$

definition *inf-d* (*inf*)

where

$\text{inf} \equiv \text{inot ifinite}$

5.1.4 weak chop

definition *chop-d* ((- ; -) [84,84] 83)

where $f ; g \equiv (f \text{ cee } g) \text{ ior } (f \text{ iand } \text{inf})$

5.1.5 more, skip, next, wnext, prev and wprev

definition *more-d* (*more*)

where

$\text{more} \equiv \text{inot empty}$

definition *skip-d* (*skip*)

where

$\text{skip} \equiv \text{more iand inot}(\text{more cee more})$

definition *next-d* ((*next* -))

where $\text{next } f \equiv \text{skip cee } f$

definition *wnext-d* ((*wnext* -))

where $\text{wnext } f \equiv (\text{empty ior } \text{next } f)$

definition *prev-d* ((*prev* -))

where $\text{prev } f \equiv f ; \text{skip}$

definition *wprev-d* ((*wprev* -))

where $\text{wprev } f \equiv (\text{empty ior } \text{prev } f)$

5.1.6 Box and Diamond Operators

definition *diamond-d* ((*diamond* -) [88] 87)

where

$\text{diamond } f \equiv \text{itrue cee } f$

definition *box-d* ((*box* -) [88] 87)

where

$box\ f \equiv inot\ (diamond\ (inot\ f))$

definition *df-d* ((*df* -) [88] 87)

where

$df\ f \equiv f\ cee\ itrue$

definition *bf-d* ((*bf* -) [88] 87)

where

$bf\ f \equiv inot\ (df\ (inot\ f))$

definition *ldiamond-d* ((*ldiamond* -) [88] 87)

where *ldiamond* $f \equiv f\ dee\ itrue$

definition *lbox-d* ((*lbox* -) [88] 87)

where

$lbox\ f = inot\ (ldiamond\ (inot\ f))$

definition *rdiamond-d* ((*rdiamond* -) [88] 87)

where *rdiamond* $f \equiv (f\ tee\ ifinite)$

definition *rbox-d* ((*rbox* -) [88] 87)

where

$rbox\ f = inot\ (rdiamond\ (inot\ f))$

5.1.7 Initial and Final Operators

definition *init-d* ((*init* -) [88] 87)

where

$init\ f \equiv ((empty\ iand\ f)\ cee\ itrue)$

definition *fin-d* ((*fin* -) [88] 87)

where

$fin\ f \equiv diamond\ (empty\ iand\ f)$

5.1.8 Horizontal

definition

$hor\ q = ((lbox\ \$q)\ iand\ (box\ \$q)\ iand\ (ifinite\ imp\ rbox\ (fin\ \$q)))$

5.1.9 Compass operators

definition *ediamond-d* ((*ediamond* -) [88] 87)

where

$ediamond\ f \equiv more\ cee\ f$

definition *wdiamond-d* ((*wdiamond* -) [88] 87)

where

$wdiamond\ f \equiv more\ dee\ f$

definition $sdiamond-d\ ((sdiamond\ -)\ [88]\ 87)$

where

$sdiamond\ f \equiv f\ cee\ more$

definition $ndiamond-d\ ((ndiamond\ -)\ [88]\ 87)$

where

$ndiamond\ f \equiv more\ tee\ f$

definition $ewdiamond-d\ ((ewdiamond\ -)\ [88]\ 87)$

where

$ewdiamond\ f \equiv ediamond\ f\ ior\ f\ ior\ wdiamond\ f$

definition $nsdiamond-d\ ((nsdiamond\ -)\ [88]\ 87)$

where

$nsdiamond\ f \equiv sdiamond\ f\ ior\ f\ ior\ ndiamond\ f$

definition $ebox-d\ ((ebox\ -)\ [88]\ 87)$

where

$ebox\ f = inot\ (ediamond\ (inot\ f))$

definition $wbox-d\ ((wbox\ -)\ [88]\ 87)$

where

$wbox\ f = inot\ (wdiamond\ (inot\ f))$

definition $sbox-d\ ((sbox\ -)\ [88]\ 87)$

where

$sbox\ f = inot\ (sdiamond\ (inot\ f))$

definition $nbox-d\ ((nbox\ -)\ [88]\ 87)$

where

$nbox\ f = inot\ (ndiamond\ (inot\ f))$

definition $lex-d\ ((lex))$

where $lex \equiv wbox\ ifalse$

definition $rex-d\ ((rex))$

where $rex \equiv nbox\ ifalse$

definition $ewbox-d\ ((ewbox\ -)\ [88]\ 87)$

where

$ewbox\ f = inot\ (ewdiamond\ (inot\ f))$

definition $nsbox-d\ ((nsbox\ -)\ [88]\ 87)$

where

$nsbox\ f = inot\ (nsdiamond\ (inot\ f))$

definition $ahor-d\ ((ahor\ -)\ [88]\ 87)$

where $ahor\ f \equiv ewbox\ f\ iand\ nbox\ (ewbox\ (inot\ f))\ iand\ sbox\ (ewbox\ (inot\ f))$

definition *aver-d* ((*aver* -) [88] 87)
where *aver f* \equiv *nsbox f iand ebox (nsbox(inot f)) iand wbox(nsbox (inot f))*

5.2 Semantics

5.2.1 Intervals

type-synonym *interval* = *nat set nellist*

fun *semantics-cdt* :: [*interval*, *nat*, *enat*, *cdt*] \Rightarrow *bool* ((- - \models_i -) [80,10] 10)
where
 (*g i j* \models_i *ifalse*) = *False*
 | (*g i j* \models_i *\$p*) = ((*enat i*) \leq *nlength g* \wedge *p* \in (*nnth g i*))
 | (*g i j* \models_i *F iimp G*) = ((*g i j* \models_i *F*) \longrightarrow (*g i j* \models_i *G*))
 | (*g i j* \models_i *empty*) = ((*enat i*) = *j*)
 | (*g i j* \models_i *F cee G*) = (\exists *k*. (*enat i*) \leq (*enat k*) \wedge (*enat k*) \leq *j* \wedge (*g i k* \models_i *F*) \wedge (*g k j* \models_i *G*))
 | (*g i j* \models_i *F tee G*) = (\exists *k l*. *j* < ∞ \wedge *l* = *the-enat j* \wedge *j* \leq *k* \wedge *k* \leq *nlength g* \wedge (*g l k* \models_i *F*) \wedge (*g i k* \models_i *G*))
 | (*g i j* \models_i *F dee G*) = (\exists *k*. *0* \leq *k* \wedge *k* \leq *i* \wedge (*g k i* \models_i *F*) \wedge (*g k j* \models_i *G*))

primrec *fvars* :: *cdt* \Rightarrow *nat set*

where

fvars ifalse = {}
 | *fvars (\$p)* = {*p*}
 | *fvars (f iimp g)* = (*fvars f*) \cup (*fvars g*)
 | *fvars empty* = {}
 | *fvars (f cee g)* = (*fvars f*) \cup (*fvars g*)
 | *fvars (f tee g)* = *fvars f* \cup (*fvars g*)
 | *fvars (f dee g)* = *fvars f* \cup (*fvars g*)

definition *upd* :: *interval* \Rightarrow *nat* \Rightarrow (*bool nellist*) \Rightarrow *interval*

where

upd σ *n l* = *nmap* (λ (*x,y*). (if *y* then insert *n x* else *x* - {*n*})) (*nzip* σ *l*)

lemma *upd-NCons*:

upd (*NCons x xs*) *n* (*NCons y ys*) =
 (if *y* then *NCons* (*insert n x*) (*upd xs n ys*) else *NCons* (*x* - {*n*})) (*upd xs n ys*)
by (*simp add: upd-def*)

lemma *upd-NNil*:

upd (NNil *a*) *n* (NNil *b*) =
 (if *b* then (NNil (insert *n a*)) else (NNil (*a* - {*n*})))

by (*simp add: nfirst-eq-nnth-zero nnth-NNil upd-def*)

lemma *nnth-upd*:

assumes *nlength* σ = *nlength* *l*

(*enat* *k*) \leq *nlength* σ

shows (*nnth* (*upd* σ *n l*) *k*) = (if (*nnth* *l k*) then (insert *n* (*nnth* σ *k*)) else (*nnth* σ *k*) - {*n*})

using *assms*

by (*simp add: upd-def nnth-nzip*)

lemma *nlength-upd*:

assumes *nlength* σ = *nlength* *l*

shows *nlength* ((*upd* σ *n l*)) = *nlength* σ

using *assms* **by** (*simp add: upd-def*)

lemma *upd-swap*:

assumes *nlength* σ = *nlength* *l1*

nlength σ = *nlength* *l2*

x1 \neq *x2*

shows *upd* (*upd* σ *x1 l1*) *x2 l2* = (*upd* (*upd* σ *x2 l2*) *x1 l1*)

proof –

have 1: *nlength* (*upd* (*upd* σ *x1 l1*) *x2 l2*) = *nlength* (*upd* (*upd* σ *x2 l2*) *x1 l1*)

using *assms nlength-upd* **by** *auto*

have 2: $\bigwedge i. i \leq \text{nlength } (\text{upd } (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \implies$

(*nnth* (*upd* (*upd* σ *x1 l1*) *x2 l2*) *i*) =

(if (*nnth* *l2 i*) then insert *x2* (*nnth* (*upd* σ *x1 l1*) *i*) else (*nnth* (*upd* σ *x1 l1*) *i*) - {*x2*})

using *assms* **by** (*simp add: nlength-upd nnth-upd*)

have 3: $\bigwedge i. i \leq \text{nlength } (\text{upd } (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \implies$

(*nnth* (*upd* (*upd* σ *x2 l2*) *x1 l1*) *i*) =

(if (*nnth* *l1 i*) then insert *x1* (*nnth* (*upd* σ *x2 l2*) *i*) else (*nnth* (*upd* σ *x2 l2*) *i*) - {*x1*})

using *assms* **by** (*simp add: nlength-upd nnth-upd*)

have 4: $\bigwedge i. i \leq \text{nlength } (\text{upd } (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \implies$

(if (*nnth* *l2 i*) then insert *x2* (*nnth* (*upd* σ *x1 l1*) *i*) else (*nnth* (*upd* σ *x1 l1*) *i*) - {*x2*}) =

(if (*nnth* *l1 i*) then insert *x1* (*nnth* (*upd* σ *x2 l2*) *i*) else (*nnth* (*upd* σ *x2 l2*) *i*) - {*x1*})

using *assms nnth-upd nlength-upd* **by** *auto*

show *?thesis*

by (*simp add: 1 2 3 4 nllist-eq-nnth-eq*)

qed

lemma *upd-absorb*:

assumes *nlength* σ = *nlength* *l1*

nlength σ = *nlength* *l2*

x1 = *x2*

shows *upd* (*upd* σ *x1 l1*) *x2 l2* = (*upd* σ *x2 l2*)

proof –

have 1: *nlength* (*upd* (*upd* σ *x1 l1*) *x2 l2*) = *nlength* (*upd* σ *x2 l2*)

using *assms nlength-upd* **by** *auto*
have 2: $\bigwedge i. i \leq \text{nlength } (\text{upd } (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \implies$
 $(\text{nnth } (\text{upd } (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \ i) =$
 $(\text{if } (\text{nnth } l2 \ i) \text{ then insert } x2 \ (\text{nnth } (\text{upd } \sigma \ x1 \ l1) \ i) \text{ else } (\text{nnth } (\text{upd } \sigma \ x1 \ l1) \ i) - \{x2\})$
using *assms* **by** (*simp add: nlength-upd nnth-upd*)
have 3: $\bigwedge i. i \leq \text{nlength } (\text{upd } (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \implies$
 $(\text{nnth } (\text{upd } \sigma \ x2 \ l2) \ i) =$
 $(\text{if } (\text{nnth } l2 \ i) \text{ then insert } x2 \ (\text{nnth } \sigma \ i) \text{ else } (\text{nnth } \sigma \ i) - \{x2\})$
using *assms* **by** (*simp add: nlength-upd nnth-upd*)
have 4: $\bigwedge i. i \leq \text{nlength } (\text{upd } (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \implies$
 $(\text{if } (\text{nnth } l2 \ i) \text{ then insert } x2 \ (\text{nnth } (\text{upd } \sigma \ x1 \ l1) \ i) \text{ else } (\text{nnth } (\text{upd } \sigma \ x1 \ l1) \ i) - \{x2\}) =$
 $(\text{if } (\text{nnth } l2 \ i) \text{ then insert } x2 \ (\text{nnth } \sigma \ i) \text{ else } (\text{nnth } \sigma \ i) - \{x2\})$
using *assms nnth-upd nlength-upd* **by** *auto*
show *?thesis*
by (*simp add: 1 2 3 4 nllist-eq-nnth-eq*)
qed

lemma *upd-nappend*:
assumes $\text{nlength } \sigma 1 = \text{nlength } l1$
 $\text{nlength } \sigma 2 = \text{nlength } l2$
shows $\text{upd } (\text{nappend } \sigma 1 \ \sigma 2) \ x \ (\text{nappend } l1 \ l2) =$
 $(\text{nappend } (\text{upd } \sigma 1 \ x \ l1) \ (\text{upd } \sigma 2 \ x \ l2))$
using *assms*
by (*simp add: nmap-nappend-distrib nzip-nappend upd-def*)

lemma *ntaken-upd*:
assumes $\text{nlength } \sigma = \text{nlength } l$
 $j \leq \text{nlength } \sigma$
shows $\text{ntaken } j \ (\text{upd } \sigma \ x \ l) = (\text{upd } (\text{ntaken } j \ \sigma) \ x \ (\text{ntaken } j \ l))$
using *assms*
by (*simp add: ntaken-nzip upd-def*)

lemma *ndropn-upd*:
assumes $\text{nlength } \sigma = \text{nlength } l$
 $j \leq \text{nlength } \sigma$
shows $\text{ndropn } j \ (\text{upd } \sigma \ x \ l) = (\text{upd } (\text{ndropn } j \ \sigma) \ x \ (\text{ndropn } j \ l))$
using *assms ndropn-nzip upd-def* **by** (*simp add: ndropn-nmap*)

lemma *nsubn-upd*:
assumes $\text{nlength } \sigma = \text{nlength } l$
 $(\text{enat } j) \leq \text{enat } k$
 $(\text{enat } k) \leq \text{nlength } \sigma$
shows $\text{nsubn } (\text{upd } \sigma \ x \ l) \ j \ k = (\text{upd } (\text{nsubn } \sigma \ j \ k) \ x \ (\text{nsubn } l \ j \ k))$
proof –
have 1: $\text{nsubn } (\text{upd } \sigma \ x \ l) \ j \ k = \text{ntaken } (k - j) \ (\text{ndropn } j \ (\text{upd } \sigma \ x \ l))$
using *assms unfolding nsubn-def1* **by** *simp*
have 2: $(\text{ndropn } j \ (\text{upd } \sigma \ x \ l)) = (\text{upd } (\text{ndropn } j \ \sigma) \ x \ (\text{ndropn } j \ l))$

using *assms* **by** (*meson dual-order.trans ndropn-upd*)
have 3: *ntaken* (*k - j*) (*upd* (*ndropn j σ*) *x* (*ndropn j l*)) =
 (*upd* (*ntaken* (*k-j*) (*ndropn j σ*)) *x* (*ntaken* (*k-j*) (*ndropn j l*)))
using *assms* *ntaken-upd[of (ndropn j σ) (ndropn j l) (k - j) x]*
by (*metis ndropn-nlength nle-le nlength-upd ntaken-all*)
have 4: (*upd* (*ntaken* (*k-j*) (*ndropn j σ*)) *x* (*ntaken* (*k-j*) (*ndropn j l*))) =
 (*upd* (*nsubn σ j k*) *x* (*nsubn l j k*))
unfolding *nsubn-def1* **by** *simp*
show ?thesis **using** 1 2 3 4 **by** *presburger*
qed

lemma *nfirst-upd*:
assumes *nlength σ = nlength l*
shows *nfirst* (*upd σ x l*) = (*if nfirst l then insert x (nfirst σ) else (nfirst σ) - {x}*)
by (*metis assms nfirst-eq-nnth-zero nnth-upd zero-enat-def zero-le*)

lemma *nlast-upd*:
assumes *nlength σ = nlength l*
 nfinite σ
shows *nlast* (*upd σ x l*) = (*if nlast l then insert x (nlast σ) else (nlast σ) - {x}*)
using *assms*
by (*metis dual-order.order-iff-strict nfinite-conv-nlength-enat nlength-upd nnth-nlast nnth-upd the-enat.simps*)

lemma *nfinite-upd*:
assumes *nlength σ = nlength l*
shows *nfinite* (*upd σ x l*) \longleftrightarrow *nfinite σ* \wedge *nfinite l*
using *assms*
by (*simp add: nfinite-conv-nlength-enat nlength-upd*)

lemma *upd-nfuse*:
assumes *nlength σ1 = nlength l1*
 nlength σ2 = nlength l2
 nfirst σ2 = nlast σ1
 nfinite σ1
 nfirst l2 = nlast l1
shows *upd* (*nfuse σ1 σ2*) *x* (*nfuse l1 l2*) =
 (*nfuse* (*upd σ1 x l1*) (*upd σ2 x l2*))
using *assms* **by** (*simp add: nfuse-nlength nmap-nfuse nzip-nfuse upd-def*)

lemma *nrev-upd*:
assumes *nfinite g*
 nlength g = nlength l
shows *nrev* (*upd g n l*) = (*upd* (*nrev g*) *n* (*nrev l*))
proof –
have 1: *nlength* (*nrev* (*upd g n l*)) = *nlength* (*upd* (*nrev g*) *n* (*nrev l*))
by (*simp add: assms(2) nlength-upd*)
have 2: $\bigwedge i. i \leq nlength$ (*nrev* (*upd g n l*)) \longrightarrow
 (*nnth* (*nrev* (*upd g n l*)) *i*) =
 (*nnth* (*upd* (*nrev g*) *n* (*nrev l*)) *i*)
using *assms* **by** (*auto simp add: nlength-upd nnth-nrev nnth-upd nfinite-conv-nlength-enat*)

show ?thesis
 using 1 2 nellist-eq-nnth-eq by blast
 qed

lemma upd-exist:

$\exists r2\ ur2. \text{length } r2 = \text{length } r1 \wedge \text{length } ur2 = \text{length } r1 \wedge r1 = \text{upd } r2\ n\ ur2$

proof –

have 1: $\exists ur1. \text{length } ur1 = \text{length } r1 \wedge (\forall i. i \leq \text{length } r1 \longrightarrow (\text{nnth } ur1\ i) = (n \in (\text{nnth } r1\ i)))$

using nlength-nmap nnth-nmap by blast

have 2: $\exists r3. \text{length } r3 = \text{length } r1 \wedge (\forall i. i \leq \text{length } r1 \longrightarrow (\text{nnth } r3\ i) = (\text{nnth } r1\ i) - \{n\})$

by (metis nlength-nmap nnth-nmap Set.remove-eq)

obtain ur1 where 3: $\text{length } ur1 = \text{length } r1 \wedge (\forall i. i \leq \text{length } r1 \longrightarrow (\text{nnth } ur1\ i) = (n \in (\text{nnth } r1\ i)))$

using 1 by blast

obtain r3 where 4: $\text{length } r3 = \text{length } r1 \wedge (\forall i. i \leq \text{length } r1 \longrightarrow (\text{nnth } r3\ i) = (\text{nnth } r1\ i) - \{n\})$

using 2 by blast

have 5: $r1 = \text{upd } r3\ n\ ur1$

by (metis 3 4 Diff-empty Diff-insert0 insert-Diff nellist-eq-nnth-eq nlength-upd nnth-upd)

show ?thesis

using 3 4 5 by auto

qed

lemma not-fvar-upd:

assumes $n \notin \text{fvars } f$

$\text{length } ug = \text{length } g$

$i \leq j$

$j \leq \text{length } g$

shows $(g\ i\ j \models_i f) = ((\text{upd } g\ n\ ug)\ i\ j \models_i f)$

using assms

proof (induction f arbitrary: g i j ug)

case false-d

then show ?case by auto

next

case (atom-d x)

then show ?case

by (simp add: nlength-upd nnth-upd)

next

case (iimp-d f1 f2)

then show ?case by simp

next

case empty-d

then show ?case using nlength-upd by auto

next

case (c-d f1 f2)

then show ?case

proof –

have c1: $(g\ i\ j \models_i f1\ \text{cee } f2) =$

$(\exists k \geq i. \text{enat } k \leq j \wedge (g\ i\ (\text{enat } k) \models_i f1) \wedge (g\ k\ j \models_i f2))$


```

by simp
have c2: ((upd g n ug) i j  $\models_i$  f1 cee f2) =
  ( $\exists k \geq i. \text{enat } k \leq j \wedge ((\text{upd } g \text{ n } ug) i (\text{enat } k) \models_i f1) \wedge ((\text{upd } g \text{ n } ug) k j \models_i f2)$ )
  by simp
have c3: ( $\exists k \geq i. \text{enat } k \leq j \wedge (g i (\text{enat } k) \models_i f1) \wedge (g k j \models_i f2)$ )  $\implies$ 
  ( $\exists k \geq i. \text{enat } k \leq j \wedge ((\text{upd } g \text{ n } ug) i (\text{enat } k) \models_i f1) \wedge ((\text{upd } g \text{ n } ug) k j \models_i f2)$ )
using c-d by auto
have c5: ( $\exists k \geq i. \text{enat } k \leq j \wedge ((\text{upd } g \text{ n } ug) i (\text{enat } k) \models_i f1) \wedge ((\text{upd } g \text{ n } ug) k j \models_i f2)$ )
 $\implies$  ( $\exists k \geq i. \text{enat } k \leq j \wedge (g i (\text{enat } k) \models_i f1) \wedge (g k j \models_i f2)$ )
using c-d by auto
show ?thesis using c1 c2 c3 c5 by argo
qed
next
case (d-d f1 f2)
then show ?case
proof -
  have d1: (g i j  $\models_i$  f1 dee f2) =
    ( $\exists k \leq i. (g k (\text{enat } i) \models_i f1) \wedge (g k j \models_i f2)$ )
    by simp
  have d2: ((upd g n ug) i j  $\models_i$  f1 dee f2) =
    ( $\exists k \leq i. ((\text{upd } g \text{ n } ug) k (\text{enat } i) \models_i f1) \wedge ((\text{upd } g \text{ n } ug) k j \models_i f2)$ )
    by simp
  have d3: ( $\exists k \leq i. (g k (\text{enat } i) \models_i f1) \wedge (g k j \models_i f2)$ )  $\implies$ 
    ( $\exists k \leq i. ((\text{upd } g \text{ n } ug) k (\text{enat } i) \models_i f1) \wedge ((\text{upd } g \text{ n } ug) k j \models_i f2)$ )
  using d-d
  by (metis Un-iff dual-order.trans fvars.simps(7) of-nat-eq-enat of-nat-mono)
  have d4: ( $\exists k \leq i. ((\text{upd } g \text{ n } ug) k (\text{enat } i) \models_i f1) \wedge ((\text{upd } g \text{ n } ug) k j \models_i f2)$ )  $\implies$ 
    ( $\exists k \leq i. (g k (\text{enat } i) \models_i f1) \wedge (g k j \models_i f2)$ )
  using d-d
  by (metis Un-iff dual-order.trans enat-ord-simps(1) fvars.simps(7))
  show ?thesis
  using d1 d2 d3 d4 by argo
qed
next
case (t-d f1 f2)
then show ?case
proof -
  have t1: (g i j  $\models_i$  f1 tee f2) =
    ( $(\exists i. j = \text{enat } i) \wedge (\exists k \geq j. k \leq \text{nlength } g \wedge (g (\text{the-enat } j) k \models_i f1) \wedge (g i k \models_i f2))$ )
    by simp
  have t2: ((upd g n ug) i j  $\models_i$  f1 tee f2) =
    ( $(\exists i. j = \text{enat } i) \wedge (\exists k \geq j. k \leq \text{nlength } (\text{upd } g \text{ n } ug) \wedge ((\text{upd } g \text{ n } ug) (\text{the-enat } j) k \models_i f1) \wedge ((\text{upd } g \text{ n } ug) i k \models_i f2))$ )
    by simp
  have t3: ( $(\exists i. j = \text{enat } i) \wedge (\exists k \geq j. k \leq \text{nlength } g \wedge (g (\text{the-enat } j) k \models_i f1) \wedge (g i k \models_i f2))$ )  $\implies$ 
    ( $(\exists i. j = \text{enat } i) \wedge (\exists k \geq j. k \leq \text{nlength } (\text{upd } g \text{ n } ug) \wedge ((\text{upd } g \text{ n } ug) (\text{the-enat } j) k \models_i f1) \wedge ((\text{upd } g \text{ n } ug) i k \models_i f2))$ )
  using t-d by simp
  (metis (no-types, lifting) nlength-upd order-trans the-enat.simps)
  have t4: ( $(\exists i. j = \text{enat } i) \wedge (\exists k \geq j. k \leq \text{nlength } (\text{upd } g \text{ n } ug) \wedge ((\text{upd } g \text{ n } ug) (\text{the-enat } j) k \models_i f1) \wedge$ 

```

$((\text{upd } g \ n \ ug) \ i \ k \models_i f2)))$
 \implies
 $((\exists i. j = \text{enat } i) \wedge (\exists k \geq j. k \leq \text{nlength } g \wedge (g \ (\text{the-enat } j) \ k \models_i f1) \wedge (g \ i \ k \models_i f2))))$
using *t-d* **by** *simp*
 $(\text{metis dual-order.trans nlength-upd the-enat.simps})$
show *?thesis*
using *t1 t2 t3 t4* **by** *argo*
qed
qed

lemma *inot-defs* :
shows $(g \ i \ j \models_i \text{inot } f) = \text{Not } (g \ i \ j \models_i f)$
by (*simp add: inot-d-def*)

lemma *ior-defs* :
 $(g \ i \ j \models_i f1 \ \text{ior} \ f2) = ((g \ i \ j \models_i f1) \vee (g \ i \ j \models_i f2))$
by (*metis inot-defs ior-d-def semantics-cdt.simps(3)*)

lemma *iand-defs* :
 $(g \ i \ j \models_i f1 \ \text{iand} \ f2) = ((g \ i \ j \models_i f1) \wedge (g \ i \ j \models_i f2))$
using *iand-d-def inot-defs ior-defs* **by** *presburger*

lemma *ieqv-defs* :
 $(g \ i \ j \models_i f1 \ \text{ieqv} \ f2) = ((g \ i \ j \models_i f1) = (g \ i \ j \models_i f2))$
using *iand-defs ieqv-d-def* **by** *auto*

lemma *itrue-defs* :
 $(g \ i \ j \models_i \text{itrue})$
by (*simp add: inot-d-def itrue-d-def*)

lemma *empty-defs* :
 $(g \ i \ j \models_i \text{empty}) = (i = j)$
by *simp*

lemma *more-defs*:
 $(g \ i \ j \models_i \text{more}) \longleftrightarrow (i \neq j)$
unfolding *more-d-def*
using *empty-defs i0-less inot-defs* **by** *presburger*

lemma *skip-defs*:
assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{skip}) \longleftrightarrow j = \text{enat } (\text{Suc } i)$
proof –
have 1: $(g \ i \ j \models_i \text{skip}) \longleftrightarrow (g \ i \ j \models_i \text{more} \ \text{iand} \ \text{inot } (\text{more } \text{cee } \text{more}))$
unfolding *skip-d-def* **by** *blast*
have 2: $(g \ i \ j \models_i \text{more} \ \text{iand} \ \text{inot } (\text{more } \text{cee } \text{more})) \longleftrightarrow$
 $(g \ i \ j \models_i \text{more}) \wedge (g \ i \ j \models_i \text{inot } (\text{more } \text{cee } \text{more}))$
using *iand-defs* **by** *presburger*
have 3: $(g \ i \ j \models_i \text{inot } (\text{more } \text{cee } \text{more})) \longleftrightarrow \text{Not } (g \ i \ j \models_i (\text{more } \text{cee } \text{more}))$

using *inot-defs* **by** *presburger*
have 4: $(g\ i\ j \models_i (\text{more cee more})) \longleftrightarrow$
 $(\exists k \geq i. \text{enat } k \leq j \wedge (g\ i\ (\text{enat } k) \models_i \text{more}) \wedge (g\ k\ j \models_i \text{more}))$
by *simp*
have 5: $(\exists k \geq i. \text{enat } k \leq j \wedge (g\ i\ (\text{enat } k) \models_i \text{more}) \wedge (g\ k\ j \models_i \text{more})) \longleftrightarrow$
 $(\exists k \geq i. \text{enat } k \leq j \wedge i \neq (\text{enat } k) \wedge k \neq j)$
using *more-defs* **by** *presburger*
have 7: $(\exists k \geq i. \text{enat } k \leq j \wedge i \neq (\text{enat } k) \wedge k \neq j) \implies$
 $j > \text{enat } (\text{Suc } i)$
by (*metis Suc-ile-eq assms(1) enat-ord-simps(1) nle-le order-neq-le-trans*)
have 8: $j > \text{enat } (\text{Suc } i) \implies (\exists k \geq i. \text{enat } k \leq j \wedge i \neq (\text{enat } k) \wedge k \neq j)$
by (*metis enat.inject le-add2 linorder-neq-iff n-not-Suc-n order-less-imp-le plus-1-eq-Suc*)
have 9: $(\exists k \geq i. \text{enat } k \leq j \wedge i \neq (\text{enat } k) \wedge k \neq j) \longleftrightarrow$
 $j > \text{enat } (\text{Suc } i)$
using 7 8 **by** *blast*
have 10: $(g\ i\ j \models_i (\text{more cee more})) \longleftrightarrow (j > \text{enat } (\text{Suc } i))$
using 4 5 7 8 **by** *argo*
have 11: $(g\ i\ j \models_i \text{more}) \wedge (g\ i\ j \models_i \text{inot } (\text{more cee more})) \longleftrightarrow$
 $i \neq j \wedge \text{Not}(j > \text{enat } (\text{Suc } i))$
using 10 3 *more-defs* **by** *presburger*
have 12: $i \neq j \wedge \text{Not}(j > \text{enat } (\text{Suc } i)) \longleftrightarrow j = \text{enat } (\text{Suc } i)$
by (*metis Suc-ile-eq assms(1) dual-order.irrefl order-neq-le-trans*)
show ?thesis
using 1 11 12 2 **by** *presburger*
qed

lemma *skip-cee-itrue-defs*:

assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g\ i\ j \models_i (\text{skip cee itrue})) = (i < j)$
proof –
have 1: $(g\ i\ j \models_i (\text{skip cee itrue})) \implies (i < j)$
using *assms nless-le skip-defs* **by** *fastforce*
have 2: $(i < j) \implies (g\ i\ j \models_i (\text{skip cee itrue}))$
using *assms*
by (*metis eSuc-enat ileI1 ile-eSuc itrue-defs order.trans semantics-cdt.simps(5) skip-defs*)
show ?thesis **using** 1 2 **by** *blast*
qed

lemma *itrue-cee-skip-defs*:

assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g\ i\ j \models_i (\text{itrue cee skip})) = (i < j \wedge (\exists k. j = (\text{enat } k)))$
proof –
have 1: $(g\ i\ j \models_i (\text{itrue cee skip})) \implies (i < j \wedge (\exists k. j = (\text{enat } k)))$
using *assms* **by** *simp*
 $(\text{metis enat-ord-simps(1) iand-defs more-defs nle-le order-neq-le-trans skip-d-def skip-defs})$
have 2: $(i < j \wedge (\exists k. j = (\text{enat } k))) \implies (g\ i\ j \models_i (\text{itrue cee skip}))$
proof –
assume *a0*: $(i < j \wedge (\exists k. j = (\text{enat } k)))$

```

show ( $g\ i\ j \models_i (\text{itrue}\ cee\ skip)$ )
proof –
  have 21: ( $g\ i\ j \models_i (\text{itrue}\ cee\ skip)$ ) = ( $\exists k \geq i. \text{enat}\ k \leq j \wedge j = (\text{enat}\ (Suc\ k))$ )
    using assms
    by (meson enat-ord-simps(1) itrue-defs semantics-cdt.simps(5) skip-defs)
  have 22: ( $\exists k \geq i. \text{enat}\ k \leq j \wedge j = (\text{enat}\ (Suc\ k))$ )
    using assms a0
    using Nat.lessE by fastforce
  show ?thesis
  using 21 22 by blast
qed
qed
show ?thesis using 1 2 by blast
qed

```

```

lemma ifinite-defs:
assumes  $i \leq j$ 
           $j \leq \text{nlength}\ g$ 
shows ( $g\ i\ j \models_i \text{ifinite}$ )  $\longleftrightarrow (\exists k. j = (\text{enat}\ k))$ 
using assms unfolding finite-d-def itrue-d-def inot-d-def
by auto

```

```

lemma inf-defs:
assumes  $i \leq j$ 
           $j \leq \text{nlength}\ g$ 
shows ( $g\ i\ j \models_i \text{inf}$ )  $\longleftrightarrow j = \infty$ 
using assms
unfolding inf-d-def by (simp add: ifinite-defs inot-defs)

```

```

lemma chop-defs:
assumes  $i \leq j$ 
           $j \leq \text{nlength}\ g$ 
shows ( $g\ i\ j \models_i f1 ; f2$ ) =
  ( $\exists k. (\text{enat}\ i) \leq (\text{enat}\ k) \wedge (\text{enat}\ k) \leq j \wedge (g\ i\ k \models_i f1) \wedge (g\ k\ j \models_i f2)$ )  $\vee$ 
  ( $j = \infty \wedge (g\ i\ j \models_i f1)$ )
using assms apply (simp add: iand-defs ior-defs chop-d-def inf-defs)
by auto

```

```

lemma diamond-defs:
assumes  $i \leq j$ 
           $j \leq \text{nlength}\ g$ 
shows ( $g\ i\ j \models_i \text{diamond}\ f$ )  $\longleftrightarrow$ 
  ( $\exists k. i \leq k \wedge k \leq j \wedge (g\ k\ j \models_i f)$ )
using assms
unfolding diamond-d-def
by (simp add: inot-defs itrue-defs)

```

```

lemma box-defs:
assumes  $i \leq j$ 

```

$j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{box } f) \longleftrightarrow$
 $(\forall k. i \leq k \wedge k \leq j \longrightarrow (g \ k \ j \models_i f))$
using *assms* **unfolding** *box-d-def*
by (*simp add: inot-defs diamond-defs*)
blast

lemma *fin-defs*:
assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i (\text{fin } \$p)) \longleftrightarrow ((\exists k. j = (\text{enat } k)) \wedge p \in (\text{nnth } g \ (\text{the-enat } j)))$
using *assms*
unfolding *fin-d-def*
by (*auto simp add: diamond-defs iand-defs*)

lemma *df-defs*:
assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{df } f) \longleftrightarrow$
 $(\exists k. i \leq k \wedge k \leq j \wedge (g \ i \ (\text{enat } k) \models_i f))$
using *assms*
unfolding *df-d-def*
by (*simp add: inot-defs itrue-defs*)

lemma *atom-diamond-defs*:
assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{diamond } \$q) \longleftrightarrow (\exists k. i \leq k \wedge k \leq j \wedge q \in (\text{nnth } g \ k))$
using *assms diamond-defs*
by *auto*

lemma *atom-df-fin-defs*:
assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{df } (\text{fin } \$q)) \longleftrightarrow (\exists k. i \leq k \wedge k \leq j \wedge q \in (\text{nnth } g \ k))$
using *assms df-defs fin-defs*
by *auto*

lemma *atom-box-defs*:
assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{box } \$q) \longleftrightarrow (\forall k. i \leq k \wedge k \leq j \longrightarrow q \in (\text{nnth } g \ k))$
using *assms box-defs*
by *auto*

lemma *atom-fin-defs*:
assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{fin } \$q) \longleftrightarrow ((\exists k. j = (\text{enat } k) \wedge q \in (\text{nnth } g \ (\text{the-enat } j))))$
using *assms*

using *fin-defs nnth-nlast* **by** *blast*

lemma *atom-lbox-defs*:

assumes $i \leq j$

$j \leq \text{nlength } g$

shows $(g \ i \ j \models_i (\text{lbox } \$q)) \longleftrightarrow$
 $(\forall k. k \leq i \longrightarrow q \in (\text{nnth } g \ k))$

using *assms* **by** (*auto simp add: inot-defs lbox-d-def ldiamond-d-def ittrue-defs*)
(meson enat-ord-simps(1) order-trans)

lemma *atom-rbox-defs*:

assumes $i \leq j$

$j \leq \text{nlength } g$

shows $(g \ i \ j \models_i \text{ifinite iimp rbox } (\text{fin } \$q)) \longleftrightarrow$
 $(\exists k. j = (\text{enat } k)) \longrightarrow (\forall k. j \leq k \wedge k \leq \text{nlength } g \longrightarrow q \in (\text{nnth } g \ k))$

using *assms*

by (*auto simp add: rbox-d-def rdiamond-d-def inot-defs ittrue-defs iand-defs ifinite-defs*)
fin-d-def diamond-d-def)

lemma *hor-defs*:

assumes $i \leq j$

$j \leq \text{nlength } g$

shows $(g \ i \ j \models_i (\text{hor } q)) \longleftrightarrow$
 $(\forall k. k \leq \text{nlength } g \longrightarrow q \in (\text{nnth } g \ k))$

using *assms*

using *atom-box-defs hor-def iand-defs atom-lbox-defs atom-rbox-defs*

by *simp*

(meson dual-order.trans enat-ile enat-ord-simps(1) nle-le)

lemma *ediamond-defs*:

assumes $i \leq j$

$j \leq \text{nlength } g$

shows $(g \ i \ j \models_i \text{ediamond } f) = (\exists k. i < k \wedge \text{enat } k \leq j \wedge (g \ k \ j \models_i f))$

using *assms*

apply (*simp add: ediamond-d-def more-d-def inot-defs*)

using *dual-order.strict-iff-order* **by** *blast*

lemma *wdiamond-defs*:

assumes $i \leq j$

$j \leq \text{nlength } g$

shows $(g \ i \ j \models_i \text{wdiamond } f) = (\exists k. k < i \wedge (g \ k \ j \models_i f))$

using *assms*

apply (*simp add: wdiamond-d-def more-d-def inot-defs*)

using *order.order-iff-strict* **by** *blast*

lemma *sdiamond-defs*:

assumes $i \leq j$

$j \leq \text{nlength } g$

shows $(g \ i \ j \models_i \text{sdiamond } f) = (\exists k. i \leq k \wedge \text{enat } k < j \wedge (g \ i \ (\text{enat } k) \models_i f))$

using *assms*

apply (*simp add: sdiamond-d-def more-d-def inot-defs*)
by (*metis linorder-neq-iff order-le-imp-less-or-eq order-less-imp-le*)

lemma *ndiamond-defs*:

assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{ndiamond } f) =$
 $(j < \infty \wedge (\exists k. j < k \wedge k \leq \text{nlength } g \wedge (g \ i \ k \models_i f)))$
using *assms*
apply (*simp add: ndiamond-d-def more-d-def inot-defs*)
by (*metis linorder-neq-iff order-le-imp-less-or-eq order-less-imp-le the-enat.simps*)

lemma *ewdiamond-defs*:

assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{ewdiamond } f) =$
 $(\exists k. \text{enat } k \leq j \wedge (g \ k \ j \models_i f))$
using *assms*
apply (*simp add: ewdiamond-d-def ediamond-defs wdiamond-defs ior-defs*)
by (*metis enat-ord-simps(2) linorder-less-linear order-less-imp-not-less order-less-le-subst2*
verit-comp-simplify1(3))

lemma *nsdiamond-defs*:

assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{nsdiamond } f) =$
 $(\exists k. i \leq k \wedge k \leq \text{nlength } g \wedge (g \ i \ k \models_i f))$
using *assms*
apply (*auto simp add: ior-defs nsdiamond-d-def ndiamond-defs sdiamond-defs*)
apply (*meson dual-order.trans enat-ord-simps(1) order-less-imp-le*)
apply (*metis assms(1) dual-order.trans order-less-imp-le*)
apply (*metis enat-ord-simps(1) enat-ord-simps(4) ifinite-defs inf-d-def inf-defs inot-defs*)
by (*metis enat-ord-simps(1) ifinite-defs inf-d-def inf-defs inot-defs le-iff-add*
not-less-iff-gr-or-eq order-neq-le-trans plus-eq-infty-iff-enat)

lemma *ebox-defs*:

assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{ebox } f) = (\forall k. i < k \wedge \text{enat } k \leq j \longrightarrow (g \ k \ j \models_i f))$
using *assms ediamond-defs ebox-d-def inot-defs* **by** *auto*

lemma *ebox-ifalse-defs*:

assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{ebox ifalse}) = (i = j)$
using *assms ebox-defs[of i j g ifalse]*
by (*metis Suc-ile-eq antisym-conv2 enat-ord-simps(1) leD lessI semantics-cdt.simps(1)*)

lemma *wbox-defs*:

assumes $i \leq j$

$j \leq \text{nlength } g$

shows $(g \ i \ j \models_i \text{wbox } f) = (\forall k. k < i \longrightarrow (g \ k \ j \models_i f))$

using *assms wdiamond-defs wbox-d-def inot-defs* **by** *auto*

lemma *lex-defs*:

assumes $i \leq j$

$j \leq \text{nlength } g$

shows $(g \ i \ j \models_i \text{lex}) = (i=0)$

using *assms using lex-d-def wbox-defs*

by (*metis antisym-conv1 le-add1 le-add-same-cancel1 less-nat-zero-code semantics-cdt.simps(1)*)

lemma *sbox-defs*:

assumes $i \leq j$

$j \leq \text{nlength } g$

shows $(g \ i \ j \models_i \text{sbox } f) = (\forall k. i \leq k \wedge \text{enat } k < j \longrightarrow (g \ i \ (\text{enat } k) \models_i f))$

using *assms sdiamond-defs sbox-d-def inot-defs* **by** *auto*

lemma *sbox-ifalse-defs*:

assumes $i \leq j$

$j \leq \text{nlength } g$

shows $(g \ i \ j \models_i \text{sbox ifalse}) = (i = j)$

using *assms sbox-defs[of i j g ifalse]*

by *fastforce*

lemma *nbox-defs*:

assumes $i \leq j$

$j \leq \text{nlength } g$

shows $(g \ i \ j \models_i \text{nbox } f) =$

$(j < \infty \longrightarrow (\forall k. j < k \wedge k \leq \text{nlength } g \longrightarrow (g \ i \ k \models_i f)))$

using *assms ndiamond-defs nbox-d-def inot-defs* **by** *auto*

lemma *rex-defs*:

assumes $i \leq j$

$j \leq \text{nlength } g$

shows $(g \ i \ j \models_i \text{rex}) = (j = \text{nlength } g)$

using *assms*

by (*metis enat-ord-simps(4) rex-d-def linorder-not-le nbox-defs order.order-iff-strict semantics-cdt.simps(1)*)

lemma *ewbox-defs*:

assumes $i \leq j$

$j \leq \text{nlength } g$

shows $(g \ i \ j \models_i \text{ewbox } f) =$

$(\forall k. \text{enat } k \leq j \longrightarrow (g \ k \ j \models_i f))$

using *assms ewdiamond-defs ewbox-d-def inot-defs* **by** *auto*

lemma *ebox-ifalse-iand-lex-defs*:

assumes $i \leq j$

$j \leq \text{nlength } g$

shows $(g \ i \ j \models_i \text{ebox ifalse iand lex}) = (i = 0 \wedge i = j)$
using *assms ebox-ifalse-defs iand-defs lex-d-def lex-defs zero-enat-def* **by** *auto*

lemma *nsbox-defs*:

assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{nsbox } f) =$
 $(\ (\forall k. \ i \leq k \wedge k \leq \text{nlength } g \longrightarrow (g \ i \ k \models_i f)) \) \)$
using *assms nsdiamond-defs nsbox-d-def inot-defs* **by** *auto*

lemma *rex-iand-sbox-ifalse-defs*:

assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{rex iand sbox ifalse}) = (i = j \wedge j = \text{nlength } g)$
using *assms iand-defs rex-d-def rex-defs sbox-ifalse-defs* **by** *auto*

lemma *nbox-ewbox-inot-defs*:

assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{nbox (ewbox inot } f)) =$
 $(j < \infty \longrightarrow$
 $(\ (\forall k. \ j < k \wedge k \leq \text{nlength } g \longrightarrow$
 $(\ (\forall k1. \ \text{enat } k1 \leq k \longrightarrow \neg(g \ k1 \ k \models_i f))$
 $) \)$
using *assms ewbox-defs inot-defs nbox-defs*[*of i j g (ewbox inot f)*]
apply *simp*
by (*meson order.order-iff-strict order-trans*)

lemma *sbox-ewbox-inot-defs*:

assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{sbox (ewbox inot } f)) =$
 $(\ (\forall k. \ i \leq k \wedge \text{enat } k < j \longrightarrow$
 $(\ (\forall k1. \ \text{enat } k1 \leq (\text{enat } k) \longrightarrow \neg(g \ k1 \ (\text{enat } k) \models_i f))$
 $) \)$
using *assms sbox-defs*[*of i j g (ewbox inot f)*]
using *ewbox-defs inot-defs* **by** *auto*

lemma *ahor-defs*:

assumes $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i \text{ahor } f) =$
 $(\ (\forall k. \ \text{enat } k \leq j \longrightarrow (g \ k \ j \models_i f)) \) \ \wedge$
 $(j < \infty \longrightarrow$
 $(\ (\forall k. \ j < k \wedge k \leq \text{nlength } g \longrightarrow$
 $(\ (\forall k1. \ \text{enat } k1 \leq k \longrightarrow \neg(g \ k1 \ k \models_i f))$
 $) \) \) \ \wedge$
 $(\ (\forall k. \ i \leq k \wedge \text{enat } k < j \longrightarrow$
 $(\ (\forall k1. \ \text{enat } k1 \leq (\text{enat } k) \longrightarrow \neg(g \ k1 \ (\text{enat } k) \models_i f))$
 $) \)$

```

)
)
using assms ahor-d-def ewbox-defs nbox-ewbox-inot-defs sbbox-ewbox-inot-defs
using iand-defs by presburger

```

lemma *ahor-defs-alt*:

```

assumes  $i \leq j$ 
 $j \leq \text{nlength } g$ 
shows  $(g \ i \ j \models_i \text{ahor } f) =$ 
 $(\forall k \ k1 . i \leq k \wedge \text{enat } k1 \leq k \wedge k \leq \text{nlength } g \longrightarrow (k = j \longleftrightarrow (g \ k1 \ k \models_i f)))$ 
using assms ahor-defs apply auto
apply (metis enat-ord-simps(1) enat-ord-simps(4) less-infinityE)
apply (metis enat-ord-simps(1) enat-ord-simps(4) leI less-infinityE nle-le)
apply (metis assms(1) dual-order.strict-trans2 less-le-not-le)
by (metis dual-order.trans enat-ord-simps(1) linorder-neq-iff order-less-imp-le)

```

5.2.2 Validity

definition *valid* :: *cdt* \Rightarrow *bool* $((\vdash_i \ -) \ 5)$
where $(\vdash_i f) = (\forall \ g \ i \ j. (\text{enat } i) \leq j \wedge j \leq \text{nlength } g \longrightarrow (g \ i \ j \models_i f))$

lemma *cdt-valid [simp]* :

```

 $(\vdash_i f) = (\forall \ g \ i \ j. (\text{enat } i) \leq j \wedge j \leq \text{nlength } g \longrightarrow (g \ i \ j \models_i f))$ 
by (simp add: valid-def)

```

lemma *cdt-ieq*:

```

 $(\vdash_i f1 \text{ieqv } f2) = (\forall \ g \ i \ j. i \leq j \wedge j \leq \text{nlength } g \longrightarrow ((g \ i \ j \models_i f1) = (g \ i \ j \models_i f2)))$ 
using ieqv-defs valid-def by presburger

```

lemma *more-iand-help*:

```

assumes  $i \leq j$ 
 $j \leq \text{nlength } g$ 
shows  $g \ i \ j \models_i (\text{more } iand \ ifinite) \ iimp (\text{skip } cee \ ittrue) \ iand (ittrue \ cee \ skip)$ 
using assms skip-cee-ittrue-defs ittrue-cee-skip-defs iand-defs more-defs ifinite-defs
apply simp
by force

```

lemma *Venema-discrete*:

```

 $\vdash_i (\text{more } iand \ ifinite) \ iimp (\text{skip } cee \ ittrue) \ iand (ittrue \ cee \ skip)$ 
using more-iand-help valid-def by presburger

```

lemma *Venema-B1a*:

```

 $\vdash_i (f1 \ ior \ f2) \ cee \ g \text{ieqv } (f1 \ cee \ g \ ior \ f2 \ cee \ g)$ 
unfolding valid-def
by (auto simp add: ieqv-defs ior-defs)

```

lemma *Venema-B1b*:

$\vdash_i f \text{ cee } (g1 \text{ ior } g2) \text{ ieqv } (f \text{ cee } g1 \text{ ior } f \text{ cee } g2)$
unfolding *valid-def*
by (*auto simp add: ieqv-defs ior-defs*)

lemma *Venema-B2a*:
 $\vdash_i (f1 \text{ ior } f2) \text{ tee } g \text{ ieqv } (f1 \text{ tee } g \text{ ior } f2 \text{ tee } g)$
unfolding *valid-def*
by (*auto simp add: ieqv-defs ior-defs*)

lemma *Venema-B2b*:
 $\vdash_i f \text{ tee } (g1 \text{ ior } g2) \text{ ieqv } (f \text{ tee } g1 \text{ ior } f \text{ tee } g2)$
unfolding *valid-def*
by (*auto simp add: ieqv-defs ior-defs*)

lemma *Venema-B3a*:
 $\vdash_i (f1 \text{ ior } f2) \text{ dee } g \text{ ieqv } (f1 \text{ dee } g \text{ ior } f2 \text{ dee } g)$
unfolding *valid-def*
by (*auto simp add: ieqv-defs ior-defs*)

lemma *Venema-B3b*:
 $\vdash_i f \text{ dee } (g1 \text{ ior } g2) \text{ ieqv } (f \text{ dee } g1 \text{ ior } f \text{ dee } g2)$
unfolding *valid-def*
by (*auto simp add: ieqv-defs ior-defs*)

lemma *Venema-C1*:
 $\vdash_i (\text{inot}(f \text{ tee } g)) \text{ cee } f \text{ iand } \text{ifinite } \text{iimp } \text{inot } g$
by (*auto simp add: inot-defs iand-defs ifinite-defs*)

lemma *Venema-C2*:
 $\vdash_i (\text{inot}(f \text{ tee } g)) \text{ dee } g \text{ iimp } \text{inot } (f)$
by (*auto simp add: inot-defs iand-defs inf-d-def ifinite-defs*)

lemma *Venema-C3*:
 $\vdash_i f \text{ tee } (\text{inot } (g \text{ cee } f)) \text{ iimp } \text{inot } g$
by (*auto simp add: inot-defs*)

lemma *Venema-D*:
 $\vdash_i \text{more} \text{ cee } \text{itrue} \text{ ieqv } \text{more}$
by (*auto simp add: ieqv-defs more-defs inot-defs itrue-defs skip-defs*)
 (*metis Suc-ile-eq Suc-n-not-le-n linorder-linear order.strict-iff-order*)

lemma *Venema-E1*:
 $\vdash_i \text{empty} \text{ cee } f \text{ ieqv } f$
by (*auto simp add: ieqv-defs*)

lemma *Venema-E2*:
 $\vdash_i \text{empty} \text{ tee } f \text{ ieqv } (f \text{ iand } \text{ifinite})$
by (*auto simp add: ifinite-defs ieqv-defs iand-defs*)

lemma *Venema-E3*:

$\vdash_i f \text{ tee empty } \text{ieqv} (f \text{ iand empty})$
by (auto simp add: *ieqv-defs iand-defs ifinite-defs*)

lemma *Venema-E4*:
 $\vdash_i \text{empty } \text{dee } f \text{ieqv } f$
by (auto simp add: *ieqv-defs*)

lemma *Venema-E5*:
 $\vdash_i f \text{ cee empty } \text{ieqv} (f \text{ iand ifinite})$
by (auto simp add: *ifinite-defs ieqv-defs iand-defs*)

lemma *Venema-F*:
 $\vdash_i ((\text{empty } \text{iand } f) \text{ cee } \text{itrue } \text{iand} ((\text{empty } \text{iand } g) \text{ cee } \text{itrue}) \text{ cee } \text{itrue}) \text{ iimp} (\text{empty } \text{iand } g) \text{ cee } \text{itrue}$
by (auto simp add: *iand-defs itrue-defs*)

lemma *cee-assoc-1*:
assumes $(g \text{ i } j \models_i f1 \text{ cee } (f2 \text{ cee } f3))$
shows $(g \text{ i } j \models_i (f1 \text{ cee } f2) \text{ cee } f3)$
using *assms dual-order.trans* **by** auto

lemma *cee-assoc-2*:
assumes $(g \text{ i } j \models_i (f1 \text{ cee } f2) \text{ cee } f3)$
shows $(g \text{ i } j \models_i f1 \text{ cee } (f2 \text{ cee } f3))$
using *assms*
by (*meson order.trans semantics-cdt.simps(5)*)

lemma *Venema-G1*:
 $\vdash_i f \text{ cee } (g \text{ cee } h) \text{ieqv} (f \text{ cee } g) \text{ cee } h$
unfolding *valid-def ieqv-defs*
using *cee-assoc-1 cee-assoc-2*
by *blast*

lemma *Venema-G2a*:
 $\vdash_i f \text{ tee } (g \text{ cee } h) \text{ iimp} (g \text{ cee } (f \text{ tee } h) \text{ ior} (h \text{ tee } f) \text{ tee } g)$
by (*simp add: ior-defs*)
(*metis nle-le order-trans the-enat.simps*)

lemma *Venema-G2b*:
 $\vdash_i (g \text{ cee } (f \text{ tee } h) \text{ ior} (h \text{ tee } f) \text{ tee } g) \text{ iimp } f \text{ tee } (g \text{ cee } h)$
by (*simp add: ior-defs*)
(*metis enat-ord-simps(1) order-trans the-enat.simps*)

lemma *Venema-G3*:
 $\vdash_i f \text{ cee } (g \text{ tee } h) \text{ iimp } g \text{ tee } (f \text{ cee } h)$
by (*auto*)
(*meson enat-ord-simps(1) order.trans*)

lemma *Venema-G4*:
 $\vdash_i ((f \text{ iand ifinite}) \text{ tee } g) \text{ cee } h \text{ iimp} ((h \text{ dee } f) \text{ tee } g \text{ ior } g \text{ cee } (f \text{ dee } h))$
unfolding *valid-def*

apply (*auto simp add: ior-defs iand-defs ifinite-defs*)
using *ifinite-defs* **apply** *fastforce*
by (*metis dual-order.trans enat-ile enat-ord-simps(1) nle-le the-enat.simps*)

lemma *Venema-MP*:
assumes $\vdash_i f$
 $\vdash_i f \text{ iimp } g$
shows $\vdash_i g$
using *assms* **unfolding** *valid-def* **by** *simp*

lemma *Venema-Gen-1a*:
assumes $\vdash_i f$
shows $\vdash_i \text{inot}((\text{inot } f) \text{ cee } g)$
using *assms* **unfolding** *valid-def*
apply (*simp add: inot-defs*)
by *force*

lemma *Venema-Gen-1b*:
assumes $\vdash_i g$
shows $\vdash_i \text{inot}(g \text{ cee } (\text{inot } g))$
using *assms* **unfolding** *valid-def*
by (*simp add: inot-defs*)

lemma *Venema-Gen-2a*:
assumes $\vdash_i f$
shows $\vdash_i \text{inot}((\text{inot } f) \text{ tee } g)$
using *assms* **unfolding** *valid-def*
apply (*simp add: inot-defs*)
by *force*

lemma *Venema-Gen-2b*:
assumes $\vdash_i g$
shows $\vdash_i \text{inot}(f \text{ tee } (\text{inot } g))$
using *assms* **unfolding** *valid-def*
apply (*simp add: inot-defs*)
by (*meson dual-order.trans*)

lemma *Venema-Gen-3a*:
assumes $\vdash_i f$
shows $\vdash_i \text{inot}((\text{inot } f) \text{ dee } g)$
using *assms* **unfolding** *valid-def*
apply (*simp add: inot-defs*)
by *force*

lemma *Venema-Gen-3b*:
assumes $\vdash_i g$
shows $\vdash_i \text{inot}(f \text{ dee } (\text{inot } g))$
using *assms* **unfolding** *valid-def*
apply (*simp add: inot-defs*)

by (meson enat-ord-simps(1) order-trans)

lemma hor-rule:

assumes $q \notin \text{fvars } f$
 $\forall g \ i \ j.$
 $(i \leq j \wedge j \leq \text{nlength } g \wedge$
 $(\forall k. k \leq \text{nlength } g \longrightarrow q \in (\text{nnth } g \ k)))$
 $\longrightarrow (g \ i \ j \models_i f)$
 $i \leq j$
 $j \leq \text{nlength } g$
shows $(g \ i \ j \models_i f)$
proof –
have 1: $\exists ul. \text{nlength } ul = \text{nlength } g \wedge (\forall i. i \leq \text{nlength } g \longrightarrow (\text{nnth } ul \ i) = \text{True})$
by (metis inj-enat nlength-nmap nnth-nmap)
obtain ug **where** 4: $\text{nlength } ug = \text{nlength } g \wedge (\forall i. i \leq \text{nlength } g \longrightarrow (\text{nnth } ug \ i) = \text{True})$
using 1 **by** blast
have 8: $(g \ i \ j \models_i f) \longleftrightarrow ((\text{upd } g \ q \ ug) \ i \ j \models_i f)$
using not-fvar-upd[of q f ug g i j]
using 4 **assms** **by** blast
show ?thesis **using** assms
by (simp add: 4 8 nlength-upd nnth-upd)
qed

lemma Venema-Consistency:

assumes $\vdash_i \text{hor}(q) \text{ iimp } f$
 $q \notin \text{fvars } f$
shows $\vdash_i f$
using hor-rule[of q f] hor-defs assms **by** force

definition valid-finite :: $\text{cdt} \Rightarrow \text{bool}$ ($(\vdash_f -) \ 5$)

where $(\vdash_f f) = (\forall g \ i \ j. (\text{enat } i) \leq j \wedge j \leq \text{nlength } g \wedge \text{nfinite } g \longrightarrow (g \ i \ j \models_i f))$

lemma valid-finite:

assumes $\vdash_i f$
shows $\vdash_f f$
using assms valid-finite-def
using valid-def **by** blast

primrec rva :: $\text{cdt} \Rightarrow \text{cdt}$ ((rva -) [85] 85)

where

$\text{rva ifalse} = \text{ifalse}$
 $\mid \text{rva } (\$p) = (\text{fin } \$p)$
 $\mid \text{rva } (f \text{ iimp } g) = ((\text{rva } f) \text{ iimp } (\text{rva } g))$
 $\mid \text{rva empty} = \text{empty}$
 $\mid \text{rva } (f \text{ cee } g) = ((\text{rva } g) \text{ cee } (\text{rva } f))$
 $\mid \text{rva } (f \text{ tee } g) = ((\text{rva } f) \text{ dee } (\text{rva } g))$
 $\mid \text{rva } (f \text{ dee } g) = ((\text{rva } f) \text{ tee } (\text{rva } g))$

```

lemma rva-sem:
  assumes nfinite g
    i ≤ j
    j ≤ nlength g
  shows (g i j ⊨i (rva f)) = ( (nrev g) (the-enat(nlength g) - j) (nlength g - i) ⊨i f)
  using assms
  proof (induction f arbitrary: g i j)
  case false-d
  then show ?case by simp
  next
  case (atom-d x)
  then show ?case apply (simp add: fin-defs)
  by (metis diff-diff-cancel diff-le-self enat-ord-simps(1) nfinite-conv-nlength-enat nnth-nrev the-enat.simps)
  next
  case (iimp-d f1 f2)
  then show ?case by simp
  next
  case empty-d
  then show ?case apply (simp )
  by (metis diff-diff-cancel dual-order.trans enat-ord-simps(1) idiff-enat-enat nfinite-conv-nlength-enat the-enat.simps)
  next
  case (c-d f1 f2)
  then show ?case
  proof -
    have c1: (g i j ⊨i (rva (f1 cee f2))) = (g i j ⊨i (rva f2) cee (rva f1))
      by simp
    have c2: (g i j ⊨i (rva f2) cee (rva f1)) =
      (∃ k ≥ i. k ≤ j ∧ (g i (enat k) ⊨i rva f2) ∧ (g k (enat j) ⊨i rva f1))
      by simp
    have c3: ((nrev g) (the-enat (nlength g) - j) (nlength g - enat i) ⊨i f1 cee f2 ) =
      (∃ k ≥ the-enat (nlength g) - j. enat k ≤ nlength g - enat i ∧
      ((nrev g) (the-enat (nlength g) - j) (enat k) ⊨i f1) ∧ ((nrev g) k (nlength g - enat i) ⊨i f2))
      by simp
    have c4: (∃ k ≥ i. k ≤ j ∧ (g i (enat k) ⊨i rva f2) ∧ (g k (enat j) ⊨i rva f1)) ⇒
      (∃ k ≥ the-enat (nlength g) - j. enat k ≤ nlength g - enat i ∧
      ((nrev g) (the-enat (nlength g) - j) (enat k) ⊨i f1) ∧ ((nrev g) k (nlength g - enat i) ⊨i f2))
    proof -
      assume ca0: (∃ k ≥ i. k ≤ j ∧ (g i (enat k) ⊨i rva f2) ∧ (g k (enat j) ⊨i rva f1))
      show (∃ k ≥ the-enat (nlength g) - j. enat k ≤ nlength g - enat i ∧
      ((nrev g) (the-enat (nlength g) - j) (enat k) ⊨i f1) ∧ ((nrev g) k (nlength g - enat i) ⊨i f2))
      proof -
        obtain k where c100: k ≥ i ∧ k ≤ j ∧ (g i (enat k) ⊨i rva f2) ∧ (g k (enat j) ⊨i rva f1)
        using ca0 by blast
        have c101: ( (nrev g) (the-enat(nlength g) - k) (nlength g - i) ⊨i f2)
          by (meson c100 c-d.IH(2) c-d.premis(1) c-d.premis(3) enat-ord-simps(1) order-trans)
        have c102: ( (nrev g) (the-enat(nlength g) - j) (nlength g - k) ⊨i f1)
          using c100 c-d.IH(1) c-d.premis(1) c-d.premis(3) by blast
      proof
        show ?thesis
      end
    end
  end
end

```

by (metis c100 c101 c102 c-d.premis(1) diff-le-mono2 enat-ord-simps(1) idiff-enat-enat nfinite-conv-nlength-enat the-enat.simps)
 qed
 qed
 have c5: $(\exists k \geq \text{the-enat } (\text{nlength } g) - j. \text{ enat } k \leq \text{nlength } g - \text{ enat } i \wedge ((\text{nrev } g) (\text{the-enat } (\text{nlength } g) - j) (\text{ enat } k) \models_i f1) \wedge ((\text{nrev } g) k (\text{nlength } g - \text{ enat } i) \models_i f2))$
 \Rightarrow
 $(\exists k \geq i. k \leq j \wedge (g \ i \ (\text{ enat } k) \models_i \text{ rva } f2) \wedge (g \ k \ (\text{ enat } j) \models_i \text{ rva } f1))$
 proof -
 assume ca0: $(\exists k \geq \text{the-enat } (\text{nlength } g) - j. \text{ enat } k \leq \text{nlength } g - \text{ enat } i \wedge ((\text{nrev } g) (\text{the-enat } (\text{nlength } g) - j) (\text{ enat } k) \models_i f1) \wedge ((\text{nrev } g) k (\text{nlength } g - \text{ enat } i) \models_i f2))$
 show $(\exists k \geq i. k \leq j \wedge (g \ i \ (\text{ enat } k) \models_i \text{ rva } f2) \wedge (g \ k \ (\text{ enat } j) \models_i \text{ rva } f1))$
 proof -
 obtain k where c200: $k \geq \text{the-enat } (\text{nlength } g) - j \wedge \text{ enat } k \leq \text{nlength } g - \text{ enat } i \wedge ((\text{nrev } g) (\text{the-enat } (\text{nlength } g) - j) (\text{ enat } k) \models_i f1) \wedge ((\text{nrev } g) k (\text{nlength } g - \text{ enat } i) \models_i f2)$
 using ca0 by blast
 have c201: $i \leq \text{the-enat } (\text{nlength } g) - k$
 by (metis c200 c-d.premis(1) c-d.premis(2) c-d.premis(3) diff-diff-cancel diff-le-mono2 dual-order.trans enat-ord-simps(1) idiff-enat-enat nfinite-conv-nlength-enat the-enat.simps)
 have c2011: $\text{the-enat } (\text{nlength } g) - (\text{the-enat } (\text{nlength } g) - k) = k$
 by (metis c200 c201 c-d.premis(1) diff-diff-cancel diff-is-0-eq enat.distinct(2) enat-ord-simps(1) enat-the-enat idiff-0-right nfinite-conv-nlength-enat nle-le zero-enat-def)
 have c202: $\text{nlength } g - \text{ enat } (\text{the-enat } (\text{nlength } g) - k) = (\text{ enat } k)$
 by (metis c200 c-d.premis(1) diff-diff-cancel diff-le-self dual-order.trans enat-ord-simps(1) idiff-enat-enat nfinite-conv-nlength-enat the-enat.simps)
 have c203: $\text{the-enat } (\text{nlength } g) - k \leq j$
 using c200 by linarith
 have c204: $(g \ (\text{the-enat } (\text{nlength } g) - k) \ (\text{ enat } j) \models_i \text{ rva } f1)$ using c-d.IH(1)[of $g \ (\text{the-enat } (\text{nlength } g) - k) \ j$]
 using c200 c202 c203 c-d.premis(1) c-d.premis(3) by presburger
 have c205: $(g \ i \ (\text{ enat } (\text{the-enat } (\text{nlength } g) - k)) \models_i (\text{ rva } f2))$ using c-d.IH(2)[of $g \ i \ (\text{the-enat } (\text{nlength } g) - k)$]
 by (metis c200 c201 c2011 c203 c-d.premis(1) c-d.premis(3) enat-ord-simps(1) order.trans)
 show ?thesis
 using c201 c203 c204 c205 by blast
 qed
 qed
 show ?thesis using c1 c2 c3 c4 c5 by blast
 qed
 next
 case (d-d f1 f2)
 then show ?case
 proof -
 have d1: $(g \ i \ j \models_i \text{ rva } (f1 \text{ dee } f2)) =$
 $(\exists k \geq \text{ enat } j. k \leq \text{nlength } g \wedge (g \ j \ k \models_i \text{ rva } f1) \wedge (g \ i \ k \models_i \text{ rva } f2))$
 by simp
 have d2: $((\text{nrev } g) (\text{the-enat } (\text{nlength } g) - j) (\text{nlength } g - \text{ enat } i) \models_i f1 \text{ dee } f2) =$
 $(\exists k \leq \text{the-enat } (\text{nlength } g) - j. ((\text{nrev } g) k (\text{ enat } (\text{the-enat } (\text{nlength } g) - j)) \models_i f1) \wedge ((\text{nrev } g) k (\text{nlength } g - \text{ enat } i) \models_i f2))$
 by simp

have $d3$: $(\exists k \geq \text{enat } j. k \leq \text{nlength } g \wedge (g \ j \ k \models_i \text{rva } f1) \wedge (g \ i \ k \models_i \text{rva } f2)) \implies$
 $(\exists k \leq \text{the-enat } (\text{nlength } g) - j. ((\text{nrev } g) \ k \ (\text{enat } (\text{the-enat } (\text{nlength } g) - j)) \models_i f1) \wedge$
 $((\text{nrev } g) \ k \ (\text{nlength } g - \text{enat } i) \models_i f2))$
proof –
assume $da0$: $(\exists k \geq \text{enat } j. k \leq \text{nlength } g \wedge (g \ j \ k \models_i \text{rva } f1) \wedge (g \ i \ k \models_i \text{rva } f2))$
show $(\exists k \leq \text{the-enat } (\text{nlength } g) - j. ((\text{nrev } g) \ k \ (\text{enat } (\text{the-enat } (\text{nlength } g) - j)) \models_i f1) \wedge$
 $((\text{nrev } g) \ k \ (\text{nlength } g - \text{enat } i) \models_i f2))$
proof –
obtain k **where** $d100$: $k \geq \text{enat } j \wedge k \leq \text{nlength } g \wedge (g \ j \ k \models_i \text{rva } f1) \wedge (g \ i \ k \models_i \text{rva } f2)$
using $da0$ **by** *blast*
have $d103$: $(\text{the-enat}(\text{nlength } g - k)) \leq \text{the-enat } (\text{nlength } g) - j$
by $(\text{metis } d100 \text{ d-d.premis}(1) \text{ diff-le-mono2 } \text{enat-ile } \text{enat-ord-simps}(1) \text{ idiff-enat-enat } \text{nfinite-conv-nlength-enat}$
 $\text{the-enat.simps})$
have $d101$: $((\text{nrev } g) \ (\text{the-enat}(\text{nlength } g - k)) \ (\text{enat } (\text{the-enat } (\text{nlength } g) - j)) \models_i f1)$
using $d\text{-d.IH}(1)[\text{of } g \ j \ -] \ d100$
by $(\text{metis } d\text{-d.premis}(1) \text{ enat-ile } \text{enat-ord-simps}(1) \text{ idiff-enat-enat } \text{nfinite-conv-nlength-enat}$
 $\text{the-enat.simps})$
have $d102$: $((\text{nrev } g) \ (\text{the-enat}(\text{nlength } g - k)) \ (\text{nlength } g - \text{enat } i) \models_i f2)$
using $d100 \text{ d-d.IH}(2) \text{ d-d.premis}(1) \text{ d-d.premis}(2) \text{ enat-ile } \text{nfinite-nlength-enat}$ **by** *fastforce*
show *?thesis*
using $d101 \ d102 \ d103$ **by** *blast*
qed
qed
have $d4$: $(\exists k \leq \text{the-enat } (\text{nlength } g) - j. ((\text{nrev } g) \ k \ (\text{enat } (\text{the-enat } (\text{nlength } g) - j)) \models_i f1) \wedge$
 $((\text{nrev } g) \ k \ (\text{nlength } g - \text{enat } i) \models_i f2)) \implies$
 $(\exists k \geq \text{enat } j. k \leq \text{nlength } g \wedge (g \ j \ k \models_i \text{rva } f1) \wedge (g \ i \ k \models_i \text{rva } f2))$
proof –
assume $da0$: $(\exists k \leq \text{the-enat } (\text{nlength } g) - j. ((\text{nrev } g) \ k \ (\text{enat } (\text{the-enat } (\text{nlength } g) - j)) \models_i f1) \wedge$
 $((\text{nrev } g) \ k \ (\text{nlength } g - \text{enat } i) \models_i f2))$
show $(\exists k \geq \text{enat } j. k \leq \text{nlength } g \wedge (g \ j \ k \models_i \text{rva } f1) \wedge (g \ i \ k \models_i \text{rva } f2))$
proof –
obtain k **where** $d200$: $k \leq \text{the-enat } (\text{nlength } g) - j \wedge ((\text{nrev } g) \ k \ (\text{enat } (\text{the-enat } (\text{nlength } g) - j)) \models_i$
 $f1) \wedge$
 $((\text{nrev } g) \ k \ (\text{nlength } g - \text{enat } i) \models_i f2)$
using $da0$ **by** *blast*
have $d201$: $j \leq \text{the-enat } (\text{nlength } g) - k$
using $d200 \text{ d-d.premis}(1) \text{ d-d.premis}(3) \text{ nfinite-nlength-enat}$ **by** *fastforce*
have $d202$: $\text{enat } (\text{the-enat } (\text{nlength } g) - k) \leq \text{nlength } g$
by $(\text{metis } \text{diff-le-self } \text{enat-ord-simps}(1) \text{ enat-ord-simps}(3) \text{ enat-the-enat})$
have $d203$: $\text{the-enat } (\text{nlength } g) - (\text{the-enat } (\text{nlength } g) - k) = k$
by $(\text{meson } d200 \text{ diff-diff-cancel } \text{diff-le-self } \text{dual-order.trans})$
have $d204$: $(g \ j \ (\text{enat } (\text{the-enat}(\text{nlength } g) - k)) \models_i \text{rva } f1)$
using $d\text{-d.IH}(1)[\text{of } g \ j \ (\text{the-enat}(\text{nlength } g) - k)]$
by $(\text{metis } d200 \ d201 \ d202 \text{ d-d.premis}(1) \text{ diff-diff-cancel } \text{diff-is-0-eq'} \text{ enat.distinct}(1) \text{ enat-the-enat}$
 $\text{idiff-enat-enat } \text{nfinite-conv-nlength-enat } \text{nle-le})$
have $d205$: $(g \ i \ (\text{nlength } g - k) \models_i \text{rva } f2)$
using $d\text{-d.IH}(2)[\text{of } g \ i \ (\text{the-enat}(\text{nlength } g) - k)]$
by $(\text{metis } d200 \ d201 \ d202 \ d203 \text{ d-d.premis}(1) \text{ d-d.premis}(2) \text{ dual-order.trans } \text{enat.distinct}(1) \text{ enat-the-enat}$
 $\text{idiff-enat-enat } \text{nfinite-conv-nlength-enat})$
show *?thesis*

```

    by (metis d201 d202 d204 d205 d-d.premis(1) enat-ord-simps(1) idiff-enat-enat nfinite-conv-nlength-enat
the-enat.simps)
  qed
  qed
  show ?thesis using d1 d2 d3 d4 by blast
  qed
next
case (t-d f1 f2)
then show ?case
proof -
  have t1: (g i j  $\models_i$  rva ( f1 tee f2 )) =
    ( $\exists k \leq i$ . (g k (enat i)  $\models_i$  rva f1)  $\wedge$  (g k (enat j)  $\models_i$  rva f2))
  by simp
  have t2: ((nrev g) (the-enat (nlength g) - j) (nlength g - enat i)  $\models_i$  f1 tee f2) =
    ( $\exists k \geq \text{nlength } g - \text{enat } i$ .  $k \leq \text{nlength } g \wedge ((\text{nrev } g) (\text{the-enat } (\text{nlength } g - \text{enat } i)) k \models_i f1) \wedge$ 
    ((nrev g) (the-enat (nlength g) - j) k  $\models_i$  f2))
  apply simp
  apply auto
  by (metis enat-ord-code(5) enat-the-enat nfinite-conv-nlength-enat t-d.premis(1))

  have t3: ( $\exists k \leq i$ . (g k (enat i)  $\models_i$  rva f1)  $\wedge$  (g k (enat j)  $\models_i$  rva f2))  $\implies$ 
    ( $\exists k \geq \text{nlength } g - \text{enat } i$ .  $k \leq \text{nlength } g \wedge ((\text{nrev } g) (\text{the-enat } (\text{nlength } g - \text{enat } i)) k \models_i f1) \wedge$ 
    ((nrev g) (the-enat (nlength g) - j) k  $\models_i$  f2))
  proof -
    assume ta0: ( $\exists k \leq i$ . (g k (enat i)  $\models_i$  rva f1)  $\wedge$  (g k (enat j)  $\models_i$  rva f2))
    show ( $\exists k \geq \text{nlength } g - \text{enat } i$ .  $k \leq \text{nlength } g \wedge ((\text{nrev } g) (\text{the-enat } (\text{nlength } g - \text{enat } i)) k \models_i f1) \wedge$ 
    ((nrev g) (the-enat (nlength g) - j) k  $\models_i$  f2))
  proof -
    obtain k where t100:  $k \leq i \wedge (g k (\text{enat } i) \models_i \text{rva } f1) \wedge (g k (\text{enat } j) \models_i \text{rva } f2)$ 
    using ta0 by blast
    have t101: ((nrev g) (the-enat (nlength g - enat i)) (nlength g - k)  $\models_i$  f1)
      by (metis enat-ord-simps(1) idiff-enat-enat nfinite-conv-nlength-enat order-subst2 t100 t-d.IH(1)
t-d.premis(1) t-d.premis(2) t-d.premis(3) the-enat.simps)
    have t102: ((nrev g) (the-enat (nlength g) - j) (nlength g - k)  $\models_i$  f2)
      by (meson order.trans t100 t-d.IH(2) t-d.premis(1) t-d.premis(2) t-d.premis(3))
    have t103: (nlength g - k)  $\geq \text{nlength } g - \text{enat } i$  using t100
    by (metis diff-le-mono2 enat-ord-simps(1) idiff-enat-enat nfinite-conv-nlength-enat t-d.premis(1))
    have t104: (nlength g - k)  $\leq \text{nlength } g$  using t100
    by (metis enat.distinct(2) enat-add-sub-same enat-le-plus-same(2) enat-minus-mono1)
    show ?thesis
      using t101 t102 t103 t104 by blast
  qed
  qed
  have t4: ( $\exists k \geq \text{nlength } g - \text{enat } i$ .  $k \leq \text{nlength } g \wedge ((\text{nrev } g) (\text{the-enat } (\text{nlength } g - \text{enat } i)) k \models_i f1) \wedge$ 
    ((nrev g) (the-enat (nlength g) - j) k  $\models_i$  f2))  $\implies$ 
    ( $\exists k \leq i$ . (g k (enat i)  $\models_i$  rva f1)  $\wedge$  (g k (enat j)  $\models_i$  rva f2))
  proof -
    assume ta0: ( $\exists k \geq \text{nlength } g - \text{enat } i$ .  $k \leq \text{nlength } g \wedge ((\text{nrev } g) (\text{the-enat } (\text{nlength } g - \text{enat } i)) k \models_i$ 
f1)  $\wedge$ 
    ((nrev g) (the-enat (nlength g) - j) k  $\models_i$  f2))

```

```

show ( $\exists k \leq i. (g \ k \ (enat \ i) \models_i \ rva \ f1) \wedge (g \ k \ (enat \ j) \models_i \ rva \ f2)$ )
proof -
  obtain  $k$  where  $t200: k \geq nlength \ g - enat \ i \wedge k \leq nlength \ g \wedge ((nrev \ g) \ (the-enat \ (nlength \ g - enat \ i)) \ k \models_i \ f1) \wedge$ 
    ( $(nrev \ g) \ (the-enat \ (nlength \ g) - j) \ k \models_i \ f2$ )
  using  $ta0$  by blast
  have  $t201: (the-enat(nlength \ g) - k) \leq i$  using  $t200$ 
  using  $enat-ile \ nfinite-nlength-enat \ t-d.prem(1)$  by fastforce
  have  $t202: nlength \ g - enat \ (the-enat \ (nlength \ g - k)) = k$ 
  by ( $metis \ add.commute \ enat-add-sub-same \ enat-ord-code(4) \ enat-the-enat \ leD \ le-iff-add \ t200$ )
  have  $t203: the-enat \ (nlength \ g) - i = (the-enat \ (nlength \ g - enat \ i))$ 
  by ( $metis \ enat.distinct(2) \ enat.inject \ enat-the-enat \ idiff-enat-enat \ nfinite-conv-nlength-enat \ t-d.prem(1)$ )
  have  $t2031: ((nrev \ g) \ (the-enat \ (nlength \ g) - i) \ (nlength \ g - enat \ (the-enat \ (nlength \ g - k))) \models_i \ f1)$ 
    using  $t200 \ t202 \ t203$  by presburger
  have  $t2032: nfinite \ g \wedge the-enat \ (nlength \ g - k) \leq i \wedge enat \ i \leq nlength \ g$ 
  by ( $metis \ enat-ile \ enat-ord-simps(1) \ nfinite-conv-nlength-enat \ order.trans \ t201 \ t-d.prem(1) \ t-d.prem(2) \ t-d.prem(3) \ the-enat.simps$ )
  have  $t204: (g \ (the-enat(nlength \ g - k)) \ (enat \ i) \models_i \ rva \ f1)$ 
    using  $t-d.IH(1)[of \ g \ (the-enat(nlength \ g - k)) \ i]$  using  $t200 \ t201 \ t202 \ t2031 \ t2032$  by blast
  have  $t205: (g \ (the-enat(nlength \ g - k)) \ (enat \ j) \models_i \ rva \ f2)$ 
    using  $t-d.IH(2)[of \ g \ (the-enat(nlength \ g - k)) \ j]$ 
    using  $t200 \ t202 \ t2032 \ t-d.prem(2) \ t-d.prem(3)$  by auto
  show ?thesis
  using  $t201 \ t202 \ t203 \ t204 \ t205$ 
  using  $t2032$  by blast
qed
qed
show ?thesis using  $t1 \ t2 \ t3 \ t4$  by blast
qed
qed

```

```

lemma  $rva-atom$ :
  assumes  $nfinite \ g$ 
     $i \leq j$ 
     $j \leq nlength \ g$ 
  shows  $(g \ i \ j \models_i \ (rva \ (\$p))) = (g \ i \ j \models_i \ fin \ (\$p))$ 
  using  $assms$  by simp

```

```

lemma  $rva-fin-atom$ :
  assumes  $nfinite \ g$ 
     $i \leq j$ 
     $j \leq nlength \ g$ 
  shows  $(g \ i \ j \models_i \ (rva \ (fin \ (\$p)))) = (g \ i \ j \models_i \ \$p)$ 
  using  $assms$  unfolding  $fin-d-def \ diamond-d-def \ iand-d-def \ ior-d-def \ inot-d-def \ itrue-d-def$ 
  apply simp
  unfolding  $fin-d-def \ diamond-d-def \ iand-d-def \ ior-d-def \ inot-d-def \ itrue-d-def$ 
  by simp

```

```

lemma  $rva-rva$ :
  assumes  $nfinite \ g$ 

```

```

       $i \leq j$ 
       $j \leq \text{nlength } g$ 
shows  $(g \ i \ j \models_i (rva \ (rva \ f))) = (g \ i \ j \models_i f)$ 
using assms
proof (induction f arbitrary: g i j)
case false-d
then show ?case by simp
next
case (atom-d x)
then show ?case apply simp
using rva-fin-atom
using semantics-cdt.simps(2) by presburger
next
case (iimp-d f1 f2)
then show ?case by simp
next
case empty-d
then show ?case by simp
next
case (c-d f1 f2)
then show ?case apply simp
by (meson dual-order.trans enat-ord-simps(1))
next
case (d-d f1 f2)
then show ?case apply simp
by (meson dual-order.trans enat-ord-simps(1))
next
case (t-d f1 f2)
then show ?case apply simp
by (metis (no-types, lifting) dual-order.trans the-enat.simps)
qed

```

lemma *rva-valid-1:*

assumes $\bigwedge g \ i \ j. \text{ enat } i \leq j \wedge j \leq \text{nlength } g \wedge \text{nfinite } g \implies (g \ i \ j \models_i f)$

$\text{enat } i \leq j$

$\text{enat } j \leq \text{nlength } g$

$\text{nfinite } g$

shows $(g \ i \ j \models_i rva \ f)$

using *assms* **using** *rva-sem[of g i j f]*

by (*metis diff-diff-left diff-le-self enat-ord-simps(1) idiff-enat-enat le-iff-add nfinite-conv-nlength-enat nlength-nrev the-enat.simps*)

lemma *rva-valid-2:*

assumes $\bigwedge g \ i \ j. \text{ enat } i \leq j \wedge j \leq \text{nlength } g \wedge \text{nfinite } g \implies (g \ i \ j \models_i rva \ f)$

$\text{enat } i \leq j$

$\text{enat } j \leq \text{nlength } g$

$\text{nfinite } g$

shows $(g \ i \ j \models_i f)$

```

using assms rva-sem[of ]
by simp (meson assms(2) rva-rva rva-valid-1)

lemma rva-valid:
   $(\vdash_f f) = (\vdash_f \text{rva } f)$ 
unfolding valid-finite-def
by (metis enat-ile nfinite-nlength-enat rva-valid-1 rva-valid-2)

end

```

Chapter 6

NL-ITL

theory *NL-ITL-Deep imports*
Extended-Int NELList-Extras

begin

This theory contains the syntax and semantics of ITL with neighbourhood operators, i.e., NL-ITL[4]. Note, we have both infinite past and infinite future, therefore we need the theory of extended integers.

6.1 Syntax

6.1.1 Primitive formulae

datatype *nl-itl* =

<i>false-d</i>	<i>(ifalse)</i>
<i>atom-d nat</i>	<i>((\$ -) [100] 99)</i>
<i>iimp-d nl-itl nl-itl</i>	<i>((- iimp -) [26,25] 25)</i>
<i>next-d nl-itl</i>	<i>((next -) [88] 87)</i>
<i>chop-d nl-itl nl-itl</i>	<i>((- schop -) [84,84] 83)</i>
<i>ldiamond-d nl-itl</i>	<i>((ldiamond -) [88] 87)</i>
<i>rdiamond-d nl-itl</i>	<i>((rdiamond -) [88] 87)</i>
<i>chopstar-d nl-itl</i>	<i>((chopstar -) [85] 85)</i>
<i>exists-d nat nl-itl</i>	<i>(Ex -. - 10)</i>

datatype *introspective-nl-itl* =

<i>cfalse-d</i>	<i>(cifalse)</i>
<i>catom-d nat</i>	<i>((\$ \$ -) [100] 99)</i>
<i>ciimp-d introspective-nl-itl introspective-nl-itl</i>	<i>((- ciimp -) [26,25] 25)</i>
<i>cnext-d introspective-nl-itl</i>	<i>((cnext -) [88] 87)</i>
<i>cchop-d introspective-nl-itl introspective-nl-itl</i>	<i>((- cschop -) [84,84] 83)</i>
<i>cchopstar-d introspective-nl-itl</i>	<i>((cchopstar -) [85] 85)</i>
<i>cexists-d nat introspective-nl-itl</i>	<i>(cEx -. - 10)</i>

datatype *future-nl-itl* =

<i>fintro-d introspective-nl-itl</i>	<i>((fintro -) [88] 87)</i>
--------------------------------------	------------------------------

```

| finot-d future-nl-itl          ((finot -) [90] 90)
| fior-d future-nl-itl future-nl-itl ((- fior -) [31,30] 30)
| frdiamond-d future-nl-itl      ((frdiamond -) [88] 87)

```

```

datatype stricly-future-nl-itl =
  sfrdiamond-d future-nl-itl      ((sfrdiamond -) [88] 87)

```

```

datatype past-nl-itl =
  pintro-d introspective-nl-itl   ((pintro -) [88] 87)
| pinot-d past-nl-itl             ((pinot -) [90] 90)
| pior-d past-nl-itl past-nl-itl ((- pior -) [31,30] 30)
| pldiamond-d past-nl-itl         ((pldiamond -) [88] 87)

```

```

datatype strictly-past-nl-itl =
  spldiamond-d past-nl-itl        ((spldiamond -) [88] 87)

```

```

datatype separated-nl-itl =
  past-d past-nl-itl              ((past -) [88] 87)
| future-d future-nl-itl          ((future -) [88] 87)
| sinot-d separated-nl-itl        ((sinot -) [90] 90)
| sior-d separated-nl-itl separated-nl-itl ((- sior -) [31,30] 30)

```

6.1.2 state, introspective, future and past formula

```

fun state-nl-itl :: nl-itl  $\Rightarrow$  bool

```

where

```

  state-nl-itl (ifalse)    = True
| state-nl-itl ($n)        = True
| state-nl-itl (f iimp g)  = ((state-nl-itl f)  $\wedge$  (state-nl-itl g))
| state-nl-itl (next f)    = False
| state-nl-itl (f schop g) = False
| state-nl-itl (ldiamond f) = False
| state-nl-itl (rdiamond f) = False
| state-nl-itl (chopstar f) = False
| state-nl-itl (Ex n. f)    = state-nl-itl f

```

```

fun introspective-nl-itl-prop :: nl-itl  $\Rightarrow$  bool

```

where

```

  introspective-nl-itl-prop (ifalse)    = True
| introspective-nl-itl-prop ($n)        = True
| introspective-nl-itl-prop (f iimp g)  = ((introspective-nl-itl-prop f)  $\wedge$  (introspective-nl-itl-prop g))
| introspective-nl-itl-prop (next f)    = introspective-nl-itl-prop f
| introspective-nl-itl-prop (f schop g) = ((introspective-nl-itl-prop f)  $\wedge$  (introspective-nl-itl-prop g))
| introspective-nl-itl-prop (ldiamond f) = False
| introspective-nl-itl-prop (rdiamond f) = False
| introspective-nl-itl-prop (chopstar f) = introspective-nl-itl-prop f
| introspective-nl-itl-prop (Ex n. f)    = introspective-nl-itl-prop f

```

```

fun future-nl-itl-prop :: nl-itl  $\Rightarrow$  bool
where
  future-nl-itl-prop (ifalse)    = True
| future-nl-itl-prop ($n)        = True
| future-nl-itl-prop (f iimp g)  = ((future-nl-itl-prop f)  $\wedge$  (future-nl-itl-prop g))
| future-nl-itl-prop (next f)    = future-nl-itl-prop f
| future-nl-itl-prop (f schop g) = ((future-nl-itl-prop f)  $\wedge$  (future-nl-itl-prop g))
| future-nl-itl-prop (ldiamond f) = False
| future-nl-itl-prop (rdiamond f) = future-nl-itl-prop f
| future-nl-itl-prop (chopstar f) = future-nl-itl-prop f
| future-nl-itl-prop (Ex n. f)   = future-nl-itl-prop f

```

```

fun past-nl-itl-prop :: nl-itl  $\Rightarrow$  bool
where
  past-nl-itl-prop (ifalse)    = True
| past-nl-itl-prop ($n)        = True
| past-nl-itl-prop (f iimp g)  = ((past-nl-itl-prop f)  $\wedge$  (past-nl-itl-prop g))
| past-nl-itl-prop (next f)    = past-nl-itl-prop f
| past-nl-itl-prop (f schop g) = ((past-nl-itl-prop f)  $\wedge$  (past-nl-itl-prop g))
| past-nl-itl-prop (ldiamond f) = (past-nl-itl-prop f)
| past-nl-itl-prop (rdiamond f) = False
| past-nl-itl-prop (chopstar f) = past-nl-itl-prop f
| past-nl-itl-prop (Ex n. f)   = past-nl-itl-prop f

```

6.1.3 Derived Operators

definition *inot-d* ((*inot* -) [90] 90)

where

inot *f* \equiv *f* iimp ifalse

definition *cinot-d* ((*cinot* -) [90] 90)

where

cinot *f* \equiv *f* ciimp cifalse

definition *itrue-d* (*itrue*)

where

itrue \equiv *inot* ifalse

definition *citrue-d* (*citrue*)

where

citrue \equiv *cinot* cifalse

definition *ior-d* ((- *ior* -) [31,30] 30)

where

f *ior* *g* \equiv (*inot* *f*) iimp *g*

definition *cior-d* ((- *cior* -) [31,30] 30)

where

$f \text{ cior } g \equiv (\text{cnot } f) \text{ ciimp } g$

definition *iand-d* ((- *iand* -) [36,35] 35)

where

$f \text{ iand } g \equiv \text{inot } (\text{inot } f \text{ ior } \text{inot } g)$

definition *ciand-d* ((- *ciand* -) [36,35] 35)

where

$f \text{ ciand } g \equiv \text{cnot } (\text{cnot } f \text{ cior } \text{cnot } g)$

definition *ieqv-d* ((- *ieqv* -) [21,20] 20)

where

$f \text{ ieqv } g \equiv ((f \text{ iimp } g) \text{ iand } (g \text{ iimp } f))$

definition *cieqv-d* ((- *cieqv* -) [21,20] 20)

where

$f \text{ cieqv } g \equiv ((f \text{ ciimp } g) \text{ ciand } (g \text{ ciimp } f))$

6.1.4 more, empty, skip and wnext

definition *more-d* (*more*)

where

$\text{more} \equiv \text{itrue } \text{schop } (\text{next } \text{itrue})$

definition *cmore-d* (*cmore*)

where

$\text{cmore} \equiv \text{cittrue } \text{cschop } (\text{cnext } \text{cittrue})$

definition *empty-d* (*empty*)

where

$\text{empty} \equiv \text{inot } \text{more}$

definition *cempty-d* (*cempty*)

where

$\text{cempty} \equiv \text{cnot } \text{cmore}$

definition *skip-d* (*skip*)

where

$\text{skip} \equiv \text{next } \text{empty}$

definition *cskip-d* (*cskip*)

where

$\text{cskip} \equiv \text{cnext } \text{cempty}$

definition *wnext-d* ((*wnext* -) [88] 87)

where

$\text{wnext } f \equiv \text{inot } (\text{next } (\text{inot } f))$

definition *cwnext-d* ((*cwnext* -) [88] 87)

where

$cwnext\ f \equiv cinot\ (cnext\ (cinot\ f))$

definition *prev-d* ((*prev* -) [88] 87)

where

$prev\ f \equiv f\ schop\ skip$

definition *cprev-d* ((*cprev* -) [88] 87)

where

$cprev\ f \equiv f\ cschop\ cskip$

definition *wprev-d* ((*wprev* -) [88] 87)

where

$wprev\ f \equiv inot(\ prev\ (inot\ f))$

definition *cwprev-d* ((*cwprev* -) [88] 87)

where

$cwprev\ f \equiv cinot(\ cprev\ (cinot\ f))$

6.1.5 rfinite, lfinite, rinf and linf

definition *rfinite-d* (*rfinite*)

where

$rfinite \equiv itrue\ schop\ empty$

definition *crfinite-d* (*crfinite*)

where

$crfinite \equiv citrue\ cschop\ cempty$

definition *rinf-d* (*rinf*)

where

$rinf \equiv inot\ rfinite$

definition *crinf-d* (*crinf*)

where

$crinf \equiv cinot\ crfinite$

definition *lfinite-d* (*lfinite*)

where

$lfinite \equiv (next\ itrue)\ ior\ empty$

definition *clfinite-d* (*clfinite*)

where

$clfinite \equiv (cnext\ citrue)\ cior\ cempty$

definition *linf-d* (*linf*)

where

$linf \equiv inot\ lfinite$

definition *clinf-d* ((*clinf*)

where

$clinf \equiv cinot\ clfinite$

6.1.6 weak chop

definition *wchop-d* ((*- wchop -*) [84,84] 83)

where $f\ wchop\ g \equiv (f\ schop\ g)\ ior\ (f\ iand\ rinf)$

definition *cwchop-d* ((*- cwchop -*) [84,84] 83)

where $f\ cwchop\ g \equiv (f\ cschop\ g)\ cior\ (f\ ciand\ crinf)$

6.1.7 Box and Diamond Operators

definition *sometimes-d* ((*diamond -*) [88] 87)

where

$diamond\ f \equiv itrue\ schop\ f$

definition *csometimes-d* ((*cdiamond -*) [88] 87)

where

$cdiamond\ f \equiv citrue\ cschop\ f$

definition *always-d* ((*box -*) [88] 87)

where

$box\ f \equiv inot\ (diamond\ (inot\ f))$

definition *calways-d* ((*cbox -*) [88] 87)

where

$cbox\ f \equiv cinot\ (cdiamond\ (cinot\ f))$

definition *di-d* ((*di -*) [88] 87)

where

$di\ f \equiv f\ schop\ itrue$

definition *cdi-d* ((*cdi -*) [88] 87)

where

$cdi\ f \equiv f\ cschop\ citrue$

definition *bi-d* ((*bi -*) [88] 87)

where

$bi\ f \equiv inot\ (di\ (inot\ f))$

definition *cbi-d* ((*cbi -*) [88] 87)

where

$cbi\ f \equiv cinot\ (cdi\ (cinot\ f))$

6.1.8 Initial and Final Operators

definition *init-d* ((*init* -) [88] 87)

where

init f \equiv (*empty iand f*) *schop itrue*

definition *cinit-d* ((*cinit* -) [88] 87)

where

cinit f \equiv (*cempty ciand f*) *cschop citrue*

definition *fin-d* ((*fin* -) [88] 87)

where

fin f \equiv *itrue schop (empty iand f)*

definition *cfin-d* ((*cfin* -) [88] 87)

where

cfin f \equiv *citrue cschop (cempty ciand f)*

6.1.9 Forall

definition *forall-d* (*Fa* -. - 10)

where

Fa v. f \equiv *inot (Ex v. inot f)*

definition *cforall-d* (*cFa* -. - 10)

where

cFa v. f \equiv *cinot (cEx v. cinot f)*

6.1.10 Big operations

definition *big-ior* :: *nl-itl list* \Rightarrow *nl-itl*

where *big-ior F* = *foldr* ($\lambda x y. x \text{ ior } y$) *F ifalse*

definition *big-cior* :: *introspective-nl-itl list* \Rightarrow *introspective-nl-itl*

where *big-cior F* = *foldr* ($\lambda x y. x \text{ cior } y$) *F cifalse*

definition *big-iand* :: *nl-itl list* \Rightarrow *nl-itl*

where *big-iand F* = *foldr* ($\lambda x y. x \text{ iand } y$) *F itrue*

definition *big-ciand* :: *introspective-nl-itl list* \Rightarrow *introspective-nl-itl*

where *big-ciand F* = *foldr* ($\lambda x y. x \text{ ciand } y$) *F citrue*

definition *state-nl-itl-list* :: *nl-itl list* \Rightarrow *bool*

where *state-nl-itl-list L* = *foldr* ($\lambda x y. \text{state-nl-itl } x \wedge y$) *L True*

6.2 Semantics

6.2.1 Intervals

type-synonym $interval = int \Rightarrow nat\ set$

6.2.2 Semantics Primitive Operators

definition $similar :: interval \Rightarrow nat \Rightarrow interval \Rightarrow bool \ ((- \sim -))$
where $similar\ \sigma 0\ n\ \sigma 1 = (\forall\ k\ p. (p \in (\sigma 0\ k) \wedge p \neq n) \longleftrightarrow (p \in (\sigma 1\ k) \wedge p \neq n))$

definition $csimilar :: interval \Rightarrow nat \Rightarrow eint \Rightarrow eint \Rightarrow interval \Rightarrow bool$
where $csimilar\ \sigma 0\ n\ i\ j\ \sigma 1 =$
 $(\forall\ k. \text{if } i \leq k \wedge k \leq j \text{ then } (\forall p. (p \in (\sigma 0\ k) \wedge p \neq n) \longleftrightarrow (p \in (\sigma 1\ k) \wedge p \neq n))$
 $\text{else } (\forall p. (p \in (\sigma 0\ k)) \longleftrightarrow (p \in (\sigma 1\ k))))$

definition $similar_s :: interval \Rightarrow nat\ set \Rightarrow interval \Rightarrow bool \ ((- \approx -))$
where $similar_s\ \sigma 0\ A\ \sigma 1 = (\forall\ k\ p. p \notin A \longrightarrow (p \in (\sigma 0\ k) \longleftrightarrow p \in (\sigma 1\ k)))$

definition $upd\text{-}add\text{-}sigma :: interval \Rightarrow int \Rightarrow nat \Rightarrow interval$
where $upd\text{-}add\text{-}sigma\ \sigma\ k\ n = \sigma(k := insert\ n\ (\sigma\ k))$

definition $upd\text{-}remove\text{-}sigma :: interval \Rightarrow int \Rightarrow nat\ set \Rightarrow interval$
where $upd\text{-}remove\text{-}sigma\ \sigma\ k\ A = \sigma(k := (\sigma\ k) - A)$

definition $upd\text{-}remove\text{-}all\text{-}sigma :: interval \Rightarrow nat\ set \Rightarrow interval$
where $upd\text{-}remove\text{-}all\text{-}sigma\ \sigma\ A = (\lambda\ i. (upd\text{-}remove\text{-}sigma\ \sigma\ i\ A)\ i)$

primrec $fvars :: nl\text{-}itl \Rightarrow nat\ set$
where

$fvars\ ifalse = \{\}$
 $| fvars\ (\$n) = \{n\}$
 $| fvars\ (f\ iimp\ g) = (fvars\ f) \cup (fvars\ g)$
 $| fvars\ (next\ f) = fvars\ f$
 $| fvars\ (f\ schop\ g) = (fvars\ f) \cup (fvars\ g)$
 $| fvars\ (ldiamond\ f) = fvars\ f$
 $| fvars\ (rdiamond\ f) = fvars\ f$
 $| fvars\ (chopstar\ f) = fvars\ f$
 $| fvars\ (Ex\ n. f) = (fvars\ f) - \{n\}$

primrec $cfvars :: introspective\text{-}nl\text{-}itl \Rightarrow nat\ set$
where

$cfvars\ cifalse = \{\}$
 $| cfvars\ (\$ \$n) = \{n\}$
 $| cfvars\ (f\ ciimp\ g) = (cfvars\ f) \cup (cfvars\ g)$
 $| cfvars\ (cnext\ f) = cfvars\ f$
 $| cfvars\ (f\ cschop\ g) = (cfvars\ f) \cup (cfvars\ g)$
 $| cfvars\ (cchopstar\ f) = cfvars\ f$
 $| cfvars\ (cEx\ n. f) = (cfvars\ f) - \{n\}$

primrec $bvars :: nl\text{-}itl \Rightarrow nat\ set$

where

$bvars\ ifalse = \{\}$
 $bvars\ (\$n) = \{\}$
 $bvars\ (f\ iimp\ g) = (bvars\ f) \cup (bvars\ g)$
 $bvars\ (next\ f) = bvars\ f$
 $bvars\ (f\ schop\ g) = (bvars\ f) \cup (bvars\ g)$
 $bvars\ (ldiamond\ f) = bvars\ f$
 $bvars\ (rdiamond\ f) = bvars\ f$
 $bvars\ (chopstar\ f) = bvars\ f$
 $bvars\ (Ex\ n.\ f) = (insert\ n\ (bvars\ f))$

primrec $cbvars :: introspective-nl-itl \Rightarrow nat\ set$

where

$cbvars\ cifalse = \{\}$
 $cbvars\ (\$n) = \{\}$
 $cbvars\ (f\ ciimp\ g) = (cbvars\ f) \cup (cbvars\ g)$
 $cbvars\ (cnext\ f) = cbvars\ f$
 $cbvars\ (f\ cschop\ g) = (cbvars\ f) \cup (cbvars\ g)$
 $cbvars\ (cchopstar\ f) = cbvars\ f$
 $cbvars\ (cEx\ n.\ f) = (insert\ n\ (cbvars\ f))$

primrec $suform :: nl-itl \Rightarrow nat \Rightarrow nl-itl \Rightarrow nl-itl$

where

$suform\ ifalse\ n\ h = ifalse$
 $suform\ (\$k)\ n\ h = (if\ k = n\ then\ h\ else\ (\$k))$
 $suform\ (f\ iimp\ g)\ n\ h = ((suform\ f\ n\ h)\ iimp\ (suform\ g\ n\ h))$
 $suform\ (next\ f)\ n\ h = (next\ (suform\ f\ n\ h))$
 $suform\ (f\ schop\ g)\ n\ h = ((suform\ f\ n\ h)\ schop\ (suform\ g\ n\ h))$
 $suform\ (ldiamond\ f)\ n\ h = (ldiamond\ (suform\ f\ n\ h))$
 $suform\ (rdiamond\ f)\ n\ h = (rdiamond\ (suform\ f\ n\ h))$
 $suform\ (chopstar\ f)\ n\ h = (chopstar\ (suform\ f\ n\ h))$
 $suform\ (Ex\ k.\ f)\ n\ h = (Ex\ k.\ (suform\ f\ n\ h))$

primrec $csuform :: introspective-nl-itl \Rightarrow nat \Rightarrow introspective-nl-itl \Rightarrow introspective-nl-itl$

where

$csuform\ cifalse\ n\ h = cifalse$
 $csuform\ (\$k)\ n\ h = (if\ k = n\ then\ h\ else\ (\$k))$
 $csuform\ (f\ ciimp\ g)\ n\ h = ((csuform\ f\ n\ h)\ ciimp\ (csuform\ g\ n\ h))$
 $csuform\ (cnext\ f)\ n\ h = (cnext\ (csuform\ f\ n\ h))$
 $csuform\ (f\ cschop\ g)\ n\ h = ((csuform\ f\ n\ h)\ cschop\ (csuform\ g\ n\ h))$
 $csuform\ (cchopstar\ f)\ n\ h = (cchopstar\ (csuform\ f\ n\ h))$
 $csuform\ (cEx\ k.\ f)\ n\ h = (cEx\ k.\ (csuform\ f\ n\ h))$

primrec $crename :: introspective-nl-itl \Rightarrow nat \Rightarrow nat \Rightarrow introspective-nl-itl$

where

$crename\ cifalse\ n\ k = cifalse$
 $crename\ (\$m)\ n\ k = (if\ m = n\ then\ \$k\ else\ \$m)$
 $crename\ (f\ ciimp\ g)\ n\ k = ((crename\ f\ n\ k)\ ciimp\ (crename\ g\ n\ k))$

```

|  crename (cnext f) n k    = (cnext (crename f n k))
|  crename (f cschop g) n k = ((crename f n k) cschop (crename g n k))
|  crename (cchopstar f) n k = (cchopstar (crename f n k))
|  crename (cEx m. f) n k   = (cEx (if m = n then k else m). (crename f n k))

```

primrec *rename* :: *nl-itl* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *nl-itl*

where

```

|  rename ifalse n k      = ifalse
|  rename ($m) n k       = (if m = n then $k else $m)
|  rename (f iimp g) n k  = ((rename f n k) iimp (rename g n k))
|  rename (next f) n k    = (next (rename f n k))
|  rename (f schop g) n k = ((rename f n k) schop (rename g n k))
|  rename (ldiamond f) n k = (ldiamond (rename f n k))
|  rename (rdiamond f) n k = (rdiamond (rename f n k))
|  rename (chopstar f) n k = (chopstar (rename f n k))
|  rename (Ex m. f) n k   = (Ex (if m = n then k else m). (rename f n k))

```

primrec *max-var* :: *nl-itl* \Rightarrow *nat* \Rightarrow *nat*

where

```

|  max-var ifalse m      = m
|  max-var ($k) m       = (if m < k then k else m)
|  max-var (f iimp g) m  = (if (max-var f m) < (max-var g m) then (max-var g m) else (max-var f m))
|  max-var (next f) m    = max-var f m
|  max-var (f schop g) m = (if (max-var f m) < (max-var g m) then (max-var g m) else (max-var f m))
|  max-var (ldiamond f) m = max-var f m
|  max-var (rdiamond f) m = max-var f m
|  max-var (chopstar f) m = max-var f m
|  max-var (Ex k. f) m   = (if (max-var f m) < k then k else (max-var f m))

```

primrec *cmax-var* :: *introspective-nl-itl* \Rightarrow *nat* \Rightarrow *nat*

where

```

|  cmax-var cifalse m      = m
|  cmax-var ($$k) m       = (if m < k then k else m)
|  cmax-var (f ciimp g) m  = (if (cmax-var f m) < (cmax-var g m) then (cmax-var g m) else (cmax-var f m))
|  cmax-var (cnext f) m    = cmax-var f m
|  cmax-var (f cschop g) m = (if (cmax-var f m) < (cmax-var g m) then (cmax-var g m) else (cmax-var f m))
|  cmax-var (cchopstar f) m = cmax-var f m
|  cmax-var (cEx k. f) m   = (if (cmax-var f m) < k then k else (cmax-var f m))

```

definition *upd* :: *interval* \Rightarrow *nat* \Rightarrow (*int* \Rightarrow *bool*) \Rightarrow *interval*

where *upd* σ *n* *l* = (λ *k*. if (*l* *k*) then insert *n* (σ *k*) else ((σ *k*) - {*n*}))

fun *semantics-nl-itl* :: *interval* \Rightarrow *eint* \Rightarrow *eint* \Rightarrow *nl-itl* \Rightarrow *bool*

((- - - \models -) [80,80,80,10] 10)

where

(σ *i j* \models *ifalse*) = *False*
| (σ *i j* \models ($\$p$)) = ($i \neq -\infty \wedge (p \in (\sigma$ (*int-of-eint* *i*))))
| (σ *i j* \models (*f iimp g*)) = ((σ *i j* \models *f*) \longrightarrow (σ *i j* \models *g*))
| (σ *i j* \models (*next f*)) = ($i \neq -\infty \wedge i < j \wedge (\sigma$ (*i+ (eint 1)*) *j* \models *f*))
| (σ *i j* \models (*f schop g*)) = ($\exists k. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma$ *i k* \models *f*) \wedge (σ *k j* \models *g*))
| (σ *i j* \models *ldiamond f*) = ($i > -\infty \wedge (\exists k. k \leq i \wedge (\sigma$ *k i* \models *f*)))
| (σ *i j* \models *rdiamond f*) = ($j < \infty \wedge (\exists k. k \geq j \wedge (\sigma$ *j k* \models *f*)))
| (σ *i j* \models (*chopstar f*)) =
($i = j \vee$
($i \neq j \wedge i \neq -\infty \wedge j \neq \infty \wedge$
(\exists (*lrf* :: *eint nellist*).
(*nnth lrf 0*) = $i \wedge \text{nfinite lrf} \wedge \text{nlast lrf} = j \wedge$
 $\text{nridx } (\lambda a b. a < b \wedge (\sigma a b \models f)) \text{ lrf}$
))
) \vee
($i \neq j \wedge i = -\infty \wedge j \neq \infty \wedge$
(\exists (*li* :: *eint nellist*).
(*nnth li 0*) = $j \wedge \neg \text{nfinite li} \wedge$
 $\text{nridx } (\lambda a b. b < a \wedge (\sigma b a \models f)) \text{ li}$
))
) \vee
($i \neq j \wedge i \neq -\infty \wedge j = \infty \wedge$
(\exists (*ri* :: *eint nellist*).
(*nnth ri 0*) = $i \wedge \neg \text{nfinite ri} \wedge$
 $\text{nridx } (\lambda a b. a < b \wedge (\sigma a b \models f)) \text{ ri}$
))
) \vee
($i \neq j \wedge i = -\infty \wedge j = \infty \wedge$
(\exists (*li* :: *int nellist*) (*ri* :: *int nellist*) .
(*nnth li 0*) = (*nnth ri 0*) \wedge
 $\neg \text{nfinite li} \wedge \text{nridx } (\lambda a b. b < a \wedge (\sigma b a \models f)) \text{ li} \wedge$
 $\neg \text{nfinite ri} \wedge \text{nridx } (\lambda a b. a < b \wedge (\sigma a b \models f)) \text{ ri}$
))
))
)

| (σ *i j* \models *Ex n. f*) = ($\exists l. ((\text{upd } \sigma n l) i j \models f)$))

fun *semantics-introspective-nl-itl* :: *interval* \Rightarrow *eint* \Rightarrow *eint* \Rightarrow *introspective-nl-itl* \Rightarrow *bool*

((- - - *c* \models -) [80,80,80,10] 10)

where

(σ *i j* *c* \models *cifalse*) = *False*
| (σ *i j* *c* \models ($\$\p)) = ($i \neq -\infty \wedge (p \in (\sigma$ (*int-of-eint* *i*))))
| (σ *i j* *c* \models (*f ciimp g*)) = ((σ *i j* *c* \models *f*) \longrightarrow (σ *i j* *c* \models *g*))

$$\begin{aligned}
& | (\sigma \ i \ j \ c \models (cnext \ f)) = (i \neq -\infty \wedge i < j \wedge (\sigma \ (i+ (eint \ 1)) \ j \ c \models f)) \\
& | (\sigma \ i \ j \ c \models (f \ cschop \ g)) = (\exists k. \ i \leq eint \ k \wedge eint \ k \leq j \wedge (\sigma \ i \ k \ c \models f) \wedge (\sigma \ k \ j \ c \models g)) \\
& | (\sigma \ i \ j \ c \models (cchopstar \ f)) = \\
& \quad (i = j \vee \\
& \quad (i \neq j \wedge i \neq -\infty \wedge j \neq \infty \wedge \\
& \quad \quad (\exists (lrf :: eint \ nellist). \\
& \quad \quad \quad (nnth \ lrf \ 0) = i \wedge nfinite \ lrf \wedge nlast \ lrf = j \wedge \\
& \quad \quad \quad nridx \ (\lambda a \ b. a < b \wedge (\sigma \ a \ b \ c \models f)) \ lrf \\
& \quad \quad) \\
& \quad) \vee \\
& \quad (i \neq j \wedge i = -\infty \wedge j \neq \infty \wedge \\
& \quad \quad (\exists (li :: eint \ nellist). \\
& \quad \quad \quad (nnth \ li \ 0) = j \wedge \neg nfinite \ li \wedge \\
& \quad \quad \quad nridx \ (\lambda a \ b. b < a \wedge (\sigma \ b \ a \ c \models f)) \ li \\
& \quad \quad) \\
& \quad) \vee \\
& \quad (i \neq j \wedge i \neq -\infty \wedge j = \infty \wedge \\
& \quad \quad (\exists (ri :: eint \ nellist). \\
& \quad \quad \quad (nnth \ ri \ 0) = i \wedge \neg nfinite \ ri \wedge \\
& \quad \quad \quad nridx \ (\lambda a \ b. a < b \wedge (\sigma \ a \ b \ c \models f)) \ ri \\
& \quad \quad) \\
& \quad) \vee \\
& \quad (i \neq j \wedge i = -\infty \wedge j = \infty \wedge \\
& \quad \quad (\exists (li :: int \ nellist) (ri :: int \ nellist) . \\
& \quad \quad \quad (nnth \ li \ 0) = (nnth \ ri \ 0) \wedge \\
& \quad \quad \quad \neg nfinite \ li \wedge nridx \ (\lambda a \ b. b < a \wedge (\sigma \ b \ a \ c \models f)) \ li \wedge \\
& \quad \quad \quad \neg nfinite \ ri \wedge nridx \ (\lambda a \ b. a < b \wedge (\sigma \ a \ b \ c \models f)) \ ri \\
& \quad \quad) \\
& \quad) \\
&) \\
&) \\
& | (\sigma \ i \ j \ c \models cEx \ n. \ f) = (\exists l. ((upd \ \sigma \ n \ l) \ i \ j \ c \models f) \)
\end{aligned}$$

fun *semantics-future-nl-itl* :: *interval* \Rightarrow *eint* \Rightarrow *eint* \Rightarrow *future-nl-itl* \Rightarrow *bool*
 ((- - - f \models -) [80,80,80,10] 10)

where

$$\begin{aligned}
& (\sigma \ i \ j \ f \models fintro \ f) = (\sigma \ i \ j \ c \models f) \\
& | (\sigma \ i \ j \ f \models finot \ f) = (\neg (\sigma \ i \ j \ f \models f)) \\
& | (\sigma \ i \ j \ f \models (f \ fior \ g)) = ((\sigma \ i \ j \ f \models f) \vee (\sigma \ i \ j \ f \models g)) \\
& | (\sigma \ i \ j \ f \models frdiamond \ f) = (j < \infty \wedge (\exists k. \ k \geq j \wedge (\sigma \ j \ k \ f \models f)))
\end{aligned}$$

fun *semantics-strictly-future-nl-itl* :: *interval* \Rightarrow *eint* \Rightarrow *eint* \Rightarrow *stricly-future-nl-itl* \Rightarrow *bool*
 ((- - - sf \models -) [80,80,80,10] 10)

where

$$(\sigma \ i \ j \ sf \models sfrdiamond \ f) = (j < \infty \wedge (\exists k. \ k = j \vee (k > j \wedge (\sigma \ j \ k \ f \models f))))$$

fun *semantics-past-nl-itl* :: *interval* \Rightarrow *eint* \Rightarrow *eint* \Rightarrow *past-nl-itl* \Rightarrow *bool*

((- - - p \models -) [80,80,80,10] 10)

where

(σ i j p \models pintro f) = (σ i j c \models f)
 | (σ i j p \models pinot f) = (\neg (σ i j p \models f))
 | (σ i j p \models (f pior g)) = ((σ i j p \models f) \vee (σ i j p \models g))
 | (σ i j p \models pldiamond f) = (i > $-\infty$ \wedge (\exists k. k \leq i \wedge (σ k i p \models f)))

fun semantics-strictly-past-nl-itl :: interval \Rightarrow eint \Rightarrow eint \Rightarrow strictly-past-nl-itl \Rightarrow bool
 ((- - - sp \models -) [80,80,80,10] 10)

where

(σ i j sp \models spldiamond f) = (i > $-\infty$ \wedge (\exists k. k = i \vee (k < i \wedge (σ k i p \models f))))

fun semantics-separated-nl-itl :: interval \Rightarrow eint \Rightarrow eint \Rightarrow separated-nl-itl \Rightarrow bool
 ((- - - s \models -) [80,80,80,10] 10)

where

(σ i j s \models past f) = (σ i j p \models f)
 | (σ i j s \models future f) = (σ i j f \models f)
 | (σ i j s \models sinot f) = (\neg (σ i j s \models f))
 | (σ i j s \models (f sior g)) = ((σ i j s \models f) \vee (σ i j s \models g))

lemma similar-defs :

similar $\sigma 0$ n $\sigma 1 \longleftrightarrow (\forall$ k p . (p \in ($\sigma 0$ k) \wedge p \neq n) \longleftrightarrow (p \in ($\sigma 1$ k) \wedge p \neq n))

unfolding similar-def **by** auto

lemma csimilar-defs [simp]:

csimilar $\sigma 0$ n i j $\sigma 1 \longleftrightarrow$
 (\forall k p. if i \leq k \wedge k \leq j then (p \in ($\sigma 0$ k) \wedge p \neq n) \longleftrightarrow (p \in ($\sigma 1$ k) \wedge p \neq n)
 else (p \in ($\sigma 0$ k)) \longleftrightarrow (p \in ($\sigma 1$ k)))

unfolding csimilar-def **by** auto

lemma similar-defs-alt:

(\forall k . ($\sigma 0$ k) - {n} = ($\sigma 1$ k) - {n}) \longleftrightarrow similar $\sigma 0$ n $\sigma 1$

unfolding similar-def **by** auto

lemma similars-defs [simp]:

similars $\sigma 0$ A $\sigma 1 \longleftrightarrow (\forall$ k p. p \notin A \longrightarrow (p \in ($\sigma 0$ k) \longleftrightarrow p \in ($\sigma 1$ k)))

unfolding similars-def **by** auto

lemma similars-defs-alt:

similars $\sigma 0$ A $\sigma 1 \longleftrightarrow (\forall$ k. ($\sigma 0$ k) - A = ($\sigma 1$ k) - A)

unfolding similars-def **by** auto

lemma similar-similars:

similar $\sigma 0$ n $\sigma 1$ = similars $\sigma 0$ {n} $\sigma 1$

using similar-defs-alt similars-defs-alt **by** presburger

lemma *similar-mono*:
assumes $A \subseteq B$
 similar $\sigma 0 A \sigma 1$
shows *similar* $\sigma 0 B \sigma 1$
using *assms* **by** *auto*

lemma *similar-refl*:
 similar $\sigma n \sigma$
using *similar-defs-alt* **by** *blast*

lemma *csimilar-refl*:
 csimilar $\sigma n i j \sigma$
by *simp*

lemma *similar-commute*:
 similar $\sigma n \sigma' = \text{similar } \sigma' n \sigma$
by (*metis similar-defs-alt*)

lemma *csimilar-commute*:
 csimilar $\sigma n i j \sigma' = \text{csimilar } \sigma' n i j \sigma$
by *simp blast*

lemma *similar-trans*:
assumes *similar* $\sigma n \sigma'$
 similar $\sigma' n \sigma''$
shows *similar* $\sigma n \sigma''$
using *assms*
by (*metis similar-defs-alt*)

lemma *csimilar-trans*:
assumes *csimilar* $\sigma n i j \sigma'$
 csimilar $\sigma' n i j \sigma''$
shows *csimilar* $\sigma n i j \sigma''$
using *assms* **by** *simp*

lemma *similar-refl*:
 similar $\sigma A \sigma$
by *simp*

lemma *similar-commute*:
 similar $\sigma A \sigma' = \text{similar } \sigma' A \sigma$
by *simp blast*

lemma *similar-trans*:
assumes *similar* $\sigma A \sigma'$

$\text{similar} \sigma' A \sigma''$
shows $\text{similar} \sigma A \sigma''$
using *assms* **by** *simp*

lemma *upd-add-sigma-defs* [*simp*]:
 $\text{upd-add-sigma } \sigma \ k \ n = \sigma(k := \text{insert } n (\sigma \ k))$
unfolding *upd-add-sigma-def* **by** *simp*

lemma *upd-remove-sigma-defs* [*simp*]:
 $\text{upd-remove-sigma } \sigma \ k \ A = \sigma(k := (\sigma \ k) - A)$
unfolding *upd-remove-sigma-def* **by** *simp*

lemma *upd-remove-all-sigma-defs* [*simp*]:
 $\text{upd-remove-all-sigma } \sigma \ A = (\lambda i. (\text{upd-remove-sigma } \sigma \ i \ A) \ i)$
unfolding *upd-remove-all-sigma-def* **by** *simp*

lemma *similar-upd-remove-all*:
 $\text{similar} \sigma \ A (\text{upd-remove-all-sigma } \sigma \ A)$
by *simp*

lemma *finite-bvars*:
 $\text{finite } (\text{bvars } f)$
by (*induct* *f*) *simp-all*

lemma *finite-fvars*:
 $\text{finite } (\text{fvars } f)$
by (*induct* *f*) *simp-all*

lemma *similar-upd*:
shows $\text{similar} \sigma \ n (\text{upd } \sigma \ n \ l)$
using *similar-defs-alt*[*of* $\sigma \ n (\text{upd } \sigma \ n \ l)$]
using *upd-def* **by** *force*

lemma *exist-sigma-exist-value*:
 $(\exists \sigma'. (\sigma' \ i \ j \models f) \wedge (\text{similar } \sigma \ n \ \sigma')) \longleftrightarrow (\exists l. (\text{upd } \sigma \ n \ l) \ i \ j \models f)$
proof –
have 2: $(\exists \sigma'. (\sigma' \ i \ j \models f) \wedge (\text{similar } \sigma \ n \ \sigma')) \implies$
 $(\exists l. ((\text{upd } \sigma \ n \ l) \ i \ j \models f))$
proof –
assume *a0*: $(\exists \sigma'. (\sigma' \ i \ j \models f) \wedge (\text{similar } \sigma \ n \ \sigma'))$

```

show (∃ l . ((upd σ n l) i j ⊨ f))
proof -
  obtain σ' where 4: (σ' i j ⊨ f) ∧ (similar σ n σ')
    using a0 by blast
  have 6: σ' = (upd σ' n (λ k. n ∈ (σ' k) ))
    unfolding upd-def
    by (metis Diff-empty Diff-insert0 insert-absorb)
  have 7: σ = (upd σ n (λ k. n ∈ (σ k) ))
    unfolding upd-def
    by (metis Diff-empty Diff-insert0 insert-absorb)
  have 8: ∧x . (σ' x) = (upd σ n (λ k. n ∈ (σ' k) )) x
    unfolding upd-def using 4
    by (metis 6 insert-Diff-single similar-defs-alt upd-def)
  have 9: σ' = (upd σ n (λ k. n ∈ (σ' k) ))
    using 8 by blast
  have 10: (upd σ n (λ k. n ∈ (σ' k) )) i j ⊨ f
    using 4 9 by auto
  show ?thesis using 10 by blast
qed
qed
have 11: (∃ l . ((upd σ n l) i j ⊨ f)) ⇒
  (∃ σ'. (σ' i j ⊨ f) ∧ (similar σ n σ' ))
proof -
  assume a1: (∃ l . ((upd σ n l) i j ⊨ f))
  show (∃ σ'. (σ' i j ⊨ f) ∧ (similar σ n σ' ))
  proof -
    obtain l where 12: ((upd σ n l) i j ⊨ f)
      using a1 by blast
    have 13: similar σ n (upd σ n l)
      by (simp add: similar-upd)
    show ?thesis
      using 12 13 by blast
  qed
qed
show ?thesis
using 11 2 by blast
qed

```

lemma *ceexist-sigma-ceexist-value*:

$$(\exists \sigma'. (\sigma' i j c \models f) \wedge (\text{similar } \sigma n \sigma')) \longleftrightarrow (\exists l. (\text{upd } \sigma n l) i j c \models f)$$

```

proof -
  have 2: (∃ σ'. (σ' i j c ⊨ f) ∧ (similar σ n σ' )) ⇒
    (∃ l . ((upd σ n l) i j c ⊨ f))
  proof -
    assume a0: (∃ σ'. (σ' i j c ⊨ f) ∧ (similar σ n σ' ))
    show (∃ l . ((upd σ n l) i j c ⊨ f))
    proof -
      obtain σ' where 4: (σ' i j c ⊨ f) ∧ (similar σ n σ')
        using a0 by blast

```

```

have 6:  $\sigma' = (\text{upd } \sigma' \ n \ (\lambda \ k. \ n \in (\sigma' \ k) \ ))$ 
  unfolding upd-def
  by (metis Diff-empty Diff-insert0 insert-absorb)
have 7:  $\sigma = (\text{upd } \sigma \ n \ (\lambda \ k. \ n \in (\sigma \ k) \ ))$ 
  unfolding upd-def
  by (metis Diff-empty Diff-insert0 insert-absorb)
have 8:  $\bigwedge x. (\sigma' \ x) = (\text{upd } \sigma \ n \ (\lambda \ k. \ n \in (\sigma' \ k) \ )) \ x$ 
  unfolding upd-def using 4
  by (metis 6 insert-Diff-single similar-defs-alt upd-def)
have 9:  $\sigma' = (\text{upd } \sigma \ n \ (\lambda \ k. \ n \in (\sigma' \ k) \ ))$ 
  using 8 by blast
have 10:  $(\text{upd } \sigma \ n \ (\lambda \ k. \ n \in (\sigma' \ k) \ )) \ i \ j \ c \models f$ 
  using 4 9 by auto
show ?thesis using 10 by blast
qed
qed
have 11:  $(\exists \ l. ((\text{upd } \sigma \ n \ l) \ i \ j \ c \models f)) \implies$ 
   $(\exists \ \sigma'. (\sigma' \ i \ j \ c \models f) \wedge (\text{similar } \sigma \ n \ \sigma'))$ 
proof -
  assume a1:  $(\exists \ l. ((\text{upd } \sigma \ n \ l) \ i \ j \ c \models f))$ 
  show  $(\exists \ \sigma'. (\sigma' \ i \ j \ c \models f) \wedge (\text{similar } \sigma \ n \ \sigma'))$ 
  proof -
    obtain l where 12:  $(\text{upd } \sigma \ n \ l) \ i \ j \ c \models f$ 
    using a1 by blast
    have 13:  $\text{similar } \sigma \ n \ (\text{upd } \sigma \ n \ l)$ 
    by (simp add: similar-upd)
    show ?thesis
    using 12 13 by blast
  qed
qed
show ?thesis
using 11 2 by blast
qed

```

```

lemma upd-absorb:
  assumes  $x1 = x2$ 
  shows  $\text{upd } (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2 = (\text{upd } \sigma \ x2 \ l2)$ 
proof -
  have 1:  $\bigwedge i. (\text{upd } (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \ i = (\text{upd } \sigma \ x2 \ l2) \ i$ 
    using assms unfolding upd-def by auto
  show ?thesis using 1 by auto
qed

```

```

lemma upd-swap:
  assumes  $x1 \neq x2$ 
  shows  $\text{upd } (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2 = (\text{upd } (\text{upd } \sigma \ x2 \ l2) \ x1 \ l1)$ 
proof -
  have 1:  $\bigwedge i. (\text{upd } (\text{upd } \sigma \ x1 \ l1) \ x2 \ l2) \ i = (\text{upd } (\text{upd } \sigma \ x2 \ l2) \ x1 \ l1) \ i$ 
    using assms unfolding upd-def by auto
  show ?thesis using 1 by auto

```

qed

lemma *not-fvar-upd*:

assumes $n \notin \text{fvars } f$

shows $(\sigma \ i \ j \models f) = ((\text{upd } \sigma \ n \ l) \ i \ j \models f)$

using *assms*

proof (induction *f* arbitrary: $\sigma \ l \ i \ j$)

case *false-d*

then show ?case by *simp*

next

case (*atom-d* x)

then show ?case

by (metis *fvars.simps*(2) *insertCI semantics-nl-itl.simps*(2) *similar-defs similar-upd*)

next

case (*iimp-d* $f1 \ f2$)

then show ?case by *simp*

next

case (*next-d* f)

then show ?case by *auto*

next

case (*chop-d* $f1 \ f2$)

then show ?case by *simp*

next

case (*ldiamond-d* f)

then show ?case by *simp*

next

case (*rdiamond-d* f)

then show ?case by *simp*

next

case (*chopstar-d* f)

then show ?case

by *force*

next

case (*exists-d* $x1 \ f$)

then show ?case

proof –

have $e1: (\sigma \ i \ j \models \text{Ex } x1. f) = (\exists \ l. ((\text{upd } \sigma \ x1 \ l) \ i \ j \models f))$

using *exist-sigma-exist-value* by *auto*

have $e2: ((\text{upd } \sigma \ n \ l) \ i \ j \models \text{Ex } x1. f) =$
 $(\exists \ l1. ((\text{upd } (\text{upd } \sigma \ n \ l) \ x1 \ l1) \ i \ j \models f))$

using *exist-sigma-exist-value* by *force*

have $e3: (\exists \ l. ((\text{upd } \sigma \ x1 \ l) \ i \ j \models f)) \implies$
 $(\exists \ l1. ((\text{upd } (\text{upd } \sigma \ n \ l) \ x1 \ l1) \ i \ j \models f))$

by (metis *Diff-iff exists-d.IH exists-d.premis fvars.simps*(9) *singletonD upd-absorb upd-swap*)

have $e4: (\exists \ l1. ((\text{upd } (\text{upd } \sigma \ n \ l) \ x1 \ l1) \ i \ j \models f)) \implies$
 $(\exists \ l. ((\text{upd } \sigma \ x1 \ l) \ i \ j \models f))$

by (metis *Diff-iff emptyE exists-d.IH exists-d.premis fvars.simps*(9) *insertE upd-absorb upd-swap*)

show ?thesis

using $e1 \ e2 \ e3 \ e4$ by *blast*

qed
qed

lemma *not-fvar-upd-introspective*:

assumes $n \notin \text{cfvars } f$
shows $(\sigma \ i \ j \ c \models f) = ((\text{upd } \sigma \ n \ l) \ i \ j \ c \models f)$
using *assms*
proof (induction *f* arbitrary: $\sigma \ l \ i \ j$)
case *cfalse-d*
then show ?case by simp
next
case (catom-d x)
then show ?case
by (metis *cfvars.simps(2)* *insertCI semantics-introspective-nl-itl.simps(2)* *similar-defs similar-upd*)
next
case (ciimp-d $f1 \ f2$)
then show ?case by simp
next
case (cnext-d f)
then show ?case by auto
next
case (cchop-d $f1 \ f2$)
then show ?case by simp
next
case (cchopstar-d f)
then show ?case by force
next
case (cexists-d $x1 \ f$)
then show ?case
proof –
have $e1: (\sigma \ i \ j \ c \models \text{cEx } x1. f) = (\exists \ l. ((\text{upd } \sigma \ x1 \ l) \ i \ j \ c \models f))$
using *cexist-sigma-cexist-value* by auto
have $e2: ((\text{upd } \sigma \ n \ l) \ i \ j \ c \models \text{cEx } x1. f) =$
 $(\exists \ l1. ((\text{upd } (\text{upd } \sigma \ n \ l) \ x1 \ l1) \ i \ j \ c \models f))$
using *cexist-sigma-cexist-value* by force
have $e3: (\exists \ l. ((\text{upd } \sigma \ x1 \ l) \ i \ j \ c \models f)) \implies$
 $(\exists \ l1. ((\text{upd } (\text{upd } \sigma \ n \ l) \ x1 \ l1) \ i \ j \ c \models f))$
by (metis *Diff-iff cexists-d.IH cexists-d.prem* *cfvars.simps(7)* *singletonD upd-absorb upd-swap*)
have $e4: (\exists \ l1. ((\text{upd } (\text{upd } \sigma \ n \ l) \ x1 \ l1) \ i \ j \ c \models f)) \implies$
 $(\exists \ l. ((\text{upd } \sigma \ x1 \ l) \ i \ j \ c \models f))$
by (metis *Diff-iff emptyE cexists-d.IH cexists-d.prem* *cfvars.simps(7)* *insertE upd-absorb upd-swap*)
show ?thesis
using $e1 \ e2 \ e3 \ e4$ by blast
qed
qed

lemma *inot-defs [simp]*:

$(\sigma \ i \ j \models \text{inot } f) = (\neg(\sigma \ i \ j \models f))$
by (*simp add: inot-d-def*)

lemma *cinot-defs* [*simp*]:
 $(\sigma \ i \ j \models \text{cinot } f) = (\neg(\sigma \ i \ j \models f))$
by (*simp add: cinot-d-def*)

lemma *ior-defs* [*simp*]:
 $(\sigma \ i \ j \models f1 \ \text{ior} \ f2) = ((\sigma \ i \ j \models f1) \vee (\sigma \ i \ j \models f2))$
by (*metis inot-defs ior-d-def semantics-nl-itl.simps(3)*)

lemma *cior-defs* [*simp*]:
 $(\sigma \ i \ j \models f1 \ \text{cior} \ f2) = ((\sigma \ i \ j \models f1) \vee (\sigma \ i \ j \models f2))$
by (*metis cinot-defs cior-d-def semantics-introspective-nl-itl.simps(3)*)

lemma *iand-defs* [*simp*]:
 $(\sigma \ i \ j \models f1 \ \text{iand} \ f2) = ((\sigma \ i \ j \models f1) \wedge (\sigma \ i \ j \models f2))$
by (*simp add: iand-d-def*)

lemma *ciand-defs* [*simp*]:
 $(\sigma \ i \ j \models f1 \ \text{ciand} \ f2) = ((\sigma \ i \ j \models f1) \wedge (\sigma \ i \ j \models f2))$
by (*simp add: ciand-d-def*)

lemma *ieqv-defs* [*simp*]:
 $(\sigma \ i \ j \models f1 \ \text{ieqv} \ f2) = ((\sigma \ i \ j \models f1) = (\sigma \ i \ j \models f2))$
by (*metis iand-defs ieqv-d-def semantics-nl-itl.simps(3)*)

lemma *cieqv-defs* [*simp*]:
 $(\sigma \ i \ j \models f1 \ \text{cieqv} \ f2) = ((\sigma \ i \ j \models f1) = (\sigma \ i \ j \models f2))$
by (*metis ciand-defs cieqv-d-def semantics-introspective-nl-itl.simps(3)*)

lemma *itrue-defs* [*simp*]:
 $(\sigma \ i \ j \models \text{itrue})$
by (*simp add: itrue-d-def*)

lemma *cittrue-defs* [*simp*]:
 $(\sigma \ i \ j \models \text{cittrue})$
by (*simp add: cittrue-d-def*)

lemma *empty-defs* [*simp*]:
assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{empty}) = (i = j)$
using *assms*
by (*auto simp add: empty-d-def more-d-def*)
 $(\text{metis eint-bot eint-cases eint-less-eq(2) nle-le})$

lemma *cempty-defs* [*simp*]:
assumes $i \leq j$

$i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{empty}) = (i = j)$
using *assms*
by (*auto simp add: empty-d-def cmore-d-def*)
(metis eint-bot eint-cases eint-less-eq(2) nle-le)

lemma *skip-defs* [*simp*]:
assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{skip}) = (i + (\text{eint } 1) = j)$
proof –
have 1: $(\sigma \ i \ j \models \text{skip}) = (\sigma \ i \ j \models \text{next empty})$
unfolding *skip-d-def* **by** *simp*
have 2: $(\sigma \ i \ j \models \text{next empty}) = (i \neq -\infty \wedge i < j \wedge (\sigma \ (i + (\text{eint } 1)) \ j \models \text{empty}))$
by *simp*
have 3: $i \neq -\infty \wedge i < j \longrightarrow (i + (\text{eint } 1)) \leq j$
by (*metis abs-eint.cases assms(2) eint-less-le-alt plus-eint.simps(1)*)
have 4: $i + \text{eint } (1::\text{int}) \neq \infty$
by (*simp add: assms(2)*)
have 5: $(i \neq -\infty \wedge i < j \wedge (\sigma \ (i + (\text{eint } 1)) \ j \models \text{empty})) \longrightarrow ((i + (\text{eint } 1)) = j)$
using *assms empty-defs[of i+(eint 1) j] 3 4* **by** *blast*
have 6: $(i + (\text{eint } 1) = j) \longrightarrow (i \neq -\infty \wedge i < j \wedge (\sigma \ (i + (\text{eint } 1)) \ j \models \text{empty}))$
using *assms empty-defs[of i+(eint 1) j]*
by (*metis 4 eint-0-less-1 eint-between(2) eint-infinity-cases less-eq-eint-def one-eint-def plus-eint.simps(5)*)
show ?thesis **using** 1 2 5 6 **by** *blast*
qed

lemma *cskip-defs* [*simp*]:
assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{cskip}) = (i + (\text{eint } 1) = j)$
proof –
have 1: $(\sigma \ i \ j \models \text{cskip}) = (\sigma \ i \ j \models \text{cnext empty})$
unfolding *cskip-d-def* **by** *simp*
have 2: $(\sigma \ i \ j \models \text{cnext empty}) = (i \neq -\infty \wedge i < j \wedge (\sigma \ (i + (\text{eint } 1)) \ j \models \text{empty}))$
by *simp*
have 3: $i \neq -\infty \wedge i < j \longrightarrow (i + (\text{eint } 1)) \leq j$
by (*metis abs-eint.cases assms(2) eint-less-le-alt plus-eint.simps(1)*)
have 4: $i + \text{eint } (1::\text{int}) \neq \infty$
by (*simp add: assms(2)*)
have 5: $(i \neq -\infty \wedge i < j \wedge (\sigma \ (i + (\text{eint } 1)) \ j \models \text{empty})) \longrightarrow ((i + (\text{eint } 1)) = j)$
using *assms empty-defs[of i+(eint 1) j] 3 4* **by** *blast*
have 6: $(i + (\text{eint } 1) = j) \longrightarrow (i \neq -\infty \wedge i < j \wedge (\sigma \ (i + (\text{eint } 1)) \ j \models \text{empty}))$
using *assms empty-defs[of i+(eint 1) j]*
by (*metis 4 eint-0-less-1 eint-between(2) eint-infinity-cases less-eq-eint-def one-eint-def plus-eint.simps(5)*)

show *?thesis* **using** 1 2 5 6 **by** *blast*
qed

lemma *lfinite-defs[simp]*:

assumes $i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models \text{lfinite}) = (i \neq -\infty)$

using *assms*

by (*auto simp add: empty-d-def more-d-def lfinite-d-def*)
(metis eint-bot nle-le)

lemma *clfinite-defs*:

assumes $i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models \text{clfinite}) = (i \neq -\infty)$

using *assms*

by (*auto simp add: cempty-d-def cmore-d-def clfinite-d-def*)
(metis eint-bot nle-le)

lemma *rfinite-defs*:

assumes $i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models \text{rfinite}) = (j \neq \infty)$

using *assms*

by (*auto simp add: empty-d-def more-d-def rfinite-d-def*)
(metis eint-cases eint-less-eq(3) nle-le)

lemma *crfinite-defs*:

assumes $i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models \text{crfinite}) = (j \neq \infty)$

using *assms*

by (*auto simp add: cempty-d-def cmore-d-def crfinite-d-def*)
(metis eint-cases eint-less-eq(3) nle-le)

lemma *prev-defs*:

assumes $i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models \text{prev } f) = (j \neq \infty \wedge i < j \wedge (\sigma \ i \ (j - (\text{eint } 1)) \models f))$

proof –

have 1: $(\sigma \ i \ j \models \text{prev } f) =$

$(\exists k. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ i \ (\text{eint } k) \models f) \wedge (\sigma \ (\text{eint } k) \ j \models \text{skip}))$

unfolding *prev-d-def* **by** *simp*

have 2: $(\exists k. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ i \ (\text{eint } k) \models f) \wedge (\sigma \ (\text{eint } k) \ j \models \text{skip})) \longrightarrow$

$(j \neq \infty \wedge i < j \wedge (\sigma \ i \ (j - (eint \ 1)) \models f))$
using *assms(3) le-eint-less* **by** *auto*
have 3: $(j \neq \infty \wedge i < j \wedge (\sigma \ i \ (j - (eint \ 1)) \models f)) \longrightarrow (\exists k. j = (eint \ k))$
by *(metis less-eint.elims(2))*
have 4: $(j \neq \infty \wedge i < j \wedge (\sigma \ i \ (j - (eint \ 1)) \models f)) \longrightarrow$
 $i \leq eint \ (int-of-eint \ (j-1))$
using *assms 3 using int-less-eint-iff int-of-eint-minus* **by** *fastforce*
have 5: $(j \neq \infty \wedge i < j \wedge (\sigma \ i \ (j - (eint \ 1)) \models f)) \longrightarrow$
 $eint \ (int-of-eint \ (j-1)) \leq j \wedge$
 $(\sigma \ i \ (eint \ (int-of-eint \ (j-1)))) \models f \wedge (\sigma \ (eint \ (int-of-eint \ (j-1))) \ j \models skip)$
using *assms 3* **by** *(auto simp add: one-eint-def)*
have 6: $(j \neq \infty \wedge i < j \wedge (\sigma \ i \ (j - (eint \ 1)) \models f)) \longrightarrow$
 $(\exists k. i \leq eint \ k \wedge eint \ k \leq j \wedge (\sigma \ i \ (eint \ k) \models f) \wedge (\sigma \ (eint \ k) \ j \models skip))$
using 4 5 **by** *fastforce*
show ?thesis **using** 2 6 1 **by** *blast*
qed

lemma *cprev-defs*:

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \ c \models cprev \ f) = (j \neq \infty \wedge i < j \wedge (\sigma \ i \ (j - (eint \ 1)) \ c \models f))$
proof –
have 1: $(\sigma \ i \ j \ c \models cprev \ f) =$
 $(\exists k. i \leq eint \ k \wedge eint \ k \leq j \wedge (\sigma \ i \ (eint \ k) \ c \models f) \wedge (\sigma \ (eint \ k) \ j \ c \models cskip))$
unfolding *cprev-d-def* **by** *simp*
have 2: $(\exists k. i \leq eint \ k \wedge eint \ k \leq j \wedge (\sigma \ i \ (eint \ k) \ c \models f) \wedge (\sigma \ (eint \ k) \ j \ c \models cskip)) \longrightarrow$
 $(j \neq \infty \wedge i < j \wedge (\sigma \ i \ (j - (eint \ 1)) \ c \models f))$
using *assms(3) le-eint-less* **by** *auto*
have 3: $(j \neq \infty \wedge i < j \wedge (\sigma \ i \ (j - (eint \ 1)) \ c \models f)) \longrightarrow (\exists k. j = (eint \ k))$
by *(metis less-eint.elims(2))*
have 4: $(j \neq \infty \wedge i < j \wedge (\sigma \ i \ (j - (eint \ 1)) \ c \models f)) \longrightarrow$
 $i \leq eint \ (int-of-eint \ (j-1))$
using *assms 3 using int-less-eint-iff int-of-eint-minus* **by** *fastforce*
have 5: $(j \neq \infty \wedge i < j \wedge (\sigma \ i \ (j - (eint \ 1)) \ c \models f)) \longrightarrow$
 $eint \ (int-of-eint \ (j-1)) \leq j \wedge$
 $(\sigma \ i \ (eint \ (int-of-eint \ (j-1)))) \ c \models f \wedge (\sigma \ (eint \ (int-of-eint \ (j-1))) \ j \ c \models cskip)$
using *assms 3* **by** *(auto simp add: one-eint-def)*
have 6: $(j \neq \infty \wedge i < j \wedge (\sigma \ i \ (j - (eint \ 1)) \ c \models f)) \longrightarrow$
 $(\exists k. i \leq eint \ k \wedge eint \ k \leq j \wedge (\sigma \ i \ (eint \ k) \ c \models f) \wedge (\sigma \ (eint \ k) \ j \ c \models cskip))$
using 4 5 **by** *fastforce*
show ?thesis **using** 2 6 1 **by** *blast*
qed

lemma *wnext-defs* :

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$

shows $(\sigma \ i \ j \models \text{wnext } f) = (i = -\infty \vee i = j \vee (i < j \wedge (\sigma \ (i + \text{eint } 1) \ j \models f)))$
proof –
have 2: $(\sigma \ i \ j \models \text{inot } (\text{next inot } f)) = (\neg(\sigma \ i \ j \models (\text{next inot } f)))$
using *inot-defs* **by** *blast*
have 3: $(\sigma \ i \ j \models (\text{next inot } f)) = (i \neq -\infty \wedge i < j \wedge \neg(\sigma \ (i + \text{eint } 1) \ j \models f))$
by *simp*
have 4: $(\neg(\sigma \ i \ j \models (\text{next inot } f))) =$
 $(\neg(i \neq -\infty \wedge i < j \wedge \neg(\sigma \ (i + \text{eint } 1) \ j \models f)))$
using 3 **by** *blast*
have 5: $(\neg(i \neq -\infty \wedge i < j \wedge \neg(\sigma \ (i + \text{eint } 1) \ j \models f))) =$
 $(i = -\infty \vee i = j \vee (i < j \wedge (\sigma \ (i + \text{eint } 1) \ j \models f)))$
using *assms*(1) **by** *auto*
show ?thesis **using** 2 3 5 *wnext-d-def* **by** *presburger*
qed

lemma *cwnext-defs* :

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{cwnext } f) = (i = -\infty \vee i = j \vee (i < j \wedge (\sigma \ (i + \text{eint } 1) \ j \models f)))$
proof –
have 2: $(\sigma \ i \ j \models \text{cinot } (\text{cnext cinot } f)) = (\neg(\sigma \ i \ j \models (\text{cnext cinot } f)))$
using *cinot-defs* **by** *blast*
have 3: $(\sigma \ i \ j \models (\text{cnext cinot } f)) = (i \neq -\infty \wedge i < j \wedge \neg(\sigma \ (i + \text{eint } 1) \ j \models f))$
by *simp*
have 4: $(\neg(\sigma \ i \ j \models (\text{cnext cinot } f))) =$
 $(\neg(i \neq -\infty \wedge i < j \wedge \neg(\sigma \ (i + \text{eint } 1) \ j \models f)))$
using 3 **by** *blast*
have 5: $(\neg(i \neq -\infty \wedge i < j \wedge \neg(\sigma \ (i + \text{eint } 1) \ j \models f))) =$
 $(i = -\infty \vee i = j \vee (i < j \wedge (\sigma \ (i + \text{eint } 1) \ j \models f)))$
using *assms*(1) **by** *auto*
show ?thesis **using** 2 3 5 *cwnext-d-def* **by** *presburger*
qed

lemma *wprev-defs*:

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{wprev } f) = (j = \infty \vee i = j \vee (i < j \wedge (\sigma \ i \ (j - (\text{eint } 1)) \models f)))$
proof –
have 1: $(\sigma \ i \ j \models \text{wprev } f) = (\sigma \ i \ j \models \text{inot } (\text{prev inot } f))$
unfolding *wprev-d-def* **by** *blast*
have 2: $(\sigma \ i \ j \models \text{inot } (\text{prev inot } f)) = (\neg(\sigma \ i \ j \models \text{prev inot } f))$
using *assms* *inot-defs*[of $\sigma \ i \ j \text{ prev inot } f$] **by** *blast*
have 3: $(\sigma \ i \ j \models \text{prev inot } f) = (j \neq \infty \wedge i < j \wedge (\sigma \ i \ (j - \text{eint } 1) \models \text{inot } f))$
using *assms* *prev-defs*[of $i \ j \ \sigma \ \text{inot } f$] **by** *blast*
have 4: $(j \neq \infty \wedge i < j \wedge (\sigma \ i \ (j - \text{eint } 1) \models \text{inot } f)) =$
 $(j \neq \infty \wedge i < j \wedge \neg(\sigma \ i \ (j - \text{eint } 1) \models f))$
by *simp*
have 5: $(\neg(\sigma \ i \ j \models \text{prev inot } f)) =$

$(\neg(j \neq \infty \wedge i < j \wedge \neg(\sigma i (j - \text{eint } 1) \models f)))$
using 3 4 **by** *presburger*
have 6: $(\neg(j \neq \infty \wedge i < j \wedge \neg(\sigma i (j - \text{eint } 1) \models f))) =$
 $(j = \infty \vee i = j \vee (i < j \wedge (\sigma i (j - (\text{eint } 1)) \models f)))$
using *assms*(1) *order-less-le* **by** *blast*
from 1 2 5 6 **show** ?thesis
by *presburger*
qed

lemma *cwprev-defs*:

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma i j c \models \text{cwprev } f) = (j = \infty \vee i = j \vee (i < j \wedge (\sigma i (j - (\text{eint } 1)) c \models f)))$
proof –
have 1: $(\sigma i j c \models \text{cwprev } f) = (\sigma i j c \models \text{cnot } (\text{cprev } \text{cnot } f))$
unfolding *cwprev-d-def* **by** *blast*
have 2: $(\sigma i j c \models \text{cnot } (\text{cprev } \text{cnot } f)) = (\neg(\sigma i j c \models \text{cprev } \text{cnot } f))$
using *assms* *cnot-defs*[of $\sigma i j \text{ cprev } \text{cnot } f$] **by** *blast*
have 3: $(\sigma i j c \models \text{cprev } \text{cnot } f) = (j \neq \infty \wedge i < j \wedge (\sigma i (j - \text{eint } 1) c \models \text{cnot } f))$
using *assms* *cprev-defs*[of $i j \sigma \text{ cnot } f$] **by** *blast*
have 4: $(j \neq \infty \wedge i < j \wedge (\sigma i (j - \text{eint } 1) c \models \text{cnot } f)) =$
 $(j \neq \infty \wedge i < j \wedge \neg(\sigma i (j - \text{eint } 1) c \models f))$
by *simp*
have 5: $(\neg(\sigma i j c \models \text{cprev } \text{cnot } f)) =$
 $(\neg(j \neq \infty \wedge i < j \wedge \neg(\sigma i (j - \text{eint } 1) c \models f)))$
using 3 4 **by** *presburger*
have 6: $(\neg(j \neq \infty \wedge i < j \wedge \neg(\sigma i (j - \text{eint } 1) c \models f))) =$
 $(j = \infty \vee i = j \vee (i < j \wedge (\sigma i (j - (\text{eint } 1)) c \models f)))$
using *assms*(1) *order-less-le* **by** *blast*
from 1 2 5 6 **show** ?thesis
by *presburger*
qed

lemma *not-atom-defs*:

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma i j \models \text{inot } (\$p)) = (i = -\infty \vee (p \notin (\sigma (\text{int-of-eint } i))))$
using *assms* **by** *simp*

lemma *not-catom-defs*:

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma i j c \models \text{cnot } (\$p)) = (i = -\infty \vee (p \notin (\sigma (\text{int-of-eint } i))))$
using *assms* **by** *simp*

lemma *diamond-defs*:

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{diamond } f) = (\exists k. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ (\text{eint } k) \ j \models f))$
using *assms unfolding sometimes-d-def by simp*

lemma *cdiamond-defs:*

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{cdiamond } f) = (\exists k. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ (\text{eint } k) \ j \models f))$
using *assms unfolding csometimes-d-def by simp*

lemma *box-defs:*

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{box } f) = (\forall k. i \leq \text{eint } k \wedge \text{eint } k \leq j \longrightarrow (\sigma \ (\text{eint } k) \ j \models f))$
unfolding *always-d-def sometimes-d-def using assms by auto*

lemma *cbox-defs:*

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{cbox } f) = (\forall k. i \leq \text{eint } k \wedge \text{eint } k \leq j \longrightarrow (\sigma \ (\text{eint } k) \ j \models f))$
unfolding *calways-d-def csometimes-d-def using assms by auto*

lemma *di-defs:*

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{di } f) = (\exists k. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ i \ (\text{eint } k) \models f))$
using *assms unfolding di-d-def by simp*

lemma *cdi-defs:*

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{cdi } f) = (\exists k. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ i \ (\text{eint } k) \models f))$
using *assms unfolding cdi-d-def by simp*

lemma *bi-defs:*

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{bi } f) = (\forall k. i \leq \text{eint } k \wedge \text{eint } k \leq j \longrightarrow (\sigma \ i \ (\text{eint } k) \models f))$
unfolding *bi-d-def di-d-def using assms by auto*

lemma *cbi-defs:*

assumes $i \leq j$

$i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \ c \models cbi \ f) = (\forall k. \ i \leq \text{eint } k \wedge \text{eint } k \leq j \longrightarrow (\sigma \ i \ (\text{eint } k) \ c \models f))$
unfolding *cbi-d-def cdi-d-def* **using** *assms* **by** *auto*

lemma *init-defs*:

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{init } f) = (i \neq -\infty \wedge (\sigma \ i \ i \models f))$
proof –
have 1: $(\sigma \ i \ j \models \text{init } f) =$
 $(\exists k::\text{int}. \ i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ i \ (\text{eint } k) \models \text{empty}) \wedge (\sigma \ i \ (\text{eint } k) \models f))$
unfolding *init-d-def* **by** *simp*
have 2: $(\exists k::\text{int}. \ i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ i \ (\text{eint } k) \models \text{empty}) \wedge (\sigma \ i \ (\text{eint } k) \models f))$
 $\longrightarrow (i \neq \infty \wedge (\sigma \ i \ i \models f))$
using *assms*(2) **by** *force*
have 3: $i \neq -\infty \wedge (\sigma \ i \ i \models f) \longrightarrow (\exists k. \ i = (\text{eint } k))$
by (*meson* *assms*(2) *int-of-eint.elims*)
have 4: $i \neq -\infty \wedge (\sigma \ i \ i \models f) \longrightarrow$
 $i \leq \text{eint } (\text{int-of-eint } i) \wedge \text{eint } (\text{int-of-eint } i) \leq j \wedge$
 $(\sigma \ i \ (\text{eint } (\text{int-of-eint } i)) \models \text{empty}) \wedge (\sigma \ i \ (\text{eint } (\text{int-of-eint } i)) \models f)$
using *assms* 3 **by** *fastforce*
have 5: $i \neq -\infty \wedge (\sigma \ i \ i \models f) \longrightarrow$
 $(\exists k::\text{int}. \ i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ i \ (\text{eint } k) \models \text{empty}) \wedge (\sigma \ i \ (\text{eint } k) \models f))$
using 4 **by** *fastforce*
show *?thesis* **using** 1 2 4 **by** *auto*
qed

lemma *cinit-defs*:

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \ c \models \text{cinit } f) = (i \neq -\infty \wedge (\sigma \ i \ i \ c \models f))$
proof –
have 1: $(\sigma \ i \ j \ c \models \text{cinit } f) =$
 $(\exists k::\text{int}. \ i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ i \ (\text{eint } k) \ c \models \text{empty}) \wedge (\sigma \ i \ (\text{eint } k) \ c \models f))$
unfolding *cinit-d-def* **by** *simp*
have 2: $(\exists k::\text{int}. \ i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ i \ (\text{eint } k) \ c \models \text{empty}) \wedge (\sigma \ i \ (\text{eint } k) \ c \models f))$
 $\longrightarrow (i \neq \infty \wedge (\sigma \ i \ i \ c \models f))$
using *assms*(2) **by** *force*
have 3: $i \neq -\infty \wedge (\sigma \ i \ i \ c \models f) \longrightarrow (\exists k. \ i = (\text{eint } k))$
by (*meson* *assms*(2) *int-of-eint.elims*)
have 4: $i \neq -\infty \wedge (\sigma \ i \ i \ c \models f) \longrightarrow$
 $i \leq \text{eint } (\text{int-of-eint } i) \wedge \text{eint } (\text{int-of-eint } i) \leq j \wedge$
 $(\sigma \ i \ (\text{eint } (\text{int-of-eint } i)) \ c \models \text{empty}) \wedge (\sigma \ i \ (\text{eint } (\text{int-of-eint } i)) \ c \models f)$
using *assms* 3 **by** *fastforce*
have 5: $i \neq -\infty \wedge (\sigma \ i \ i \ c \models f) \longrightarrow$
 $(\exists k::\text{int}. \ i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ i \ (\text{eint } k) \ c \models \text{empty}) \wedge (\sigma \ i \ (\text{eint } k) \ c \models f))$
using 4 **by** *fastforce*

show *?thesis* **using** 1 2 4 **by** *auto*
qed

lemma *fin-defs*:

assumes $i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models \text{fin } f) = (j \neq \infty \wedge (\sigma \ j \ j \models f))$

proof –

have 1: $(\sigma \ i \ j \models \text{fin } f) =$

$(\exists k::\text{int}. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ (\text{eint } k) \ j \models \text{empty}) \wedge (\sigma \ (\text{eint } k) \ j \models f))$

unfolding *fin-d-def* **by** *simp*

have 2: $(\exists k::\text{int}. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ (\text{eint } k) \ j \models \text{empty}) \wedge (\sigma \ (\text{eint } k) \ j \models f)) \longrightarrow$
 $(j \neq \infty \wedge (\sigma \ j \ j \models f))$

using *PInfty-neq-eint(1)* *assms(3)* *empty-defs* **by** *blast*

have 3: $(j \neq \infty \wedge (\sigma \ j \ j \models f)) \longrightarrow (\exists k. j = \text{eint } k)$

using *assms* **by** (*metis* *MInfty-eq-minfinity* *PInfty-eq-infinity* *eint.exhaust*)

have 4: $(j \neq \infty \wedge (\sigma \ j \ j \models f)) \longrightarrow$

$i \leq \text{eint } (\text{int-of-eint } j) \wedge \text{eint } (\text{int-of-eint } j) \leq j \wedge$

$(\sigma \ (\text{eint } (\text{int-of-eint } j)) \ j \models \text{empty}) \wedge (\sigma \ (\text{eint } (\text{int-of-eint } j)) \ j \models f)$

using *assms 3* **by** *fastforce*

show *?thesis* **using** 1 2 4 **by** *auto*

qed

lemma *cfin-defs*:

assumes $i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models \text{cfin } f) = (j \neq \infty \wedge (\sigma \ j \ j \models f))$

proof –

have 1: $(\sigma \ i \ j \models \text{cfin } f) =$

$(\exists k::\text{int}. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ (\text{eint } k) \ j \models \text{empty}) \wedge (\sigma \ (\text{eint } k) \ j \models f))$

unfolding *cfin-d-def* **by** *simp*

have 2: $(\exists k::\text{int}. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ (\text{eint } k) \ j \models \text{empty}) \wedge (\sigma \ (\text{eint } k) \ j \models f)) \longrightarrow$
 $(j \neq \infty \wedge (\sigma \ j \ j \models f))$

using *PInfty-neq-eint(1)* *assms(3)* *empty-defs* **by** *blast*

have 3: $(j \neq \infty \wedge (\sigma \ j \ j \models f)) \longrightarrow (\exists k. j = \text{eint } k)$

using *assms* **by** (*metis* *MInfty-eq-minfinity* *PInfty-eq-infinity* *eint.exhaust*)

have 4: $(j \neq \infty \wedge (\sigma \ j \ j \models f)) \longrightarrow$

$i \leq \text{eint } (\text{int-of-eint } j) \wedge \text{eint } (\text{int-of-eint } j) \leq j \wedge$

$(\sigma \ (\text{eint } (\text{int-of-eint } j)) \ j \models \text{empty}) \wedge (\sigma \ (\text{eint } (\text{int-of-eint } j)) \ j \models f)$

using *assms 3* **by** *fastforce*

show *?thesis* **using** 1 2 4 **by** *auto*

qed

lemma *state-nl-itl-defs*:

assumes *state-nl-itl w*

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models w) \longleftrightarrow (\sigma \ i \ i \models w)$
 using *assms*
 by (*induct w arbitrary: σ*) *simp-all*

lemma *next-diamond*:

assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$

shows $(\sigma \ i \ j \models \text{next} \ (\text{diamond} \ f)) = (\sigma \ i \ j \models (\text{lfinite} \ i \ \text{and} \ \text{diamond} \ (\text{next} \ f)))$

proof –

have 1: $(\sigma \ i \ j \models \text{next} \ (\text{diamond} \ f)) = (i \neq -\infty \wedge i < j \wedge (\sigma \ (i + \text{eint} \ 1) \ j \models \text{diamond} \ f))$
 by *simp*

have 2: $(i \neq -\infty \wedge i < j \wedge (\sigma \ (i + \text{eint} \ 1) \ j \models \text{diamond} \ f)) =$
 $(i \neq -\infty \wedge i < j \wedge (\exists k. i + \text{eint} \ 1 \leq \text{eint} \ k \wedge \text{eint} \ k \leq j \wedge (\sigma \ (\text{eint} \ k) \ j \models f)))$

using *diamond-defs[of (i + eint 1) j σ f]*

using *assms(3) less-eint.elims(2)* by *fastforce*

have 30: $(i \neq -\infty \wedge i < j \wedge (\exists k. i + \text{eint} \ 1 \leq \text{eint} \ k \wedge \text{eint} \ k \leq j \wedge (\sigma \ (\text{eint} \ k) \ j \models f))) \implies$
 $(i \neq -\infty \wedge (\exists k. i \leq \text{eint} \ k \wedge \text{eint} \ k \leq j \wedge \text{eint} \ k < j \wedge (\sigma \ (\text{eint} \ (k + 1)) \ j \models f)))$

proof –

assume *a0*: $(i \neq -\infty \wedge i < j \wedge (\exists k. i + \text{eint} \ 1 \leq \text{eint} \ k \wedge \text{eint} \ k \leq j \wedge (\sigma \ (\text{eint} \ k) \ j \models f)))$

show $(i \neq -\infty \wedge (\exists k. i \leq \text{eint} \ k \wedge \text{eint} \ k \leq j \wedge \text{eint} \ k < j \wedge (\sigma \ (\text{eint} \ (k + 1)) \ j \models f)))$

proof –

obtain *k* where 31: $i + \text{eint} \ 1 \leq \text{eint} \ k \wedge \text{eint} \ k \leq j \wedge (\sigma \ (\text{eint} \ k) \ j \models f)$

using *a0* by *blast*

have 32: $i \leq \text{eint} \ (k-1)$

using 31 *assms(2) less-eint.elims(2)* by *fastforce*

have 33: $\text{eint} \ (k-1) \leq j \wedge \text{eint} \ (k-1) < j$

by (*metis 31 diff-add-cancel eint-le-less less-add-one order-less-imp-le*)

have 34: $(\sigma \ (\text{eint} \ ((k-1) + 1)) \ j \models f)$

by (*simp add: 31*)

show *?thesis*

using 32 33 34 *a0* by *blast*

qed

qed

have 35: $(i \neq -\infty \wedge (\exists k. i \leq \text{eint} \ k \wedge \text{eint} \ k \leq j \wedge \text{eint} \ k < j \wedge (\sigma \ (\text{eint} \ (k + 1)) \ j \models f))) \implies$
 $(i \neq -\infty \wedge i < j \wedge (\exists k. i + \text{eint} \ 1 \leq \text{eint} \ k \wedge \text{eint} \ k \leq j \wedge (\sigma \ (\text{eint} \ k) \ j \models f)))$

proof –

assume *a1*: $(i \neq -\infty \wedge (\exists k. i \leq \text{eint} \ k \wedge \text{eint} \ k \leq j \wedge \text{eint} \ k < j \wedge (\sigma \ (\text{eint} \ (k + 1)) \ j \models f)))$

show $(i \neq -\infty \wedge i < j \wedge (\exists k. i + \text{eint} \ 1 \leq \text{eint} \ k \wedge \text{eint} \ k \leq j \wedge (\sigma \ (\text{eint} \ k) \ j \models f)))$

proof –

obtain *k* where 36: $i \leq \text{eint} \ k \wedge \text{eint} \ k \leq j \wedge \text{eint} \ k < j \wedge (\sigma \ (\text{eint} \ (k + 1)) \ j \models f)$

using *a1* by *blast*

have 37: $i + \text{eint} \ 1 \leq \text{eint} \ (k+1)$

by (*metis 36 add.commute add-mono eint-eq-1(1) eint-plus-1(1) order-refl*)

have 38: $\text{eint} \ (k+1) \leq j$

by (*simp add: 36 eint-less-le-alt*)

have 39: $(\sigma \ (\text{eint} \ (k+1)) \ j \models f)$

by (*simp add: 36*)

```

have 40: ( $\exists k. i + \text{eint } 1 \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma (\text{eint } k) j \models f)$ )
  using 36 37 38 by auto
have 41:  $i < j$ 
  using a1 by fastforce
have 42:  $i \neq -\infty$ 
  by (simp add: a1)
show ?thesis
  by (simp add: 40 41 42)
qed
qed
have 4: ( $i \neq -\infty \wedge (\exists k. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma (\text{eint } k) j \models \text{next } f))$ ) =
  ( $i \neq -\infty \wedge (\exists k. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge \text{eint } k < j \wedge (\sigma (\text{eint } (k + 1)) j \models f))$ )
  by simp
have 5: ( $\sigma i j \models (\text{lfinite i and diamond (next f)})$ ) =
  ( $i \neq -\infty \wedge (\exists k. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma (\text{eint } k) j \models \text{next } f))$ )
  using diamond-defs[of i j  $\sigma$  next f] lfinite-defs assms by fastforce
show ?thesis
  by (metis 1 2 30 35 4 5)
qed

```

```

lemma linf-defs[simp]:
  assumes  $i \neq \infty$ 
     $j \neq -\infty$ 
     $i \leq j$ 
  shows ( $\sigma i j \models \text{linf}$ ) = ( $i = -\infty$ )
  using assms unfolding linf-d-def by (simp)

```

```

lemma rinf-defs[simp]:
  assumes  $i \neq \infty$ 
     $j \neq -\infty$ 
     $i \leq j$ 
  shows ( $\sigma i j \models \text{rinf}$ ) = ( $j = \infty$ )
  using assms unfolding rinf-d-def by (simp add: rfinite-defs)

```

```

lemma wchop-defs[simp]:
  assumes
     $i \neq \infty$ 
     $j \neq -\infty$ 
     $i \leq j$ 
  shows ( $\sigma i j \models f \text{ wchop } g$ ) =
    ( $(\exists k. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma i (\text{eint } k) \models f) \wedge (\sigma (\text{eint } k) j \models g)) \vee$ 
      ( $(\sigma i j \models f) \wedge j = \infty$ ))
  )
  using assms unfolding wchop-d-def by simp

```

```

lemma clinf-defs[simp]:
  assumes  $i \neq \infty$ 

```

$j \neq -\infty$
 $i \leq j$
shows $(\sigma \ i \ j \ c \models \text{clinf}) = (i = -\infty)$
using *assms* **unfolding** *clinf-d-def* **by** (*simp add: clfinite-defs*)

lemma *crinf-defs[simp]*:
assumes $i \neq \infty$
 $j \neq -\infty$
 $i \leq j$
shows $(\sigma \ i \ j \ c \models \text{crinf}) = (j = \infty)$
using *assms* **unfolding** *crinf-d-def* **by** (*simp add: crfinite-defs*)

lemma *cwchop-defs[simp]*:
assumes
 $i \neq \infty$
 $j \neq -\infty$
 $i \leq j$
shows $(\sigma \ i \ j \ c \models f \text{ cwchop } g) =$
 $((\exists k. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma \ i \ (\text{eint } k) \ c \models f) \wedge (\sigma \ (\text{eint } k) \ j \ c \models g)) \vee$
 $((\sigma \ i \ j \ c \models f) \wedge j = \infty))$
 $)$
using *assms* **unfolding** *cwchop-d-def* **by** *simp*

definition *agree-on* :: *interval* \Rightarrow *interval* \Rightarrow *eint* \Rightarrow *eint* \Rightarrow *bool*
where *agree-on* $\sigma \ \sigma' \ i \ j = (\forall k. i \leq k \wedge k \leq j \longrightarrow (\sigma \ k) = (\sigma' \ k))$

lemma *agree-on-defs[simp]*:
 $\text{agree-on } \sigma \ \sigma' \ i \ j = (\forall k. i \leq k \wedge k \leq j \longrightarrow (\sigma \ k) = (\sigma' \ k))$
unfolding *agree-on-def* **by** *simp*

lemma *agree-on-sub*:
assumes $i \leq j$
 $\text{agree-on } \sigma \ \sigma' \ i \ j$
 $i \leq l0$
 $l0 \leq l1$
 $l1 \leq j$
shows $\text{agree-on } \sigma \ \sigma' \ l0 \ l1$
using *assms*
using *order-trans* **by** *auto*

lemma *nridx-eint-gr-first*:
fixes $i :: \text{eint}$
assumes *nfinite* l
 $i \neq -\infty$
 $i \neq \infty$
 $(\text{nnth } l \ 0) = i$
 $\text{nridx } (<) \ l$
 $(\text{enat } k) \leq \text{nlength } l$

```

shows     $i \leq (\text{nnth } l \ k)$ 
using assms
 $\text{nridx-gr-first}[\text{of } \lambda (a :: \text{eint}) \ b. \ a < b \ l \ k]$ 
using order-le-less by auto

lemma nridx-eint-less-last:
fixes  $j :: \text{eint}$ 
assumes nfinite l
 $j \neq -\infty$ 
 $j \neq \infty$ 
 $\text{nlast } l = j$ 
 $\text{nridx } (<) \ l$ 
 $(\text{enat } k) < \text{nlength } l$ 
shows     $(\text{nnth } l \ (\text{Suc } k)) \leq j$ 
using assms
 $\text{nridx-less-last}[\text{of } \lambda (a :: \text{eint}) \ b. \ a < b \ l \ k \ \text{the-enat } (\text{nlength } l)]$ 
apply simp
by (metis Suc-lessI enat-ord-simps(2) less-eq-eint-def less-imp-Suc-add nfinite-conv-nlength-enat
 $\text{nle-le nnth-nlast nridx-less the-enat.simps transp-on-less}$ )

```

```

lemma
assumes  $i \leq j$ 
 $i \neq \infty$ 
 $j \neq -\infty$ 
 $(\sigma \ i \ j \ c \models f)$ 
 $\text{agree-on } \sigma \ \sigma' \ i \ j$ 
shows  $(\sigma' \ i \ j \ c \models f)$ 
using assms
proof (induction f arbitrary:  $\sigma \ \sigma' \ i \ j$ )
case cfalse-d
then show ?case by simp
next
case (catom-d x)
then show ?case apply simp
by (metis eint-infty-less(2) int-of-eint.simps(1) less-eint.elims(2) order-refl)
next
case (ciimp-d f1 f2)
then show ?case apply simp by metis
next
case (cnext-d f)
then show ?case
proof –
have c1:  $(\sigma' \ i \ j \ c \models \text{cnext } f) \longleftrightarrow (i \neq -\infty \wedge i < j \wedge (\sigma' \ (i + \text{eint } 1) \ j \ c \models f))$ 
by simp
have c2:  $(i \neq -\infty \wedge i < j \wedge (\sigma \ (i + \text{eint } 1) \ j \ c \models f))$ 
using cnext-d by simp
have c3:  $\forall k. \ i \leq \text{eint } k \wedge \text{eint } k \leq j \longrightarrow \sigma \ k = \sigma' \ k$ 
using cnext-d by auto

```

```

have c4: agree-on  $\sigma \sigma' (i + \text{eint } 1) j$ 
  unfolding agree-on-def
  by (metis MInfty-neq-eint(2) PInfty-neq-eint(1) add.right-neutral add-increasing2 c3 eint-0-plus
    eint-add-le-add-iff2 eint-less-eq(3) zero-less-one-class.zero-le-one)
have c5:  $(i \neq -\infty \wedge i < j \wedge (\sigma (i + \text{eint } 1) j \models f))$ 
  using cnext-d.IH using c2 cnext-d.prem(4) by blast
have c6:  $i + \text{eint } 1 \leq j$ 
  by (metis c5 cnext-d.prem(2) eint-less-le-alt int-of-eint.cases plus-eint.simp(1))
have c7:  $i + \text{eint } 1 \neq \infty$ 
  by (simp add: cnext-d.prem(2))
have c8:  $(\sigma' (i + \text{eint } 1) j \models f)$ 
  using cnext-d.IH[of  $(i + \text{eint } 1) j \sigma \sigma'$ ]
  using c4 c5 c6 c7 cnext-d.prem(3) by blast
show ?thesis by (simp add: c5 c8)
qed
next
case (cchop-d f1 f2)
then show ?case
  proof -
    have 1:  $(\exists k. i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma i (\text{eint } k) \models f1) \wedge (\sigma (\text{eint } k) j \models f2))$ 
      using cchop-d.prem(4) by auto
    obtain k where 2:  $i \leq \text{eint } k \wedge \text{eint } k \leq j \wedge (\sigma i (\text{eint } k) \models f1) \wedge (\sigma (\text{eint } k) j \models f2)$ 
      using 1 by auto
    have 3: agree-on  $\sigma \sigma' i k$ 
      by (meson 2 agree-on-defs cchop-d.prem(5) order-trans)
    have 4: agree-on  $\sigma \sigma' k j$ 
      by (meson 2 agree-on-defs cchop-d.prem(5) order-trans)
    have 5:  $\sigma' i k \models f1$ 
      by (metis 2 3 MInfty-neq-eint(2) cchop-d.IH(1) cchop-d.prem(2))
    have 6:  $\sigma' k j \models f2$ 
      using 2 4 PInfty-neq-eint(1) cchop-d.IH(2) cchop-d.prem(3) by blast
    have 7:  $\sigma' i j \models f1$  cschop f2
      using 5 6 2 semantics-introspective-nl-itl.simp(5) by blast
    show ?thesis
      using 7 by blast
  qed
next
case (cchopstar-d f)
then show ?case
  proof (cases  $i=j$ )
    case True
    then show ?thesis
      using semantics-introspective-nl-itl.simp(6) by blast
    next
    case False
    then show ?thesis
      proof (cases  $i \neq -\infty \wedge j \neq \infty$ )
        case True
        then show ?thesis
          proof -

```

```

have 1:  $\sigma \ i \ j \ c \models cchopstar \ f$ 
  using cchopstar-d.prems by blast
have 2:  $(\exists (lrf :: eint \ nelist)).$ 
 $(nnth \ lrf \ 0) = i \wedge nfinite \ lrf \wedge nlast \ lrf = j \wedge$ 
 $nridx \ (\lambda \ a \ b. \ a < b \wedge (\sigma \ a \ b \ c \models f)) \ lrf$ 
)
  using  $\langle i \neq -\infty \wedge j \neq \infty \rangle \langle i \neq j \rangle$  cchopstar-d.prems by simp
obtain lrf where 3:  $(nnth \ lrf \ 0) = i \wedge nfinite \ lrf \wedge nlast \ lrf = j \wedge$ 
 $nridx \ (\lambda \ a \ b. \ a < b \wedge (\sigma \ a \ b \ c \models f)) \ lrf$ 
  using 2 by blast
have 4:  $(\forall (k :: nat) . ( (enat \ k) < (nlength \ lrf)) \longrightarrow$ 
 $(\sigma \ (nnth \ lrf \ k) \ (nnth \ lrf \ (Suc \ k)) \ c \models f) )$ 
  by (metis (no-types, lifting) 3 eSuc-enat ileI1 nridx-expand)
have 5:  $\bigwedge k. ( (enat \ k) < (nlength \ lrf)) \longrightarrow (nnth \ lrf \ k) \leq (nnth \ lrf \ (Suc \ k))$ 
  by (metis (no-types, lifting) 3 eSuc-enat ileI1 nridx-expand order-less-imp-le)
have 51:  $nridx \ (<) \ lrf$ 
  by (metis (no-types, lifting) 3 nridx-expand)
have 6:  $\bigwedge k. ( (enat \ k) < (nlength \ lrf)) \longrightarrow i \leq (nnth \ lrf \ k)$ 
  using 51 nridx-eint-gr-first
  by (simp add: 3 True cchopstar-d.prems(2))
have 7:  $\bigwedge k. ( (enat \ k) < (nlength \ lrf)) \longrightarrow (nnth \ lrf \ (Suc \ k)) \leq j$ 
  using nridx-eint-less-last
  by (simp add: 3 51 True cchopstar-d.prems(3))
have 8:  $\bigwedge k. ( (enat \ k) < (nlength \ lrf)) \longrightarrow$ 
 $agree-on \ \sigma \ \sigma' \ (nnth \ lrf \ k) \ (nnth \ lrf \ (Suc \ k))$ 
  using agree-on-sub[of i j  $\sigma \ \sigma'$ ]
  by (meson 5 6 7 cchopstar-d.prems(1) cchopstar-d.prems(5))
have 9:  $\bigwedge k. ( (enat \ k) < (nlength \ lrf)) \longrightarrow$ 
 $(\sigma' \ (nnth \ lrf \ k) \ (nnth \ lrf \ (Suc \ k)) \ c \models f)$ 
  by (metis 4 5 6 7 8 True cchopstar-d.IH eint-infity-less-eq(2) eint-less-eq(1) nle-le)
have 10:  $nridx \ (\lambda \ a \ b. \ a < b \wedge (\sigma' \ a \ b \ c \models f)) \ lrf$ 
  using 9
  by (metis (no-types, lifting) 3 Suc-ile-eq nridx-expand)
show ?thesis
  using 10 3 True by auto
qed
next
case False
then show ?thesis
  proof (cases  $i = -\infty \wedge j \neq \infty$ )
  case True
  then show ?thesis
    proof -
    have b1:  $(\exists (li :: eint \ nelist)).$ 
 $(nnth \ li \ 0) = j \wedge \neg nfinite \ li \wedge nridx \ (\lambda \ a \ b. \ b < a \wedge (\sigma \ b \ a \ c \models f)) \ li$ 
    )
    using  $\langle i = -\infty \wedge j \neq \infty \rangle$  cchopstar-d.prems by simp
    obtain li where b2:  $(nnth \ li \ 0) = j \wedge \neg nfinite \ li \wedge$ 
 $nridx \ (\lambda \ a \ b. \ b < a \wedge (\sigma \ b \ a \ c \models f)) \ li$ 
    using b1 by blast

```

```

have b3: (∀ (k::nat) .
  ( σ (nnth li (Suc k)) (nnth li k)  c⊨ f) )
by (metis (no-types, lifting) b2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand)
have b5: ∧k. (nnth li (Suc k)) ≤(nnth li k)
by (metis (no-types, lifting) b2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand
order-less-imp-le)
have b51: nridx (>) li
by (metis (no-types, lifting) b2 nridx-expand)
have b8: ∧k. agree-on σ σ' (nnth li (Suc k)) (nnth li k)
by (metis (mono-tags, lifting) b5 True agree-on-sub b2 cchopstar-d.prem5 eint-infty-less-eq(2)
lift-Suc-antimono-le nle-le zero-le)
have b9: ∧k.
  ( σ' (nnth li (Suc k)) (nnth li k)  c⊨ f)
by (metis (no-types, lifting) b5 b8 MInfty-neq-eint(1) PInfty-neq-eint(1) True b2 cchopstar-d.IH
cchopstar-d.prem5(2) enat-ile less-eint.elims(2) linorder-le-cases nfinite-conv-nlength-enat nridx-expand)
have b10: nridx (λ a b. a > b ∧ (σ' b a  c⊨ f)) li
using b9
by (metis (no-types, lifting) b2 nridx-expand)
show ?thesis
using b10 True b2 by auto
qed
next
case False
then show ?thesis
proof (cases i ≠ -∞ ∧ j = ∞ )
case True
then show ?thesis
proof -
have c1: (∃ (ri :: eint nellist).
  (nnth ri 0) = i ∧ ¬ nfinite ri  ∧ nridx (λ a b. a < b ∧ (σ a b c⊨ f)) ri
)
using ⟨i ≠ -∞ ∧ j = ∞⟩ cchopstar-d.prem5 by simp
obtain ri where c2: (nnth ri 0) = i ∧ ¬ nfinite ri  ∧ nridx (λ a b. a < b ∧ (σ a b c⊨ f)) ri
using c1 by blast
have c3: (∀ (k::nat) .
  ( σ (nnth ri k) (nnth ri (Suc k))  c⊨ f) )
by (metis (no-types, lifting) c2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand)
have c5: ∧k. (nnth ri k) ≤ (nnth ri (Suc k))
by (metis (no-types, lifting) c2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand
order-less-imp-le)
have c51: nridx (<) ri
by (metis (no-types, lifting) c2 nridx-expand)

have c6: ∧k. i ≤ (nnth ri k)
using c51 nridx-eint-gr-first
using c2 c5 lift-Suc-mono-le by blast
have c8: ∧k. agree-on σ σ' (nnth ri k) (nnth ri (Suc k))
using True agree-on-sub c2 c5 cchopstar-d.prem5(5) eint-less-eq(1) lift-Suc-mono-le by blast
have c9: ∧k.
  ( σ' (nnth ri k) (nnth ri (Suc k))  c⊨ f)

```



```

by (metis (no-types, lifting) PInfty-neq-eint(1) True c2 c5 c6 c8 chopstar-d.IH eint-infty-less-eq(2)
enat-ile less-eint.elims(2) linorder-le-cases nfinite-conv-nlength-enat nridx-expand)
have c10: nridx ( $\lambda a b. a < b \wedge (\sigma' a b \quad c \models f)$ ) ri
using c9
by (metis (no-types, lifting) c2 nridx-expand)
show ?thesis
using True c10 c2 by auto
qed
next
case False
then show ?thesis
proof -
have d0:  $i = -\infty \wedge j = \infty$ 
using  $\langle i \neq j \rangle \langle \neg(i \neq -\infty \wedge j = \infty) \rangle \langle \neg(i = -\infty \wedge j \neq \infty) \rangle \langle \neg(i \neq -\infty \wedge j \neq \infty) \rangle$ 
by blast
have d1:  $(\exists (li :: int\ nelist) (ri :: int\ nelist) .$ 
 $(nnth\ li\ 0) = (nnth\ ri\ 0) \wedge$ 
 $\neg\ nfinite\ li \wedge nridx\ (\lambda a b. b < a \wedge (\sigma\ b\ a\ c \models f))\ li \wedge$ 
 $\neg\ nfinite\ ri \wedge nridx\ (\lambda a b. a < b \wedge (\sigma\ a\ b\ c \models f))\ ri$ 
 $)$ 
using d0 chopstar-d.prem by simp
obtain ri li where d2:  $(nnth\ (li :: int\ nelist)\ 0) = (nnth\ (ri :: int\ nelist)\ 0) \wedge$ 
 $\neg\ nfinite\ li \wedge nridx\ (\lambda a b. b < a \wedge (\sigma\ b\ a\ c \models f))\ li \wedge$ 
 $\neg\ nfinite\ ri \wedge nridx\ (\lambda a b. a < b \wedge (\sigma\ a\ b\ c \models f))\ ri$ 
using d1 by blast
have d3:  $(\forall (k::nat) .$ 
 $(\sigma\ (nnth\ ri\ k)\ (nnth\ ri\ (Suc\ k))\ c \models f))$ 
by (metis (no-types, lifting) d2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand)
have d4:  $(\forall (k::nat) .$ 
 $(\sigma\ (nnth\ li\ (Suc\ k))\ (nnth\ li\ k)\ c \models f))$ 
by (metis (no-types, lifting) d2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand)
have d5:  $\bigwedge k. (nnth\ ri\ k) \leq (nnth\ ri\ (Suc\ k))$ 
by (metis (no-types, lifting) d2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand
order-less-imp-le)
have d50:  $\bigwedge k. (nnth\ li\ (Suc\ k)) \leq (nnth\ li\ k)$ 
by (metis (no-types, lifting) d2 enat-ile linorder-le-cases nfinite-conv-nlength-enat nridx-expand
order-less-imp-le)
have d51: nridx ( $<$ ) ri
by (metis (no-types, lifting) d2 nridx-expand)
have d52: nridx ( $>$ ) li
by (metis (no-types, lifting) d2 nridx-expand)
have d8:  $\bigwedge k. agree-on\ \sigma\ \sigma'\ (nnth\ ri\ k)\ (nnth\ ri\ (Suc\ k))$ 
using chopstar-d.prem(5) d0 by auto
have d80:  $\bigwedge k. agree-on\ \sigma\ \sigma'\ (nnth\ li\ (Suc\ k))\ (nnth\ li\ k)$ 
using chopstar-d.prem(5) d0 by auto
have d9:  $\bigwedge k.$ 
 $(\sigma'\ (nnth\ ri\ k)\ (nnth\ ri\ (Suc\ k))\ c \models f)$ 
by (meson MInfty-neq-eint(1) PInfty-neq-eint(1) chopstar-d.IH d3 d5 d8 eint-less-eq(3))
have d90:  $\bigwedge k.$ 
 $(\sigma'\ (nnth\ li\ (Suc\ k))\ (nnth\ li\ k)\ c \models f)$ 

```

```

    by (metis MInfty-neq-eint(1) PInfty-neq-eint(2) cchopstar-d.IH d4 d50 d80 eint-less-eq(3))
  have d10: nridx ( $\lambda a b. a < b \wedge (\sigma' a b \models f)$ ) ri
    using d9 by (metis (no-types, lifting) d2 nridx-expand)
  have d11: nridx ( $\lambda a b. b < a \wedge (\sigma' b a \models f)$ ) li
    using d90 by (metis (no-types, lifting) d2 nridx-expand)
  show ?thesis
  using d10 d2 d11 d0 by auto
qed
qed
qed
qed
qed
next
case (cerists-d x1 f)
then show ?case
proof -
  have 1: ( $\exists l. (\text{upd } \sigma \ x1 \ l) \ i \ j \models f$ )
    using cerist-sigma-cerist-value cerists-d.prem(4) by auto
  obtain l where 2: ( $\text{upd } \sigma \ x1 \ l) \ i \ j \models f$ 
    using 1 by blast
  have 3: ( $\bigwedge i \ j \ \sigma \ \sigma'. i \leq j \implies i \neq \infty \implies j \neq -\infty \implies$ 
 $\sigma \ i \ j \models f \implies \text{agree-on } \sigma \ \sigma' \ i \ j \implies \sigma' \ i \ j \models f$ )
    using cerists-d.IH by blast
  have 4:  $\sigma \ i \ j \models cEx \ x1. f$ 
    using cerists-d.prem by blast
  have 5:  $\text{agree-on } \sigma \ \sigma' \ i \ j$ 
    using cerists-d.prem by blast
  have 6:  $\text{agree-on } (\text{upd } \sigma \ x1 \ l) (\text{upd } \sigma' \ x1 \ l) \ i \ j$ 
    using 5 upd-def by force
  have 7: ( $\text{upd } \sigma' \ x1 \ l) \ i \ j \models f$ 
    using 2 6 cerists-d.IH cerists-d.prem(1) cerists-d.prem(2) cerists-d.prem(3) by blast
  show ?thesis using 7 cerist-sigma-cerist-value by auto
qed
qed

```

lemma schop-dist-ior:

```

assumes  $i \leq j$ 
         $i \neq \infty$ 
         $j \neq -\infty$ 
shows  $(\sigma \ i \ j \models ((f1 \ ior \ f2) \ schop \ f3) \ ieqv \ (f1 \ schop \ f3 \ ior \ f2 \ schop \ f3))$ 
using assms by auto

```

lemma cschop-dist-ior:

```

assumes  $i \leq j$ 
         $i \neq \infty$ 
         $j \neq -\infty$ 
shows  $(\sigma \ i \ j \models ((f1 \ cior \ f2) \ cschop \ f3) \ cieqv \ (f1 \ cschop \ f3 \ cior \ f2 \ cschop \ f3))$ 
using assms by auto

```

lemma *exists-atom*:

assumes $i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models \text{Ex } n. \$n) = (i \neq -\infty)$

proof –

have 1: $(\sigma \ i \ j \models \text{Ex } n. \$n) \longrightarrow (i \neq -\infty)$

by *simp*

have 2: $(i \neq -\infty) \longrightarrow$

$n \in (\text{upd-add-sigma } \sigma \ (\text{int-of-eint } i) \ n) \ (\text{int-of-eint } i)$

$\wedge (\text{similar } \sigma \ n \ (\text{upd-add-sigma } \sigma \ (\text{int-of-eint } i) \ n))$ **apply** *simp*

by (*metis fun-upd-apply insert-Diff-if similar-defs-alt singletonI*)

have 3: $(i \neq -\infty) \longrightarrow (\exists \sigma'. n \in \sigma' \ (\text{int-of-eint } i) \wedge \text{similar } \sigma \ n \ \sigma')$

using 2 **by** *blast*

have 4: $(i \neq -\infty) \longrightarrow (\sigma \ i \ j \models \text{Ex } n. \$n)$

using 3

by (*meson exist-sigma-exist-value semantics-nl-itl.simps(2) semantics-nl-itl.simps(9)*)

show *?thesis* **using** 1 4 **by** *blast*

qed

lemma *ceexists-catom*:

assumes $i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models \text{cEx } n. \$\$n) = (i \neq -\infty)$

proof –

have 1: $(\sigma \ i \ j \models \text{cEx } n. \$\$n) \longrightarrow (i \neq -\infty)$

by *simp*

have 2: $(i \neq -\infty) \longrightarrow$

$n \in (\text{upd-add-sigma } \sigma \ (\text{int-of-eint } i) \ n) \ (\text{int-of-eint } i)$

$\wedge (\text{similar } \sigma \ n \ (\text{upd-add-sigma } \sigma \ (\text{int-of-eint } i) \ n))$ **apply** *simp*

by (*metis fun-upd-apply insert-Diff1 similar-defs-alt singletonI*)

have 3: $(i \neq -\infty) \longrightarrow (\exists \sigma'. n \in \sigma' \ (\text{int-of-eint } i) \wedge \text{similar } \sigma \ n \ \sigma')$

using 2 **by** *blast*

have 4: $(i \neq -\infty) \longrightarrow (\sigma \ i \ j \models \text{cEx } n. \$\$n)$

using 3

by (*meson cexist-sigma-cexist-value semantics-introspective-nl-itl.simps(2) semantics-introspective-nl-itl.simps(7)*)

show *?thesis* **using** 1 4 **by** *blast*

qed

lemma *exists-not-atom*:

assumes $i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models \text{Ex } n. \text{inot } \$n)$

using *assms* **apply** *simp* **unfolding** *upd-def*

by *auto*

lemma *ceexists-not-catom*:
assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \ c \models cEx \ n. \ cnot \ \$\$n)$
using *assms* **apply** *simp* **unfolding** *upd-def*
by *auto*

lemma *exists-exists*:
assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models Ex \ n. \ (Ex \ n. \ f)) = (\sigma \ i \ j \models Ex \ n. \ f)$
using *assms*
apply *simp*
using *similar-refl similar-trans*
using *upd-absorb* **by** *presburger*

lemma *ceexists-ceexists*:
assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \ c \models cEx \ n. \ (cEx \ n. \ f)) = (\sigma \ i \ j \ c \models cEx \ n. \ f)$
using *assms*
apply *simp*
using *upd-absorb similar-refl similar-trans* **by** *presburger*

lemma *exists-ior-dist*:
assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models (Ex \ n. \ f1 \ ior \ f2)) = (\sigma \ i \ j \models (Ex \ n. \ f1) \ ior \ (Ex \ n. \ f2))$
using *assms* **by** *auto*

lemma *ceexists-cior-dist*:
assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \ c \models (cEx \ n. \ f1 \ cior \ f2)) = (\sigma \ i \ j \ c \models (cEx \ n. \ f1) \ cior \ (cEx \ n. \ f2))$
using *assms* **by** *auto*

lemma *exists-schop-imp*:
assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$

shows $(\sigma \ i \ j \models (Ex \ n. (f1 \ schop \ f2))) \implies (\sigma \ i \ j \models (Ex \ n. f1) \ schop \ (Ex \ n. f2))$
using *assms by auto*

lemma *ceexists-cschop-imp*:

assumes $i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models (cEx \ n. (f1 \ cschop \ f2))) \implies (\sigma \ i \ j \models (cEx \ n. f1) \ cschop \ (cEx \ n. f2))$

using *assms by auto*

lemma *sfrdiamond-defs*:

assumes $i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models rdiamond \ (inot(skip \ schop \ (inot \ f)))) =$
 $(j < \infty \wedge (\exists k. k = j \vee (k > j \wedge (\sigma \ j \ k \models f))))$

using *assms apply auto*

apply (*metis dual-order.refl less-le-not-le semantics-nl-itl.simps(4) skip-d-def*)

by (*metis MInfty-neq-eint(1) not-less-iff-gr-or-eq order-le-less skip-defs*)

lemma *spldiamond-defs*:

assumes $i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models ldiamond \ (inot \ ((inot \ f) \ schop \ skip))) =$
 $(i > -\infty \wedge (\exists k. k = i \vee (k < i \wedge (\sigma \ k \ i \models f))))$

using *assms apply auto*

apply *fastforce*

using *add.commute by auto*

lemma *strictly-future-equiv-future-sem*:

assumes $i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $\exists f. (\sigma \ i \ j \models sfrdiamond \ g) = (\sigma \ i \ j \models f)$

using *assms*

proof (*induction g arbitrary: j*)

case (*fintr-d x*)

then show *?case apply auto*

by (*metis semantics-future-nl-itl.simps(2)*)

next

case (*finot-d g*)

then show *?case by auto*

next

case (*fior-d g1 g2*)

then show *?case apply auto*

by *metis*

next

case (*frdiamond-d g*)

```

then show ?case by auto
qed

lemma strictly-past-equiv-past-sem:
assumes  $i \leq j$ 
         $i \neq \infty$ 
         $j \neq -\infty$ 
shows  $\exists f. (\sigma \ i \ j \ sp \models \text{spldiamond } g) = (\sigma \ i \ j \ p \models f)$ 
using assms
proof (induction g arbitrary: i)
case (pintro-d x)
then show ?case apply auto
by (metis semantics-past-nl-itl.simps(2))
next
case (pinot-d g)
then show ?case by auto
next
case (pior-d g1 g2)
then show ?case apply auto by metis
next
case (pldiamond-d g)
then show ?case by auto
qed

```

```

lemma state-nl-itl-list-defs:
state-nl-itl-list L  $\longleftrightarrow$  ( $\forall i < \text{length } L. \text{state-nl-itl } (L!i)$ )
proof (induct L)
case Nil
then show ?case unfolding state-nl-itl-list-def by simp
next
case (Cons a L)
then show ?case
proof -
have 1:  $\text{state-nl-itl-list } (a \# L) \longleftrightarrow \text{state-nl-itl } a \wedge \text{state-nl-itl-list } L$ 
unfolding state-nl-itl-list-def by simp
have 2: ( $\forall i < \text{length } (a \# L). \text{state-nl-itl } ((a \# L)!i)$ )  $\longleftrightarrow$ 
state-nl-itl  $((a \# L)!0) \wedge$ 
( $\forall i. 0 < i \wedge i < \text{length } (a \# L) \longrightarrow \text{state-nl-itl } ((a \# L)!i)$ )
by (metis length-greater-0-conv list.discI zero-less-iff-neq-zero)
have 3:  $\text{state-nl-itl } ((a \# L)!0) \longleftrightarrow \text{state-nl-itl } a$ 
by simp
have 4: ( $\forall i. 0 < i \wedge i < \text{length } (a \# L) \longrightarrow \text{state-nl-itl } ((a \# L)!i)$ )  $\longleftrightarrow$ 
( $\forall i < \text{length } (L). \text{state-nl-itl } (L!i)$ )
by auto
show ?thesis
using 1 2 3 4 local.Cons by presburger
qed
qed

```

```

lemma big-ior-defs :
  ( $\sigma \ i \ j \models (big-ior \ F)$ )  $\longleftrightarrow$  ( $\exists \ k < length \ F. (\sigma \ i \ j \models F \ ! \ k)$ )
proof (induct F)
case Nil
then show ?case unfolding big-ior-def by simp
next
case (Cons a F)
then show ?case
  proof –
    have 1: ( $\sigma \ i \ j \models (big-ior \ (a \# F))$ )  $\longleftrightarrow$  ( $\sigma \ i \ j \models a \ ior \ (big-ior \ F)$ )
      unfolding big-ior-def by simp
    have 2: ( $\exists \ k < length \ (a \ \# \ F). \sigma \ i \ j \models (a \ \# \ F) \ ! \ k$ )  $\longleftrightarrow$ 
      ( $\sigma \ i \ j \models (a \ \# \ F) \ ! \ 0$ )  $\vee$ 
      ( $\exists \ k. \ 0 < k \wedge k < length \ (a \ \# \ F) \wedge (\sigma \ i \ j \models (a \ \# \ F) \ ! \ k)$ )
      by (metis length-greater-0-conv neq-Nil-conv zero-less-iff-neq-zero)
    have 3: ( $\sigma \ i \ j \models (a \ \# \ F) \ ! \ 0$ ) = ( $\sigma \ i \ j \models a$ )
      by auto
    have 4: ( $\exists \ k. \ 0 < k \wedge k < length \ (a \ \# \ F) \wedge (\sigma \ i \ j \models (a \ \# \ F) \ ! \ k)$ )  $\longleftrightarrow$ 
      ( $\exists \ k < length \ ( \ F). (\sigma \ i \ j \models ( \ F) \ ! \ k)$ )
      by auto
      (metis Suc-le-eq diff-Suc-Suc diff-zero gr0-conv-Suc less-Suc-eq-le)
    show ?thesis
    using 1 2 4 local.Cons by force
  qed
qed

```

```

lemma big-iand-defs :
  ( $\sigma \ i \ j \models (big-iand \ F)$ )  $\longleftrightarrow$  ( $\forall \ k < length \ F. (\sigma \ i \ j \models F \ ! \ k)$ )
proof (induct F)
case Nil
then show ?case unfolding big-iand-def by simp
next
case (Cons a F)
then show ?case
  proof –
    have 1: ( $\sigma \ i \ j \models (big-iand \ (a \# F))$ )  $\longleftrightarrow$  ( $\sigma \ i \ j \models a \ iand \ (big-iand \ F)$ )
      unfolding big-iand-def by simp
    have 2: ( $\forall \ k < length \ (a \ \# \ F). \sigma \ i \ j \models (a \ \# \ F) \ ! \ k$ )  $\longleftrightarrow$ 
      ( $\sigma \ i \ j \models (a \ \# \ F) \ ! \ 0$ )  $\wedge$ 
      ( $\forall \ k. \ 0 < k \wedge k < length \ (a \ \# \ F) \longrightarrow (\sigma \ i \ j \models (a \ \# \ F) \ ! \ k)$ )
      by (metis length-greater-0-conv neq-Nil-conv zero-less-iff-neq-zero)
    have 3: ( $\sigma \ i \ j \models (a \ \# \ F) \ ! \ 0$ ) = ( $\sigma \ i \ j \models a$ )
      by auto
    have 4: ( $\forall \ k. \ 0 < k \wedge k < length \ (a \ \# \ F) \longrightarrow (\sigma \ i \ j \models (a \ \# \ F) \ ! \ k)$ )  $\longleftrightarrow$ 
      ( $\forall \ k < length \ ( \ F). (\sigma \ i \ j \models ( \ F) \ ! \ k)$ )
      by auto
    show ?thesis
  qed

```

```

    using 1 2 4 local.Cons by force
  qed
qed

```

```

lemma big-ior-map-iand-zip :
assumes length F = length G
shows  $(\sigma \ i \ j \models \text{big-ior} (\text{map2 } \text{iand-d } F \ G)) \longleftrightarrow$ 
 $(\exists k < \text{length } F. (\sigma \ i \ j \models F \ ! \ k \ \text{iand } G \ ! \ k))$ 
using assms
proof (induct F arbitrary: G)
case Nil
then show ?case by (simp add: big-ior-defs)
next
case (Cons a F)
then show ?case
  proof (cases G=[])
  case True
  then show ?thesis using Cons.prem by auto
  next
  case False
  then show ?thesis using Cons by (auto simp add: big-ior-defs)
  qed
qed

```

```

lemma big-iand-map-imp-zip :
assumes length F = length G
shows  $(\sigma \ i \ j \models \text{big-iand} (\text{map2 } \text{imp-d } F \ G)) \longleftrightarrow$ 
 $(\forall k < \text{length } F. (\sigma \ i \ j \models F \ ! \ k \ \text{imp } G \ ! \ k))$ 
using assms
proof (induct F arbitrary: G)
case Nil
then show ?case by (simp add: big-iand-defs)
next
case (Cons a F)
then show ?case
  proof (cases G=[])
  case True
  then show ?thesis using Cons.prem by auto
  next
  case False
  then show ?thesis using Cons by (auto simp add: big-iand-defs)
  qed
qed

```

```

lemma map2-iand-defs :
assumes length F = length G

```



```

shows ( $\exists k < \text{length } G. \sigma \ i \ j \models \text{map2 } iand\text{-}d \ F \ G \ ! \ k$ )  $\longleftrightarrow$ 
  ( $\exists k < \text{length } G. (\sigma \ i \ j \models F \ ! \ k) \wedge (\sigma \ i \ j \models G \ ! \ k)$ )
using assms
proof (induct G arbitrary: F)
case Nil
then show ?case by simp
next
case (Cons a G)
then show ?case
  proof (cases F = [])
  case True
  then show ?thesis using Cons.prems by auto
  next
  case False
  then show ?thesis using Cons by auto
  qed
qed

```

```

lemma map2-iand-inot-defs :
assumes length F = length G
shows ( $\exists k < \text{length } G. \sigma \ i \ j \models \text{map2 } (\lambda x \ y. x \ iand \ inot \ y) \ F \ G \ ! \ k$ )  $\longleftrightarrow$ 
  ( $\exists k < \text{length } G. (\sigma \ i \ j \models F \ ! \ k) \wedge \neg(\sigma \ i \ j \models G \ ! \ k)$ )
using assms
proof (induct G arbitrary: F)
case Nil
then show ?case by simp
next
case (Cons a G)
then show ?case
  proof (cases F = [])
  case True
  then show ?thesis using Cons.prems by auto
  next
  case False
  then show ?thesis using Cons by auto
  qed
qed

```

6.3 Validity

```

definition valid :: nl-itl  $\Rightarrow$  bool (( $\vdash$  -) 5)
where ( $\vdash f$ ) = ( $\forall \sigma \ i \ j. i \leq j \wedge i \neq \infty \wedge j \neq -\infty \longrightarrow (\sigma \ i \ j \models f)$ )

```

```

lemma itl-valid [simp] :
  ( $\vdash f$ ) = ( $\forall \sigma \ i \ j. i \leq j \wedge i \neq \infty \wedge j \neq -\infty \longrightarrow (\sigma \ i \ j \models f)$ )
by (simp add: valid-def)

```

```

lemma itl-ieq:

```

$(\vdash f \text{ieqv} g) = (\forall \sigma \ i \ j. i \leq j \wedge i \neq \infty \wedge j \neq -\infty \longrightarrow (\sigma \ i \ j \models f) = (\sigma \ i \ j \models g))$
by (*auto simp add: ieqv-d-def iand-d-def ior-d-def inot-d-def*)

lemma *big-ior-iimp-subset-eq*:

assumes *set L1* \subseteq *set L2*

shows $\vdash \text{big-ior } L1 \text{ iimp } \text{big-ior } L2$

using *assms*

proof (*induct rule: list-induct2'*)

case 1

then show ?*case* **by** *simp*

next

case (2 *x xs*)

then show ?*case* **by** *simp*

next

case (3 *y ys*)

then show ?*case* **by** (*simp add: big-ior-defs*)

next

case (4 *x xs y ys*)

then show ?*case*

proof (*cases x=y*)

case *True*

then show ?*thesis* **using** 4 **apply** (*auto simp add: big-ior-defs*)

by (*metis diff-Suc-1 in-set-conv-nth length-Cons less-Suc-eq-0-disj list.simps(15) nth-Cons' subsetD*)

next

case *False*

then show ?*thesis* **using** 4 **apply** (*auto simp add: big-ior-defs*)

by (*metis 4.premis in-set-conv-nth length-Cons subsetD*)

qed

qed

lemma *big-ior-ieqv-set-eq*:

assumes *set L1* = *set L2*

shows $\vdash \text{big-ior } L1 \text{ ieqv } \text{big-ior } L2$

using *assms*

by (*metis big-ior-iimp-subset-eq dual-order.refl iand-defs ieqv-d-def valid-def*)

lemma *big-ior-ieqv-remdups*:

shows $\vdash \text{big-ior } L \text{ ieqv } \text{big-ior } (\text{remdups } L)$

by (*metis big-ior-ieqv-set-eq set-remdups*)

lemma *big-iand-iimp-subset-eq*:

assumes *set L1* \subseteq *set L2*

shows $\vdash \text{big-iand } L2 \text{ iimp } \text{big-iand } L1$

using *assms*

proof (*induct rule: list-induct2'*)

```

case 1
then show ?case by simp
next
case (2 x xs)
then show ?case by (simp add: big-iand-defs)
next
case (3 y ys)
then show ?case by simp
next
case (4 x xs y ys)
then show ?case
  proof (cases x=y)
  case True
  then show ?thesis using 4 apply (auto simp add: big-iand-defs )
    using 4 all-nth-imp-all-set length-Cons subsetD
    by (metis nth-mem)
  next
  case False
  then show ?thesis using 4 apply (auto simp add: big-iand-defs)
    by (metis 4.prem1 all-nth-imp-all-set length-Cons nth-mem subsetD)
  qed
qed

```

```

lemma big-iand-ieqv-set-eq:
  assumes set L1 = set L2
  shows  $\vdash$  big-iand L1 ieqv big-iand L2
using assms
by (metis big-iand-imp-subset-eq dual-order.refl iand-defs ieqv-d-def valid-def)

```

```

lemma big-iand-ieqv-remdups:
   $\vdash$  big-iand L ieqv big-iand (remdups L)
by (metis big-iand-ieqv-set-eq set-remdups)

```

```

lemma linf-subst:
  assumes  $n \notin \text{bvars } f$ 
     $\bigwedge z . z \in \text{fvars } w \implies z \notin \text{bvars } f$ 
    state-nl-ittl w
     $i \leq j$ 
     $i \neq \infty$ 
     $j \neq -\infty$ 
  shows  $(\sigma \ i \ j \models \text{suform linf } n \ w) = ((\text{upd } \sigma \ n \ (\lambda c. (\sigma \ c \ c \models w))) \ i \ j \models \text{linf})$ 
using assms
unfolding linf-d-def iand-d-def lfinite-d-def itrue-d-def empty-d-def more-d-def ior-d-def inot-d-def
by simp

```

```

lemma linf-subst-1:
  assumes  $n \notin \text{bvars } f$ 

```

$\bigwedge z . z \in \text{fvars } w \implies z \notin \text{bvars } f$
 $\text{state-nl-itl } w$
 $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{suform } \text{linf } n \ w) = (\sigma \ i \ j \models \text{linf})$
using *assms*
unfolding *linf-d-def iand-d-def lfinite-d-def itrue-d-def empty-d-def more-d-def ior-d-def inot-d-def*
by *simp*

lemma *state-nl*:
assumes *state-nl-itl w*
 $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
 $(\sigma \ i \ j \models \$x)$
shows $i \neq -\infty$
using *assms* **by** *simp*

lemma *lfinite-subst-2*:
assumes $n \notin \text{bvars } f$
 $\bigwedge z . z \in \text{fvars } w \implies z \notin \text{bvars } f$
 $\text{state-nl-itl } w$
 $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
shows $(\sigma \ i \ j \models \text{suform } (\$x) \ n \ (w \ \text{iand} \ \text{lfinite})) = ((\text{upd } \sigma \ n \ (\lambda \ c. (\sigma \ c \ c \models w) \)) \ i \ j \models \$x)$
proof –
have 1: $(\sigma \ i \ j \models \text{suform } (\$x) \ n \ (w \ \text{iand} \ \text{lfinite})) =$
 $(\text{if } x = n \text{ then } (\sigma \ i \ j \models (w \ \text{iand} \ \text{lfinite})) \text{ else } (i \neq -\infty \wedge x \in \sigma \ (\text{int-of-eint } i)))$
by *simp*
have 15: $(\text{if } x = n \text{ then } (\sigma \ i \ j \models (w \ \text{iand} \ \text{lfinite})) \text{ else } (i \neq -\infty \wedge x \in \sigma \ (\text{int-of-eint } i))) =$
 $(i \neq -\infty \wedge (\text{if } x = n \text{ then } (\sigma \ i \ j \models (w)) \text{ else } (x \in \sigma \ (\text{int-of-eint } i))))$
using *assms(4) assms(5) assms(6) lfinite-defs* **by** *force*
have 2: $((\text{upd } \sigma \ n \ (\lambda \ c. (\sigma \ c \ c \models w) \)) \ i \ j \models \$x) =$
 $(i \neq -\infty \wedge x \in (\text{upd } \sigma \ n \ (\lambda x. \sigma \ (\text{eint } x) \ (\text{eint } x) \models w)) \ (\text{int-of-eint } i))$
by *simp*
have 3: $i \neq -\infty \implies$
 $x \in (\text{upd } \sigma \ n \ (\lambda x. \sigma \ (\text{eint } x) \ (\text{eint } x) \models w)) \ (\text{int-of-eint } i) =$
 $(\text{if } x = n \text{ then } (\sigma \ i \ j \models (w)) \text{ else } (x \in \sigma \ (\text{int-of-eint } i)))$

unfolding *upd-def* **apply** *simp*
by (*metis PInfy-neq-eint(1) abs-eint.elims assms(3) assms(5) assms(6) eint-int state-nl-itl-defs*)
show *?thesis*
using 1 15 2 3 **by** *blast*
qed

lemma *subst-suform*:

```

assumes  $n \notin \text{bvars } f$ 
 $\bigwedge z . z \in \text{fvars } w \implies z \notin \text{bvars } f$ 
 $\text{state-nl-itl } w$ 
 $i \leq j$ 
 $i \neq \infty$ 
 $j \neq -\infty$ 
shows  $(\sigma \ i \ j \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite})) = ((\text{upd } \sigma \ n \ (\lambda \ c. (\sigma \ c \ c \models w)) \ )) \ i \ j \models f)$ 
using assms
proof (induction f arbitrary:  $\sigma \ w \ i \ j \ n$ )
case false-d
then show ?case by simp
next
case (atom-d x)
then show ?case
using lfinite-subst-2 by blast
next
case (iimp-d f1 f2)
then show ?case
by simp
next
case (next-d f)
then show ?case
proof –
  have n1:  $(\sigma \ i \ j \models \text{suform } (\text{next } f) \ n \ (w \ \text{iand} \ \text{lfinite})) =$ 
 $(i \neq -\infty \wedge i < j \wedge (\sigma \ (i + \text{eint } 1) \ j \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite})))$ 
    by simp
  have n2:  $((\text{upd } \sigma \ n \ (\lambda x. \sigma \ (\text{eint } x) \ (\text{eint } x) \models w)) \ i \ j \models \text{next } f) =$ 
 $(i \neq -\infty \wedge i < j \wedge ((\text{upd } \sigma \ n \ (\lambda x. \sigma \ (\text{eint } x) \ (\text{eint } x) \models w)) \ (i + \text{eint } 1) \ j \models f))$ 
    by simp
  have n3:  $i \neq -\infty \wedge i < j \implies i + \text{eint } 1 \leq j$ 
    by (metis eint-less-le-alt int-of-eint.cases next-d.premis(5) plus-eint.simps(1))
  have n4:  $i \neq -\infty \wedge i < j \implies$ 
 $(\sigma \ (i + \text{eint } 1) \ j \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite})) =$ 
 $((\text{upd } \sigma \ n \ (\lambda x. \sigma \ (\text{eint } x) \ (\text{eint } x) \models w)) \ (i + \text{eint } 1) \ j \models f)$ 
    using next-d.IH[of n w (i + eint 1) j  $\sigma$ ]
    using n3 next-d.premis(1) next-d.premis(2) next-d.premis(3) next-d.premis(5) next-d.premis(6) by auto
  show ?thesis
  using n1 n2 n4 by blast
qed
next
case (chop-d f1 f2)
then show ?case by auto
next
case (ldiamond-d f)
then show ?case
proof –
  have l1:  $(\sigma \ i \ j \models \text{suform } (\text{ldiamond } f) \ n \ (w \ \text{iand} \ \text{lfinite})) =$ 
 $(i \neq -\infty \wedge (\exists k \leq i. \sigma \ k \ i \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite})))$ 
    by simp
  have l2:  $((\text{upd } \sigma \ n \ (\lambda c. \sigma \ (\text{eint } c) \ (\text{eint } c) \models w)) \ i \ j \models \text{ldiamond } f) =$ 

```

$$(i \neq -\infty \wedge (\exists k \leq i. (\text{upd } \sigma \ n \ (\lambda c. \sigma \ (\text{eint } c) \ (\text{eint } c) \models w)) \ k \ i \models f))$$

```

by simp
have l3:  $i \neq -\infty \implies$ 
   $(\exists k \leq i. (\sigma \ k \ i \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite}))) =$ 
   $(\exists k \leq i. ((\text{upd } \sigma \ n \ (\lambda c. \sigma \ (\text{eint } c) \ (\text{eint } c) \models w)) \ k \ i \models f))$ 
using ldiamond-d.IH[of n w - i σ]
using bvars.simps(6) eint-infty-less-eq(1) ldiamond-d.prem(1) ldiamond-d.prem(2) ldiamond-d.prem(3) ldiamond-d.prem(5) by blast
show ?thesis
using l1 l2 l3 by blast
qed
next
case (rdiamond-d f)
then show ?case
proof -
have r1:  $(\sigma \ i \ j \models \text{suform } (\text{rdiamond } f) \ n \ (w \ \text{iand} \ \text{lfinite})) =$ 
   $(j \neq \infty \wedge (\exists k \geq j. \sigma \ j \ k \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite})))$ 
by simp
have r2:  $((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \sigma \ (\text{eint } x) \ (\text{eint } x) \models w)) \ i \ j \models \text{rdiamond } f) =$ 
   $(j \neq \infty \wedge (\exists k \geq j. (\text{upd } \sigma \ n \ (\lambda x::\text{int}. \sigma \ (\text{eint } x) \ (\text{eint } x) \models w)) \ j \ k \models f))$ 
by simp
have r3:  $j \neq \infty \implies$ 
   $(\exists k \geq j. \sigma \ j \ k \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite})) =$ 
   $(\exists k \geq j. (\text{upd } \sigma \ n \ (\lambda x::\text{int}. \sigma \ (\text{eint } x) \ (\text{eint } x) \models w)) \ j \ k \models f)$ 
using rdiamond-d.IH[of n w j - σ]
using bvars.simps(7) eint-infty-less-eq(2) rdiamond-d.prem(1) rdiamond-d.prem(2) rdiamond-d.prem(3) rdiamond-d.prem(6) by blast
show ?thesis
using r1 r2 r3 by blast
qed
next
case (chopstar-d f)
then show ?case
proof -
have c1:  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } f) \ n \ (w \ \text{iand} \ \text{lfinite})) =$ 
   $(i = j \vee$ 
     $i \neq j \wedge i \neq -\infty \wedge j \neq \infty \wedge$ 
     $(\exists \text{ lrf}::\text{eint nellist}. \text{nnth lrf } (0::\text{nat}) = i \wedge \text{nfinite lrf} \wedge \text{nlast lrf} = j \wedge$ 
     $\text{nridx } (\lambda(a::\text{eint}) \ b::\text{eint}. a < b \wedge (\sigma \ a \ b \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite}))) \ \text{lrf}) \vee$ 
     $i \neq j \wedge i = -\infty \wedge j \neq \infty \wedge$ 
     $(\exists \text{ li}::\text{eint nellist}. \text{nnth li } (0::\text{nat}) = j \wedge \neg \text{nfinite li} \wedge$ 
     $\text{nridx } (\lambda(a::\text{eint}) \ b::\text{eint}. b < a \wedge (\sigma \ b \ a \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite}))) \ \text{li}) \vee$ 
     $i \neq j \wedge i \neq -\infty \wedge j = \infty \wedge$ 
     $(\exists \text{ ri}::\text{eint nellist}. \text{nnth ri } (0::\text{nat}) = i \wedge \neg \text{nfinite ri} \wedge$ 
     $\text{nridx } (\lambda(a::\text{eint}) \ b::\text{eint}. a < b \wedge (\sigma \ a \ b \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite}))) \ \text{ri}) \vee$ 
     $i \neq j \wedge i = -\infty \wedge j = \infty \wedge$ 
     $(\exists (\text{li}::\text{int nellist}) \ \text{ri}::\text{int nellist}.$ 
     $\text{nnth li } (0::\text{nat}) = \text{nnth ri } (0::\text{nat}) \wedge$ 
     $\neg \text{nfinite li} \wedge$ 

```

$$\text{nrIdx } (\lambda(a::\text{int}) \ b::\text{int}. \ b < a \wedge (\sigma \ (eint \ b) \ (eint \ a) \models \text{suform } f \ n \ (w \ iand \ lfinite))) \ li \wedge$$

$$\neg \text{nfinite } ri \wedge \text{nrIdx } (\lambda(a::\text{int}) \ b::\text{int}. \ a < b \wedge (\sigma \ (eint \ a) \ (eint \ b) \models \text{suform } f \ n \ (w \ iand \ lfinite))) \ ri))$$

by simp

have $c2: ((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \ \sigma \ (eint \ x) \ (eint \ x) \models w)) \ i \ j \models \text{chopstar } f) =$

$$(i = j \vee$$

$$i \neq j \wedge i \neq -\infty \wedge j \neq \infty \wedge$$

$$(\exists \text{lrf}::\text{eint nellist}. \ \text{nnth } \text{lrf} \ (0::\text{nat}) = i \wedge \text{nfinite } \text{lrf} \wedge \text{nlast } \text{lrf} = j \wedge$$

$$\text{nrIdx } (\lambda(a::\text{eint}) \ b::\text{eint}. \ a < b \wedge$$

$$((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \ \sigma \ (eint \ x) \ (eint \ x) \models w)) \ a \ b \models f)) \ \text{lrf}) \vee$$

$$i \neq j \wedge i = -\infty \wedge j \neq \infty \wedge$$

$$(\exists \text{li}::\text{eint nellist}. \ \text{nnth } \text{li} \ (0::\text{nat}) = j \wedge \neg \text{nfinite } \text{li} \wedge$$

$$\text{nrIdx } (\lambda(a::\text{eint}) \ b::\text{eint}. \ b < a \wedge$$

$$((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \ \sigma \ (eint \ x) \ (eint \ x) \models w)) \ b \ a \models f)) \ \text{li}) \vee$$

$$i \neq j \wedge i \neq -\infty \wedge j = \infty \wedge$$

$$(\exists \text{ri}::\text{eint nellist}. \ \text{nnth } \text{ri} \ (0::\text{nat}) = i \wedge \neg \text{nfinite } \text{ri} \wedge$$

$$\text{nrIdx } (\lambda(a::\text{eint}) \ b::\text{eint}. \ a < b \wedge$$

$$((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \ \sigma \ (eint \ x) \ (eint \ x) \models w)) \ a \ b \models f)) \ \text{ri}) \vee$$

$$i \neq j \wedge i = -\infty \wedge j = \infty \wedge$$

$$(\exists (\text{li}::\text{int nellist}) \ \text{ri}::\text{int nellist}. \ \text{nnth } \text{li} \ (0::\text{nat}) = \text{nnth } \text{ri} \ (0::\text{nat}) \wedge$$

$$\neg \text{nfinite } \text{li} \wedge$$

$$\text{nrIdx } (\lambda(a::\text{int}) \ b::\text{int}. \ b < a \wedge$$

$$((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \ \sigma \ (eint \ x) \ (eint \ x) \models w)) \ (eint \ b) \ (eint \ a) \models f)) \ \text{li} \wedge$$

$$\neg \text{nfinite } \text{ri} \wedge$$

$$\text{nrIdx } (\lambda(a::\text{int}) \ b::\text{int}. \ a < b \wedge$$

$$((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \ \sigma \ (eint \ x) \ (eint \ x) \models w)) \ (eint \ a) \ (eint \ b) \models f)) \ \text{ri}))$$

by simp

have $c3: i = j \implies ?thesis$

using $c1 \ c2$ **by presburger**

have $c4: i \neq j \implies i \neq -\infty \wedge j \neq \infty \implies$

$$(\exists \text{lrf}::\text{eint nellist}. \ \text{nnth } \text{lrf} \ (0::\text{nat}) = i \wedge \text{nfinite } \text{lrf} \wedge \text{nlast } \text{lrf} = j \wedge$$

$$\text{nrIdx } (\lambda(a::\text{eint}) \ b::\text{eint}. \ a < b \wedge (\sigma \ a \ b \models \text{suform } f \ n \ (w \ iand \ lfinite))) \ \text{lrf}) \implies$$

$$(\exists \text{lrf}::\text{eint nellist}. \ \text{nnth } \text{lrf} \ (0::\text{nat}) = i \wedge \text{nfinite } \text{lrf} \wedge \text{nlast } \text{lrf} = j \wedge$$

$$\text{nrIdx } (\lambda(a::\text{eint}) \ b::\text{eint}. \ a < b \wedge ((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \ \sigma \ (eint \ x) \ (eint \ x) \models w)) \ a \ b \models f)) \ \text{lrf})$$

proof –

assume $c4a0: i \neq j$

assume $c4a1: i \neq -\infty \wedge j \neq \infty$

assume $c4a2: (\exists \text{lrf}::\text{eint nellist}. \ \text{nnth } \text{lrf} \ (0::\text{nat}) = i \wedge \text{nfinite } \text{lrf} \wedge \text{nlast } \text{lrf} = j \wedge$

$$\text{nrIdx } (\lambda(a::\text{eint}) \ b::\text{eint}. \ a < b \wedge (\sigma \ a \ b \models \text{suform } f \ n \ (w \ iand \ lfinite))) \ \text{lrf})$$

show $(\exists \text{lrf}::\text{eint nellist}. \ \text{nnth } \text{lrf} \ (0::\text{nat}) = i \wedge \text{nfinite } \text{lrf} \wedge \text{nlast } \text{lrf} = j \wedge$

$$\text{nrIdx } (\lambda(a::\text{eint}) \ b::\text{eint}. \ a < b \wedge ((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \ \sigma \ (eint \ x) \ (eint \ x) \models w)) \ a \ b \models f)) \ \text{lrf})$$

proof –

obtain lrf **where** $c41: \text{nnth } \text{lrf} \ (0::\text{nat}) = i \wedge \text{nfinite } \text{lrf} \wedge \text{nlast } \text{lrf} = j \wedge$

$$\text{nrIdx } (\lambda(a::\text{eint}) \ b::\text{eint}. \ a < b \wedge (\sigma \ a \ b \models \text{suform } f \ n \ (w \ iand \ lfinite))) \ \text{lrf}$$

using $c4a2$ **by blast**

have $c42: \text{nrIdx } (\lambda(a::\text{eint}) \ b::\text{eint}. \ a < b \wedge ((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \ \sigma \ (eint \ x) \ (eint \ x) \models w)) \ a \ b \models f)) \ \text{lrf}$

using $\text{chopstar-d.IH}[\text{of } n \ w \ - \ \sigma] \ c41 \ \text{chopstar-d.premis}$

by $(\text{metis } (\text{no-types}, \text{lifting}) \ \text{bvars.simps}(8) \ \text{eint-minus-less-minus eint-uminus-eq-reorder less-eint.simps}(2) \ \text{nrIdx-expand order-less-imp-le})$

```

show ?thesis
using c41 c42 by blast
qed
qed
have c5:  $i \neq j \implies i \neq -\infty \wedge j \neq \infty \implies$ 
  ( $\exists \text{ lrf}::\text{eint nellist. nth lrf } (0::\text{nat}) = i \wedge \text{nfinite lrf} \wedge \text{nlast lrf} = j \wedge$ 
     $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint. } a < b \wedge ((\text{upd } \sigma \text{ n } (\lambda x::\text{int. } \sigma (\text{eint } x) (\text{eint } x) \models w)) \text{ a b } \models f)) \text{ lrf}) \implies$ 
    ( $\exists \text{ lrf}::\text{eint nellist. nth lrf } (0::\text{nat}) = i \wedge \text{nfinite lrf} \wedge \text{nlast lrf} = j \wedge$ 
       $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint. } a < b \wedge (\sigma \text{ a b } \models \text{suform f n } (w \text{ iand lfinite}))) \text{ lrf})$ )
proof -
  assume c5a0:  $i \neq j$ 
  assume c5a1:  $i \neq -\infty \wedge j \neq \infty$ 
  assume c5a2: ( $\exists \text{ lrf}::\text{eint nellist. nth lrf } (0::\text{nat}) = i \wedge \text{nfinite lrf} \wedge \text{nlast lrf} = j \wedge$ 
     $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint. } a < b \wedge ((\text{upd } \sigma \text{ n } (\lambda x::\text{int. } \sigma (\text{eint } x) (\text{eint } x) \models w)) \text{ a b } \models f)) \text{ lrf})$ )
  show ( $\exists \text{ lrf}::\text{eint nellist. nth lrf } (0::\text{nat}) = i \wedge \text{nfinite lrf} \wedge \text{nlast lrf} = j \wedge$ 
     $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint. } a < b \wedge (\sigma \text{ a b } \models \text{suform f n } (w \text{ iand lfinite}))) \text{ lrf})$ )
  proof -
    obtain lrf where c51:  $\text{nth lrf } (0::\text{nat}) = i \wedge \text{nfinite lrf} \wedge \text{nlast lrf} = j \wedge$ 
       $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint. } a < b \wedge ((\text{upd } \sigma \text{ n } (\lambda x::\text{int. } \sigma (\text{eint } x) (\text{eint } x) \models w)) \text{ a b } \models f)) \text{ lrf}$ 
    using c5a2 by blast
    have c52:  $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint. } a < b \wedge (\sigma \text{ a b } \models \text{suform f n } (w \text{ iand lfinite}))) \text{ lrf}$ 
    using chopstar-d.IH[of n w - -  $\sigma$ ] c51 chopstar-d.premis
    by (metis (no-types, lifting) bvars.simps(8) less-eint.simps(2) less-eint.simps(3) nridx-expand order-less-imp-le)
    show ?thesis using c51 c52 by blast
  qed
qed
have c6:  $i \neq j \wedge i \neq -\infty \wedge j \neq \infty \implies ?thesis$ 
  using c4 c5 apply simp by argo
have c7:  $i \neq j \implies i = -\infty \wedge j \neq \infty \implies$ 
  ( $\exists \text{ li}::\text{eint nellist. nth li } (0::\text{nat}) = j \wedge \neg \text{nfinite li} \wedge$ 
     $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint. } b < a \wedge (\sigma \text{ b a } \models \text{suform f n } (w \text{ iand lfinite}))) \text{ li}) \implies$ 
  ( $\exists \text{ li}::\text{eint nellist. nth li } (0::\text{nat}) = j \wedge \neg \text{nfinite li} \wedge$ 
     $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint. } b < a \wedge$ 
       $((\text{upd } \sigma \text{ n } (\lambda x::\text{int. } \sigma (\text{eint } x) (\text{eint } x) \models w)) \text{ b a } \models f)) \text{ li})$ )
proof -
  assume c7a0:  $i \neq j$ 
  assume c7a1:  $i = -\infty \wedge j \neq \infty$ 
  assume c7a2: ( $\exists \text{ li}::\text{eint nellist. nth li } (0::\text{nat}) = j \wedge \neg \text{nfinite li} \wedge$ 
     $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint. } b < a \wedge (\sigma \text{ b a } \models \text{suform f n } (w \text{ iand lfinite}))) \text{ li})$ )
  show ( $\exists \text{ li}::\text{eint nellist. nth li } (0::\text{nat}) = j \wedge \neg \text{nfinite li} \wedge$ 
     $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint. } b < a \wedge$ 
       $((\text{upd } \sigma \text{ n } (\lambda x::\text{int. } \sigma (\text{eint } x) (\text{eint } x) \models w)) \text{ b a } \models f)) \text{ li})$ )
  proof -
    obtain li where c71:  $\text{nth li } (0::\text{nat}) = j \wedge \neg \text{nfinite li} \wedge$ 
       $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint. } b < a \wedge (\sigma \text{ b a } \models \text{suform f n } (w \text{ iand lfinite}))) \text{ li}$ 
    using c7a2 by blast
    have c72:  $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint. } b < a \wedge$ 
       $((\text{upd } \sigma \text{ n } (\lambda x::\text{int. } \sigma (\text{eint } x) (\text{eint } x) \models w)) \text{ b a } \models f)) \text{ li}$ 
    using chopstar-d.IH[of n w - -  $\sigma$ ] c71 chopstar-d.premis
  qed

```


by (metis (no-types, lifting) bvvars.simps(8) dual-order.strict-iff-order eint-minus-less-minus
 eint-uminus-uminus less-eint.simps(2) nridx-expand)
 show ?thesis using c71 c72 by blast
 qed
 qed
 have c8: $i \neq j \implies i = -\infty \wedge j \neq \infty \implies$
 $(\exists li::eint\ nellist. \text{nnth } li\ (0::nat) = j \wedge \neg \text{nfinite } li \wedge$
 $\text{nridx } (\lambda(a::eint)\ b::eint. b < a \wedge$
 $((\text{upd } \sigma\ n\ (\lambda x::int. \sigma\ (eint\ x)\ (eint\ x) \models w))\ b\ a \models f))\ li) \implies$
 $(\exists li::eint\ nellist. \text{nnth } li\ (0::nat) = j \wedge \neg \text{nfinite } li \wedge$
 $\text{nridx } (\lambda(a::eint)\ b::eint. b < a \wedge (\sigma\ b\ a \models \text{suform } f\ n\ (w\ iand\ lfinite))))\ li)$
 proof –
 assume c8a0: $i \neq j$
 assume c8a1: $i = -\infty \wedge j \neq \infty$
 assume c8a2: $(\exists li::eint\ nellist. \text{nnth } li\ (0::nat) = j \wedge \neg \text{nfinite } li \wedge$
 $\text{nridx } (\lambda(a::eint)\ b::eint. b < a \wedge$
 $((\text{upd } \sigma\ n\ (\lambda x::int. \sigma\ (eint\ x)\ (eint\ x) \models w))\ b\ a \models f))\ li)$
 show $(\exists li::eint\ nellist. \text{nnth } li\ (0::nat) = j \wedge \neg \text{nfinite } li \wedge$
 $\text{nridx } (\lambda(a::eint)\ b::eint. b < a \wedge (\sigma\ b\ a \models \text{suform } f\ n\ (w\ iand\ lfinite))))\ li)$
 proof –
 obtain li where c81: $\text{nnth } li\ (0::nat) = j \wedge \neg \text{nfinite } li \wedge$
 $\text{nridx } (\lambda(a::eint)\ b::eint. b < a \wedge$
 $((\text{upd } \sigma\ n\ (\lambda x::int. \sigma\ (eint\ x)\ (eint\ x) \models w))\ b\ a \models f))\ li$
 using c8a2 by blast
 have c82: $\text{nridx } (\lambda(a::eint)\ b::eint. b < a \wedge (\sigma\ b\ a \models \text{suform } f\ n\ (w\ iand\ lfinite))))\ li$
 using chopstar-d.IH[of n w - - σ] c81 chopstar-d.premis
 by (metis (no-types, lifting) bvvars.simps(8) less-eint.simps(2) less-eint.simps(3) nridx-expand or-
 der-less-imp-le)
 show ?thesis using c81 c82 by blast
 qed
 qed
 have c9: $i \neq j \implies i = -\infty \wedge j \neq \infty \implies ?thesis$
 using c7 c8 apply simp by argo
 have c10: $i \neq j \implies i \neq -\infty \wedge j = \infty \implies$
 $(\exists ri::eint\ nellist. \text{nnth } ri\ (0::nat) = i \wedge \neg \text{nfinite } ri \wedge$
 $\text{nridx } (\lambda(a::eint)\ b::eint. a < b \wedge (\sigma\ a\ b \models \text{suform } f\ n\ (w\ iand\ lfinite))))\ ri) \implies$
 $(\exists ri::eint\ nellist. \text{nnth } ri\ (0::nat) = i \wedge \neg \text{nfinite } ri \wedge$
 $\text{nridx } (\lambda(a::eint)\ b::eint. a < b \wedge$
 $((\text{upd } \sigma\ n\ (\lambda x::int. \sigma\ (eint\ x)\ (eint\ x) \models w))\ a\ b \models f))\ ri)$
 proof –
 assume c10a0: $i \neq j$
 assume c10a1: $i \neq -\infty \wedge j = \infty$
 assume c10a2: $(\exists ri::eint\ nellist. \text{nnth } ri\ (0::nat) = i \wedge \neg \text{nfinite } ri \wedge$
 $\text{nridx } (\lambda(a::eint)\ b::eint. a < b \wedge (\sigma\ a\ b \models \text{suform } f\ n\ (w\ iand\ lfinite))))\ ri)$
 show $(\exists ri::eint\ nellist. \text{nnth } ri\ (0::nat) = i \wedge \neg \text{nfinite } ri \wedge$
 $\text{nridx } (\lambda(a::eint)\ b::eint. a < b \wedge$
 $((\text{upd } \sigma\ n\ (\lambda x::int. \sigma\ (eint\ x)\ (eint\ x) \models w))\ a\ b \models f))\ ri)$
 proof –
 obtain ri where c101: $\text{nnth } ri\ (0::nat) = i \wedge \neg \text{nfinite } ri \wedge$
 $\text{nridx } (\lambda(a::eint)\ b::eint. a < b \wedge (\sigma\ a\ b \models \text{suform } f\ n\ (w\ iand\ lfinite))))\ ri$

```

using c10a2 by blast
have c102:  $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint}. a < b \wedge$ 
   $((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \sigma \ (\text{eint } x) \ (\text{eint } x) \models w)) \ a \ b \models f)) \ ri$ 
using chopstar-d.IH[of n w - -  $\sigma$ ] c101 chopstar-d.premis
by (metis (no-types, lifting) bvars.simps(8) less-eint.simps(2) less-eint.simps(3) nridx-expand order-less-le)
show ?thesis using c101 c102 by blast
qed
qed
have c11:  $i \neq j \implies i \neq -\infty \wedge j = \infty \implies$ 
   $(\exists ri::\text{eint} \text{ nellist}. \text{nnth } ri \ (0::\text{nat}) = i \wedge \neg \text{nfinite } ri \wedge$ 
   $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint}. a < b \wedge$ 
   $((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \sigma \ (\text{eint } x) \ (\text{eint } x) \models w)) \ a \ b \models f)) \ ri) \implies$ 
   $(\exists ri::\text{eint} \text{ nellist}. \text{nnth } ri \ (0::\text{nat}) = i \wedge \neg \text{nfinite } ri \wedge$ 
   $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint}. a < b \wedge (\sigma \ a \ b \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite}))) \ ri)$ 
proof -
assume c11a0:  $i \neq j$ 
assume c11a1:  $i \neq -\infty \wedge j = \infty$ 
assume c11a2:  $(\exists ri::\text{eint} \text{ nellist}. \text{nnth } ri \ (0::\text{nat}) = i \wedge \neg \text{nfinite } ri \wedge$ 
   $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint}. a < b \wedge$ 
   $((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \sigma \ (\text{eint } x) \ (\text{eint } x) \models w)) \ a \ b \models f)) \ ri)$ 
show  $(\exists ri::\text{eint} \text{ nellist}. \text{nnth } ri \ (0::\text{nat}) = i \wedge \neg \text{nfinite } ri \wedge$ 
   $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint}. a < b \wedge (\sigma \ a \ b \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite}))) \ ri)$ 
proof -
obtain ri where c111:  $\text{nnth } ri \ (0::\text{nat}) = i \wedge \neg \text{nfinite } ri \wedge$ 
   $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint}. a < b \wedge$ 
   $((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \sigma \ (\text{eint } x) \ (\text{eint } x) \models w)) \ a \ b \models f)) \ ri$ 
using c11a2 by blast
have c112:  $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint}. a < b \wedge (\sigma \ a \ b \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite}))) \ ri$ 
using chopstar-d.IH[of n w - -  $\sigma$ ] c111 chopstar-d.premis
by (metis (no-types, lifting) bvars.simps(8) less-eint.simps(2) less-eint.simps(3) nridx-expand order-less-imp-le)
show ?thesis using c111 c112 by blast
qed
qed
have c12:  $i \neq j \wedge i \neq -\infty \wedge j = \infty \implies ?thesis$ 
using c1 c2 apply simp
using c10 c11 by argo
have c13:  $i \neq j \implies i = -\infty \wedge j = \infty \implies$ 
   $(\exists (li::\text{int} \text{ nellist}) \ ri::\text{int} \text{ nellist}.$ 
   $\text{nnth } li \ (0::\text{nat}) = \text{nnth } ri \ (0::\text{nat}) \wedge \neg \text{nfinite } li \wedge$ 
   $\text{nridx } (\lambda(a::\text{int}) b::\text{int}. b < a \wedge (\sigma \ (\text{eint } b) \ (\text{eint } a) \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite}))) \ li \wedge$ 
   $\neg \text{nfinite } ri \wedge$ 
   $\text{nridx } (\lambda(a::\text{int}) b::\text{int}. a < b \wedge (\sigma \ (\text{eint } a) \ (\text{eint } b) \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite}))) \ ri) \implies$ 
   $(\exists (li::\text{int} \text{ nellist}) \ ri::\text{int} \text{ nellist}.$ 
   $\text{nnth } li \ (0::\text{nat}) = \text{nnth } ri \ (0::\text{nat}) \wedge$ 
   $\neg \text{nfinite } li \wedge$ 
   $\text{nridx } (\lambda(a::\text{int}) b::\text{int}. b < a \wedge$ 
   $((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \sigma \ (\text{eint } x) \ (\text{eint } x) \models w)) \ (\text{eint } b) \ (\text{eint } a) \models f)) \ li \wedge$ 
   $\neg \text{nfinite } ri \wedge$ 

```

$nridx (\lambda(a::int) b::int. a < b \wedge$
 $((upd \sigma n (\lambda x::int. \sigma (eint x) (eint x) \models w)) (eint a) (eint b) \models f)) ri)$
proof –
assume $c13a0: i \neq j$
assume $c13a1: i = -\infty \wedge j = \infty$
assume $c13a2: (\exists (li::int nellist) ri::int nellist.$
 $nnth li (0::nat) = nnth ri (0::nat) \wedge \neg nfinite li \wedge$
 $nridx (\lambda(a::int) b::int. b < a \wedge (\sigma (eint b) (eint a) \models suform f n (w iand lfinite))) li \wedge$
 $\neg nfinite ri \wedge$
 $nridx (\lambda(a::int) b::int. a < b \wedge (\sigma (eint a) (eint b) \models suform f n (w iand lfinite))) ri)$
show $(\exists (li::int nellist) ri::int nellist.$
 $nnth li (0::nat) = nnth ri (0::nat) \wedge$
 $\neg nfinite li \wedge$
 $nridx (\lambda(a::int) b::int. b < a \wedge$
 $((upd \sigma n (\lambda x::int. \sigma (eint x) (eint x) \models w)) (eint b) (eint a) \models f)) li \wedge$
 $\neg nfinite ri \wedge$
 $nridx (\lambda(a::int) b::int. a < b \wedge$
 $((upd \sigma n (\lambda x::int. \sigma (eint x) (eint x) \models w)) (eint a) (eint b) \models f)) ri)$
proof –
obtain $li ri$ **where** $c131: nnth li (0::nat) = nnth ri (0::nat) \wedge \neg nfinite li \wedge$
 $nridx (\lambda(a::int) b::int. b < a \wedge (\sigma (eint b) (eint a) \models suform f n (w iand lfinite))) li \wedge$
 $\neg nfinite ri \wedge$
 $nridx (\lambda(a::int) b::int. a < b \wedge (\sigma (eint a) (eint b) \models suform f n (w iand lfinite))) ri$
using $c13a2$ **by** $blast$
have $c132: nridx (\lambda(a::int) b::int. b < a \wedge$
 $((upd \sigma n (\lambda x::int. \sigma (eint x) (eint x) \models w)) (eint b) (eint a) \models f)) li$
using $chopstar-d.IH[of n w - - \sigma]$ $c131$ $chopstar-d.premis$ **unfolding** $nridx-expand$
by $simp$ $auto$
have $c133: nridx (\lambda(a::int) b::int. a < b \wedge$
 $((upd \sigma n (\lambda x::int. \sigma (eint x) (eint x) \models w)) (eint a) (eint b) \models f)) ri$
using $chopstar-d.IH[of n w - - \sigma]$ $c131$ $chopstar-d.premis$ **unfolding** $nridx-expand$ **by** $auto$
show $?thesis$ **using** $c131 c132 c133$ **by** $blast$
qed
qed
have $c14: i \neq j \implies i = -\infty \wedge j = \infty \implies$
 $(\exists (li::int nellist) ri::int nellist.$
 $nnth li (0::nat) = nnth ri (0::nat) \wedge$
 $\neg nfinite li \wedge$
 $nridx (\lambda(a::int) b::int. b < a \wedge$
 $((upd \sigma n (\lambda x::int. \sigma (eint x) (eint x) \models w)) (eint b) (eint a) \models f)) li \wedge$
 $\neg nfinite ri \wedge$
 $nridx (\lambda(a::int) b::int. a < b \wedge$
 $((upd \sigma n (\lambda x::int. \sigma (eint x) (eint x) \models w)) (eint a) (eint b) \models f)) ri) \implies$
 $(\exists (li::int nellist) ri::int nellist.$
 $nnth li (0::nat) = nnth ri (0::nat) \wedge \neg nfinite li \wedge$
 $nridx (\lambda(a::int) b::int. b < a \wedge (\sigma (eint b) (eint a) \models suform f n (w iand lfinite))) li \wedge$
 $\neg nfinite ri \wedge$
 $nridx (\lambda(a::int) b::int. a < b \wedge (\sigma (eint a) (eint b) \models suform f n (w iand lfinite))) ri)$
proof –
assume $c14a0: i \neq j$

```

assume c14a1:  $i = -\infty \wedge j = \infty$ 
assume c14a2:  $(\exists (li::int\ nellist)\ ri::int\ nellist.$ 
   $nnth\ li\ (0::nat) = nnth\ ri\ (0::nat) \wedge$ 
   $\neg\ nfinite\ li \wedge$ 
   $nridx\ (\lambda(a::int)\ b::int. b < a \wedge$ 
   $((upd\ \sigma\ n\ (\lambda x::int. \sigma\ (eint\ x)\ (eint\ x) \models w))\ (eint\ b)\ (eint\ a) \models f))\ li \wedge$ 
   $\neg\ nfinite\ ri \wedge$ 
   $nridx\ (\lambda(a::int)\ b::int. a < b \wedge$ 
   $((upd\ \sigma\ n\ (\lambda x::int. \sigma\ (eint\ x)\ (eint\ x) \models w))\ (eint\ a)\ (eint\ b) \models f))\ ri)$ 
show  $(\exists (li::int\ nellist)\ ri::int\ nellist.$ 
   $nnth\ li\ (0::nat) = nnth\ ri\ (0::nat) \wedge \neg\ nfinite\ li \wedge$ 
   $nridx\ (\lambda(a::int)\ b::int. b < a \wedge (\sigma\ (eint\ b)\ (eint\ a) \models suform\ f\ n\ (w\ iand\ lfinite)))\ li \wedge$ 
   $\neg\ nfinite\ ri \wedge$ 
   $nridx\ (\lambda(a::int)\ b::int. a < b \wedge (\sigma\ (eint\ a)\ (eint\ b) \models suform\ f\ n\ (w\ iand\ lfinite)))\ ri)$ 
proof -
obtain  $li\ ri$  where c141:  $nnth\ li\ (0::nat) = nnth\ ri\ (0::nat) \wedge$ 
   $\neg\ nfinite\ li \wedge$ 
   $nridx\ (\lambda(a::int)\ b::int. b < a \wedge$ 
   $((upd\ \sigma\ n\ (\lambda x::int. \sigma\ (eint\ x)\ (eint\ x) \models w))\ (eint\ b)\ (eint\ a) \models f))\ li \wedge$ 
   $\neg\ nfinite\ ri \wedge$ 
   $nridx\ (\lambda(a::int)\ b::int. a < b \wedge$ 
   $((upd\ \sigma\ n\ (\lambda x::int. \sigma\ (eint\ x)\ (eint\ x) \models w))\ (eint\ a)\ (eint\ b) \models f))\ ri$ 
using c14a2 by blast
have c142:  $nridx\ (\lambda(a::int)\ b::int. b < a \wedge (\sigma\ (eint\ b)\ (eint\ a) \models suform\ f\ n\ (w\ iand\ lfinite)))\ li$ 
using chopstar-d.IH[of  $n\ w\ -\ -\ \sigma$ ] c141 chopstar-d.premis unfolding nridx-expand by auto
have c143:  $nridx\ (\lambda(a::int)\ b::int. a < b \wedge (\sigma\ (eint\ a)\ (eint\ b) \models suform\ f\ n\ (w\ iand\ lfinite)))\ ri$ 
using chopstar-d.IH[of  $n\ w\ -\ -\ \sigma$ ] c141 chopstar-d.premis unfolding nridx-expand by auto
show ?thesis using c141 c142 c143 by blast
qed
qed
have c15:  $i \neq j \implies i = -\infty \wedge j = \infty \implies ?thesis$ 
using c13 c14 apply simp by argo
show ?thesis
using c12 c15 c3 c6 c9 by argo
qed
next
case (exists-d x1 f)
then show ?case
proof (cases  $n = x1$ )
case True
then show ?thesis using exists-d by simp
next
case False
then show ?thesis
proof -
have e4:  $n \notin bvars\ f$ 
using exists-d.premis(1) by auto
have e5:  $((upd\ \sigma\ n\ (\lambda c. \sigma\ (eint\ c)\ (eint\ c) \models w))\ i\ j \models Ex\ x1. f) =$ 
 $(\exists l. (upd\ (upd\ \sigma\ n\ (\lambda c. \sigma\ (eint\ c)\ (eint\ c) \models w))\ x1\ l)\ i\ j \models f)$ 
using exist-sigma-exist-value by auto

```

```

have e6: ( $\bigwedge z. z \in \text{fvars } w \implies z \notin \text{bvars } f$ )
  using exists-d.premis(2) by auto
have e60:  $x1 \notin \text{fvars } w$ 
  using exists-d.premis(2) by auto
have e7: ( $\sigma \ i \ j \models \text{Ex } x1. \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite})$ ) =
  ( $\exists l. (\text{upd } \sigma \ x1 \ l) \ i \ j \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite})$ )
  using exist-sigma-exist-value by auto
have e8: ( $\exists l. (\text{upd } \sigma \ x1 \ l) \ i \ j \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite})$ )  $\implies$ 
  ( $\exists l. (\text{upd } (\text{upd } \sigma \ n \ (\lambda c. \sigma \ (\text{eint } c) \ (\text{eint } c) \models w)) \ x1 \ l) \ i \ j \models f$ )
proof -
  assume ae0: ( $\exists l. (\text{upd } \sigma \ x1 \ l) \ i \ j \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite})$ )
  show ( $\exists l. (\text{upd } (\text{upd } \sigma \ n \ (\lambda c. \sigma \ (\text{eint } c) \ (\text{eint } c) \models w)) \ x1 \ l) \ i \ j \models f$ )
proof -
  obtain l where e9: ( $\text{upd } \sigma \ x1 \ l) \ i \ j \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite})$ )
    using ae0 by blast
  have e10: ( $(\text{upd } (\text{upd } \sigma \ x1 \ l) \ n \ (\lambda c. (\text{upd } \sigma \ x1 \ l) \ (\text{eint } c) \ (\text{eint } c) \models w)) \ i \ j \models f$ )
    using exists-d.IH[of n w i j (upd σ x1 l)]
    using e4 e6 e9 exists-d.premis(3) exists-d.premis(4) exists-d.premis(5) exists-d.premis(6) by fastforce

  have e11: ( $\text{upd } (\text{upd } \sigma \ x1 \ l) \ n \ (\lambda c. (\text{upd } \sigma \ x1 \ l) \ (\text{eint } c) \ (\text{eint } c) \models w)) =$ 
     $\text{upd } (\text{upd } \sigma \ n \ (\lambda c. (\text{upd } \sigma \ x1 \ l) \ (\text{eint } c) \ (\text{eint } c) \models w)) \ x1 \ l$ 
  using upd-swap[of x1 n σ l (λc. (upd σ x1 l) (eint c) (eint c) ⊨ w)]
  using False by blast
  have e12: ( $\text{upd } (\text{upd } \sigma \ n \ (\lambda c. (\text{upd } \sigma \ x1 \ l) \ (\text{eint } c) \ (\text{eint } c) \models w)) \ x1 \ l =$ 
     $(\text{upd } (\text{upd } \sigma \ n \ (\lambda c. \sigma \ (\text{eint } c) \ (\text{eint } c) \models w)) \ x1 \ l)$ 
    using e60 not-fvar-upd by auto
  show ?thesis
  using e10 e11 e12 by auto
qed
qed
have e13: ( $\exists l. (\text{upd } (\text{upd } \sigma \ n \ (\lambda c. \sigma \ (\text{eint } c) \ (\text{eint } c) \models w)) \ x1 \ l) \ i \ j \models f$ )  $\implies$ 
  ( $\exists l. (\text{upd } \sigma \ x1 \ l) \ i \ j \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite})$ )
proof -
  assume ae1: ( $\exists l. (\text{upd } (\text{upd } \sigma \ n \ (\lambda c. \sigma \ (\text{eint } c) \ (\text{eint } c) \models w)) \ x1 \ l) \ i \ j \models f$ )
  show ( $\exists l. (\text{upd } \sigma \ x1 \ l) \ i \ j \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite})$ )
proof -
  obtain l where e14: ( $\text{upd } (\text{upd } \sigma \ n \ (\lambda c. \sigma \ (\text{eint } c) \ (\text{eint } c) \models w)) \ x1 \ l) \ i \ j \models f$ )
    using ae1 by blast
  have e15: ( $\text{upd } (\text{upd } \sigma \ n \ (\lambda c. (\text{upd } \sigma \ x1 \ l) \ (\text{eint } c) \ (\text{eint } c) \models w)) \ x1 \ l =$ 
     $(\text{upd } (\text{upd } \sigma \ n \ (\lambda c. \sigma \ (\text{eint } c) \ (\text{eint } c) \models w)) \ x1 \ l)$ 
    using e60 not-fvar-upd by auto
  have e16: ( $\text{upd } (\text{upd } \sigma \ n \ (\lambda c. (\text{upd } \sigma \ x1 \ l) \ (\text{eint } c) \ (\text{eint } c) \models w)) \ x1 \ l =$ 
     $(\text{upd } (\text{upd } \sigma \ x1 \ l) \ n \ (\lambda c. (\text{upd } \sigma \ x1 \ l) \ (\text{eint } c) \ (\text{eint } c) \models w))$ 
    using False upd-swap by auto
  have e17: ( $(\text{upd } (\text{upd } \sigma \ x1 \ l) \ n \ (\lambda c. (\text{upd } \sigma \ x1 \ l) \ (\text{eint } c) \ (\text{eint } c) \models w)) \ i \ j \models f$ )
    using e14 e15 e16 by auto
  have e18: ( $\text{upd } \sigma \ x1 \ l) \ i \ j \models \text{suform } f \ n \ (w \ \text{iand} \ \text{lfinite})$ )
    by (simp add: e17 e4 e6 exists-d.IH exists-d.premis(3) exists-d.premis(4) exists-d.premis(5) exists-d.premis(6))
  show ?thesis

```

```

    using e18 by auto
  qed
qed
show ?thesis
using e13 e5 e7 e8 by auto
qed
qed
qed

```

lemma *state-nl-itl-point-interval*:

fixes $i :: \text{int}$

assumes *state-nl-itl w*

$\sigma' i = \sigma i$

shows $(\sigma' i i \models w) \longleftrightarrow (\sigma i i \models w)$

using *assms*

proof (*induction w arbitrary: $\sigma \sigma' i$*)

case *false-d*

then show *?case* **by** *simp*

next

case (*atom-d x*)

then show *?case* **by** *simp*

next

case (*iimp-d w1 w2*)

then show *?case*

using *semantics-nl-itl.simps(3) state-nl-itl.simps(3)* **by** *blast*

next

case (*next-d w*)

then show *?case* **by** *simp*

next

case (*chop-d w1 w2*)

then show *?case* **by** *simp*

next

case (*ldiamond-d w*)

then show *?case* **by** *simp*

next

case (*rdiamond-d w*)

then show *?case* **by** *simp*

next

case (*chopstar-d w*)

then show *?case* **by** *simp*

next

case (*exists-d x1 w*)

then show *?case*

proof –

have 1: $(\sigma (eint i) (eint i) \models Ex x1. w) = (\exists l. ((upd \sigma x1 l) (eint i) (eint i) \models w))$ (**is** - = *?lhs*)

by *simp*

have 2: $(\sigma' (eint i) (eint i) \models Ex x1. w) = (\exists l. ((upd \sigma' x1 l) (eint i) (eint i) \models w))$ (**is** - = *?rhs*)

by *simp*

have 3: *?lhs* \implies *?rhs*

proof –

```

assume  $a$ : ?lhs
show ?rhs
proof –
  obtain  $l$  where 4:  $((\text{upd } \sigma \ x1 \ l) \ (eint \ i) \ (eint \ i) \models w)$ 
    using  $a$  by blast
  have 5:  $(\text{upd } \sigma \ x1 \ l) \ i = (\text{upd } \sigma' \ x1 \ l) \ i$ 
    using exists-d.premis(2) upd-def by presburger
  show ?thesis
    using 4 5 exists-d.IH exists-d.premis(1) state-nl-itl.simps(9) by blast
qed
qed
have 4: ?rhs  $\implies$  ?lhs
proof –
  assume  $b$ : ?rhs
  show ?lhs
  proof –
    obtain  $l$  where 6:  $((\text{upd } \sigma' \ x1 \ l) \ (eint \ i) \ (eint \ i) \models w)$ 
      using  $b$  by blast
    have 7:  $(\text{upd } \sigma' \ x1 \ l) \ i = (\text{upd } \sigma \ x1 \ l) \ i$ 
      using exists-d.premis(2) upd-def by presburger
    show ?thesis
      using 6 7 exists-d.IH exists-d.premis(1) state-nl-itl.simps(9) by blast
    qed
  qed
  show ?thesis
    using 1 2 3 4 by blast
  qed
qed

```

lemma exists-elim-state-nl-itl-help:

assumes

$i \neq \infty$
 $state-nl-itl \ w$
 $n \notin bvars \ w$
 $i \neq -\infty$

shows $(\exists l. ((\text{upd } \sigma \ n \ l) \ i \ i \models w)) \longleftrightarrow$
 $((\text{upd } \sigma \ n \ (\lambda x::int. \text{True})) \ i \ i \models w) \vee ((\text{upd } \sigma \ n \ (\lambda x::int. \text{False})) \ i \ i \models w)$

using assms

proof (induction w arbitrary: $\sigma \ n \ i$)

case false-d

then show ?case **by** simp

next

case (atom-d x)

then show ?case **using** upd-def **by** auto

next

case (iimp-d $w1 \ w2$)

then show ?case

proof –

have 1: state-nl-itl $w1$

```

using iimp-d.prems(2) by auto
have 2: state-nl-itl w2
using iimp-d.prems(2) by auto
have 3:  $n \notin \text{bvars } w1$ 
using iimp-d.prems(3) by simp
have 4:  $n \notin \text{bvars } w2$ 
using iimp-d.prems(3) by simp
have 5:  $(\exists l. (\text{upd } \sigma \ n \ l) \ i \ i \models w1 \ iimp \ w2) =$ 
 $(\exists l. ((\text{upd } \sigma \ n \ l) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ l) \ i \ i \models w2))$ 
by simp
have 6:  $((\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i \ i \models w1 \ iimp \ w2) =$ 
 $((\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i \ i \models w2)$ 
by simp
have 7:  $((\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i \ i \models w1 \ iimp \ w2) =$ 
 $((\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i \ i \models w2)$ 
by simp
have 8:  $((\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i \ i \models w2) \implies$ 
 $(\exists l. ((\text{upd } \sigma \ n \ l) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ l) \ i \ i \models w2))$ 
by blast
have 9:  $((\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i \ i \models w2) \implies$ 
 $(\exists l. ((\text{upd } \sigma \ n \ l) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ l) \ i \ i \models w2))$ 
by blast
have 10:  $(\exists l. ((\text{upd } \sigma \ n \ l) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ l) \ i \ i \models w2)) \implies$ 
 $((\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i \ i \models w2) \vee$ 
 $((\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i \ i \models w2)$ 
proof –
assume a1:  $(\exists l. ((\text{upd } \sigma \ n \ l) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ l) \ i \ i \models w2))$ 
show  $((\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i \ i \models w2) \vee$ 
 $((\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i \ i \models w2)$ 
proof –
obtain l where 11:  $((\text{upd } \sigma \ n \ l) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ l) \ i \ i \models w2)$ 
using a1 by blast
have 111:  $\exists i1. \text{eint } i1 = i$ 
by (metis MInfty-eq-minfinity PInfty-eq-infinity eint.exhaust iimp-d.prems(1,4))
obtain i1 where 112:  $\text{eint } i1 = i$ 
using 111 by blast
have 12:  $((\text{upd } \sigma \ n \ l) \ i \ i \models w1) \implies$ 
 $((\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i \ i \models w1) \vee ((\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i \ i \models w1)$ 
using 1 3 iimp-d.IH(1) iimp-d.prems(1)
using iimp-d.prems(4) by blast
have 13:  $((\text{upd } \sigma \ n \ l) \ i \ i \models w2) \implies$ 
 $((\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i \ i \models w2) \vee ((\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i \ i \models w2)$ 
using 2 4 iimp-d.IH(2) iimp-d.prems(1) using iimp-d.prems(4) by blast
have 14:  $((\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i \ i \models w2) \vee ((\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i \ i \models w2) \implies$ 
 $((\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i \ i \models w2) \vee ((\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i \ i \models w2)$ 
by (simp add: 11 13)
have 141:  $(\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i1 \neq (\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i1$ 
unfolding upd-def by auto
have 15:  $((\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ (\lambda x. \text{True})) \ i \ i \models w2) \vee$ 
 $((\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i \ i \models w1) \longrightarrow ((\text{upd } \sigma \ n \ (\lambda x. \text{False})) \ i \ i \models w2)$ 

```



```

    using 1 11 12 13 112 141 iimp-d.premis state-nl-itl-point-interval[of w1 - i1]
    by (metis upd-def)
    show ?thesis
    using 15 by blast
qed
qed
show ?thesis
using 10 5 6 7 by blast
qed
next
case (next-d w)
then show ?case by simp
next
case (chop-d w1 w2)
then show ?case by simp
next
case (ldiamond-d w)
then show ?case by simp
next
case (rdiamond-d w)
then show ?case by simp
next
case (chopstar-d w)
then show ?case by simp
next
case (exists-d x1 w)
then show ?case apply simp
by (metis upd-swap)
qed

```

lemma *suform-lfinite-export:*

```

  ⊢ suform (f iand lfinite) n g ieqv (lfinite iand (suform f n g))
unfolding valid-def unfolding iand-d-def lfinite-d-def inot-d-def ior-d-def ittrue-d-def empty-d-def more-d-def
apply simp
by meson

```

lemma *subst-suform-state:*

```

assumes n ∉ bvars f
  ∧ z . z ∈ fvars w ⇒ z ∉ bvars f
  state-nl-itl w
  i ≤ j
  i ≠ ∞
  j ≠ -∞
  state-nl-itl f
shows (σ i j ⊨ suform (f iand lfinite) n (w)) = (σ i j ⊨ suform (f iand lfinite) n (w iand lfinite))
using assms
proof (induction f arbitrary: σ )
case false-d

```

```

then show ?case
using itl-ieq suform.simps(1) suform-lfinite-export by presburger
next
case (atom-d x)
then show ?case
using suform-lfinite-export by auto
next
case (iimp-d f1 f2)
then show ?case
proof -
  have i1:  $(\sigma \ i \ j \models \text{suform } ((\text{inot } f1) \ \text{iand } \text{lfinite}) \ n \ w) = (\sigma \ i \ j \models \text{suform } ((\text{inot } f1) \ \text{iand } \text{lfinite}) \ n \ (w \ \text{iand } \text{lfinite}))$ 
  using iimp-d inot-d-def suform-lfinite-export by force
  have i2:  $(\sigma \ i \ j \models \text{suform } (f2 \ \text{iand } \text{lfinite}) \ n \ w) = (\sigma \ i \ j \models \text{suform } (f2 \ \text{iand } \text{lfinite}) \ n \ (w \ \text{iand } \text{lfinite}))$ 
  using iimp-d by force
  show ?thesis
  using i1 i2 iimp-d.premis(4) iimp-d.premis(5) iimp-d.premis(6) inot-d-def suform-lfinite-export by auto
qed
next
case (next-d f)
then show ?case by simp
next
case (chop-d f1 f2)
then show ?case by simp
next
case (ldiamond-d f)
then show ?case by simp
next
case (rdiamond-d f)
then show ?case by simp
next
case (chopstar-d f)
then show ?case by simp
next
case (exists-d x1 f)
then show ?case
proof -
  have e1:  $(\sigma \ i \ j \models \text{suform } ((\text{Ex } x1. \ f) \ \text{iand } \text{lfinite}) \ n \ w) =$ 
     $(\sigma \ i \ j \models \text{suform } ((\text{Ex } x1. \ f \ \text{iand } \text{lfinite})) \ n \ w)$ 
  using exists-d.premis(4) exists-d.premis(5) exists-d.premis(6) suform-lfinite-export by auto
  have e2:  $(\sigma \ i \ j \models \text{suform } ((\text{Ex } x1. \ f) \ \text{iand } \text{lfinite}) \ n \ (w \ \text{iand } \text{lfinite})) =$ 
     $(\sigma \ i \ j \models \text{suform } ((\text{Ex } x1. \ f \ \text{iand } \text{lfinite})) \ n \ (w \ \text{iand } \text{lfinite}))$ 
  using exists-d.premis(4) exists-d.premis(5) exists-d.premis(6) suform-lfinite-export by force
  have e3:  $(\sigma \ i \ j \models \text{suform } (f \ \text{iand } \text{lfinite}) \ n \ w) = (\sigma \ i \ j \models \text{suform } (f \ \text{iand } \text{lfinite}) \ n \ (w \ \text{iand } \text{lfinite}))$ 
  using exists-d by auto
  show ?thesis
  using e1 e2 exists-d.IH exists-d.premis(1) exists-d.premis(2) exists-d.premis(3) exists-d.premis(4) exists-d.premis(5) exists-d.premis(6) exists-d.premis(7) by force
qed
qed

```

lemma *subst-suform-state-1:*

assumes $n \notin \text{bvars } f$

$\bigwedge z. z \in \text{fvars } w \implies z \notin \text{bvars } f$

state-nl-itl w

$i \leq j$

$i \neq \infty$

$j \neq -\infty$

state-nl-itl f

shows $(\sigma \ i \ j \models \text{suform } (f \text{ iand } l\text{finite}) \ n \ (w)) = ((\text{upd } \sigma \ n \ (\lambda c. (\sigma \ c \ c \models w))) \ i \ j \models f \text{ iand } l\text{finite})$

proof –

have 1: $(n::\text{nat}) \notin \text{bvars } ((f::\text{nl-itl}) \text{ iand } l\text{finite})$

unfolding *iand-d-def inot-d-def lfinite-d-def ittrue-d-def empty-d-def more-d-def*

by (*simp add: assms(1) inot-d-def ior-d-def*)

have 2: $(\bigwedge z::\text{nat}. z \in \text{fvars } (w::\text{nl-itl}) \implies z \notin \text{bvars } (f \text{ iand } l\text{finite}))$

unfolding *iand-d-def inot-d-def lfinite-d-def ittrue-d-def empty-d-def more-d-def ior-d-def*

by (*simp add: assms(2)*)

have 3: $(\sigma \ i \ j \models \text{suform } (f \text{ iand } l\text{finite}) \ n \ (w \text{ iand } l\text{finite})) = ((\text{upd } \sigma \ n \ (\lambda c. (\sigma \ c \ c \models w))) \ i \ j \models f \text{ iand } l\text{finite})$

using *subst-suform[of n f iand lfinite w i j σ] assms*

using 1 2 **by** *blast*

have 4: $(\sigma \ i \ j \models \text{suform } (f \text{ iand } l\text{finite}) \ n \ (w \text{ iand } l\text{finite})) =$

$(\sigma \ i \ j \models \text{suform } (f \text{ iand } l\text{finite}) \ n \ (w))$

by (*simp add: assms(1) assms(2) assms(3) assms(4) assms(5) assms(6) assms(7) subst-suform-state*)

show *?thesis*

using 3 4 **by** *blast*

qed

lemma *rename-nl-pitl:*

assumes $v1 \notin \text{bvars } f$

$v2 \notin \text{bvars } f$

$v1 \neq v2$

$v2 \notin \text{fvars } f$

$i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models \text{Ex } v1. f) = (\sigma \ i \ j \models \text{Ex } v2. \text{suform } f \ v1 \ (\$v2 \text{ iand } l\text{finite}))$

proof –

have 1: $(\sigma \ i \ j \models \text{Ex } v1. f) = (\exists l. ((\text{upd } \sigma \ v1 \ l) \ i \ j \models f))$

using *exist-sigma-exist-value* **by** *auto*

have 2: $(\sigma \ i \ j \models \text{Ex } v2. \text{suform } f \ v1 \ (\$v2 \text{ iand } l\text{finite})) =$

$(\exists l. ((\text{upd } \sigma \ v2 \ l) \ i \ j \models \text{suform } f \ v1 \ (\$v2 \text{ iand } l\text{finite})))$

using *exist-sigma-exist-value* **by** *auto*

have 3: $(\exists l. ((\text{upd } \sigma \ v2 \ l) \ i \ j \models \text{suform } f \ v1 \ (\$v2 \text{ iand } l\text{finite}))) \implies$

$(\exists l. ((\text{upd } \sigma \ v1 \ l) \ i \ j \models f))$

proof –

```

assume a0: ( $\exists l. ((\text{upd } \sigma \ v2 \ l) \ i \ j \models \text{suform } f \ v1 \ (\$v2 \ \text{iand} \ \text{lfinite}) \ ))$ )
show ( $\exists l. ((\text{upd } \sigma \ v1 \ l) \ i \ j \models f)$ )
proof –
obtain l where 4: ( $((\text{upd } \sigma \ v2 \ l) \ i \ j \models \text{suform } f \ v1 \ (\$v2 \ \text{iand} \ \text{lfinite}) \ ))$ )
using a0 by blast
have 5: ( $((\text{upd } \sigma \ v2 \ l) \ i \ j \models \text{suform } f \ v1 \ (\$v2 \ \text{iand} \ \text{lfinite}) \ )) =$ 
 $((\text{upd } (\text{upd } \sigma \ v2 \ l) \ v1 \ (\lambda x::\text{int}. (\text{upd } \sigma \ v2 \ l) \ (\text{eint } x) \ (\text{eint } x) \models \$v2)) \ i \ j \models f)$ )
using subst-suform[of v1 f $v2 i j (upd  $\sigma$  v2 l)]
by (simp add: assms(1) assms(2) assms(5) assms(6) assms(7))
have 6: ( $(\text{upd } (\text{upd } \sigma \ v2 \ l) \ v1 \ (\lambda x::\text{int}. (\text{upd } \sigma \ v2 \ l) \ (\text{eint } x) \ (\text{eint } x) \models \$v2)) =$ 
 $\text{upd } (\text{upd } \sigma \ v1 \ (\lambda x::\text{int}. (\text{upd } \sigma \ v2 \ l) \ (\text{eint } x) \ (\text{eint } x) \models \$v2)) \ v2 \ l$ )
using upd-swap[of v2 v1  $\sigma$  l ( $\lambda x::\text{int}. (\text{upd } \sigma \ v2 \ l) \ (\text{eint } x) \ (\text{eint } x) \models \$v2$ )]
using assms(3) by blast
have 8: ( $((\text{upd } (\text{upd } \sigma \ v1 \ (\lambda x::\text{int}. (\text{upd } \sigma \ v2 \ l) \ (\text{eint } x) \ (\text{eint } x) \models \$v2)) \ v2 \ l) \ i \ j \models f) =$ 
 $((\text{upd } \sigma \ v1 \ (\lambda x::\text{int}. (\text{upd } \sigma \ v2 \ l) \ (\text{eint } x) \ (\text{eint } x) \models \$v2)) \ i \ j \models f)$ )
using not-fvar-upd[of v2 f] using assms(4) by blast
show ?thesis using 4 5 6 8 by auto
qed
qed
have 9: ( $\exists l. ((\text{upd } \sigma \ v1 \ l) \ i \ j \models f) \implies$ 
 $(\exists l. ((\text{upd } \sigma \ v2 \ l) \ i \ j \models \text{suform } f \ v1 \ (\$v2 \ \text{iand} \ \text{lfinite}) \ ))$ )
proof –
assume a1: ( $\exists l. ((\text{upd } \sigma \ v1 \ l) \ i \ j \models f)$ )
show ( $\exists l. ((\text{upd } \sigma \ v2 \ l) \ i \ j \models \text{suform } f \ v1 \ (\$v2 \ \text{iand} \ \text{lfinite}) \ ))$ )
proof –
obtain l where 10: ( $((\text{upd } \sigma \ v1 \ l) \ i \ j \models f)$ )
using a1 by blast
have 11: ( $((\text{upd } \sigma \ v1 \ (\lambda x::\text{int}. (\text{upd } \sigma \ v2 \ l) \ (\text{eint } x) \ (\text{eint } x) \models \$v2)) \ i \ j \models f) =$ 
 $((\text{upd } (\text{upd } \sigma \ v1 \ (\lambda x::\text{int}. (\text{upd } \sigma \ v2 \ l) \ (\text{eint } x) \ (\text{eint } x) \models \$v2)) \ v2 \ l) \ i \ j \models f)$ )

using not-fvar-upd[of v2 f] using assms(4) by blast
have 12:  $\bigwedge k. (\text{upd } \sigma \ v1 \ l) \ k = (\text{upd } \sigma \ v1 \ (\lambda x::\text{int}. (\text{upd } \sigma \ v2 \ l) \ (\text{eint } x) \ (\text{eint } x) \models \$v2)) \ k$ 
using upd-def by fastforce
have 13:  $(\text{upd } \sigma \ v1 \ l) = (\text{upd } \sigma \ v1 \ (\lambda x::\text{int}. (\text{upd } \sigma \ v2 \ l) \ (\text{eint } x) \ (\text{eint } x) \models \$v2))$ 
using 12 by blast
have 14: ( $((\text{upd } \sigma \ v2 \ l) \ i \ j \models \text{suform } f \ v1 \ (\$v2 \ \text{iand} \ \text{lfinite}) \ )) =$ 
 $((\text{upd } (\text{upd } \sigma \ v2 \ l) \ v1 \ (\lambda x::\text{int}. (\text{upd } \sigma \ v2 \ l) \ (\text{eint } x) \ (\text{eint } x) \models \$v2)) \ i \ j \models f)$ )
using subst-suform[of v1 f $v2 i j (upd  $\sigma$  v2 l)]
by (simp add: assms(1) assms(2) assms(5) assms(6) assms(7))
have 15: ( $(\text{upd } (\text{upd } \sigma \ v2 \ l) \ v1 \ (\lambda x::\text{int}. (\text{upd } \sigma \ v2 \ l) \ (\text{eint } x) \ (\text{eint } x) \models \$v2)) =$ 
 $(\text{upd } (\text{upd } \sigma \ v1 \ (\lambda x::\text{int}. (\text{upd } \sigma \ v2 \ l) \ (\text{eint } x) \ (\text{eint } x) \models \$v2)) \ v2 \ l)$ )
using upd-swap[of v2 v1  $\sigma$  l ( $\lambda x::\text{int}. (\text{upd } \sigma \ v2 \ l) \ (\text{eint } x) \ (\text{eint } x) \models \$v2$ )]
using assms(3) by blast
show ?thesis
using 10 11 13 14 15 by auto
qed
qed
show ?thesis
using 1 2 3 9 by blast
qed

```

lemma *atom-lfinite-absorb*:

$\vdash (\$n) \text{ ieqv } \$n \text{ iand lfinite}$

unfolding *valid-def* **apply** *simp* **by** *meson*

lemma *suform-ieqv-sem*:

assumes $\bigwedge \sigma \ i \ j . i \leq j \wedge i \neq \infty \wedge j \neq -\infty \implies (\sigma \ i \ j \models f) = (\sigma \ i \ j \models g)$

$i \leq j$

$i \neq \infty$

$j \neq -\infty$

shows $(\sigma \ i \ j \models (\text{suform } h \ n \ f)) = (\sigma \ i \ j \models (\text{suform } h \ n \ g))$

using *assms*

proof (*induction* *h* *arbitrary*: $\sigma \ i \ j$)

case *false-d*

then show *?case* **by** *simp*

next

case (*atom-d* *x*)

then show *?case* **by** *simp*

next

case (*iimp-d* *h1* *h2*)

then show *?case*

proof –

have *i1*: $(\sigma \ i \ j \models \text{suform } (\text{inot } h1) \ n \ f \text{ ieqv } \text{suform } (\text{inot } h1) \ n \ g)$

using *assms iimp-d.IH(1)* **unfolding** *inot-d-def* **using** *iimp-d.prem*s **by** *auto*

have *i2*: $(\sigma \ i \ j \models \text{suform } (h2) \ n \ f \text{ ieqv } \text{suform } (h2) \ n \ g)$

using *assms iimp-d.IH(1) iimp-d.prem*s **using** *iimp-d.IH(2) ieqv-defs* **by** *presburger*

show *?thesis* **using** *i1 i2 inot-d-def* **by** *auto*

qed

next

case (*next-d* *h*)

then show *?case*

proof –

have *n1*: $(\sigma \ i \ j \models \text{suform } (\text{next } h) \ n \ f) = (i \neq -\infty \wedge i < j \wedge (\sigma \ (i + \text{eint } (1)) \ j \models \text{suform } h \ n \ f))$

by *simp*

have *n2*: $(\sigma \ i \ j \models \text{suform } (\text{next } h) \ n \ g) = (i \neq -\infty \wedge i < j \wedge (\sigma \ (i + \text{eint } (1)) \ j \models \text{suform } h \ n \ g))$

by *simp*

have *n3*: $i \neq -\infty \wedge i < j \implies (\sigma \ (i + \text{eint } (1)) \ j \models \text{suform } h \ n \ f) = (\sigma \ (i + \text{eint } (1)) \ j \models \text{suform } h \ n \ g)$

using *next-d.IH[of (i + eint (1)) j sigma]* *next-d.prem*s **apply** *auto*

apply (*metis eint-less-le-alt int-of-eint.cases plus-eint.simps(1)*)

by (*metis eint-less-le-alt int-of-eint.cases plus-eint.simps(1)*)

show *?thesis*

using *n1 n2 n3* **by** *blast*

qed

next

case (*chop-d* *h1* *h2*)

then show *?case* **by** *auto*

next

case (*ldiamond-d* *h*)

```

then show ?case apply auto
apply (metis eint-less-eq(1) less-eq-eint-def linorder-not-le)
by (metis eint-less-eq(1) linorder-not-le order-less-le)
next
case (rdiamond-d h)
then show ?case apply auto
apply (metis less-eint.simps(3) linorder-not-le not-less-iff-gr-or-eq)
by (metis eint-less-eq(2) linorder-not-less order-less-le)
next
case (chopstar-d h)
then show ?case
proof -
  have c1:  $i = j \implies ?thesis$ 
  using semantics-nl-itl.simps(8) suform.simps(8) by presburger
  have c10:  $i \neq j \wedge i \neq -\infty \wedge j \neq \infty \implies ?thesis$ 
  proof -
    assume c10a:  $i \neq j \wedge i \neq -\infty \wedge j \neq \infty$ 
    show  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ f) = (\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ g)$ 
    proof -
      have c11:  $\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ f \implies \sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ g$ 
      proof -
        assume c11a:  $\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ f$ 
        show  $\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ g$ 
        proof -
          have c12:  $\exists \text{ lrf. } \text{nnth } \text{lrf } (0::\text{nat}) = i \wedge \text{nfinite } \text{lrf} \wedge \text{nlast } \text{lrf} = j \wedge \text{nridx } (\lambda(a::\text{eint}) \ b::\text{eint. } a < b \wedge (\sigma \ a \ b \models \text{suform } h \ n \ f)) \ \text{lrf}$ 
          using c10a c11a by simp
          obtain lrf where c13:  $\text{nnth } \text{lrf } (0::\text{nat}) = i \wedge \text{nfinite } \text{lrf} \wedge \text{nlast } \text{lrf} = j \wedge \text{nridx } (\lambda(a::\text{eint}) \ b::\text{eint. } a < b \wedge (\sigma \ a \ b \models \text{suform } h \ n \ f)) \ \text{lrf}$ 
          using c12 by blast
          have c14:  $\text{nridx } (\lambda(a::\text{eint}) \ b::\text{eint. } a < b \wedge (\sigma \ a \ b \models \text{suform } h \ n \ g)) \ \text{lrf}$ 
          using chopstar-d.IH c13 nridx-expand
          by (metis (no-types, lifting) chopstar-d.prem(1) eint-less-eq(1) less-eint.simps(3) linorder-not-le order-less-le)
          have c15:  $\exists \text{ lrf. } \text{nnth } \text{lrf } (0::\text{nat}) = i \wedge \text{nfinite } \text{lrf} \wedge \text{nlast } \text{lrf} = j \wedge \text{nridx } (\lambda(a::\text{eint}) \ b::\text{eint. } a < b \wedge (\sigma \ a \ b \models \text{suform } h \ n \ g)) \ \text{lrf}$ 
          using c14 c13 by blast
          show ?thesis using c15 c10a by simp
        qed
      qed
    qed
  have c16:  $\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ g \implies \sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ f$ 
  proof -
    assume c16a:  $\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ g$ 
    show  $\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ f$ 
    proof -
      have c17:  $\exists \text{ lrf. } \text{nnth } \text{lrf } (0::\text{nat}) = i \wedge \text{nfinite } \text{lrf} \wedge \text{nlast } \text{lrf} = j \wedge \text{nridx } (\lambda(a::\text{eint}) \ b::\text{eint. } a < b \wedge (\sigma \ a \ b \models \text{suform } h \ n \ g)) \ \text{lrf}$ 
      using c10a c16a by simp
      obtain lrf where c18:  $\text{nnth } \text{lrf } (0::\text{nat}) = i \wedge \text{nfinite } \text{lrf} \wedge \text{nlast } \text{lrf} = j \wedge \text{nridx } (\lambda(a::\text{eint}) \ b::\text{eint. } a < b \wedge (\sigma \ a \ b \models \text{suform } h \ n \ g)) \ \text{lrf}$ 

```

```

using c17 by blast
have c19:  $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint}. a < b \wedge (\sigma a b \models \text{suform } h \ n \ f)) \text{ lrf}$ 
using chopstar-d.IH c18 nridx-expand
by (metis (no-types, lifting) chopstar-d.prem1 eint-less-eq(1) less-eint.simps(3) linorder-not-le
order-less-le)
have c19a:  $\exists \text{lrf}. \text{nnth } \text{lrf } (0::\text{nat}) = i \wedge \text{nfinite } \text{lrf} \wedge \text{nlast } \text{lrf} = j \wedge \text{nridx } (\lambda(a::\text{eint}) b::\text{eint}. a < b \wedge (\sigma a b \models \text{suform } h \ n \ f)) \text{ lrf}$ 
using c18 c19 by blast
show ?thesis using c19a c10a by simp
qed
qed
show ?thesis
using c11 c16 by blast
qed
qed
have c20:  $i \neq j \wedge i = -\infty \wedge j \neq \infty \implies ?thesis$ 
proof –
assume c20a:  $i \neq j \wedge i = -\infty \wedge j \neq \infty$ 
show  $(\sigma i \ j \models \text{suform } (\text{chopstar } h) \ n \ f) = (\sigma i \ j \models \text{suform } (\text{chopstar } h) \ n \ g)$ 
proof –
have c21:  $\sigma i \ j \models \text{suform } (\text{chopstar } h) \ n \ f \implies \sigma i \ j \models \text{suform } (\text{chopstar } h) \ n \ g$ 
proof –
assume c21a:  $\sigma i \ j \models \text{suform } (\text{chopstar } h) \ n \ f$ 
show  $\sigma i \ j \models \text{suform } (\text{chopstar } h) \ n \ g$ 
proof –
have c22:  $(\exists li::\text{eint} \text{ nellist}. \text{nnth } li \ (0::\text{nat}) = j \wedge \neg \text{nfinite } li \wedge \text{nridx } (\lambda(a::\text{eint}) b::\text{eint}. b < a \wedge (\sigma b \ a \models \text{suform } h \ n \ f)) \ li)$ 
using c20a c21a by fastforce
obtain li where c23:  $\text{nnth } li \ (0::\text{nat}) = j \wedge \neg \text{nfinite } li \wedge \text{nridx } (\lambda(a::\text{eint}) b::\text{eint}. b < a \wedge (\sigma b \ a \models \text{suform } h \ n \ f)) \ li$ 
using c22 by blast
have c24:  $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint}. b < a \wedge (\sigma b \ a \models \text{suform } h \ n \ g)) \ li$ 
by (metis (no-types, lifting) c23 chopstar-d.IH chopstar-d.prem1 eint-less-eq(1) less-eint.simps(3)
linorder-not-le nridx-expand order-less-le)
show ?thesis using c20a c23 c24 by auto
qed
qed
have c25:  $\sigma i \ j \models \text{suform } (\text{chopstar } h) \ n \ g \implies \sigma i \ j \models \text{suform } (\text{chopstar } h) \ n \ f$ 
proof –
assume c25a:  $\sigma i \ j \models \text{suform } (\text{chopstar } h) \ n \ g$ 
show  $\sigma i \ j \models \text{suform } (\text{chopstar } h) \ n \ f$ 
proof –
have c26:  $(\exists li::\text{eint} \text{ nellist}. \text{nnth } li \ (0::\text{nat}) = j \wedge \neg \text{nfinite } li \wedge \text{nridx } (\lambda(a::\text{eint}) b::\text{eint}. b < a \wedge (\sigma b \ a \models \text{suform } h \ n \ g)) \ li)$ 
using c25a c20a by fastforce
obtain li where c27:  $\text{nnth } li \ (0::\text{nat}) = j \wedge \neg \text{nfinite } li \wedge \text{nridx } (\lambda(a::\text{eint}) b::\text{eint}. b < a \wedge (\sigma b \ a \models \text{suform } h \ n \ g)) \ li$ 
using c26 by blast
have c28:  $\text{nridx } (\lambda(a::\text{eint}) b::\text{eint}. b < a \wedge (\sigma b \ a \models \text{suform } h \ n \ f)) \ li$ 
by (metis (no-types, lifting) c27 chopstar-d.IH chopstar-d.prem1 eint-less-eq(1) less-eint.simps(3)

```

```

linorder-not-le nridx-expand order-less-le)
  show ?thesis using c20a c27 c28 by auto
qed
qed
show ?thesis
using c21 c25 by blast
qed
qed
have c30:  $i \neq j \wedge i \neq -\infty \wedge j = \infty \implies ?thesis$ 
proof -
  assume c30a:  $i \neq j \wedge i \neq -\infty \wedge j = \infty$ 
  show  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ f) = (\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ g)$ 
  proof -
    have c31:  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ f) \implies (\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ g)$ 
    proof -
      assume c31a:  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ f)$ 
      show  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ g)$ 
      proof -
        have c32:  $(\exists ri::\text{eint} \ \text{nellist}. \ \text{nnth } ri \ (0::\text{nat}) = i \wedge \neg \text{nfinite } ri \wedge \text{nridx } (\lambda(a::\text{eint}) \ b::\text{eint}. \ a < b \wedge$ 
 $(\sigma \ a \ b \models \text{suform } h \ n \ f)) \ ri)$ 
        using c30a c31a by auto
        obtain ri where c33:  $\text{nnth } ri \ (0::\text{nat}) = i \wedge \neg \text{nfinite } ri \wedge \text{nridx } (\lambda(a::\text{eint}) \ b::\text{eint}. \ a < b \wedge (\sigma \ a$ 
 $b \models \text{suform } h \ n \ f)) \ ri$ 
        using c32 by blast
        have c34:  $\text{nridx } (\lambda(a::\text{eint}) \ b::\text{eint}. \ a < b \wedge (\sigma \ a \ b \models \text{suform } h \ n \ g)) \ ri$ 
        by (metis (no-types, lifting) c33 chopstar-d.IH chopstar-d.prem1 eint-less-eq(1) less-eint.simps(3))
linorder-not-le nridx-expand order-less-le)
      show ?thesis using c33 c34 c30a by auto
    qed
  qed
  have c35:  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ g) \implies (\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ f)$ 
  proof -
    assume c35a:  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ g)$ 
    show  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ f)$ 
    proof -
      have c36:  $(\exists ri::\text{eint} \ \text{nellist}. \ \text{nnth } ri \ (0::\text{nat}) = i \wedge \neg \text{nfinite } ri \wedge \text{nridx } (\lambda(a::\text{eint}) \ b::\text{eint}. \ a < b \wedge$ 
 $(\sigma \ a \ b \models \text{suform } h \ n \ g)) \ ri)$ 
      using c30a c35a by auto
      obtain ri where c37:  $\text{nnth } ri \ (0::\text{nat}) = i \wedge \neg \text{nfinite } ri \wedge \text{nridx } (\lambda(a::\text{eint}) \ b::\text{eint}. \ a < b \wedge (\sigma \ a$ 
 $b \models \text{suform } h \ n \ g)) \ ri$ 
      using c36 by blast
      have c38:  $\text{nridx } (\lambda(a::\text{eint}) \ b::\text{eint}. \ a < b \wedge (\sigma \ a \ b \models \text{suform } h \ n \ f)) \ ri$ 
      by (metis (no-types, lifting) c37 chopstar-d.IH chopstar-d.prem1 eint-less-PIfty less-eint.simps(3))
not-less-iff-gr-or-eq nridx-expand order-less-le)
    show ?thesis using c37 c38 c30a by auto
  qed
qed
show ?thesis using c35 c31 by blast
qed
qed

```



```

have c40:  $i \neq j \wedge i = -\infty \wedge j = \infty \implies ?thesis$ 
proof -
  assume c40a:  $i \neq j \wedge i = -\infty \wedge j = \infty$ 
  show  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ f) = (\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ g)$ 
  proof -
    have c41:  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ f) \implies (\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ g)$ 
    proof -
      assume c41a:  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ f)$ 
      show  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ g)$ 
      proof -
        have c42:  $\exists (li::\text{int nellist}) \ ri::\text{int nellist}.$ 
           $nnth \ li \ (0::\text{nat}) = nnth \ ri \ (0::\text{nat}) \wedge \neg \text{nfinite } li \wedge$ 
           $nridx \ (\lambda(a::\text{int}) \ b::\text{int}. \ b < a \wedge (\sigma \ (eint \ b) \ (eint \ a) \models \text{suform } h \ n \ f)) \ li \wedge$ 
           $\neg \text{nfinite } ri \wedge$ 
           $nridx \ (\lambda(a::\text{int}) \ b::\text{int}. \ a < b \wedge (\sigma \ (eint \ a) \ (eint \ b) \models \text{suform } h \ n \ f)) \ ri$ 
        using c40a c41a by simp
      obtain li and ri where c43:  $nnth \ li \ (0::\text{nat}) = nnth \ ri \ (0::\text{nat}) \wedge \neg \text{nfinite } li \wedge$ 
           $nridx \ (\lambda(a::\text{int}) \ b::\text{int}. \ b < a \wedge (\sigma \ (eint \ b) \ (eint \ a) \models \text{suform } h \ n \ f)) \ li \wedge$ 
           $\neg \text{nfinite } ri \wedge$ 
           $nridx \ (\lambda(a::\text{int}) \ b::\text{int}. \ a < b \wedge (\sigma \ (eint \ a) \ (eint \ b) \models \text{suform } h \ n \ f)) \ ri$ 
      using c42 by blast
      have c44a:  $nridx \ (\lambda(a::\text{int}) \ b::\text{int}. \ b < a \wedge (\sigma \ (eint \ b) \ (eint \ a) \models \text{suform } h \ n \ g)) \ li$ 
      by (metis (no-types, lifting) PInfty-neq-eint(2) c43 chopstar-d.IH chopstar-d.prem(1) less-eint.simps(1)
less-eint.simps(3) less-eq-eint-def nridx-expand)
      have c44b:  $nridx \ (\lambda(a::\text{int}) \ b::\text{int}. \ a < b \wedge (\sigma \ (eint \ a) \ (eint \ b) \models \text{suform } h \ n \ g)) \ ri$ 
      by (metis (no-types, lifting) PInfty-neq-eint(2) c43 chopstar-d.IH chopstar-d.prem(1) less-eint.simps(1)
less-eint.simps(3) less-eq-eint-def nridx-expand)
      show ?thesis using c40a c44a c44b c43 by auto
    qed
  qed
have c45:  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ g) \implies (\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ f)$ 
proof -
  assume c45a:  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ g)$ 
  show  $(\sigma \ i \ j \models \text{suform } (\text{chopstar } h) \ n \ f)$ 
  proof -
    have c46:  $\exists (li::\text{int nellist}) \ ri::\text{int nellist}.$ 
       $nnth \ li \ (0::\text{nat}) = nnth \ ri \ (0::\text{nat}) \wedge \neg \text{nfinite } li \wedge$ 
       $nridx \ (\lambda(a::\text{int}) \ b::\text{int}. \ b < a \wedge (\sigma \ (eint \ b) \ (eint \ a) \models \text{suform } h \ n \ g)) \ li \wedge$ 
       $\neg \text{nfinite } ri \wedge$ 
       $nridx \ (\lambda(a::\text{int}) \ b::\text{int}. \ a < b \wedge (\sigma \ (eint \ a) \ (eint \ b) \models \text{suform } h \ n \ g)) \ ri$ 
    using c40a c45a by simp
  obtain li and ri where c47:  $nnth \ li \ (0::\text{nat}) = nnth \ ri \ (0::\text{nat}) \wedge \neg \text{nfinite } li \wedge$ 
       $nridx \ (\lambda(a::\text{int}) \ b::\text{int}. \ b < a \wedge (\sigma \ (eint \ b) \ (eint \ a) \models \text{suform } h \ n \ g)) \ li \wedge$ 
       $\neg \text{nfinite } ri \wedge$ 
       $nridx \ (\lambda(a::\text{int}) \ b::\text{int}. \ a < b \wedge (\sigma \ (eint \ a) \ (eint \ b) \models \text{suform } h \ n \ g)) \ ri$ 
    using c46 by blast
    have c48a:  $nridx \ (\lambda(a::\text{int}) \ b::\text{int}. \ b < a \wedge (\sigma \ (eint \ b) \ (eint \ a) \models \text{suform } h \ n \ f)) \ li$ 
    by (metis (no-types, lifting) PInfty-neq-eint(2) c47 chopstar-d.IH chopstar-d.prem(1) less-eint.simps(1)
less-eint.simps(3) less-eq-eint-def nridx-expand)
    have c48b:  $nridx \ (\lambda(a::\text{int}) \ b::\text{int}. \ a < b \wedge (\sigma \ (eint \ a) \ (eint \ b) \models \text{suform } h \ n \ f)) \ ri$ 

```

```

    by (metis (no-types, lifting) PInfty-neq-eint(2) c47 chopstar-d.IH chopstar-d.prem(1) less-eint.simps(1)
less-eint.simps(3) less-eq-eint-def nridx-expand)
    show ?thesis using c40a c48a c48b c47 by auto
  qed
  qed
  show ?thesis using c45 c41 by blast
  qed
  qed
  show ?thesis
  using c1 c10 c20 c30 c40 by argo
  qed
next
case (exists-d x1 h)
then show ?case
  by auto
qed

```

```

lemma suform-ieqv:
  assumes  $\vdash f \text{ieqv } g$ 
  shows  $\vdash (\text{suform } h \text{ } n \text{ } f) \text{ieqv } (\text{suform } h \text{ } n \text{ } g)$ 
  using assms unfolding valid-def using suform-ieqv-sem
  by fastforce

```

```

lemma rename-nl-pitl-1:
  assumes  $v1 \notin \text{bvars } f$ 
     $v2 \notin \text{bvars } f$ 
     $v1 \neq v2$ 
     $v2 \notin \text{fvars } f$ 
     $i \leq j$ 
     $i \neq \infty$ 
     $j \neq -\infty$ 
  shows  $(\sigma \text{ } i \text{ } j \models \text{Ex } v1. f) = (\sigma \text{ } i \text{ } j \models \text{Ex } v2. \text{suform } f \text{ } v1 \text{ } (\$v2))$ 
proof -
  have 1:  $(\sigma \text{ } i \text{ } j \models \text{Ex } v1. f) = (\sigma \text{ } i \text{ } j \models \text{Ex } v2. \text{suform } f \text{ } v1 \text{ } (\$v2 \text{ iand } \text{lfinite}))$ 
    using assms rename-nl-pitl by blast
  have 2:  $\vdash \$v2 \text{ieqv } \$v2 \text{ iand } \text{lfinite}$ 
    using atom-lfinite-absorb by blast
  have 3:  $\vdash \text{suform } f \text{ } v1 \text{ } (\$v2 \text{ iand } \text{lfinite}) \text{ieqv } \text{suform } f \text{ } v1 \text{ } (\$v2)$ 
    using 2 suform-ieqv-sem by fastforce
  have 4:  $(\sigma \text{ } i \text{ } j \models \text{Ex } v2. \text{suform } f \text{ } v1 \text{ } (\$v2 \text{ iand } \text{lfinite})) =$ 
     $(\sigma \text{ } i \text{ } j \models \text{Ex } v2. \text{suform } f \text{ } v1 \text{ } (\$v2))$ 
    using 3 assms(5) assms(6) assms(7) by auto
  show ?thesis
  using 1 4 by blast
qed

```

```

lemma ExistSubst:
  assumes  $n \notin \text{bvars } f$ 

```

$\bigwedge z . z \in \text{fvars } w \implies z \notin \text{bvars } f$
 $\text{state-nl-itl } w$
shows $\vdash \text{suform } f \ n \ (w \text{ iand } \text{lfinite}) \text{ iimp } (Ex \ n. \ f)$
using *assms unfolding valid-def*
using *similar-upd[of - n] subst-suform[of n f w]*
using *semantics-nl-itl.simps(3) semantics-nl-itl.simps(9)* **by** *blast*

lemma *ForallSubst*:
assumes $n \notin \text{bvars } f$
 $\bigwedge z . z \in \text{fvars } w \implies z \notin \text{bvars } f$
 $\text{state-nl-itl } w$
shows $\vdash (Fa \ n. \ f) \text{ iimp } \text{suform } f \ n \ (w \text{ iand } \text{lfinite})$
using *assms*
using *forall-d-def similar-upd subst-suform* **by** *auto*

lemma *ExistsImp*:
assumes $v1 \notin \text{fvars } f1$
shows $\vdash (Ex \ v1. \ (f1 \text{ iimp } f2)) \text{ ieqv } (f1 \text{ iimp } (Ex \ v1. \ f2))$
unfolding *valid-def* **apply** *simp*
by (*meson assms exist-sigma-exist-value not-fvar-upd similar-commute similar-refl*)

lemma *ExistsSChopRight*:
assumes $v1 \notin \text{fvars } f2$
shows $\vdash (Ex \ v1. \ f1 \text{ schop } f2) \text{ iimp } (Ex \ v1. \ f1) \text{ schop } f2$
using *assms*
unfolding *valid-def* **apply** *auto*
using *not-fvar-upd* **by** *blast*

lemma *ExistsSChopLeft*:
assumes $v1 \notin \text{fvars } f1$
shows $\vdash (Ex \ v1. \ f1 \text{ schop } f2) \text{ iimp } f1 \text{ schop } (Ex \ v1. \ f2)$
using *assms*
unfolding *valid-def* **apply** *auto*
using *not-fvar-upd* **by** *blast*

lemma *ForallGen*:
assumes $\vdash f$
shows $\vdash Fa \ v. \ f$
using *assms*
by (*simp add: forall-d-def*)

lemma *ExistsI*:
assumes $i \leq j$
 $i \neq \infty$
 $j \neq -\infty$
 $(\sigma \ i \ j \models f)$
shows $(\sigma \ i \ j \models Ex \ v. \ f)$
using *assms* **apply** *simp*
using *exist-sigma-exist-value similar-refl* **by** *blast*

lemma *init-state-nl-itl*:
assumes *state-nl-itl w*
shows $\vdash \text{init } w \text{ ieqv } \text{lfinite } i \text{ and } w$
using *assms*
unfolding *valid-def*
by (*simp add: init-defs state-nl-itl-defs*)

lemma *exists-elim-state-nl-itl*:
assumes *state-nl-itl w*
 $n \notin \text{bvars } w$
shows $\vdash (\text{Ex } n. w \text{ iand } \text{lfinite}) \text{ ieqv}$
 $(\text{suform } (w \text{ iand } \text{lfinite}) \ n \ (\text{itrue } i \text{ and } \text{lfinite})) \text{ ior}$
 $(\text{suform } (w \text{ iand } \text{lfinite}) \ n \ (\text{ifalse } i \text{ and } \text{lfinite}))$

proof –
have 1: *state-nl-itl itrue*
by (*simp add: inot-d-def itrue-d-def*)
have 2: $(\bigwedge z. z \in \text{fvars } \text{itrue} \implies z \notin \text{bvars } w)$
by (*simp add: inot-d-def itrue-d-def*)
have 3: $\bigwedge \sigma \ i \ j. \ i \leq j \wedge i \neq \infty \wedge j \neq -\infty \implies$
 $(\sigma \ i \ j \models (\text{suform } (w \text{ iand } \text{lfinite}) \ n \ (\text{itrue } i \text{ and } \text{lfinite}))) =$
 $(i \neq -\infty \wedge ((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \sigma \ (\text{eint } x) \ (\text{eint } x) \models \text{itrue})) \ i \ j \models w))$
using *subst-suform[of n w itrue] 1 2 assms suform-lfinite-export[of w n itrue iand lfinite]*
suform-lfinite-export[of w n ifalse iand lfinite] **by** *simp*
have 4: *state-nl-itl ifalse*
by *simp*
have 5: $(\bigwedge z. z \in \text{fvars } \text{ifalse} \implies z \notin \text{bvars } w)$
by *simp*
have 6: $\bigwedge \sigma \ i \ j. \ i \leq j \wedge i \neq \infty \wedge j \neq -\infty \implies$
 $(\sigma \ i \ j \models (\text{suform } (w \text{ iand } \text{lfinite}) \ n \ (\text{ifalse } i \text{ and } \text{lfinite}))) =$
 $(i \neq -\infty \wedge ((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \sigma \ (\text{eint } x) \ (\text{eint } x) \models \text{ifalse})) \ i \ j \models w))$
using *subst-suform[of n w ifalse] 4 5 assms*
suform-lfinite-export[of w n itrue iand lfinite]
suform-lfinite-export[of w n ifalse iand lfinite]
by *simp*
have 7: $\bigwedge \sigma \ i \ j. \ i \leq j \wedge i \neq \infty \wedge j \neq -\infty \implies$
 $(\sigma \ i \ j \models (\text{Ex } n. w \text{ iand } \text{lfinite})) =$
 $(\exists l. i \neq -\infty \wedge ((\text{upd } \sigma \ n \ l) \ i \ i \models w))$
using *assms(1) lfinite-defs state-nl-itl-defs* **by** *auto*
have 8: $\bigwedge \sigma \ i \ j. \ i \leq j \wedge i \neq \infty \wedge j \neq -\infty \implies$
 $(i \neq -\infty \wedge ((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \sigma \ (\text{eint } x) \ (\text{eint } x) \models \text{itrue})) \ i \ j \models w)) =$
 $(i \neq -\infty \wedge ((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \text{True})) \ i \ i \models w))$
using *state-nl-itl-defs[of w] assms* **by** *simp*
have 9: $\bigwedge \sigma \ i \ j. \ i \leq j \wedge i \neq \infty \wedge j \neq -\infty \implies$
 $(i \neq -\infty \wedge ((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \sigma \ (\text{eint } x) \ (\text{eint } x) \models \text{ifalse})) \ i \ j \models w)) =$
 $(i \neq -\infty \wedge ((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \text{False})) \ i \ i \models w))$
using *state-nl-itl-defs[of w] assms* **by** *simp*
have 10: $\bigwedge \sigma \ i \ j. \ i \leq j \wedge i \neq \infty \wedge j \neq -\infty \implies$

$$\begin{aligned}
& (\exists l. i \neq -\infty \wedge ((\text{upd } \sigma \ n \ l) \ i \ i \models w)) \longleftrightarrow \\
& (i \neq -\infty \wedge ((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \text{True})) \ i \ i \models w) \vee i \neq -\infty \wedge ((\text{upd } \sigma \ n \ (\lambda x::\text{int}. \text{False})) \ i \ i \models w))
\end{aligned}$$

using *assms exists-elim-state-nl-itl-help* **by** *blast*
show *?thesis unfolding valid-def* **using** *lfinite-defs* **apply** *simp*
by (*meson 3 6 8 9 assms(1) assms(2) exists-elim-state-nl-itl-help state-nl-itl-defs*)
qed

lemma *exists-elim-state-nl-itl-1:*

assumes *state-nl-itl w*

n \notin *bvars w*

shows $\vdash (Ex \ n. \ w \ \text{iand} \ lfinite) \ \text{ieqv}$
 $(\text{suform } (w \ \text{iand} \ lfinite) \ n \ (\text{itrue})) \ \text{ior}$
 $(\text{suform } (w \ \text{iand} \ lfinite) \ n \ (\text{ifalse}))$

proof –

have 1: $(\bigwedge z::\text{nat}. \ z \in \text{fvars } \text{itrue} \implies z \notin \text{bvars } w)$

by (*simp add: inot-d-def itrue-d-def*)

have 2: *state-nl-itl itrue*

by (*simp add: inot-d-def itrue-d-def*)

have 3: $\vdash \text{suform } (w \ \text{iand} \ lfinite) \ n \ (\text{itrue} \ \text{iand} \ lfinite) \ \text{ieqv}$
 $\text{suform } (w \ \text{iand} \ lfinite) \ n \ (\text{itrue})$

using *subst-suform-state[of n w itrue] assms*

by (*simp add: 1 2*)

have 4: $\vdash \text{suform } (w \ \text{iand} \ lfinite) \ n \ (\text{ifalse} \ \text{iand} \ lfinite) \ \text{ieqv}$
 $\text{suform } (w \ \text{iand} \ lfinite) \ n \ (\text{ifalse})$

using *subst-suform-state[of n w ifalse] assms* **by** (*simp*)

have 5: $\vdash (\text{suform } (w \ \text{iand} \ lfinite) \ n \ (\text{itrue} \ \text{iand} \ lfinite)) \ \text{ior}$
 $(\text{suform } (w \ \text{iand} \ lfinite) \ n \ (\text{ifalse} \ \text{iand} \ lfinite))$
 ieqv

$(\text{suform } (w \ \text{iand} \ lfinite) \ n \ (\text{itrue})) \ \text{ior}$

$(\text{suform } (w \ \text{iand} \ lfinite) \ n \ (\text{ifalse}))$

using 3 4 **by** *auto*

show *?thesis*

using 5 *assms exists-elim-state-nl-itl* **by** *force*

qed

end

Bibliography

- [1] A. Armstrong, G. Struth, and T. Weber. Kleene algebra. *Archive of Formal Proofs*, 2013. http://isa-afp.org/entries/Kleene_Algebra.shtml, Formal proof development.
- [2] H. Bowman and S. J. Thompson. A Decision Procedure and Complete Axiomatization of Finite Interval Temporal Logic with Projection. *Journal of Logic and Computation*, 13(2):195–239, Apr. 2003.
- [3] G. Grov and S. Merz. A Definitional Encoding of TLA* in Isabelle/HOL. *Archive of Formal Proofs*, 2011. <https://www.isa-afp.org/entries/TLA.html>, Formal proof development.
- [4] D. P. Guelev and B. Moszkowski. A separation theorem for discrete-time interval temporal logic. *Journal of Applied Non-Classical Logics*, 32(1):28–54, 2022.
- [5] A. Lochbihler. Coinductive. *Archive of Formal Proofs*, feb 2010. <https://www.isa-afp.org/entries/Coinductive.html>, Formal proof development.
- [6] S. Merz. An Encoding of TLA in Isabelle. <http://www.pst.informatik.uni-muenchen.de/~merz/isabelle/>. Part of the Isabelle distribution., 1998.
- [7] B. Moszkowski. Compositional reasoning about projected and infinite time. In *Proceedings of the First IEEE Int'l Conf. on Engineering of Complex Computer Systems (ICECCS'95)*, pages 238–245. IEEE Computer Society Press, 1995. [Download pdf](#).
- [8] B. Moszkowski. A Hierarchical Completeness Proof for Propositional Interval Temporal Logic with Finite Time. *Journal of Applied Non-Classical Logics*, 14(1–2):55–104, 2004.
- [9] B. Moszkowski. Compositional reasoning using intervals and time reversal. *Annals of Mathematics and Artificial Intelligence*, 71(1–3):175–250, 2014.
- [10] B. Moszkowski and D. Guelev. An application of temporal projection to interleaving concurrency. *Formal Aspects of Computing*, 29(4):705–750, July 2017.
- [11] B. C. Moszkowski. Imperative reasoning in interval temporal logic. Technical report, University of Newcastle upon Tyne, 1996.
- [12] B. C. Moszkowski. A Complete Axiom System for Propositional Interval Temporal Logic with Infinite Time. *Logical Methods in Computer Science Journal*, 8(3), 2012.
- [13] B. C. Moszkowski and D. P. Guelev. An Application of Temporal Projection to Interleaving Concurrency. In *Dependable Software Engineering: Theories, Tools, and Applications - First International Symposium, SETTA 2015, Nanjing, China, November 4-6, 2015, Proceedings*, pages 153–167, 2015.

- [14] Y. Venema. A modal logic for chopping intervals. *Journal of Logic and Computation*, 1(4):453–476, 09 1991.
- [15] J. S. Warford, D. Vega, and S. M. Staley. A Calculational Deductive System for Linear Temporal Logic. <https://www.cslab.pepperdine.edu/warford/Papers/Vega-Paper.pdf>, 2019.