

An encoding of Interval Temporal Logic in Isabelle/HOL

Antonio Cau

Ben Moszkowski

David Smallwood

September 13, 2023

Abstract

These Isabelle theories introduce the semantics and syntax of Finite and Infinite Interval Temporal Logic (ITL). The ITL proof system, as introduced in [6, 9], has been encoded and its soundness has been checked. The encoding is shallow using the Intensional Logic technique of [4]. An extensive library of Finite and Infinite ITL theorems, taken from [8], has been checked. Non-empty coinductive lists are used to denote intervals.

Contents

1	Extra operations on LLists	6
1.1	Auxiliary lemmas	7
1.2	lbutlast	8
1.3	lfuse	14
1.4	ridx and lidx	21
1.5	lsub	35
1.6	llastlfirst	41
1.7	lfusecat	41
1.8	kfilter	75
1.9	Transfer rules	109
2	Coinductive non-empty lists and their operations	110
2.1	Type definition	110
2.2	Code generator setup	111
2.3	Connection with 'a llist	113
2.4	The nlast element <i>nlast</i>	126
2.5	<i>nset</i>	127
2.6	<i>nmap</i>	130
2.7	Appending two nonempty lazy lists <i>nappend</i>	130
2.8	Appending a nonempty lazy list to a lazy list <i>lappendn</i>	132
2.9	The length of a nonempty lazy list <i>nlength</i>	133
2.10	The nth element of a nonempty lazy list <i>nnth</i>	133
2.11	<i>ntake</i>	135
2.12	<i>ntaken</i>	139
2.13	Concatenating a nonempty lazy list of nonempty lazy lists <i>nconcat</i>	143
2.14	<i>nellist-all2</i>	157
2.15	From a nonempty lazy list to a lazy list <i>llist-of-nellist</i>	163

2.16	<i>ndropn</i>	163
2.17	<i>nzip</i>	170
2.18	<i>niterates</i>	175
2.19	Filtering non-empty lazy lists <i>nfilter</i>	177
2.20	Setup for Lifting/Transfer	187
2.20.1	Relator and predicate properties	187
2.20.2	Transfer rules for the Transfer package	187
3	Extra operations on nellists	191
3.1	<i>ndropns</i>	191
3.2	Definitions	194
3.3	<i>nbutlast</i>	195
3.4	<i>nsubn</i>	198
3.5	<i>nfuse</i>	202
3.6	<i>nridx</i> and <i>nidx</i>	208
3.7	<i>nlastnfirst</i>	223
3.8	<i>nfusecat</i>	225
3.9	<i>nkfilter</i>	240
4	Finite and Infinite ITL Semantics	280
4.1	Types of Formulas	281
4.2	Semantics of ITL	281
4.3	Abbreviations	282
4.4	Properties of Operators	289
4.5	Soundness Axioms	297
4.5.1	ChopAssoc	297
4.5.2	OrChopImp	301
4.5.3	ChopOrImp	301
4.5.4	EmptyChop	301
4.5.5	ChopEmpty	302
4.5.6	StateImpBi	302
4.5.7	NextImpNotNextNot	302
4.5.8	BiBoxChopImpChop	302
4.5.9	BoxInduct	302
4.6	Quantification over State (Flexible) Variables	303
4.7	Temporal Quantifiers	303
5	Finite and Infinite ITL: Axioms and Rules	304
5.1	Rules	304
5.2	Axioms	304
5.3	Additional Lemmas	305
5.4	Quantification	306
5.5	Lemmas about <i>current-val</i>	306
5.6	Lemmas about <i>next-val</i>	307
5.7	Lemmas about <i>fin-val</i>	307
5.8	Lemmas about <i>penult-val</i>	308
5.9	Basic temporal variables properties	308

6	Finite and Infinite ITL theorems using Weak Chop	309
6.1	Propositional reasoning	309
6.2	State formulas	311
6.3	finite and inf properties	311
6.4	Basic Theorems	314
6.5	Further Properties Di and Bi	328
6.6	Properties of Da and Ba	334
6.7	Properties of Fin	342
6.8	Properties of Halt	369
6.9	Properties of Groups of chops	374
7	Finite and Infinite ITL theorems using strong chop	374
7.1	Strong Chop axioms	375
7.2	ITL operators in terms of SChop	376
7.3	Basic Theorems	377
7.4	Further Properties Df and Bf	383
7.5	Properties of SDa and SBa	391
7.6	Properties of SFin	400
7.7	Properties of Halt	427
7.8	Properties of Groups of strong chops	431
8	Finite and Infinite ITL theorems using Weak Chop	431
8.1	Definitions	432
8.2	Semantic lemmas	435
8.3	Helper lemmas	446
8.4	Properties of Chopstar and Chopplus	446
8.5	Kleene Algebra	453
8.6	WPowerstar	495
8.7	Len	510
8.8	Properties of While	513
9	Omega and variants	525
9.1	Definitions	525
9.1.1	Omega	525
9.1.2	Alternative definition for Omega	527
9.1.3	Weak Omega	539
9.2	Omega algebra	542
9.3	Properties of Omega	564
10	Projection operator	571
10.1	Definitions	571
10.2	Lemmas	574
10.2.1	Misc	574
10.2.2	pfilt Lemmas	575
10.2.3	powerinterval lemmas	585
10.2.4	cppl lemmas	596
10.2.5	lcpl lemmas	601
10.2.6	lsum lemmas	620

10.3	Soundness of Projection Axioms	658
10.3.1	PJ1	659
10.3.2	PJ2	659
10.3.3	PJ3	660
10.3.4	PJ4	661
10.3.5	PJ5	671
10.3.6	PJ6	671
10.3.7	PJ7	678
10.3.8	PJ8	701
10.3.9	PJ9	702
10.4	Axioms	702
10.5	Theorems	705
10.5.1	Projection	705
10.5.2	fdp, fbp odp and obp	713
11	Infinite and Finite Interval Temporal Algebra	726
11.1	Definition of Set of intervals and Operations on them	726
11.2	Simplification Lemmas	729
11.3	Algebraic Laws	733
11.3.1	Commutative Additive Monoid	733
11.3.2	Boolean algebra	733
11.3.3	multiplicative monoid	734
11.3.4	Subsumption order	737
11.3.5	Helper lemmas	737
11.3.6	Kleene Algebra	748
11.3.7	Omega Algebra	750
11.3.8	ITL specific Laws	751
11.4	Derived Laws	752
11.4.1	Helper Lemmas	752
11.4.2	ITL Axioms derived	759
11.5	Extra Laws	760
11.5.1	Boolean Laws	760
11.5.2	Chop	762
11.5.3	Next	763
11.5.4	SInit	767
11.5.5	SStar	771
11.5.6	Box and Diamond	773
11.5.7	Finite and Infinite	779
11.5.8	Omega	787
11.6	Link between Set of Intervals and ITL	795
12	Until operator for Finite and Infinite Intervals	801
12.1	Definitions	801
12.2	Axioms	802
12.2.1	NextUntil	802
12.2.2	UntilNextUntil	804
12.2.3	NotUntilFalse	806

12.2.4	UntilOrDist	806
12.2.5	UntilRightDistOr	806
12.2.6	UntilLeftDistAnd	806
12.2.7	UntilAndDist	806
12.2.8	untilNotImp	807
12.2.9	UntilUntil	807
12.2.10	UntilRightor	808
12.2.11	UntilRightAnd	808
12.2.12	DiamondEqvTrueUntil	809
12.2.13	TrueUntillImpNotUntil	809
12.2.14	WaitNotDistUntil	810
12.2.15	UntillInduction	811
12.3	Theorems	813
13	Pi operator for infinite and finite ITL	839
13.1	Definitions	839
13.2	Semantic Lemmas	840
13.3	Soundness of Axioms	846
13.3.1	PiK	846
13.3.2	PiDc	846
13.3.3	PiN	847
13.3.4	PiTrueEqvDiamond	847
13.3.5	PiOr	847
13.3.6	UPiFalseEqvBoxNot:	847
13.3.7	BoxEqvImpPiEqv	847
13.3.8	PiDiamondImpDiamond	848
13.3.9	PiAssoc	848
13.3.10	PiNotEqvDiamondAndNotPi	850
13.3.11	PiSChopDist	851
13.3.12	PiProp	854
13.3.13	PiNext	856
13.3.14	PiUntil	858
13.3.15	PiChopstar	870
13.3.16	TruePiEqv	877
13.3.17	BoxImpEqvPi	877
13.3.18	PiEqvDiamondUPi	877
13.3.19	PiEqvUntilPi	877
13.3.20	UPiEqvBoxOrPi	878
13.4	Theorems	878
13.5	SChopstar and Pi	892
13.6	Omega and Pi	903
14	First Order Finite and Infinite ITL theorems	911

15 The First Occurrence Operator in finite and infinite ITL	917
15.1 Definitions	917
15.1.1 Definitions Strict Initial and Final	917
15.1.2 Definition First and Last Operators	918
15.2 First and Time Reversal	919
15.3 Semantic Theorems	919
15.4 Bi induction	919
15.4.1 Semantics First Operator	920
15.4.2 Various Semantic Lemmas	922
15.5 Theorems	924
15.5.1 Fixed length intervals	924
15.5.2 Additional ITL theorems	931
15.5.3 Strict initial intervals	950
15.5.4 First occurrence	965
16 Monitors	994
16.1 Syntax	994
16.2 Derived Monitors	995
16.3 Monitor Laws	998
17 Monitor Example	1020

1 Extra operations on LLists

the operations `kfilter`, `lleast`, `lbutlast`, `lidx`, `ridx`, `lfirst`, `lfuse`, `lsub`, `lsubc`, `llastlfirst`, `lltl`, `llbutlast`, `is_lfirst` and `lfusecat` 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 xs'))))

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

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  $\implies$   $\forall$  i. enat i < llength ys  $\longrightarrow$  lnth xs i = lnth ys i  $\implies$  xs = ys
proof (coinduction arbitrary: xs ys)
case (Eq-llist xsa ysa)
then show ?case
  proof -
    assume a: llength xsa = llength ysa
    assume b:  $\forall$  i. enat i < llength ysa  $\longrightarrow$  lnth xsa i = lnth ysa i
    have 1: lnull xsa = lnull ysa
      using a by auto
    have 2:  $\neg$  lnull xsa  $\longrightarrow$ 
       $\neg$  lnull ysa  $\longrightarrow$ 
      lhd xsa = lhd ysa
    using b lhd-conv-lnth zero-enat-def by force
    have 3:  $\neg$  lnull xsa  $\longrightarrow$ 
       $\neg$  lnull ysa  $\longrightarrow$ 
      ( $\exists$  xs. ltl xsa = xs  $\wedge$  ltl ysa = ys  $\wedge$  llength xs = llength ys  $\wedge$ 
        ( $\forall$  i. enat i < llength ys  $\longrightarrow$  lnth xs i = 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  $\in$  lset xs. P x)  $\longleftrightarrow$  ( $\exists$  i < llength xs. P (lnth xs i))
by (metis in-lset-conv-lnth)

```

lemma exist-llength-gr-zero:

```

assumes ( $\exists$  x  $\in$  lset xs. P x)
shows 0 < llength xs
by (metis assms gr-implies-not-zero gr-zeroI in-lset-conv-lnth)

```

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
      qed
  qed

```



```

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

lemma lbutlast-ltl:
   $lbutlast\ (ltl\ xs) = ltl\ (lbutlast\ xs)$ 
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:  $\neg \textit{lnull (lbutlast (ltl xsa))} \longrightarrow$ 
       $\neg \textit{lnull (ltl (lbutlast xsa))} \longrightarrow$ 
       $\textit{lhs (lbutlast (ltl xsa))} = \textit{lhs (ltl (lbutlast xsa))}$ 
      by (metis Eq-llist lappend-inf lbutlast-snoc lfinite-ltl)
    have 3:  $\neg \textit{lnull (lbutlast (ltl xsa))} \longrightarrow$ 
       $\neg \textit{lnull (ltl (lbutlast xsa))} \longrightarrow$ 
       $(\exists \textit{xs. ltl (lbutlast (ltl xsa))} = \textit{lbutlast (ltl xs)} \wedge$ 
       $\textit{ltl (ltl (lbutlast xsa))} = \textit{ltl (lbutlast xs)} \wedge \neg \textit{lfinite xs})$ 
      by (metis Eq-llist lappend-inf lbutlast-snoc lfinite-ltl)
    show ?thesis using 1 2 3 by auto
    qed
  qed
qed

```

```

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

```

```

lemma lbutlast-lfinite:
   $\textit{lfinite (lbutlast xs)} \longleftrightarrow \textit{lfinite xs}$ 
proof
show  $\textit{lfinite (lbutlast xs)} \implies \textit{lfinite xs}$ 
  proof (induct zs≡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-lnull-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
     $\neg \text{lnull } xs$ 
  shows  $(\text{lappend } (\text{lbutlast } xs) (\text{LCons } (\text{llast } xs) \text{ LNil})) = xs$ 
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  $\neg \text{lfinite } xs$ 
     $\neg \text{lnull } xs$ 
  shows  $(\text{lappend } (\text{lbutlast } xs) (\text{LCons } (\text{llast } xs) \text{ LNil})) = xs$ 
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  $\neg \text{lnull } xs \implies (\text{lappend } (\text{lbutlast } xs) (\text{LCons } (\text{llast } xs) \text{ LNil})) = xs$ 
using lappend-lbutlast-llast-id-lfinite lappend-lbutlast-llast-id-not-lfinite by blast

```

```

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

```

```

lemma lbutlast-eq-LCons-conv:
   $\text{lbutlast } xs = (\text{LCons } x \text{ ys}) \iff xs = (\text{LCons } x (\text{lappend } ys (\text{LCons } (\text{llast } xs) \text{ LNil})))$ 
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*:
 $x \in lset\ (lbutlast\ xs) \vee (lfinite\ xs \wedge x \in lset(lbutlast\ xs)) \implies$
 $x \in lset\ (lbutlast\ (lappend\ xs\ ys))$
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

```

1.3 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:
   $\neg \text{lnull } xs \longleftrightarrow 1 \leq \text{llength } xs$ 
by (metis gr-zeroI ileI1 llength-eq-0 not-one-le-zero one-eSuc)

lemma lfuse-llength-atmost-one:
  shows llength (lfuse xs ys)  $\leq 1 \longleftrightarrow \text{llength } xs \leq 1 \wedge \text{llength } ys \leq 1$ 
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  $1 < \text{llength } xs$ 
            $1 < \text{llength } ys$ 
           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 $1 < \text{llength } xs$

$1 < \text{llength } ys$

$\text{llast } xs = \text{lfirst } ys$

$\text{lfinite } ys$

shows $\text{llength } ys < \text{llength } (\text{lfuse } xs \text{ } ys)$

proof –

have 1: $\text{lfuse } xs \text{ } ys = \text{lappend } (\text{lbutlast } xs) \text{ } ys$

by (metis *assms*(2) *assms*(3) *lfuse-lbutlast llength-lnull not-less-zero*)

have 2: $\text{llength } (\text{lappend } (\text{lbutlast } xs) \text{ } ys) = \text{epred}(\text{llength } xs) + \text{llength } ys$

by *simp*

have 3: $0 < \text{epred}(\text{llength } xs)$

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

have 4: $\text{llength } ys < \text{epred}(\text{llength } xs) + \text{llength } ys$

using 3 *assms*(4) *lfinite-llength-enat* **by** *auto*

show *?thesis*

using 1 2 4 **by** *presburger*

qed

lemma *lfuse-llength-le-a:*

$\text{llength } xs \leq \text{llength } (\text{lfuse } xs \text{ } ys)$

by (*simp add: lfuse-llength*)

lemma *lfuse-llength-le-b:*

assumes $\text{llast } xs = \text{lfirst } ys$

shows $\text{llength } ys \leq \text{llength } (\text{lfuse } xs \text{ } ys)$

by (*simp add: assms lfuse-lbutlast*)

lemma *lfuse-lnth-a:*

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

shows $\text{lnth } (\text{lfuse } xs \text{ } ys) \text{ } i = \text{lnth } xs \text{ } i$

using *assms*

by (*simp add: lfuse-def lnth-lappend1*)

lemma *lfuse-lnth-b:*

assumes $\text{llength } xs \leq (\text{enat } i)$

$(\text{enat } i) < \text{llength } (\text{lfuse } xs \text{ } ys)$

shows $\text{lnth } (\text{lfuse } xs \text{ } ys) \text{ } i = (\text{lnth } ys \text{ } (i - (\text{the-enat}(\text{epred}(\text{llength } xs))))))$

proof (*cases* $\neg \text{lnull } xs \wedge \neg \text{lnull } ys$)

case *True*

then show *?thesis*

proof –

have 1: $\text{lfinite } xs$

using *assms*(1) *lfinite-ldropn lnull-imp-lfinite lnull-ldropn* **by** *blast*

obtain *n* **where** 15: $(\text{enat } n) = \text{llength } xs$ **using** 1 **by** (metis *assms*(1) *enat-ile*)

have 16: $(\text{the-enat}(\text{epred}(\text{llength } xs))) = n - 1$

by (metis 15 *epred-enat the-enat.simps*)

have 17: $0 < n$


```

    by (metis 15 True grOI 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:  $n-1 \leq i$ 
    by (metis 15 17 Suc-pred' assms(1) enat-ord-simps(2) leD le-imp-less-Suc wlog-linorder-le)
  have 4:  $(\text{Suc } (i - n)) = (i - (n-1))$ 
    by (metis 15 17 Suc-diff-eq-diff-pred Suc-diff-le assms(1) enat-ord-simps(1))
  have 5:  $\text{lnth } (\text{ltl } ys) (i - \text{the-enat } (\text{llength } xs)) = (\text{lnth } ys (i - (\text{the-enat } (\text{epred } (\text{llength } xs))))))$ 
    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  $\text{epred } (\text{llength } xs) \leq (\text{enat } i)$ 
     $(\text{enat } i) < \text{llength } (\text{lfuse } xs \text{ } ys)$ 
     $\text{llast } xs = \text{lfirst } ys$ 
  shows  $\text{lnth } (\text{lfuse } xs \text{ } ys) i =$ 
     $(\text{if } \text{lnull } ys \text{ then } \text{lnth } xs (\text{the-enat } (\text{epred } (\text{llength } xs))))$ 
     $\text{else } \text{lnth } ys (i - (\text{the-enat } (\text{epred } (\text{llength } xs))))$ 
  proof (cases  $\neg \text{lnull } xs \wedge \neg \text{lnull } ys$ )
  case True
  then show ?thesis
    using assms
    by (simp add: leD lfuse-lbutlast lnth-lappend)
  next
  case False
  then show ?thesis
    proof (cases  $\text{lnull } xs$ )
    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
qed

```

lemma *lfirst-lfuse-1*:

```

  shows  $\text{lfirst } (\text{lfuse } xs \text{ } ys) = (\text{if } \neg \text{lnull } xs \text{ then } \text{lfirst } xs \text{ else } \text{lfirst } ys)$ 
  by (simp add: lfirst-def)

```

lemma *llast-lfuse*:
assumes $\neg \text{lnull } xs$
 $\neg \text{lnull } ys$
 $\text{lfinit}e\ xs$
 $\text{lfinit}e\ ys$
 $\text{llast } xs = \text{lfir}st\ ys$
shows $\text{llast}(\text{lfuse } xs\ ys) = \text{llast } ys$
using *assms*
by (*metis lfirst-def lfuse-def lhd-LCons-ltl llast-LCons llast-lappend*)

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

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

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

lemma *llast-lfirst-LNil*:
 $\text{llast } LNil = \text{lfir}st\ LNil$
by (*simp add: lfirst-def lhd-def llast-LNil*)

lemma *ldrop-lappend-either-LNil*:
assumes $\text{lnull } xs \vee \text{lnull } ys$
shows $\text{ldrop } (\text{llength } xs)\ (\text{lappend } xs\ ys) =$
 $(\text{if } \text{lfinit}e\ xs \text{ then } ys \text{ else } LNil)$
proof –
have 1: $\text{lnull } xs \implies ?thesis$
by (*simp add: lappend-lnull1*)
have 2: $\text{lnull } ys \implies ?thesis$
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 $\text{llast } xs = \text{lfir}st\ ys$
shows $\text{ldrop } (\text{if } \neg \text{lnull } xs \wedge \neg \text{lnull } ys \text{ then } \text{epred}(\text{llength } xs) \text{ else } (\text{llength } xs))\ (\text{lfuse } xs\ ys) =$
 $(\text{if } \text{lfinit}e\ xs \text{ then } ys \text{ else } LNil)$

```

proof –
  have 3: lfuse xs ys = (if  $\neg \text{lnull } xs \wedge \neg \text{lnull } ys$ 
    then lappend (lbutlast xs) ys
    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  $\neg \text{lnull } xs$ 
   $\neg \text{lnull } ys$ 
  llast xs = 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 < 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*
 $\text{llist.case-eq-if fun-ord-def lprefix-ltlI monotone-def dest: monotoneD}$)
 $(\text{metis llist.collapse}(1) \text{ lprefix-ltlI ltl-simps}(1))$

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

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

lemma *lfuse-LNil-eq-id*:
 $\text{lfuse LNil} = \text{id}$
by (*simp add: fun-eq-iff llist.case-eq-if*)

lemma *mcont2mcont-lfuse2* [*THEN llist.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 $\text{lfuse } xs = (\lambda -. \text{xs})$
by (*simp add: fun-eq-iff lappend-inf lfuse-def*)
thus *?thesis* **by**(*simp add: ccpo.cont-const[OF llist-ccpo]*)
qed

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

lemma *llist-all2-lfuseI*:

```

assumes 1: llist-all2 P xs ys
and 2:  $\llbracket \text{lfinite } xs; \text{lfinite } ys \rrbracket \implies \text{llist-all2 } P \text{ } xs' \text{ } ys'$ 
shows llist-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  $\neg \text{lfinite } ys$  by(auto dest: llist-all2-lfiniteD)
          with False 1 show ?thesis
            by (simp add: lfuse-inf)
          qed
    qed

```

1.4 ridx and lidx

lemma *ridx-lidx*:

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

unfolding *lidx-def ridx-def*

by *simp*

lemma *ridx-expand-1*:

ridx R 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*:

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*:

ridx R (LCons x xs) \longleftrightarrow

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*:

$lidx (LCons x xs) \longleftrightarrow$
 $lnull xs \vee$
 $(0 < llength xs \wedge$
 $(\forall n. (Suc n) < eSuc (llength xs) \longrightarrow lnth (LCons x xs) n < lnth (LCons x xs) (Suc n)))$
using $ridx-lidx[of (LCons x xs)]$ $ridx-LCons[of (<) x xs]$ **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: $(0 < \text{length } xs \wedge$
 $R \ x \ (\text{lhs } xs) \wedge$
 $(\forall n. 0 \leq (n-1) \wedge (\text{Suc } (n-1)) < (\text{length } xs) \longrightarrow R \ (\text{nth } xs \ (n-1)) \ (\text{nth } xs \ (\text{Suc } (n-1))))$
 \longleftrightarrow
 $(0 < \text{length } xs \wedge$
 $R \ x \ (\text{lhs } xs) \wedge$
 $(\forall n. 0 \leq n \wedge (\text{Suc } n) < (\text{length } xs) \longrightarrow R \ (\text{nth } xs \ n) \ (\text{nth } xs \ (\text{Suc } n)))$
 by (metis diff-Suc-1)
 show ?thesis
 using 2 3 4 5 6 by auto
 qed

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

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

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

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

lemma *lidx-less*:
assumes *lidx* xs

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

lemma *ridx-less-eq*:
assumes *ridx R xs*
 $k \leq j$
 $j < llength\ xs$
transp R
reflp R
shows $R\ (lnth\ xs\ k)\ (lnth\ xs\ j)$
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*
 $k \leq j$
 $j < llength\ xs$
shows $lnth\ xs\ k \leq lnth\ xs\ j$
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*
 $0 < i$
 $i < llength\ xs$
transp R
shows $R\ (lnth\ xs\ 0)\ (lnth\ xs\ i)$
using *assms ridx-less*[of $R\ xs\ 0\ i-1$] **unfolding** *ridx-def* **by** *simp*

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

lemma *ridx-ltake-a*:
assumes *ridx R xs*
 $n \leq llength\ xs$
shows $ridx\ R\ (ltake\ n\ xs)$
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 *lidx (ltake n xs)*
using *assms*
using *ridx-lidx ridx-ltake-a* **by** *blast*

lemma *ridx-lappend-lfinite*:

assumes *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 *assms*
proof (*induction xs arbitrary: ys*)
case *lfinite-LNil*
then show *?case* **by** (*simp add: ridx-expand-1*)
next
case (*lfinite-LConsI xs x*)
then show *?case*
 proof (*cases lnull xs*)
 case *True*
 then show *?thesis*
 by (*metis lappend-code(1) lappend-code(2) llast-singleton llength-LCons llist.collapse(1)*
 one-eSuc order-neq-le-trans ridx-LCons-1 ridx-expand-1 zero-le)
 next
 case *False*
 then show *?thesis*
 by (*simp add: lfinite-LConsI.IH llast-LCons*)
 qed
qed

lemma *lidx-lappend-lfinite*:

assumes *lfinite xs*
shows *lidx (lappend xs 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 *assms* **by** (*metis ridx-lappend-lfinite ridx-lidx*)

lemma *ridx-ldrop*:

assumes *ridx R xs*
 $n \leq \text{length } xs$
shows *ridx R (ldrop n xs)*
proof –
have 1: $xs = \text{lappend } (\text{ltake } n \text{ } xs) (\text{ldrop } n \text{ } xs)$
 by (*simp add: lappend-ltake-ldrop*)
have 2: $\neg \text{lfinite } (\text{ltake } n \text{ } xs) \implies \text{?thesis}$
 by (*simp add: ridx-expand-1*)
have 3: $\text{lfinite } (\text{ltake } n \text{ } xs) \implies \text{ridx } R (\text{lappend } (\text{ltake } n \text{ } xs) (\text{ldrop } n \text{ } xs)) \longleftrightarrow$
 $\text{ridx } R (\text{ltake } n \text{ } xs) \wedge$

```

      ((lnull (ltake n xs) ∨ lnull (ldrop n xs)) ∨ R (llast (ltake n xs)) (lhd (ldrop n xs))) ∧
      ridx R (ldrop n xs)
    using ridx-lappend-lfinite[of (ltake n xs) R (ldrop n xs)] by blast
  have 4: lfinite (ltake n xs)  $\implies$  ?thesis
    using 1 3 assms by metis
  show ?thesis using 2 4 by blast
qed

```

```

lemma lidx-ldrop:
  assumes lidx xs
    n ≤ llength xs
  shows lidx (ldrop n xs)
using assms ridx-ldrop ridx-lidx by blast

```

```

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

```

```

lemma ridx-ltake:
  assumes ridx R (ltake n xs)
    n ≤ llength xs
    k ≤ 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 ≤ llength xs
    k ≤ 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: lhd xsa  $\notin$  lset (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
    (lnth zs 0) = 0
    lfinite ys
    lfinite xs
    ¬ lnull xs
    ¬ lnull zs
    llast ys = cp
shows llast ys = lfirst(lmap (λx. x+cp) zs)
using assms
by (simp add: lfirst-def lnth-0-conv-lhd)

lemma ridx-lfuse-lnth-cp:
assumes ridx R ys
    (lnth ys 0) = 0
    ridx R zs
    (lnth zs 0) = 0
    lfinite ys
    lfinite zs
    lfinite xs
    ¬ lnull xs
    ¬ lnull zs
    ¬ lnull ys
    llast ys = cp
    llast zs = the-enat(epred (llength xs)) - cp
    i < (llength zs)
    cp < llength xs
shows lnth (lfuse ys (lmap (λx. x+cp) zs)) (the-enat(epred(llength ys)) + i) = cp + (lnth zs i)
proof -
have 1: llast ys = lfirst(lmap (λx. x+cp) 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: lfirst(lmap (λx. x+cp) zs) = cp + (lnth zs 0)
  using 1 assms(11) assms(4) by force
have 8: i ≤ epred(llength zs)
  using assms
  by (metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0)
have 81: enat (the-enat (epred (llength ys)) + i) ≤
  enat (the-enat (epred (llength ys)) + (the-enat (epred(llength zs))))
  by (metis 8 add-mono-thms-linordered-semiring(2) assms(6) assms(9) co.enat.exhaust-sel
    enat-ord-simps(1) enat-ord-simps(4) enat-the-enat ile-eSuc iless-Suc-eq lfinite-llength-enat
    llength-eq-0 order-le-less-trans)
have 9: epred(llength zs) < llength zs
  by (metis assms(6) assms(9) co.enat.exhaust-sel ile-eSuc iless-Suc-eq lfinite-llength-enat
    llength-eq-0 order-neq-le-trans)
have 10: epred (llength ys) ≤ enat (the-enat (epred (llength ys)) + (i::nat))
  by (metis assms(10) assms(5) co.enat.exhaust-sel enat-ord-simps(1) enat-the-enat ile-eSuc
    infinity-ileE le-add1 lfinite-llength-enat llength-eq-0)
have 83: enat (the-enat (epred (llength ys)) + (the-enat (epred(llength zs)))) =
  enat (the-enat (epred (llength ys)) + epred(llength zs))
  by (metis 10 9 enat-ord-code(4) enat-the-enat leD order-less-imp-le plus-enat-simps(1))
have 84: enat (the-enat (epred (llength ys)) + epred(llength zs)) =

```

```

    ( (epred (llength ys) + epred(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: epred(llength ys) < llength ys
  by (metis 10 assms(10) co.enat.exhaust-sel enat-ord-code(4) enat-the-enat ile-eSuc ileSS-Suc-eq
      leD llength-eq-0 order-neq-le-trans)
have 86: llength (lfuse ys (lmap (λx::nat. x + cp) zs)) = llength ys + epred(llength zs)
  by (simp add: assms(10) lfuse-llength)
have 87: ( (epred (llength ys) + epred(llength zs))) < llength ys + epred(llength zs)
  by (metis 83 84 85 add commute enat-le-plus-same(2) enat-less-enat-plusI2 enat-the-enat
      infinity-ileE)
have 82: enat (the-enat (epred (llength ys)) + (the-enat (epred(llength zs)))) <
  llength (lfuse ys (lmap (λx::nat. x + cp) zs))
  using 83 84 86 87 by presburger
have 11: enat (the-enat (epred (llength ys)) + i) < llength (lfuse ys (lmap (λx::nat. x + cp) zs))
  using 81 82 order-le-less-trans by blast
have 12: (the-enat (epred (llength ys)) + i - the-enat (epred (llength ys))) = i
  by auto
have 4: lnth (lfuse ys (lmap (λx. x+cp) zs)) (the-enat(epred(llength ys)) + i) =
  (lnth (lmap (λx. x+cp) zs) i)
  by (simp add: 1 10 11 assms(13) assms(9) lfuse-lnth-c)
have 5: (lnth (lmap (λx. x+cp) zs) i) = cp + (lnth zs i)
  by (simp add: assms)
show ?thesis
using 4 5 by presburger
qed

```

lemma *lidx-lfuse-lnth-cp*:

assumes *lidx ys*

(*lnth ys 0*) = 0

lidx zs

(*lnth zs 0*) = 0

lfinite ys

lfinite zs

lfinite xs

¬ *lnull xs*

¬ *lnull zs*

¬ *lnull ys*

llast ys = *cp*

llast zs = *the-enat(epred (llength xs)) - cp*

i < (*llength zs*)

cp < *llength xs*

shows *lnth (lfuse ys (lmap (λx. x+cp) zs)) (the-enat(epred(llength ys)) + i) = cp + (lnth zs i)*

using *assms ridx-lidx ridx-lfuse-lnth-cp* **by** *blast*

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

assumes *ridx R ys*

(*lnth ys 0*) = 0

ridx R zs

(*lnth zs 0*) = 0

lfinite ys

```

  lfinite zs
  lfinite xs
  ¬ lnull xs
  ¬ lnull zs
  ¬ lnull ys
  llast ys = cp
  llast zs = the-enat(epred (llength xs)) - cp
  i < (epred(llength ys)) + (llength zs)
  the-enat(epred(llength ys)) ≤ i
  cp < llength xs
shows lnth (lfuse ys (lmap (λx. 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
  ¬ lnull xs
  ¬ lnull zs
  ¬ lnull ys
  llast ys = cp
  llast zs = the-enat(epred (llength xs)) - cp
  i < (epred(llength ys)) + (llength zs)
  the-enat(epred(llength ys)) ≤ i
  cp < llength xs
shows lnth (lfuse ys (lmap (λx. 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
  ¬ lnull xs

```

```

    ¬ lnull zs
    ¬ lnull ys
    llast ys = cp
    llast zs = the-enat(epred (llength xs)) - cp
    i < (llength zs)
    cp < llength xs
shows llast (lfuse ys (lmap (λx. x+cp) zs)) = (the-enat (epred(llength xs)))
proof -
have 1: llast (lfuse ys (lmap (λx. x+cp) zs)) = llast (lmap (λx. x+cp) zs)
    using assms
by (metis add-cancel-left-left lfinite-lmap lfirst-def llast-lfuse llist.map-disc-iff
      lmap.map-sel(1) lnth-0-conv-lhd)
have 2: llast (lmap (λx. x+cp) zs) = cp + (llast zs)
    by (simp add: assms(6) assms(9) llast-lmap)
have 3: cp + (llast zs) = (the-enat (epred(llength 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-llength-enat llength-eq-0)
show ?thesis using 1 2 3 by auto
qed

```

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

```

assumes lidx ys
    (lnth ys 0) = 0
    lidx zs
    (lnth zs 0) = 0
    lfinite ys
    lfinite zs
    lfinite xs
    ¬ lnull xs
    ¬ lnull zs
    ¬ lnull ys
    llast ys = cp
    llast zs = the-enat(epred (llength xs)) - cp
    i < (llength zs)
    cp < llength xs
shows llast (lfuse ys (lmap (λx. x+cp) zs)) = (the-enat (epred(llength xs)))
using assms ridx-lidx ridx-lfuse-lnth-cp-llast by blast

```

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

```

assumes ridx R ys
    (lnth ys 0) = (0::nat)
    ridx R zs
    (lnth zs 0) = 0
    lfinite ys
    ¬ lfinite zs
    ¬ lfinite xs
    ¬ lnull ys
    llast ys = cp
shows lnth (lfuse ys (lmap (λx. x+cp) zs)) (the-enat(epred(llength ys)) + i) = cp + (lnth zs i)

```

proof –

have 1: $llast\ ys = lfirst(lmap\ (\lambda x. x+cp)\ zs)$
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: $lfirst(lmap\ (\lambda x. x+cp)\ zs) = cp + (lnth\ zs\ 0)$
using 1 assms **by** auto
have 8: $i \leq epred(llength\ zs)$
by (metis assms(6) enat.simps(3) ldrop-eq-LNil lfinite-ldrop lfinite-ltl linorder-le-cases llength-ltl)
have 10: $epred\ (llength\ ys) \leq enat\ (the-enat\ (epred\ (llength\ ys)) + (i::nat))$
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: $enat\ (the-enat\ (epred\ (llength\ ys)) + i) < llength\ (lfuse\ ys\ (lmap\ (\lambda x::nat. x + cp)\ zs))$
using assms
by (metis 1 enat-ile ldrop-lfuse lfinite-conv-llength-enat lfinite-ldrop
 lfinite-lmap not-le-imp-less)
have 12: $(the-enat\ (epred\ (llength\ ys)) + i - the-enat\ (epred\ (llength\ ys))) = i$
by auto
have 4: $lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x+cp)\ zs))\ (the-enat(epred(llength\ ys)) + i) =$
 $(lnth\ (lmap\ (\lambda x. x+cp)\ zs)\ i)$
using assms
by (metis 1 10 11 12 lfuse-lnth-c llist.map-disc-iff lnull-imp-lfinite)
have 5: $(lnth\ (lmap\ (\lambda x. x+cp)\ zs)\ i) = cp + (lnth\ zs\ i)$
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 lid_x-lfuse-lnth-cp-infinite:

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

lemma lid_x-lfuse-lid_x:

assumes lid_x ys
 $lnth\ ys\ 0 = 0$
 lid_x zs
 $lnth\ zs\ 0 = 0$
 lfinite ys
 $\neg lnull\ ys$

$llast\ ys = cp$
 $lfinite\ zs$
 $\neg lnull\ zs$
 $lfinite\ xs$
 $llast\ zs = the-enat(epred(llength\ xs)) - cp$
 $cp < llength\ xs$
 $i < llength\ zs$
 $cp < llength\ xs$
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(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: $lfirst\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs)) = lfirst\ ys$
by (simp add: assms(6) assms(9) lfirst-lfuse-1)
have 3: $llength\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs)) = llength\ ys + epred(llength\ zs)$
by (simp add: assms(13) assms(6) lfuse-llength)
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: $\bigwedge j. epred(llength\ ys) \leq j \wedge j < epred(llength\ ys) + llength\ zs \implies$
 $lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs))\ j =$
 $cp + (lnth\ zs\ (j - the-enat(epred(llength\ ys))))$
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: $\bigwedge j. k1 - 1 \leq j \wedge j < (enat\ (k1 - 1)) + llength\ zs \implies$
 $lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs))\ j =$
 $cp + (lnth\ zs\ (j - (k1 - 1)))$
using 5 44 **by** (metis 42 enat-ord-simps(1))
have 50: $llength(lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs)) = epred(llength\ ys) + llength\ zs$
using assms
by (metis 3 add commute epred-iadd1 llength-eq-0)
have 51: $\bigwedge j. enat\ (Suc\ j) < llength\ ys \implies$
 $(lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs))\ j) <$
 $(lnth\ (lfuse\ ys\ (lmap\ (\lambda x. x + cp)\ zs))\ (Suc\ j))$
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: $\bigwedge j. k1 - 1 \leq j \wedge (Suc\ j) < enat\ (k1 - 1) + llength\ zs \implies$
 $cp + (lnth\ zs\ (j - (k1 - 1))) <$

$cp + (l\text{nth } zs ((\text{Suc } j) - (k1 - 1)))$
using *assms(3) unfolding lidx-def*
by *simp*
 $(\text{metis } \text{Suc-diff-le } \text{add-Suc-right } \text{assms(8)} \text{ enat-ord-simps(2)} \text{ leD } \text{lfinite-llength-enat}$
 $\text{nat-add-left-cancel-le } \text{not-le-imp-less } \text{ordered-cancel-comm-monoid-diff-class.add-diff-inverse}$
 $\text{plus-enat-simps(1)})$
have 6: $\bigwedge j. \text{ enat } (\text{Suc } j) < \text{llength}(\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) \implies$
 $(\text{lnth } (\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) j) <$
 $(\text{lnth } (\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) (\text{Suc } j))$
proof –
fix *j*
assume *a*: $\text{ enat } (\text{Suc } j) < \text{llength}(\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs))$
show $(\text{lnth } (\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) j) < (\text{lnth } (\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) (\text{Suc } j))$
proof –
have 61: $\text{ enat } (\text{Suc } j) < \text{llength } ys \implies$
 $(\text{lnth } (\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) j) < (\text{lnth } (\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) (\text{Suc } j))$
using 51 **by** *blast*
have 62: $k1 - 1 \leq j \wedge (\text{Suc } j) < \text{llength}(\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) \implies$
 $(\text{lnth } (\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) j) < (\text{lnth } (\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) (\text{Suc } j))$
by $(\text{metis } 42 \ 45 \ 50 \ 52 \ \text{Suc-ile-eq } \text{nle-le } \text{not-less-eq-eq } \text{order-less-imp-le})$
show *?thesis*
by $(\text{metis } 41 \ 43 \ 61 \ 62 \ \text{Suc-diff-1 } \text{Suc-mono } a \ \text{enat-ord-simps(2)} \ \text{leI})$
qed
qed
show *?thesis* **unfolding** *lidx-def*
by $(\text{simp add: } 41 \ 43 \ 6 \ \text{assms(13)} \ \text{assms(2)} \ \text{lfuse-lnth-a})$
qed

lemma *lidx-lfuse-lidx-infinite*:

assumes *lidx ys*
 $\text{lnth } ys \ 0 = 0$
 $\text{lidx } zs$
 $\text{lnth } zs \ 0 = 0$
 $\text{lfinite } ys$
 $\neg \text{lnull } ys$
 $\text{llast } ys = cp$
 $\neg \text{lfinite } zs$
 $\neg \text{lfinite } xs$
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 $(\text{metis } \text{assms(4)} \ \text{assms(7)} \ \text{assms(8)} \ \text{lfirst-def } \text{lmap-sel(1)} \ \text{lnth-0-conv-lhd}$
 $\text{lnull-imp-lfinite } \text{plus-nat.add-0})$
have 2: $\text{lfirst } (\text{lfuse } ys (\text{lmap } (\lambda x. x + cp) zs)) = \text{lfirst } ys$
by $(\text{simp add: } \text{assms(6)} \ \text{lfirst-lfuse-1})$
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 $(\text{simp add: } \text{assms(5)} \ \text{lfinite-llength-enat})$
obtain *k1* **where** 41: $\text{llength } ys = (\text{enat } k1)$

```

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 lidz-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 ( $\lambda x::\text{nat. } x + cp$ ) zs) ]
  llength-lmap[of ( $\lambda x::\text{nat. } x + cp$ ) 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
    lidz-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 lidz-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 lidz-def
  by (simp add: 41 43 6 assms lfuse-lnth-a)
qed

```

1.5 lsub

lemma lsub-eq-lsubc:

assumes $k \leq n$
 $n < \text{length } xs$
shows $\text{lsub } k \ n \ xs = \text{lsubc } k \ n \ xs$
using *assms*
unfolding *lsub-def lsubc-def*
by (*auto simp add: min-def*)
 $(\text{metis } \text{co.enat.exhaust-sel } \text{enat-ord-simps}(1) \ \text{iless-Suc-eq } \text{ldrop-lnull } \text{length-eq-0}$
 $\text{order-le-less-trans})$

lemma *lsub-same*:
assumes $(\text{enat } k) < \text{length } xs$
shows $\text{lsub } k \ k \ xs = (\text{LCons } (\text{lnth } xs \ k) \ \text{LNil})$
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*
 $\text{ltake-eSuc-LCons order.order-iff-strict})$

lemma *lsubc-same*:
assumes $(\text{enat } k) < \text{length } xs$
shows $\text{lsubc } k \ k \ xs = (\text{LCons } (\text{lnth } xs \ k) \ \text{LNil})$
using *assms*
 $\text{lsub-eq-lsubc}[of \ k \ k \ xs] \ \text{lsub-same}[of \ k \ xs]$
by *fastforce*

lemma *length-lsub*:
assumes $k \leq n$
 $n < \text{length } xs$
shows $\text{length } (\text{lsub } k \ n \ xs) = (\text{eSuc } (n - k))$
proof –
have 1: $\text{length } (\text{ldrop } (\text{enat } k) \ xs) = (\text{length } xs - \text{enat } k)$
by (*simp add: ldrop-enat*)
have 2: $\min (\text{eSuc } (\text{enat } (n - k))) (\text{length } xs - \text{enat } k) = (\text{eSuc } (n - k))$
by (*metis 1 Suc-diff-le assms(1) assms(2) eSuc-enat enat-length-ldropn ileI1 ldrop-enat min.absorb1*)
have 3: $\text{length } (\text{ltake } (\text{eSuc } (\text{enat } (n - k))) (\text{ldrop } (\text{enat } k) \ xs)) = (\text{eSuc } (n - k))$
by (*simp add: 1 2*)
show *?thesis*
by (*metis 3 idiff-enat-enat lsub-def*)
qed

lemma *length-lsubc*:
assumes $k \leq n$
 $n < \text{length } xs$
shows $\text{length } (\text{lsubc } k \ n \ xs) = (\text{eSuc } (n - k))$
using *assms lsub-eq-lsubc[of k n xs] length-lsub[of k n xs]* **by** *presburger*

lemma *length-lsub-a*:
shows $\text{length } (\text{lsub } k \ n \ xs) =$
 $\min (\text{eSuc } (n - k))$
 $(\text{if } k = \infty \text{ then } 0::\text{enat} \text{ else } \text{length } xs - k)$
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 *llength* (*lsubc* k n *xs*) =
 \min (*eSuc* ($n - k$))
 (if $\min k$ (*epred* (*llength* *xs*)) = ∞
 then $0::\text{enat}$
 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 $k \leq n$
 $n < \text{length } xs$
shows $\neg \text{lnull}$ (*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 $k < n$
 $n < \text{length } xs$
shows $1 < \text{length}$ (*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 $k \leq n$
 $n < \text{length } xs$
shows *lfinite* (*lsub* k n *xs*)

using *assms*

by (*simp add: eSuc-enat llength-eq-enat-lfiniteD llength-lsub*)

lemma *lnth-lsub*:

assumes $n < \text{length } xs$
 $k+j \leq n$
shows *lnth* (*lsub* k n *xs*) j = (*lnth* *xs* ($k+j$))

proof –

have 1: *enat* ($j::\text{nat}$) < *eSuc* (*enat* (($n::\text{nat}$) - ($k::\text{nat}$)))

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 $n < \text{length } xs$

$m \leq n$

$k \leq m$

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 $n < \text{length } xs$

$m + k \leq n$

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 $n < \text{length } xs$

$m + k \leq n$

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 $n < \text{length } xs$

$k \leq n$

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: $k \neq n \implies (ePred \ (eSuc \ (n - k))) = (eSuc \ (\text{enat } (n - (1::nat) - k)))$

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*:
 $ltl\ xs = ldrop\ 1\ xs$
by (*metis ldrop-0 ldrop-ltl one-eSuc*)

lemma *lappend-lsub-ltl-lsub*:

assumes $k \leq n$
 $n \leq m$
 $m < llength\ xs$
shows $lappend\ (lsub\ k\ n\ xs)\ (ltl\ (lsub\ n\ m\ xs)) = lsub\ k\ m\ xs$
proof –
have 0: $ltl\ (lsub\ n\ m\ xs) = ldrop\ 1\ (lsub\ n\ m\ xs)$
by (*metis ldrop-0 ldrop-ltl one-eSuc*)
have 1: $(lsub\ k\ n\ xs) = (ltake\ (eSuc\ (n - k))\ (ldrop\ k\ xs))$
unfolding *lsub-def* **by** *simp*
have 2: $(lsub\ n\ m\ xs) = (ltake\ (eSuc\ (m - n))\ (ldrop\ n\ xs))$
unfolding *lsub-def* **by** *simp*
have 3: $lsub\ k\ m\ xs = (ltake\ (eSuc\ (m - k))\ (ldrop\ k\ xs))$
unfolding *lsub-def* **by** *simp*
have 8: $ltake\ (eSuc\ (enat\ ((n::nat) - (k::nat))) + enat\ ((m::nat) - n))\ (ldrop\ (enat\ k)\ (xs::'a\ llist)) =$
 $(ltake\ (eSuc\ (m - k))\ (ldrop\ k\ xs))$
by (*simp add: assms(1) assms(2) eSuc-enat*)
have 9: $(ltake\ (enat\ (m - n))\ (ldrop\ (eSuc\ (enat\ (n - k)))\ (ldrop\ (enat\ k)\ xs))) =$
 $ltl(ltake\ (eSuc\ (m - n))\ (ldrop\ n\ xs))$
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: $ltake\ (eSuc\ (enat\ ((n::nat) - (k::nat))) + enat\ ((m::nat) - n))\ (ldrop\ (enat\ k)\ (xs::'a\ llist)) =$
 $lappend\ (ltake\ (eSuc\ (enat\ (n - k)))\ (ldrop\ (enat\ k)\ xs))$
 $(ltake\ (enat\ (m - n))\ (ldrop\ (eSuc\ (enat\ (n - k)))\ (ldrop\ (enat\ k)\ xs)))$
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
 $k \leq n$
 $n \leq m$
 $m < llength\ xs$
shows $lfuse\ (lsub\ k\ n\ xs)\ (lsub\ n\ m\ xs) = lsub\ k\ m\ xs$
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 *lnull xs*
 $k \leq n$
 $n < \text{llength } xs$
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 *lnull xs*
 $k \leq n$
 $n < \text{llength } xs$
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*
lfinite ls
lfinite xs
 \neg *lnull xs*
 $\text{llast } ls = \text{epred}(\text{llength } xs)$
 $(\text{Suc } i) < (\text{llength } ls)$
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: $\text{llast } ls = (\text{lnth } ls \ (\text{the-enat}(\text{epred } (\text{llength } ls))))$
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: $1 < \text{llength } ls$
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: $\text{enat } (\text{lnth } ls \ (\text{Suc } i)) < \text{llength } xs$
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 ile-less-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

1.6 llastfirst

lemma *llastfirst-ridx*:
 $llastfirst\ xss = ridx\ (\lambda\ a\ b.\ llast\ a = lfirst\ b)\ xss$
by (*simp* *add*: *llastfirst-def* *ridx-def*)

lemma *llastfirst-LNil*[*simp*]:
 $llastfirst\ LNil$
unfolding *llastfirst-def* **by** *simp*

lemma *llastfirst-LOne*[*simp*]:
 $llastfirst\ (LCons\ xs\ LNil)$
unfolding *llastfirst-def* **by** (*simp* *add*: *zero-enat-def*)

lemma *llastfirst-LCons*[*simp*]:
assumes $\neg lnull\ xss$
shows $llastfirst\ (LCons\ xs\ xss) \longleftrightarrow llast\ xs = lfirst\ (lfirst\ xss) \wedge llastfirst\ xss$
using *assms* **unfolding** *llastfirst-def*
by *auto*
(*metis* *One-nat-def* *lfirst-def* *lhd-LCons-ltl* *lnth-LCons'* *not-llnull-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 *llastfirst-lappend-lfinite*:
assumes $lfinite\ xss$
shows $llastfirst\ (lappend\ xss\ yss) \longleftrightarrow$
 $(llastfirst\ xss \wedge llastfirst\ yss \wedge$
 $(if\ lnull\ xss \vee lnull\ yss\ then\ True\ else\ llast(llast\ xss) = lfirst(lfirst\ yss)))$
using *assms*
by (*metis* (*mono-tags*, *lifting*) *lfirst-def* *llastfirst-ridx* *ridx-lappend-lfinite*)

1.7 lfusecat

context *notes* [*function-internals*]
begin

partial-function (*llist*) *lfusecat* :: 'a *llist* *llist* \Rightarrow 'a *llist*
where $lfusecat\ xss = (case\ xss\ of\ LNil \Rightarrow LNil \mid LCons\ xs\ xss' \Rightarrow lfuse\ xs\ (lfusecat\ xss'))$

end

lemma *lfusecat-simps* [*simp*, *code*]:
shows *lfusecat-LNil*: $lfusecat\ LNil = LNil$

and *lfusecat-LCons*: $\text{lfusecat } (LCons \ xs \ xss) = \text{lfuse } xs \ (\text{lfusecat } xss)$
by (*simp-all add*: *lfusecat.simps*)

declare *lfusecat.mono*[*cont-intro*]

lemma *mono2mono-lfusecat*[*THEN llist.mono2mono, cont-intro, simp*]:
shows *monotone-lfusecat*: $\text{monotone } (\text{lprefix}) \ (\text{lprefix}) \ \text{lfusecat}$
by(*rule llist.fixp-preserves-mono1*[*OF lfusecat.mono lfusecat-def*]) *simp*

lemma *mcont2mcont-lfusecat*[*THEN llist.mcont2mcont, cont-intro, simp*]:
shows *mcont-lfusecat*: $\text{mcont } lSup \ (\text{lprefix}) \ lSup \ (\text{lprefix}) \ \text{lfusecat}$
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*:
 $\text{llist-all2 } (\text{llist-all2 } A) \ xss \ yss$
 $\implies \text{llist-all2 } A \ (\text{lfusecat } xss) \ (\text{lfusecat } yss)$
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*:
 $\neg \text{lnull } xss \wedge (\forall \ xs \in \text{lset } xss. \neg \text{lnull } xs) \longleftrightarrow \text{lset } xss \neq \{\} \wedge LNil \notin \text{lset } xss$
using *llist.collapse(1) llist.disc(1) lset-eq-empty* **by** *blast*

lemma *not-lnull-lset-conv-b*:
 $\neg \text{lnull } xss \wedge (\forall \ xs \in \text{lset } xss. \neg \text{lnull } xs) \longleftrightarrow \text{lset } xss \neq \{\} \wedge \text{lset } xss \cap \{LNil\} = \{\}$
using *llist.collapse(1)* **by** *force*

lemma *lfusecat-eq-LNil*:
 $\text{lfusecat } xss = LNil \longleftrightarrow \text{lset } xss \subseteq \{LNil\}$
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 ltl-def
by (auto simp add: ltl-ldrop-one)

```

```

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

```

```

lemma lset-lltl-llength-var2:
  ( $\forall xs \in \text{lset } xss. \text{llength } xs > 1$ )  $\implies$   $\text{lset}(\text{lltl } xss) \cap \{LNil\} = \{\}$ 
unfolding ltl-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  $\text{lset}(\text{lltl } xss) \subseteq \{LNil\}$ 

```

```

shows   llength (lfusecat xss) ≤ 1
using   assms
proof   (induction xss)
case   adm
then show ?case unfolding ltl-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-llength-atmost-one llist.set-intros(1) lset-lltl-llength-var)
qed

```

```

lemma lfusecat-all-empty-or-LNil-b:
assumes llength (lfusecat xss) ≤ 1
shows   lset(ltl 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-llength-atmost-one ltl-ldrop-one)
qed

```

```

lemma lfusecat-all-empty-or-LNil:
shows   llength (lfusecat xss) ≤ 1 ⟷ lset(ltl 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
  ∀ xs ∈ lset xss. ¬lnull xs

```

```

shows  $\neg \text{lnull}(\text{lfusecat } xss)$ 
using assms
using lfusecat-not-lnull llist.set-sel(1) by blast

lemma lfirst-lfusecat-lfirst:
assumes  $\neg \text{lnull } xss$ 
 $\neg \text{lnull } (\text{lfirst } xss)$ 
shows  $\text{lfirst}(\text{lfusecat } xss) = \text{lfirst}(\text{lfirst } xss)$ 
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  $\neg \text{lnull } xs$ 
shows  $\text{lfirst}(\text{lfusecat } (\text{LCons } xs \ xss)) = \text{lfirst } xs$ 
using assms by (simp add: lfirst-def)

```

```

lemma ltl-lfusecat :
assumes  $\neg \text{lnull } xss$ 
 $\neg \text{lnull } (\text{lfirst } xss)$ 
shows  $\text{ltl}(\text{lfusecat } xss) = \text{lappend } (\text{ltl } (\text{lhs } xss)) (\text{ltl } (\text{lfusecat } (\text{tl } xss)))$ 
using assms
unfolding lfirst-def
by (metis lappend-lnull2 lfusecat-LCons lhs-LCons-ltl ltl-lfuse)

```

```

lemma lfusecat-lappend:
assumes lfinite xss
shows  $\text{lfusecat } (\text{lappend } xss \ yss) = \text{lfuse } (\text{lfusecat } xss) (\text{lfusecat } yss)$ 
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:  $\text{lfusecat } (\text{lappend } (\text{LCons } xs \ xss) \ yss) = \text{lfuse } xs (\text{lfusecat } (\text{lappend } xss \ yss))$ 
by simp
have 2: lfinite xss
by (simp add: lfinite-LConsI.hyps(1))
have 3:  $(\text{lfusecat } (\text{lappend } xss \ yss)) = \text{lfuse } (\text{lfusecat } xss) (\text{lfusecat } yss)$ 
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-lnull1 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
   $\forall xs \in \text{lset } xss. \text{lfinite } xs$ 
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)
  next
  case False
  then show ?thesis
    proof -
    have 1: lfinite xs
      by (simp add: lfinite-LConsI.prem)
    have 5:  $\forall xs \in \text{lset } xss. \text{lfinite } xs$ 
      by (simp add: lfinite-LConsI.prem)
    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
qed
qed

```

```

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

lemma llastfirst-lfusecat-llast:
  assumes lfinite xss
     $\neg \text{lnull } xss$ 
     $\forall xs \in \text{lset } xss. \neg \text{lnull } xs$ 
    llastlfirst xss
     $\forall xs \in \text{lset } xss. \text{lfinite } xs$ 
  shows  $\text{llast}(\text{lfusecat } xss) = \text{llast}(\text{llast } xss)$ 
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:  $\forall xs \in \text{lset } xss. \text{lfinite } xs$ 
        by (simp add: lfinite-LConsI.prem(4))
      have 3:  $\forall xs \in \text{lset } xss. \neg \text{lnull } xs$ 
        using lfinite-LConsI.prem(2) by auto
      have 4: lfinite xss
        by (simp add: lfinite-LConsI.hyps)
      have 5:  $\text{llast}(\text{lfusecat } xss) = \text{llast}(\text{llast } xss)$ 
        using 1 2 3 False lfinite-LConsI.IH llist.collapse(1) by blast
      have 6:  $\text{llast}(\text{llast}(\text{LCons } xs \ xss)) = \text{llast}(\text{llast } xss)$ 
        by (metis False llast-LCons llist.collapse(1))
      have 7:  $(\text{lfusecat}(\text{LCons } xs \ xss)) = \text{lfuse } xs \ (\text{lfusecat } xss)$ 
        by simp
      have 8:  $\neg \text{lnull } xs$ 
        by (simp add: lfinite-LConsI.prem(2))
      have 9: lfinite xs
        by (simp add: lfinite-LConsI.prem(4))
      have 10:  $\neg \text{lnull}(\text{lfusecat } xss)$ 
        using 3 False lfusecat-not-tnull-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-lnull 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)
  by (simp add: 13)
show ?thesis
using 13 5 6 by auto
qed
qed
qed

```

lemma *lfusecat-llength-LNil:*
 $llength (lfusecat LNil) = 0$
by *simp*

lemma *lfusecat-llength-lfinite:*
assumes *lfinite xss*
 $\neg lnull xss$
 $\forall xs \in lset xss. \neg lnull xs$
 $\forall xs \in lset xss. lfinite xs$
 $llastlfirst xss$
shows $llength(lfusecat xss) =$
 $eSuc(\sum i = 0 .. (the-enat(epred(llength xss))) . epred(llength (lnth xss i)))$
using *assms*
proof (*induction xss*)
case *lfinite-LNil*
then show ?*case* **by** *simp*
next
case (*lfinite-LConsI xss xs*)
then show ?*case*
proof (*cases xss = LNil*)
case *True*
then show ?*thesis*
proof –
have 1: $lfusecat (LCons xs xss) = xs$
by (*simp add: True*)
have 2: $llength (lfusecat (LCons xs xss)) = llength xs$
using 1 **by** *auto*
have 4: $(\sum i = 0 .. (the-enat(epred(llength(LCons xs xss))))).$
 $epred (llength (lnth (LCons xs xss) i))) =$
 $epred(llength (lnth (LCons xs xss) 0))$
using *True* **by** *simp*
have 5: $eSuc(epred(llength (lnth (LCons xs xss) 0))) = llength xs$
by (*simp add: lfinite-LConsI.prem(2)*)
show ?*thesis*
using 2 4 5 **by** *presburger*


```

qed
next
case False
then show ?thesis
proof -
have 6: (lfusecat (LCons xs xss)) = lfuse xs (lfusecat xss)
  by simp
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 $n \leq \text{length } xss$
 $\text{llastlfirst } xss$
shows $\text{llastlfirst } (\text{ltake } (\text{enat } n) xss)$
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 $\neg \text{lnull } xss$
 $n \leq \text{length } xss$
 $\forall xs \in \text{lset } xss. \neg \text{lnull } xs$
 $\forall xs \in \text{lset } xss. \text{lfinite } xs$
 $\text{llastlfirst } xss$
shows $\text{lfusecat } (\text{ltake } (\text{enat } n) xss) =$
 $\text{ltake } (\text{if } n = 0 \text{ then } 0 \text{ else } \text{eSuc}(\sum i = 0 .. (n-1) . \text{epred}(\text{length } (\text{lnth } xss i))))$
 $(\text{lfusecat } xss)$
proof –
have 1: $\text{lfinite } (\text{ltake } (\text{enat } n) xss)$
using enat-ord-code(4) lfinite-ltake by blast
have 2: $\text{length } (\text{ltake } (\text{enat } n) xss) = n$
by (simp add: assms(2))
have 3: $(\text{the-enat}(\text{epred}(\text{length } (\text{ltake } (\text{enat } n) xss)))) = n-1$
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)
have 7: $\bigwedge j. 0 < n \wedge j < n-1 \longrightarrow$
 $\text{epred}(\text{length } (\text{lnth } (\text{ltake } (\text{enat } n) xss) j)) =$
 $\text{epred}(\text{length } (\text{lnth } xss j))$
by (metis diff-le-self dual-order.strict-trans1 enat-ord-simps(2) lnth-ltake)
have 71: $\text{length } (\text{lfusecat } (\text{ltake } (\text{enat } n) xss)) =$
 $(\text{if } n = 0 \text{ then } 0 \text{ else } \text{eSuc}(\sum i = 0 .. (n-1) . \text{epred}(\text{length } (\text{lnth } (\text{ltake } (\text{enat } n) xss) i))))$

```

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

```

lemma *lfusecat-ldrop*:

assumes $\neg \text{lnull } xss$

$n < \text{llength } xss$

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

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

llastlfirst *xss*

shows *lfusecat* (*ldrop* (*enat* *n*) *xss*) =

ldrop (*if* *n* = 0 *then* 0 *else* ($\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } xss i)))$)
 (*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: $\forall xs \in \text{lset } (\text{ltake } (\text{enat } n) \text{ } xss). \neg \text{lnull } xs$

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

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

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

have 6: *llastlfirst* (*ltake* (*enat* *n*) *xss*)

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

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(∑ i = 0 .. (n-1) . epred(llength (lnth (ltake (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(∑ i = 0 .. (n-1) . epred(llength (lnth (ltake (enat n) xss) i)))) =
  (if n = 0 then 0 else eSuc(∑ 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}]
    λi. epred(llength (lnth (ltake (enat n) xss) i))
    λi. epred(llength (lnth xss i)) ]
  by (simp add: Suc-ile-eq dual-order.refl lnth-ltake)
have 9: llength (lfusecat (ltake (enat n) xss)) ≤ llength (lfusecat xss)
  by (metis lfuse-llength-le-a lfusecat-split)
have 10: llength (ltake (if n = 0 then 0 else eSuc(∑ i = 0 .. (n-1) . epred(llength (lnth xss i))))
  (lfusecat xss)) =
  (if n = 0 then 0 else eSuc(∑ 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(∑ 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 ⟹ ¬ lnull (lfusecat (ltake (enat (n::nat)) (xss::'a llist llist)))
  using 71 8 by force
have 14: 0 < n ⟹ ¬ 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 ⟹ 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 ⟹ ldrop (epred (eSuc(∑ 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 $\sum i = 0 .. (n-1) . \text{epred}(\text{llength} (\text{lnth } xss \ i)))$)
 (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

($\exists x s' x s s' . xss = \text{lappend} (\text{llist-of } x s s') (LCons (LCons x x s') x s s'') \wedge$
 $x s = \text{lappend } x s' (\text{ltl } (\text{lfusecat } x s s'')) \wedge \text{set } x s s' \subseteq \{x s . \text{lnull } x s\}$)
 (is ?lhs \longleftrightarrow ?rhs)

proof

assume ?rhs

then obtain x s' x s s''

where xss = lappend (llist-of x s s') (LCons (LCons x x s') x s s'')

and xs = (lappend x s' (ltl (lfusecat x s s'')))

and set x s s' $\subseteq \{x s . \text{lnull } x s\}$ by blast

moreover from $\langle \text{set } x s s' \subseteq \{x s . \text{lnull } x s\} \rangle$

have lnul (lfusecat (llist-of x s s'))

by (metis lfusecat-not-lnul lset-llist-of mem-Collect-eq subset-eq)

have 1: lfusecat xss = lfuse (lfusecat (llist-of x s s')) (lfusecat (LCons (LCons x x s') x s s''))

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

have 2: lfuse (lfusecat (llist-of x s s')) (lfusecat (LCons (LCons x x s') x s s'')) =

(lfusecat (LCons (LCons x x s') x s s''))

by (simp add: $\langle \text{lnul } (\text{lfusecat } (\text{llist-of } (x s s' :: 'a \text{ llist list}))) \rangle$ lfuse-conv-lnul llist.expand)

have 3: (lfusecat (LCons (LCons x x s') x s s'')) = lfuse (LCons x x s') (lfusecat x s s'')

by simp

have 4: lfuse (LCons x x s') (lfusecat x s s'') =

(LCons x (lappend x s' (ltl (lfusecat x s s''))))

unfolding lfuse-def

using llist.collapse(1) by force

ultimately show ?lhs

using 1 2 by auto

next

assume ?lhs

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

hence $\neg \text{lset } xss \subseteq \{x s . \text{lnull } x s\}$ using lfusecat-eq-LNil lnul-def by blast

hence $\neg \text{lnul } (\text{lfilter } (\lambda x s . \neg \text{lnull } x s) xss)$ by (auto)

then obtain y ys where yss: lfilter $(\lambda x s . \neg \text{lnull } x s) xss = LCons y ys$

unfolding not-lnul-conv by auto

from lfilter-eq-LConsD[OF this]

obtain us vs where xss: xss = lappend us (LCons y ys)

and lfinite us

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

and ys: ys = lfilter $(\lambda x s . \neg \text{lnull } x s) vs$ by blast

from $\langle \text{lfinit} \text{ } us \rangle$ **obtain** us' **where** $[simp]: us = \text{llist-of } us'$
unfolding $\text{lfinit-eq-range-llist-of}$ **by** blast
from $\langle \text{lset } us \subseteq \{xs. \text{lnull } xs\} \rangle$ **have** $us: \text{lnull } (\text{lfusecat } us)$
using $\text{lfusecat-eq-LNil lnull-def}$ **by** blast
from $\langle \neg \text{lnull } y \rangle$ **obtain** $y' \text{ } ys'$ **where** $y: y = \text{LCons } y' \text{ } ys'$
unfolding not-lnull-conv **by** blast
from $\langle ?lhs \rangle us$ **have** $[simp]: y' = x$
 $xs = (\text{lappend } ys' (\text{ltl } (\text{lfusecat } vs)))$
unfolding $xss \text{ } y$
by $(\text{metis } \langle \neg \text{lnull } (y::'a \text{ llist}) \rangle \langle \text{lfinit } (us::'a \text{ llist llist}) \rangle \text{eq-LConsD lfusecat-LCons}$
 $\text{lfusecat-lappend lhd-lfuse } y)$
 $(\text{metis } \langle \neg \text{lnull } (y::'a \text{ llist}) \rangle \langle \text{lfusecat } (xss::'a \text{ llist llist}) = \text{LCons } (x::'a) (xss::'a \text{ llist}) \rangle$
 $\text{eq-LConsD lappend-lnull2 lfusecat-LCons lfusecat-lfilter-neq-LNil ltl-lfuse } y \text{ } ys \text{ } yss)$
from $\langle \text{lset } us \subseteq \{xs. \text{lnull } xs\} \rangle ys$ **show** $?rhs$ **unfolding** $xss \text{ } y$ **by** simp blast
qed

lemma not-lnull-expand :

$\neg \text{lnull } xs \longleftrightarrow (\exists b. xs = (\text{LCons } b \text{ LNil})) \vee (\exists b \text{ } xs'. xs = (\text{LCons } b \text{ } xs') \wedge \neg \text{lnull } xs')$
by $(\text{metis lhd-LCons-ltl llist.disc(1) llist.disc(2) llist.expand})$

lemma is-LMore-llength :

$(\exists b \text{ } xs'. xs = (\text{LCons } b \text{ } xs') \wedge \neg \text{lnull } xs') \longleftrightarrow 1 < \text{llength } xs$
by $(\text{metis dual-order.strict-implies-not-eq gr-implies-not-zero lhd-LCons-ltl llength-LCons}$
 $\text{llength-eq-0 lstrict-prefix-code(2) lstrict-prefix-code(4) lstrict-prefix-llength-less one-eSuc })$

lemma is-LEmpty-llength :

$(\exists b. xs = (\text{LCons } b \text{ LNil})) \longleftrightarrow \text{llength } xs = 1$
by $(\text{metis epred-llength llength-LCons llength-eq-0 llist.disc(1) ltl-simps(2) not-lnull-expand}$
 $\text{one-eSuc})$

lemma $\text{lfusecat-all-llength-one-lfuse}$:

assumes $\forall ys \in \text{lset } (\text{llist-of } xss'). \text{llength } ys = 1$
shows $\text{lfuse } (\text{lfusecat } (\text{llist-of } xss')) \text{ } ys =$
 $\text{lfuse } (\text{if } \neg \text{lnull } (\text{llist-of } xss') \text{ then } (\text{LCons } (\text{lfirst } (\text{lfirst } (\text{llist-of } xss')))) \text{ LNil} \text{ else LNil}) \text{ } ys$

proof –

have $1: (\text{lfusecat } (\text{llist-of } xss')) =$
 $(\text{if } \neg \text{lnull } (\text{llist-of } xss') \text{ then } (\text{LCons } (\text{lfirst } (\text{lfirst } (\text{llist-of } xss')))) \text{ LNil} \text{ else LNil})$
using assms
proof $(\text{induction } xss')$
case Nil
then show $?case$ **by** simp
next
case $(\text{Cons } as \text{ } xss')$
then show $?case$
proof $(\text{cases } xss' = \text{Nil })$
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-lnull zero-one-enat-neq(1))
qed
qed
show ?thesis
using 1 by presburger
qed

lemma lfusecat-eq-LCons-conv-all-lset-singleton:
  assumes  $\forall ys \in \text{lset } (\text{llist-of } xss). \text{llength } ys = 1$ 
     $\neg \text{lnull } (\text{llist-of } xss)$ 
     $\text{llastlfirst } (\text{llist-of } xss)$ 
  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.prem2 List.set-insert insert-iff is-LEmpty-llength
      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-lnull-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.prem2 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  $\forall ys \in \text{lset } (\text{llist-of } xss'). \text{llength } ys = 1$ 
     $\neg \text{lnull } (\text{llist-of } xss')$ 
     $\text{llastlfirst } (\text{llist-of } xss')$ 
     $(\text{llast } (\text{llist-of } xss')) = (\text{LCons } x \text{ LNil})$ 
     $ys \in \text{lset } (\text{llist-of } xss')$ 
  shows  $ys = (\text{LCons } x \text{ LNil})$ 
  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: llast (llist-of xss') = lnth (llist-of xss') (the-enat(epred(llength (llist-of xss'))))
  by (metis assms(2) co.enat.exhaust-sel enat-the-enat ile-eSuc infinity-ileE llast-conv-lnth
    llength-eq-0 llength-llist-of)
have 3: lnth (llist-of xss') (the-enat(epred(llength (llist-of xss')))) ∈ lset (llist-of xss')
  by (metis 2 assms(2) co.enat.exhaust-sel ile-eSuc in-lset-lappend-iff infinity-ileE
    lappend-lbutlast-llast-id llength-eq-0 llength-eq-infnty-conv-lfinite llength-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
   $\forall ys \in \text{lset } (uss). \text{llength } ys = 1$ 
   $\neg \text{lnull } (uss)$ 
  llastlfirst (uss)
  (llast (uss)) = (LCons x LNil)
   $ys \in \text{lset } (uss)$ 

```

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

```

shows $\text{lfusecat } xss = LCons x xs \longleftrightarrow$

```

   $(\exists xs' xss' xss''. xss = \text{lappend } (\text{llist-of } xss') (LCons (LCons x xs') xss'') \wedge \neg \text{lnull } xs' \wedge$ 
     $xs = \text{lfuse } xs' (\text{lfusecat } xss'') \wedge \text{set } xss' \subseteq \{xs. \text{lnull } (\text{ltl } xs)\})$ 
   $(\text{is } ?lhs \longleftrightarrow ?rhs)$ 

```

proof

assume *?rhs*

then obtain $xs' xss' xss''$

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

and $\neg \text{lnull } xs'$

and $xs = (\text{lfuse } xs' (\text{lfusecat } xss''))$

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

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

have 00: $\text{llength } (\text{lfusecat } (\text{llist-of } xss')) \leq 1$

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: $\forall y \in \text{lset } (\text{llist-of } xss'). \text{llength } y = 1$

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-lnull-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')  $\vee$  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:  $\neg$  lnull (LCons (LCons x xs') xss'')
  by simp
have 07:  $\neg$  lnull (llist-of xss')  $\implies$ 
  llast (llast (llist-of xss')) = lfirst (lfirst (LCons (LCons x xs') xss''))
  using 05 06 by presburger
have 08:  $\neg$  lnull (llist-of xss')  $\implies$  (llast (llist-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:  $\neg$  lnull (llist-of xss')  $\implies$  ( $\forall$  ys  $\in$  lset (llist-of xss'). ys = (LCons x LNil))
  using lfusecat-eq-LCons-conv-all-the-same
  by (metis 01 04 08)
have 0: (lfusecat (llist-of xss') = LNil  $\vee$  (lfusecat (llist-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: lfusecat xss = lfuse (lfusecat (llist-of xss')) (lfusecat (LCons (LCons x xs') xss''))
  by (simp add: calculation(1) lfusecat-lappend)
have 2: lfuse (lfusecat (llist-of xss')) (lfusecat (LCons (LCons x xs') xss'')) =
  (lfusecat (LCons (LCons x xs') xss''))
  using 0 by fastforce
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 ( (lfuse xs' ( (lfusecat xss''))
    )))
  unfolding lfuse-def
  using calculation(2) by auto
ultimately show ?lhs
using 1 2 by auto
next
assume ?lhs
hence  $\neg$  lnull (lfusecat xss) by simp
hence  $\neg$  lset xss  $\subseteq$  {xs. lnull (ltl xs)}
by (metis  $\langle$ lfusecat (xss::'a llist llist) = LCons (x::'a) (xs::'a 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  $\neg$  lnull (lfilter ( $\lambda$ xs.  $\neg$  lnull (ltl xs)) xss) by(auto)
then obtain y ys where yss: lfilter ( $\lambda$ xs.  $\neg$  lnull (ltl xs)) xss = LCons y ys
  unfolding not-nullable-conv by auto
from lfilter-eq-LConsD[OF this]

```

```

obtain us vs where xss: xss = lappend us (LCons y vs)
  and lfinite us
  and lset us  $\subseteq \{xs. \text{lnull (ltl xs)}\} \multimap \text{lnull (ltl y)}$ 
  and ys: ys = lfilter ( $\lambda xs. \multimap \text{lnull (ltl xs)}$ ) vs using lnull-ltII by blast
from  $\langle \text{lfinite us} \rangle$  obtain us' where [simp]: us = llist-of us'
  unfolding lfinite-eq-range-llist-of by blast
from  $\langle \text{lset us} \subseteq \{xs. \text{lnull (ltl xs)}\} \rangle$  have us: llength (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  $\langle \multimap \text{lnull (ltl y)} \rangle$  obtain y' ys' where y: y = LCons y' ys'
  unfolding not-lnull-conv
  by (metis  $\langle \multimap \text{lnull (ltl y)} \rangle$  lhd-LCons-ltl lnull-ltII)
have 100: llastfirst us
  using  $\langle \text{lfinite us} \rangle$  assms(1) llastfirst-lappend-lfinite xss by blast
have 101:  $\multimap \text{lnull ys'}$ 
  using  $\langle \multimap \text{lnull (ltl y)} \rangle$  y by auto
have 102:  $\multimap \text{lnull (LCons y vs)}$ 
  by simp
have 103:  $\multimap \text{lnull us} \implies \text{llast (llast us)} = \text{lfirst (lfirst (LCons y vs))}$ 
  using llastfirst-lappend-lfinite[of us (LCons y vs)]
  using assms(1) xss by auto
have 104: ( $\forall z \in \text{lset us. llength } z = 1$ )
  by (metis  $\langle \text{lset us} \subseteq \{xs. \text{lnull (ltl xs)}\} \rangle$  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:  $\multimap \text{lnull us} \implies \text{llast us} \in \text{lset us}$ 
  by simp
have 1041:  $\multimap \text{lnull us} \implies \text{llast us} = \text{LCons } x \text{ LNil}$ 
  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  $\langle \multimap \text{lnull (lfusecat xss)} \rangle \langle \text{lfusecat xss} = \text{LCons } x \text{ xs} \rangle \langle \text{us} = \text{llist-of us'} \rangle$ 
    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:  $\multimap \text{lnull us} \implies (\forall z \in \text{lset us. } z = (\text{LCons } x \text{ LNil}))$ 
  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:  $\multimap \text{lnull us} \implies \text{lfusecat us} = (\text{LCons } x \text{ LNil})$ 
  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  $\langle \multimap \text{lnull (ltl y)} \rangle$  eq-LConsD lappend-lnull1 lfirst-def lfirst-lfusecat
    llast-singleton lnull-ltII xss y)
from  $\langle ?lhs \rangle$  us have [simp]: xs = ( (lfuse ys' ( (lfusecat vs))) )
  using lfusecat-lappend[of us (LCons y vs)] lfusecat-LCons[of y vs] y xss
  101  $\langle \text{lfinite us} \rangle \langle \text{lfusecat xss} = \text{LCons } x \text{ xs} \rangle$ 
  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-ltII)

```

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*:

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

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

$\text{llength } (\text{lfusecat } xss) = 1 \longleftrightarrow$
 $(\exists x xss' xss''. xss = \text{lappend } (\text{llist-of } xss') (\text{LCons } (\text{LCons } x \text{ LNil}) xss'') \wedge$
 $\text{LNil} = (\text{ltl } (\text{lfusecat } xss'')) \wedge \text{set } xss' \subseteq \{xs. \text{lnull } xs\})$
 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*≡(*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: $(\exists xs' xss' xss''. xss = \text{lappend } (\text{llist-of } xss') (\text{LCons } (\text{LCons } x xs') xss'') \wedge$
 $xs = \text{lappend } xs' (\text{ltl } (\text{lfusecat } xss'')) \wedge \text{set } xss' \subseteq \{xs. \text{lnull } xs\})$

using *lfinite-LConsI.hyps(3) lfusecat-eq-LCons-conv* **by** *fastforce*

obtain $xs' xss' xss''$ **where** 2: $xss = \text{lappend } (\text{llist-of } xss') (\text{LCons } (\text{LCons } x xs') xss'') \wedge$
 $xs = \text{lappend } xs' (\text{ltl } (\text{lfusecat } xss'')) \wedge \text{set } xss' \subseteq \{xs. \text{lnull } xs\}$

using 1 **by** *blast*

have 3: $(\text{llist-of } xss') = \text{LNil}$

by (metis 2 *in-lset-lappend-iff lconcat-eq-LNil lfinite-LConsI.prem(1) llist.collapse(1)*
lset.set-sel(1) lnull-lconcat lset-eq-forall-lnull lset-llist-of ltl-simps(1))

have 4: $xss = (\text{LCons } (\text{LCons } x xs') xss'')$

by (*simp add: 2 3*)

have 5: *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.prems)
  have 11:  $\forall xs \in \text{lset } xss''. \text{lfinite } xs$ 
    by (simp add: 4 lfinite-LConsI.prems)
  have 12:  $\text{llastlfirst } xss''$ 
    using 4 5 a by auto
  have 13:  $\text{lfinite}(\text{lfusecat } xss'')$ 
    using 2 lfinite-LConsI.hyps(1) by auto
  have 14:  $xs = \text{lappend } (\text{lbutlast } xs')\ (\text{lfusecat } xss'')$ 
    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:  $xs = \text{lfuse } xs'\ (\text{lfusecat } xss'')$ 
    by (simp add: 2 6 lappend-lnull2 lfuse-def)
  have 16:  $xs = \text{lfusecat } (LCons\ xs'\ xss'')$ 
    by (simp add: 15)
  have 17:  $\text{lnull } (\text{ltl } xs') \implies ?thesis$ 
    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:  $\neg \text{lnull } (\text{ltl } xs') \implies (\forall xs \in \text{lset } (LCons\ xs'\ xss''). \neg \text{lnull } (\text{ltl } xs))$ 
    by (simp add: 10)
  have 19:  $\forall xs \in \text{lset } (LCons\ xs'\ xss''). \text{lfinite } xs$ 
    by (metis 11 2 insert-iff lappend-inf lfinite-LConsI.hyps(1) llist.simps(19))
  have 20:  $\text{llastlfirst } (LCons\ xs'\ xss'')$ 
    by (metis 12 6 9 a llast-LCons llastlfirst-LCons)
  have 21:  $\text{lfinite } (LCons\ xs'\ xss'')$ 
    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 $\forall xs \in \text{lset } xss. \neg \text{lnull } (\text{ltl } xs)$

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

$\text{llastlfirst } xss$

shows $\text{lfinite } (\text{lfusecat } xss) \longleftrightarrow \text{lfinite } xss$

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

lemma *ridx-lfusecat-ltake*:

assumes *ridx* *R* (*lfusecat* *xss*)

$n \leq \text{length } xss$
shows $\text{ridx } R (\text{lfusecat } (\text{ltake } n \ xss))$
using *assms*
by (*metis lfusecat-split ltake-all ltake-lfuse nle-le ridx-ltake-a*)

lemma *ridx-lfusecat-ldrop*:

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

lemma *ridx-lfusecat-a*:

assumes $\text{llastlfirst } xss$
 $\forall xs \in \text{lset } xss. \neg \text{lnull } xs$
 $\forall xs \in \text{lset } xss. \text{lfinite } xs$
 $\text{ridx } R (\text{lfusecat } xss)$
 $i < \text{length } xss$
shows $\text{ridx } R (\text{lnth } xss \ i)$
proof –
have 1: $\text{lsub } i \ i \ xss = (\text{LCons } (\text{lnth } xss \ i) \ \text{LNil})$
by (*simp add: assms(5) lsub-same*)
have 2: $\text{lsub } i \ i \ xss = \text{ltake } (\text{eSuc } (\text{enat } (i - i))) (\text{ldrop } (\text{enat } i) \ xss)$
by (*simp add: lsub-def*)
have 3: $\text{ltake } (\text{eSuc } (\text{enat } (i - i))) (\text{ldrop } (\text{enat } i) \ xss) = \text{ltake } 1 (\text{ldrop } (\text{enat } i) \ xss)$

```

  by (simp add: eSuc-enat one-enat-def)
have 4: ridx R (lfusecat (ltake 1 (ldrop (enat i) xss)))
  by (metis assms ltake-all nle-le ridx-lfusecat-ldrop ridx-lfusecat-ltake)
have 5: (lfusecat (ltake 1 (ldrop (enat i) xss))) = (lnth xss i)
  by (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 ridx R (lfuse l1 l2)  $\longleftrightarrow$  ridx R l1  $\wedge$  ridx R l2
using assms
proof (cases  $\neg$  lnull l1  $\wedge$   $\neg$  lnull l2)
case True
then show ?thesis
  proof (cases  $\neg$  lnull (ltl l2))
  case True
  then show ?thesis
    proof -
    have 1: (lfuse l1 l2) = lappend l1 (ltl l2)
      unfolding lfuse-def using  $\langle \neg$  lnull l1  $\wedge$   $\neg$  lnull l2  $\rangle$  by simp
    have 2: ridx R (lappend l1 (ltl l2)) = (ridx R l1  $\wedge$  ( R (llast l1) (lhd (ltl l2)))  $\wedge$  ridx R (ltl l2))
      using assms ridx-lappend-lfinite[of l1 R ltl l2] True  $\langle \neg$  lnull l1  $\wedge$   $\neg$  lnull l2  $\rangle$ 
      by auto
    have 3: ( R (llast l1) (lhd (ltl l2)))  $\wedge$  ridx R (ltl l2) = ridx R l2
      using True  $\langle \neg$  lnull l1  $\wedge$   $\neg$  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: (lfuse l1 l2) = lappend l1 (ltl l2)
    unfolding lfuse-def using  $\langle \neg$  lnull l1  $\wedge$   $\neg$  lnull l2  $\rangle$  by simp
  have 5: ridx R (lappend l1 (ltl l2)) = (ridx R l1  $\wedge$  ridx R l2)
    using assms ridx-lappend-lfinite[of l1 R ltl l2] False  $\langle \neg$  lnull l1  $\wedge$   $\neg$  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*

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

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

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

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

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*

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

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

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

$n \leq \text{llength } xss$

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: $(\text{lfusecat } (\text{ltake } (\text{enat } (\text{Suc } n)) \ xss)) =$

$\text{ltake } (\text{eSuc } (\sum i::\text{nat} = 0::\text{nat}. \text{Suc } n - (1::\text{nat}). \text{epred } (\text{llength } (\text{lnth } xss \ i)))) (\text{lfusecat } xss)$

using *lfusecat-ltake[of xss (Suc n)]*

```

    using Suc.premis(1) Suc.premis(2) Suc.premis(3) Suc.premis(5) Suc-ile-eq by force
  have 2: ridx R (lfusecat (ltake (enat n) xss))
    using Suc Suc-ile-eq order.strict-implies-order by blast
  have 20: lfinite (ltake (enat n) xss)
    by simp
  have 21: ¬ lnull (ltake (enat n) xss)
    using Suc.premis(5) a enat-0-iff(2) by fastforce
  have 22: ∀ xs ∈ lset (ltake (enat n) xss). ¬ lnull xs
    by (meson Suc.premis(2) lset-ltake subsetD)
  have 23: ∀ xs ∈ lset (ltake (enat n) xss). lfinite xs
    by (meson Suc.premis(3) lset-ltake subsetD)
  have 24: llastlfirst (ltake (enat n) xss)
    using Suc.premis(1) Suc.premis(5) Suc-ile-eq less-imp-le llastlfirst-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.premis(3) Suc.premis(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.premis(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.premis(1) Suc-pred' a llastlfirst-def)
  have 30: ridx R (lfuse (lfusecat (ltake (enat n) xss)) (lnth xss (n)))
    by (metis 2 25 27 28 29 Suc.premis(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 *llastlfirst 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: $\neg lfinite\ xss \implies llength\ (lfusecat\ xss) = \infty$
using *not-lfinite-llength* 2 **by** *blast*
have 50: $\neg lnull\ xss \implies lfusecat\ (ltake\ (enat\ n)\ xss) =$
 $ltake\ (if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i))))$
 $(lfusecat\ xss)$
using *assms lfusecat-ltake*[of *xss* *n*] 0 **by** *fastforce*
have 51: $lfinite\ (lfusecat\ (ltake\ (enat\ n)\ xss))$
by (*meson assms*(3) *enat-ord-code*(4) *lfinite-ltake lfusecat-lfinite-a lset-ltake subsetD*)
have 52: $lfinite\ (ltake\ (if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i))))$
 $(lfusecat\ xss))$
using 50 51 *llist.collapse*(1) **by** *fastforce*
have 53: $\exists m. llength\ (ltake\ (if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i))))$
 $(lfusecat\ xss)) = (enat\ m)$
using 52 *lfinite-llength-enat* **by** *blast*
have 54: $llength\ (ltake\ (if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i))))$
 $(lfusecat\ xss)) \leq llength\ (lfusecat\ xss)$
by *auto*
have 55: $llength\ (ltake\ (if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i))))$
 $(lfusecat\ xss)) =$
 min
 $(if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i))))$
 $(llength\ (lfusecat\ xss))$
using *llength-ltake*[of $(if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i))))$
 $(lfusecat\ xss)]$ **by** *auto*
have 56: min
 $(if\ n = 0\ then\ 0\ else\ eSuc(\sum i = 0 .. (n-1) . epred(llength\ (lnth\ xss\ i))))$
 $(llength\ (lfusecat\ xss)) < \infty$
by (*metis* 53 55 *enat-ord-code*(4))
have 57: $(llength\ (lfusecat\ xss)) = llength\ (lfusecat\ (lappend\ (ltake\ n\ xss)\ (ldrop\ n\ xss)))$
by (*simp add: lappend-ltake-ldrop*)
have 58: $llength\ (lfusecat\ (lappend\ (ltake\ n\ xss)\ (ldrop\ n\ xss))) =$
 $llength\ (lfuse\ (lfusecat\ (ltake\ n\ xss))\ (lfusecat\ (ldrop\ n\ xss)))$
by (*metis* 57 *lfusecat-split*)
have 59: $lnull\ (lfusecat\ (ltake\ (enat\ n)\ xss)) \longleftrightarrow n = 0$
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: $llength\ (lfuse\ (lfusecat\ (ltake\ n\ xss))\ (lfusecat\ (ldrop\ n\ xss))) =$

```

    llength (lfusecat (ltake (enat n) xss)) +
    (if n= 0 then llength (lfusecat (ldrop (enat n) xss))
     else epred (llength (lfusecat (ldrop (enat n) xss))))
using lfuse-llength[of (lfusecat (ltake n xss)) (lfusecat (ldrop n xss))]
using 59 by presburger
have 62:  $n = 0 \implies \text{llength (lfusecat (ldrop (enat n) xss))} = \text{llength (lfusecat xss)}$ 
    using 57 58 59 llist.collapse(1) by force
have 620:  $n > 0 \implies n < \text{llength xss} \implies \neg \text{lnull (ldrop (enat n) xss)}$ 
    using Suc-ile-eq assms by simp
have 621:  $n > 0 \implies n < \text{llength xss} \implies (\exists xs \in \text{lset (ldrop (enat n) xss)}. \neg \text{lnull xs})$ 
    by (meson 0 620 in-lset-ldropD lfusecat-not-lnull lfusecat-not-lnull-var)
have 622:  $n > 0 \implies n < \text{llength xss} \implies (\neg \text{lnull (lfusecat (ldrop (enat n) xss)))}$ 
    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:  $n > 0 \implies n < \text{llength xss} \implies (\text{llength (lfusecat (ldrop (enat n) xss)))} =$ 
     $(\text{llength (ldrop ((\sum i = 0 .. (n-1) . \text{epred}(\text{llength (lnth xss i))})) (lfusecat xss) ))$ 
    using lfusecat-ldrop[of xss n] assms
    using 623 llength-LNil not-one-less-zero 0 by presburger
have 630:  $n > 0 \implies n < \text{llength xss} \implies \text{epred}(\text{llength (lfusecat (ldrop (enat n) xss)))} =$ 
     $\text{epred}(\text{llength (ldrop ((\sum i = 0 .. (n-1) . \text{epred}(\text{llength (lnth xss i))})) (lfusecat xss) ))$ 
    using 63 by presburger
have 631:  $n > 0 \implies n < \text{llength xss} \implies$ 
     $(\text{llength (ldrop ((\sum i = 0 .. (n-1) . \text{epred}(\text{llength (lnth xss i))})) (lfusecat xss) )) > 0$ 
    by (metis 622 63 gr-zeroI llength-eq-0)
have 64:  $\text{llength (lfusecat (ltake (enat n) xss))} \leq$ 
     $\text{llength (lfuse (lfusecat (ltake n xss)) (lfusecat (ldrop n xss)))}$ 
    using lfuse-llength-le-a by blast
have 65:  $0 < n \implies n < \text{llength xss} \implies 0 < (\text{llength (lfusecat (ldrop (enat n) xss)))}$ 
    using 63 631 by presburger
have 66:  $n > 0 \implies \text{llength xss} > 0$ 
    using 623 gr-zeroI llength-eq-0 by blast
have 67:  $n > 0 \implies n-1 \leq (\text{epred} (\text{llength xss}))$ 
    using assms 66
    by (metis epred-enat epred-le-epredI)
have 68:  $\min$ 
     $(\text{if } n = 0 \text{ then } 0 \text{ else } e\text{Suc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength (lnth xss i)})))$ 
     $(\text{llength (lfusecat xss)}) =$ 
     $(\text{if } n = 0 \text{ then } 0 \text{ else } e\text{Suc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength (lnth xss i)})))$ 
proof (cases lfinite xss)
case True
then show ?thesis
proof -
    have 681:  $n > 0 \implies n < \text{llength xss} \implies e\text{Suc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength (lnth xss i)})) \leq$ 
     $e\text{Suc}(\sum i = 0..the-enat (\text{epred} (\text{llength xss})). \text{epred} (\text{llength (lnth xss i)}))$ 
    by (metis 1 631 True gr-implies-not-zero ileI1 linorder-not-less llength-lnull lnull-ldrop)
    have 682:  $n > 0 \implies n = \text{llength xss} \implies e\text{Suc}(\sum i = 0 .. (n-1) . \text{epred}(\text{llength (lnth xss i)})) \leq$ 
     $e\text{Suc}(\sum i = 0..the-enat (\text{epred} (\text{llength xss})). \text{epred} (\text{llength (lnth xss i)}))$ 
    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: $\bigwedge i. i < \text{llength } xss \implies \neg \text{null } (\text{lnth } xss \ i)$

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

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

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*

(metis Suc-ile-eq gr-implies-not-zero llength-eq-0 order.strict-implies-order)
have 61: llength (lfusecat (ltake n xss)) =
 llength (ltake (eSuc($\sum i = 0 .. (n-1)$. epred(llength (lnth xss i)))) (lfusecat xss))
using 60 **by** fastforce
have 62: llength (ltake (eSuc($\sum i = 0 .. (n-1)$. epred(llength (lnth xss i)))) (lfusecat xss)) =
 min (eSuc($\sum i = 0 .. (n-1)$. epred(llength (lnth xss i))))
 (llength (lfusecat xss))
using llength-ltake **by** blast
have 63: min (eSuc($\sum i = 0 .. (n-1)$. epred(llength (lnth xss i))))
 (llength (lfusecat xss)) =
 (eSuc($\sum i = 0 .. (n-1)$. 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: $0 < n \implies$ eSuc($\sum i = 0 .. (n-1)$. epred(llength (lnth xss i))) <
 eSuc($\sum i = 0 .. (n-1)$. 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: $0 < n \implies$?thesis
using 4 7 eSuc-plus **by** auto
show ?thesis
using 3 8 **by** blast
qed

lemma ridx-lfusecat-b:

assumes llastlfirst xss

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

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

$\forall i. i < \text{llength } xss \implies \text{ridx } R (\text{lnth } xss i)$

shows ridx R (lfusecat xss)

proof –

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

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

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

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

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

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: $\bigwedge n. (\text{enat } n) \leq \text{llength } xss \implies$

(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: $n \leq \text{llength } xss$

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( $\sum i = 0 .. (n-1)$  . 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( $\sum i = 0 .. (n-1)$  . 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: $\bigwedge k. (\exists n. \text{enat } k < (\text{if } n=0 \text{ then } 0 \text{ else } e\text{Suc}(\sum_{i=0}^{n-1} . \text{epred}(\text{llength } (\text{lnth } xss \ i))))$
 $i))))$

$\wedge n \leq \text{llength } xss \wedge$
 $\text{ridx } R (\text{ltake } (\text{if } n=0 \text{ then } 0 \text{ else } e\text{Suc}(\sum_{i=0}^{n-1} . \text{epred}(\text{llength } (\text{lnth } xss \ i)))) (\text{lfusecat } xss)))$

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-infity-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: $\text{enat } k < (\text{if } n=0 \text{ then } 0 \text{ else } e\text{Suc}(\sum_{i=0}^{n-1} . \text{epred}(\text{llength } (\text{lnth } xss \ i)))) \wedge$

$n \leq \text{llength } xss \wedge$
 $\text{ridx } R (\text{ltake } (\text{if } n=0 \text{ then } 0 \text{ else } e\text{Suc}(\sum_{i=0}^{n-1} . \text{epred}(\text{llength } (\text{lnth } xss \ i)))) (\text{lfusecat } xss)))$

```

                                eSuc( $\sum i = 0 .. (n-1) . \text{epred}(\text{length} (\text{lnth } xss \ i)))$ ) (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*

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

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

shows $\text{ridx } R (\text{lfusecat } xss) \longleftrightarrow (\forall i. i < \text{length } xss \longrightarrow \text{ridx } R (\text{lnth } xss \ i))$

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*

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

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

shows $(\forall i. i < \text{length } xss \longrightarrow \text{ridx } (\lambda a b. f (\text{lsub } a b \ \sigma)) (\text{lnth } xss \ i)) \longleftrightarrow$
 $\text{ridx } (\lambda a b. f (\text{lsub } a b \ \sigma)) (\text{lfusecat } xss)$

using *assms ridx-lfusecat* **by** *blast*

lemma *lidx-lfuse-lfinite*:

assumes *lfinite xs*

$\text{llast } xs = \text{lfirst } ys$

shows $\text{lidx } (\text{lfuse } xs \ ys) \longleftrightarrow \text{lidx } xs \wedge \text{lidx } ys$

using *assms*

using *ridx-lfuse-lfinite ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-ltake*:

assumes *lidx (lfusecat xss)*

$n \leq \text{length } xss$

shows $\text{lidx } (\text{lfusecat } (\text{ltake } n \ xss))$

using *assms ridx-lfusecat-ltake ridx-lidx* **by** *blast*

lemma *lidx-lfusecat-ldrop*:

assumes *lidx (lfusecat xss)*

$(\text{enat } n) < \text{length } xss$

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

lemma $lidx\text{-}lfusecat\text{-}a$:
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 $assms\ ridx\text{-}lfusecat\text{-}a\ ridx\text{-}lidx$ **by** $blast$

lemma $lidx\text{-}lfusecat\text{-}lfuse$:
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 $assms\ ridx\text{-}lfusecat\text{-}lfuse\ ridx\text{-}lidx$ **by** $blast$

lemma $lidx\text{-}lfusecat\text{-}ltake\text{-}a$:
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 $assms\ ridx\text{-}lfusecat\text{-}ltake\text{-}a\ ridx\text{-}lidx$ **by** $blast$

lemma $lidx\text{-}lfusecat\text{-}b$:
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 $assms\ ridx\text{-}lfusecat\text{-}b\ ridx\text{-}lidx$ **by** $blast$

lemma $lidx\text{-}lfusecat$:
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 $\neg \text{lnull } xss$

$\text{llastlfirst } xss$

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

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

$(\text{enat } i) < \text{llength } xss$

$\text{lfinite } xss$

shows $(\sum i = 0 .. (\text{the-enat}(\text{epred}(\text{llength } xss))) . \text{epred}(\text{llength } (\text{lnth } xss \ i))) =$
 $\text{epred}(\text{llength } (\text{lnth } xss \ i)) +$
 $(\sum j \in \{k. k \neq i \ \wedge \ k \leq (\text{the-enat}(\text{epred}(\text{llength } xss)))\} . \text{epred}(\text{llength } (\text{lnth } xss \ j)))$

proof –

have 1: $\{k. k \leq (\text{the-enat}(\text{epred}(\text{llength } xss)))\} = \text{insert } i \ \{k. k \neq i \ \wedge \ k \leq (\text{the-enat}(\text{epred}(\text{llength } xss)))\}$

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: $\{0.. (\text{the-enat}(\text{epred}(\text{llength } xss)))\} = \{k. k \leq (\text{the-enat}(\text{epred}(\text{llength } xss)))\}$

by *auto*

have 3: $(\sum i = 0 .. (\text{the-enat}(\text{epred}(\text{llength } xss))) . \text{epred}(\text{llength } (\text{lnth } xss \ i))) =$
 $(\sum i \in \{k. k \leq (\text{the-enat}(\text{epred}(\text{llength } xss)))\} . \text{epred}(\text{llength } (\text{lnth } xss \ i)))$

using 2 by *presburger*

have 4: $(\sum i \in \{k. k \leq (\text{the-enat}(\text{epred}(\text{llength } xss)))\} . \text{epred}(\text{llength } (\text{lnth } xss \ i))) =$
 $(\sum j \in (\text{insert } i \ \{k. k \neq i \ \wedge \ k \leq (\text{the-enat}(\text{epred}(\text{llength } xss)))\}) . \text{epred}(\text{llength } (\text{lnth } xss \ j)))$

using 1 by *presburger*

have 5: $(\sum j \in (\text{insert } i \ \{k. k \neq i \ \wedge \ k \leq (\text{the-enat}(\text{epred}(\text{llength } xss)))\}) . \text{epred}(\text{llength } (\text{lnth } xss \ j))) =$
 $\text{epred}(\text{llength } (\text{lnth } xss \ i)) +$

$(\sum j \in \{k. k \neq i \ \wedge \ k \leq (\text{the-enat}(\text{epred}(\text{llength } xss)))\} . \text{epred}(\text{llength } (\text{lnth } xss \ j)))$

by *auto*

show *?thesis*

using 3 4 5 by *presburger*

qed

lemma *lfusecat-llength-a*:

assumes $\text{llastlfirst } xss$

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

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

$i < \text{llength } xss$

$(\text{enat } j) < \text{llength } (\text{lnth } xss \ i)$

shows $(\text{enat } j) < \text{llength } (\text{lfusecat } xss)$

proof (*cases lfinite xss*)

case *True*

then show *?thesis*

proof –

have 1: $\text{lnull } xss \implies \text{?thesis}$

using *assms(4) gr-implies-not-zero llength-eq-0* by *blast*

have 2: $\neg \text{lnull } xss \implies$

$\text{llength } (\text{lfusecat } xss) =$

```

      eSuc( $\sum i = 0 \dots (the-enat(epred(llength\ xss)))$  .  $epred(llength\ (lnth\ xss\ i))$ )
    using True assms lfusecat-llength-lfinite by fastforce
  have 3:  $\neg lnull\ xss \implies$ 
    llength (lnth xss i)  $\leq$ 
    eSuc( $\sum i = 0 \dots (the-enat(epred(llength\ xss)))$  .  $epred(llength\ (lnth\ xss\ i))$ )
    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:  $\neg lnull\ xss \implies ?thesis$ 
    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:  $\neg lfinite\ (lfusecat\ xss)$ 
      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-llength one-enat-def)
    show ?thesis
      using 5 enat-iless lfinite-conv-llength-enat not-less-iff-gr-or-eq by blast
    qed
  qed
qed

```

```

lemma lmap-lfusecat:
  lmap f (lfusecat xss) = (lfusecat (lmap ( 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  $\forall i. i < llength\ xss \longrightarrow is-lfirst(lnth\ xss\ i)$ 
    is-lfirst xss
  shows is-lfirst(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

```

1.8 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 ∘ fst) (lzip xs (iterates Suc n))))
using assms lset-lzipD1 by fastforce

```

```

lemma lmap-fst-imp-b:
assumes lnull (lmap fst (lfilter (P ∘ fst) (lzip xs (iterates Suc n))))
shows lnull (lfilter P xs)
proof -
  have 0: lnull (lmap fst (lfilter (λ (x,k). P x) (lzip xs (iterates Suc n))))
    using assms by auto
  have 1: lnull (lfilter (λ (x,k). P x) (lzip xs (iterates Suc n)))
    using 0 by auto
  have 2: (∀ (x,k) ∈ lset(lzip xs (iterates Suc n)). ¬ P x)
    using 1 by (simp add: prod.case-eq-if)
  have 3: (∀ (x,k) ∈ { (lnth xs i, n+i) | i. enat i < llength xs }. ¬ P x)
    using 2 by (simp add: lset-lzip)
  have 4: (∀ x ∈ { lnth xs i | i. enat i < llength xs }. ¬ P x)
    using 3 by blast
  from 4 show ?thesis by (simp add: lset-conv-lnth)
qed

```

```

lemma lmap-snd-llnull:
  lnull (lmap snd (lfilter (P ∘ 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-llnull-conv:
  lnull (kfilter P n xs) ⟷ (∀ x ∈ lset xs. ¬ P x)
unfolding kfilter-def using lmap-snd-llnull[of P xs] lnull-lfilter[of P xs] by blast

```

```

lemma kfilter-not-llnull-conv:

```

$\neg \text{lnull} (\text{kfilter } P \ n \ xs) \longleftrightarrow (\exists \ x \in \text{lset } xs. \ P \ x)$
by (*simp add: kfilter-lnull-conv*)

lemma *lmap-fst-lfilter*:

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

using *lfilter-lmap* **by** *blast*

lemma *lmap-fst-lzip*:

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

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

lemma *lfilter-kfilter-1*:

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

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

lemma *lfilter-kfilter-snd-llength*:

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

by *simp*

lemma *kfilter-llength*:

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

proof –

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

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

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

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

by (*metis lfilter-kfilter-1*)

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

qed

lemma *ldropWhile-LEAST*:

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

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

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

by *atomize-elim*

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

thus *?thesis* **unfolding** \ast

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:  $\text{ldropWhile } P \text{ (ltl xs)} = \text{ldropn } m \text{ (ltl xs)}$  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:  $P \text{ (lnth xs 0)}$ 
    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 $\text{ldropWhile } (Not \circ P) \text{ xs} = \text{ldropn } (LEAST \text{ } n. n < \text{length } xs \wedge P \text{ (lnth xs } n)) \text{ xs}$
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 $(\text{ltakeWhile } P \text{ xs}) = (\text{ltake } (\text{lleast } (Not \circ P) \text{ xs}) \text{ xs})$
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 *: $(\text{lleast } (Not \circ P) \text{ xs}) = m$ **unfolding** *lleast-def*
by *atomize-elim*
 $(\text{metis (no-types, lifting) Least-le dual-order.trans enat-ord-code(2) not-less not-less-iff-gr-or-eq}$
 $\text{wellorder-Least-lemma(1)})$
thus ?thesis **unfolding** *
proof (*induct m arbitrary: xs*)
case 0
then show ?case
proof –
have $\text{ltakeWhile } P \text{ (LCons (lnth xs 0) (ldropn (Suc 0) xs))} = LNil$
using 0 **by** *fastforce*
then show ?thesis
by (*metis (no-types) lappend.disc-iff(2) lappend-ltakeWhile-ldropWhile ldropn-0*
 $\text{ldropn-Suc-conv-ldropn llist.collapse(1) lnull-ldropn ltake.ctr(1) not-less zero-enat-def}$)
qed
next
case (Suc m)
then show ?case

proof –

have 1: $ltakeWhile\ P\ (lth\ xs) = ltake\ m\ (lth\ xs)$

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 $\exists\ n < llength\ xs.\ (Not \circ P)\ (lnth\ xs\ n)$

shows $(lleast\ (Not \circ P)\ xs) < llength\ xs$

using *assms* **unfolding** *lleast-def*

by (*metis (no-types, lifting) LeastI*)

lemma *llength-LEAST*:

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

shows $(lleast\ P\ xs) < llength\ xs$

using *assms llength-LEAST-not[of xs Not \circ P]* **unfolding** *lleast-def* **by** *simp*

lemma *llength-ltakeWhile-LEAST*:

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

shows $(llength\ (ltakeWhile\ P\ xs)) = (lleast\ (Not \circ P)\ xs)$

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 $\exists\ n < llength\ xs.\ P\ (lnth\ xs\ n)$

shows $(llength\ (ltakeWhile\ (Not \circ P)\ xs)) = (lleast\ P\ xs)$

using *assms llength-ltakeWhile-LEAST[of xs Not \circ P]* **unfolding** *lleast-def* **by** *simp*

lemma *lzip-ldropWhile-fst*:

assumes $llength\ (lzip\ xs\ ys) = llength\ xs$

shows $lzip\ (ldropWhile\ P\ xs)\ (lmap\ snd\ (ldropWhile\ (P \circ fst)\ (lzip\ xs\ ys))) =$
 $ldropWhile\ (P \circ fst)\ (lzip\ xs\ ys)$

using *assms*

proof –

have *f1*: $ldropWhile\ P\ xs = lmap\ fst\ (ldropWhile\ (P \circ fst)\ (lzip\ xs\ ys))$

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*:

ldropWhile (Not \circ P \circ fst) (lzip xs (iterates Suc n)) =
lzip (ldropWhile (Not \circ P) xs) (lmap snd (ldropWhile (Not \circ P \circ fst) (lzip xs (iterates Suc n))))
by (metis llength-lmap lmap-fst-lzip lzip-ldropWhile-fst)

lemma *ldropWhile-iterates-split*:

assumes $\exists n < \text{length } xs. P (\text{nth } xs \ n)$
shows ((ldropWhile (Not \circ P \circ fst) (lzip xs (iterates Suc n)))) =
(lzip (ldropWhile (Not \circ P) xs)
(iterates Suc (n + (least P xs))))

proof –

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

using lmap-fst-lzip[of xs n]

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

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

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

have 3: (ldropn (LEAST na. na < length(lzip xs (iterates Suc n)) \wedge
((Not \circ (Not \circ P \circ fst))) (lnth (lzip xs (iterates Suc n)) na)) (lzip xs (iterates Suc n))) =
(ldropn (LEAST na. na < length(lzip xs (iterates Suc n)) \wedge
((P \circ fst))) (lnth (lzip xs (iterates Suc n)) na)) (lzip xs (iterates Suc n)))

by auto

have 4: (ldropn (LEAST na. na < length(lzip xs (iterates Suc n)) \wedge
((P \circ fst))) (lnth (lzip xs (iterates Suc n)) na)) (lzip xs (iterates Suc n))) =
(lzip (ldropn (LEAST na. na < length(lzip xs (iterates Suc n)) \wedge
((P \circ fst))) (lnth (lzip xs (iterates Suc n)) na)) xs)
(ldropn (LEAST na. na < length(lzip xs (iterates Suc n)) \wedge
((P \circ fst))) (lnth (lzip xs (iterates Suc n)) na)) (iterates Suc n)))

using ldropn-lzip by blast

have 5: (ldropn (LEAST na. na < length(lzip xs (iterates Suc n)) \wedge
((P \circ fst))) (lnth (lzip xs (iterates Suc n)) na)) (iterates Suc n)) =
(iterates Suc (n + (LEAST na. na < length(lzip xs (iterates Suc n)) \wedge
((P \circ fst))) (lnth (lzip xs (iterates Suc n)) na))))

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

have 6: (LEAST na. na < length(lzip xs (iterates Suc n)) \wedge
((P \circ fst))) (lnth (lzip xs (iterates Suc n)) na)) =
(LEAST na. na < length(lzip xs (iterates Suc n)) \wedge
((P \circ fst))) (lnth (xs) na, lnth (iterates Suc n) na))

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

have 7: length(lzip xs (iterates Suc n)) = length xs

by simp

have 8: (LEAST na. na < length(lzip xs (iterates Suc n)) \wedge
((P \circ fst))) (lnth (xs) na, lnth (iterates Suc n) na)) =
(LEAST na. na < length xs \wedge P (lnth xs na))

by auto
have 9: (lzip (ldropn (LEAST na. na < llength(lzip xs (iterates Suc n)) \wedge
 (((P \circ fst))) (lnth (lzip xs (iterates Suc n)) na)) xs)
 (ldropn (LEAST na. na < llength(lzip xs (iterates Suc n)) \wedge
 (((P \circ fst))) (lnth (lzip xs (iterates Suc n)) na)) (iterates Suc n))) =
 (lzip (ldropn (LEAST na. na < llength xs \wedge P (lnth xs na)) xs)
 (iterates Suc (n+(LEAST na. na < llength xs \wedge P (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 lnull(kfilter P n xs)

shows kfilter P n xs =

(LCons (n+ (lleast P xs))
 (lmap snd (lfilter (P \circ fst)
 (lzip (ldrop (Suc (lleast P xs)) (xs))
 (iterates Suc (Suc (n+(lleast P xs))))))))

proof –

let ?Least = (lleast P xs)

have 1: lhd(lfilter (P \circ fst) (lzip xs (iterates Suc n))) =
 lhd (ldropWhile (Not \circ P \circ fst) (lzip xs (iterates Suc n)))

by simp

have 2: ltl(lfilter (P \circ fst) (lzip xs (iterates Suc n))) =
 (lfilter (P \circ fst) (ltl (ldropWhile (Not \circ P \circ fst) (lzip xs (iterates Suc n)))))

by (simp add: ltl-lfilter)

have 3: lfilter (P \circ fst) (lzip xs (iterates Suc n)) =
 (LCons (lhd(lfilter (P \circ fst) (lzip xs (iterates Suc n))))
 (ltl(lfilter (P \circ fst) (lzip xs (iterates 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: kfilter P n xs =
 lmap snd (lfilter (P \circ fst) (lzip xs (iterates Suc n)))

by (simp add: kfilter-def)

have 5: ltl (ldropWhile (Not \circ P \circ fst) (lzip xs (iterates Suc n))) =
 ltl((lzip (ldropWhile (Not \circ P) xs)
 (lmap snd (ldropWhile (Not \circ P \circ fst) (lzip xs (iterates Suc n)))))

by (metis lzip-ldropWhile-fst-iterates)

have 6: ltl((lzip (ldropWhile (Not \circ P) xs)
 (lmap snd (ldropWhile (Not \circ P \circ fst) (lzip xs (iterates Suc n))))) =
 (lzip (ltl (ldropWhile (Not \circ P) xs)
 (ltl (lmap snd (ldropWhile (Not \circ P \circ fst) (lzip xs (iterates Suc n)))))

using lzip-ldropWhile-fst-iterates[of P xs n]

ltl-lzip[of (ldropWhile (Not \circ P) xs)
 (lmap snd (ldropWhile (Not \circ P \circ fst) (lzip xs (iterates Suc n)))]

by force

have 7: lmap snd (
 (LCons (lhd (ldropWhile (Not \circ P \circ fst) (lzip xs (iterates Suc n))))
 (lfilter (P \circ fst) (ltl (ldropWhile (Not \circ P \circ fst) (lzip xs (iterates Suc n))))) =


```

      (LCons (snd (lhd (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))))))
        (lmap snd (lfilter (P ∘ fst) (ltl (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n)))))))
    by simp
  have 8: ∃ n. enat n < llength xs ∧ P (lnth xs n)
    by (metis assms in-lset-conv-lnth kfilter-llength llength-eq-0 lnull-lfilter)
  have 9: (snd (lhd (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n)))) =
    (snd (lhd (lzip (ldropWhile (Not ∘ P) xs)
      (iterates Suc (n + ?Least)))))
    using 8 ldropWhile-iterates-split[of xs P n] by simp
  have 10: (snd (lhd (lzip (ldropWhile (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-ldropWhile-fst-iterates snd-conv)
  have 11: lhd (iterates Suc (n + ?Least)) = (n + ?Least)
    by simp
  have 12: ltl (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))) =
    ltl (lzip (ldropWhile (Not ∘ P) xs)
      (iterates Suc (n + ?Least)))
    using 8 ldropWhile-iterates-split[of xs P n] by simp
  have 13: ltl (lzip (ldropWhile (Not ∘ P) xs) (iterates Suc (n + ?Least))) =
    (lzip (ltl (ldropWhile (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-ldropWhile-fst-iterates)
  have 14: (ltl (iterates Suc (n + ?Least))) = (iterates Suc (Suc (n + ?Least)))
    by simp
  have 15: ltl (ldropWhile (Not ∘ P) xs) = ltl (ldrop (llength (ltakeWhile (Not ∘ P) xs)) xs)
    by (simp add: ldropWhile-eq-ldrop)
  have 16: ltl (ldrop (llength (ltakeWhile (Not ∘ P) xs)) xs) =
    ldrop (eSuc (llength (ltakeWhile (Not ∘ P) xs))) (xs)
    by (simp add: ldrop-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: ldrop (eSuc (llength (ltakeWhile (Not ∘ P) xs))) (xs) = ldrop (Suc ?Least) (xs)
    using 17 18 by (simp add: eSuc-enat)
  have 20: ltl (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))) =
    (lzip (ldrop (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 (ldrop (Suc (lleast P xs)) (xs))
        (iterates Suc (Suc (n + (lleast P xs)))))))

```

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

lemma *kfilter-eq-LCons-1*:

kfilter P n xs = LCons x xs' \implies
x = (n+(lleast P xs)) \wedge
xs' = kfilter P (Suc (n+(lleast P xs))) (ldrop (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 \vee
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 $\neg \text{lnull}(kfilter P n xs)$

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 $\neg \text{lnull}(kfilter P n xs)$

shows *lleast P xs < llength xs*

using *assms exist-lset-lnth*[of *xs P*] *kfilter-not-lnull-conv*[of *P n xs*]

llength-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  $zs \equiv kfilter\ P\ n\ xs$  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-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-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-LEAST-a[of P n xs] LCons.hyps(2) enat-add-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-ldropWhile[of P n xs] 2 kfilter-eq-LCons-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-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-nat-def enat-ord-simps(2) llength-LCons llength-LNil
        llist.collapse(1) not-less-eq one-eSuc one-enat-def)
    have 10:  $i>0 \wedge \neg lnull\ xs' \implies (i-1) < llength\ xs'$ 
      using 2 LCons.prem(1) Suc-ile-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-simps(2))
    have 111:  $lappend\ (ltake\ (Suc\ (?Least))\ xs)\ (ldrop\ (Suc\ (?Least))\ xs) = xs$ 
      using lappend-ltake-ldrop by blast
    have 112:  $enat\ (Suc\ (n + ?Least)) = n + (Suc\ ?Least)$ 
      by auto
    have 113:  $Suc\ ?Least \leq (llength\ xs)$ 
      using 5 Suc-ile-eq enat-add-mono by blast
    have 114:  $llength\ (ltake\ (Suc\ (?Least))\ xs) = Suc\ ?Least$ 
      using llength-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-lappend[of (ltake (Suc (?Least)) xs) (ldrop (Suc (?Least)) xs)]
      by (metis (no-types, lifting) 111 112 114 eq-refl group-cancel.add1 plus-enat-simps(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-order.strict-implies-order by blast
    show ?thesis
      using 15 6 by blast

```

```

    qed
  qed
next
case False
then show ?thesis
by (metis enat-ord-simps(4) iadd-le-enat-iff kfilter-llength leD leI llength-eq-enat-lfiniteD
    llength-eq-infty-conv-lfinite llength-lfilter-ile)
qed

```

```

lemma kfilter-lowerbound:
  assumes  $i < \text{llength}(kfilter\ P\ n\ xs)$ 
  shows  $n \leq (\text{lnth}\ (kfilter\ P\ n\ xs)\ i)$ 
  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 = lleast P xs
    have 7:  $\exists\ x\ xs'. (kfilter\ P\ n\ xs) = LCons\ x\ xs'$ 
      using Suc.prems 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:  $\text{lnth}\ (kfilter\ P\ n\ xs)\ (Suc\ i) = \text{lnth}\ xs'\ i$ 
      by (simp add: 8)
    have 10:  $xs' = kfilter\ P\ (Suc\ (n + ?Least))\ (ldrop\ (Suc\ ?Least)\ xs)$ 
      using kfilter-ldropWhile[of P n xs] 8 kfilter-eq-LCons-1 by blast
    have 11:  $enat\ i < \text{llength}\ (kfilter\ P\ (Suc\ (n + ?Least))\ (ldrop\ (Suc\ ?Least)\ xs))$ 
      by (metis 10 8 Extended-Nat.eSuc-mono Suc.prems(1) eSuc-enat llength-LCons)
    have 12:  $lnull\ xs' \implies n \leq \text{lnth}\ (kfilter\ P\ n\ xs)\ (Suc\ i)$ 
      using 10 11 gr-implies-not-zero llength-eq-0 by blast
    have 13:  $\neg lnull\ xs' \implies (Suc\ (n + ?Least)) \leq \text{lnth}\ xs'\ i$ 
      using 10 11 Suc.hyps by blast
    have 14:  $\neg lnull\ xs' \implies n \leq \text{lnth}\ (kfilter\ P\ n\ xs)\ (Suc\ i)$ 
      using 13 9 by linarith
    show ?thesis using 12 14 by blast
  qed
qed

```

```

lemma kfilter-mono:
  assumes  $(Suc\ i) < \text{llength}(kfilter\ P\ n\ xs)$ 
  shows  $(\text{lnth}\ (kfilter\ P\ n\ xs)\ i) < (\text{lnth}\ (kfilter\ P\ n\ xs)\ (Suc\ i))$ 
  using assms
  proof (induct i arbitrary: xs n)
  case 0
  then show ?case
  proof -

```

```

let ?Least = lleast P xs
have 1:  $\exists x \, xs'. \text{kfilter } P \, n \, xs = LCons \, x \, xs'$ 
  using 0.premis gr-implies-not-zero kfilter-ldropWhile llength-lnull by blast
obtain x xs' where 2:  $\text{kfilter } P \, n \, xs = LCons \, x \, xs'$ 
  using 1 by auto
have 3:  $x = n + ?Least$ 
  using kfilter-ldropWhile[of P n xs] by (simp add: 2)
have 4:  $\text{lnth } (\text{kfilter } P \, n \, xs) \, 0 = n + ?Least$ 
  by (simp add: 2 3)
have 5:  $\text{lnth } (\text{kfilter } P \, n \, xs) \, (\text{Suc } 0) = \text{lnth } xs' \, 0$ 
  by (simp add: 2)
have 6:  $xs' = \text{kfilter } P \, (\text{Suc } (n + ?Least)) \, (\text{ldrop } (\text{Suc } ?Least) \, xs)$ 
  using kfilter-ldropWhile[of P n xs] 2 kfilter-eq-LCons-1 by blast
have 7:  $n + ?Least < \text{lnth } xs' \, 0$ 
  by (metis 0.premis 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:  $\exists x \, xs'. \text{kfilter } P \, n \, xs = LCons \, x \, xs'$ 
    using Suc.premis gr-implies-not-zero kfilter-ldropWhile llength-lnull by blast
  obtain x xs' where 9:  $\text{kfilter } P \, n \, xs = LCons \, x \, xs'$ 
    using 8 by auto
  have 10:  $xs' = \text{kfilter } P \, (\text{Suc } (n + ?Least)) \, (\text{ldrop } (\text{Suc } ?Least) \, xs)$ 
    using kfilter-ldropWhile[of P n xs] 9 kfilter-eq-LCons-1 by blast
  have 11:  $\text{lnth } (\text{kfilter } P \, n \, xs) \, (\text{Suc } (\text{Suc } i)) = \text{lnth } xs' \, (\text{Suc } i)$ 
    by (simp add: 9)
  have 12:  $\text{lnth } (\text{kfilter } P \, n \, xs) \, (\text{Suc } i) < \text{lnth } xs' \, (\text{Suc } i)$ 
    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  $i < \text{llength}(\text{kfilter } P \, n \, xs)$ 
  shows  $(\text{lnth } xs \, ((\text{lnth } (\text{kfilter } P \, n \, xs) \, i) - n)) = (\text{lnth } (\text{lfilter } P \, xs) \, i)$ 
  using assms
  proof (induct i arbitrary: xs n)
  case 0
  then show ?case
  proof -
    let ?Least = lleast P xs
    have 1:  $\text{lnth } (\text{kfilter } P \, n \, xs) \, 0 = n + ?Least$ 
      using 0.premis kfilter-ldropWhile zero-enat-def by fastforce
    have 2:  $(\text{lnth } xs \, ((\text{lnth } (\text{kfilter } P \, n \, xs) \, 0) - n)) =$ 

```

```

      (lnth xs ?Least)
    by (simp add: 1)
  have 3: (lnth (lfilter P xs) 0) = lhd (ldropWhile (Not ∘ P) xs)
    by (metis (full-types) 0.prem1 kfilter-llength lhd-conv-lnth lhd-lfilter llength-eq-0 not-iless0)
  have 4: ∃ 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) = ldroptn ?Least xs
    using ldropWhile-LEAST-not[of xs P ] 4 unfolding lleast-def by blast
  have 6: lhd (ldroptn ?Least xs) = (lnth xs ?Least)
    by (simp add: 4 lhd-ldroptn 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.prem1 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.prem1(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-lnull-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 (ldroptn (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-lnull-conv llist.disc(2))
  have 17: (ltl (ldroptn (LEAST n. n < llength xs ∧ (Not ∘ (Not ∘ P)) (lnth xs n) ) xs)) =
    (ltl (ldroptn ?Least xs))
    unfolding lleast-def by auto
  have 18: (ltl (ldroptn ?Least xs)) =
    ldroptn (Suc ?Least) xs
    by (simp add: ldroptn-ltl ltl-ldroptn)
  have 19: ldroptn (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: (Suc (n+?Least)) ≤ lnth (kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs)) i
    using kfilter-lowerbound[of i P Suc (n+?Least) (ldrop (Suc ?Least) xs) ] 11 by force
  have 24: (Suc ?Least) > llength (ltake ?Least xs)
    by (simp add: min.strict-coboundedI1)
  have 25: (ldrop (Suc ?Least) (lappend (ltake ?Least xs) (ldrop ?Least xs))) =
      ldrop ((Suc ?Least) - llength (ltake ?Least xs) ) (ldrop ?Least xs)
    by (simp add: ldrop-lappend)
  have 26: lappend (ltake (Suc ?Least) xs) (ldrop (Suc ?Least) xs) = xs
    by (simp add: lappend-ltake-ldrop)
  have 27: (Suc ?Least) ≤ (lnth xs' i - n)
    using 10 23 by auto
  have 271: lnth xs' i - n - (Suc ?Least) = lnth xs' i - (Suc (n+?Least))
    by auto
  have 28: lnth (lappend (ltake (Suc ?Least) xs) (ldrop (Suc ?Least) xs)) (lnth xs' i - n) =
      lnth (ldrop (Suc ?Least) xs) (lnth xs' i - (Suc (n+?Least)))
    using lnth-lappend[of (ltake (Suc ?Least) xs) (ldrop (Suc ?Least) xs) (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: lnth xs (lnth xs' i - n) =
      lnth (ldrop (Suc ?Least) xs) (lnth xs' i - (Suc (n+?Least)))
    using 28 by (metis (no-types, lifting) 26)
  have 30: ¬lnull xs' ⇒
      lnth xs (lnth (kfilter P n xs) (Suc i) - n) = lnth (lfilter P xs) (Suc i)
    by (metis 13 20 21 29)
  show ?thesis
    using 12 30 by blast
qed

```

lemma *in-kfilter-lset*:

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

proof

assume *?lhs*

thus *?rhs*

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

case (*find*)

then show *?case*

proof –

let *?Least = lleast P xs*

have 1: *lhd (kfilter P n xs) = n+?Least*

```

    using kfilter-ldropWhile[of P n xs] find by auto
  have 2: P (lnth xs ?Least) unfolding lleast-def
    by (metis (mono-tags, lifting) LeastI exist-lset-lnth find.hyps kfilter-lnull-conv)
  have 3: ?Least < llength xs
    by (metis find.hyps length-LEAST-a )
  have 4: n+?Least ∈ {n + i | i. enat i < llength xs ∧ P (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)) ⇒
    y ∈ {n + i | i. enat i < llength xs ∧ P (lnth xs i)}
    by (metis 5 gr-implies-not-zero in-lset-conv-lnth llength-eq-0 step.hyps(2))
  have 7: ¬ lnull (kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs)) ⇒
    y ∈ {(Suc (n+?Least)) + i | i. enat i < llength (ldrop (Suc ?Least) xs) ∧
      P (lnth (ldrop (Suc ?Least) xs) i)}
    using step.hyps using 5 by blast
  have 8: (Suc (n+?Least)) = n + Suc ?Least
    by auto
  have 9: ¬ lnull(kfilter P (Suc (n + ?Least)) (ldrop (enat (Suc ?Least)) xs)) ⇒
    ∃ i. y = n + Suc (?Least) + i ∧ enat i < llength (ldrop (enat (Suc ?Least)) xs) ∧
      P (lnth (ldrop (enat (Suc ?Least)) xs) i) ⇒
    ∃ i. y = n + i ∧ enat i < llength xs ∧ P (lnth xs i)
  proof -
    assume a0: ¬ lnull(kfilter P (Suc (n + ?Least)) (ldrop (enat (Suc ?Least)) xs))
    assume a1: ∃ i. y = n + Suc (?Least) + i ∧ enat i < llength (ldrop (enat (Suc ?Least)) xs) ∧
      P (lnth (ldrop (enat (Suc ?Least)) xs) i)
    show ∃ i. y = n + i ∧ enat i < llength xs ∧ P (lnth xs i)
  proof -
    obtain i where 10: y = n + Suc (?Least) + i ∧ enat i < llength (ldrop (enat (Suc ?Least)) xs) ∧
      P (lnth (ldrop (enat (Suc ?Least)) xs) i)
    using a1 by auto
    have 11: Suc (?Least) + i < llength xs
      by (metis 10 add commute ldrop-enat ldrop-ldrop leD le-less-linear lnull-ldropn
        plus-enat-simps(1))
    have 12: P (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: ¬ lnull (kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs)) ⇒
  y ∈ {n + i | i. enat i < llength xs ∧ P (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-ldropWhile[of P n xs] using 0.premis
    by (metis in-lset-conv-lnth kfilter-lnull-conv lhd-LCons)
  show ?thesis
    by (metis 0.premis add.right-neutral kfilter-LCons ldropsn-0 ldropsn-Suc-conv-ldropsn
      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 Suc(2) by auto
  have 19:  $\neg \text{lnull } xs \implies P (\text{lnth } xs \ (Suc \ i)) = P (\text{lnth } (\text{ltl } xs) \ (i))$ 
    by (simp add: lnth-ltl)
  have 20:  $\neg \text{lnull } xs \implies x = (Suc \ n) + i \wedge \text{enat } i < \text{llength } (\text{ltl } xs) \wedge P (\text{lnth } (\text{ltl } xs) \ (i))$ 
    by (metis 19 Extended-Nat.eSuc-mono Suc.premis(1) add-Suc-shift eSuc-enat lhd-LCons-ltl
      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 in-lset-conv-lnth kfilter-lnull-conv)
  have 22:  $\neg \text{lnull } xs \implies \neg \text{lnull } (kfilter \ P \ n \ (\text{ltl } xs)) \implies x \in \text{lset } (kfilter \ P \ (Suc \ n) \ (\text{ltl } xs))$ 
    by (metis 20 Suc.hyps)
  have 23:  $\neg \text{lnull } xs \implies x \in \text{lset } (kfilter \ P \ n \ xs)$ 
    using kfilter-LCons[of P n lhd xs ltl xs]
      in-lset-ltlD lhd-LCons-ltl[of (kfilter P n (ltl xs))]
    using 21 22 by fastforce
  show ?thesis
    using 18 23 by blast
qed
qed
qed

lemma kfilter-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-lnth-exist*:

assumes

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

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

using *assms lset-lfilter[of P xs]*

by (*metis (no-types, opaque-lifting) add.left-neutral diff-zero kfilter-llength kfilter-upperbound lfilter-kfilter zero-enat-def*)

lemma *ldistinct-kfilter*:

$\text{ldistinct}(\text{kfilter } P \ n \ xs)$

proof (*coinduction arbitrary: n xs*)

case (*ldistinct n1 xs1*)

then show *?case*

proof –

have 1: $\text{lhd } (\text{kfilter } P \ n1 \ xs1) \notin \text{lset } (\text{ltl } (\text{kfilter } P \ n1 \ xs1))$

proof –

have *f1*: $\text{kfilter } P \ n1 \ xs1 =$

$\text{LCons } (n1 + \text{lleast } P \ xs1)$

$(\text{lmap } \text{snd}$

$(\text{lfilter } (P \circ \text{fst})$

$(\text{lzip}$

$(\text{ldrop } (\text{enat } (\text{Suc } (\text{lleast } P \ xs1))) \ xs1)$

$(\text{iterates } \text{Suc } (\text{Suc } (n1 + \text{lleast } P \ xs1))))))$

by (*meson kfilter-ldropWhile ldistinct*)

then have $\text{lmap } \text{snd}$

$(\text{lfilter } (P \circ \text{fst})$

$(\text{lzip } (\text{ldrop } (\text{enat } (\text{Suc } (\text{lleast } P \ xs1))) \ xs1)$

$(\text{iterates } \text{Suc } (\text{Suc } (n1 + \text{lleast } P \ xs1)))))) =$

$\text{kfilter } P \ (\text{Suc } (n1 + \text{lleast } P \ xs1)) \ (\text{ldrop } (\text{enat } (\text{Suc } (\text{lleast } P \ xs1))) \ xs1)$

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: $((\exists n \ xs. \text{ltl } (\text{kfilter } P \ n1 \ xs1) = \text{kfilter } P \ n \ xs) \vee \text{ldistinct } (\text{ltl } (\text{kfilter } P \ n1 \ xs1)))$

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*:

$\text{llength}(\text{kfilter } P \ n \ (\text{ltake } k \ xs)) \leq \text{llength}(\text{kfilter } P \ n \ xs)$

by (*simp add: kfilter-llength lprefix-lfilterI lprefix-llength-le*)

lemma *kfilter-ldropn-lset*:

assumes $k < \text{llength } xs$

shows $\text{lset}(\text{kfilter } P \ n \ (\text{ldropn } k \ xs)) =$

$\{ n+i \mid i. i < \text{llength } xs - k \wedge P \ (\text{lnth } xs \ (k+i)) \}$

using *assms*

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

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

lemma *kfilter-llength-n-zero*:
shows $llength(kfilter\ P\ n\ xs) = llength(kfilter\ P\ 0\ xs)$
by (*simp add: kfilter-llength*)

lemma *kfilter-lnth-n-zero-a*:
assumes $k < llength\ (kfilter\ P\ n\ xs)$
shows $n \leq (lnth\ (kfilter\ P\ n\ xs)\ k)$
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 length-eq-0 lnth-0 not-gr-zero zero-enat-def)
next
case (Suc k)
then show ?case
  proof –
    have 1:  $\text{lnull}(\text{kfilter } P \ n \ xs) \implies \text{lnth } (\text{kfilter } P \ n \ xs) \ (\text{Suc } k) - n = \text{lnth } (\text{kfilter } P \ 0 \ xs) \ (\text{Suc } k)$ 
      using Suc.premis gr-implies-not-zero length-tnull by blast
    have 2:  $\neg \text{lnull}(\text{kfilter } P \ n \ xs) \implies \text{lnth } (\text{kfilter } P \ n \ xs) \ (\text{Suc } k) - n = \text{lnth } (\text{kfilter } P \ 0 \ xs) \ (\text{Suc } k)$ 
      proof –
        assume a0:  $\neg \text{lnull}(\text{kfilter } P \ n \ xs)$ 
        show  $\text{lnth } (\text{kfilter } P \ n \ xs) \ (\text{Suc } k) - n = \text{lnth } (\text{kfilter } P \ 0 \ xs) \ (\text{Suc } k)$ 
        proof –
          let ?Least = lleast P xs
          have 3:  $\exists \ x \ xs'. (\text{kfilter } P \ n \ xs) = \text{LCons } x \ xs'$ 
            using a0 kfilter-ldropWhile by blast
          obtain x xs' where 4:  $(\text{kfilter } P \ n \ xs) = \text{LCons } x \ xs'$ 
            using 3 by auto
          have 5:  $x = n + ?Least$ 
            using kfilter-ldropWhile[of P n xs]
            by (simp add: 4)
          have 6:  $\neg \text{lnull}(\text{kfilter } P \ n \ xs) \longleftrightarrow \neg \text{lnull}(\text{kfilter } P \ 0 \ xs)$ 
            by (simp add: kfilter-not-tnull-conv)
          have 7:  $\exists \ y \ ys'. (\text{kfilter } P \ 0 \ xs) = \text{LCons } y \ ys'$ 
            using 6 a0 kfilter-ldropWhile by blast
          obtain y ys' where 8:  $(\text{kfilter } P \ 0 \ xs) = \text{LCons } y \ ys'$ 
            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:  $xs' = \text{kfilter } P \ (\text{Suc } (n + ?Least)) \ (\text{ldrop } (\text{Suc } ?Least) \ xs)$ 
            using kfilter-ldropWhile[of P n xs]
            by (metis (no-types, lifting) 4 a0 kfilter-def ltl-simps(2))
          have 12:  $ys' = \text{kfilter } P \ (\text{Suc } (0 + ?Least)) \ (\text{ldrop } (\text{Suc } ?Least) \ xs)$ 
            using kfilter-ldropWhile[of P 0 xs]
            by (metis (no-types, lifting) 6 8 a0 kfilter-def ltl-simps(2))
          have 13:  $\text{lnth } (\text{kfilter } P \ n \ xs) \ (\text{Suc } k) - n = \text{lnth } xs' \ k - n$ 
            by (simp add: 4)
          have 14:  $\text{lnth } (\text{kfilter } P \ 0 \ xs) \ (\text{Suc } k) = \text{lnth } ys' \ k$ 
            by (simp add: 8)

```

```

have 15:  $\neg(\exists x \in \text{lset } (\text{ldrop } (\text{Suc } ?\text{Least}) \text{ xs}). P x) \implies$ 
   $\text{lnth } (\text{kfilter } P \ n \ \text{xs}) \ (\text{Suc } k) - n = \text{lnth } (\text{kfilter } P \ 0 \ \text{xs}) \ (\text{Suc } k)$ 
  using 8 Suc by auto
  (metis (full-types) gr-implies-not-zero kfilter-eq-conv kfilter-not-lnull-conv
    ldropsn-Suc-LCons ldropsn-Suc-conv-ldropsn ldropsn-eq-LConsD llength-LNil llist.disc(2))
have 16:  $\text{enat } k < \text{llength } (\text{kfilter } P \ (\text{Suc } (n+?\text{Least})) \ (\text{ldrop } (\text{Suc } ?\text{Least}) \ \text{xs}))$ 
  by (metis (no-types, lifting) 12 8 Extended-Nat.eSuc-mono Suc.premis eSuc-enat
    kfilter-llength llength-LCons)
have 17:  $(\exists x \in \text{lset } (\text{ldrop } (\text{Suc } ?\text{Least}) \ \text{xs}). P x) \implies$ 
   $\text{lnth } \text{xs}' \ k - (\text{Suc } (n+?\text{Least})) = \text{lnth } (\text{kfilter } P \ 0 \ (\text{ldrop } (\text{Suc } ?\text{Least}) \ \text{xs})) \ k$ 
  using Suc.hyps using 11 16 by blast
have 18:  $\text{enat } k < \text{llength } (\text{kfilter } P \ (\text{Suc } (?\text{Least})) \ (\text{ldrop } (\text{Suc } ?\text{Least}) \ \text{xs}))$ 
  by (metis 16 kfilter-llength)
have 19:  $(\exists x \in \text{lset } (\text{ldrop } (\text{Suc } ?\text{Least}) \ \text{xs}). P x) \implies$ 
   $\text{lnth } \text{ys}' \ k - (\text{Suc } (0+?\text{Least})) =$ 
   $\text{lnth } (\text{kfilter } P \ 0 \ (\text{ldrop } (\text{Suc } ?\text{Least}) \ \text{xs})) \ k$ 
  using Suc.hyps by (simp add: 12 18)
have 20:  $(\exists x \in \text{lset } (\text{ldrop } (\text{Suc } ?\text{Least}) \ \text{xs}). P x) \implies$ 
   $\text{lnth } (\text{kfilter } P \ n \ \text{xs}) \ (\text{Suc } k) - n = \text{lnth } \text{xs}' \ k - n$ 
  using 13 by blast
have 21:  $(\exists x \in \text{lset } (\text{ldrop } (\text{Suc } ?\text{Least}) \ \text{xs}). P x) \implies$ 
   $\text{lnth } \text{xs}' \ k - n = \text{lnth } (\text{kfilter } P \ 0 \ (\text{ldrop } (\text{Suc } ?\text{Least}) \ \text{xs})) \ k + (\text{Suc } (?\text{Least}))$ 
  using 11 16 17 kfilter-lowerbound by fastforce
have 22:  $(\exists x \in \text{lset } (\text{ldrop } (\text{Suc } ?\text{Least}) \ \text{xs}). P x) \implies$ 
   $\text{lnth } (\text{kfilter } P \ 0 \ (\text{ldrop } (\text{Suc } ?\text{Least}) \ \text{xs})) \ k + (\text{Suc } (?\text{Least})) = \text{lnth } \text{ys}' \ k$ 
  using 12 18 19 kfilter-lowerbound by fastforce
have 23:  $(\exists x \in \text{lset } (\text{ldrop } (\text{Suc } ?\text{Least}) \ \text{xs}). P x) \implies$ 
   $\text{lnth } (\text{kfilter } P \ n \ \text{xs}) \ (\text{Suc } k) - n = \text{lnth } (\text{kfilter } P \ 0 \ \text{xs}) \ (\text{Suc } k)$ 
  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 $(\text{kfilter } P \ n \ \text{xs}) = \text{lmap } (\lambda i. i+n) (\text{kfilter } P \ 0 \ \text{xs})$

proof –

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

by (simp add: kfilter-llength)

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

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

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: $\bigwedge k. k < llength(kfilter\ P\ 0\ xs) \longrightarrow$
 $lnth\ (kfilter\ P\ 0\ xs)\ k = lnth\ (lmap\ (\lambda i. i-n)\ (kfilter\ P\ n\ xs))\ k$
by (*simp add: kfilter-llength kfilter-lnth-n-zero*)
show ?thesis
using 1 2 *lset-eq-lnth-eq* **by** *blast*
qed

lemma *kfilter-holds*:
assumes $y \in lset(kfilter\ P\ n\ xs)$
shows $P\ (lnth\ xs\ (y-n))$
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 $y \in (\{i+n \mid i. i < llength\ xs\} - (lset\ (kfilter\ P\ n\ xs)))$
shows $\neg P\ (lnth\ xs\ (y-n))$
using *assms kfilter-lset[of P n xs] kfilter-lnull-conv[of P n xs]*
by *auto*

lemma *kfilter-holds-a*:
assumes $i < llength\ xs$
 $(i+n) \in lset(kfilter\ P\ n\ xs)$
shows $P\ (lnth\ xs\ i)$
using *assms kfilter-holds[of i+n P n xs]* **by** *simp*

lemma *kfilter-holds-not-a*:
assumes $i < llength\ xs$
 $P\ (lnth\ xs\ i)$
shows $(i+n) \in lset(kfilter\ P\ n\ xs)$
using *assms*
by (*simp add: in-kfilter-lset kfilter-lnull-conv*)

lemma *kfilter-holds-b*:
assumes $i < llength\ xs$
shows $(i+n) \in lset(kfilter\ P\ n\ xs) = P\ (lnth\ xs\ i)$
using *assms*
by (*meson kfilter-holds-a kfilter-holds-not-a*)

lemma *kfilter-holds-c*:
assumes $n \leq i$
 $i-n < llength\ xs$
shows $i \in lset(kfilter\ P\ n\ xs) = P\ (lnth\ xs\ (i-n))$
using *assms*
by (*metis diff-add idiff-enat-enat kfilter-holds kfilter-holds-not-a*)

lemma *kfilter-holds-not-b*:

assumes $n \leq i$

$i - n < \text{length } xs$

shows $i \notin \text{lset}(\text{kfilter } P \ n \ xs) = (\neg P \ (\text{lnth } xs \ (i-n)))$

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-length-def kfilter-upperbound length-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* $\equiv (\text{kfilter } P \ 0 \ xs)$ *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  $\forall ns\ n. \exists na. ((n::nat) \notin lset\ ns \vee lnth\ ns\ na = n) \wedge (n \notin lset\ ns \vee enat\ na < llength\ ns)$ 
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:  $i \notin lset\ (kfilter\ P\ 0\ xs) \wedge i < llength\ xs \longrightarrow \neg P\ (lnth\ xs\ i)$ 
by (simp add: 1 in-kfilter-lset)
have 4:  $i < llength\ xs$ 
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 < llength (kfilter P n (ldropn n xs) )
        n < llength xs
        n ≤ i
        i < lnth (kfilter P n (ldropn n xs) ) 0
shows   ¬ P (lnth xs i)
proof -
have 00: ¬ lnull (kfilter P n (ldropn n xs))
by (metis assms(1) ldrop-enat less-numeral-extra(3) llength-LNil llist.collapse(1))
have 0: lnth (kfilter P n (ldropn n xs) ) 0 < llength 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:  $i \in lset\ (kfilter\ P\ n\ (ldropn\ n\ xs)) \implies$ 
         $\neg lnull\ (kfilter\ P\ n\ (ldropn\ n\ xs)) \implies$ 
         $i < lnth\ (kfilter\ P\ n\ (ldropn\ n\ xs) )\ 0 \implies$ 
         $n < llength\ xs \implies$ 
         $n \leq i \implies$ 
        False
proof (induct zs≡(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:  $i \notin lset\ (kfilter\ P\ n\ (ldropn\ n\ xs)) \wedge n \leq i \wedge i < llength\ xs \longrightarrow \neg P\ (lnth\ xs\ i)$ 
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  $P \ n \ \text{ldropn } n \ xs$ ]
  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 assms(3) gr0I zero-enat-def by fastforce
  have 02:  $i \notin \text{lset}(kfilter \ P \ 0 \ xs)$ 
  by (metis 01 One-nat-def Suc-pred assms(2) assms(3) diff-less enat-ord-simps(2)
    in-lset-conv-lnth leD less-Suc-eq-le lidX-kfilter-gr-eq zero-less-one)
  from 0 01 02 show ?thesis
  by (metis add.right-neutral assms(4) 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 add: zero-enat-def)
  have 2:  $i \notin \text{lset}(kfilter \ P \ n \ (\text{ldropn } n \ xs))$ 
    using assms
    by (metis 1 Suc-diff-1 enat-ord-simps(2) in-lset-conv-lnth leD less-Suc-eq-le
      lidX-kfilter-gr-eq order-refl)
  from 0 1 2 show ?thesis
using assms kfilter-ldropn-lset-a[of  $n \ xs \ P \ n$ ] kfilter-lowerbound[of  $- \ P \ n \ (\text{ldropn } n \ xs)$ ]
  by auto (metis One-nat-def diff-less le-trans less-imp-le zero-less-one)
qed

```

```

lemma kfilter-not-between:
  assumes  $\text{lnth}(kfilter \ P \ 0 \ xs) \ (k) < i$ 
     $i < \text{lnth}(kfilter \ P \ 0 \ xs) \ (\text{Suc } k)$ 

```

```

      (Suc k) < llength (kfilter P 0 xs)
shows   ¬ P (lnth xs i)
proof -
have 0: ∃ x ∈ lset xs. P x
  using assms(3) gr-implies-not-zero kfilter-not-lnull-conv by fastforce
have 1: ¬ lnull (kfilter P 0 xs)
  by (simp add: 0 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  $k < \text{length}(\text{lfilter } P \text{ } xs)$ 
  shows  $\text{lidx } (k\text{filter } P \text{ } (\text{lnth } (k\text{filter } P \text{ } n \text{ } xs) \text{ } k)) (\text{ldropn } (\text{lnth } (k\text{filter } P \text{ } n \text{ } xs) \text{ } k) \text{ } xs))$ 
using assms using lidx-kfilter by blast

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

lemma ldropn-lset:
  assumes  $k < \text{length } xs$ 
  shows  $\text{lset } (\text{ldropn } k \text{ } xs) = \{(\text{lnth } xs \text{ } i) \mid i. k \leq i \wedge i < \text{length } xs\}$ 
using assms
proof (auto simp add: in-lset-conv-lnth )
  fix  $n :: \text{nat}$ 
  assume  $a1: \text{enat } n < \text{length } xs - \text{enat } k$ 
  assume  $\text{enat } k < \text{length } xs$ 
  then have  $\text{enat } k \leq \text{length } xs$ 
    by (meson dual-order.strict-implies-order)
  then have  $\text{enat } n + \text{enat } k < \text{length } xs$ 
    using  $a1$  by (meson enat-min)
  then show  $\exists na. \text{lnth } (\text{ldropn } k \text{ } xs) \text{ } n = \text{lnth } xs \text{ } na \wedge k \leq na \wedge \text{enat } na < \text{length } xs$ 
    using  $a1$  by (metis ldropn-ldropn le-add2 lhd-ldropn length-ldropn plus-enat-simps(1))
next
  fix  $i :: \text{nat}$ 
  assume  $a1: \text{enat } i < \text{length } xs$ 
  assume  $a2: k \leq i$ 
  have  $1: \text{enat } (i-k) < \text{length } xs - \text{enat } k$ 
    by (metis a1 a2 diff-add ldropn-Suc-conv-ldropn ldropn-ldropn length-ldropn llist.disc(2))
    lnull-ldropn not-le-imp-less)
  have  $2: \text{lnth } (\text{ldropn } k \text{ } xs) \text{ } (i-k) = \text{lnth } xs \text{ } i$ 
    by (simp add: a1 a2)
  show  $\exists n. \text{enat } n < \text{length } xs - \text{enat } k \wedge \text{lnth } (\text{ldropn } k \text{ } xs) \text{ } n = \text{lnth } xs \text{ } i$ 
    using  $1 \ 2$  by blast
qed

lemma lfilter-kfilter-ltake-lset-eq:
  assumes
     $k < \text{length}(\text{lfilter } P \text{ } xs)$ 
  shows  $\text{lset } (\text{ltake } k \text{ } (k\text{filter } P \text{ } 0 \text{ } xs)) =$ 
     $\text{lset } (k\text{filter } P \text{ } 0 \text{ } (\text{ltake } ((\text{lnth } (k\text{filter } P \text{ } 0 \text{ } xs) \text{ } k)) \text{ } xs))$ 
proof -
  have  $1: (\text{lnth } (k\text{filter } P \text{ } 0 \text{ } xs) \text{ } k) < \text{length } xs$ 
    using kfilter-length kfilter-upperbound
    by (metis assms gen-length-def kfilter-length kfilter-upperbound length-code )
  have  $2: \exists x \in \text{lset } xs. P \text{ } x$ 
    using assms gr-implies-not-zero length-eq-0 lnull-lfilter by blast

```

have 3: $\{i. i < \text{llength}(\text{ltake } ((\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k)) \ xs) \wedge$
 $P (\text{lnth } (\text{ltake } ((\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k)) \ xs) \ i)\} =$
 $\{i. i < (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \wedge P (\text{lnth } xs \ i)\}$
by (auto simp add: lnth-ltake 1 order-less-subst2)
have 5: $\{i. i < (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \wedge P (\text{lnth } xs \ i)\} =$
 $\{i. i < (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \wedge i \in \text{lset}(\text{kfilter } P \ 0 \ xs)\}$
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 lid_x-kfilter-gr kfilter-llength)
(metis kfilter-llength leD lid_x-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 ($\text{kfilter } P \ 0 \ xs$)] 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 $(\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs \ P (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k)$]

by linarith

have 5: $\{(l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) + i \mid i. \ i < l\text{length } xs - (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \wedge$
 $P \ (l\text{nth } xs \ (i + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k))) \} =$
 $\{(l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) + i \mid i. \ i < l\text{length } xs - (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \wedge$
 $(i + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k)) \in l\text{set}(k\text{filter } P \ 0 \ xs) \}$
using *kfilter-holds-b[of - xs 0 P]* **using** 1 2
by *auto*
 $(metis \text{ldropn-eq-LNil } \text{ldropn-ldropn } leD \ l\text{length-ldropn } not-le-imp-less,$
 $metis \text{gen-llength-def } in-lset-conv-lnth \ k\text{filter-upperbound } l\text{length-code})$
have 51: $\{(l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) + i \mid i. \ i < l\text{length } xs - (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \wedge$
 $(i + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k)) \in l\text{set}(k\text{filter } P \ 0 \ xs) \} =$
 $\{(l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) + i \mid i.$
 $(l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \leq i + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \wedge$
 $i + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) < l\text{length } xs \wedge$
 $(i + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k)) \in l\text{set}(k\text{filter } P \ 0 \ xs) \}$
by *auto*
 $(metis \ 1 \ \text{enat-min less-imp-le plus-enat-simps}(1),$
 $metis \text{ldropn-eq-LNil } \text{ldropn-ldropn } leD \ l\text{length-ldropn } not-le-imp-less)$
have 52: $\{(l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) + i \mid i.$
 $(l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \leq i + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \wedge$
 $i + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) < l\text{length } xs \wedge$
 $(i + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k)) \in l\text{set}(k\text{filter } P \ 0 \ xs) \} =$
 $\{j. (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \leq j \wedge j < l\text{length } xs \wedge j \in l\text{set}(k\text{filter } P \ 0 \ xs)\}$
by $(metis \ (no-types, \text{lifting}) \ \text{add.commute } le-iff-add)$
have 53: $\{j. (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \leq j \wedge j < l\text{length } xs \wedge j \in l\text{set}(k\text{filter } P \ 0 \ xs)\} =$
 $\{j. (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \leq j \wedge j < l\text{length } xs \wedge$
 $j \in \{ (l\text{nth } (k\text{filter } P \ 0 \ xs) \ jj) \mid jj. jj < l\text{length}(k\text{filter } P \ 0 \ xs)\}\}$
by $(simp \ \text{add: } l\text{set-conv-lnth})$
have 54: $\{j. (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \leq j \wedge j < l\text{length } xs \wedge$
 $j \in \{ (l\text{nth } (k\text{filter } P \ 0 \ xs) \ jj) \mid jj. jj < l\text{length}(k\text{filter } P \ 0 \ xs)\}\} =$
 $\{ (l\text{nth } (k\text{filter } P \ 0 \ xs) \ j) \mid j. k \leq j \wedge j < l\text{length } (k\text{filter } P \ 0 \ xs)\}$
by *auto*
 $(metis \ \text{assms } k\text{filter-llength } lid\text{x-kfilter-gr } not-less,$
 $\text{meson } lid\text{x-kfilter-gr-eq},$
 $metis \ \text{gen-llength-def } k\text{filter-upperbound } l\text{length-code})$
have 8: $k < l\text{length}(k\text{filter } P \ 0 \ xs)$
by $(simp \ \text{add: } \text{assms } k\text{filter-llength})$
have 9: $\{ (l\text{nth } (k\text{filter } P \ 0 \ xs) \ j) \mid j. k \leq j \wedge j < l\text{length } (k\text{filter } P \ 0 \ xs)\} =$
 $l\text{set}(\text{ldropn } k \ (k\text{filter } P \ 0 \ xs))$
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 $k < l\text{length}(l\text{filter } P \ xs)$

shows $(ltake \ k \ (k\text{filter } P \ 0 \ xs)) =$

$(k\text{filter } P \ 0 \ (ltake \ ((l\text{nth } (k\text{filter } P \ 0 \ xs) \ k)) \ xs))$

using *assms*

by $(simp \ \text{add: } l\text{filter-kfilter-ltake-lidx-a } l\text{filter-kfilter-ltake-lset-eq } lid\text{x-kfilter } lid\text{x-lset-eq})$

lemma *kfilter-kfilter-ldropn*:
assumes $k < \text{llength}(\text{lfilter } P \text{ } xs)$
shows $(\text{ldropn } k (\text{kfilter } P \text{ } 0 \text{ } xs)) =$
 $(\text{kfilter } P (\text{lnth } (\text{kfilter } P \text{ } 0 \text{ } xs) \text{ } k) (\text{ldropn } (\text{lnth } (\text{kfilter } P \text{ } 0 \text{ } xs) \text{ } k) \text{ } xs))$
using *assms*
by (*simp add: lfilter-kfilter-ldropn-lidx-a lfilter-kfilter-ldropn-lidx-b*
lfilter-kfilter-ldropn-lset-eq lidx-lset-eq)

lemma *kfilter-lmap-lfilter*:
shows $\text{lmap } (\lambda n. (\text{lnth } xs \text{ } n)) (\text{kfilter } P \text{ } 0 \text{ } xs) = \text{lfilter } P \text{ } xs$
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 $k < \text{llength}(\text{lfilter } P \text{ } xs)$
shows $\text{ltake } k (\text{lfilter } P \text{ } xs) =$
 $(\text{lfilter } P (\text{ltake } (\text{lnth } (\text{kfilter } P \text{ } 0 \text{ } xs) \text{ } k) \text{ } xs))$
proof –
have 2: $\text{lmap } (\lambda n. (\text{lnth } xs \text{ } n)) (\text{kfilter } P \text{ } 0 \text{ } xs) = \text{lfilter } P \text{ } xs$
using *kfilter-lmap-lfilter* **by** *blast*
have 3: $\text{ltake } k (\text{lfilter } P \text{ } xs) =$
 $\text{ltake } k (\text{lmap } (\lambda n. (\text{lnth } xs \text{ } n)) (\text{kfilter } P \text{ } 0 \text{ } xs))$
by (*simp add: 2*)
have 4: $\text{ltake } k (\text{lmap } (\lambda n. (\text{lnth } xs \text{ } n)) (\text{kfilter } P \text{ } 0 \text{ } xs)) =$
 $\text{lmap } (\lambda s. \text{lnth } xs \text{ } s) (\text{ltake } k (\text{kfilter } P \text{ } 0 \text{ } xs))$
using *ltake-lmap* **by** *blast*
have 6: $(\text{lfilter } P (\text{ltake } (\text{lnth } (\text{kfilter } P \text{ } 0 \text{ } xs) \text{ } k) \text{ } xs)) =$
 $\text{lmap } (\lambda s. (\text{lnth } (\text{ltake } (\text{lnth } (\text{kfilter } P \text{ } 0 \text{ } xs) \text{ } k) \text{ } xs) \text{ } s))$
 $(\text{kfilter } P \text{ } 0 (\text{ltake } (\text{lnth } (\text{kfilter } P \text{ } 0 \text{ } xs) \text{ } k) \text{ } xs))$
by (*simp add: kfilter-lmap-lfilter*)
have 7: $\text{lmap } (\lambda s. \text{lnth } xs \text{ } s) (\text{ltake } k (\text{kfilter } P \text{ } 0 \text{ } xs)) =$
 $\text{lmap } (\lambda s. \text{lnth } xs \text{ } s) (\text{kfilter } P \text{ } 0 (\text{ltake } (\text{lnth } (\text{kfilter } P \text{ } 0 \text{ } xs) \text{ } k) \text{ } xs))$
by (*simp add: assms kfilter-kfilter-ltake*)
have 8: $\text{lmap } (\lambda s. \text{lnth } xs \text{ } s) (\text{kfilter } P \text{ } 0 (\text{ltake } (\text{lnth } (\text{kfilter } P \text{ } 0 \text{ } xs) \text{ } k) \text{ } xs)) =$
 $\text{lmap } (\lambda s. (\text{lnth } (\text{ltake } (\text{lnth } (\text{kfilter } P \text{ } 0 \text{ } xs) \text{ } k) \text{ } xs) \text{ } s))$
 $(\text{kfilter } P \text{ } 0 (\text{ltake } (\text{lnth } (\text{kfilter } P \text{ } 0 \text{ } xs) \text{ } k) \text{ } xs))$
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 $\text{lmap } (\lambda s. \text{lnth } xs \text{ } (s + (\text{lnth } (\text{kfilter } P \text{ } 0 \text{ } xs) \text{ } k))))$
 $(\text{kfilter } P \text{ } 0 (\text{ldropn } (\text{lnth } (\text{kfilter } P \text{ } 0 \text{ } xs) \text{ } k) \text{ } xs)) =$
 $\text{lmap } (\lambda s. \text{lnth } xs \text{ } s)$
 $(\text{kfilter } P (\text{lnth } (\text{kfilter } P \text{ } 0 \text{ } xs) \text{ } k) (\text{ldropn } (\text{lnth } (\text{kfilter } P \text{ } 0 \text{ } xs) \text{ } k) \text{ } xs))$
proof –

have 1: $l\text{length } (l\text{map } (\lambda s. l\text{nth } xs \ (s + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k))))$
 $(k\text{filter } P \ 0 \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) =$
 $l\text{length } (l\text{map } (\lambda s. l\text{nth } xs \ s)$
 $(k\text{filter } P \ (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))))$
by (*simp add: kfilter-llength*)
have 2: $\bigwedge i. i < l\text{length } (l\text{map } (\lambda s. l\text{nth } xs \ (s + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k))))$
 $(k\text{filter } P \ 0 \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \implies$
 $l\text{nth } (l\text{map } (\lambda s. l\text{nth } xs \ (s + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k))))$
 $(k\text{filter } P \ 0 \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \ i =$
 $l\text{nth } xs \ ((l\text{nth } (k\text{filter } P \ 0 \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \ i) +$
 $(l\text{nth } (k\text{filter } P \ 0 \ xs) \ k))$
by *simp*
have 3: $\bigwedge i. i < l\text{length } (l\text{map } (\lambda s. l\text{nth } xs \ s)$
 $(k\text{filter } P \ (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \implies$
 $l\text{nth } (l\text{map } (\lambda s. l\text{nth } xs \ s)$
 $(k\text{filter } P \ (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k)$
 $(l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \ i =$
 $l\text{nth } xs \ (l\text{nth } (k\text{filter } P \ (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k)$
 $(l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \ i)$
by *simp*
have 4: $\bigwedge i. i < l\text{length } (l\text{map } (\lambda s. l\text{nth } xs \ (s + (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k))))$
 $(k\text{filter } P \ 0 \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \implies$
 $l\text{nth } xs \ ((l\text{nth } (k\text{filter } P \ 0 \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \ i) +$
 $(l\text{nth } (k\text{filter } P \ 0 \ xs) \ k)) =$
 $l\text{nth } xs \ (l\text{nth } (k\text{filter } P \ (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k)$
 $(l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))) \ i)$
using *kfilter-lnth-n-zero*[*of - P (lnth (kfilter P 0 xs) k) (ldropn (lnth (kfilter P 0 xs) k) xs)*]
kfilter-lowerbound[*of - P (lnth (kfilter P 0 xs) k) (ldropn (lnth (kfilter P 0 xs) k) xs)*]
using 1 *diff-add* **by** *fastforce*
show *?thesis*
using *l\text{list-eq-lnth-eq}*[*of lmap (\lambda s. lnth xs (s + (lnth (kfilter P 0 xs) k)))*
 $(k\text{filter } P \ 0 \ (l\text{dropn } (l\text{nth } (k\text{filter } P \ 0 \ xs) \ k) \ xs))]$
using 1 2 3 4 **by** *presburger*
qed

lemma *lfilter-kfilter-ldropn:*

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

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

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

proof —

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

using *assms gr-implies-not-zero llength-eq-0 lnull-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: $(l\text{drop } k \ (l\text{map } (\lambda s. l\text{nth } xs \ s) \ (k\text{filter } P \ 0 \ xs))) =$

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

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

have 6: $(\text{lfilter } P (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)) =$
 $\text{lmap } (\lambda s. \text{lnth } (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs) \ s)$
 $(\text{kfilter } P \ 0 (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs))$
by (*simp add: kfilter-lmap-lfilter*)
have 61: $\bigwedge z. z \in \text{lset } ((\text{kfilter } P \ 0 (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)))$
 $\implies (\lambda s. \text{lnth } (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs) \ s) \ z =$
 $(\lambda s. \text{lnth } xs \ (s + (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k))) \ z$
using *assms in-lset-conv-lnth[of - ((kfilter P 0 (ldropn (lnth (kfilter P 0 xs) k) xs)))]*
by *simp*
 $(\text{metis add.commute add.left-neutral kfilter-upperbound ldropn-eq-LNil ldropn-ldropn leD}$
 $\text{lnth-ldropn not-le-imp-less zero-enat-def})$
have 7: $\text{lmap } (\lambda s. \text{lnth } (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs) \ s)$
 $(\text{kfilter } P \ 0 (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)) =$
 $\text{lmap } (\lambda s. \text{lnth } xs \ (s + (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k)))$
 $(\text{kfilter } P \ 0 (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs))$
using 61 **by** *auto*
have 8: $\text{lmap } (\lambda s. \text{lnth } xs \ (s + (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k)))$
 $(\text{kfilter } P \ 0 (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)) =$
 $\text{lmap } (\lambda s. \text{lnth } xs \ s)$
 $(\text{kfilter } P (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs))$
by (*simp add: kfilter-lmap-shift-ldropn*)
have 9: $\text{lmap } (\lambda s. \text{lnth } xs \ s)$
 $(\text{kfilter } P (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)) =$
 $\text{lmap } (\lambda s. \text{lnth } xs \ s) (\text{ldropn } k (\text{kfilter } P \ 0 \ xs))$
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*
 $\text{lnth-lmap zero-enat-def}$)
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: $(\forall j < \text{llength } xs. j \neq (\text{lnth } (kfilter\ P\ 0\ xs)\ 0) \longrightarrow \neg P\ (\text{lnth } xs\ j))$
 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 $i :: \text{nat}$

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

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

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

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

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

```

      lnth (kfilter P 0 xs) k - 0 = lnth (kfilter P 0 xs) k
    using kfilter-mono[of - P 0 xs] by auto
  have g4:  $\bigwedge n. \text{enat } n < \text{llength } (\text{lfilter } P \text{ xs}) \implies P (\text{lnth } (\text{lfilter } P \text{ xs}) n)$ 
    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  $\neg 1 < \text{gen-llength } 0 (\text{lfilter } P \text{ xs}) \vee \text{enat } 0 < \text{gen-llength } 0 (\text{lfilter } P \text{ xs})$ 
    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 $\text{llength}(\text{lfilter } P \text{ xs}) = 1$

proof –

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

by (metis assms exist-one-lfilter-llength-one)

have 2: $0 < \text{llength}(\text{lfilter } P \text{ xs})$

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 $(\exists k < \text{llength } xs. P (\text{lnth } xs \ k) \wedge$
 $(\forall j < \text{llength } xs. j \neq k \longrightarrow \neg P (\text{lnth } xs \ j))) \longleftrightarrow$
 $\text{llength}(\text{lfilter } P \text{ xs}) = 1$

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 $(\exists k < \text{llength } xs. P (\text{lnth } xs \ k) \wedge$
 $(\forall j < \text{llength } xs. j < k \vee k < j \longrightarrow \neg P (\text{lnth } xs \ j))) \longleftrightarrow$
 $\text{llength}(\text{lfilter } P \text{ xs}) = 1$

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 $(\exists k < \text{llength } xs. P (\text{lnth } xs \ k) \wedge$
 $(\forall j. j < k \longrightarrow \neg P (\text{lnth } xs \ j)) \wedge$
 $(\forall j < \text{llength } xs. k < j \longrightarrow \neg P (\text{lnth } xs \ j))) \longleftrightarrow$
 $\text{llength}(\text{lfilter } P \text{ xs}) = 1$

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

lemma *kfilter-lappend-lfinite*:
 $\text{lfinite } xs \implies$
 $\text{kfilter } P \text{ } n (\text{lappend } xs \text{ } ys) =$
 $\text{lappend } (\text{kfilter } P \text{ } n \text{ } xs) (\text{kfilter } P \text{ } (n + (\text{the-enat}(\text{llength } xs))) \text{ } ys)$
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: $(\text{lzip } (\text{lappend } (\text{LCons } x \text{ } xs) \text{ } ys) (\text{iterates } \text{Suc } n)) =$
 $(\text{LCons } (x, n) (\text{lzip } (\text{lappend } xs \text{ } ys) (\text{iterates } \text{Suc } (\text{Suc } n))))$
by (*simp add: lzip.ctr(2)*)
have 3: $\text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) (\text{lzip } (\text{lappend } (\text{LCons } x \text{ } xs) \text{ } ys) (\text{iterates } \text{Suc } n))) =$
 $(\text{if } P \text{ } x \text{ then } (\text{LCons } n (\text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) (\text{lzip } (\text{lappend } xs \text{ } ys) (\text{iterates } \text{Suc } (\text{Suc } n))))))$
 $\text{else } \text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) (\text{lzip } (\text{lappend } xs \text{ } ys) (\text{iterates } \text{Suc } (\text{Suc } n))))))$
using 1 **by** *auto*
have 4: $(\text{if } P \text{ } x \text{ then } (\text{LCons } n (\text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) (\text{lzip } (\text{lappend } xs \text{ } ys) (\text{iterates } \text{Suc } (\text{Suc } n))))))$
 $\text{else } \text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) (\text{lzip } (\text{lappend } xs \text{ } ys) (\text{iterates } \text{Suc } (\text{Suc } n)))) =$
 $(\text{if } P \text{ } x \text{ then}$
 $(\text{LCons } n (\text{lappend } (\text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) (\text{lzip } xs (\text{iterates } \text{Suc } (\text{Suc } n))))))$
 $(\text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) (\text{lzip } ys (\text{iterates } \text{Suc } ((\text{Suc } n) + \text{the-enat } (\text{llength } xs))))))))$
 $\text{else } (\text{lappend } (\text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) (\text{lzip } xs (\text{iterates } \text{Suc } (\text{Suc } n))))))$
 $(\text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) (\text{lzip } ys (\text{iterates } \text{Suc } ((\text{Suc } n) + \text{the-enat } (\text{llength } xs))))))))$
using *lfinite-LConsI.hyps(2)* **by** *auto*
have 5: $(\text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) (\text{lzip } (\text{LCons } x \text{ } xs) (\text{iterates } \text{Suc } n)))) =$
 $(\text{if } P \text{ } x \text{ then } (\text{LCons } n (\text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) (\text{lzip } (xs) (\text{iterates } \text{Suc } (\text{Suc } n))))))$
 $\text{else } (\text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) (\text{lzip } (xs) (\text{iterates } \text{Suc } (\text{Suc } n))))))$
by (*simp add: lzip.ctr(2)*)
have 6: $(\text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) (\text{lzip } ys (\text{iterates } \text{Suc } (n + \text{the-enat } (\text{llength } (\text{LCons } x \text{ } xs))))))) =$
 $(\text{lmap } \text{snd } (\text{lfilter } (P \circ \text{fst}) (\text{lzip } ys (\text{iterates } \text{Suc } ((\text{Suc } n) + \text{the-enat } (\text{llength } xs))))))$

```

using eSuc-enat lfinite-LConsI.hyps(1) llength-eq-infty-conv-lfinite by force
have 7: lappend (lmap snd (lfilter (P ∘ fst) (lzip (LCons x xs) (iterates Suc n))))
  (lmap snd (lfilter (P ∘ fst) (lzip ys (iterates Suc (n + the-enat (llength (LCons x xs))))))) =
  lappend (if P x then (LCons n (lmap snd (lfilter (P ∘ fst) (lzip xs (iterates Suc (Suc n))))))
    else (lmap snd (lfilter (P ∘ fst) (lzip xs (iterates Suc (Suc n))))))
    (lmap snd (lfilter (P ∘ 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 < llength xs
shows kfilter P n (ltake k xs) = ltake (llength(kfilter P n (ltake k xs))) (kfilter P n xs)
proof –
have 1: kfilter P n xs = lappend (kfilter P n (ltake (the-enat k) xs))
  (kfilter P (n+ (the-enat k)) (ldropn (the-enat k) xs))
using kfilter-lappend-lfinite[of (ltake (the-enat k) xs) P n (ldropn (the-enat k) xs)]
by (metis assms enat-iless enat-ord-code(4) lappend-ltake-ldrop ldrop-enat lfinite-ltake
  llength-ltake min.strict-order-iff neq-iff the-enat.simps)
have 2: ltake (llength(kfilter P n (ltake k xs)))
  (lappend (kfilter P n (ltake (the-enat k) xs))
    (kfilter P (n+ (the-enat k)) (ldropn (the-enat k) xs))) =
  kfilter P n (ltake k xs)
by (metis assms enat-the-enat lfinite-ltake llength-eq-infty-conv-lfinite llength-ltake
  ltake-all ltake-lappend1 min.strict-order-iff order-refl)
show ?thesis using 1 2 by simp
qed

lemma lfilter-ldropn-llength:
assumes k < llength xs
shows llength (lfilter P (ldropn k xs)) ≤ llength (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 $\text{llength } (\text{lfilter } (\lambda x. P\ x \wedge Q\ x)\ xs) \leq \text{llength } (\text{lfilter } P\ xs)$
proof –
have 0: $\text{lfilter } (\lambda x. P\ x \wedge Q\ x)\ xs = \text{lfilter } (\lambda x. Q\ x \wedge P\ x)\ xs$
by *meson*
have 1: $\text{lfilter } (\lambda x. Q\ x \wedge P\ x)\ xs = \text{lfilter } Q\ (\text{lfilter } P\ xs)$
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-lfilter-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*

1.9 Transfer rules

context includes *lifting-syntax*
begin

lemma *lbutlast-transfer* [*transfer-rule*]:
 $(\text{llist-all2 } A \implies \text{llist-all2 } A) \text{ lbutlast lbutlast}$
by (*auto simp add: rel-fun-def lbutlast-conv-ltake llist-all2-llengthD llist-all2-ltakeI*)

lemma *lleast-transfer* [*transfer-rule*]:
 $((A \implies (=)) \implies \text{llist-all2 } A \implies (=)) \text{ lleast lleast}$
unfolding *lleast-def*[*abs-def*]
by (*auto simp add: rel-fun-def*)
 $(\text{metis } (\text{full-types}, \text{opaque-lifting}) \text{ llist-all2-conv-all-lnth})$

lemma *lfuse-transfer* [*transfer-rule*]:
 $(\text{llist-all2 } A \implies \text{llist-all2 } A \implies \text{llist-all2 } A) \text{ lfuse lfuse}$
by (*auto simp add: rel-fun-def intro: llist-all2-lfuseI*)

lemma *ridx-transfer* [*transfer-rule*]:
 $((R \implies R \implies (=)) \implies \text{llist-all2 } R \implies (=)) \text{ ridx ridx}$
by (*simp add: llist-all2-rsp rel-fun-def ridx-def llist-all2-conv-all-lnth*)
 $(\text{meson } \text{Suc-ile-eq order-less-imp-le})$

lemma *lsub-transfer* [*transfer-rule*]:
 $((=) \implies (=) \implies \text{llist-all2 } A \implies \text{llist-all2 } A) \text{ lsub lsub}$
by (*auto simp add: lsub-def rel-fun-def intro: llist-all2-ltakeI llist-all2-ldropI*)

lemma *lsubc-transfer* [*transfer-rule*]:
 $((=) \implies (=) \implies \text{llist-all2 } A \implies \text{llist-all2 } A) \text{ lsubc lsubc}$
by (*auto simp add: lsubc-def rel-fun-def min-def llist-all2-llengthD*
 $\text{intro: llist-all2-ltakeI llist-all2-ldropI}$)

```

lemma lfusecat-transfer [transfer-rule]:
  (llist-all2 (llist-all2 A) ==> llist-all2 A) lfusecat lfusecat
by(auto intro: llist-all2-lfusecatI)

```

end

end

2 Coinductive non-empty lists and their operations

Coinductive lists are formalised by Andreas Lochbihler in [3]. 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 \Rightarrow 'a nellist* and *NCons* of type *'a \Rightarrow 'a nellist \Rightarrow 'a nellist*.

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:  $\bigwedge x. \text{nlast0 } x = \text{nlast } x$ 
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]:  $\text{nlast } (NCons \ x \ \text{nell}) = \text{nlast } \text{nell}$ 
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]

```

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  $\Longrightarrow$  is-NNil (ntl nell)
by (cases nell) simp-all

```

```

lemma nlast-ntl [simp]:  $\text{nlast } (\text{ntl } \text{nell}) = \text{nlast } \text{nell}$ 
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  $\Longrightarrow$  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 *$x \in nset nell$*

and $\bigwedge nell. is-NNil\ nell \implies P\ (nlast\ nell)\ nell$

and $\bigwedge nell. \neg is-NNil\ nell \implies P\ (nhd\ nell)\ nell$

and $\bigwedge nell\ y. \llbracket \neg is-NNil\ nell; y \in nset\ (ntl\ nell); P\ y\ (ntl\ nell) \rrbracket \implies P\ y\ nell$

shows *$P\ x\ nell$*

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: $x \in nset nell$*

and *0: $\bigwedge nell. is-NNil\ nell \implies P\ (nlast\ nell)$*

and *1: $\bigwedge nell. P\ (NCons\ x\ nell)$*

and *2: $\bigwedge x'\ nell. \llbracket x \in nset\ nell; P\ nell \rrbracket \implies P\ (NCons\ x'\ nell)$*

shows *$P\ nell$*

using *major*

proof (*induct $y \equiv x$ nell rule: nellist-set-induct*)


```

case (findnil nell)
then show ?case using 0 1 2 by (metis nelllist.collapse(1) nelllist.map-disc-iff nelllist.map-sel(1))
next
case (find nell)
then show ?case by (metis 1 nelllist.collapse(2))
next
case (step nell)
then show ?case by (metis 2 nelllist.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:  $\bigwedge x' nell. \llbracket x \in \text{nset } nell; x \neq x'; P \text{ } nell \rrbracket \implies P (\text{NCons } x' \text{ } nell)$ 
  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

```

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 |
```

$$\text{NCons } x \text{ } nell' \Rightarrow \text{LCons } x \text{ } (\text{llist-of-nellist } nell'))$$

```

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 |
```

$$\text{LCons } x \text{ } ll' \Rightarrow \text{NCons } x \text{ } (\text{nellist-of-llist-a } ll'))$$

```

end

```

```

abbreviation nellist-of-llist == ( $\lambda ll. \text{nellist-of-llist-a } (\text{llast } ll) \text{ } (\text{lbutlast } ll)$ )

```

```

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]:
 $lnull\ ll \implies nlast\ (nellist-of-llist-a\ b\ ll) = b$
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*:
 $llist-of-nellist\ nell = LCons\ (nlast\ nell)\ LNil \longleftrightarrow is-NNil\ nell$
by (*simp add: nellist.case-eq-if llist-of-nellist.code*)

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

lemmas *llist-of-nellist-NNil* = *llist-of-nellist-simps(1)*
and *llist-of-nellist-NCons* = *llist-of-nellist-simps(2)*

declare *llist-of-nellist.sel* [simp *del*]

lemma *lhd-llist-of-nellist* [simp]:
 $\neg is-NNil\ nell \implies lhd\ (llist-of-nellist\ nell) = nhd\ nell$
by (*cases nell*) *simp-all*

lemma *lhd-llist-of-nellist1* [simp]:
 $is-NNil\ nell \implies lhd\ (llist-of-nellist\ nell) = nlast\ nell$
by (*cases nell*) *simp-all*

lemma *lhd-llist-of-nellist2* [simp]:
 $(case\ nell\ of\ (NNil\ b) \Rightarrow lhd\ LNil \mid (NCons\ b\ nell') \Rightarrow lhd\ (llist-of-nellist\ nell)) = nhd\ nell$
by (*cases nell*) (*simp-all add: lhd-LNil nhd-NNil*)

lemma *ltl-llist-of-nellist* [simp]:
 $\neg is-NNil\ nell \implies ltl\ (llist-of-nellist\ nell) = llist-of-nellist\ (ntl\ nell)$
by (*cases nell*) *simp-all*

lemma *ltl-llist-of-nellist1* [simp]:

is-NNil nell \implies *ltl (llist-of-nellist nell)* = *LNil*

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

lemma *ltl-llist-of-nellist2* [simp]:

(*case nell of* (*NNil b*) \implies (*LCons b LNil*) |

(*NCons b nell'*) \implies *ltl (llist-of-nellist nell)*) = *llist-of-nellist (ntl nell)*

by (*simp add: llist-of-nellist.code nellist.case-eq-if*)

lemma *nellist-of-llist-a-cong* [cong]:

assumes *ll = ll' lfinite ll'* \implies *b = b'*

shows *nellist-of-llist-a b ll* = *nellist-of-llist-a b' ll'*

proof(*unfold* $\langle ll = ll' \rangle$)

from *assms* **have** *lfinite ll'* \longrightarrow *b = b'* **by** *simp*

thus *nellist-of-llist-a b ll' = nellist-of-llist-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*) \implies *NCons x (NNil a)* |
(*NCons x nell'*) \implies *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 *lfinite (llist-of-nellist nell)*

lemma *nfinite-induct* [*consumes 1, case-names NNil NCons*]:

assumes *nfinite nell*

```

and  $\bigwedge y. P \ (NNil \ y)$ 
and  $\bigwedge x \text{ nell. } \llbracket nfinite \text{ nell}; P \text{ nell} \rrbracket \implies P \ (NCons \ x \text{ nell})$ 
shows  $P \text{ nell}$ 
using assms
unfolding nfinite-def
proof (induct ll  $\equiv$  llist-of-nellist nell arbitrary: nell rule: lfinite-induct)
case LNil
then show ?case by simp
next
case LCons
then show ?case by (metis lfinite.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:  $nfinite \text{ nell} \implies nfinite \ (NCons \ x \text{ nell})$ 
unfolding nfinite-def
by auto

```

```

declare nfinite-NNil [iff]

```

```

lemma is-NNil-imp-nfinite [simp]:
   $is-NNil \text{ nell} \implies nfinite \text{ nell}$ 
using lfinite.simps llist-of-nellist-eq-LNil by (auto simp add: nfinite-def )

```

```

lemma nfinite-NCons [simp]:
   $nfinite \ (NCons \ x \text{ nell}) = nfinite \text{ nell}$ 
by (simp add: nfinite-def)

```

```

lemma nfinite-ntl [simp]:
   $nfinite \ (ntl \text{ nell}) = nfinite \text{ nell}$ 
by (cases nell) simp-all

```

```

lemma nfinite-code [code]:
   $nfinite \ (NNil \ x) = True$ 
   $nfinite \ (NCons \ x \text{ nell}) = nfinite \text{ nell}$ 
by simp-all

```

```

lemma nfinite-imp-finite-nset:
assumes  $nfinite \text{ nell}$ 
shows  $finite \ (nset \text{ nell})$ 
using assms
by (induct nell rule: nfinite-induct) simp-all

```

```

lemma nfinite-snocn [simp]:
   $nfinite(snocn \text{ nell } a) \longleftrightarrow nfinite \text{ nell}$ 
(is ?lhs  $\longleftrightarrow$  ?rhs)

```

```

proof
  assume ?lhs thus ?rhs
proof (induct zs≡snocn nell a arbitrary: nell rule: nfinite-induct)
case (NNil y)
then show ?case
by (metis is-NNil-snocn nellist.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:
   $\neg \text{nfinite nell} \implies \text{snocn nell } a = \text{nell}$ 
proof (coinduction arbitrary: nell)
case (Eq-nellist nella)
then show ?case
  proof –
    have 1:  $\text{is-NNil (snocn nella } a) = \text{is-NNil nella}$ 
      using Eq-nellist by auto
    have 2:  $(\text{is-NNil (snocn nella } a) \longrightarrow \text{is-NNil nella} \longrightarrow \text{nlast (snocn nella } a) = \text{nlast nella})$ 
      by auto
    have 3:  $(\neg \text{is-NNil (snocn nella } a) \longrightarrow \neg \text{is-NNil nella} \longrightarrow$ 
       $\text{nhd (snocn nella } a) = \text{nhd nella} \wedge$ 
       $(\exists \text{nell. ntl (snocn nella } a) = \text{snocn nell } a \wedge \text{ntl nella} = \text{nell} \wedge \neg \text{nfinite nell}))$ 
      by (simp add: Eq-nellist nellist.case-eq-if)
    from 1 2 3 show ?thesis by blast
  qed
qed

lemma nfinite-nmap [simp]:
   $\text{nfinite (nmap f nell)} = \text{nfinite nell} \text{ (is ?lhs} \longleftrightarrow \text{?rhs)}$ 
proof
assume ?lhs thus ?rhs
  proof (induct zs≡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  $\langle ?lhs \rangle$ 
    unfolding  $\langle llx = lly \rangle$  by(induct) simp-all
qed

lemma nellist-of-llist-a-inverse-1:
assumes  $\neg$  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 $\neg \text{lnull } ll$
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 $\neg \text{lnull } ll$
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:
 lmap f (llist-of-nellist nell) = llist-of-nellist (nmap f 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 = $(\lambda ll \text{ nell. } \text{llist-of-nellist } nell = ll)$

lemma llist-of-nellist-not-lnull:
 $\neg (\text{lnull } (\text{llist-of-nellist } nell))$
by simp

lemma not-lnull-eq-lappend-lbutlast-llast:
 $\neg(\text{lnull } ll) \longleftrightarrow ll = \text{lappend } (\text{lbutlast } ll) \text{ (LCons (llast } ll) \text{ LNil)}$
using llist.collapse(1) **by** fastforce

lemma Domainp-help:
 $\neg \text{lnull } ll \Longrightarrow \exists \text{ nell. } \text{llist-of-nellist } nell = ll$
using nellist-of-llist-inverse **by** blast

lemma not-lnull-conv-llist-of-nellist:
 $\neg \text{lnull } ll \longleftrightarrow (\exists \text{ nell. } \text{llist-of-nellist } nell = ll)$
using Domainp-help llist-of-nellist-not-lnull **by** blast

lemma *Domainp-cr-nellist* [*transfer-domain-rule*]:

Domainp cr-nellist = ($\lambda ll. \neg(\text{lnull } ll)$)

unfolding *cr-nellist-def* *Domainp-iff*[*abs-def*]

using *Domainp-help* **by** *fastforce*

lemma *bi-unique-cr-nellist-help*:

llist-of-nellist nelly = *llist-of-nellist nellz* \implies *nelly* = *nellz*

by (*coinduction arbitrary: nelly nellz*)

(*metis llist-of-nellist-inverse-b*)

lemma *quotient-help-2*:

($\neg \text{lnull } (\text{llist-of-nellist } nell) \wedge \text{nellist-of-llist } (\text{llist-of-nellist } nell) = nell$)

by (*simp add: bi-unique-cr-nellist-help*)

lemma *quotient-help-nellist-1*:

cr-nellist ll nell \longrightarrow *nellist-of-llist ll* = *nell*

by (*metis cr-nellist-def llist-of-nellist-inverse-b*)

lemma *quotient-help-nellist-2*:

(*cr-nellist (llist-of-nellist nell) nell*)

by (*simp add: cr-nellist-def*)

lemma *quotient-help-3-nellist*:

($(\neg \text{lnull } llx \wedge llx = lly) =$
(*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 ($\lambda llx lly. \neg \text{lnull } llx \wedge llx = lly$)

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]:*
 $(A == => \text{pcr-nellist } A) (\lambda b. \text{LCons } b \text{ LNil}) \text{ NNil}$
by (*auto simp add: pcr-nellist-def OO-def cr-nellist-def rel-fun-def*)

lemma *NCons-transfer [transfer-rule]:*
 $(A == => \text{pcr-nellist } A == => \text{pcr-nellist } A) \text{ LCons } \text{NCons}$
by (*auto simp add: pcr-nellist-def OO-def cr-nellist-def rel-fun-def*)

lemma *nmap-transfer [transfer-rule]:*
 $((=) == => \text{pcr-nellist } (=) == => \text{pcr-nellist } (=)) \text{ lmap } \text{nmap}$
by (*auto simp add: cr-nellist-def nellist.pcr-cr-eq rel-fun-def lmap-llist-of-nellist*)

lemma *is-NNil-transfer [transfer-rule]:*
 $(\text{pcr-nellist } (=) == => (=)) \text{ is-lfirst } \text{is-NNil}$
by (*simp add: cr-nellist-def nellist.pcr-cr-eq rel-fun-def*)
 $(\text{metis lbutlast-simps}(2) \text{ llast-singleton llist.disc}(2) \text{ llist-of-nellist-eq-LNil}$
 $\text{ llist-of-nellist-inverse-a nellist-of-llist-inverse})$

lemma *is-NNil-nellist-of-llist-conv-is-lfirst:*
assumes $\neg \text{lnull } lx$
shows $\text{is-NNil}(\text{nellist-of-llist } lx) \longleftrightarrow \text{is-lfirst } lx$
using *assms*
by (*cases lx*)
 $(\text{simp, metis lbutlast.ctr}(1) \text{ lbutlast.disc-iff}(2) \text{ lbutlast-eq-LNil-conv llist.disc}(1)$
 $\text{ nellist-of-llist-a.disc-iff}(2))$

lemma *nfirst-transfer-a [transfer-rule]:*
 $(\text{pcr-nellist } (=) == => (=)) \text{ lhd } \text{nfirst}$
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]:*
 $(\text{pcr-nellist } (=) == => (=)) (\lambda ll. \text{if is-lfirst } ll \text{ then lhd } \text{LNil} \text{ else lhd } ll) \text{ nhd}$
by (*simp add: cr-nellist-def nellist.pcr-cr-eq rel-fun-def OO-def*)
 $(\text{metis lbutlast-simps}(2) \text{ lhd-llist-of-nellist llist-of-nellist-eq-LNil nhd-nellist-of-llist-a}$
 $\text{ quotient-help-2})$

lemma *ntl-transfer [transfer-rule]:*
 $(\text{pcr-nellist } A == => \text{pcr-nellist } A) (\lambda ll. \text{if is-lfirst } ll \text{ then } ll \text{ else ltl } ll) \text{ ntl}$
proof (*auto simp add: pcr-nellist-def cr-nellist-def OO-def rel-fun-def intro!: llist-all2-ltlI*
 $\text{ lfinite-LConsI dest: llist-all2-lnullD})$
show $\bigwedge b y. \text{ llist-all2 } A (\text{LCons } b \text{ LNil}) (\text{ llist-of-nellist } y) \implies$
 $\text{ llist-all2 } A (\text{LCons } b \text{ LNil}) (\text{ llist-of-nellist } (\text{ntl } y))$
using *llist-all2-ltlI*
by (*metis eq-LConsD is-NNil-ntl llist-all2-LNil1 llist-of-nellist.code llist-of-nellist.simps}(3)*
 $\text{ llist-of-nellist-eq-LNil ltl-llist-of-nellist nlast-ntl})$
next

show $\bigwedge x y. \forall b. x \neq LCons\ b\ LNil \implies llist-all2\ A\ x\ (l\text{list-of-nellist}\ y) \implies$
 $l\text{list-all2}\ A\ (l\text{tl}\ x)\ (l\text{list-of-nellist}\ (n\text{tl}\ y))$
by (*metis lhd-LCons-ltl llist-all2-LNil2 llist-all2-lnullD llist-all2-ltlI*
 $l\text{list-of-nellist.disc-iff}\ l\text{tl-llist-of-nellist}\ l\text{tl-llist-of-nellist1}$)
qed

lemma *nfinite-transfer* [*transfer-rule*]:
 $(\text{pcr-nellist}\ (=) \implies (=))\ \text{lfinite}\ \text{nfinite}$
by (*auto simp add: nellist.pcr-cr-eq cr-nellist-def nfinite-def rel-fun-def*)

lemma *llist-of-nellist-transfer* [*transfer-rule*]:
 $(\text{pcr-nellist}\ (=) \implies (=))\ \text{id}\ l\text{list-of-nellist}$
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*]:
 $((=) \implies (=) \implies \text{pcr-nellist}\ (=))\ (\lambda b\ ll. \text{lappend}\ ll\ (LCons\ b\ LNil))\ \text{nellist-of-llist-a}$
by (*auto simp add: pcr-nellist-def cr-nellist-def OO-def rel-fun-def*)

lemma *nlast-nellist-of-llist-a-lfinite* [*simp*]:
 $\text{lfinite}\ ll \implies \text{nlast}\ (\text{nellist-of-llist-a}\ b\ ll) = b$
by (*induct rule: lfinite.induct*) *simp-all*

lemma *snocn-transfer* [*transfer-rule*]:
 $(\text{pcr-nellist}\ (A) \implies (A) \implies \text{pcr-nellist}\ (A))\ (\lambda ll\ a. \text{lappend}\ ll\ (LCons\ a\ LNil))\ \text{snocn}$
unfolding *rel-fun-def*
by (*auto simp add: pcr-nellist-def nellist.pcr-cr-eq cr-nellist-def OO-def*)
 $(\text{metis}\ LNil\text{-transfer}\ l\text{list.rel-intros}(2)\ l\text{list-all2-lappendI}\ l\text{list-of-nellist-inverse}$
 $\text{nellist-of-llist-a-inverse})$

lemma *nellist-all2-help-a*:
 $l\text{list-all2}\ P\ (\text{l\text{list-of-nellist}\ nella})\ (\text{l\text{list-of-nellist}\ nellb}) \implies \text{nellist-all2}\ P\ nella\ nellb$
by (*coinduction arbitrary: nella nellb*)
 $(\text{metis}\ \text{lhd-llist-of-nellist}\ l\text{list.disc}(1)\ l\text{list-all2-LCons-LCons}\ l\text{list-all2-LNil1}$
 $l\text{list-all2-LNil2}\ l\text{list-all2-lhdD}\ l\text{list-all2-ltlI}\ l\text{list-of-nellist.disc-iff}$
 $l\text{list-of-nellist-eq-LNil}\ l\text{tl-llist-of-nellist}\ l\text{tl-simps}(2))$

lemma *nellist-all2-help-b*:
 $\text{nellist-all2}\ P\ nella\ nellb \implies l\text{list-all2}\ P\ (\text{l\text{list-of-nellist}\ nella})\ (\text{l\text{list-of-nellist}\ nellb})$

proof (*coinduction arbitrary: nella nellb*)

case *LNil*

then show *?case*

by *simp*

next

case *LCons*

then show *?case*

by *auto*

$(\text{metis}\ l\text{list-of-nellist.simps}(2)\ \text{nellist.case-eq-if}\ \text{nellist.rel-sel},$
 $\text{metis}\ l\text{list-all2-LNil1}\ l\text{tl-llist-of-nellist}\ l\text{tl-llist-of-nellist1}\ \text{nellist.rel-sel})$

qed

lemma *nellist-all2-transfer* [*transfer-rule*]:
 $((=) ==> \text{pcr-nellist } (=) ==> \text{pcr-nellist } (=) ==> (=)) \text{ llist-all2 nellist-all2}$
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 $(\lambda \text{ llx lly. lappend llx lly})$
by *auto*

lemma *llist-all2-llist-of-nellist-1*:
assumes $\neg \text{lnull } y$
 $\text{llist-all2 } (\lambda x z. \text{llist-of-nellist } z = x) \text{ llist1 } y$
 $\text{llist-all2 } (\lambda x z. \text{llist-of-nellist } z = x) \text{ llist2 } y$
shows $\text{llist1} = \text{llist2}$
proof –
have 1: $\text{llist-all2 } (\lambda x z. \text{llist-of-nellist } z = x) \text{ llist1 } y =$
 $\text{llist-all2 } (=) \text{ llist1 } (\text{lmap } \text{llist-of-nellist } y)$
using *llist-all2-lmap2[of (=) llist1 llist-of-nellist y]*
using *llist-all2-mono* **by** *fastforce*
have 2: $\text{llist-all2 } (\lambda x z. \text{llist-of-nellist } z = x) \text{ llist2 } y =$
 $\text{llist-all2 } (=) \text{ llist2 } (\text{lmap } \text{llist-of-nellist } y)$
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 $\neg \text{lnull } y$
 $\text{llist-all2 } (\lambda z x. \text{llist-of-nellist } z = x) y \text{ llist1}$
 $\text{llist-all2 } (\lambda z x. \text{llist-of-nellist } z = x) y \text{ llist2}$
shows $\text{llist1} = \text{llist2}$
proof –
have 1: $\text{llist-all2 } (\lambda z x. \text{llist-of-nellist } z = x) y \text{ llist1} =$
 $\text{llist-all2 } (=) (\text{lmap } \text{llist-of-nellist } y) \text{ llist1}$
using *llist-all2-lmap1[of (=)]*
using *llist-all2-mono* **by** *fastforce*
have 2: $\text{llist-all2 } (\lambda z x. \text{llist-of-nellist } z = x) y \text{ llist2} =$
 $\text{llist-all2 } (=) (\text{lmap } \text{llist-of-nellist } y) \text{ llist2}$
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 nellist nellist \Rightarrow 'a nellist
is $(\lambda \text{ lxs. lconcat lxs})$

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 bool) \Rightarrow 'a nellist \Rightarrow 'a nellist
is $\lambda P ll.$ (if lnull(lfilter P ll) then ll else lfilter P ll)
by *auto*

lift-definition *lappendn* :: 'a llist \Rightarrow 'a nellist \Rightarrow 'a nellist
is *lappend*
by *auto*

lift-definition *nzip* :: 'a nellist \Rightarrow 'b nellist \Rightarrow ('a \times 'b) nellist
is ($\lambda llx lly.$ lzip llx lly)
by *auto*

lift-definition *niterates* :: ('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'a nellist
is ($\lambda f a.$ iterates f a)
by *auto*

lift-definition *ndistinct* :: 'a nellist \Rightarrow bool
is *ldistinct*
by *auto*

lift-definition *nnth* :: 'a nellist \Rightarrow nat \Rightarrow 'a
is $\lambda ll n.$ lnth ll (the-enat (min (enat n) ((epred (llength ll)))))
by *blast*

lift-definition *nlength* :: 'a nellist \Rightarrow enat
is $\lambda ll.$ epred(llength ll)
by *auto*

lift-definition *ndropn* :: nat \Rightarrow 'a nellist \Rightarrow 'a nellist
is $\lambda n ll.$ ldrops (the-enat (min (enat n) ((epred (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 $\lambda n ll.$ ltake (eSuc n) ll
by *auto*

lift-definition *ntaken* :: nat \Rightarrow 'a nellist \Rightarrow 'a nellist
is $\lambda n ll.$ ltake (Suc n) ll
using *enat-0-iff(2)* **by** *auto*

2.4 The nlast element *nlast*

lemma *nlast-not-nfinite*:
assumes $\neg nfinite\ nell$
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 lfinite ll then y else undefined)
by (*simp add: nfinite-def nlast-not-nfinite*)

lemma *nlast-transfer* [*transfer-rule*]:
 (*pcr-nellist (=) ==> (=)*) ($\lambda ll.$ *if lfinite ll then llast ll else undefined*) *nlast*
by (*auto simp add: cr-nellist-def pcr-nellist-def nlast-nellist-of-llist-a OO-def*
 dest: llist-all2-lfiniteD)
 (*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)*)

2.5 nset

lemma *lset-llist-of-nellist-1*:
assumes *nfinite nell*
shows *lset (lbutlast (llist-of-nellist nell)) \cup {nlast nell} = nset nell* (**is** *?lhs = ?rhs*)
proof(*intro set-eqI iffI*)
 fix *x*
 assume *x \in ?lhs*
 thus *x \in ?rhs*
 proof –
 have *1: nlast nell = x \implies x \in nset nell*
 using *assms nset-nlast* **by** *blast*
 have *2: nlast nell = x \implies x \in nset nell*
 unfolding *nlast0-nlast*[*symmetric*] **by** (*simp add: 1*)
 have *3: x \in lset (lbutlast (llist-of-nellist nell)) \implies x \in nset 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 ⟨x ∈ lset (lbutlast (llist-of-nellist nell)) ∪ {nlast nell}⟩ by blast
qed
next
fix x
assume x ∈ ?rhs
thus x ∈ ?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 ¬nfinite nell
shows lset (lbutlast (llist-of-nellist nell)) = nset nell (is ?lhs = ?rhs)
proof(intro set-eqI iffI)
  fix x
  assume x ∈ ?lhs
  thus x ∈ ?rhs
  proof -
    have 3: x ∈ lset (lbutlast (llist-of-nellist nell)) ⇒ x ∈ nset 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-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 3 ⟨x ∈ lset (lbutlast (llist-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(llist-of-nellist nell))  $\cup$  {nlast nell}
    else lset (lbutlast (llist-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(llist-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  $\cup$  {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* (=) \implies (=)) *lset nset*
by(*auto simp add: cr-nellist-def nellist.pcr-cr-eq*)
end

2.6 *nmap*

lemma *nmap-eq-NCons-conv*:
 $nmap\ f\ nellx = NCons\ y\ nelly \iff$
 $(\exists z\ nellz. nellx = NCons\ z\ nellz \wedge f\ z = y \wedge nmap\ f\ nellz = nelly)$
by(*cases nellx*) *simp-all*

lemma *NCons-eq-nmap-conv*:
 $NCons\ y\ nelly = nmap\ f\ nellx \iff$
 $(\exists z\ nellz. nellx = NCons\ z\ nellz \wedge f\ z = y \wedge nmap\ f\ nellz = nelly)$
by(*cases nellx*) *auto*

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*:
 $is-NNil(nappend\ nellx\ nelly) \iff False$
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 $x \in nset\ nell$

shows $nell = (NNil\ x) \vee (\exists\ ys.\ nell = nappend\ (NNil\ x)\ ys) \vee$
 $(\exists\ ys.\ nell = nappend\ ys\ (NNil\ x) \wedge nfinite\ ys \wedge x \notin nset\ ys) \vee$
 $(\exists\ ys\ zs.\ nell = nappend\ ys\ (NCons\ x\ zs) \wedge nfinite\ ys \wedge x \notin nset\ ys)$

using *assms*

by *transfer*

(auto,
metis eq-LConsD lhd-lappend llist.disc(1) llist.expand split-llist-first)

lemma *split-nellist:*

assumes $x \in nset\ nell$

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 $x \in nset\ nell$

proof –

```

have 1:  $nell = (NNil\ x) \implies x \in nset\ nell$ 
  by simp
have 2:  $(\exists\ ys.\ nell = nappend\ (NNil\ x)\ ys) \implies x \in nset\ nell$ 
  by auto
have 3:  $(\exists\ ys\ zs.\ nell = nappend\ ys\ (NCons\ x\ zs) \wedge nfinite\ ys) \implies x \in nset\ nell$ 
  by transfer auto
have 4:  $(\exists\ ys.\ nell = nappend\ ys\ (NNil\ x) \wedge nfinite\ ys) \implies x \in nset\ nell$ 
  by transfer auto
show ?thesis using 1 2 3 4 assms by blast
qed

```

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-lappend*)

lemma *nset-lappendn-lfinite* [*simp*]:

lfinite ll \implies nset (lappendn ll nell) = lset ll \cup nset nell

by *transfer auto*

lemma *nlenght-nappend* [*simp*]:

nlenght (nappend nellx nelly) = nlenght nellx + nlenght nelly + 1

by *transfer*

(*auto, metis co.enat.exhaust-sel epred-iadd1 iadd-Suc-right llenght-eq-0 plus-1-eSuc(2)*)

lemma *nfinite-nlenght-enat*:

assumes *nfinite nell*

shows $\exists\ n.\ nlenght\ nell = enat\ n$

using *assms*

by *transfer (metis epred-conv-minus idiff-enat-enat lfinite-llenght-enat one-enat-def)*

lemma *nlenght-eq-enat-nfiniteD*:

nlenght nell = enat n \implies nfinite nell

by *transfer (metis epred-Infty llenght-eq-enat-lfiniteD not-lfinite-llenght)*

lemma *nfinite-conv-nlenght-enat*:

nfinite nell $\longleftrightarrow (\exists\ n.\ nlenght\ nell = enat\ n)$

using *nfinite-nlenght-enat nlenght-eq-enat-nfiniteD* **by** *blast*

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)*

2.10 The nth element of a nonempty lazy list *nnth*

lemma *nnth-NNil [nitpick-simp]*:

$nnth\ (NNil\ b)\ n = b$

by *transfer simp*

lemma *nnth-NCons*:

$nnth\ (NCons\ x\ nell)\ n = (case\ n\ of\ 0 \Rightarrow x \mid Suc\ n' \Rightarrow nnth\ nell\ n')$

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*: $nnth\ (NCons\ x\ nell)\ (Suc\ n) = nnth\ nell\ n$

by *(simp-all add: nnth-NCons)*

lemma *lnth-llist-of-nellist [simp]*:

$lnth\ (llist-of-nellist\ nell)\ (the-enat\ (min\ (enat\ n)\ ((epred(llength\ (llist-of-nellist\ nell))))))$
 $= nnth\ nell\ n$

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 illess-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*:

$nnth\ (ntl\ nell)\ n = nnth\ nell\ (Suc\ n)$

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 illess-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*:

$(\exists\ x \in nset\ nell. P\ x) = (\exists\ n. n \leq nlength\ nell \wedge P\ (nnth\ nell\ n))$

by *(metis in-nset-conv-nnth)*

lemma *nset-conv-nnth*:

$nset\ nell = \{nnth\ nell\ n \mid n. enat\ n \leq nlength\ nell\}$

by *(auto simp add: in-nset-conv-nnth)*

lemma *nnth-nappend1*:

$enat\ n \leq nlength\ nellx \implies nnth\ (nappend\ nellx\ nelly)\ n = nnth\ nellx\ n$

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.premys Suc-ile-eq by auto
qed
qed

```

lemma *nnth-nappend2*:

```

   $\llbracket \text{nlength nellx} = \text{enat } k; k < n \rrbracket \implies \text{nnth } (\text{nappend nellx nelly}) \text{ } n = \text{nnth nelly } (n - \text{Suc } k)$ 
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*:

```

   $\text{nnth } (\text{nappend nellx nelly}) \text{ } n =$ 
  (if  $\text{enat } n \leq \text{nlength nellx}$  then  $\text{nnth nellx } n$  else  $\text{nnth nelly } (n - \text{Suc}(\text{the-enat}(\text{nlength nellx})))$ )
by (cases nlength nellx)
  (auto simp add: nnth-nappend1 nnth-nappend2)

```

lemma *nnth-nlast*:

```

   $\text{nfinite nell} \implies \text{nlast nell} = \text{nnth nell } (\text{the-enat } (\text{nlength nell}))$ 
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)

```

2.11 ntake

lemma *ntake-NNil* [simp, code, nitpick-simp]:

```

   $\text{ntake } n \text{ } (\text{NNil } b) = (\text{NNil } b)$ 
by transfer auto

```

lemma *ntake-0* [simp]:

```

   $\text{ntake } 0 \text{ } nell = (\text{NNil } (\text{nfirst nell}))$ 
by transfer (auto simp add: ltake.ctr(2))

```

lemma *ntake-Suc-NCons* [simp]:

```

   $\text{ntake } (\text{eSuc } n) \text{ } (\text{NCons } x \text{ } nell) = (\text{NCons } x \text{ } (\text{ntake } n \text{ } nell))$ 
by transfer auto

```

lemma *ntake-Suc*:

```

   $\text{ntake } (\text{eSuc } n) \text{ } nell =$ 
  (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 nellist.disc(1) nellist.disc(2) ntake-0 ntake-Suc-NCons)
    next
    case infinity
      then show ?thesis
      by (metis NCons eSuc-infinity i0-ne-infinity nellist.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  $\Longrightarrow$  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:

$n \neq 0 \implies \text{ntl}(\text{ntake } n \text{ nell}) = \text{ntake } (\text{epred } n) (\text{ntl } \text{nell})$

by (cases nell) (simp, metis eSuc-epred nellist.sel(5) ntake-Suc-NCons)

lemma ntl-ntake-0:

$\text{ntl}(\text{ntake } 0 \text{ nell}) = (\text{NNil } (\text{nfirst } \text{nell}))$

by simp

lemma ntake-ntl:

$\text{ntake } n (\text{ntl } \text{nell}) = \text{ntl}(\text{ntake } (\text{Suc } n) \text{ nell})$

by (simp add: enat-0-iff(1) ntl-ntake)

lemma nlength-ntake [simp]:

$\text{nlength } (\text{ntake } n \text{ nell}) = \min n (\text{nlength } \text{nell})$

by transfer simp

lemma ntake-nmap [simp]:

$\text{ntake } n (\text{nmap } f \text{ nell}) = \text{nmap } f (\text{ntake } n \text{ nell})$

by transfer simp

lemma ntake-ntake [simp]:

$\text{ntake } n (\text{ntake } m \text{ nell}) = \text{ntake } (\min n m) \text{ nell}$

by transfer simp

lemma nset-ntake:

$\text{nset } (\text{ntake } n \text{ nell}) \subseteq \text{nset } \text{nell}$

by transfer (simp add: lset-ltake)

lemma ntake-all:

$\text{nlength } \text{nell} \leq m \implies \text{ntake } m \text{ nell} = \text{nell}$

by transfer (auto, metis eSuc-epred eSuc-ile-mono llength-eq-0 ltake-all)

lemma nfinite-ntake [simp]:

$\text{nfinite } (\text{ntake } n \text{ nell}) \longleftrightarrow \text{nfinite } \text{nell} \vee n < \infty$

by transfer (metis Extended-Nat.eSuc-mono eSuc-infinity lfinite-ltake)

lemma ntake-nappend1:

$n \leq \text{nlength } \text{nellx} \implies \text{ntake } n (\text{nappend } \text{nellx } \text{nelly}) = \text{ntake } n \text{ nellx}$

by transfer (auto, metis eSuc-epred eSuc-ile-mono llength-eq-0 ltake-lappend1)

lemma ntake-nappend2:

assumes $\text{nlength } \text{nellx} < n$

shows $\text{ntake } n (\text{nappend } \text{nellx } \text{nelly}) = \text{nappend } \text{nellx } (\text{ntake } (n - \text{nlength } \text{nellx} - 1) \text{ nelly})$

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:  $\neg \text{lnull } llxa \wedge llxa = llxa$ 
    assume a1:  $\text{lfinite } llxa$ 
    assume a2:  $\text{epred } (\text{llength } llxa) < na$ 
    assume a3:  $\neg \text{lnull } llya \wedge llya = llya$ 
    show  $\neg \text{lnull } (\text{ltake } (\text{eSuc } na) (\text{lappend } llxa llya)) \wedge$ 
       $\text{ltake } (\text{eSuc } na) (\text{lappend } llxa llya) =$ 
       $\text{lappend } llxa (\text{ltake } (\text{eSuc } (na - \text{epred } (\text{llength } llxa) - 1)) llya)$ 
    proof -
    have 1:  $\neg \text{lnull } (\text{ltake } (\text{eSuc } na) (\text{lappend } llxa llya))$ 
      using a0 by force
    have 2:  $\text{ltake } (\text{eSuc } na) (\text{lappend } llxa llya) =$ 
       $\text{lappend } (\text{ltake } (\text{eSuc } na) llxa) (\text{ltake } ((\text{eSuc } na) - \text{llength } llxa) llya)$ 
      by (meson ltake-lappend)
    have 21:  $\text{llength } llxa \leq (\text{eSuc } na)$ 
      by (metis a0 a2 co.enat.exhaust-sel eSuc-ile-mono leD le-cases llength-eq-0)
    have 3:  $\text{lappend } (\text{ltake } (\text{eSuc } na) llxa) (\text{ltake } ((\text{eSuc } na) - \text{llength } llxa) llya) =$ 
       $\text{lappend } llxa (\text{ltake } (\text{eSuc } na - \text{llength } llxa) llya)$ 
      using ltake-lappend2[of llxa (eSuc na) llya] 21 2 by auto
    have 4:  $(\text{eSuc } na - \text{llength } llxa) = (\text{eSuc } (na - \text{epred } (\text{llength } llxa) - 1))$ 
      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-inf-conv-lfinite)
    have 5:  $(\text{ltake } (\text{eSuc } na - \text{llength } llxa) llya) =$ 
       $(\text{ltake } (\text{eSuc } (na - \text{epred } (\text{llength } llxa) - 1)) llya)$ 
      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:

$\llbracket \text{ntake } n \text{ nellx} = \text{ntake } n \text{ nelly}; m \leq n \rrbracket \implies \text{ntake } m \text{ nellx} = \text{ntake } m \text{ nelly}$
by (metis min.orderE ntake-ntake)

lemma ntake-nnth:

assumes $\text{enat } m \leq n$

```

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.prem
      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.prem 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.prem Suc-ile-eq eSuc by force
      qed
    qed
  qed

```

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*]:

$$(NNil\ x) = ntaken\ n\ nell \iff nell = (NNil\ x) \vee (n = 0 \wedge nfirst\ nell = x)$$

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 nelllist.exhaust-sel nelllist.sel(4) nelllist.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*:

$n \leq nlength\ nellx \implies ntaken\ n\ (nappend\ nellx\ nelly) = ntaken\ n\ nellx$

by *transfer*

(*simp add: zero-enat-def, metis eSuc-enat eSuc-epred eSuc-ile-mono llength-eq-0 ltake-lappend1*)

lemma *ntaken-nappend2*:

$nlength\ nellx < (enat\ n) \implies$

$ntaken\ n\ (nappend\ nellx\ nelly) = nappend\ nellx\ (ntaken\ (n - (the-enat(nlength\ nellx)) - 1)\ nelly)$

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*:

$\llbracket \text{ntaken } n \text{ nell}x = \text{ntaken } n \text{ nell}y; m \leq n \rrbracket \implies \text{ntaken } m \text{ nell}x = \text{ntaken } m \text{ nell}y$
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*

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*:

$((\text{llist-of-nellist} \circ \text{nconcat}) \text{ nells}) =$
 $((\text{lconcat} \circ (\text{lmap } \text{llist-of-nellist} \circ \text{llist-of-nellist})) \text{ nells})$

proof –

let $?n2l = (\text{lmap } \text{llist-of-nellist} \circ \text{llist-of-nellist})$

have 1: $\text{llist-of-nellist} \circ \text{nconcat} =$

$\text{llist-of-nellist} \circ \text{nellist-of-llist} \circ \text{lconcat} \circ ?n2l$

by (*simp add: nconcat-def2 rewriteL-comp-comp*)

have 2: $\neg \text{lnull}((\text{lconcat} \circ ?n2l) \text{ nells})$

using *not-null-lconcat by blast*

have 3: $(\text{llist-of-nellist} \circ \text{nellist-of-llist} \circ \text{lconcat} \circ ?n2l) \text{ nells} =$

$(\text{lconcat} \circ ?n2l) \text{ nells}$

using 2 *nellist-of-llist-inverse by simp*

show *?thesis*

by (*metis 1 3*)

qed

lemma *nmap-lmap-inverse*:

$\text{nmap } \text{nellist-of-llist } (\text{nellist-of-llist } (\text{lmap } \text{llist-of-nellist } (\text{llist-of-nellist } \text{nells}))) = \text{nells}$

proof –

let $?n2l = (\text{lmap } \text{llist-of-nellist} \circ \text{llist-of-nellist})$

```

have 1:  $\text{nmap } \text{nellist-of-llist } (\text{nellist-of-llist } (?n2l \text{ nells})) =$ 
 $\text{nmap } \text{nellist-of-llist } (\text{nmap } \text{llist-of-nellist } \text{nells})$ 
by (simp add: lmap-llist-of-nellist)
have 2:  $\text{nmap } \text{nellist-of-llist } (\text{nmap } \text{llist-of-nellist } \text{nells}) =$ 
 $\text{nmap } (\text{nellist-of-llist} \circ \text{llist-of-nellist}) \text{ nells}$ 
by (simp add: nellist.map-comp)
have 3:  $(\text{nellist-of-llist} \circ \text{llist-of-nellist}) = \text{id}$ 
by auto
show ?thesis
by (metis 1 2 3 comp-apply nellist.map-id)
qed

```

lemma *lmap-nmap-inverse:*

assumes $\neg \text{lnull } \text{nells}$

$\forall \text{nell} \in \text{lset } \text{nells}. \neg \text{lnull } \text{nell}$

shows $(\text{lmap } \text{llist-of-nellist } (\text{llist-of-nellist } (\text{nmap } \text{nellist-of-llist } (\text{nellist-of-llist } \text{nells})))) = \text{nells}$

proof –

let $?l2n = \text{nmap } \text{nellist-of-llist} \circ \text{nellist-of-llist}$

have 0: $(\text{nmap } \text{nellist-of-llist } (\text{nellist-of-llist } \text{nells})) =$
 $\text{nellist-of-llist } (\text{lmap } \text{nellist-of-llist } \text{nells})$

using *nmap-nellist-of-llist assms* **by** *simp*

have 00: $(\text{llist-of-nellist } (\text{nellist-of-llist } (\text{lmap } \text{nellist-of-llist } \text{nells}))) =$
 $(\text{lmap } \text{nellist-of-llist } \text{nells})$

using *assms* **by** *simp*

have 1: $(\text{lmap } \text{llist-of-nellist } (\text{llist-of-nellist } (?l2n \text{ nells}))) =$
 $(\text{lmap } \text{llist-of-nellist } ((\text{lmap } \text{nellist-of-llist } \text{nells}))))$

using 0 00 **by** *auto*

have 2: $(\text{lmap } \text{llist-of-nellist } ((\text{lmap } \text{nellist-of-llist } \text{nells})))) =$
 $(\text{lmap } (\text{llist-of-nellist} \circ \text{nellist-of-llist}) \text{ nells})$

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: $(\text{lmap } (\text{llist-of-nellist} \circ \text{nellist-of-llist}) \text{ nells}) = \text{nells}$

using 3 **by** *auto*

show *?thesis*

using 1 2 4 0 00 **by** *presburger*

qed

lemma *nconcat-expand:*

$\text{nconcat } \text{nells} =$

$(\text{if is-NNil } \text{nells} \text{ then } \text{nfirst } \text{nells} \text{ else } \text{nappend } (\text{nfirst } \text{nells}) (\text{nconcat } (\text{ntl } \text{nells})))$

by (*metis nconcat-NCons nconcat-NNil nellist.case-eq-if nellist.collapse(1) nellist.collapse(2)*
nfirst-def)

lemma *lmap-llist-of-nellist-nmap:*

$(\text{lmap } (\text{lmap } f) (\text{lmap } \text{llist-of-nellist } (\text{llist-of-nellist } \text{nells}))) =$
 $(\text{lmap } \text{llist-of-nellist } (\text{lmap } (\text{nmap } f) (\text{llist-of-nellist } \text{nells})))$

proof –

have 5: $(\text{lmap } (\text{lmap } f) (\text{lmap } \text{llist-of-nellist } (\text{llist-of-nellist } \text{nells}))) =$


```

      (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*:

$nconcat\ nells = (NNil\ x) \longleftrightarrow nells = (NNil\ (NNil\ x))$

by (metis is-NNil-def is-NNil-nappend nconcat-NNil nconcat-expand)

lemma *nhd-nconcat* [simp]:

$\llbracket \neg is-NNil\ nells; \neg is-NNil\ (nhd\ nells) \rrbracket \implies nhd\ (nconcat\ nells) = nhd\ (nhd\ nells)$

by (metis nconcat-NCons nellist.collapse(2) nhd-nappend)

lemma *ntl-nconcat* [simp]:

$\llbracket \neg is-NNil\ nells; \neg is-NNil\ (nhd\ nells) \rrbracket \implies$

$ntl (nconcat\ nells) = nappend\ (ntl\ (nhd\ nells))\ (nconcat\ (ntl\ nells))$
by (*metis nconcat-NCons nellist.collapse(2) ntl-nappend*)

lemma *nconcat-nappend [simp]*:

assumes *nfinite nells*

shows $nconcat\ (nappend\ nells\ nell1) = nappend\ (nconcat\ nells)\ (nconcat\ nell1)$

using *assms*

by (*induct rule:nfinite-induct*) (*simp-all add: nappend-assoc*)

lemma *nconcat-eq-NCons-conv*:

$nconcat\ nells = NCons\ x\ nell \longleftrightarrow$

$nells = (NNil\ (NCons\ x\ nell)) \vee$

$(\exists\ nells'.\ nells = (NCons\ (NNil\ x)\ nells') \wedge\ nell = nconcat\ nells') \vee$

$(\exists\ nell'\ nells''.\ nells = (NCons\ (NCons\ x\ nell')\ nells'') \wedge\ nell = nappend\ nell'\ (nconcat\ nells''))$

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 *nlength-nconcat*:

shows $nlength\ (nconcat\ nells) =$

$(case\ nells\ of\ (NNil\ nell) \Rightarrow nlength\ nell \mid$

$(NCons\ nell\ nell1) \Rightarrow eSuc(nlength\ nell) + nlength\ (nconcat\ nell1))$

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 *nlength-nconcat-nfinite-conv-sum*:

assumes *nfinite nells*

shows $nlength\ (nconcat\ nells) =$

$nlength\ nells + (\sum i = 0.. (the-enat\ (nlength\ nells)).\ nlength\ (nnth\ nells\ i))$
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: $nlength\ (nconcat\ (NCons\ nell\ nells)) = 1 + nlength\ nell + nlength\ (nconcat\ nells)$
by *simp*
have 2: $nlength\ (nconcat\ nells) =$
 $nlength\ nells + (\sum i = 0.. (the-enat\ (nlength\ nells)).\ nlength\ (nnth\ nells\ i))$
using *NCons.hyps(2)* **by** *blast*
have 3: $nlength\ (NCons\ x\ nells) = 1 + nlength\ nells$
by (*simp add: plus-1-eSuc(1)*)
have 4: $(\sum i = 0..the-enat\ (nlength\ (NCons\ nell\ nells)).\ nlength\ (nnth\ (NCons\ nell\ nells)\ i)) =$
 $nlength\ (nnth\ (NCons\ nell\ nells)\ 0) +$
 $(\sum i = 1..the-enat\ (nlength\ (NCons\ nell\ nells)).\ nlength\ (nnth\ (NCons\ nell\ nells)\ i))$
by (*simp add: sum.atLeast-Suc-atMost*)
have 5: $(\sum i = 1..the-enat\ (nlength\ (NCons\ nell\ nells)).\ nlength\ (nnth\ (NCons\ nell\ nells)\ i)) =$
 $(\sum i = 1..(Suc\ (the-enat\ (nlength\ (nells))))).\ nlength\ (nnth\ (NCons\ nell\ nells)\ i)$
using *NCons.hyps(1) eSuc-enat nfinite-nlength-enat* **by** *fastforce*
have 6: $(\sum i = 1..(Suc\ (the-enat\ (nlength\ (nells))))).\ nlength\ (nnth\ (NCons\ nell\ nells)\ i) =$
 $(\sum i = 0..(the-enat\ (nlength\ (nells))))).\ nlength\ (nnth\ (NCons\ nell\ nells)\ (Suc\ i))$
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 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: $(\sum i = 0..the-enat\ (nlength\ (NCons\ nell\ nells)).\ nlength\ (nnth\ (NCons\ nell\ nells)\ i)) =$
 $nlength\ (nnth\ (NCons\ nell\ nells)\ 0) +$

```

      (∑ i = 1..the-enat (nlength (NCons nell nells)). nlength (nnth (NCons nell nells) i))
    by (simp add: sum.atLeast-Suc-atMost)
  have 2: (∑ i = 1..the-enat (nlength (NCons nell nells)). nlength (nnth (NCons nell nells) i)) =
    (∑ 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: (∑ i = 1..(Suc (the-enat (nlength ( nells)))). nlength (nnth (NCons nell nells) i)) =
    (∑ i = 0..( (the-enat (nlength ( nells)))). nlength (nnth (NCons nell nells) (Suc i)))
    using sum.shift-bounds-cl-nat-ivl[of λ i. nlength (nnth (NCons nell nells) i) 0 1
      ( (the-enat (nlength ( nells)))]
    by simp
  have 4: (∑ i = 0..( (the-enat (nlength ( nells)))). nlength (nnth (NCons nell nells) (Suc i))) =
    (∑ i = 0..( (the-enat (nlength ( nells)))). nlength (nnth ( nells) ( i)))
    by auto
  have 5: (∑ i = 0.. (the-enat (nlength (NCons nell nells))). eSuc (nlength (nnth (NCons nell nells) i)))
=
    eSuc (nlength (nnth (NCons nell nells) 0)) +
    (∑ i = 1.. (the-enat (nlength (NCons nell nells))). eSuc (nlength (nnth (NCons nell nells) i)))
    by (simp add: sum.atLeast-Suc-atMost)
  have 6: (∑ i = 1.. (the-enat (nlength (NCons nell nells))). eSuc (nlength (nnth (NCons nell nells) i)))
=
    (∑ 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: (∑ i = 1.. (Suc (the-enat (nlength ( nells)))). eSuc (nlength (nnth (NCons nell nells) i))) =
    (∑ i = 0.. ( (the-enat (nlength ( nells)))). eSuc (nlength (nnth (NCons nell nells) (Suc i))))
    using sum.shift-bounds-cl-nat-ivl[of λ i. eSuc(nlength (nnth (NCons nell nells) i)) 0 1
      ( (the-enat (nlength ( nells)))]
    by simp
  have 8: (∑ i = 0.. ( (the-enat (nlength ( nells)))). eSuc (nlength (nnth (NCons nell nells) (Suc i)))) =
    (∑ i = 0.. ( (the-enat (nlength ( nells)))). eSuc (nlength (nnth ( nells) ( i))))
    by auto
  have 9: (∑ i = 0.. (the-enat (nlength (NCons nell nells))). eSuc (nlength (nnth (NCons nell nells) i)))
=
    ( eSuc(nlength nell) +
      (∑ i = 0.. ( (the-enat (nlength ( nells)))). eSuc (nlength (nnth ( nells) ( i)))))
    by (metis 5 6 7 8 nnth-0)
  have 10: epred(∑ i = 0.. (the-enat (nlength (NCons nell nells))). eSuc (nlength (nnth (NCons nell nells)
i))) =
    epred( eSuc(nlength nell) +
      (∑ i = 0.. ( (the-enat (nlength ( nells)))). eSuc (nlength (nnth ( nells) ( i)))))
    using 9 by presburger
  have 11: epred( eSuc(nlength nell) +
    (∑ i = 0.. ( (the-enat (nlength ( nells)))). eSuc (nlength (nnth ( nells) ( i))))) =
    eSuc(nlength nell) +
    epred(∑ 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 $\forall xs \in \text{nset } \text{nells}. \text{nfinite } xs$
shows $\text{nset } (\text{nconcat } \text{nells}) = (\bigcup xs \in \text{nset } \text{nells}. \text{nset } xs)$
proof –
let $?n2l = (\text{lmap } \text{llist-of-nellist} \circ \text{llist-of-nellist})$
have 1: $\text{nset } (\text{nconcat } \text{nells}) =$
 $\text{nset } (((\text{nellist-of-llist} \circ \text{lconcat} \circ ?n2l) \text{ nells}))$
by (*simp add: nconcat-def2*)
have 2: $\text{nset } (((\text{nellist-of-llist} \circ \text{lconcat} \circ ?n2l) \text{ nells})) =$
 $\text{lset } (\text{llist-of-nellist } (((\text{nellist-of-llist} \circ \text{lconcat} \circ ?n2l) \text{ nells})))$
by (*metis lset-llist-of-nellist-a*)
have 3: $(\text{llist-of-nellist } (((\text{nellist-of-llist} \circ \text{lconcat} \circ ?n2l) \text{ nells}))) =$
 $((((\text{lconcat} \circ ?n2l) \text{ nells})))$
using *nellist-of-llist-inverse not-null-lconcat* **by** *fastforce*
have 4: $\text{lset } (\text{llist-of-nellist } (((\text{nellist-of-llist} \circ \text{lconcat} \circ ?n2l) \text{ nells}))) =$
 $\text{lset } ((((\text{lconcat} \circ ?n2l) \text{ nells})))$
using 3 **by** *presburger*
have 5: $\forall \text{nell} \in \text{lset } (?n2l \text{ nells}). \text{lfinite } \text{nell}$
using *assms nfinite-def* **by** *fastforce*
have 6: $\text{lset } ((((\text{lconcat} \circ ?n2l) \text{ nells}))) = (\bigcup \text{nell} \in \text{lset } (?n2l \text{ nells}). \text{lset } \text{nell})$
by (*simp add: 5 lset-lconcat-lfinite*)
have 7: $(\bigcup \text{nell} \in \text{lset } (?n2l \text{ nells}). \text{lset } \text{nell}) = \bigcup (\text{nset } ' \text{nset } \text{nells})$
by *simp*
show *?thesis*
by (*metis 6 7 lset-llist-of-nellist-a nconcat-def3 o-apply*)
qed

lemma *nconcat-ntake*:
shows $\text{nconcat } (\text{ntake } (\text{enat } n) \text{ nells}) =$
 $\text{ntake } (n + (\sum i=0..n. \text{nlength } (\text{nnth } \text{nells } i))) (\text{nconcat } \text{nells})$
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]*)
 $(\text{metis dual-order.refl nlast-NNil nnth-0 ntake-all ntake-nappend1 ntaken-0 ntaken-nlast})$
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 + ($\sum i = 0..n. nlength (nnth nells1 i)$)) (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 + ($\sum i = 0..n. nlength (nnth nells1 i)$)) (nconcat nells1)) =
ntake (nlength nell + 1 + (enat n + ($\sum i = 0..n. nlength (nnth nells1 i)$))) (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: *($\sum i = 0..Suc n. nlength (nnth nells i)$) =
nlength (nnth nells 0) + ($\sum i = 1..Suc n. nlength (nnth nells i)$)*

by (*simp add: sum.atLeast-Suc-atMost*)

have 6: *nlength (nnth nells 0) = nlength nell*

by (*simp add: NCons*)

have 7: *($\sum i = 1..Suc n. nlength (nnth nells i)$) =
($\sum i = 0..n. nlength (nnth nells (Suc i))$)*

using *sum.shift-bounds-cl-nat-ivl[of $\lambda i. nlength (nnth nells i)$ 0 1 n]*

by *simp*

have 8: *($\sum i = 0..n. nlength (nnth nells (Suc i))$) =
($\sum i = 0..n. nlength (nnth nells1 (i))$)*

using *NCons* **by** *auto*

have 9: *nlength nell + 1 + (enat n + ($\sum i = 0..n. nlength (nnth nells1 i)$)) =
(enat (Suc n) + ($\sum i = 0..Suc n. nlength (nnth nells i)$))*

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 *enat n \leq nlength (nconcat nells)*

shows $\exists m n'. nnth (nconcat nells) n = nnth (nnth nells m) n' \wedge enat n' \leq nlength (nnth nells m) \wedge$

proof –

$$\text{let } ?n2l = (\text{lmap } \text{llist-of-nellist} \circ \text{llist-of-nellist})$$

have 1: $\bigwedge nell. nlength\ nell = epred\ (llength\ (l\text{list-of-}nellist\ nell))$

unfolding *nlength-def* **by** *auto*

have 2: $\bigwedge nell\ j. j \leq nlength\ nell \longrightarrow nth\ nell\ j = lnth\ (l\text{list-of-nellist}\ nell)\ j$

unfolding *n*th-def by auto

have 3: $nlength\ (nconcat\ nells) =$

$$nlength \ ((nlist-of-llist \circ lconcat \circ ?n2l) \ nells)$$

by (*simp add: nconcat-def2*)

have \downarrow : $nlength \ ((nlist-of-llist \circ lconcat \circ ?n2l) \ nells) =$

```
epred (llength (llist-of-nellist (((nellist-of-llist o lconcat o ?n2l) nells))))
```

using *nlength.rep-eq* **by** *blast*

have 5: $(l\text{list-of-nellist } (((n\text{ellist-of-l\text{list}} \circ l\text{concat} \circ ?n2l) \text{ nell\text{s}})))) =$

$$((((lconcat \circ ?n2l) nells)))$$

using *nellist-of-llist-inverse not-null-lconcat* **by** *fastforce*

have 6: $epred\ (llength\ (l\text{list-of-nellist}\ (((nellist\text{-of-llist} \circ lconcat \circ ?n2l)\ nells)))) =$

$$\text{epred} (\text{length} (((\text{lconcat} \circ ?n2l) \text{ nells})))$$

using 5 by *presburger*

have γ : $enat\ n < (llength\ (((\ lconcat\ \circ\ ?n2l)\ nells)))$

by (*metis* (*no-types*, *lifting*) 3 4 5 *assms* *co.enat.collapse* *iless-Suc-eq* *llength-eq-0*

$$l\text{list-of-nellist-not-lnull})$$

have δ : (((*lconcat* \circ ?*n2l*) *nells*)) =

lconcat (?n2l nells)

by *simp*

have $g: \exists m \ n'$.

$$l nth \ (l concat \ (?n2l \ nells)) \ n = l nth \ (l nth \ (?n2l \ nells) \ m) \ n' \wedge$$
$$\text{enat } n' < \text{llength } (\text{lnth } (?n2l \text{ nells}) \ m) \wedge \text{enat } m < \text{llength } (?n2l \text{ nells}) \wedge$$
$$enat\ n = (\sum_{i < m. \text{length}(\text{lnth}(\text{?n2l nells})\ i)) + enat\ n'$$

using 7 *lnth-lconcat-conv*[of *n* (?*n2l nells*)]

by *fastforce*

obtain $m\ n'$ **where** 10: $lnth\ (lconcat\ (?n2l\ nells))\ n = lnth\ (lnth\ (?n2l\ nells)\ m)\ n' \wedge$

$$\text{enat } n' < \text{length } (\text{lnth } (?n2l \text{ nells}) \ m) \wedge \text{enat } m < \text{length } (?n2l \text{ nells}) \wedge$$
$$enat\ n = (\sum_{i < m}. llength\ (lnth\ (?n2l\ nells)\ i)) + enat\ n'$$

using 9 by *blast*

have 11: $l\text{nth } (l\text{concat } (?n2l \text{ nells})) \ n = n\text{nth } (n\text{concat } \text{nells}) \ n$

by (*metis 2 5 8 assms nconcat-def2*)

have 12: $lnth\ (lnth\ (?n2l\ nells)\ m)\ n' = nnth\ (nnth\ nells\ m)\ n'$

by (*metis 10 2 comp-apply co.enat.collapse illess-Suc-eq llength-eq-0 llength-lmap*

```
llist-of-nellist-not-lnull lnth-lmap nlength.rep-eq)
```

have 13: $length\ (l_{nth}\ (?\mathit{n2l}\ nells)\ m) = eSuc\ (nlength\ (n_{nth}\ nells\ m))$

by (*metis 10 2 comp-apply co.enat.collapse illess-Suc-eq llength-eq-0 llength-lmap*

```
llist-of-nellist-not-lnull lnth-lmap nlength.rep-eq)
```

have 14: $llength\ (?n2l\ nells) = eSuc(nlength\ nells)$

by (*metis comp-apply co.enat.exhaust-sel llength-eq-0 llength-llist-of-nellist llength-lmap*

$$llist-of-nellist-not-lnull)$$

have 15: $\bigwedge i. i < m \implies \text{length}(\text{lnth}(\text{?n2l nells}) i) = \text{eSuc}(\text{nlength}(\text{nnth nells } i))$

proof –

fix *i*

```

assume  $a: i < m$ 
show  $\text{length } (\text{lnth } (?n2l \text{ nells}) i) = e\text{Suc}(\text{nlength}(\text{nnth } \text{nells } i))$ 
proof –
  have 151:  $(\text{lnth } (?n2l \text{ nells}) i) = \text{llist-of-nellist } (\text{nnth } \text{nells } i)$ 
  using  $a \ 10 \ 14 \ 2$ 
  by ( $\text{metis comp-apply enat-ord-simps}(2) \text{ illess-Suc-eq llength-lmap lnth-lmap order-less-trans}$ )
  have 152:  $\text{llength } (\text{llist-of-nellist } (\text{nnth } \text{nells } i)) = e\text{Suc}(\text{nlength}(\text{nnth } \text{nells } i))$ 
    using  $\text{epred-inject}$  by  $\text{force}$ 
  show  $?thesis$  using 151 152 by  $\text{presburger}$ 
  qed
qed
have 16:  $(\sum i < m. \text{length } (\text{lnth } (?n2l \text{ nells}) i)) = (\sum i < m. e\text{Suc}(\text{nlength}(\text{nnth } \text{nells } i)))$ 
  by ( $\text{meson } 15 \text{ lessThan-iff sum.cong}$ )
show  $?thesis$ 
using 10 11 12 13 14 16 by ( $\text{metis illess-Suc-eq}$ )
qed

lemma  $\text{nnth-nconcat-ntake}$ :
  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  $\text{assms}$  by ( $\text{simp add: nconcat-ntake ntake-nnth}$ )

lemma  $\text{nfinite-nconcat [simp]}$ :
   $\text{nfinite } (\text{nconcat } \text{nells}) \longleftrightarrow \text{nfinite } \text{nells} \wedge (\forall \text{ nell} \in \text{nset } \text{nells}. \text{nfinite } \text{nell})$ 
  (is  $?lhs \longleftrightarrow ?rhs$ )
proof
  assume  $?lhs$ 
  thus  $?rhs$  (is  $?concl \text{ nells}$ )
  proof( $\text{induct nconcat nells arbitrary: nells rule: nfinite-induct}$ )
  case ( $\text{NNil } y$ )
  then show  $?case$ 
  by ( $\text{metis is-NNil-imp-nfinite nconcat-eq-NNil nellist.discI}(1) \text{ nellist.simps}(20) \text{ singleton-iff}$ )
  next
  case ( $\text{NCons } x \text{ nell}$ )
  then show  $?case$ 
    proof ( $\text{cases } \text{nells}$ )
    case ( $\text{NNil } \text{nell1}$ )
    then show  $?thesis$  using  $\text{NCons.hyps}$  by  $\text{auto}$ 
    next
    case ( $\text{NCons } \text{nell1 } \text{nells1}$ )
    then show  $?thesis$  using  $\text{NCons.hyps}$  by  $\text{simp}$ 
    ( $\text{metis nconcat-NCons nellist.sel}(5) \text{ nellist.set-intros}(3) \text{ nfinite-NCons nfinite-nappend ntl-nappend}$ )
    qed
  qed
next
  assume  $?rhs$ 
  then obtain  $\text{nfinite } ( \text{ nells})$ 
    and  $\forall \text{ nell} \in \text{nset } \text{nells}. \text{nfinite } \text{nell} \dots$ 
  thus  $?lhs$ 
  proof( $\text{induct nells rule: nfinite-induct}$ )

```



```

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 $(\forall \text{nell} \in \text{nset nells}. (\exists x \in \text{nset nell}. P x))$

shows $(\exists \text{nell} \in \text{nset} (\text{nconcat nells}). P \text{nell})$

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 $\forall \text{nell} \in \text{nset nells}. \text{nfinite nell}$

$\forall \text{nell} \in \text{nset nells}. (\exists x \in \text{nset nell}. P x)$

shows $\text{nfilter } P (\text{nconcat nells}) = \text{nconcat} (\text{nmap} (\text{nfilter } P) \text{ nells})$

proof –

let ?n2l = (*lmap llist-of-nellist* \circ *lmap llist-of-nellist*)

have 0: $\exists \text{nell} \in \text{nset} (\text{nconcat nells}). P \text{nell}$

using *assms nfilter-nconcat-nfinite-help* **by** *blast*

have 1: $\text{nset nells} = \text{lset}(\text{llist-of-nellist nells})$

by *simp*

have 2: $\bigwedge \text{nell}. \text{nell} \in \text{lset}(\text{llist-of-nellist nells}) \longrightarrow \text{lfinite} (\text{llist-of-nellist nell})$

using 1 *assms nfinite-def* **by** *blast*

have 3: $\bigwedge \text{nell } Q. (\exists x \in \text{nset nell}. Q x) \longrightarrow$

$\text{nfilter } Q \text{ nell} = \text{nlist-of-llist} (\text{lfilter } Q (\text{llist-of-nellist nell}))$

unfolding *nfilter-def* **by** *simp*

have 4: $\text{nfilter } P (\text{nconcat nells}) =$

$\text{nfilter } P (((\text{nlist-of-llist} \circ \text{lconcat} \circ ?\text{n2l}) \text{ nells}))$

by (*simp add: nconcat-def2*)

have 5: $\text{nfilter } P (((\text{nlist-of-llist} \circ \text{lconcat} \circ ?\text{n2l}) \text{ nells})) =$

$\text{nlist-of-llist} (\text{lfilter } P (\text{llist-of-nellist} (((\text{nlist-of-llist} \circ \text{lconcat} \circ ?\text{n2l}) \text{ nells}))))$

by (*metis 3 0 nconcat-def2*)

have 6: $(\text{llist-of-nellist} (((\text{nlist-of-llist} \circ \text{lconcat} \circ ?\text{n2l}) \text{ nells}))) =$

$((((\text{lconcat} \circ ?\text{n2l}) \text{ nells})))$

using *nlist-of-llist-inverse not-null-lconcat* **by** *fastforce*

have 7: $(\text{lfilter } P (\text{llist-of-nellist} (((\text{nlist-of-llist} \circ \text{lconcat} \circ ?\text{n2l}) \text{ nells})))) =$

$(\text{lfilter } P ((((\text{lconcat} \circ ?\text{n2l}) \text{ nells}))))$

using 6 **by** *presburger*

have 8: $(((\text{lconcat} \circ ?\text{n2l}) \text{ nells})) = \text{lconcat} (?\text{n2l} \text{ nells})$

```

  by simp
have 9:  $\forall \text{nell} \in \text{lset } (?n2l \text{ nells}). \text{lfinite nell}$ 
  by (simp add: 2)
have 10:  $(\text{lfilter } P \ ( ((( \text{lconcat} \circ ?n2l) \text{ nells} )))) = \text{lconcat} \ ( \text{lmap} \ (\text{lfilter } P) \ ( \ ?n2l \text{ nells} ) )$ 
  by (simp add: 9 lfilter-lconcat-lfinite)
have 11:  $\text{nconcat} \ ( \text{nmap} \ ( \text{nfilter } P) \text{ nells} ) =$ 
   $((\text{nellist-of-llist} \circ \text{lconcat} \circ ?n2l) \ ( \text{nmap} \ ( \text{nfilter } P) \text{ nells} ) )$ 
  by (simp add: nconcat-def2)
have 12:  $\bigwedge \text{nell } f. \text{nmap } f \text{ nell} = \text{nellist-of-llist} \ ( \text{lmap } f \ ( \text{llist-of-nellist nell} ) )$ 
  by (metis llist-of-nellist-inverse-b llist-of-nellist-not-lnull nmap-nellist-of-llist)
have 13:  $( \text{nmap} \ ( \text{nfilter } P) \text{ nells} ) = \text{nellist-of-llist} \ ( \text{lmap} \ ( \text{nfilter } P) \ ( \text{llist-of-nellist nells} ) )$ 
  using 12 by blast
have 14:  $\text{nellist-of-llist} \ ( \text{lmap} \ ( \text{nfilter } P) \ ( \text{llist-of-nellist nells} ) ) =$ 
 $\text{nellist-of-llist} \ ( \text{lmap} \ ( \lambda y s. \text{nellist-of-llist} \ ( \text{lfilter } P \ ( \text{llist-of-nellist } y s ) ) ) \ ( \text{llist-of-nellist nells} ) )$ 
  by (metis (mono-tags, lifting) 1 3 assms(2) llist.map-cong)
have 15:  $\text{lmap} \ ( \text{lfilter } P) \ ( \text{lmap} \ \text{llist-of-nellist} \ ( \text{llist-of-nellist nells} ) ) =$ 
 $\text{lmap} \ ( ( \text{lfilter } P ) \circ \text{llist-of-nellist} ) \ ( \text{llist-of-nellist nells} )$ 
  using llist.map-comp by blast
have 16:  $( \ ( ?n2l \ ( \text{nmap} \ ( \text{nfilter } P) \text{ nells} ) ) ) =$ 
 $( \ ( ?n2l \ ( \text{nellist-of-llist} \ ( \text{lmap} \ ( \text{nfilter } P) \ ( \text{llist-of-nellist nells} ) ) ) ) )$ 
  using 13 by presburger
have 17:  $( \ ( ?n2l \ ( \text{nellist-of-llist} \ ( \text{lmap} \ ( \text{nfilter } P) \ ( \text{llist-of-nellist nells} ) ) ) ) ) =$ 
 $( \ ( ?n2l \ ( \text{nellist-of-llist} \ ( \text{lmap} \ ( \lambda y s. \text{nellist-of-llist} \ ( \text{lfilter } P \ ( \text{llist-of-nellist } y s ) ) ) \ ( \text{llist-of-nellist nells} ) ) ) ) )$ 
  using 14 by presburger
have 18:  $( \ ( ?n2l \ ( \text{nellist-of-llist} \ ( \text{lmap} \ ( \lambda y s. \text{nellist-of-llist} \ ( \text{lfilter } P \ ( \text{llist-of-nellist } y s ) ) ) \ ( \text{llist-of-nellist nells} ) ) ) ) ) =$ 
 $( \ ( \text{lmap} \ \text{llist-of-nellist} \ ( \ ( \text{lmap} \ ( \lambda y s. \text{nellist-of-llist} \ ( \text{lfilter } P \ ( \text{llist-of-nellist } y s ) ) ) \ ( \text{llist-of-nellist nells} ) ) ) ) ) )$ 
  by simp
have 19:  $( \ ( \text{lmap} \ \text{llist-of-nellist} \ ( \ ( \text{lmap} \ ( \lambda y s. \text{nellist-of-llist} \ ( \text{lfilter } P \ ( \text{llist-of-nellist } y s ) ) ) \ ( \text{llist-of-nellist nells} ) ) ) ) ) =$ 
 $( \ ( \text{lmap} \ ( \text{llist-of-nellist} \circ ( \lambda y s. \text{nellist-of-llist} \ ( \text{lfilter } P \ ( \text{llist-of-nellist } y s ) ) ) ) \ ( \text{llist-of-nellist nells} ) ) )$ 
  using llist.map-comp by blast
have 20:  $\bigwedge \text{nell}. \text{nell} \in \text{lset} \ ( \text{llist-of-nellist nells} ) \longrightarrow \neg \text{lnull} \ ( \text{lfilter } P \ ( \text{llist-of-nellist nell} ) )$ 
  by (simp add: assms(2))
have 21:  $( \ ( \text{lmap} \ ( \text{llist-of-nellist} \circ ( \lambda y s. \text{nellist-of-llist} \ ( \text{lfilter } P \ ( \text{llist-of-nellist } y s ) ) ) ) ) \ ( \text{llist-of-nellist nells} ) ) )$ 
=
 $( \ ( \text{lmap} \ ( ( \lambda y s. ( \text{lfilter } P \ ( \text{llist-of-nellist } y s ) ) ) ) \ ( \text{llist-of-nellist nells} ) ) )$ 
  by (metis (no-types, lifting) 20 comp-def llist.map-cong0 nellist-of-llist-inverse)
have 22:  $( \ ( \text{lmap} \ \text{llist-of-nellist} \ ( \text{llist-of-nellist} \ ( \text{nmap} \ ( \text{nfilter } P) \text{ nells} ) ) ) ) =$ 
 $( \text{lmap} \ ( \text{lfilter } P) \ ( \ ( ( \text{lmap} \ \text{llist-of-nellist} \circ \text{llist-of-nellist} ) \text{ nells} ) ) )$ 
  using 13 14 15 19 21 by force
have 23:  $\text{nellist-of-llist} \ ( \text{lconcat} \ ( \text{lmap} \ ( \text{lfilter } P) \ ( \ ?n2l \text{ nells} ) ) ) =$ 
 $\text{nconcat} \ ( \text{nmap} \ ( \text{nfilter } P) \text{ nells} )$ 
  by (simp add: 22 nconcat-def2)
show ?thesis
  by (metis 10 23 5 6 nconcat-def2)
qed

```

lemma *nconcat-nmap-singleton* [simp]:

$nconcat (nmap (\lambda x. NNil (f x)) nell) = nmap f nell$

proof –

let $?n2l = (lmap \text{llist-of-nellist} \circ \text{llist-of-nellist})$

have 1: $nconcat (nmap (\lambda x. NNil (f x)) nell) =$

$((\text{llist-of-nellist} \circ lconcat \circ ?n2l) (nmap (\lambda x. NNil (f x)) nell))$

by (simp add: nconcat-def2)

have 2: $(nmap (\lambda x. NNil (f x)) nell) =$

$\text{nellist-of-llist} (lmap (\lambda x. NNil (f x)) (\text{llist-of-nellist nell}))$

by (simp add: lmap-llist-of-nellist)

have 3: $\text{nellist-of-llist} (lmap (\lambda x. NNil (f x)) (\text{llist-of-nellist nell})) =$

$\text{nellist-of-llist} (lmap (\lambda x. \text{nellist-of-llist} (LCons (f x) LNil)) (\text{llist-of-nellist nell}))$

by force

have 4: $nmap f nell = \text{nellist-of-llist} (lmap f (\text{llist-of-nellist nell}))$

by (simp add: lmap-llist-of-nellist)

have 5: $lconcat (?n2l (\text{nellist-of-llist}$

$(lmap (\lambda x. \text{nellist-of-llist} (LCons (f x) LNil)) (\text{llist-of-nellist nell}))) =$

$lconcat (lmap \text{llist-of-nellist} (lmap (\lambda z. NNil (f z)) (\text{llist-of-nellist nell})))$

by simp

have 6: $(lmap \text{llist-of-nellist} (lmap (\lambda z. NNil (f z)) (\text{llist-of-nellist nell}))) =$

$(lmap (\text{llist-of-nellist} \circ (\lambda z. NNil (f z))) (\text{llist-of-nellist nell}))$

using *llist.map-comp* **by** *metis*

have 7: $(lmap (\text{llist-of-nellist} \circ (\lambda z. NNil (f z))) (\text{llist-of-nellist nell})) =$

$(lmap (\lambda z. LCons (f z) LNil) (\text{llist-of-nellist nell}))$

by *auto*

have 8: $lconcat (?n2l (\text{nellist-of-llist}$

$(lmap (\lambda x. \text{nellist-of-llist} (LCons (f x) LNil)) (\text{llist-of-nellist nell})))$

$= (lmap f (\text{llist-of-nellist nell}))$

using 6 **by** force

have 9: $((\text{llist-of-nellist} \circ lconcat \circ ?n2l) (nmap (\lambda x. NNil (f x)) nell)) =$

$\text{nellist-of-llist} (lmap f (\text{llist-of-nellist nell}))$

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 $?n2l = (lmap \text{llist-of-nellist} \circ \text{llist-of-nellist})$

have 1: $nset (nconcat nells) =$

$nset (((\text{llist-of-nellist} \circ lconcat \circ ?n2l) nells))$

by (simp add: nconcat-def2)

have 2: $nset (((\text{llist-of-nellist} \circ lconcat \circ ?n2l) nells)) =$

$lset (\text{llist-of-nellist} (((\text{llist-of-nellist} \circ lconcat \circ ?n2l) nells)))$

by (*metis* *lset-llist-of-nellist-a*)

have 3: $(\text{llist-of-nellist} (((\text{llist-of-nellist} \circ lconcat \circ ?n2l) nells))) =$

$((((lconcat \circ ?n2l) nells)))$

using *nellist-of-llist-inverse* *not-null-lconcat* **by** *fastforce*

have 4: $lset (\text{llist-of-nellist} (((\text{llist-of-nellist} \circ lconcat \circ ?n2l) nells))) =$

```

      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*

$\bigwedge \text{nell}. \text{nell} \in \text{nset nells} \implies \text{ndistinct nell}$

$\bigwedge \text{nell nell1}. [\text{nell} \in \text{nset nells}; \text{nell1} \in \text{nset nells}; \text{nell} \neq \text{nell1}] \implies \text{nset nell} \cap \text{nset nell1} = \{\}$

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: $\bigwedge \text{nell}. \text{nell} \in \text{lset (llist-of-nellist nells)} \implies \text{ldistinct (llist-of-nellist nell)}$

using *assms(2) ndistinct.rep-eq* **by** *auto*

have 3: $\bigwedge \text{nell nell1}. [\text{nell} \in \text{lset (llist-of-nellist nells)}; \text{nell1} \in \text{lset (llist-of-nellist nells)}; \text{nell} \neq \text{nell1}] \implies \text{lset (llist-of-nellist nell)} \cap \text{lset (llist-of-nellist nell1)} = \{\}$

using *assms(3) by simp*

have 4: *ndistinct (nconcat nells)* =

ndistinct (((nellist-of-llist ○ lconcat ○ ?n2l) nells))

by (*simp add: nconcat-def2*)

have 5: *ndistinct (((nellist-of-llist ○ lconcat ○ ?n2l) nells))* =

ldistinct (llist-of-nellist (((nellist-of-llist ○ lconcat ○ ?n2l) nells)))

using *ndistinct.rep-eq* **by** *blast*

have 6: *(llist-of-nellist (((nellist-of-llist ○ lconcat ○ ?n2l) nells)))* =
((((lconcat ○ ?n2l) nells)))

using *nellist-of-llist-inverse not-null-lconcat* **by** *fastforce*

have 7: *ldistinct (llist-of-nellist (((nellist-of-llist ○ lconcat ○ ?n2l) nells)))* =
ldistinct ((((lconcat ○ ?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: $\bigwedge \text{nell}. \text{nell} \in \text{lset (?n2l nells)} \implies \text{ldistinct nell}$

using 2 **by** *auto*

have 10: $\bigwedge \text{nell nell1}. [\text{nell} \in \text{lset (?n2l nells)};$

$\text{nell1} \in \text{lset (?n2l nells)}; \text{nell} \neq \text{nell1}] \implies$

$\text{lset nell} \cap \text{lset nell1} = \{\}$

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

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-all2 Q (NNil b) nell \longleftrightarrow $(\exists b'. \text{nell} = \text{NNil } b' \wedge Q \text{ } b \text{ } b')$

using *nellist.rel-cases* **by** *fastforce*

lemma *nellist-all2-NNil2*:

nellist-all2 Q nell (NNil b') \longleftrightarrow $(\exists b. \text{nell} = \text{NNil } b \wedge Q \text{ } b \text{ } b')$

using *nellist.rel-sel*

by (*metis is-NNil-def nellist-all2-NNil*)

lemma *nellist-all2-NCons1*:

nellist-all2 P (NCons x nell) nell' \longleftrightarrow
 $(\exists x' \text{nell''}. \text{nell}' = \text{NCons } x' \text{nell''} \wedge P \text{ } x \text{ } x' \wedge \text{nellist-all2 } P \text{ nell nell''})$

using *nellist.rel-sel*

by (*metis nellist.collapse*(2) *nellist.disc*(2) *nellist.sel*(3) *nellist.sel*(5))

lemma *nellist-all2-NCons2*:

nellist-all2 P nell' (NCons x nell) \longleftrightarrow
 $(\exists x' \text{nell''}. \text{nell}' = \text{NCons } x' \text{nell''} \wedge P \text{ } x' \text{ } x \wedge \text{nellist-all2 } P \text{ nell'' nell})$

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 nellx nelly*

and $\bigwedge \text{nellix nellly.}$

X nellix nellly \implies

$(\text{is-NNil nellix} = \text{is-NNil nellly}) \wedge$
 $(\text{is-NNil nellix} \longrightarrow \text{is-NNil nellly} \longrightarrow P (\text{nlast nellix}) (\text{nlast nellly})) \wedge$
 $(\neg \text{is-NNil nellix} \longrightarrow \neg \text{is-NNil nellly} \longrightarrow P (\text{nhd nellix}) (\text{nhd nellly})) \wedge$
 $(X (\text{ntl nellix}) (\text{ntl nellly}) \vee \text{nellist-all2 } P (\text{ntl nellix}) (\text{ntl nellly}))$

shows *nellist-all2 P nellx nelly*

using *assms*

nellist.rel-coinduct[*of* $(\lambda \text{nelx nely. } X \text{ nelx nely} \vee \text{nellist-all2 } P \text{ nelx nely}) \text{ nellx nelly } P$]

by (*metis nellist.rel-sel*)

lemma *nellist-all2-cases*[*consumes 1, case-names NNil NCons, cases pred*]:

assumes *nellist-all2 P nellx nelly*

obtains $(\text{NNil } b \text{ } b' \text{ where } \text{nellx} = \text{NNil } b \text{ nellly} = \text{NNil } b' \text{ } P \text{ } b \text{ } b')$

| $(\text{NCons } x \text{ nellx}' \text{ } y \text{ nellly}'$

where $\text{nellx} = \text{NCons } x \text{ nellx}' \text{ and } \text{nellly} = \text{NCons } y \text{ nellly}'$

and $P \text{ } x \text{ } y \text{ and } \text{nellist-all2 } P \text{ nellx}' \text{ nellly}'$

using *assms*

using *nellist.rel-cases* **by** *blast*

lemma *nellist-all2-nmap:*

nellist-all2 P (nmap f nellx) nelly \longleftrightarrow nellist-all2 ($\lambda x y. P (f x) y$) nellx nelly
using *nellist.rel-map(1)* **by** *blast*

lemma *nellist-all2-nmap2:*

nellist-all2 P nellx (nmap f nelly) \longleftrightarrow nellist-all2 ($\lambda x y. P x (f y)$) nellx nelly
using *nellist.rel-map(2)* **by** *blast*

lemma *nellist-all2-mono:*

$\llbracket \text{nellist-all2 } P \text{ nellx nelly; } \bigwedge x y. P x y \implies P' x y \rrbracket$
 $\implies \text{nellist-all2 } P' \text{ nellx nelly}$
using *nellist.rel-mono-strong* **by** *blast*

lemma *nellist-all2-nlengthD:*

nellist-all2 P nellx nelly \implies nlength nellx = nlength nelly
by(*transfer*)(*auto dest: llist-all2-llengthD*)

lemma *nellist-all2-nfiniteD:*

nellist-all2 P nellx nelly \implies nfinite nellx = nfinite nelly
by *transfer*
(*auto dest: llist-all2-lfiniteD*)

lemma *nellist-all2-nfinite1-nlastD:*

$\llbracket \text{nellist-all2 } P \text{ nellx nelly; nfinite nellx } \rrbracket \implies P (\text{nlast nellx}) (\text{nlast nelly})$
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:*

nellist-all2 P nellx nelly \implies llist-all2 P (llist-of-nellist nellx) (llist-of-nellist nelly)
by *transfer*
(*simp add: nellist-all2-help-b*)

lemma *nellist-all2-is-NNilD:*

nellist-all2 P nellx nelly \implies is-NNil nellx \longleftrightarrow 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-ntII:*

$\llbracket \text{nellist-all2 } P \text{ nellx nelly; } \neg \text{is-NNil nellx} \vee \neg \text{is-NNil nelly } \rrbracket \implies$

nellist-all2 P (*ntl nellx*) (*ntl nelly*)
by (*cases nellx*) (*auto simp add: nellist-all2-NNil1 nellist-all2-NCons1*)

lemma *nellist-all2-refl*:

nellist-all2 P *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*
lhist.set-intros(1))

lemma *nellist-all2-reflI*:

$\llbracket \bigwedge x. x \in \text{nset } nell \implies P \ x \ x; \text{nfinite } nell \implies P \ (\text{nlast } nell) \ (\text{nlast } nell) \rrbracket$
 $\implies \text{nellist-all2 } P \ nell \ nell$

by(*simp add: nellist-all2-refl*)

lemma *nellist-all2-conv-all-nnth-help1*:

$\neg \text{lnull } nellx \implies \neg \text{lnull } nelly \implies \text{lfinite } nelly \implies \text{lhist-all2 } P \ nellx \ nelly \implies$
 $P \ (\text{llast } nellx) \ (\text{llast } nelly)$

proof –

assume *a1*: *lfinite nelly*

assume *a2*: *lhist-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: lhist-all2-lfiniteD*)

have *f6*: *llength (ltl nellx) = epred (llength nelly)*

using *a2* **by** (*metis (no-types) epred-llength lhist-all2-llengthD*)

have *f7*: *lappend (lbutlast nelly) (LCons (llast nelly) LNil) = nelly*

using *a4* **by** (*meson lappend-lbutlast-llast-id*)

have *llength (lbutlast nellx) = llength (ltl nellx)*

using *epred-llength* **by** *auto*

then show *?thesis*

using *f7 f6 f5 a3 a2 a1*

by (*metis (no-types) lappend-eq-lappend-conv lappend-lbutlast-llast-id lappend-ltake-ldrop*
lbutlast-conv-ltake lfinite-lappend lhd-LCons lhist.disc(2) lhist-all2-lappend1D(2)
lhist-all2-lhdD2)

qed

lemma *nellist-all2-conv-all-nnth*:

nellist-all2 P *nellx nelly* \longleftrightarrow
 $\text{nlength } nellx = \text{nlength } nelly \wedge$
 $(\forall n. \text{enat } n \leq \text{nlength } nellx \longrightarrow P \ (\text{nnth } nellx \ n) \ (\text{nnth } nelly \ n))$

by *transfer*

(*auto simp add: lhist-all2-llengthD,*
metis eSuc-epred iless-Suc-eq llength-eq-0 lhist-all2-lnthD2,
metis eSuc-epred iless-Suc-eq llength-eq-0 lhist-all2-conv-all-lnth)

lemma *nellist-all2-True* [*simp*]:

nellist-all2 $(\lambda - . \text{True})$ *nellx nelly* $\longleftrightarrow \text{nlength } nellx = \text{nlength } nelly$

by(*simp add: nellist-all2-conv-all-nnth*)

lemma *nellist-all2-nnthD*:

$\llbracket \text{nellist-all2 } P \text{ nellx nelly; enat } n \leq \text{nlength nellx} \rrbracket \implies P (\text{nnth nellx } n) (\text{nnth nelly } n)$
by (*simp add: nellist-all2-conv-all-nnth*)

lemma *nellist-all2-nnthD2*:

$\llbracket \text{nellist-all2 } P \text{ nellx nelly; enat } n \leq \text{nlength nelly} \rrbracket \implies P (\text{nnth nellx } n) (\text{nnth nelly } n)$
by (*simp add: nellist-all2-conv-all-nnth*)

lemmas *nellist-all2-eq = nellist.rel-eq*

lemma *nmap-eq-nmap-conv-nellist-all2*:

$\text{nmap } f \text{ nellx} = \text{nmap } f' \text{ nelly} \longleftrightarrow$
 $\text{nellist-all2 } (\lambda x y. f x = f' y) \text{ nellx nelly}$
by *transfer*
(clarsimp simp add: lmap-eq-lmap-conv-llist-all2)

lemma *nellist-all2-trans*:

$\llbracket \text{nellist-all2 } P \text{ nellx nelly; nellist-all2 } P \text{ nelly nellz; transp } P \rrbracket$
 $\implies \text{nellist-all2 } P \text{ nellx nellz}$
by *transfer (auto elim: llist-all2-trans dest: llist-all2-lfiniteD transpD)*

lemma *nellist-all2-nappendI*:

$\llbracket \text{nellist-all2 } P \text{ nellx nelly;}$
 $\llbracket \text{nfinite nellx; nfinite nelly; } P (\text{nlast nellx}) (\text{nlast nelly}) \rrbracket$
 $\implies \text{nellist-all2 } P \text{ nellx' nelly' } \rrbracket$
 $\implies \text{nellist-all2 } P (\text{nappend nellx nellx'}) (\text{nappend nelly nelly'})$
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*:

$\text{nellist-all2 } (\text{nellist-all2 } A) \text{ nells nells1} =$
 $\text{llist-all2 } (\text{llist-all2 } A) (\text{lmap llist-of-nellist } (\text{llist-of-nellist nells}))$
 $(\text{lmap llist-of-nellist } (\text{llist-of-nellist nells1}))$

proof –

have 1: $\text{nellist-all2 } (\text{nellist-all2 } A) \text{ nells nells1} =$
 $\text{llist-all2 } (\text{nellist-all2 } A) (\text{llist-of-nellist nells}) (\text{llist-of-nellist nells1})$
using *nellist-all2-help-a nellist-all2-help-b* **by** *blast*
have 2: $(\lambda xx yy. \text{nellist-all2 } A \text{ xx yy}) =$
 $(\lambda xx yy. \text{llist-all2 } A (\text{llist-of-nellist xx}) (\text{llist-of-nellist yy}))$
by (*meson nellist-all2D-llist-all2-llist-of-nellist nellist-all2-help-a*)
have 3: $\text{llist-all2 } (\text{nellist-all2 } A) (\text{llist-of-nellist nells}) (\text{llist-of-nellist nells1}) =$
 $\text{llist-all2 } (\lambda xx yy. \text{llist-all2 } A (\text{llist-of-nellist xx}) (\text{llist-of-nellist yy}))$
 $(\text{llist-of-nellist nells}) (\text{llist-of-nellist nells1})$
using 2 **by** *auto*
have 6: $\text{llist-all2 } (\lambda xx yy. \text{llist-all2 } A (\text{llist-of-nellist xx}) (\text{llist-of-nellist yy}))$
 $(\text{llist-of-nellist nells}) (\text{llist-of-nellist nells1}) =$
 $\text{llist-all2 } (\text{llist-all2 } A) (\text{lmap llist-of-nellist } (\text{llist-of-nellist nells}))$


```

      (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)) ⇒
  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 *l1ist-all2-nellist-of-l1istI*:

nellist-all2 A nellx nelly \implies

l1ist-all2 A (lbutlast(l1ist-of-nellist nellx)) (lbutlast(l1ist-of-nellist nelly))

proof (*coinduction arbitrary: nellx nelly*)

case *LNil*

then show *?case*

by (*metis lbutlast.disc-iff(1) l1ist.disc(1) l1ist-of-nellist-inverse-a ltl-l1ist-of-nellist1*
nellist-all2-is-NNilD nellist-of-l1ist-a.disc(1))

next

case (*LCons nell1 nell2*)

then show *?case*

proof –

assume *a0: nellist-all2 A nell1 nell2*

assume *a1: \neg lnull (lbutlast (l1ist-of-nellist nell1))*

assume *a2: \neg lnull (lbutlast (l1ist-of-nellist nell2))*

have *1: A (lhd (lbutlast (l1ist-of-nellist nell1))) (lhd (lbutlast (l1ist-of-nellist nell2)))*

using *a0 a1 a2*

by (*metis lbutlast.disc(1) l1ist.disc(1) l1ist-of-nellist-inverse-a ltl-l1ist-of-nellist1*
nellist-all2-nhdD nhd-nellist-of-l1ist-a)

have *2: ((\exists nellx nelly.*

ltl (lbutlast (l1ist-of-nellist nell1)) = lbutlast (l1ist-of-nellist nellx) \wedge

ltl (lbutlast (l1ist-of-nellist nell2)) = lbutlast (l1ist-of-nellist nelly) \wedge

nellist-all2 A nellx nelly) \vee

l1ist-all2 A (ltl (lbutlast (l1ist-of-nellist nell1))) (ltl (lbutlast (l1ist-of-nellist nell2))))

by (*metis a0 a1 lbutlast.ctr(1) lbutlast-ltl l1ist.discI(1) ltl-l1ist-of-nellist ltl-l1ist-of-nellist1*
nellist.rel-sel)

show *?thesis* **using** *1 2* **by** *blast*

qed

qed

lemma *nellist-all2-nellist-of-l1ist-a [simp]*:

nellist-all2 A (nellist-of-l1ist-a b llx) (nellist-of-l1ist-a c lly) \longleftrightarrow

l1ist-all2 A llx lly \wedge (lfinite llx \longrightarrow A b c)

proof (*cases lfinite llx*)

case *True*

then show *?thesis*

using *l1ist-all2-nellist-of-l1istI*

by (*auto simp add: l1ist-all2-lappendI nellist-all2-help-a, fastforce,*

metis lbutlast-lfinite lbutlast-snoc l1ist-all2-lfiniteD nellist-all2-nfinite1-nlastD

nellist-of-l1ist-a-inverse nfinite-def nlast-nellist-of-l1ist-a-lfinite)

next

case *False*

then show *?thesis*

by (*metis lbutlast-snoc l1ist-all2-lappendI l1ist-all2-nellist-of-l1istI nellist-all2-help-a*
nellist-of-l1ist-a-inverse)

qed

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 $\exists x \in \text{nset } nell. P x$

shows *llist-of-nellist* (*nfilter* *P* *nell*) = *lfilter* *P* (*llist-of-nellist* *nell*)

using *assms*

by *transfer* *auto*

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*:

(($\lambda ll.$ if $\exists b. ll = LCons b LNil$ then ll else $ltl ll$) $\sim n$) (LCons b LNil) = LCons b LNil
by (*induction n*) *simp-all*

lemma *ntl-power-NNil*:

(ntl $\sim n$) (NNil b) = (NNil b)
by *transfer (auto simp add: lnull-def ltl-power-NNil-help)*

lemma *ntl-power-NCons*:

ntl ((ntl $\sim n$) (NCons x nell)) = (ntl $\sim n$) nell
by (*induction n*) (*transfer, auto*)

lemma *ntl-power-Suc* [*simp*]:

(ntl $\sim (Suc n)$) nell = (case nell of NNil b \Rightarrow NNil b | NCons x nell' \Rightarrow (ntl $\sim n$) 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.prem **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.prem **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 nlength nell ≤ enat(n+m)
shows nnth (ndropn n nell) m = (nlast nell)
proof -
have 1: nfinite nell
using assms enat-ile nfinite-conv-nlength-enat by auto
have 2: nfinite nell ⇒ nlength 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-nlength-def iless-Suc-eq leD le-add1 ndropn-0 nellist.sel(2) nlast0-nlast nlength-NCons
  nlength-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 nlength-NCons)
qed
qed
show ?thesis using 1 2 assms by auto
qed

```

lemma *ndropn-ntl* :

ndropn n nell = (ntl \sim 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

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≡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 ≤ enat n ⇒ ndropn n nell = (NNil (nlast nell))

by (*metis enat-ile ndropn-eq-NNil ndropn-nlast nlength-eq-enat-nfiniteD*)

lemma *ndropn-nappend1:*

nfinite nellx ⇒ n ≤ nlength nelly ⇒

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 ≤ (nlength nellx) ⇒ 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 ⇒

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*)

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* *nellx'* *y* *nelly'*. *nellx* = (*NCons* *x* *nellx'*) \wedge *nelly* = (*NCons* *y* *nelly'*) \wedge
z = (*x*, *y*) \wedge *zs* = (*nzip* *nellx'* *nelly'*))
by (*cases nellx nelly rule: nellist.exhaust*[*case-product nellist.exhaust*])
auto

lemma *nzip-nappend*:
nlength *nellx* = *nlength* *nellu*
 \implies *nzip* (*nappend* *nellx* *nelly*) (*nappend* *nellu* *nellv*) =
nappend (*nzip* *nellx* *nellu*) (*nzip* *nelly* *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*]:
 $n \leq nlength\ nellx \wedge n \leq nlength\ nelly \implies$
 $ndropn\ n\ (nzip\ nellx\ nelly) = nzip\ (ndropn\ n\ nellx)\ (ndropn\ n\ nelly)$
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 $n \leq nlength\ nellx$
 $n \leq nlength\ nelly$
shows $nnth\ (nzip\ nellx\ nelly)\ n = (nnth\ nellx\ n,\ nnth\ nelly\ n)$
using *assms*
by *transfer*
(auto simp add: min-def,
metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0 lnth-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-lset-conv-lnth,
metis Pair-inject co.enat.exhaust-sel iless-Suc-eq llength-eq-0 lnth-lzip min.orderE
the-enat.simps,
metis eSuc-epred iless-Suc-eq llength-eq-0 lnth-lzip)

lemma *nset-nzipD1*:
 $(x,\ y) \in nset\ (nzip\ nellx\ nelly) \implies x \in nset\ nellx$
by *transfer*
(meson lset-lzipD1)

lemma *nset-nzipD2*:
 $(x,\ y) \in nset\ (nzip\ nellx\ nelly) \implies y \in nset\ nelly$
by *transfer*
(meson lset-lzipD2)

lemma *nset-nzip-same* [simp]:

$nset (nzip\ nellx\ nelly) = (\lambda x. (x, x)) \text{ ` } nset\ nellx$

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 $\exists\ nellx'\ nellx'' nelly'\ nelly''.$

$nellx = nappend\ nellx'\ nellx'' \wedge nelly = nappend\ nelly'\ nelly'' \wedge$

$nlength\ nellx' = nlength\ nelly' \wedge$

$nellu = nzip\ nellx'\ nelly' \wedge nellv = nzip\ nellx''\ nelly''$

using *assms*

apply *transfer*

using *lzip-eq-lappend-conv*

by (*auto simp add: lzip-eq-lappend-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*:

$nzip\ nellx\ nelly =$

$nzip\ (ntake\ (min\ (nlength\ nellx)\ (nlength\ nelly))\ nellx)$
 $\quad (ntake\ (min\ (nlength\ nellx)\ (nlength\ nelly))\ nelly)$
by *transfer*
 $(auto,$
 $\quad metis\ co.enat.exhaust-sel\ epred-min\ i0-lb\ llength-lzip\ ltake-all\ ltake-lzip\ order-refl)$

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 $nellist-all2\ P\ nellx\ nelly$
 $y \in nset\ nelly$
shows $\exists x \in nset\ nellx. P\ x\ y$
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 $2: (is-NNil\ (nzip\ nx\ nx') \implies$
 $\quad is-NNil\ (nzip\ ny\ ny') \implies$
 $\quad rel-prod\ P\ P'\ (nlast\ (nzip\ nx\ nx'))\ (nlast\ (nzip\ ny\ ny')))$
by $(metis\ (no-types,\ lifting)\ enat-min-eq-0-iff\ ilist-all2(1)\ ilist-all2(2)$
 $\quad is-NNil-nzip\ min-def-raw\ nellist.sel(1)\ nellist-all2-conv-all-nnth$
 $\quad nzip-eq-NNil-conv\ order-refl\ rel-prod-inject\ zero-enat-def)$

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)

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 $(nmap\ f\ \frown n)\ (h\ x) = (nmap\ f\ \frown n)\ (niterates\ f\ x)$
proof *(coinduction arbitrary: n)*
case *(Eq-nellist nn)*
then show *?case*
proof –
have *1:* $is-NNil\ ((nmap\ f\ \frown nn)\ (h\ x)) = is-NNil\ ((nmap\ f\ \frown nn)\ (niterates\ f\ x))$
by *auto*
have *2:* $(is-NNil\ ((nmap\ f\ \frown nn)\ (h\ x)) \longrightarrow is-NNil\ ((nmap\ f\ \frown nn)\ (niterates\ f\ x)) \longrightarrow nlast\ ((nmap\ f\ \frown nn)\ (h\ x)) = nlast\ ((nmap\ f\ \frown nn)\ (niterates\ f\ x))$
by *simp*
have *3:* $(\neg is-NNil\ ((nmap\ f\ \frown nn)\ (h\ x)) \longrightarrow \neg is-NNil\ ((nmap\ f\ \frown nn)\ (niterates\ f\ x)) \longrightarrow nhd\ ((nmap\ f\ \frown nn)\ (h\ x)) = nhd\ ((nmap\ f\ \frown nn)\ (niterates\ f\ x))$
by *simp*
have *4:* $(\neg is-NNil\ ((nmap\ f\ \frown nn)\ (h\ x)) \longrightarrow \neg is-NNil\ ((nmap\ f\ \frown nn)\ (niterates\ f\ x)) \longrightarrow (ntl\ ((nmap\ f\ \frown nn)\ (h\ x)) = (nmap\ f\ \frown (Suc\ nn))\ (h\ x) \wedge ntl\ ((nmap\ f\ \frown nn)\ (niterates\ f\ x)) = (nmap\ f\ \frown (Suc\ nn))\ (niterates\ f\ x))$


```

    by (metis  $\langle \bigwedge x. \neg \text{is-NNil } (h\ x) \rangle \langle \bigwedge x. \text{ntl } (h\ x) = \text{nmap } f\ (h\ x) \rangle \text{funpow-simps-right}(2)$ 
        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  $\frown$  n) x)
by transfer
  (simp add: ldropn-iterates)

```

```

lemma nnth-niterates [simp]:
  nnth (niterates f x) n = (f  $\frown$  n) x
by transfer auto

```

```

lemma nset-niterates:
  nset (niterates f x) = { (f  $\frown$  n) x | 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

```

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 $nfilter (\lambda x. False) nell = nell$
using *assms* **by** *transfer auto*

lemma *nfilter-NCons [simp]*:
assumes $(\exists x \in nset\ nell. P\ x)$
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 $\neg(\exists x \in nset\ nell. P\ x)$
 $P\ x$
shows $nfilter\ P\ (NCons\ x\ nell) = (NNil\ x)$
using *assms* **by** *transfer auto*

lemma *nfilter-expand*:
assumes $\exists x \in nset\ nell. P\ x$
shows $nfilter\ P\ nell =$
 $(if\ is_NNil\ nell\ then\ nell$
 $\quad else$
 $\quad (if\ (\exists x \in nset(ntl\ nell). P\ x)\ then$
 $\quad \quad (if\ P\ (nhd\ nell)\ then\ (NCons\ (nhd\ nell)\ (nfilter\ P\ (ntl\ nell)))$
 $\quad \quad \quad else\ (nfilter\ P\ (ntl\ nell)\)\)$
 $\quad else\ (NNil\ (nhd\ nell))\)\)$
using *assms* **by** *(cases nell) auto*

lemma *nset-nfilter*:
assumes $\exists x \in nset\ nell. P\ x$
shows $nset\ (nfilter\ P\ nell) = nset\ nell \cap \{xa. P\ xa\}$
using *assms*
by *transfer auto*

lemma *exist-conj*:
assumes $\exists x \in nset\ (nfilter\ Q\ nell). P\ x$
 $\exists x \in nset\ nell. Q\ x$
shows $\exists x \in nset\ nell. P\ x \wedge Q\ x$
using *assms*
by *transfer (auto split: if-split-asm)*

lemma *nfilter-nfilter [simp]*:
assumes $\exists x \in nset\ (nfilter\ Q\ nell). P\ x$
 $\exists x \in nset\ nell. Q\ x$
shows $nfilter\ P\ (nfilter\ Q\ nell) = nfilter\ (\lambda x. P\ x \wedge Q\ x)\ nell$
using *assms*
proof *(transfer fixing: P Q)*
fix *xsa* :: 'a *l*list
assume $\neg\ lnull\ xsa \wedge xsa = xsa$
assume *a1*: $Bex\ (lset\ (if\ lnull\ (lfilter\ Q\ xsa)\ then\ xsa\ else\ lfilter\ Q\ xsa))\ P$
assume $Bex\ (lset\ xsa)\ Q$
then have $\neg\ lnull\ (lfilter\ Q\ xsa)$
by *simp*

then have $\text{lfilter } P \text{ (if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa) = lfilter } P \text{ (lfilter } Q \text{ xsa) } \wedge$
 $\neg \text{lnull (lfilter } P \text{ (lfilter } Q \text{ xsa)) } \wedge$
 $\neg \text{lnull (if lnull (lfilter } P \text{ (if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa))$
 $\text{then if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa}$
 $\text{else lfilter } P \text{ (if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa))}$
using *a1* **by** (*auto split: if-split-asm*)
then show $\neg \text{lnull (if lnull (lfilter } P \text{ (if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa))$
 $\text{then if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa}$
 $\text{else lfilter } P \text{ (if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa)) } \wedge$
 $\text{(if lnull (lfilter } P \text{ (if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa))$
 $\text{then if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa}$
 $\text{else lfilter } P \text{ (if lnull (lfilter } Q \text{ xsa) then xsa else lfilter } Q \text{ xsa)) =}$
 $\text{(if lnull (lfilter } (\lambda a. P \ a \wedge Q \ a) \text{ xsa) then xsa else lfilter } (\lambda a. P \ a \wedge Q \ a) \text{ xsa)}$
using *lfilter-lfilter* **by** *auto*
qed

lemma *length-nfilter-le [simp]:*
 $\text{nlength (nfilter } P \text{ nell) } \leq \text{nlength nell}$
by *transfer (simp add: epred-le-epredI llength-lfilter-ile)*

lemma *nfilter-nnth:*
assumes $(\exists x \in \text{nset nell. } P \ x)$
 $i \leq \text{nlength (nfilter } P \text{ nell)}$
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 $\forall x \in \text{nset nell. } \neg P \ x$
 nfinite nell
 $\exists y \in \text{nset nell1. } P \ y$
shows $\text{nfilter } P \text{ (nappend nell nell1) = nfilter } P \text{ nell1}$
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 $\text{nfilter } P \text{ (nappend nell1 nell) = nfilter } P \text{ nell1}$
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 $(\exists x \in \text{nset (nappend nell nell1). } P \ x)$
 $(\exists x \in \text{nset nell. } P \ x)$
 $(\exists x \in \text{nset nell1. } P \ x)$
shows $\text{nfilter } P \text{ (nappend nell nell1) =}$

```

      (if nfinite nell then nappend (nfilter P nell) (nfilter P nell1)
       else (nfilter P 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 nfilter P (nmap f nell) = nmap f (nfilter (P ∘ f) nell)
by transfer (auto simp add: lfilter-lmap)

```

```

lemma nlength-nfilter-nmap[simp]:
shows nlength (nfilter P (nmap f nell)) = nlength(nfilter (P ∘ f) nell)
by (simp add: nfilter-nmap)

```

```

lemma nfilter-is-subset [simp]:
assumes  $\exists x \in \text{nset } nell. P x$ 
shows  $\text{nset } (nfilter P nell) \leq \text{nset } nell$ 
using assms by (simp add: nset-nfilter)

```

```

lemma nfilter-cong[fundef-cong]:
assumes  $nell = nell1$ 
       $(\bigwedge x. x \in \text{nset } nell1 \implies P x = Q x)$ 
shows  $nfilter P nell = nfilter Q nell1$ 
using assms by transfer auto

```

```

lemma nset-nappend:
   $\text{nset } (nappend nell nell1) = (\text{if } nfinite\ nell \text{ then } \text{nset } nell \cup \text{nset } nell1 \text{ else } \text{nset } nell)$ 
by transfer (simp add: lappend-inf)

```

```

lemma split-nellist-nappend:
assumes  $\exists i \leq nlength\ nell. nnth\ nell\ i = x \wedge P\ (nnth\ nell\ i) \wedge$ 
       $(\forall j. j \neq i \wedge j \leq nlength\ nell \longrightarrow \neg P(nnth\ nell\ j))$ 
shows  $P x \wedge$ 
       $(nell = (NNil\ x) \vee$ 
         $(\exists vs. nell = (NCons\ x\ vs) \wedge (\forall v \in \text{nset } vs. \neg P v)) \vee$ 
         $(\exists us. nell = nappend\ us\ (NNil\ x) \wedge nfinite\ us \wedge (\forall u \in \text{nset } us. \neg P u)) \vee$ 
         $(\exists us\ vs. nell = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$ 
           $(\forall u \in \text{nset } us. \neg P u) \wedge (\forall v \in \text{nset } vs. \neg P v))$ 
      )
proof –
obtain  $i$  where  $1: i \leq nlength\ nell \wedge nnth\ nell\ i = x \wedge P\ (nnth\ nell\ i) \wedge$ 
       $(\forall j. j \neq i \wedge j \leq nlength\ nell \longrightarrow \neg P(nnth\ nell\ j))$ 
using assms by auto
have  $01: (\forall j. (j < i \vee i < j) \wedge j \leq nlength\ nell \longrightarrow \neg P(nnth\ nell\ j))$ 
using  $1$ 

```

by *auto*
have 2: $i=0 \wedge \text{nlength } nell = 0 \implies P x \wedge nell = (NNil x)$
 by (metis 1 ndropn-0 ndropn-eq-NNil the-enat-0 zero-enat-def)
have 3: $i=0 \wedge \text{nlength } nell > 0 \implies P x \wedge nell = (NCons x (\text{ntl } nell)) \wedge (\forall v \in \text{nset } (\text{ntl } nell). \neg P v)$
 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: $i=0 \wedge \text{nlength } nell > 0 \implies P x \wedge (\exists vs. nell = (NCons x vs) \wedge (\forall v \in \text{nset } vs. \neg P v))$
 using 3 by *auto*
have 5: $i > 0 \wedge i = \text{nlength } nell \wedge \text{nfinite } nell \implies P x \wedge nell = \text{nappend } (\text{ntaken } (i-1) nell) (NNil x)$
 using 1
 by transfer
 (auto,
 metis eSuc-epred lappend-lbutlast-llast-id-lfinite lbutlast-conv-ltake llast-conv-lnth
 llength-eq-0 the-enat.simps)
have 6: $i > 0 \wedge i = \text{nlength } nell \wedge \text{nfinite } nell \implies (\forall u \in \text{nset } (\text{ntaken } (i-1) nell). \neg P u)$
 using 1
 by transfer
 (auto,
 metis in-lset-conv-lnth lbutlast-conv-ltake less-imp-le llength-lbutlast lnth-ltake
 min.strict-order-iff)
have 7: $i > 0 \wedge i = \text{nlength } nell \wedge \text{nfinite } nell \implies P x \wedge (\exists us. nell = \text{nappend } us (NNil x) \wedge$
 $\text{nfinite } us \wedge (\forall u \in \text{nset } us. \neg P u))$
 using 5 6 by *auto*
have 8: $i > 0 \wedge i < \text{nlength } nell \implies$
 $P x \wedge nell = \text{nappend } (\text{ntaken } (i-1) nell) (NCons x (\text{ndropn } (i+1) nell))$
 using 1
 by transfer
 (auto,
 metis co-enat.collapse eSuc-enat ileI1 iless-Suc-eq lappend-ltake-enat-ldropn
 ldropn-Suc-conv-ldropn llength-eq-0 min.absorb1 the-enat.simps)
have 9: $i > 0 \wedge i < \text{nlength } nell \implies \text{nfinite } (\text{ntaken } (i-1) nell)$
 using enat-ord-code(4) nfinite-ntaken by blast
have 10: $i > 0 \wedge i < \text{nlength } nell \implies (\forall u \in \text{nset } (\text{ntaken } (i-1) nell). \neg P u)$
 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: $i > 0 \wedge i < \text{nlength } nell \implies (\forall v \in \text{nset } (\text{ndropn } (i+1) nell). \neg P v)$
 using 1 01 in-nset-conv-nnth[of - (ndropn (i+1) nell)]
 by simp
 (metis Suc-ile-eq iless-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: $i > 0 \wedge i < \text{nlength } nell \implies (\exists us vs. nell = \text{nappend } us (NCons x vs) \wedge \text{nfinite } us \wedge$
 $(\forall u \in \text{nset } us. \neg P u) \wedge (\forall v \in \text{nset } vs. \neg P v))$
 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 $0 < \text{nlength } nell$
shows $nell = (NCons (\text{nnth } nell\ 0) (\text{ntl } nell))$
using *assms* **by** (*metis ndropn-0 ndropn-Suc-conv-ndropn nellist.sel(5) zero-enat-def*)

lemma *nellist-split-2-last*:

assumes $0 < i$
 $i = \text{nlength } nell$
 $\text{nfinite } nell$
shows $nell = \text{nappend } (\text{ntaken } (i-1) \text{ nell}) (NNil (\text{nnth } nell\ i))$
using *assms*
by *transfer*
(simp,
 $\text{metis Suc-ile-eq co.enat.exhaust-sel eSuc-enat llength-eq-0 ltake-Suc-conv-snoc-lnth ltake-all}$
 $\text{order-refl the-enat.simps})$

lemma *nellist-split-3*:

assumes $0 < i$
 $i < \text{nlength } nell$
shows $nell = \text{nappend } (\text{ntaken } (i-1) \text{ nell}) (NCons (\text{nnth } nell\ i) (\text{ndropn } (i+1) \text{ nell}))$
using *assms* **by** *transfer*
(auto,
 $\text{metis eSuc-enat eSuc-epred ileI1 iless-Suc-eq lappend-ltake-enat-ldropn ldropn-Suc-conv-ldropn}$
 $\text{less-imp-le llength-eq-0 min-def the-enat.simps})$

lemma *NNil-eq-nfilterD*:

assumes $\exists x \in \text{nset } nell. P\ x$
 $(NNil\ x) = \text{nfilter } P\ nell$
shows $(\exists\ us\ vs. (nell = (NNil\ x) \vee$
 $(nell = (NCons\ x\ vs) \wedge (\forall\ v \in \text{nset } vs. \neg P\ v)) \vee$
 $(nell = \text{nappend } us\ (NNil\ x) \wedge \text{nfinite } us \wedge (\forall\ u \in \text{nset } us. \neg P\ u)) \vee$
 $(nell = \text{nappend } us\ (NCons\ x\ vs) \wedge \text{nfinite } us \wedge$
 $(\forall\ u \in \text{nset } us. \neg P\ u) \wedge (\forall\ v \in \text{nset } vs. \neg P\ v))$
 $) \wedge P\ x)$

proof –

have 1: $(\exists\ i \leq \text{nlength } nell. P (\text{nnth } nell\ i))$
by (*metis assms(1) in-nset-conv-nnth*)
obtain *i* **where** 2: $i \leq \text{nlength } nell \wedge P (\text{nnth } nell\ i)$
using 1 **by** *auto*
have 3: $i = 0 \wedge \text{nlength } nell = 0 \implies P\ x \wedge nell = (NNil\ x)$
by (*metis 2 assms(2) ndropn-0 ndropn-eq-NNil nfilter-NNil the-enat-0 zero-enat-def*)
have 4: $i=0 \wedge \text{nlength } nell > 0 \implies P\ x \wedge nell = (NCons\ x (\text{ntl } nell)) \wedge (\forall\ v \in \text{nset } (\text{ntl } nell). \neg P\ v)$
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: $i=0 \wedge \text{nlength } nell > 0 \implies P\ x \wedge (\exists\ vs. nell = (NCons\ x\ vs) \wedge (\forall\ v \in \text{nset } vs. \neg P\ v))$
using 4 **by** *auto*
have 60: $0 < i \wedge i = \text{nlength } nell \wedge \text{nfinite } nell \implies x = (\text{nnth } nell\ i)$
using 2 *assms nellist-split-2-last[of i nell]*
proof *simp*
assume *a1*: $nell = \text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ nell}) (NNil (\text{nnth } nell\ i))$
assume *a2*: $P (\text{nnth } nell\ i)$

```

assume a3:  $NNil\ x = nfilter\ P\ nell$ 
assume a4:  $\exists x \in nset\ nell.\ P\ x$ 
have f5:  $\forall as\ p\ asa.\ ((\exists a.\ (a::'a) \in nset\ as \wedge p\ a) \vee \neg nfinite\ as \vee (\forall a.\ a \notin nset\ asa \vee \neg p\ a))$ 
 $\vee nfilter\ p\ (nappend\ as\ asa) = nfilter\ p\ asa$ 
by (metis (full-types) nfilter-nappend1)
have f7:  $\exists a.\ a \in nset\ (NNil\ (nnth\ nell\ i)) \wedge P\ a$ 
using a2 by auto
have f8:  $nfilter\ P\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NNil\ (nnth\ nell\ i))) \neq$ 
 $nappend\ (nfilter\ P\ (ntaken\ (i - Suc\ 0)\ nell))\ (nfilter\ P\ (NNil\ (nnth\ nell\ i)))$ 
using a3 a1 by (metis (full-types) is-NNil-nappend nellist.disc(1))
have f9:  $nfinite\ (ntaken\ (i - Suc\ 0)\ nell)$ 
by simp
obtain aaa :: 'a where
f9:  $aaa \in nset\ nell \wedge P\ aaa$ 
using a4 by blast
then have  $aaa \in nset\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NNil\ (nnth\ nell\ i)))$ 
using a1 by presburger
then have f10:  $\forall a.\ a \notin nset\ (ntaken\ (i - Suc\ 0)\ nell) \vee \neg P\ a$ 
using f9 f8 f7 by (meson nfilter-nappend nfinite-ntaken)
then show ?thesis
using f9 f7 f5 a3 a1
proof -
have  $NNil\ (nnth\ nell\ i) = NNil\ x$ 
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:  $0 < i \wedge i = nlength\ nell \wedge nfinite\ nell \implies P\ x \wedge nell = nappend\ (ntaken\ (i-1)\ nell)\ (NNil\ x)$ 
using 2 60 assms nellist-split-2-last[of  $i\ nell$ ]
by blast
have 7:  $i > 0 \wedge i = nlength\ nell \wedge nfinite\ nell \implies (\forall u \in nset\ (ntaken\ (i-1)\ nell).\ \neg P\ u)$ 
using assms 6 nfilter-nappend[of  $(ntaken\ (i-1)\ nell) - P$ ]
by (metis is-NNil-nappend nellist.disc(1) nfinite-ntaken split-nellist-a)
have 8:  $i > 0 \wedge i = nlength\ nell \wedge nfinite\ nell \implies P\ x \wedge (\exists us.\ nell = nappend\ us\ (NNil\ x) \wedge$ 
 $nfinite\ us \wedge (\forall u \in nset\ us.\ \neg P\ u))$ 
using 6 7 by auto
have 9:  $i > 0 \wedge i < nlength\ nell \implies$ 
 $P\ x \wedge nell = nappend\ (ntaken\ (i-1)\ nell)\ (NCons\ x\ (ndropn\ (i+1)\ nell))$ 
using 2 assms nellist-split-3[of  $i\ nell$ ]
 $nfilter-nappend1$ [of  $(ntaken\ (i-1)\ nell)\ P\ (NCons\ x\ (ndropn\ (i+1)\ nell))$ ]
 $nfilter-nappend$ [of  $(ntaken\ (i-1)\ nell) - P$ ]
proof simp
assume a1:  $enat\ i \leq nlength\ nell \wedge P\ (nnth\ nell\ i)$ 
assume a2:  $NNil\ x = nfilter\ P\ nell$ 
assume a3:  $\exists x \in nset\ nell.\ P\ x$ 
assume a4:  $nell = nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\ nell))$ 
have f5:  $P\ (nnth\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\ nell)))\ i$ 
using a1 a4 by auto
have f6:  $NNil\ x = nfilter\ P\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\$ 

```

```

nell)))
  using a2 a4 by auto
  have f7:  $\forall as\ p\ asa. ((\exists a. (a::'a) \in nset\ as \wedge p\ a) \vee \neg\ nfinite\ as \vee (\forall a. a \notin nset\ asa \vee \neg\ p\ a)) \vee$ 
     $nfilter\ p\ (nappend\ as\ asa) = nfilter\ p\ asa$ 
  by (metis (full-types) nfilter-nappend1)
  have nfilter P (nappend (ntaken (i - Suc 0) nell) (NCons (nnth nell i) (ndropn (Suc i) nell))) =
    nappend (nfilter P (ntaken (i - Suc 0) nell)) (nfilter P (NCons (nnth nell i) (ndropn (Suc i)
nell)))
     $\longrightarrow$ 
    nappend (nfilter P (ntaken (i - Suc 0) nell)) (nfilter P (NCons (nnth nell i) (ndropn (Suc i)
nell))) =
      NNil x
  using f6 by presburger
  then have nfilter P (nappend (ntaken (i - Suc 0) nell) (NCons (nnth nell i) (ndropn (Suc i) nell)))
 $\neq$ 
    nappend (nfilter P (ntaken (i - Suc 0) nell)) (nfilter P (NCons (nnth nell i) (ndropn (Suc i)
nell)))
  by (metis (no-types) is-NNil-nappend nellist.disc(1))
  then have f9:  $(\forall a. a \notin nset\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\ nell)) \vee \neg\ P\ a) \vee$ 
     $nfilter\ P\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\ nell)))$ 
  =
     $nfilter\ P\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\ nell)) \vee$ 
     $(\forall a. a \notin nset\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\$ 
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:  $aaa \in nset\ nell \wedge P\ aaa$ 
  using a3 by blast
  then have f11:  $aaa \in nset\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\$ 
nell)))
  using a4 by presburger
  have f12:  $nnth\ (nappend\ (ntaken\ (i - Suc\ 0)\ nell)\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\ nell)))\ i \in$ 
     $nset\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\ nell))$ 
  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)))
    i) (ndropn (Suc i) nell))

  using a4 by presburger
  then have f13:  $\forall a. a \notin nset\ (ndropn\ (Suc\ i)\ nell) \vee \neg\ P\ a$ 
  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)))
    i) (ndropn (Suc i) nell)) = NNil x
  using  $\langle NNil\ x = nfilter\ P\ (NCons\ (nnth\ nell\ i)\ (ndropn\ (Suc\ i)\ nell)) \rangle$  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 $f14$: $x = \text{nnth } nell \ i$
using $a4$ **by** *presburger*
then have $nell = \text{nappend } (\text{ntaken } (i - \text{Suc } 0) \ nell) \ (NCons \ x \ (\text{ndropn } (\text{Suc } i) \ nell))$
using $a4$ **by** *blast*
then show $P \ x \wedge \ nell = \text{nappend } (\text{ntaken } (i - \text{Suc } 0) \ nell) \ (NCons \ x \ (\text{ndropn } (\text{Suc } i) \ nell))$
using $f14 \ a1$ **by** *blast*
qed
have 10 : $i > 0 \wedge i < \text{nlength } nell \implies (\forall u \in \text{nset } (\text{ntaken } (i-1) \ nell). \neg P \ u)$
using $9 \ \text{assms}$ $\text{nfilter-nappend[of } (\text{ntaken } (i-1) \ nell) - P]$
by (*metis is-NNil-nappend nellist.disc(1) nellist.set-intros(2) nfinite-ntaken*)
have 11 : $i > 0 \wedge i < \text{nlength } nell \implies (\forall v \in \text{nset } (\text{ndropn } (i+1) \ nell). \neg P \ v)$
using $9 \ 10 \ \text{assms}$ $\text{nfilter-nappend[of } (\text{ntaken } (i-1) \ nell) - P]$
 $\text{nfilter-nappend1[of } (\text{ntaken } (i-1) \ nell) \ P \ (NCons \ x \ (\text{ndropn } (i+1) \ nell))]$
by (*metis nellist.distinct(1) nellist.set-intros(2) nfilter-NCons nfinite-ntaken*)
have 12 : $i > 0 \wedge i < \text{nlength } nell \implies$
 $P \ x \wedge (\exists \ us \ vs. \ nell = \text{nappend } us \ (NCons \ x \ vs) \wedge \text{nfinite } us \wedge$
 $(\forall u \in \text{nset } us. \neg P \ u) \wedge (\forall v \in \text{nset } vs. \neg P \ v))$
using $9 \ 10 \ 11 \ \text{using } \text{assms}(1)$ **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 $\exists x \in \text{nset } nell. \ P \ x$
 $\text{nfilter } P \ nell = (NNil \ x)$
shows $(\exists \ us \ vs. \ (nell = (NNil \ x) \vee$
 $(nell = (NCons \ x \ vs) \wedge (\forall v \in \text{nset } vs. \neg P \ v)) \vee$
 $(nell = \text{nappend } us \ (NNil \ x) \wedge \text{nfinite } us \wedge (\forall u \in \text{nset } us. \neg P \ u)) \vee$
 $(nell = \text{nappend } us \ (NCons \ x \ vs) \wedge \text{nfinite } us \wedge$
 $(\forall u \in \text{nset } us. \neg P \ u) \wedge (\forall v \in \text{nset } vs. \neg P \ v))$
 $) \wedge P \ x)$
using assms $NNil\text{-eq-nfilterD[of } nell \ P \ x]$ **by** *simp*

lemma *nfilter-eq-NNil-iff*:

assumes $\exists x \in \text{nset } nell. \ P \ x$
shows $(\text{nfilter } P \ nell = (NNil \ x)) \longleftrightarrow$
 $(\exists \ us \ vs. \ (nell = (NNil \ x) \vee$
 $(nell = (NCons \ x \ vs) \wedge (\forall v \in \text{nset } vs. \neg P \ v)) \vee$
 $(nell = \text{nappend } us \ (NNil \ x) \wedge \text{nfinite } us \wedge (\forall u \in \text{nset } us. \neg P \ u)) \vee$
 $(nell = \text{nappend } us \ (NCons \ x \ vs) \wedge \text{nfinite } us \wedge$
 $(\forall u \in \text{nset } us. \neg P \ u) \wedge (\forall v \in \text{nset } vs. \neg P \ v))$
 $) \wedge P \ x)$

proof –

have 1 : $(\text{nfilter } P \ nell = (NNil \ x)) \implies$
 $(\exists \ us \ vs. \ (nell = (NNil \ x) \vee$
 $(nell = (NCons \ x \ vs) \wedge (\forall v \in \text{nset } vs. \neg P \ v)) \vee$
 $(nell = \text{nappend } us \ (NNil \ x) \wedge \text{nfinite } us \wedge (\forall u \in \text{nset } us. \neg P \ u)) \vee$
 $(nell = \text{nappend } us \ (NCons \ x \ vs) \wedge \text{nfinite } us \wedge$
 $(\forall u \in \text{nset } us. \neg P \ u) \wedge (\forall v \in \text{nset } vs. \neg P \ v))$

$) \wedge P x)$
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 $\exists x \in nset\ nell. P\ x$
 $(NCons\ x\ nell1) = nfilter\ P\ nell$
shows $(\exists us\ vs.$
 $(nell = (NCons\ x\ vs) \vee$
 $nell = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge (\forall u \in nset\ us. \neg P\ u)) \wedge$
 $(\exists x \in nset\ vs. P\ x) \wedge P\ x \wedge nell1 = nfilter\ P\ vs)$
using *assms* **by** *transfer (auto split: if-split-asm simp add: NCons-eq-nfilterD-help)*

lemma *nfilter-eq-NConsD*:

assumes $\exists x \in nset\ nell. P\ x$
 $nfilter\ P\ nell = (NCons\ x\ nell1)$
shows $(\exists us\ vs.$
 $(nell = (NCons\ x\ vs) \vee$
 $nell = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge (\forall u \in nset\ us. \neg P\ u)) \wedge$
 $(\exists x \in nset\ vs. P\ x) \wedge P\ x \wedge nell1 = nfilter\ P\ vs)$
using *assms NCons-eq-nfilterD*[of *nell P x nell1*] **by** *simp*

lemma *nfilter-eq-NCons-iff*:

assumes $\exists x \in nset\ nell. P\ x$
shows $nfilter\ P\ nell = (NCons\ x\ nell1) \longleftrightarrow$
 $(\exists us\ vs.$

```

    ( nell = (NCons x vs) ∨
      nell = nappend us (NCons x vs) ∧ nfinite us ∧ (∀ u ∈ nset us. ¬ P u) ) ∧
    (∃ x ∈ nset vs. P x) ∧ P x ∧ nell1 = nfilter P vs )
proof –
have 1: nfilter P nell = (NCons x nell1) ⇒
  (∃ us vs.
    ( nell = (NCons x vs) ∨
      nell = nappend us (NCons x vs) ∧ nfinite us ∧ (∀ u ∈ nset us. ¬ P u) ) ∧
    (∃ x ∈ nset vs. P x) ∧ P x ∧ nell1 = nfilter P vs )
    using assms nfilter-eq-NConsD[of nell P x nell1] by simp
have 2: (∃ us vs.
  ( nell = (NCons x vs) ∨
    nell = nappend us (NCons x vs) ∧ nfinite us ∧ (∀ u ∈ nset us. ¬ P u) ) ∧
    (∃ x ∈ nset vs. P x) ∧ P x ∧ nell1 = nfilter P vs ) ⇒ 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 (∃ x ∈ nset nell. P x)
shows (nfilter P nell = nell) = (∀ x ∈ nset nell. P x) (is ?lhs = ?rhs)
using assms by transfer (auto simp add: lfilter-id-conv)

```

lemma *nfilter-idem*:

```

assumes (∃ x ∈ nset nell. P x)
shows nfilter P (nfilter P nell) = nfilter P nell
using assms by transfer auto

```

lemma *nfilter-ndropn-nlength*:

```

assumes k ≤ nlength nell
  ∃ x ∈ nset ( (ndropn k nell)). P x
shows nlength(nfilter P ( (ndropn k nell))) ≤ nlength (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)

```

2.20 Setup for Lifting/Transfer

2.20.1 Relator and predicator properties

abbreviation *nellist-all* == *pred-nellist*

2.20.2 Transfer rules for the Transfer package

context *includes lifting-syntax*

begin

lemma *set1-pre-nellist-transfer* [*transfer-rule*]:

```

  (rel-pre-nellist A C ==> rel-set A) 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*]:

(*rel-pre-nellist A C* \implies *rel-set C*) *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* \implies *nellist-all2 A* \implies *nellist-all2 A*) *NCons NCons*

unfolding *rel-fun-def* **by** *simp*

declare *NCons-transfer* [*transfer-rule*]

lemma *case-nellist-transfer* [*transfer-rule*]:

((*A* \implies *C*) \implies (*A* \implies *nellist-all2 A* \implies *C*) \implies *nellist-all2 A* \implies *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* \implies (=)) \implies (*A* \implies *C*) \implies (*A* \implies *C*) \implies (*A* \implies *A*) \implies *A* \implies *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* \implies (=)) *IS-NNIL1 IS-NNIL2 (A* \implies *C*) *NLAST1 NLAST2*

(*A* \implies *C*) *NHD1 NHD2 (A* \implies *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 $\langle A \ x \ y \rangle$ **using** *rel* **by** (*coinduction arbitrary: x y*) (*auto 4 4 elim: rel-funE*)

qed

lemma *corec-nellist-transfer* [*transfer-rule*]:

((*A* \implies (=)) \implies (*A* \implies *C*) \implies (*A* \implies *C*) \implies (*A* \implies (=)) \implies (*A* \implies *nellist-all2 C*)

\implies (*A* \implies *A*) \implies *A* \implies *nellist-all2 C*) *corec-nellist corec-nellist*

by (*simp add: nellist.corec-transfer*)

lemma *ntl-transfer2* [*transfer-rule*]:

(*nellist-all2 A* \implies *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* \implies *rel-set A*) *nset nset*

by (*simp add: nellist.set-transfer*)

lemma *nmap-transfer4* [*transfer-rule*]:

$((A \text{====>} B) \text{====>} \text{nellist-all2 } A \text{====>} \text{nellist-all2 } B) \text{ nmap nmap}$
by (*simp add: nellist.map-transfer*)
declare *nmap-transfer* [*transfer-rule*]

lemma *is-NNil-transfer2* [*transfer-rule*]:
 $(\text{nellist-all2 } A \text{====>} (=)) \text{ is-NNil is-NNil}$
by (*auto dest: nellist-all2-is-NNilD*)
declare *is-NNil-transfer* [*transfer-rule*]

lemma *snocn-transfer2* [*transfer-rule*]:
 $(\text{nellist-all2 } A \text{====>} A \text{====>} \text{nellist-all2 } A) \text{ snocn snocn}$
unfolding *rel-fun-def*
by (*metis nappend-snocn nellist-all2-NNil nellist-all2-nappendI*)
declare *snocn-transfer* [*transfer-rule*]

lemma *nappend-transfer* [*transfer-rule*]:
 $(\text{nellist-all2 } A \text{====>} (\text{nellist-all2 } A) \text{====>} \text{nellist-all2 } A) \text{ nappend nappend}$
by (*auto intro: nellist-all2-nappendI elim: rel-funE*)
declare *nappend.transfer* [*transfer-rule*]

lemma *lappendn-transfer* [*transfer-rule*]:
 $(\text{llist-all2 } A \text{====>} \text{nellist-all2 } A \text{====>} \text{nellist-all2 } A) \text{ lappendn lappendn}$
unfolding *rel-fun-def*
by *transfer* (*auto intro: llist-all2-lappendI*)
declare *lappendn.transfer* [*transfer-rule*]

lemma *nellist-of-llist-a-transfer2* [*transfer-rule*]:
 $(A \text{====>} \text{llist-all2 } A \text{====>} \text{nellist-all2 } A) \text{ nellist-of-llist-a nellist-of-llist-a}$
by (*simp add: rel-funI*)
declare *nellist-of-llist-a-transfer* [*transfer-rule*]

lemma *nlenght-transfer* [*transfer-rule*]:
 $(\text{nellist-all2 } A \text{====>} (=)) \text{ nlenght nlenght}$
by (*auto dest: nellist-all2-nlenghtD*)
declare *nlenght.transfer* [*transfer-rule*]

lemma *ndropn-transfer* [*transfer-rule*]:
 $((=) \text{====>} \text{nellist-all2 } A \text{====>} \text{nellist-all2 } A) \text{ ndropn ndropn}$
unfolding *rel-fun-def*
by *transfer* (*auto intro: llist-all2-ldropnI simp add: llist-all2-ldropnI llist-all2-llenghtD*)
declare *ndropn.transfer* [*transfer-rule*]

lemma *ntake-transfer* [*transfer-rule*]:
 $((=) \text{====>} \text{nellist-all2 } A \text{====>} \text{nellist-all2 } A) \text{ ntake ntake}$
unfolding *rel-fun-def*
by *transfer* (*auto simp add: llist-all2-ltakeI*)
declare *ntake.transfer* [*transfer-rule*]

lemma *ntaken-transfer* [*transfer-rule*]:
 $((=) \text{====>} \text{nellist-all2 } A \text{====>} \text{nellist-all2 } A) \text{ ntaken ntaken}$

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

3 Extra operations on nellists

The operations *ndropns*, *nkfilter*, *nleast*, *nidx*, *nfuse*, *nbutlast*, *nsubn*, *nridx*, *nlastnfirst* and *nfusecat* are defined for nellists together with a library of lemmas.

```

theory NELList-Extras
imports NELList
begin

```

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 ∈ nset(ndropns nellx) ⟷ (∃ i. i ≤ nlength nellx ∧ nell = ndropn i nellx)
by (metis in-nset-conv-nnth ndropns-nlength ndropns-nnth)

```

```

lemma nset-ndropns:
  nset (ndropns nell) = { ndropn i nell | i. i ≤ 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 (λ nell. nnth nell 0) (ndropns nell) = nell
by (coinduction arbitrary: nell)
   (case-tac nell, auto simp add: nnth-NNil)

```

```

lemma ndropn-ndropns:
  assumes i ≤ 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: $\text{enat } \text{nellx} \leq \text{nlength } (\text{ndropns } \text{ia}) \implies$
 $\text{is-NNil } (\text{ndropn } \text{nellx } (\text{ndropns } \text{ia})) = \text{is-NNil } (\text{ndropns } (\text{ndropn } \text{nellx } \text{ia}))$
by (*simp add: is-NNil-ndropn ndropns-nlength*)
have 2: $\text{enat } \text{nellx} \leq \text{nlength } (\text{ndropns } \text{ia}) \implies$
 $(\text{is-NNil } (\text{ndropn } \text{nellx } (\text{ndropns } \text{ia}))) \longrightarrow$
 $\text{is-NNil } (\text{ndropns } (\text{ndropn } \text{nellx } \text{ia})) \longrightarrow$
 $\text{nlast } (\text{ndropn } \text{nellx } (\text{ndropns } \text{ia})) = \text{nlast } (\text{ndropns } (\text{ndropn } \text{nellx } \text{ia}))$
by (*metis dual-order.antisym ndropn-eq-NNil ndropns.disc(2) ndropns.simps(3) ndropns-nlength*
 $\text{ndropns-nnth nellist.case-eq-if nellist.collapse(1) the-enat.simps}$)
have 3: $\text{enat } \text{nellx} \leq \text{nlength } (\text{ndropns } \text{ia}) \implies$
 $(\neg \text{is-NNil } (\text{ndropn } \text{nellx } (\text{ndropns } \text{ia}))) \longrightarrow$
 $\neg \text{is-NNil } (\text{ndropns } (\text{ndropn } \text{nellx } \text{ia})) \longrightarrow$
 $\text{nhd } (\text{ndropn } \text{nellx } (\text{ndropns } \text{ia})) = \text{nhd } (\text{ndropns } (\text{ndropn } \text{nellx } \text{ia}))$
by (*metis Nat.add-0-right ndropn-nnth ndropns.disc(1) ndropns-nlength ndropns-nnth nhd-conv-nnth*
 nhd-ndropns)
have 4: $\text{enat } \text{nellx} \leq \text{nlength } (\text{ndropns } \text{ia}) \implies$
 $(\neg \text{is-NNil } (\text{ndropn } \text{nellx } (\text{ndropns } \text{ia}))) \longrightarrow$
 $\neg \text{is-NNil } (\text{ndropns } (\text{ndropn } \text{nellx } \text{ia})) \longrightarrow$
 $(\exists \text{nell } i.$
 $\text{ntl } (\text{ndropn } \text{nellx } (\text{ndropns } \text{ia})) = \text{ndropn } i (\text{ndropns } \text{nell}) \wedge$
 $\text{ntl } (\text{ndropns } (\text{ndropn } \text{nellx } \text{ia})) = \text{ndropns } (\text{ndropn } i \text{ nell}) \wedge$
 $\text{enat } i \leq \text{nlength } (\text{ndropns } \text{nell}))$
by (*metis Suc-ile-eq is-NNil-ndropn ndropn-Suc-conv-ndropn ndropns-code(2) ndropns-nlength*
 $\text{nellist.sel(5) order.order-iff-strict}$)
show ?thesis
using 1 2 3 4 *Eq-nellist* **by** blast
qed
qed

lemma *ndropns-nfilter-nnth*:

assumes $i \leq \text{nlength } (\text{nfilter } P (\text{ndropns } \text{nell}))$
 $\exists \text{nelly} \in \text{nset}(\text{ndropns } \text{nell}). P \text{ nelly}$
shows $P (\text{nnth } (\text{nfilter } P (\text{ndropns } \text{nell})) i)$
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*:

$x \in \text{nset } \text{nell} \longleftrightarrow (\exists \text{ys}. x = (\text{nnth } \text{ys } 0) \wedge \text{ys} \in \text{nset}(\text{ndropns } \text{nell}))$
by *auto*
 $(\text{metis in-nset-conv-nnth ndropns-nlength nmap-first-ndropns nnth-nmap},$
 $\text{metis Nat.add-0-right in-nset-conv-nnth in-nset-ndropns ndropn-nnth})$

lemma *nset-ndropns-nhd*:

$\text{nset } \text{nell} = \{(\text{nnth } \text{nelly } 0) \mid \text{nelly}. \text{nelly} \in \text{nset}(\text{ndropns } \text{nell})\}$
by *auto*
 $(\text{meson in-nset-ndropns-nhd},$

metis in-nset-conv-nnth in-nset-ndropns nnth-zero-ndropn)

lemma *nellist-all2-ndropnsI*:

nellist-all2 A nellx nelly \implies nellist-all2 (nellist-all2 A) (ndropns nellx) (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)

3.2 Definitions

context

includes *nellist.lifting*

begin

lift-definition *nkfilter* :: (*'a* \Rightarrow *bool*) \Rightarrow *nat* \Rightarrow *'a* *nellist* \Rightarrow *nat* *nellist*

is $\lambda P n xs.$ (*if* *lnull*(*kfilter* *P* *n* *xs*) *then* (*iterates* *Suc* *n*) *else* *kfilter* *P* *n* *xs*)

by *simp*

lift-definition *nleast* :: (*'a* \Rightarrow *bool*) \Rightarrow *'a* *nellist* \Rightarrow *nat*

is $\lambda P xs.$ *lleast* *P* *xs*

by *auto*

lift-definition *nridx* :: (*'a* \Rightarrow *'a* \Rightarrow *bool*) \Rightarrow *'a* *nellist* \Rightarrow *bool*

is $\lambda R xs.$ *ridx* *R* *xs*

by *auto*

lift-definition *nidx* :: *nat* *nellist* \Rightarrow *bool*

is $\lambda xs.$ *lidx* *xs*

by *auto*

lift-definition *nbutlast* :: *'a* *nellist* \Rightarrow *'a* *nellist*

is $\lambda xs.$ (*if* *lnull* (*lbutlast* *xs*) *then* (*LCons* (*llast* *xs*) *LNil*) *else* *lbutlast* *xs*)

by *auto*

lift-definition *nfuse* :: *'a* *nellist* \Rightarrow *'a* *nellist* \Rightarrow *'a* *nellist*

is $\lambda xs ys.$ *lfuse* *xs* *ys*

using *lfuse-conv-lnull* **by** *blast*

lift-definition *nsubn* :: *'a* *nellist* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *'a* *nellist*

is ($\lambda xs n1 n2.$ *lsubc* *n1* *n2* *xs*)

unfolding *lsubc-def*

by *auto*

(metis co.enat.exhaust-sel dual-order.refl enat-ile illess-Suc-eq leD llength-eq-0)

lift-definition *nlastnfirst* :: *'a* *nellist* *nellist* \Rightarrow *bool*

is $\lambda xss.$ *llastlfirst* *xss*

apply (*simp* *add: pcr-nellist-def cr-nellist-def OO-def rel-fun-def llist.rel-eq*)

unfolding *llastlfirst-def*

using *llist-all2-llist-of-nellist-1* **by** *blast*

lemma *nfusecat-help*:
assumes $\exists y. \text{llist-all2 } (\lambda x z. \text{llist-of-nellist } z = x) \text{ llist1 } y \wedge y \neq \text{LNil} \wedge$
 $\text{llist-all2 } (\lambda x z. \text{llist-of-nellist } z = x) \text{ llist2 } y$
shows $\text{lfusecat llist1} \neq \text{LNil} \wedge \text{lfusecat llist1} = \text{lfusecat llist2}$
proof –
have 1: $\text{lfusecat llist1} = \text{lfusecat llist2}$
using *assms llist-all2-llist-of-nellist-1 lnull-def* **by** *blast*
have 2: $\text{lfusecat llist1} \neq \text{LNil}$
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 nellist nellist \Rightarrow 'a nellist
is $\lambda xss. \text{lfusecat } xss$
using *nfusecat-help*
by (*simp add: pcr-nellist-def OO-def cr-nellist-def nellist.pcr-cr-eq rel-fun-def lnull-def*
 $\text{llist-all2-cases llist-all2-rsp llist.rel-eq not-lnull-conv-llist-of-nellist}$) *blast*

3.3 nbutlast

lemma *nbutlast-NNil*[*simp*]:
 $\text{nbutlast } (\text{NNil } x) = (\text{NNil } x)$
apply *transfer*
by *auto*

lemma *nbutlast-snoc* [*simp*]:
 $\text{nbutlast } (\text{nappend nell } (\text{NNil } x)) = \text{nell}$
apply *transfer*
by *auto*

lemma *nbutlast-def1*:
 $\text{nbutlast nell} =$
 $(\text{case nell of } (\text{NNil } x) \Rightarrow (\text{NNil } x) \mid$
 $(\text{NCons } x \text{ nell1}) \Rightarrow$
 $(\text{case nell1 of } (\text{NNil } y) \Rightarrow (\text{NNil } x) \mid$
 $(\text{NCons } z \text{ nell2}) \Rightarrow (\text{NCons } x (\text{nbutlast nell1}))))$
proof (*cases nell*)
case (*NNil x1*)
then show *?thesis* **by** *simp*
next
case (*NCons x21 x22*)
then show *?thesis*
proof –
have 1: $\text{is-NNil } x22 \Longrightarrow \text{nbutlast } (\text{NCons } x21 \text{ } x22) = (\text{NNil } x21)$
by (*metis nappend-NNil nbutlast-snoc nellist.collapse(1)*)
have 2: $\neg \text{is-NNil } x22 \Longrightarrow \text{nbutlast } (\text{NCons } x21 \text{ } x22) = (\text{NCons } x21 (\text{nbutlast } x22))$
apply *transfer*
by *auto*
 $(\text{metis lhd-LCons-ltl llist.collapse(1)},$

```

      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 nlength-nbutlast [simp]:
  nlength (nbutlast nell) = epred (nlength nell)
apply transfer
by (simp, simp add: epred-llength)

```

```

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*:

$x \in \text{nset } (\text{nbutlast } \text{nell}) \vee (\text{nfinite } \text{nell} \wedge \neg \text{is-NNil } \text{nell1} \wedge x \in \text{nset}(\text{nbutlast } \text{nell1})) \implies$
 $x \in \text{nset } (\text{nbutlast } (\text{nappend } \text{nell } \text{nell1}))$

unfolding *nbutlast-nappend*

by (*metis* (*full-types*) *Un-iff in-nset-nbutlastD nset-nappend*)

lemma *nnth-nbutlast*:

assumes $n \leq \text{nlength}(\text{nbutlast } \text{nell})$

shows $\text{nnth } (\text{nbutlast } \text{nell}) \ n = \text{nnth } \text{nell } \ n$

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)

3.4 nsubn

lemma *nsubn-def1*:

$\text{nsubn } \text{nell } \ i \ j = \text{ntaken } (j-i) \ (\text{ndropn } \ i \ \text{nell})$

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 $\text{nsubn } \text{nell } \ k \ k = (\text{NNil } (\text{nnth } \text{nell } \ k))$

unfolding *nsubn-def1*

by (*simp* *add: ndropn-nfirst*)

lemma *nsubn-nlength*:

$\text{nlength}(\text{nsubn } \text{nell } \ i \ j) = \min (j-i) \ (\text{nlength } \text{nell} - i)$

by (*simp* *add: nsubn-def1*)

lemma *nsubn-nlength-gr-one*:

assumes $k < n$

$n \leq \text{nlength } \text{nell}$

shows $0 < \text{nlength } (\text{nsubn } \text{nell } \ k \ n)$

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 $\text{nfinite } (\text{nsubn } \text{nell } \ k \ n)$

by (*simp* *add: nsubn-def1*)

lemma *nsubn-nnth*:

shows $\text{nnth } (\text{nsubn } \text{nell } \ i \ j) \ k = \text{nnth } \text{nell } \ (i + \min k (j - i))$

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 $\text{nnth } (\text{ntaken } ia \ (\text{ndropn } i \ \text{nell})) \ m = \text{nnth } (\text{ndropn } i \ (\text{ntaken } (ia+i) \ \text{nell})) \ m$
using *assms*
by (*simp add: ntaken-nnth*)

lemma *nellist-eq-nnth-eq*:

$(\text{nellx} = \text{nelly}) \longleftrightarrow \text{nlength } \text{nellx} = \text{nlength } \text{nelly} \wedge (\forall \ i \leq \text{nlength } \text{nellx}. \text{nnth } \text{nellx } i = \text{nnth } \text{nelly } i)$
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 $ia + i \leq \text{nlength } \text{nell}$
shows $(\text{ntaken } ia \ (\text{ndropn } i \ \text{nell})) = (\text{ndropn } i \ (\text{ntaken } (ia+i) \ \text{nell}))$
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 $n \leq \text{nlength } \text{nell}$
 $m + k \leq n$
shows $\text{ntaken } m \ (\text{nsubn } \text{nell } k \ n) = \text{nsubn } \text{nell } k \ (m+k)$
using *assms*
unfolding *nsubn-def1*
by *simp*

lemma *ndropn-nsubn*:

assumes $n \leq \text{nlength } \text{nell}$
 $m + k \leq n$
shows $\text{ndropn } m \ (\text{nsubn } \text{nell } k \ n) = \text{nsubn } \text{nell } (m+k) \ n$
proof –
have 1: $\text{ntaken } (n-k) \ (\text{ndropn } k \ \text{nell}) = \text{ndropn } k \ (\text{ntaken } n \ \text{nell})$
by (*metis add-leD2 assms(1) assms(2) diff-add ntaken-ndropn-swap plus-enat-simps(1)*)
have 2: $\text{ndropn } m \ (\text{ndropn } k \ (\text{ntaken } n \ \text{nell})) =$
 $\text{ndropn } (m+k) \ (\text{ntaken } n \ \text{nell})$
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 $n \leq \text{nlength } \text{nell}$
 $k \leq n$
shows $\text{ntl}(\text{nsubn } \text{nell } k \ n) = (\text{if } n=k \text{ then } (\text{NNil } (\text{nnth } \text{nell } k)) \text{ else } \text{nsubn } (\text{ntl } \text{nell}) \ k \ (n-1))$
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 $n1 \leq n2$
 $n0 \leq n4$

$n2 \leq n4 - n0$
 $n4 \leq n3$
 $n3 \leq \text{nlength } nell$
shows $(\text{nsbn } (\text{nsbn } nell \ n0 \ n3) \ n1 \ n2) = (\text{nsbn } (\text{nsbn } nell \ n0 \ n4) \ n1 \ n2)$
proof –
have 1: $\text{nlength}(\text{nsbn } nell \ n0 \ n3) = n3 - n0$
using *assms* **by** (*metis enat-minus-mono1 idiff-enat-enat min.orderE nsbn-nlength*)
have 2: $\text{nlength}(\text{nsbn } (\text{nsbn } nell \ n0 \ n3) \ n1 \ n2) = n2 - n1$
using *assms*
by (*metis Nat.le-diff-conv2 enat-minus-mono1 enat-ord-simps(1) idiff-enat-enat le-trans min.orderE nsbn-nlength*)
have 3: $\text{nlength}(\text{nsbn } nell \ n0 \ n4) = n4 - n0$
using *assms nsbn-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: $\text{nlength}(\text{nsbn } (\text{nsbn } nell \ n0 \ n4) \ n1 \ n2) = n2 - n1$
using *assms*
by (*metis 3 enat-minus-mono1 enat-ord-simps(1) idiff-enat-enat min.orderE nsbn-nlength*)
have 5: $\bigwedge i. i \leq (n3 - n0) \longrightarrow (\text{nnth } (\text{nsbn } nell \ n0 \ n3) \ i) = (\text{nnth } nell \ (n0 + i))$
using *assms* **by** (*simp add: nsbn-def1 ntaken-nnth*)
have 6: $\bigwedge i. i \leq (n2 - n1) \longrightarrow$
 $(\text{nnth } (\text{nsbn } (\text{nsbn } nell \ n0 \ n3) \ n1 \ n2) \ i) = (\text{nnth } (\text{nsbn } nell \ n0 \ n3) \ (n1 + i))$
using *assms* **by** (*simp add: Nat.le-diff-conv2 nsbn-nnth*)
have 7: $\bigwedge i. i \leq (n2 - n1) \longrightarrow$
 $(\text{nnth } (\text{nsbn } (\text{nsbn } nell \ n0 \ n3) \ n1 \ n2) \ i) = (\text{nnth } nell \ (n0 + (n1 + i)))$
using 5 6 *assms* **by** *auto*
have 8: $n0 \leq n4 \wedge n4 \leq \text{nlength } nell$
using *assms* **by** (*simp add: order-subst2*)
have 9: $\bigwedge i. i \leq (n4 - n0) \longrightarrow (\text{nnth } (\text{nsbn } nell \ n0 \ n4) \ i) = (\text{nnth } nell \ (n0 + i))$
using 8 **by** (*simp add: nsbn-def1 ntaken-nnth*)
have 10: $\bigwedge i. i \leq (n2 - n1) \longrightarrow$
 $(\text{nnth } (\text{nsbn } (\text{nsbn } nell \ n0 \ n4) \ n1 \ n2) \ i) = (\text{nnth } (\text{nsbn } nell \ n0 \ n4) \ (n1 + i))$
by (*simp add: nsbn-def1 ntaken-nnth*)
have 11: $\bigwedge i. i \leq (n2 - n1) \longrightarrow$
 $(\text{nnth } (\text{nsbn } (\text{nsbn } nell \ n0 \ n4) \ n1 \ n2) \ i) = (\text{nnth } nell \ (n0 + (n1 + i)))$
by (*metis 10 9 Nat.le-diff-conv2 add commute assms(1) assms(3) le-trans*)
have 12: $\bigwedge i. i \leq (n2 - n1) \longrightarrow$
 $(\text{nnth } (\text{nsbn } (\text{nsbn } nell \ n0 \ n3) \ n1 \ n2) \ i) = (\text{nnth } (\text{nsbn } (\text{nsbn } nell \ n0 \ n4) \ n1 \ n2) \ i)$
by (*simp add: 11 7*)
from 12 2 4 **show** *?thesis*
using *nellist-eq-nnth-eq[of (nsbn (nsbn nell n0 n3) n1 n2) (nsbn (nsbn nell n0 n4) n1 n2)]*
enat-ord-simps(1) **by** *presburger*
qed

lemma *nsbn-nsbn-1:*

assumes $n1 \leq n2$
 $n0 \leq n3$
 $n2 \leq n3 - n0$
 $n3 \leq \text{nlength } nell$

shows $(\text{nsbn } (\text{nsbn nell } n0 \ n3) \ n1 \ n2) = (\text{nsbn nell } (n0+n1) \ (n0+n2))$
proof –
have 0: $\text{nlength}(\text{nsbn } (\text{nsbn nell } n0 \ n3) \ n1 \ n2) = n2 - n1$
using *assms*
by (*metis* *enat-minus-mono1 idiff-enat-enat min.orderE nsbn-nlength of-nat-eq-enat of-nat-mono*)
have 1: $\text{nlength}(\text{nsbn nell } (n1+n0) \ (n2+n0)) = (n2+n0) - (n1+n0)$
using *assms nsbn-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: $\bigwedge i. i \leq (n2-n1) \longrightarrow$
 $(\text{nnth } (\text{nsbn } (\text{nsbn nell } n0 \ n3) \ n1 \ n2) \ i) = (\text{nnth nell } (n0+(n1+i)))$
using *assms* **by** (*simp add: nsbn-nnth*)
have 3: $\bigwedge i. i \leq (n2-n1) \longrightarrow$
 $(\text{nnth } (\text{nsbn nell } (n0+n1) \ (n0+n2)) \ i) = (\text{nnth nell } (n0+(n1+i)))$
using *assms* **by** (*metis* 10 *add.assoc add.commute min.orderE nsbn-nnth*)
have 4: $\bigwedge i. i \leq (n2-n1) \longrightarrow$
 $(\text{nnth } (\text{nsbn } (\text{nsbn nell } n0 \ n3) \ n1 \ n2) \ i) = (\text{nnth } (\text{nsbn nell } (n0+n1) \ (n0+n2)) \ i)$
by (*simp add: 2 3*)
show *?thesis* **using** *nellist-eq-nnth-eq*[*of (nsbn (nsbn nell n0 n3) n1 n2)*
 $(\text{nsbn nell } (n0+n1) \ (n0+n2))$]
0 10 4 1
by (*metis* *add.commute enat-ord-simps(1)*)
qed

lemma *nsbn-eq*:

assumes $n \leq m$
 $m \leq \text{nlength nellx}$
 $m \leq \text{nlength nelly}$
 $\forall j. n \leq j \wedge j \leq m \longrightarrow \text{nnth nellx } j = \text{nnth nelly } j$

shows $\text{nsbn nellx } n \ m = \text{nsbn nelly } n \ m$

proof –

have 1: $\text{nlength } (\text{nsbn nellx } n \ m) = \text{nlength } (\text{nsbn nelly } n \ m)$
using *assms* **by** (*metis* *enat-minus-mono1 min-def nsbn-nlength*)
have 2: $\bigwedge j. j \leq \text{nlength } (\text{nsbn nellx } n \ m) \longrightarrow \text{nnth } (\text{nsbn nellx } n \ m) \ j = \text{nnth } (\text{nsbn nelly } n \ m) \ j$
using *assms* **by** (*simp add: Nat.le-diff-conv2 nsbn-nlength nsbn-nnth*)
show *?thesis*
by (*simp add: 1 2 nellist-eq-nnth-eq*)

qed

3.5 nfuse

lemma *nfuse-def1*:

$\text{nfuse nellx nelly} = (\text{if is-NNil nelly then nellx else nappend nellx (ntl nelly)})$

apply *transfer*

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

lemma *nfuse-NCons-a*:

$\text{nfuse } (NCons \ x \ \text{nellx}) \ \text{nelly} = (NCons \ x \ (\text{nfuse nellx nelly}))$

by (*simp add: nfuse-def1*)

lemma *nfuse-NCons-b*:

nfuse nellx (NCons y nelly) = nappend nellx nelly

by (*simp add: nfuse-def1*)

lemma *nfuse-simps [simp]*:

shows *nhd-nfuse: nhd(nfuse nellx nelly) =*
(if is-NNil nellx then
(if is-NNil nelly then nhd nellx else nlast nellx)
else nhd nellx)

and *ntl-nfuse: ntl(nfuse nellx nelly) =*
(if is-NNil nellx then
(if is-NNil nelly then ntl nellx else ntl nelly)
else (if is-NNil nelly then ntl nellx
else nappend (ntl nellx) (ntl nelly)))

by (*simp-all add: nfuse-def1*)

lemma *nfuse-nbutlast*:

assumes *nlast nellx = nfirst nelly*
¬ is-NNil nellx
¬ 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 ≤ nlength (nfuse nellx nelly)*

nlast nellx = nfirst nelly

shows *(i ≤ nlength nellx → nnth (nfuse nellx nelly) i = nnth nellx i)*

∧

(nlength nellx < i ∧ i ≤ nlength (nfuse nellx nelly) →
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 ≤ 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 (nfirst nell)) nell = nell
by (simp add: nfuse-nappend)

```

```

lemma nfuse-rightneutral :
  nfuse nell (NNil (nlast nell)) = nell
unfolding nfuse-def
by simp

```

```

lemma nfuse-nfuse :
assumes nlast nellx = nfirst nelly
shows nfirst (nfuse nellx nelly) = nfirst nellx
using assms unfolding nfuse-def1 by transfer auto

```

```

lemma nlast-nfuse :
assumes nlast nellx = nfirst nelly
      nfinite nellx
shows nlast (nfuse nellx nelly) = nlast nelly
using assms unfolding nfuse-def1 by transfer (auto, metis lhd-LCons-ltl 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-ltl)

```

```

lemma ntaken-nfuse :
assumes nlast nellx = nfirst nelly
      nfinite nellx

```

shows $(\text{ntaken } (\text{the-enat } (\text{nlength } \text{nellx})) (\text{nfuse } \text{nellx } \text{nelly})) = \text{nellx}$
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 $\text{nlast } \text{nellx} = \text{nfirst } \text{nelly}$
 $\text{nfinite } \text{nellx}$
shows $(\text{ndropn } (\text{the-enat } (\text{nlength } \text{nellx})) (\text{nfuse } \text{nellx } \text{nelly})) = \text{nelly}$
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 $n \leq \text{nlength } \text{nell}$
shows $\text{nlength } (\text{nfuse } (\text{ntaken } n \text{ nell}) (\text{ndropn } n \text{ nell})) = \text{nlength } \text{nell}$
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 $n \leq \text{nlength } \text{nell}$
 $i \leq \text{nlength } \text{nell}$
shows $\text{nnth } (\text{nfuse } (\text{ntaken } n \text{ nell}) (\text{ndropn } n \text{ nell})) i = \text{nnth } \text{nell } i$
using *assms*
nfuse-nnth[of i (ntaken n nell) (ndropn n nell)]
nfuse-ntaken-ndropn-nlength[of n nell]
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 $n \leq \text{nlength } \text{nell}$
shows $\text{nfuse } (\text{ntaken } n \text{ nell}) (\text{ndropn } n \text{ nell}) = \text{nell}$
using *assms*
by (*simp add: nfuse-ntaken-ndropn-nlength nfuse-ntaken-ndropn-nnth nellist-eq-nnth-eq*)

lemma *nfuse-nnth-var*:
assumes $\text{enat } i \leq \text{nlength } (\text{nfuse } \text{nellx } \text{nelly})$

$nlast\ nelly = nfirst\ nelly$
shows $(enat\ i \leq nlength\ nellx \longrightarrow nnth\ (nfuse\ nellx\ nelly)\ i = nnth\ nellx\ i) \wedge$
 $(nlength\ nellx \leq enat\ i \wedge enat\ i \leq nlength\ (nfuse\ nellx\ nelly) \longrightarrow$
 $nnth\ (nfuse\ nellx\ nelly)\ i = nnth\ nelly\ (i - the-enat\ (nlength\ nellx)))$
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 $nset-nfuse$:
 $nset\ (nfuse\ nellx\ nelly) =$
 $(if\ nfinite\ nellx\ then$
 $(if\ is-NNil\ nelly\ then\ nset\ nellx\ else\ nset\ nellx \cup nset\ (ntl\ nelly))$
 $else\ nset\ nellx)$
by $(simp\ add: nfuse-def1\ nset-nappend)$

lemma $nsubn-nfuse$:
assumes $(enat\ k) \leq n$
 $(enat\ n) \leq m$
 $(enat\ m) \leq nlength\ nell$
shows $nfuse\ (nsubn\ nell\ k\ n)\ (nsubn\ nell\ n\ m) = (nsubn\ nell\ k\ m)$
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: $i \leq (m-k) \longrightarrow \text{nnth } (\text{nfuse } (\text{nsubn nell } k \ n) \ (\text{nsubn nell } n \ m)) \ i =$
 $(\text{if } i \leq (n-k) \text{ then } (\text{nnth nell } (k+i)) \text{ else } (\text{nnth nell } (n+(i-(n-k)))))$
using *assms* 42 5 **unfolding** *nsubn-def1*
by (*auto simp add: ntaken-nnth*)
 $(\text{metis enat-ord-simps}(1) \text{ min.bounded-iff},$
 $\text{metis enat-ord-simps}(1) \text{ min.bounded-iff})$
have 44: $i \leq (m-k) \longrightarrow \text{nnth } (\text{nfuse } (\text{nsubn nell } k \ n) \ (\text{nsubn nell } n \ m)) \ i =$
 $(\text{nnth nell } (k+i))$
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: $(\forall i. i \leq \text{nlength}(\text{nsubn nell } k \ m) \longrightarrow (\text{nnth } (\text{nsubn nell } k \ m) \ i) = (\text{nnth nell } (k+i)))$
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*:

$\text{nmap } f \ (\text{nfuse nellx nelly}) = \text{nfuse } (\text{nmap } f \ \text{nellx}) \ (\text{nmap } f \ \text{nelly})$
by (*simp add: nfuse-def1 nmap-nappend-distrib*)

3.6 nridx and nidx

lemma *nridx-nidx*:

$\text{nridx } (<) \ \text{nell} = \text{nidx nell}$
apply *transfer*
by (*simp add: ridx-lidx*)

lemma *nridx-expand*:

$\text{nridx } R \ \text{nell} \longleftrightarrow (\forall i. (\text{Suc } i) \leq \text{nlength nell} \longrightarrow R \ (\text{nnth nell } i) \ (\text{nnth nell } (\text{Suc } i)))$
by *transfer*
 $(\text{auto simp add: min-def Suc-ile-eq},$
 $\text{metis co.enat.exhaust-sel eSuc-enat ileI1 ileSS-Suc-eq llength-eq-0 ridx-def},$
 $\text{metis Extended-Nat.eSuc-mono co.enat.exhaust-sel eSuc-enat llength-eq-0 order-le-less ridx-def})$

lemma *nidx-expand*:

$\text{nidx nell} \longleftrightarrow (\forall i. (\text{Suc } i) \leq \text{nlength nell} \longrightarrow \text{nnth nell } i < \text{nnth nell } (\text{Suc } i))$
using *nridx-nidx nridx-expand* **by** *blast*

lemma *nridx-NCons*:

$\text{nridx } R \ (\text{NCons } x \ \text{nell}) \longleftrightarrow$
 $(\forall n. (\text{Suc } n) \leq \text{eSuc } (\text{nlength nell}) \longrightarrow R \ (\text{nnth } (\text{NCons } x \ \text{nell}) \ n) \ (\text{nnth } (\text{NCons } x \ \text{nell}) \ (\text{Suc } n)))$
using *nridx-expand* [of $R \ (\text{NCons } x \ \text{nell})$]
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 ($metis \ 1 \ One\text{-}nat\text{-}def \ add\text{-}diff\text{-}cancel\text{-}left' \ eSuc\text{-}ile\text{-}mono \ enat\text{-}le\text{-}plus\text{-}same(1) \ gen\text{-}nlength\text{-}def \ le\text{-}SucE \ le\text{-}add1$)

$ndropn\text{-}Suc\text{-}NCons \ ndropn\text{-}nfirst \ ndropn\text{-}nfuse \ nfuse\text{-}leftneutral \ nlast\text{-}NNil \ nlength\text{-}NNil \ nlength\text{-}code$
 $nlength\text{-}eq\text{-}enat\text{-}nfiniteD \ nnth\text{-}0 \ one\text{-}eSuc \ one\text{-}enat\text{-}def \ plus\text{-}1\text{-}eq\text{-}Suc \ the\text{-}enat\text{-}0 \ zero\text{-}enat\text{-}def$

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 ($metis \ eSuc\text{-}enat \ eSuc\text{-}ile\text{-}mono$)

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 ($metis \ add\text{-}commute \ add\text{-}right\text{-}neutral \ diff\text{-}add \ le\text{-}add1 \ nnth\text{-}Suc\text{-}NCons \ plus\text{-}1\text{-}eq\text{-}Suc$)

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 ($metis \ diff\text{-}Suc\text{-}1$)

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 ($metis \ nridx\text{-}LCons\text{-}conv$)

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 $nridx\ R\ nell$

$n \leq \text{nlength } \text{nell}$
shows $\text{nridx } R (\text{ntake } n \text{ nell})$
using *assms*
by transfer
(metis co.enat.exhaust-sel eSuc-ile-mono llength-eq-0 ridx-ltake-a)

lemma *nidx-ntake-a*:
assumes $\text{nidx } \text{nell}$
 $n \leq \text{nlength } \text{nell}$
shows $\text{nidx } (\text{ntake } n \text{ nell})$
using *assms*
using *nridx-ntake-a nridx-nidx* **by** *blast*

lemma *nridx-nappend-nfinite*:
assumes $\text{nfinite } \text{nell1}$
shows $\text{nridx } R (\text{nappend } \text{nell1 } \text{nell2}) \longleftrightarrow$
 $\text{nridx } R \text{ nell1} \wedge (R (\text{nlast } \text{nell1}) (\text{nfirst } \text{nell2})) \wedge \text{nridx } R \text{ nell2}$
using *assms*
by transfer
(simp add: ridx-lappend-lfinite)

lemma *nidx-nappend-nfinite*:
assumes $\text{nfinite } \text{nell1}$
shows $\text{nidx } (\text{nappend } \text{nell1 } \text{nell2}) \longleftrightarrow$
 $\text{nidx } \text{nell1} \wedge ((\text{nlast } \text{nell1}) < (\text{nfirst } \text{nell2})) \wedge \text{nidx } \text{nell2}$
using *assms*
by *(metis nridx-nappend-nfinite nridx-nidx)*

lemma *nidx-nfuse*:
assumes $\text{nfinite } \text{nell1}$
 $\text{nidx } \text{nell1}$
 $\text{nnth } \text{nell1 } 0 = 0$
 $\text{nidx } \text{nell2}$
 $\text{nnth } \text{nell2 } 0 = \text{nlast } \text{nell1}$
shows $\text{nidx } (\text{nfuse } \text{nell1 } \text{nell2})$
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 illess-Suc-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 ≤ nlength nell
  shows nridx R (ndropn n nell)
using assms
by transfer
  (metis co-enat.exhaust-sel illess-Suc-eq ldrop-enat llength-eq-0 min.orderE nless-le
    ridx-ldrop the-enat.simps)

```

```

lemma nidx-ndropn:
  assumes nidx nell
    n ≤ nlength nell
  shows nidx (ndropn n nell)
using assms
using nridx-ndropn nridx-nidx by blast

```

```

lemma nridx-ntake-all:
  assumes  $\bigwedge n. n \leq nlength\ nell \implies nridx\ R\ (ntake\ (enat\ n)\ nell)$ 
  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  $\bigwedge n. n \leq nlength\ nell \implies nidx\ (ntake\ (enat\ n)\ nell)$ 
  shows nidx nell
using assms
using nridx-nidx nridx-ntake-all by blast

```

```

lemma nridx-ntake:
  assumes nridx R (ntake n nell)
    n ≤ nlength nell
    k ≤ 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 ≤ 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 $ndistinct$ $nell$
 $x \in nset$ $nell$
shows $\exists!i. i \leq nlength$ $nell \wedge (nnth$ $nell$ $i) = x$
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 $nidx$ $nellx$
 $nidx$ $nelly$
 $nset$ $nellx = nset$ $nelly$
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 $nell1$
 $(nnth$ $nell1$ $0) = (0::nat)$
 $nridx$ R $nell2$
 $(nnth$ $nell2$ $0) = 0$
 $nfinite$ $nell1$
 $nfinite$ $nell$
 $nlast$ $nell1 = cp$
shows $nlast$ $nell1 = nfirst(nmap$ $(\lambda x. x+cp)$ $nell2)$
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 $nidx$ $nell1$
 $(nnth$ $nell1$ $0) = (0::nat)$
 $nidx$ $nell2$
 $(nnth$ $nell2$ $0) = 0$
 $nfinite$ $nell1$

$nfinite\ nell$
 $nlast\ nell1 = cp$
shows $nlast\ nell1 = nfirst(nmap\ (\lambda x. x+cp)\ nell2)$
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 $nridx\ R\ nell1$
 $(nnth\ nell1\ 0) = 0$
 $nridx\ R\ nell2$
 $(nnth\ nell2\ 0) = 0$
 $nfinite\ nell1$
 $nfinite\ nell2$
 $nfinite\ nell$
 $nlast\ nell1 = cp$
 $nlast\ nell2 = the-enat((nlength\ nell)) - cp$
 $i \leq (nlength\ nell2)$
 $cp \leq nlength\ nell$
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 ≤ ((nlength nell1) + (nlength nell2))
  the-enat((nlength nell1)) ≤ i
  cp ≤ nlength nell
shows nnth (nfuse nell1 (nmap (λx. x+cp) nell2)) (i) = cp + (nnth nell2 (i - the-enat((nlength nell1))))

proof -
have 1: i = (the-enat ( (nlength nell1)) + (i - the-enat ( (nlength nell1))))
  by (simp add: assms)
have 2: enat ((i::nat) - the-enat ( (nlength nell1))) ≤ nlength 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
  (nnth nell1 0) = (0::nat)
  nidx nell2
  (nnth nell2 0) = 0
  nfinite nell1
  nfinite nell2
  nfinite nell
  nlast nell1 = cp
  nlast nell2 = the-enat((nlength nell)) - cp
  i ≤ ((nlength nell1) + (nlength nell2))
  the-enat((nlength nell1)) ≤ i
  cp ≤ nlength nell
shows nnth (nfuse nell1 (nmap (λx. x+cp) nell2)) (i) = cp + (nnth nell2 (i - the-enat((nlength nell1))))

```

using assms nridx-nidx nridx-nfuse-nnth-cp-a **by** blast

lemma nridx-nfuse-nnth-cp-nlast:

```

assumes nridx R nell1
  (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

```


$i \leq (\text{nlength } \text{nell2})$
 $cp \leq \text{nlength } \text{nell}$
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: $cp + (\text{nlast } \text{nell2}) = (\text{the-enat } ((\text{nlength } \text{nell})))$
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*
 $(\text{nnth } \text{nell1 } 0) = 0$
nidx nell2
 $(\text{nnth } \text{nell2 } 0) = 0$
nfinite nell1
nfinite nell2
nfinite nell
 $\text{nlast } \text{nell1} = cp$
 $\text{nlast } \text{nell2} = \text{the-enat } ((\text{nlength } \text{nell})) - cp$
 $i \leq (\text{nlength } \text{nell2})$
 $cp \leq \text{nlength } \text{nell}$
shows $\text{nlast } (\text{nfuse } \text{nell1 } (\text{nmap } (\lambda x. x+cp) \text{ nell2})) = (\text{the-enat } ((\text{nlength } \text{nell})))$
using *assms nridx-nidx nridx-nfuse-nnth-cp-nlast* **by** *blast*

lemma *nridx-nfuse-nnth-cp-infinite:*

assumes *nridx R nell1*
 $(\text{nnth } \text{nell1 } 0) = (0::\text{nat})$
nridx R nell2
 $(\text{nnth } \text{nell2 } 0) = 0$
nfinite nell1
 $\neg \text{nfinite } \text{nell2}$
 $\neg \text{nfinite } \text{nell}$
 $\text{nlast } \text{nell1} = cp$
shows $\text{nnth } (\text{nfuse } \text{nell1 } (\text{nmap } (\lambda x. x+cp) \text{ nell2})) (\text{the-enat}((\text{nlength } \text{nell1})) + i) = cp + (\text{nnth } \text{nell2 } i)$
proof –
have 1: $\text{nlast } \text{nell1} = \text{nfirst}(\text{nmap } (\lambda x. x+cp) \text{ nell2})$
by (*metis assms(1) assms(2) assms(3) assms(4) assms(5) assms(8) nridx-nfuse-nfirst-nlast*)
have 2: $\text{nnth } (\text{nfuse } \text{nell1 } (\text{nmap } (\lambda x. x+cp) \text{ nell2})) (\text{the-enat}((\text{nlength } \text{nell1})) + 0) = \text{nlast } \text{nell1}$
using *assms* **by** (*metis 1 add.right-neutral ntaken-nfuse ntaken-nlast*)
have 3: $\text{nfirst}(\text{nmap } (\lambda x. x+cp) \text{ nell2}) = cp + (\text{nnth } \text{nell2 } 0)$
using 1 *assms* **by** *auto*
have 8: $i \leq (\text{nlength } \text{nell2})$
by (*metis assms(6) enat-ile nfinite-conv-nlength-enat wlog-linorder-le*)

```

have 10: (nlength nell1) ≤ enat (the-enat ( (nlength nell1)) + (i::nat))
  using assms(5) nfinite-nlength-enat by fastforce
have 11: enat (the-enat ( (nlength nell1)) + i) ≤ nlength (nfuse nell1 (nmap (λx::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 (λx. x+cp) nell2)) (the-enat((nlength nell1)) + i) =
  (nnth (nmap (λx. 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: ∧j. j ≤ nlength nell1 ⇒ nnth (nfuse nell1 (nmap (λx. x+ cp) nell2)) j = nnth nell1 j
  using assms by (simp add: 1 add-increasing2 nfuse-nlength nfuse-nnth)
have 40: ∃ 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  $5$ :  $\bigwedge j. (nlength\ nell1) \leq j \wedge j \leq (nlength\ nell1) + nlength\ nell2 \implies$ 
   $nnt\h (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ j =$ 
   $cp + (nnt\h\ nell2\ (j - the-enat((nlength\ nell1))))$ 
  using assms nidx-nfuse-nnt\h-cp-a[of nell1 nell2 nell cp]
  by (metis 41 enat-ord-simps(1) the-enat.simps)
have  $45$ :  $\bigwedge j. k1 \leq j \wedge j \leq (enat\ (k1)) + nlength\ nell2 \implies$ 
   $nnt\h (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ j =$ 
   $cp + (nnt\h\ nell2\ (j - (k1)))$ 
  by (simp add: 41 5 order-less-imp-le)
have  $50$ :  $nlength(nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2)) = (nlength\ nell1) + nlength\ nell2$ 
  by (simp add: nfuse-nlength)
have  $51$ :  $\bigwedge j. enat\ (Suc\ j) \leq nlength\ nell1 \implies$ 
   $(nnt\h (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ j) <$ 
   $(nnt\h (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ (Suc\ j))$ 
  using  $4$  Suc-ile-eq assms(1) nidx-expand by auto
have  $52$ :  $\bigwedge j. k1 \leq j \wedge (Suc\ j) \leq enat\ (k1) + nlength\ nell2 \implies$ 
   $cp + (nnt\h\ nell2\ (j - (k1))) <$ 
   $cp + (nnt\h\ nell2\ ((Suc\ j) - (k1)))$ 
  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  $6$ :  $\bigwedge j. enat\ (Suc\ j) \leq nlength(nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2)) \implies$ 
   $(nnt\h (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ j) <$ 
   $(nnt\h (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ (Suc\ j))$ 
  proof –
    fix  $j$ 
    assume  $a$ :  $enat\ (Suc\ j) \leq nlength(nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))$ 
    show  $(nnt\h (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ j) < (nnt\h (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\$ 
   $nell2))\ (Suc\ j))$ 
    proof –
      have  $61$ :  $enat\ (Suc\ j) \leq nlength\ nell1 \implies$ 
       $(nnt\h (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ j) < (nnt\h (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\$ 
       $(Suc\ j))$ 
      using  $51$  by blast
      have  $62$ :  $k1 \leq j \wedge (Suc\ j) \leq nlength(nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2)) \implies$ 
       $(nnt\h (nfuse\ nell1\ (nmap\ (\lambda x. x + cp)\ nell2))\ j) < (nnt\h (nfuse\ nell1\ (nmap\ (\lambda x. 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 $nidx\ nell1$
 $nnth\ nell1\ 0 = 0$
 $nidx\ nell2$
 $nnth\ nell2\ 0 = 0$
 $nfinite\ nell1$
 $nlast\ nell1 = cp$
 $\neg nfinite\ nell2$
 $\neg nfinite\ nell$
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

nlast nl = (nlength nell)

(Suc i) ≤ (nlength nl)

shows *nfuse (nsubn nell (nnth nl i) (nnth nl (Suc i))) (nsubn nell (nnth nl (Suc i)) (nlast nl)) =
(nsubn nell (nnth nl i) (nlast nl))*

proof –

have 1: *(nnth nl i) ≤ (nnth nl (Suc i))*

by (*simp add: assms(1) assms(5) nidx-less-eq*)

have 2: *nlast nl = (nnth nl (the-enat((nlength nl))))*

by (*simp add: assms(2) nnth-nlast*)

have 3: *1 ≤ 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)) ≤ (nlast 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)) ≤ nlength nell*

by (*metis 4 assms(4) enat-ord-simps(1)*)

have 6: *nlast (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)) (nlast 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 *nlast 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 ≤ nlength nell
shows   nnth nell j ≤ nlast 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 (λx .x- k) nell) ∧ nnth (nmap (λx .x- k) nell) 0 = 0 ∧ k ≤ (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 ≤ n
          n ≤ nlength nell
          nidx nell
          nnth nell 0 = 0
shows   nidx (nsubn nell k n) ∧ 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)) =
  (∀ i::nat.
    enat (Suc i) ≤ nlength (ntaken n (niterates Suc (0::nat))) →
    nnth (ntaken n (niterates Suc (0::nat))) i <
    nnth (ntaken n (niterates Suc (0::nat))) (Suc i))
  unfolding nidx-expand by auto
have 2: (∀ i::nat.
    enat (Suc i) ≤ nlength (ntaken n (niterates Suc (0::nat))) →
    nnth (ntaken n (niterates Suc (0::nat))) i <
    nnth (ntaken n (niterates Suc (0::nat))) (Suc i))
proof
fix i
show enat (Suc i) ≤ nlength (ntaken n (niterates Suc (0::nat))) →
  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

```

3.7 nlastnfirst

lemma *nlastnfirst-def2*:
 $nlastnfirst = llastlfirst \circ (lmap\ llist\text{-of}\text{-nellist} \circ llist\text{-of}\text{-nellist})$
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 $?n2l = (lmap\ llist\text{-of}\text{-nellist} \circ llist\text{-of}\text{-nellist})$
have 1: $nlastnfirst\ (NCons\ nell\ nells) =$
 $(llastlfirst \circ ?n2l)\ (NCons\ nell\ nells)$
by (*simp add: nlastnfirst-def2*)
have 2: $(llastlfirst \circ ?n2l)\ (NCons\ nell\ nells) =$
 $(llast\ (llist\text{-of}\text{-nellist}\ nell) = lfirst\ (lfirst\ (?n2l\ nells)) \wedge llastlfirst\ (?n2l\ nells))$
by *simp*
have 3: $llastlfirst\ (?n2l\ nells) = nlastnfirst\ nells$
by (*simp add: nlastnfirst.rep-eq*)
have 4: $llast\ (llist\text{-of}\text{-nellist}\ nell) = nlast\ nell$
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 $nlastnfirst\ nells =$
 $(\forall i. (Suc\ i) \leq nlength\ nells \longrightarrow nlast(nnth\ nells\ i) = nfirst(nnth\ nells\ (Suc\ i)))$
proof –
let $?n2l = (lmap\ llist\text{-of}\text{-nellist} \circ llist\text{-of}\text{-nellist})$
have 1: $nlastnfirst\ nells = (llastlfirst \circ ?n2l)\ nells$
by (*simp add: nlastnfirst-def2*)
have 2: $(llastlfirst \circ ?n2l)\ nells = llastlfirst\ (?n2l\ nells)$
by *simp*

have 3: $llastlfirst (?n2l\ nells) \longleftrightarrow$
 $(\forall i. (Suc\ i) < llength\ (?n2l\ nells) \longrightarrow$
 $llast\ (lnth\ (?n2l\ nells)\ i) = lfirst\ (lnth\ (?n2l\ nells)\ (Suc\ i)))$
using *llastlfirst-def* **by** *blast*
have 4: $epred\ (llength\ (?n2l\ nells)) = nlength\ nells$
by *simp*
have 5: $\bigwedge xs\ j. j \leq nlength\ xs \longrightarrow nnth\ xs\ j = lnth\ (llist-of-nellist\ xs)\ j$
unfolding *nnth-def* **by** *auto*
have 6: $\bigwedge lx\ j. j < llength\ lx \wedge \neg lnull\ lx \longrightarrow lnth\ lx\ j = nnth\ (nellist-of-llist\ lx)\ j$
by (*metis* 5 *co.enat.exhaust-sel* *iless-Suc-eq* *llength-eq-0* *nellist-of-llist-inverse* *nlength.abs-eq*)
have 7: $\bigwedge xs. nlast\ xs = llast\ (llist-of-nellist\ xs)$
by (*metis* *llast-linfinite* *nfinite-def* *nlast-llast* *nlast-not-nfinite*)
have 8: $\bigwedge xs. nfirst\ xs = lfirst\ (llist-of-nellist\ xs)$
by (*simp* *add: lfirst-def* *llist-of-nellist.simps(2)* *nfirst-def*)
have 9: $\bigwedge lx. \neg lnull\ lx \implies llast\ lx = nlast\ (nellist-of-llist\ lx)$
by (*simp* *add: 7*)
have 10: $\bigwedge lx. \neg lnull\ lx \implies lfirst\ lx = nfirst\ (nellist-of-llist\ lx)$
by (*simp* *add: 8*)
have 11: $\bigwedge j. j < llength\ (?n2l\ nells) \longrightarrow$
 $llast\ (lnth\ (?n2l\ nells)\ j) =$
 $nlast\ (nellist-of-llist\ (lnth\ (?n2l\ nells)\ j))$
by (*simp* *add: 9*)
have 12: $\bigwedge j. (enat\ j) < llength\ (?n2l\ nells) \longrightarrow$
 $nlast\ (nellist-of-llist\ (lnth\ (?n2l\ nells)\ j)) =$
 $nlast\ (nellist-of-llist\ (llist-of-nellist\ (lnth\ (llist-of-nellist\ nells)\ j)))$
by *simp*
have 13: $\bigwedge j. (enat\ j) < llength\ (?n2l\ nells) \longrightarrow$
 $nlast\ (nellist-of-llist\ (llist-of-nellist\ (lnth\ (llist-of-nellist\ nells)\ j))) =$
 $nlast\ (lnth\ (llist-of-nellist\ nells)\ j)$
by *auto*
have 14: $\bigwedge j. (enat\ j) < llength\ (?n2l\ nells) \longrightarrow$
 $nlast\ (lnth\ (llist-of-nellist\ nells)\ j) = nlast\ (nnth\ nells\ j)$
by (*simp* *add: 6*)
have 15: $\bigwedge j. j < llength\ (?n2l\ nells) \longrightarrow$
 $llast\ (lnth\ (?n2l\ nells)\ j) =$
 $nlast\ (nnth\ nells\ j)$
using 11 12 13 14 **by** *presburger*
have 16: $\bigwedge j. (Suc\ j) < llength\ (?n2l\ nells) \longrightarrow$
 $lfirst\ (lnth\ (?n2l\ nells)\ (Suc\ j)) =$
 $nfirst\ (nellist-of-llist\ (lnth\ (?n2l\ nells)\ (Suc\ j)))$
by (*simp* *add: 10*)
have 17: $\bigwedge j. (Suc\ j) < llength\ (?n2l\ nells) \longrightarrow$
 $nfirst\ (nellist-of-llist\ (lnth\ (?n2l\ nells)\ (Suc\ j))) =$
 $nfirst\ (nellist-of-llist\ (llist-of-nellist\ (lnth\ (llist-of-nellist\ nells)\ (Suc\ j))))$
by *simp*
have 18: $\bigwedge j. (Suc\ j) < llength\ (?n2l\ nells) \longrightarrow$
 $nfirst\ (nellist-of-llist\ (llist-of-nellist\ (lnth\ (llist-of-nellist\ nells)\ (Suc\ j)))) =$
 $nfirst\ (lnth\ (llist-of-nellist\ nells)\ (Suc\ j))$
by *auto*
have 19: $\bigwedge j. (Suc\ j) < llength\ (?n2l\ nells) \longrightarrow$


```

      nfirst (lnth (llist-of-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 nellxs
  shows nlastnfirst (nappend nellxs nellys)  $\longleftrightarrow$ 
    nlastnfirst nellxs  $\wedge$  nlastnfirst nellys  $\wedge$  nlast(nlast nellxs) = nfirst(nfirst nellys)
using assms
by (metis (mono-tags, lifting) nlastnfirst-nridx nridx-nappend-nfinite)

```

3.8 nfusecat

lemma *nfusecat-def2*:

```

  nfusecat = nellist-of-llist  $\circ$  lfusecat  $\circ$  (lmap llist-of-nellist  $\circ$  llist-of-nellist)
by (simp add: map-fun-def nfusecat-def)

```

lemma *not-null-lfusecat*:

```

   $\neg lnull((lfusecat \circ (lmap\ llist-of-nellist \circ llist-of-nellist))\ nells)$ 
by (simp add: llist-of-nellist.code)

```

lemma *nfusecat-def3*:

```

  ((llist-of-nellist  $\circ$  nfusecat) nells) =
  ((lfusecat  $\circ$  (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  $\circ$  nfusecat =
    llist-of-nellist  $\circ$  nellist-of-llist  $\circ$  lfusecat  $\circ$  ?n2l
    by (simp add: nfusecat-def2 rewriteL-comp-comp)

```

have 2: $\neg \text{lnull}((\text{lfusecat} \circ ?n2l) \text{ nells})$
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]*:
 $\text{nfusecat} (\text{NNil } \text{nell}) = \text{nell}$
apply *transfer*
by *simp*

lemma *nfusecat-NCons[simp]*:
 $\text{nfusecat} (\text{NCons } \text{nell } \text{nells}) = \text{nfuse } \text{nell} (\text{nfusecat } \text{nells})$
apply *transfer*
by (*simp add: lfuse-def llist.case-eq-if*)

lemma *nfusecat-nfirst*:
assumes $(\exists x1. \text{nells} = \text{NNil } x1)$
shows $\text{nfusecat } \text{nells} = \text{nfirst } \text{nells}$
proof –
obtain *nell* **where** 1: $\text{nells} = \text{NNil } \text{nell}$
using *assms* **by** *auto*
have 2: $\text{nfusecat } \text{nells} = \text{nell}$
by (*simp add: 1*)
have 3: $\text{nfirst } \text{nells} = \text{nell}$
by (*metis* 1 *ndropn-0 ndropn-nfirst nnth-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 $\text{nfusecat } \text{nells} = \text{nfuse } (\text{nfirst } \text{nells}) (\text{nfusecat } (\text{ntl } \text{nells}))$
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*:
 $\text{nfusecat } \text{nells} =$
 $(\text{if } \text{is-NNil } \text{nells} \text{ then } \text{nfirst } \text{nells} \text{ else } \text{nfuse } (\text{nfirst } \text{nells}) (\text{nfusecat } (\text{ntl } \text{nells})))$
unfolding *is-NNil-def*
using *nfusecat-NCons-nfirst nfusecat-nfirst* **by** *simp blast*

lemma *nfusecat-expand-case*:
 $\text{nfusecat } \text{nells} = (\text{case } \text{nells} \text{ of } (\text{NNil } \text{nell}) \Rightarrow \text{nell} \mid$
 $(\text{NCons } \text{nell}' \text{ nells1}) \Rightarrow \text{nfuse } \text{nell}' (\text{nfusecat } \text{nells1}))$

by (metis is-NNil-def nellist.case-eq-if nellist.collapse(2) nlast-NNil nfusecat-NCons nfusecat-NNil)

lemma *nmap-nfusecat*:

$nmap\ f\ (nfusecat\ nells) = (nfusecat\ (nmap\ (\ nmap\ f\)\ nells))$

proof –

let $?n2l = (lmap\ llist-of-nellist\ \circ\ llist-of-nellist)$

have 1: $nmap\ f\ (nfusecat\ nells) =$
 $nmap\ f\ ((nellist-of-llist\ \circ\ lfusecat\ \circ\ ?n2l)\ nells)$

by (simp add: nfusecat-def2)

have 2: $nmap\ f\ ((nellist-of-llist\ \circ\ lfusecat\ \circ\ ?n2l)\ nells) =$
 $nellist-of-llist\ (lmap\ f\ ((lfusecat\ \circ\ ?n2l)\ nells))$

using *nmap-nellist-of-llist not-null-lfusecat* by auto

have 3: $(lmap\ f\ ((lfusecat\ \circ\ ?n2l)\ nells)) =$
 $lfusecat\ (lmap\ (lmap\ f)\ ((lmap\ llist-of-nellist\ \circ\ llist-of-nellist)\ nells))$

by (simp add: lmap-lfusecat)

have 4: $(nfusecat\ (nmap\ (\ nmap\ f\)\ nells)) =$
 $(nellist-of-llist\ \circ\ lfusecat\ \circ\ ?n2l)\ (nmap\ (\ nmap\ f\)\ nells)$

by (simp add: nfusecat-def2)

have 8: $(lmap\ (lmap\ f)\ (lmap\ llist-of-nellist\ (llist-of-nellist\ nells))) =$
 $(lmap\ llist-of-nellist\ (lmap\ (nmap\ f)\ (llist-of-nellist\ nells)))$

using *lmap-llist-of-nellist-nmap* by blast

show ?thesis

using 1 2 3 4 8 by fastforce

qed

lemma *nfusecat-nfirst*:

$nfusecat\ (nfusecat\ nell\ nelly) = nfusecat\ nell$

by (metis nfusecat-def1 nlast-NNil ntaken-0 ntaken-nappend1 zero-enat-def zero-le)

lemma *nfusecat-nfirst*:

shows $nfusecat\ (nfusecat\ nell) = nfusecat\ (nfusecat\ nell)$

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: $nfusecat\ (NCons\ nell\ nells1) = nfusecat\ nell\ (nfusecat\ nells1)$
by (simp)

have 2: $nfusecat\ (nfusecat\ nell\ (nfusecat\ nells1)) = nfusecat\ nell$
using *nfusecat-nfusecat* by auto

have 3: $nfusecat\ (nfusecat\ (NCons\ nell\ nells1)) = nfusecat\ nell$
by (metis 1 nfusecat-nfusecat nfusecat-expand)

show ?thesis

using 1 2 3 NCons by fastforce

qed

qed

lemma *nfusecat*:

shows $\text{nfusecat}(\text{NCons } \text{nell } \text{nells}) = \text{nfusecat } \text{nell}$
by (*simp add: nfusecat-def*)

lemma *nfusecat-nlength-eq-zero-conv*:

$\text{nlength}(\text{nfusecat } \text{nells}) = 0 \longleftrightarrow (\forall \text{ nell} \in \text{nset } \text{nells}. \text{nlength } \text{nell} = 0)$

proof –

let $?n2l = (\text{lmap } \text{lmap } \text{list-of-nlist} \circ \text{lmap } \text{list-of-nlist})$

have 1: $\text{nlength}(\text{nfusecat } \text{nells}) =$
 $\text{nlength}((\text{list-of-nlist} \circ \text{list-of-nlist} \circ ?n2l) \text{ nells})$

by (*simp add: nfusecat-def*)

have 2: $\text{nlength}((\text{list-of-nlist} \circ \text{list-of-nlist} \circ ?n2l) \text{ nells}) =$
 $\text{epred}(\text{llength}(\text{list-of-nlist} \circ \text{list-of-nlist} \circ ?n2l \text{ nells}))$

by *simp*

(*metis lbutlast-snoc llength-lbutlast list-of-nlist-a-inverse nlength.rep-eq*)

have 3: $\text{nlength}(\text{nfusecat } \text{nells}) = 0 \longleftrightarrow$
 $\text{epred}(\text{llength}(\text{list-of-nlist} \circ \text{list-of-nlist} \circ ?n2l \text{ nells})) = 0$

using 1 2 **by** *presburger*

have 4: $\text{llength} (?n2l \text{ nells}) > 0$

by *simp*

have 5: $\text{llength}(\text{list-of-nlist} \circ \text{list-of-nlist} \circ ?n2l \text{ nells}) > 0$

by (*simp add: list-of-nlist-not-lnull-var*)

have 6: $\text{epred}(\text{llength}(\text{list-of-nlist} \circ \text{list-of-nlist} \circ ?n2l \text{ nells})) = 0 \longleftrightarrow$
 $(\text{llength}(\text{list-of-nlist} \circ \text{list-of-nlist} \circ \text{lmap } \text{list-of-nlist} (\text{list-of-nlist } \text{nells}))) = 1$

by (*metis 5 comp-apply epred-1 epred-inject not-less0 zero-one-enat-neq(1)*)

have 7: $(\text{llength}(\text{list-of-nlist} \circ \text{list-of-nlist} \circ ?n2l \text{ nells})) \leq 1 \longleftrightarrow$
 $(\forall \text{ nell} \in \text{lset} (?n2l \text{ nells}). \text{llength } \text{nell} \leq 1)$

using *list-of-nlist-all-empty-or-LNil lset-lt-llength-var* **by** *blast*

have 8: $(\forall \text{ nell} \in \text{lset} (?n2l \text{ nells}). \text{llength } \text{nell} > 0)$

by *simp*

have 9: $(\text{llength}(\text{list-of-nlist} \circ \text{list-of-nlist} \circ ?n2l \text{ nells})) = 1 \longleftrightarrow$
 $(\forall \text{ nell} \in \text{lset} (?n2l \text{ nells}). \text{llength } \text{nell} = 1)$

by (*metis 5 7 8 ileI1 one-eSuc order-antisym-conv*)

have 10: $(\forall \text{ nell} \in \text{lset} (?n2l \text{ nells}). \text{llength } \text{nell} = 1) \longleftrightarrow$
 $(\forall \text{ nell} \in \text{nset } \text{nells}. \text{nlength } \text{nell} = 0)$

by *simp*

(*metis co-enat-exhaust-sel epred-1 llength-eq-0 list-of-nlist-not-lnull nlength.rep-eq zero-neq-one*)

show *?thesis* **using** 3 6 9 10 **by** *simp*

qed

lemma *is-NNil-nfusecat-a*:

assumes $\forall i. i \leq \text{nlength } \text{nells} \longrightarrow \text{is-NNil} (\text{nth } \text{nells } i)$

shows $\text{is-NNil}(\text{nfusecat } \text{nells})$

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 $\forall i. i \leq \text{nlength } nells \longrightarrow \text{is-NNil } (\text{nnth } nells \ i)$
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 *lmap 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 of-nelist (nappend llx lly)*) =
 lappend (lmap of-nelist llx) (lmap of-nelist lly)
by (*simp add: lmap-of-nelist-nappend*)
have 4: (*lmap llist-of-nelist (lappend (lmap of-nelist llx) (lmap 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)  $\longleftrightarrow$  nfinite nellx ∧ nfinite nelly
apply transfer
using lfuse-lfinite by blast

lemma nfusecat-nfinite:
assumes  $\forall$  nell ∈ nset nells. nlength nell > 0
   $\forall$  nell ∈ nset nells. nfinite nell
  nlastnfirst nells
shows nfinite (nfusecat nells)  $\longleftrightarrow$  nfinite nells
proof –
  let ?n2l = (lmap llist-of-nellist ∘ llist-of-nellist)
  have 1: nfinite (nfusecat nells)  $\longleftrightarrow$ 
    nfinite (((nellist-of-llist ∘ lfusecat ∘ ?n2l) nells))
  by (simp add: nfusecat-def2)
  have 2: nfinite (((nellist-of-llist ∘ lfusecat ∘ ?n2l) nells))  $\longleftrightarrow$ 
    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)))  $\longleftrightarrow$ 
    lfinite (lfusecat (?n2l nells))
  by (simp add: lbutlast-lfinite)
  have 4:  $\forall$  nell ∈ lset (llist-of-nellist nells). nlength nell > 0
  using assms(1) by force
  have 5:  $\forall$  nell ∈ lset (llist-of-nellist nells). epred(llength (llist-of-nellist nell)) > 0
  using 4 by fastforce
  have 6:  $\forall$  nell ∈ lset (llist-of-nellist nells).  $\neg$  lnull (ltl (llist-of-nellist nell))
  by (metis 5 epred-llength less-numeral-extra(3) llength-LNil llist.collapse(1))
  have 7:  $\forall$  nell ∈ lset (?n2l nells).  $\neg$  lnull (ltl nell)
  by (simp add: 6)

```

```

have 8:  $\forall nell \in lset (?n2l\ nells). \text{lfinite } nell$ 
  using assms(2) nfinite-def by fastforce
have 9: llastlfirst (?n2l nells)
  using assms(3) nlastnfirst.rep-eq by auto
have 10:  $\text{lfinite } (\text{lfusecat } (?n2l\ nells)) \longleftrightarrow \text{lfinite } (?n2l\ nells)$ 
  using 7 8 9 lfusecat-lfinite by blast
have 11:  $\text{lfinite } (?n2l\ nells) \longleftrightarrow \text{nfinite } nells$ 
  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
   $\forall nell \in nset\ nells. \text{nfinite } nell$ 
shows  $\text{nlast}(\text{nfusecat } nells) = \text{nlast}(\text{nlast } nells)$ 
proof –
let ?n2l = (lmap llist-of-nellist  $\circ$  llist-of-nellist)
have 1:  $\text{nlast}(\text{nfusecat } nells) =$ 
   $\text{nlast } (((\text{nellist-of-llist} \circ \text{lfusecat} \circ ?n2l) \text{ } nells))$ 
  by (simp add: nfusecat-def2)
have 2:  $\text{nlast } (((\text{nellist-of-llist} \circ \text{lfusecat} \circ ?n2l) \text{ } nells)) =$ 
   $\text{llast } (\text{llist-of-nellist } (((\text{nellist-of-llist} \circ \text{lfusecat} \circ ?n2l) \text{ } nells)))$ 
  by (metis (no-types, lifting) llast-linfinite nfinite-def nlast-llast nlast-not-nfinite)
have 3:  $\text{llast } (\text{llist-of-nellist } (((\text{nellist-of-llist} \circ \text{lfusecat} \circ ?n2l) \text{ } nells))) =$ 
   $\text{llast } (\text{lfusecat } (?n2l\ nells))$ 
  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:  $\text{lfinite } (?n2l\ nells)$ 
  using assms(1) lfinite-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:  $\forall nell \in lset (?n2l\ nells). \text{lfinite } nell$ 
  using assms(3) nfinite-def by auto
have 9:  $\text{llast } (\text{lfusecat } (?n2l\ nells)) = \text{llast}(\text{llast } (?n2l\ nells))$ 
  using 4 5 6 7 8 llastfirst-lfusecat-llast by blast
have 10:  $\text{llast}(\text{llast } (?n2l\ nells)) = \text{nlast } (\text{nlast } nells)$ 
  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
   $\forall nell \in nset\ nells. \text{nfinite } nell$ 

```

$nlastnfirst\ nells$
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: $((\sum i = 0 \dots (the-enat(epred(llength\ (?n2l\ nells)))) \cdot$
 $epred(llength\ (lnth\ (?n2l\ nells\ i)))) =$
 $((\sum i = 0 \dots (the-enat(nlength\ nells)). nlength\ (nnth\ nells\ i)))$
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 $n \leq \text{nlength } \text{nells}$
 $\text{nlastnfirst } \text{nells}$

shows $\text{nlastnfirst } (\text{ntake } n \text{ nells})$

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 $(\text{enat } n) \leq \text{nlength } \text{nells}$
 $\forall \text{ nell} \in \text{nset } \text{nells}. \text{nfinite } \text{nell}$
 $\text{nlastnfirst } \text{nells}$

shows $\text{nfusecat } (\text{ntake } n \text{ nells}) =$
 $\text{ntake } ((\sum i = 0 .. n . (\text{nlength } (\text{nnth } \text{nells } i))))$
 $(\text{nfusecat } \text{nells})$

proof –

let $?n2l = (\text{lmap } \text{llist-of-nellist} \circ \text{llist-of-nellist})$

have 1: $\text{nfusecat } (\text{ntake } n \text{ nells}) =$
 $((\text{nlist-of-llist} \circ \text{lfusecat} \circ ?n2l) (\text{ntake } n \text{ nells}))$

by (simp add: nfusecat-def2)

have 2: $((\text{nlist-of-llist} \circ \text{lfusecat} \circ ?n2l) (\text{ntake } n \text{ nells})) =$
 $\text{nlist-of-llist } (\text{lfusecat } (?n2l (\text{ntake } n \text{ nells})))$

by simp

have 3: $\text{llist-of-nellist}(\text{ntake } n \text{ nells}) = \text{ltake } (\text{eSuc } n) (\text{llist-of-nellist } \text{nells})$

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: $(?n2l (\text{ntake } n \text{ nells})) = \text{ltake } (\text{eSuc } n) (?n2l \text{ nells})$

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: $\text{llastlfirst } (?n2l \text{ nells})$

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*($\sum i = 0 \dots n$.
 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*($\sum i = 0 \dots n$.
 epred(*llength* (*lnth* (*?n2l nells*) *i*)))) =
 (*eSuc*($\sum i = 0 \dots n$. *epred*(*llength* (*lnth* (*?n2l nells*) *i*))))
using *nat.simps*(3) **by** *presburger*
have 111: $\bigwedge j. j \leq n \longrightarrow (\min (\text{enat } j) (\text{epred } (\text{llength } (\text{llist-of-nellist } \text{nells})))) = j$
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*($\sum i = 0 \dots n$.
 epred(*llength* (*lnth* (*?n2l nells*) *i*))))
 (*lfusecat* (*?n2l nells*)) =
 ltake (*eSuc*($\sum i = 0 \dots n$. *nlength* (*nnth nells i*))) (*llist-of-nellist* (*nfusecat nells*))
using 12 13 **by** *auto*
have 15: *nellist-of-llist* (*ltake* (*eSuc*($\sum i = 0 \dots n$.
 nlength (*nnth nells i*))) (*llist-of-nellist* (*nfusecat nells*))) =
 ntake (($\sum i = 0 \dots n$. *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 $n < \text{nlength } \text{nells}$

$\forall \text{ nell} \in \text{nset } \text{nells}. \text{ nfinite } \text{nell}$

nlastnfirst nells

shows *nfusecat* (*ndropn* *n nells*) =

ndropn (*the-enat*(*if* $n = 0$ *then* 0 *else* ($\sum i = 0 \dots (n-1)$. *nlength*(*nnth nells i*))))
 (*nfusecat nells*)

proof –

let *?n2l* = (*lmap* *llist-of-nellist* \circ *llist-of-nellist*)

have 1: *nfusecat* (*ndropn* *n nells*) =

((*nellist-of-llist* \circ *lfusecat* \circ *?n2l*) (*ndropn* *n nells*)))

by (*simp add: nfusecat-def2*)

have 2: ((*nellist-of-llist* \circ *lfusecat* \circ *?n2l*) (*ndropn* *n nells*))) =

nellist-of-llist (*lfusecat* (*lmap* *llist-of-nellist* (*ldropn* *n* (*llist-of-nellist nells*))))

using *assms* **by** *simp*

have 4: *nellist-of-llist* (*lfusecat* (*lmap* *llist-of-nellist* (*ldropn* *n* (*llist-of-nellist nells*)))) =

nellist-of-llist (*lfusecat* (*lmap* *llist-of-nellist* (*ldrop* *n* (*llist-of-nellist nells*))))

by (simp add: ldrop-enat)
 have 5: (lmap llist-of-nellist (ldrop n (llist-of-nellist nells))) = ldrop n (?n2l nells)
 by (simp)
 have 6: $\neg \text{lnull } (?n2l \text{ nells})$
 by simp
 have 7: $n < \text{llength } (?n2l \text{ nells})$
 by (metis 5 assms(1) ldrop-enat llist.map-disc-iff llist-of-nellist-ndropn llist-of-nellist-not-lnull
 lnull-ldrop min-def nlength.rep-eq not-le-imp-less order-less-imp-le the-enat.simps)
 have 8: $\forall \text{ nell} \in \text{lset } (?n2l \text{ nells}). \neg \text{lnull nell}$
 by simp
 have 9: $\forall \text{ nell} \in \text{lset } (?n2l \text{ nells}). \text{lfinitel nell}$
 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 $(\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } (?n2l \text{ nells}) i)))$)
 (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 (llist-of-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: $n-1 < \text{llength } (?n2l \text{ nells})$
 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)) = llist-of-nellist (nfusecat nells)
 by (simp add: lfusecat-not-lnull-var nfusecat-def2)
 have 131: $\bigwedge j. j < n \implies (\text{min } (\text{enat } j) (\text{epred } (\text{llength } (\text{llist-of-nellist } \text{nells})))) = j$
 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 $\sum i = 0..n-1. \text{epred } (\text{llength } (\text{lnth } (?n2l \text{ nells}) i))$) =
 (if $n = 0$ then 0 else $(\sum i = 0 .. (n-1) . \text{nlength}(\text{nnth } \text{nells } i))$)
 using sum.cong[of $\{0..n-1\} \{0..n-1\} \lambda i. \text{epred } (\text{llength } (\text{lnth } (?n2l \text{ nells}) i))$
 $\lambda i. \text{nlength}(\text{nnth } \text{nells } i)$] 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 $(\sum i = 0 .. (n-1) . \text{epred}(\text{llength } (\text{lnth } (?n2l \text{ nells}) i)))$)
 (lfusecat (?n2l nells)) =
 ldrop (if $n = 0$ then 0 else $(\sum i = 0 .. (n-1) . \text{nlength}(\text{nnth } \text{nells } i))$) (llist-of-nellist (nfusecat
 nells))
 using 13 132 by presburger
 have 15: $0 < n \implies (\bigwedge j. j \leq n-1 \longrightarrow \text{epred}(\text{llength } (\text{lnth } (?n2l \text{ nells}) j)) < \infty)$

```

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:  $0 < n \implies ((\sum i = 0 .. (n-1) . epred(llength (lnth (?n2l nells) i)))) < \infty$ 
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:  $\exists m. (enat\ m) = (if\ n = 0\ then\ 0\ else\ (\sum i = 0 .. (n-1) . epred(llength (lnth (?n2l nells) i))))$ 
by (metis 16 less-infinityE not-gr-zero zero-enat-def)
obtain m where 18:  $(enat\ m) = (if\ n = 0\ then\ 0\ else\ (\sum i = 0 .. (n-1) . epred(llength (lnth (?n2l nells) i))))$ 
using 17 by blast
have 19:  $ldrop\ (if\ n = 0\ then\ 0\ else\ (\sum i = 0 .. (n-1) . epred(llength (lnth (?n2l nells) i))))$ 
 $(lfusecat\ (?n2l\ nells)) =$ 
 $ldropn\ m\ (lfusecat\ (?n2l\ nells))$ 
using 18 ldrop-enat[of m (lfusecat (?n2l nells))]
by presburger
have 20:  $enat\ m \leq epred\ (llength\ (lfusecat\ (?n2l\ nells)))$ 
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:  $nellist-of-llist\ (ldropn\ m\ (lfusecat\ (?n2l\ nells))) = ndropn\ m\ (nfusecat\ nells)$ 
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:  $m = (the-enat(if\ n = 0\ then\ 0\ else\ (\sum i = 0 .. (n-1) . nlength(nnth\ nells\ i))))$ 
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)
(enat n)  $\leq$  nlength nells
nlastnfirst nells
 $\forall\ nell \in nset\ nells. nfinite\ nell$ 

```

shows $nridx\ R\ (nfusecat\ (ntake\ n\ nells))$
using *assms* **by** (*metis nfusecat-ntake nle-le nridx-ntake-a ntake-all*)

lemma *nridx-nfusecat-ndrop*:

assumes $nridx\ R\ (nfusecat\ nells)$

$(enat\ n) < nlength\ nells$

$nlastnfirst\ nells$

$\forall\ nell \in nset\ nells.\ nfinite\ nell$

shows $nridx\ R\ (nfusecat\ (ndropn\ n\ nells))$

proof –

let $?n2l = (lmap\ llist-of-nellist\ \circ\ llist-of-nellist)$

have 1: $nridx\ R\ (nfusecat\ (ndropn\ n\ nells)) \longleftrightarrow$
 $nridx\ R\ ((nellist-of-llist\ \circ\ lfusecat\ \circ\ ?n2l)\ (ndropn\ n\ nells))$

by (*simp add: nfusecat-def2*)

have 2: $nridx\ R\ ((nellist-of-llist\ \circ\ lfusecat\ \circ\ ?n2l)\ (ndropn\ n\ nells)) \longleftrightarrow$
 $ridx\ R\ (llist-of-nellist\ ((nellist-of-llist\ \circ\ lfusecat\ \circ\ ?n2l)\ (ndropn\ n\ nells)))$

using *nridx.rep-eq* **by** *blast*

have 3: $ridx\ R\ (llist-of-nellist\ ((nellist-of-llist\ \circ\ lfusecat\ \circ\ ?n2l)\ (ndropn\ n\ nells))) \longleftrightarrow$
 $ridx\ R\ ((lfusecat\ \circ\ ?n2l)\ (ndropn\ n\ nells))$

using *not-null-lfusecat*

by (*metis (no-types, lifting) comp-apply nridx.abs-eq nridx.rep-eq*)

have 4: $llastlfirst\ (?n2l\ nells)$

using *assms nlastnfirst.rep-eq* **by** *auto*

have 5: $\forall\ nell \in lset\ (?n2l\ nells).\ lfinite\ nell$

using *assms nfinite-def* **by** *fastforce*

have 6: $((lfusecat\ \circ\ ?n2l)\ (ndropn\ n\ nells)) =$
 $lfusecat\ (lmap\ llist-of-nellist\ (ldropn\ n\ (llist-of-nellist\ nells)))$

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

have 7: $(lmap\ llist-of-nellist\ (ldropn\ n\ (llist-of-nellist\ nells))) =$
 $ldropn\ n\ (lmap\ llist-of-nellist\ (llist-of-nellist\ nells))$

by *auto*

have 8: $ridx\ R\ (lfusecat\ (?n2l\ nells))$

by (*metis (no-types, lifting) assms(1) comp-apply nfusecat-def2 nridx.abs-eq ridx-expand-1*)

have 9: $enat\ n < llength\ (?n2l\ nells)$

by (*metis assms(2) co.enat.exhaust-sel comp-apply iless-Suc-eq llength-eq-0 llength-lmap*
 $llist-of-nellist-not-lnull\ nlength.rep-eq\ order-less-imp-le$)

have 10: $ridx\ R\ (lfusecat\ (ldropn\ n\ (?n2l\ nells)))$

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*
 $llist-of-nellist-not-lnull\ lnth-lmap$)

show *?thesis*

using 1 10 2 3 6 7 **by** (*metis comp-apply*)

qed

lemma *nridx-nfusecat*:

assumes $nlastnfirst\ nells$

$\forall\ nell \in nset\ nells.\ 0 < nlength\ nell$

$\forall\ nell \in nset\ nells.\ nfinite\ nell$

shows $nridx\ R\ (nfusecat\ nells) \longleftrightarrow (\forall i.\ i \leq nlength\ nells \longrightarrow nridx\ R\ (nnth\ nells\ i))$

proof –

let $?n2l = (lmap \text{ llist-of-nellist} \circ \text{ llist-of-nellist})$
have 1: $nridx \ R \ (nfusecat \ nells) \longleftrightarrow$
 $nridx \ R \ ((nellist\text{-of}\text{-l}list \circ lfusecat \circ ?n2l) \ nells)$
by (*simp add: nfusecat-def*)
have 2: $nridx \ R \ ((nellist\text{-of}\text{-l}list \circ lfusecat \circ ?n2l) \ nells) \longleftrightarrow$
 $ridx \ R \ (llist\text{-of}\text{-nellist} \ ((nellist\text{-of}\text{-l}list \circ lfusecat \circ ?n2l) \ nells))$
using $nridx.rep\text{-}eq$ **by** *blast*
have 3: $ridx \ R \ (llist\text{-of}\text{-nellist} \ ((nellist\text{-of}\text{-l}list \circ lfusecat \circ ?n2l) \ nells)) \longleftrightarrow$
 $ridx \ R \ ((lfusecat \circ ?n2l) \ nells)$
using *not-null-lfusecat nridx.abs-eq* **by** *fastforce*
have 4: $llastlfirst \ (?n2l \ nells)$
using $assms(1) \ nlastnfirst.rep\text{-}eq$ **by** *auto*
have 5: $\forall \text{ nell} \in \text{lset} \ (?n2l \ nells). \ 1 < \text{length} \ \text{nell}$
by (*metis assms(2) epred-1 comp-apply imageE less-numeral-extra(3) llist.set-map*
 $llist\text{-of}\text{-nellist}\text{-not}\text{-lnull} \ \text{lset}\text{-l}list\text{-of}\text{-nellist}\text{-a} \ \text{nlength.rep}\text{-}eq \ \text{not}\text{-lnull}\text{-length}$
 $\text{order}\text{-neg}\text{-le}\text{-trans}$)
have 6: $\forall \text{ nell} \in \text{lset} \ (?n2l \ nells). \ \text{lfinite} \ \text{nell}$
using $assms(3) \ nfinite\text{-}def$ **by** *fastforce*
have 7: $ridx \ R \ ((lfusecat \circ ?n2l) \ nells) \longleftrightarrow$
 $(\forall i. \ i < \text{length} \ (?n2l \ nells) \longrightarrow ridx \ R \ (\text{lnth} \ (?n2l \ nells) \ i))$
using 4 5 6 $ridx\text{-}lfusecat$ **by** *fastforce*
have 8: $epred(\text{length} \ (?n2l \ nells)) = \text{nlength} \ nells$
by *simp*
have 9: $\bigwedge j. \ j < \text{length} \ (?n2l \ nells) \longrightarrow$
 $(\text{lnth} \ (?n2l \ nells) \ j) = \text{llist}\text{-of}\text{-nellist}(\text{nnth} \ nells \ j)$
by *simp*
 $(metis \ eSuc\text{-}epred \ eSuc\text{-}ile\text{-}mono \ \text{gr}\text{-}implies\text{-}not\text{-}zero \ ileI1 \ \text{lnth}\text{-}llist\text{-of}\text{-nellist}$
 $\text{min}\text{-}def \ \text{the}\text{-}enat.\text{simps})$
have 10: $(\forall i. \ i < \text{length} \ (?n2l \ nells) \longrightarrow ridx \ R \ (\text{lnth} \ (?n2l \ nells) \ i)) \longleftrightarrow$
 $(\forall i. \ i \leq \text{nlength} \ (nells) \longrightarrow nridx \ R \ (\text{nnth} \ nells \ i))$
by (*metis (mono-tags, lifting) 8 9 co.enat.exhaust-sel comp-apply iless-Suc-eq*
 $\text{length}\text{-}eq\text{-}0 \ \text{llist.map}\text{-}disc\text{-}iff \ \text{llist}\text{-of}\text{-nellist}\text{-not}\text{-lnull} \ nridx.rep\text{-}eq$)
show *?thesis* **using** 1 2 3 10 7 **by** *blast*
qed

lemma *nfusecat-split-nsubn*:

assumes $nlastnfirst \ nells$
 $\forall \text{ nell} \in \text{nset} \ nells. \ 0 < \text{nlength} \ \text{nell}$
 $\forall \text{ nell} \in \text{nset} \ nells. \ nfinite \ \text{nell}$
shows $(\forall i. \ i \leq \text{nlength} \ nells \longrightarrow nridx \ (\lambda a \ b. \ f \ (nsubn \ \sigma \ a \ b)) \ (\text{nnth} \ nells \ i)) \longleftrightarrow$
 $nridx \ (\lambda a \ b. \ f \ (nsubn \ \sigma \ a \ b)) \ (nfusecat \ nells)$
using $assms \ nridx\text{-}nfusecat$ **by** *blast*

lemma *nidx-nfusecat*:

assumes $nlastnfirst \ nells$
 $\forall \text{ nell} \in \text{nset} \ nells. \ 0 < \text{nlength} \ \text{nell}$
 $\forall \text{ nell} \in \text{nset} \ nells. \ nfinite \ \text{nell}$
shows $nidx \ (nfusecat \ nells) \longleftrightarrow (\forall i. \ i \leq \text{nlength} \ nells \longrightarrow nidx \ (\text{nnth} \ nells \ i))$
using $assms \ nridx\text{-}nfusecat \ nridx\text{-}nidx$ **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

```

3.9 nkfilter

```

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  $\text{nkfilter } P\ n\ (\text{NCons } x\ nell) = (\text{NNil } n)$ 
using assms
by transfer
  (auto simp add: kfilter-lnull-conv)

```

```

lemma nkfilter-nlength:
  assumes  $\exists x \in \text{nset nell. } P\ x$ 
  shows  $\text{nlength}(\text{nkfilter } P\ n\ nell) = \text{nlength}(\text{nfilter } P\ nell)$ 
using assms
by transfer
  (auto simp add: kfilter-lnull-conv kfilter-llength)

```

```

lemma nkfilter-upperbound:
  assumes  $\exists x \in \text{nset nell. } P\ x$ 
     $i \leq \text{nlength}(\text{nkfilter } P\ n\ nell)$ 
  shows  $\text{nnth}(\text{nkfilter } P\ n\ nell)\ i \leq n + \text{nlength } nell$ 
using assms
by transfer
  (auto,
    simp add: kfilter-lnull-conv,
   metis co.enat.exhaust-sel iadd-Suc-right illess-Suc-eq kfilter-upperbound llength-eq-0)

```



```

lemma nkfilter-lowerbound:
  assumes  $\exists x \in \text{nset } nell. P x$ 
     $i \leq \text{nlength } (\text{nkfilter } P \ n \ nell)$ 
  shows  $n \leq \text{nnth } (\text{nkfilter } P \ n \ nell) \ i$ 
  using assms
by transfer
  (auto,
    metis co.enat.collapse iless-Suc-eq kfilter-lnth-n-zero-a llength-eq-0)

lemma nkfilter-mono:
  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 transfer
  (auto,
    metis diff-Suc-1 eSuc-epred epred-enat epred-le-epredI lidx-kfilter-gr iless-Suc-eq leD lessI
    llength-eq-0 min.cobounded1 min-def the-enat.simps)

lemma nkfilter-nfilter:
  assumes  $\exists x \in \text{nset } nell. P x$ 
     $i \leq \text{nlength } (\text{nkfilter } P \ n \ nell)$ 
  shows  $(\text{nnth } nell \ ((\text{nnth } (\text{nkfilter } P \ n \ nell) \ i) - n)) = (\text{nnth } (\text{nfilter } P \ nell) \ i)$ 
  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:  $Pa \ x$ 
  assume a3:  $\text{enat } ia \leq \text{epred } (\text{llength } (\text{kfilter } Pa \ na \ nella))$ 
  assume a4:  $\neg \text{lnull } nella$ 
  have f5:  $\forall e \ ea. \neg (e::\text{enat}) \leq ea \vee \text{min } e \ ea = e$ 
  by (simp add: min-def-raw)
  have f6:  $\forall n \ p \ na \ as. \neg \text{enat } n < \text{llength } (\text{kfilter } p \ na \ (as::'a \text{ llist})) \vee$ 
     $\text{enat } (\text{lnth } (\text{kfilter } p \ na \ as) \ n) < \text{enat } na + \text{llength } as$ 
  by (meson kfilter-upperbound)
  have f7:  $\text{min } (\text{enat } ia) \ (\text{epred } (\text{llength } (\text{kfilter } Pa \ na \ nella))) = \text{enat } ia$ 
  using f5 a3 by blast
  then have f8:  $(\text{enat } ia < \text{llength } (\text{kfilter } Pa \ 0 \ nella)) =$ 
     $(\text{enat } (\text{the-enat } (\text{min } (\text{enat } ia) \ (\text{epred } (\text{llength } (\text{kfilter } Pa \ na \ nella))))) <$ 
     $\text{llength } (\text{kfilter } Pa \ na \ nella))$ 
  by (simp add: kfilter-llength)
  have f9:  $\forall n \ p \ na \ as. \neg \text{enat } n < \text{llength } (\text{kfilter } p \ na \ (as::'a \text{ llist})) \vee$ 
     $\text{lnth } (\text{kfilter } p \ na \ as) \ n - na = \text{lnth } (\text{kfilter } p \ 0 \ as) \ n$ 
  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  $\text{enat } (\text{the-enat } (\text{min } (\text{enat } ia) \ (\text{epred } (\text{llength } (\text{kfilter } Pa \ na \ nella))))) <$ 
     $\text{llength } (\text{kfilter } Pa \ na \ nella) \longrightarrow$ 

```

$$\begin{aligned} & \text{enat } (\text{lnth } (\text{kfilter } Pa \text{ na nella}) \\ & \quad (\text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{kfilter } Pa \text{ na nella})))) - na) \\ & < \text{eSuc } (\text{epred } (\text{llength } nella)) \end{aligned}$$
using *f9 f8 f7 f6 the-enat.simps* **by** *presburger*
then have *f10*:
$$\begin{aligned} & \text{enat } (\text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{kfilter } Pa \text{ na nella})))) < \\ & \quad \text{llength } (\text{kfilter } Pa \text{ na nella}) \longrightarrow \\ & \quad \text{enat } (\text{lnth } (\text{kfilter } Pa \text{ na nella}) \\ & \quad \quad (\text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{kfilter } Pa \text{ na nella})))) - na) \\ & \leq \text{epred } (\text{llength } nella) \end{aligned}$$
using *iless-Suc-eq* **by** *blast*
have *f11*: $\forall n \ p \ na \ as. \neg \text{enat } n < \text{llength } (\text{kfilter } p \ na \ as) \vee \text{lnth } as \ (\text{lnth } (\text{kfilter } p \ na \ as) \ n - na) =$

$$(\text{lnth } (\text{lfilter } p \ as) \ n::'a)$$
using *lfilter-kfilter* **by** *blast*
have *f12*:
$$\begin{aligned} & \text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{lfilter } Pa \text{ nella})))) = \\ & \quad \text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{kfilter } Pa \text{ na nella})))) \end{aligned}$$
by (*simp add: kfilter-llength*)
have $\neg \text{lnull } (\text{kfilter } Pa \text{ na nella})$
using *a1 a2 kfilter-not-lnull-conv* **by** *auto*
then have
$$\begin{aligned} & \text{enat } (\text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{kfilter } Pa \text{ na nella})))) < \\ & \quad \text{llength } (\text{kfilter } Pa \text{ na nella}) \end{aligned}$$
using *f7 a3* **by** (*metis co.enat.exhaust-sel illess-Suc-eq llength-eq-0 the-enat.simps*)
then show $\neg \text{lnull } nella \implies$

$$\begin{aligned} & \text{enat } ia \leq \text{epred } (\text{llength } (\text{kfilter } Pa \text{ na nella})) \implies \\ & x \in \text{lset } nella \implies \\ & Pa \ x \implies \\ & \text{lnth } nella \ (\text{the-enat } (\text{min } (\text{enat } (\text{lnth } (\text{kfilter } Pa \text{ na nella}) \ ia - na)) (\text{epred } (\text{llength } nella)))) = \\ & \text{lnth } (\text{lfilter } Pa \text{ nella}) \ (\text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{lfilter } Pa \text{ nella})))))) \end{aligned}$$
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*

$$\begin{aligned} & (\text{auto simp add: kfilter-lnull-conv min-def,} \\ & \text{metis co.enat.exhaust-sel gen-llength-def illess-Suc-eq kfilter-holds kfilter-llength-n-zero} \\ & \quad \text{kfilter-lnth-n-zero kfilter-lowerbound kfilter-upperbound ldistinct-Ex1 ldistinct-kfilter} \\ & \quad \text{le-add-diff-inverse llength-code llength-eq-0,} \\ & \text{metis add commute co.enat.exhaust-sel illess-Suc-eq kfilter-holds-not-a llength-eq-0}) \end{aligned}$$

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 $\text{nset}(\text{nkfilter } P \ n \ (\text{ndropn } k \ nell)) = \{ n + i \mid i. i \leq \text{nlength } nell - k \wedge P (\text{nnth } nell \ (i+k)) \}$

proof –

have 1: $\{ n+i \mid i. i \leq \text{nlength } nell - k \wedge P (\text{nnth } nell \ (k+i)) \} =$
 $\{ n + i \mid i. i \leq \text{nlength } nell - k \wedge P (\text{nnth } nell \ (i+k)) \}$

using *assms by (auto simp add: add.commute)*

show *?thesis using assms by (simp add: 1 nkfilter-ndropn-nset)*

qed

lemma *nfilter-nkfilter-ntaken-nlength-0:*

assumes $P (\text{nnth } nell \ ((\text{nnth } (\text{nkfilter } P \ n \ nell) \ k) - n))$

$k \leq \text{nlength } (\text{nfilter } P \ nell)$

shows $(\exists x \in \text{nset } nell. P x)$

using *assms*

by (*metis enat-ile in-nset-conv-nnth linorder-le-cases ntaken-all ntaken-nlast*)

lemma *nkfilter-nlength-n-zero:*

assumes $(\exists x \in \text{nset } nell. P x)$

shows $\text{nlength}(\text{nkfilter } P \ n \ nell) = \text{nlength}(\text{nkfilter } P \ 0 \ nell)$

using *assms by (simp add: nkfilter-nlength)*

lemma *nkfilter-nnth-n-zero:*

assumes $(\exists x \in \text{nset } nell. P x)$

$k \leq \text{nlength}(\text{nkfilter } P \ n \ \text{nell})$
shows $(\text{nnth } (\text{nkfilter } P \ n \ \text{nell}) \ k) - n = (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)$
using *assms*
by *transfer*
 $(\text{auto } \text{split: if-split-asm},$
 $\text{metis } \text{kfilter-lnull-conv},$
 $\text{metis } \text{kfilter-lnull-conv},$
 $\text{metis } \text{co.enat.exhaust-sel } \text{iless-Suc-eq } \text{kfilter-llength-n-zero } \text{kfilter-lnth-n-zero } \text{llength-eq-0}$
 $\text{min.orderE } \text{the-enat.simps})$

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 $(\text{metis } \text{diff-add } \text{nkfilter-lowerbound } \text{nlength-nmap } \text{nnth-nmap})$
show ?thesis **by** $(\text{simp add: } 1 \ 2 \ \text{nellist-eq-nnth-eq})$
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 $(\text{metis } \text{assms } \text{nkfilter-nlength-n-zero } \text{nlength-nmap})$
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 $(\text{simp add: } 1 \ \text{assms } \text{nkfilter-nnth-n-zero})$
show ?thesis **by** $(\text{simp add: } 1 \ 2 \ \text{nellist-eq-nnth-eq})$
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 $(\exists x \in \text{nset } \text{nell}. P \ x)$
 $n \leq i$
 $i-n \leq \text{nlength } \text{nell}$
shows $i \notin \text{nset}(\text{nkfilter } P \ n \ \text{nell}) = (\neg P \ (\text{nnth } \text{nell } (i-n)))$
using *assms*
by *(simp add: nkfilter-holds-c)*

lemma *nkfilter-disjoint-nset-coset:*
assumes $(\exists x \in \text{nset } \text{nell}. P \ x)$
shows $(\{i+n \mid i. i \leq \text{nlength } \text{nell}\} - \text{nset}(\text{nkfilter } P \ n \ \text{nell})) \cap \text{nset}(\text{nkfilter } P \ n \ \text{nell}) = \{\}$
using *assms by (simp add: inf.commute)*

lemma *nidx-nkfilter-expand:*
assumes $(\exists x \in \text{nset } \text{nell}. P \ x)$
 $(\text{Suc } i) \leq \text{nlength}(\text{nkfilter } P \ n \ \text{nell})$
shows $\text{nnth}(\text{nkfilter } P \ n \ \text{nell}) \ i < \text{nnth}(\text{nkfilter } P \ n \ \text{nell}) \ (\text{Suc } i)$
using *assms by (simp add: nkfilter-mono)*

lemma *nidx-nkfilter:*
assumes $(\exists x \in \text{nset } \text{nell}. P \ x)$
shows $\text{nidx}(\text{nkfilter } P \ n \ \text{nell})$
using *assms nidx-nkfilter-expand[of nell P - n] nidx-expand[of (nkfilter P n nell)]*
by *blast*

lemma *nidx-nkfilter-gr-eq*:
assumes $(\exists x \in \text{nset nell}. P x)$
 $k \leq j$
 $j \leq \text{nlength} (\text{nkfilter } P \text{ n nell})$
shows $\text{nnth} (\text{nkfilter } P \text{ n nell}) k \leq \text{nnth} (\text{nkfilter } P \text{ n nell}) j$
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 $(\forall j. k < j \wedge j \leq \text{nlength} (\text{nkfilter } P \text{ n nell}) \longrightarrow$
 $\text{nnth} (\text{nkfilter } P \text{ n nell}) k < \text{nnth} (\text{nkfilter } P \text{ n nell}) j)$
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 $(\exists x \in \text{nset nell}. P x)$
 $k \leq \text{nlength} (\text{nkfilter } P \text{ n nell})$
shows $\forall j. j \leq k \longrightarrow \text{nnth} (\text{nkfilter } P \text{ n nell}) j \leq \text{nnth} (\text{nkfilter } P \text{ n nell}) k$
using *assms* **by** (*simp add: nidx-nkfilter-gr-eq*)

lemma *nkfilter-not-before*:
assumes $(\exists x \in \text{nset nell}. P x)$
 $i < (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) \text{ 0})$
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 $(\exists x \in \text{nset} (\text{ndropn } n \text{ nell}). P x)$
 $n \leq \text{nlength nell}$
 $n \leq i$
 $i < (\text{nnth} (\text{nkfilter } P \text{ n} (\text{ndropn } n \text{ nell})) \text{ 0})$
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 ≤ i ⇒
  ¬ lnull (kfilter P n (ldropn n nell)) ⇒
  i < lnth (kfilter P n (ldropn n nell)) 0 ⇒
  x ∈ lset (ldropn n nell) ⇒ P x ⇒ enat i ≤ epred (llength nell) ⇒ P (lnth nell i) ⇒ 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 ⇒ bool and ia :: nat and x :: 'b
assume a1: ia < lnth (kfilter Pa na (ldropn na nella)) 0
assume a2: ¬ lnull (kfilter Pa na (ldropn na nella))
assume a3: enat na ≤ epred (llength nella)
assume a4: ¬ lnull nella
assume a5: ¬ enat ia ≤ epred (llength nella)
assume a6: Pa (lnth nella (the-enat (epred (llength nella))))
have f7: ∀ p n bs. lnull (kfilter p n (bs::'b llist)) ∨ lnth (kfilter p n bs) 0 = n + lleast p 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 ≤ 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 (∃ x ∈ nset nell. P x)
  nnth (nkfilter P 0 nell) k < i
  nlength(nkfilter P 0 nell) = (enat k)
  i ≤ nlength nell
shows ¬ P (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 (∃ x ∈ nset (ndropn n nell). P x)
  n ≤ nlength nell
  nnth (nkfilter P n (ndropn n nell)) k < i
  nlength(nkfilter P n (ndropn n nell)) = (enat k)
  i ≤ nlength nell
shows ¬ P (nnth nell i)
using assms
by transfer

```

(*auto simp* add: *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 $(\exists x \in \text{nset } \text{nell}. P x)$
 $\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k < i$
 $i < \text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ (\text{Suc } k)$
 $(\text{Suc } k) \leq \text{nlength } (\text{nkfilter } P \ 0 \ \text{nell})$
shows $\neg P (\text{nnth } \text{nell } i)$

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 $k \leq \text{nlength } \text{nell}$
shows $\text{nset } (\text{ntaken } k \ \text{nell}) = \{ (\text{nnth } \text{nell } i) \mid i. i \leq k \}$

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 $\text{nidx } (\text{ntaken } k \ (\text{nkfilter } P \ n \ \text{nell}))$

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 $\text{nidx } (\text{ndropn } k \ (\text{nkfilter } P \ n \ \text{nell}))$

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 (nnth nell ((nnth (nkfilter P 0 nell) k)))
 $k \leq \text{nlength (nfilter } P \text{ nell)}$
shows $\text{nidx (nkfilter } P \text{ 0 (ntaken (nnth (nkfilter } P \text{ 0 nell) } k) \text{ nell))}$
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 $\exists x \in \text{nset nell. } P x$
 $k \leq \text{nlength (nfilter } P \text{ nell)}$
shows $\text{nidx (nkfilter } P \text{ 0 (ntaken (nnth (nkfilter } P \text{ 0 nell) } k) \text{ nell))}$
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 (nnth nell ((nnth (nkfilter P n nell) k) $-n$))
 $k \leq \text{nlength (nfilter } P \text{ nell)}$
shows $\text{nidx (nkfilter } P \text{ } n \text{ (ntaken (nnth (nkfilter } P \text{ } n \text{ nell) } k) \text{ nell))}$
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 $\exists x \in \text{nset nell. } P x$
 $k \leq \text{nlength (nfilter } P \text{ nell)}$
shows $\text{nidx (nkfilter } P \text{ } n \text{ (ntaken (nnth (nkfilter } P \text{ } n \text{ nell) } k) \text{ nell))}$
using *assms nfilter-nkfilter-ntaken-nidx-b-2*[of P nell n k] *nkfilter-holds*
by (metis *in-nset-conv-nnth nkfilter-nlength*)

lemma *nfilter-nkfilter-ndropn-nidx-b:*

assumes P (nnth nell ((nnth (nkfilter P 0 nell) k)))
 $k \leq \text{nlength (nfilter } P \text{ nell)}$
shows $\text{nidx (nkfilter } P \text{ 0 (ndropn (nnth (nkfilter } P \text{ 0 nell) } k) \text{ nell))}$
using *assms nidx-nkfilter*[of (ndropn (nnth (nkfilter P 0 nell) k) 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 $\exists x \in \text{nset nell. } P x$
 $k \leq \text{nlength (nfilter } P \text{ nell)}$
shows $\text{nidx (nkfilter } P \text{ 0 (ndropn (nnth (nkfilter } P \text{ 0 nell) } k) \text{ nell))}$
using *assms nfilter-nkfilter-ndropn-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-ndropn-nidx-b-2:*

assumes P (nnth nell ((nnth (nkfilter P n nell) k) $-n$))

$k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
shows $\text{nidx } (\text{nkfilter } P ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - n)$
 $(\text{ndropn } ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - n) \text{ nell}))$
proof –
have 1: $\text{enat } (\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - n \leq \text{nlength } \text{nell}$
using $\text{nkfilter-upperbound}[\text{of } \text{nell } P] \text{ nkfilter-nnth-n-zero}[\text{of } \text{nell } P \text{ k } n] \text{ assms}$
by $(\text{metis } \text{gen-nlength-def } \text{idiff-enat-enat } \text{nfilter-nkfilter-ntaken-nlength-0}$
 $\text{nkfilter-nlength } \text{nlength-code})$
have 2: $\text{nset } (\text{ndropn } ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - n) \text{ nell}) =$
 $\{ (\text{nnth } \text{nell } i) \mid i. ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - n) \leq i \wedge i \leq \text{nlength } \text{nell} \}$
using $\text{ndropn-nset}[\text{of } (\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - n \text{ nell}] \text{ 1 by simp}$
have 3: $\exists x \in \text{nset } (\text{ndropn } ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - n) \text{ nell}). P \text{ x}$
using 1 2 $\text{assms}(1)$ **by** auto
show ?thesis **using** 3
 $\text{nidx-nkfilter}[\text{of } (\text{ndropn } ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - n) \text{ nell}) \text{ P } ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - n)]$
by blast
qed

lemma $\text{nfilter-nkfilter-ndropn-nidx-b-3}$:

assumes $\exists x \in \text{nset } \text{nell}. P \text{ x}$
 $k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
shows $\text{nidx } (\text{nkfilter } P ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - n)$
 $(\text{ndropn } ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - n) \text{ nell}))$
using $\text{assms } \text{nfilter-nkfilter-ndropn-nidx-b-2}$
by $(\text{metis } \text{in-nset-conv-nnth } \text{nkfilter-holds } \text{nkfilter-nlength})$

lemma $\text{ntaken-nkfilter-ntaken-nset-eq}$:

assumes $P (\text{nnth } \text{nell } ((\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ k})))$
 $k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
shows $\text{nset}(\text{ntaken } k (\text{nkfilter } P \text{ 0 nell})) =$
 $\text{nset}(\text{nkfilter } P \text{ 0 } (\text{ntaken } ((\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ k})) \text{ nell}))$
proof –
have 1: $((\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ k})) \leq \text{nlength } \text{nell}$
using $\text{assms } \text{nkfilter-upperbound}[\text{of } \text{nell } P \text{ k } 0]$
by $(\text{metis } \text{diff-zero } \text{gen-nlength-def } \text{nfilter-nkfilter-ntaken-nlength-0 } \text{nkfilter-nlength } \text{nlength-code})$
have 2: $\exists x \in \text{nset } \text{nell}. P \text{ x}$
using $\text{assms by } (\text{metis } 1 \text{ exists-Pred-nnth-nset})$
have 3: $\{i. i \leq \text{nlength}(\text{ntaken } (\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ k}) \text{ nell}) \wedge$
 $P (\text{nnth } (\text{ntaken } (\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ k}) \text{ nell}) i) \}$
 $= \{i. i \leq (\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ k}) \wedge P (\text{nnth } \text{nell } i)\}$
by $(\text{auto simp add: } \text{ntaken-nnth } 1 \text{ order-subst2})$
have 4: $\exists x \in \text{nset}(\text{ntaken } (\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ k}) \text{ nell}). P \text{ x}$
using 1 $\text{assms}(1)$ ntaken-nset **by** fastforce
have 5: $\{i. i \leq (\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ k}) \wedge P (\text{nnth } \text{nell } i)\} =$
 $\{i. i \leq (\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ k}) \wedge i \in \text{nset}(\text{nkfilter } P \text{ 0 nell})\}$
using 4 1 2 nkfilter-holds-b
by $(\text{metis } (\text{mono-tags, opaque-lifting}) \text{add-cancel-left-right } \text{enat-ord-simps}(1) \text{ order.trans})$
have 6: $\{i. i \leq (\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ k}) \wedge i \in \text{nset}(\text{nkfilter } P \text{ 0 nell})\} =$
 $\{i. i \leq (\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ k}) \wedge$
 $i \in \{ (\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ j}) \mid j. j \leq \text{nlength } (\text{nkfilter } P \text{ 0 nell}) \}\}$

by (simp add: nset-conv-nnth)
 have 7: $\{i. i \leq (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \wedge$
 $i \in \{(\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ j) \mid j. j \leq \text{nlength } (\text{nkfilter } P \ 0 \ \text{nell})\} =$
 $\{(\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ j) \mid j. j \leq k\}$
 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: $k \leq \text{nlength } (\text{nkfilter } P \ 0 \ \text{nell})$
 by (simp add: 2 assms(2) nkfilter-nlength)
 have 9: $\{(\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ j) \mid j. j \leq k\} = \text{nset}(\text{ntaken } k \ (\text{nkfilter } P \ 0 \ \text{nell}))$
 using ntaken-nset[of $k \ (\text{nkfilter } P \ 0 \ \text{nell})$] 8 by auto
 have 91: $\text{min } (\text{enat } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)) \ (\text{nlength } \text{nell}) = (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)$
 by (simp add: 1 min-def)
 have 10: $\text{nset}(\text{nkfilter } P \ 0 \ (\text{ntaken } ((\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)) \ \text{nell})) =$
 $\{i. i \leq (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \wedge$
 $P \ (\text{nnth } (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ \text{nell}) \ i) \}$
 using nkfilter-nset[of $(\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ \text{nell}) \ P \ 0$]
 1 4 91 ntaken-nlength[of $(\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ \text{nell}$]
 by auto
 have 11: $\text{nlength}((\text{ntaken } ((\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)) \ \text{nell})) = (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)$
 by (simp add: 1)
 have 12: $\{i. i \leq (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \wedge$
 $P \ (\text{nnth } (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ \text{nell}) \ i) \} =$
 $\{i. i \leq \text{nlength}((\text{ntaken } ((\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)) \ \text{nell})) \wedge$
 $P \ (\text{nnth } (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ \text{nell}) \ i) \}$
 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 $P \ \text{nell } k$]
 nkfilter-holds[of $\text{nell } P \ (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ 0$]
 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 $\text{nell } P \ k \ 0$] 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 $a1: \text{enat } i \leq \text{nlength} \ \text{nell} - \text{enat} (\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k)$
assume $a2: P \ (\text{nnth} \ \text{nell} \ (i + \text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k))$
have $f0: \text{enat} \ (i + (\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k)) \leq \text{nlength} \ \text{nell}$
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 $b0: \text{enat } i \leq \text{nlength} \ \text{nell} - \text{enat} (\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k)$
assume $b1: i + \text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k \in \text{nset} (\text{nkfilter } P \ 0 \ \text{nell})$
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: $\{(\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k) + i \mid i. i \leq \text{nlength} \ \text{nell} - (\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \wedge$
 $(i + (\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k)) \in \text{nset}(\text{nkfilter } P \ 0 \ \text{nell})\} =$
 $\{(\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k) + i \mid i.$
 $(\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \leq i + (\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \wedge$
 $i + (\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \leq \text{nlength} \ \text{nell} \wedge$
 $(i + (\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k)) \in \text{nset}(\text{nkfilter } P \ 0 \ \text{nell})\}$
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: $\{(\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k) + i \mid i.$
 $(\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \leq i + (\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \wedge$
 $i + (\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \leq \text{nlength} \ \text{nell} \wedge$
 $(i + (\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k)) \in \text{nset}(\text{nkfilter } P \ 0 \ \text{nell})\} =$
 $\{j. (\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \leq j \wedge j \leq \text{nlength} \ \text{nell} \wedge j \in \text{nset}(\text{nkfilter } P \ 0 \ \text{nell})\}$
by (*metis (no-types, lifting) add.commute diff-add*)
have 53 : $\{j. (\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \leq j \wedge j \leq \text{nlength} \ \text{nell} \wedge j \in \text{nset}(\text{nkfilter } P \ 0 \ \text{nell})\} =$
 $\{j. (\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \leq j \wedge j \leq \text{nlength} \ \text{nell} \wedge j \in$
 $\{(\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ jj) \mid jj. jj \leq \text{nlength} (\text{nkfilter } P \ 0 \ \text{nell})\}\}$
by (*simp add: nset-conv-nnth*)
have 54: $\{j. (\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \leq j \wedge j \leq \text{nlength} \ \text{nell} \wedge j \in$
 $\{(\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ jj) \mid jj. jj \leq \text{nlength} (\text{nkfilter } P \ 0 \ \text{nell})\}\} =$
 $\{(\text{nnth} (\text{nkfilter } P \ 0 \ \text{nell}) \ j) \mid j. k \leq j \wedge j \leq \text{nlength}(\text{nkfilter } P \ 0 \ \text{nell})\}$

```

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:  $k \leq \text{nlength } (\text{nkfilter } P \ 0 \ \text{nell})$ 
  by (simp add: 2 assms(2) nkfilter-nlength)
have 9:  $\{(\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ j) \mid j. k \leq j \wedge j \leq \text{nlength}(\text{nkfilter } P \ 0 \ \text{nell})\} =$ 
       $\text{nset}(\text{ndropn } k \ (\text{nkfilter } P \ 0 \ \text{nell}))$ 
  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  $\exists x \in \text{nset nell}. P \ x$ 
       $k \leq \text{nlength } (\text{nfilter } P \ \text{nell})$ 
shows  $\text{nset}(\text{ndropn } k \ (\text{nkfilter } P \ 0 \ \text{nell})) =$ 
       $\text{nset}(\text{nkfilter } P \ (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ (\text{ndropn } ((\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)) \ \text{nell}))$ 
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  $\exists x \in \text{nset nell}. P \ x$ 
       $k \leq \text{nlength } (\text{nfilter } P \ \text{nell})$ 
shows  $\text{ntaken } k \ (\text{nkfilter } P \ 0 \ \text{nell}) =$ 
       $(\text{nkfilter } P \ 0 \ (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ \text{nell}))$ 
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  $\exists x \in \text{nset nell}. P \ x$ 
       $k \leq \text{nlength } (\text{nfilter } P \ \text{nell})$ 
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  $\exists x \in \text{nset nell}. P \ x$ 
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* ($\lambda n. \text{nnth } nell \ n$) (*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* (*nnth* *nell* ((*nnth* (*nkfilter* *P* 0 *nell*) *k*)))

$k \leq \text{nlength } (\text{nkfilter } P \ 0 \ nell)$

shows $\text{ntaken } k \ (\text{nfilter } P \ nell) =$
 $\text{nfilter } P \ (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell)$

proof –

have 1: $\exists x \in \text{nset } nell. P \ x$

using *assms*

by (*metis in-nset-conv-nnth min.cobounded1 min-def nfinite-ntaken nset-nlast ntaken-all ntaken-nlast*)

have 2: $\text{nfilter } P \ nell = \text{nmap } (\lambda n. \text{nnth } nell \ n) \ (\text{nkfilter } P \ 0 \ nell)$

using 1 *assms* **by** (*simp add: nkfilter-nmap-nfilter*)

have 3: $\text{ntaken } k \ (\text{nfilter } P \ nell) = \text{ntaken } k \ (\text{nmap } (\lambda n. \text{nnth } nell \ n) \ (\text{nkfilter } P \ 0 \ nell))$

using 2 **by** *simp*

have 4: $\text{ntaken } k \ (\text{nmap } (\lambda n. \text{nnth } nell \ n) \ (\text{nkfilter } P \ 0 \ nell)) =$
 $\text{nmap } (\lambda n. \text{nnth } nell \ n) \ (\text{ntaken } k \ (\text{nkfilter } P \ 0 \ nell))$

by *simp*

have 5: $\exists x \in \text{nset } (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell). P \ x$

using *assms* **by** (*metis nfinite-ntaken nset-nlast ntaken-nlast*)

have 6: $(\text{nfilter } P \ (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell)) =$
 $\text{nmap } (\lambda s. \text{nnth } (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell) \ s)$
 $(\text{nkfilter } P \ 0 \ (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell))$

using 5 **by** (*simp add: nkfilter-nmap-nfilter*)

have 7: $\text{nmap } (\lambda n. \text{nnth } nell \ n) \ (\text{ntaken } k \ (\text{nkfilter } P \ 0 \ nell)) =$
 $\text{nmap } (\lambda n. \text{nnth } nell \ n) \ (\text{nkfilter } P \ 0 \ (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell))$

by (*simp add: 1 2 assms(2) nkfilter-nkfilter-ntaken*)

have 70: $\text{enat } k \leq \text{nlength } (\text{nfilter } P \ nell)$

using 2 *assms* **by** *auto*

have 71: $(\bigwedge z. z \in \text{nset } (\text{nkfilter } P \ 0 \ (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell)) \implies$
 $(\lambda n. \text{nnth } nell \ n) \ z = (\lambda s. \text{nnth } (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell) \ s) \ z)$

using *assms* 1 70 *ntaken-nkfilter-ntaken-nset-eq*[of *P* *nell* *k*]

ntaken-nset[of *k* (*nkfilter* *P* 0 *nell*)] *mem-Collect-eq*

nidx-nkfilter-gr-eq[of *nell* *P* - 0]

proof *simp*

fix *z* :: *nat*

assume *a1*: $\text{enat } k \leq \text{nlength } (\text{nkfilter } P \ 0 \ nell)$

assume *a2*: $\bigwedge k \ j. \llbracket k \leq j; \text{enat } j \leq \text{nlength } (\text{nkfilter } P \ 0 \ nell) \rrbracket \implies$
 $\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k \leq \text{nnth } (\text{nkfilter } P \ 0 \ nell) \ j$

assume *a3*: $z \in \text{nset } (\text{nkfilter } P \ 0 \ (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell))$

assume $\{\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ i \mid i. i \leq k\} =$
 $\text{nset } (\text{nkfilter } P \ 0 \ (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell))$

then have $\exists n. z = \text{nnth } (\text{nkfilter } P \ 0 \ nell) \ n \wedge n \leq k$

using *a3* **by** *blast*

then have $z \leq \text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k$

using *a2 a1* **by** *meson*

then show $\text{nnth } nell \ z = \text{nnth } (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell) \ z$

by (simp add: ntaken-nnth)
 qed
 have 8: $nmap (\lambda n. nnth\ nell\ n)$
 $(nkfilter\ P\ 0\ (ntaken\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)) =$
 $nmap (\lambda s. nnth\ (ntaken\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)\ s)$
 $(nkfilter\ P\ 0\ (ntaken\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell))$
 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 $\exists x \in nset\ nell. P\ x$
 $k \leq nlength\ (nkfilter\ P\ 0\ nell)$
shows $ntaken\ k\ (nfilter\ P\ nell) =$
 $nfilter\ P\ (ntaken\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)$
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 $\exists x \in nset\ nell. P\ x$
shows $nmap (\lambda s. nnth\ nell\ (s+n))\ (nkfilter\ P\ 0\ nell) =$
 $nmap (\lambda s. nnth\ nell\ s)\ (nkfilter\ P\ n\ nell)$
proof –
have 1: $nlength\ (nmap (\lambda s. nnth\ nell\ (s+n))\ (nkfilter\ P\ 0\ nell)) =$
 $nlength\ (nmap (\lambda s. nnth\ nell\ s)\ (nkfilter\ P\ n\ nell))$
 by (simp add: assms *nkfilter-nlength*)
have 2: $\bigwedge i. i \leq nlength\ (nmap (\lambda s. nnth\ nell\ (s+n))\ (nkfilter\ P\ 0\ nell)) \longrightarrow$
 $nnth\ (nmap (\lambda s. nnth\ nell\ (s+n))\ (nkfilter\ P\ 0\ nell))\ i =$
 $nnth\ nell\ ((nnth\ (nkfilter\ P\ 0\ nell)\ i) + n)$
 by simp
have 3: $\bigwedge i. i \leq nlength\ (nmap (\lambda s. nnth\ nell\ s)\ (nkfilter\ P\ n\ nell)) \longrightarrow$
 $nnth\ (nmap (\lambda s. nnth\ nell\ s)\ (nkfilter\ P\ n\ nell))\ i =$
 $nnth\ nell\ (nnth\ (nkfilter\ P\ n\ nell)\ i)$
 by simp
have 4: $\bigwedge i. i \leq nlength\ (nmap (\lambda s. nnth\ nell\ (s+n))\ (nkfilter\ P\ 0\ nell)) \longrightarrow$
 $nnth\ nell\ ((nnth\ (nkfilter\ P\ 0\ nell)\ i) + n) = nnth\ nell\ (nnth\ (nkfilter\ P\ n\ nell)\ i)$
 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 $\exists x \in nset\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell). P\ x$
 $k \leq nlength\ (nkfilter\ P\ 0\ nell)$
shows $nmap (\lambda s. nnth\ nell\ (s+(nnth\ (nkfilter\ P\ 0\ nell)\ k)))$
 $(nkfilter\ P\ 0\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)) =$
 $nmap (\lambda s. nnth\ nell\ s)\ (nkfilter\ P\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)$
 $(ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell))$

```

using assms
  nellist-eq-nnth-eq[of nmap ( $\lambda s. \text{nnth } nell \ (s + (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k)))$ 
    ( $\text{nkfilter } P \ 0 \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell)$ )
    nmap ( $\lambda s. \text{nnth } nell \ s$ ) ( $\text{nkfilter } P \ (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k)$ 
    ( $\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell)$ )]
using nkfilter-n-zero[of ( $\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell$ ) P
  ( $\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k$ )]
by simp

lemma nfilter-nkfilter-ndropn:
assumes P ( $\text{nnth } nell \ ( \ (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k)) \ )$ 
   $k \leq \text{nlength } (\text{nkfilter } P \ 0 \ nell)$ 
shows  $\text{ndropn } k \ (\text{nfilter } P \ nell) =$ 
   $\text{nfilter } P \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell)$ 
proof –
have 1:  $\exists x \in \text{nset } nell. \ P \ x$ 
  using assms
  by (metis in-nset-conv-nnth min.cobounded1 min-def nfinite-ntaken nset-nlast ntaken-all ntaken-nlast)
have 2:  $(\text{nfilter } P \ nell) = \text{nmap } (\lambda n. \text{nnth } nell \ n) \ (\text{nkfilter } P \ 0 \ nell)$ 
  by (simp add: 1 nkfilter-nmap-nfilter)
have 3:  $\text{ndropn } k \ (\text{nfilter } P \ nell) = \text{ndropn } k \ (\text{nmap } (\lambda n. \text{nnth } nell \ n) \ (\text{nkfilter } P \ 0 \ nell))$ 
  by (simp add: 2)
have 4:  $\text{ndropn } k \ (\text{nmap } (\lambda n. \text{nnth } nell \ n) \ (\text{nkfilter } P \ 0 \ nell)) =$ 
   $\text{nmap } (\lambda n. \text{nnth } nell \ n) \ (\text{ndropn } k \ (\text{nkfilter } P \ 0 \ nell))$ 
  using ndropn-nmap by blast
have 5:  $\exists x \in \text{nset}(\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell). \ P \ x$ 
  by (metis add.commute add.left-neutral assms(1) in-nset-conv-nnth ndropn-nnth zero-enat-def zero-le)
have 6:  $\text{nfilter } P \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell) =$ 
   $\text{nmap } (\lambda s. \text{nnth } (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell) \ s)$ 
   $(\text{nkfilter } P \ 0 \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell))$ 
  by (metis 5 nkfilter-nmap-nfilter)
have 7:  $\text{nmap } (\lambda s. \text{nnth } (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell) \ s)$ 
   $(\text{nkfilter } P \ 0 \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell)) =$ 
   $\text{nmap } (\lambda s. \text{nnth } nell \ (s + (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k)))$ 
   $(\text{nkfilter } P \ 0 \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell))$ 
  by (simp add: add.commute)
have 8:  $\text{nmap } (\lambda s. \text{nnth } nell \ (s + (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k)))$ 
   $(\text{nkfilter } P \ 0 \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell)) =$ 
   $\text{nmap } (\lambda s. \text{nnth } nell \ s)$ 
   $(\text{nkfilter } P \ (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell))$ 
  using 5 assms(2) nkfilter-nmap-shift-ndropn by fastforce
have 9:  $\text{nmap } (\lambda s. \text{nnth } nell \ s)$ 
   $(\text{nkfilter } P \ (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ k) \ nell)) =$ 
   $\text{nmap } (\lambda s. \text{nnth } nell \ s)$ 
   $(\text{ndropn } k \ (\text{nkfilter } P \ 0 \ nell))$ 
  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 $\exists x \in \text{nset } \text{nell}. P x$
 $k \leq \text{nlength } (\text{nkfilter } P \ 0 \ \text{nell})$
shows $\text{ndropn } k \ (\text{nfilter } P \ \text{nell}) =$
 $\text{nfilter } P \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ \text{nell})$
using *assms nfilter-nkfilter-ndropn*
by (*metis in-nset-conv-nnth minus-nat.diff-0 nkfilter-holds*)

lemma *nfilter-nkfilter-nsubn*:
assumes $P \ (\text{nnth } \text{nell} \ (\ (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ i)))$
 $\text{enat } i \leq \text{nlength } (\text{nkfilter } P \ 0 \ \text{nell})$
 $P \ (\text{nnth } \text{nell} \ (\ (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ j)))$
 $\text{enat } j \leq \text{nlength } (\text{nkfilter } P \ 0 \ \text{nell})$
 $i \leq j$
shows $\text{nsubn } (\text{nfilter } P \ \text{nell} \) \ i \ j =$
 $(\text{nfilter } P \ (\text{nsubn } \text{nell}$
 $\quad (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ i)$
 $\quad (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ j)$
 $\quad)$
 $\quad)$

proof –
have 0: $\text{nsubn } (\text{nfilter } P \ \text{nell} \) \ i \ j = \text{ntaken } (j - i) \ (\text{ndropn } i \ (\text{nfilter } P \ \text{nell}))$
using *nsubn-def1 by blast*
have 1: $\text{ntaken } (j - i) \ (\text{ndropn } i \ (\text{nfilter } P \ \text{nell})) =$
 $\text{ntaken } (j - i) \ (\text{nfilter } P \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ i) \ \text{nell}))$
by (*simp add: assms(1) assms(2) nfilter-nkfilter-ndropn*)
have 4: $((\text{nnth } (\text{nkfilter } P \ 0 \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ i) \ \text{nell})) \ (j - i)) +$
 $\quad (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ i)) =$
 $((\text{nnth } (\text{nkfilter } P \ (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ i) \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ i) \ \text{nell})) \ (j - i)) \)$
proof –
have f1: $\bigwedge n \ ns. (\text{nfirst } (\text{ndropn } n \ ns)::\text{nat}) = \text{nlast } (\text{ntaken } n \ ns)$
by (*metis (lifting) ndropn-0 ndropn-nfirst ndropn-nnth ntaken-ndropn-nlast*)
have $\bigwedge n \ f \ ns. \text{nlast } (\text{ntaken } n \ (\text{nmap } f \ ns)) = (f \ (\text{nlast } (\text{ntaken } n \ ns)::\text{nat}::\text{nat}))$
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: $((\text{nnth } (\text{nkfilter } P \ (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ i) \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ i) \ \text{nell})) \ (j - i)) \)$
 $=$
 $(\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ j)$
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*
 $\text{nfinite-ntaken nkfilter-nlength nnth-beyond nset-nlast ntaken-all}$)
have 10: $\exists x \in \text{nset} \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ i) \ \text{nell}). P x$
by (*metis assms(1) in-nset-conv-nnth le-zero-eq nle-le nnth-zero-ndropn zero-enat-def*)
have 11: $\text{ndropn } i \ (\text{nfilter } P \ \text{nell}) = \text{nfilter } P \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ i) \ \text{nell})$

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 ≤ 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 ≤ 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) ≤ 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 ≤ 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 $\exists x \in \text{nset nell. } P x$
 $n \leq \text{nlength (nfilter P nell)}$
 shows $P (\text{nnth (nfilter P nell) } n)$
 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 $\exists x \in \text{nset } nell. P x$

$\text{nlength}(\text{nfilter } P \text{ nell}) = 0$

shows $(\exists k \leq \text{nlength } nell. P (\text{nnth } nell k) \wedge$
 $(\forall j \leq \text{nlength } nell. j \neq k \longrightarrow \neg P (\text{nnth } nell j)))$

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*: $\neg \text{lnull } nella$

assume *a2*: $Pa x$

assume *a3*: $x \in \text{lset } nella$

assume *a4*: $\text{lnull } (\text{ltl } (\text{lfilter } Pa \text{ nella}))$

show $\exists k. (\text{enat } k \leq \text{llength } (\text{ltl } nella) \longrightarrow$

$Pa (\text{lnth } nella k) \wedge (\forall j. \text{enat } j \leq \text{llength } (\text{ltl } nella) \longrightarrow j \neq k \longrightarrow \neg Pa (\text{lnth } nella j))) \wedge$
 $\text{enat } k \leq \text{llength } (\text{ltl } nella)$

proof –

have *1*: $LCons (\text{lhs } nella) (\text{ltl } nella) = nella$

by (*metis a2 a3 lfilter-LNil llist.disc(1) llist.exhaust-sel lnull-lfilter*)

have *2*: $\text{llength } nella = \text{eSuc } (\text{llength } (\text{ltl } nella))$

by (*metis 1 llength-LCons*)

have *3*: $\neg \text{lnull } (\text{lfilter } Pa \text{ nella})$

by (*meson a2 a3 lnull-lfilter*)

show *?thesis*

by (*metis 2 3 a4 illess-Suc-eq lfilter-llength-one-conv-a lhs-LCons-ltl llength-LCons llength-LNil*
 $\text{llist.collapse}(1) \text{ one-eSuc}$)

qed

qed

lemma *nfilter-nlength-zero-conv-c:*

$(\exists k \leq \text{nlength } nell. P (\text{nnth } nell k) \wedge$

$(\forall j \leq \text{nlength } nell. j \neq k \longrightarrow \neg P (\text{nnth } nell j))) =$

$(\exists k \leq \text{nlength } nell. P (\text{nnth } nell k) \wedge$

$(\forall j \leq \text{nlength } nell. j < k \vee k < j \longrightarrow \neg P (\text{nnth } nell j)))$

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

lemma *nfilter-nlength-zero-conv-d:*

$(\exists k \leq \text{nlength } nell. P (\text{nnth } nell k) \wedge$

$(\forall j \leq \text{nlength } nell. j < k \vee k < j \longrightarrow \neg P (\text{nnth } nell j))) \longleftrightarrow$

$(\exists k \leq \text{nlength } nell. P (\text{nnth } nell k) \wedge$

$(\forall j. j < k \longrightarrow \neg P (\text{nnth } nell j)) \wedge$

$(\forall j \leq \text{nlength } nell. k < j \longrightarrow \neg P (\text{nnth } nell j)))$

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

lemma *nfilter-nlength-zero-conv-b:*

assumes $(\exists k \leq \text{nlength } nell. P (\text{nnth } nell k) \wedge$

$(\forall j \leq \text{nlength } nell. j \neq k \longrightarrow \neg P (\text{nnth } nell j)))$

shows $(\exists x \in \text{nset } nell. P x) \wedge \text{nlength}(\text{nfilter } P \text{ nell}) = 0$

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 ∧ ¬is-NNil x22 ⇒
  nfilter P (ndropn (nnth (nfilter P 0 nell) 0) nell) = nfilter P (ndropn (Suc j) nell)
using Suc NCons
by simp
  (metis Suc.premis(1) dual-order.strict-trans1 nfilter-not-before nnth-0 zero-less-Suc)
have 3: ¬P x21 ∧ ¬is-NNil x22 ⇒
  nfilter P (ndropn (nnth (nfilter P 0 nell) 0) nell) = nfilter P (ndropn (Suc j) nell)
using Suc NCons by (simp add: nfilter-nleast)
show ?thesis
using 1 2 3 by blast
qed
qed
qed

```

lemma *nfilter-nappend-ntaken*:

```

assumes ∃ x ∈ nset (ntaken k nell). P x
        k ≤ nlength nell
shows nfilter P (ntaken k nell) =
  ntaken (the-enat (nlength(nfilter P (ntaken k nell)))) (nfilter P nell)
using assms
apply transfer
proof auto
show ∧ k nell P x.
  ¬ lnull nell ⇒
  enat k ≤ epred (llength nell) ⇒
  x ∈ lset (ltake (enat (Suc k)) nell) ⇒
  P x ⇒
  ∀ x ∈ lset nell. ¬ P x ⇒
  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: ¬ lnull nell
assume a1: enat k ≤ epred (llength nell)
assume a2: x ∈ lset (ltake (enat (Suc k)) nell)
assume a3: P x
assume a4: xa ∈ 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: eSuc (enat k) = llength nell
by (simp add: True a0)
then have llength (lfilter P nell) ≠ 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: llength nell ≠ 0
    using a0 llength-eq-0 by blast
    have g1: ¬ lnull (lfilter P (ltake (enat (Suc k)) nell))
    by (meson a2 a3 lnull-lfilter)
    then show ?thesis
    using f1
    proof -
      have f1: enat k < epred (llength nell)
      using False a1 order.not-eq-order-implies-strict by blast
      have f2: ∀ e. e = 0 ∨ e = eSuc (epred e)
      by (metis co.enat.exhaust-sel)
      then have g2: enat (Suc k) < llength nell
      using f1 by (metis (no-types) Suc-ile-eq ‹llength nell ≠ 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 ‹enat (Suc k) < llength nell› 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 ∃ x ∈ nset (nellx). P x
           ∃ x ∈ 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 $\exists x \in \text{nset } (\text{ndropn } (\text{Suc } k) \text{ nell}). P x$

$\exists x \in \text{nset } (\text{ntaken } k \text{ nell}). P x$

$(\text{Suc } k) \leq \text{nlength nell}$

shows $\text{nfilter } P (\text{ndropn } (\text{Suc } k) \text{ nell}) =$

$\text{ndropn } (\text{the-enat } (\text{eSuc}(\text{nlength}(\text{nfilter } P (\text{ntaken } k \text{ nell})))) (\text{nfilter } P \text{ nell})$

proof –

have 1: $\text{nfinite } (\text{nfilter } P (\text{ntaken } k \text{ nell}))$

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: $\text{nfilter } P \text{ nell} = \text{nappend } (\text{nfilter } P (\text{ntaken } k \text{ nell})) (\text{nfilter } P (\text{ndropn } (\text{Suc } k) \text{ nell}))$

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: $\text{ndropn } (\text{the-enat } (\text{eSuc}(\text{nlength}(\text{nfilter } P (\text{ntaken } k \text{ nell}))))$

$(\text{nappend } (\text{nfilter } P (\text{ntaken } k \text{ nell})) (\text{nfilter } P (\text{ndropn } (\text{Suc } k) \text{ nell})))$

$= (\text{nfilter } P (\text{ndropn } (\text{Suc } k) \text{ nell}))$

by (*metis* 1 *antisym-conv2* *gr-zeroI* *ile-eSuc* *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 $\exists x \in \text{nset } (\text{ntaken } k \text{ nell}). P x$

$k \leq \text{nlength nell}$

shows $\text{nkfilter } P n (\text{ntaken } k \text{ nell}) =$

$\text{ntaken } (\text{the-enat } (\text{nlength}(\text{nkfilter } P n (\text{ntaken } k \text{ nell})))) (\text{nkfilter } P n \text{ nell})$

using *assms*

apply *transfer*

proof (*auto* *simp* *add*: *kfilter-not-lnull-conv* *kfilter-lnull-conv*)

show $\bigwedge k \text{ nell } P n x.$

$\neg \text{lnull nell} \implies$

$\text{enat } k \leq \text{epred } (\text{llength nell}) \implies$

$x \in \text{lset } (\text{ltake } (\text{enat } (\text{Suc } k)) \text{ nell}) \implies$

$P x \implies$

$\forall x \in \text{lset nell}. \neg P x \implies$

$\text{kfilter } P n (\text{ltake } (\text{enat } (\text{Suc } k)) \text{ nell}) =$

$\text{ltake } (\text{enat } (\text{Suc } (\text{the-enat } (\text{epred } (\text{llength } (\text{kfilter } P n (\text{ltake } (\text{enat } (\text{Suc } k)) \text{ nell})))))) (\text{iterates } \text{Suc } n)$

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

next

fix *ka* :: *nat* **and** *nella* :: '*a* *l*list **and** *Pa* :: '*a* \implies *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*: $x \in \text{lset } (\text{ltake } (\text{enat } (\text{Suc } ka)) \text{ nella})$

have *f5*: $\bigwedge e. e = \text{enat } 0 \vee \text{eSuc } (\text{epred } e) = e$

by (*metis* (*no-types*) *co.enat.exhaust-sel* *zero-enat-def*)

```

have f6:  $\bigwedge as. \min \infty (\text{llength } (as::'a \text{ llist})) = \text{llength } as$ 
by simp
have f7:  $eSuc \infty = \infty$ 
by simp
have f8:  $(\text{enat } (Suc (\text{the-enat } (\text{epred } (\text{llength } (\text{kfilter } Pa \ na \ (\text{ltake } (\text{enat } (Suc \ ka)) \ nella)))))) =$ 
 $(\text{llength } (\text{kfilter } Pa \ na \ (\text{ltake } (\text{enat } (Suc \ ka)) \ nella)))$ 
proof -
  have f1:  $\forall n. \neg \infty \leq \text{enat } n$ 
  by (meson infinity-ileE)
  have  $\neg \text{lnull } (\text{kfilter } Pa \ na \ (\text{ltake } (\text{enat } (Suc \ ka)) \ nella))$ 
  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  $\text{llength } nella \neq \infty$ 
  then have  $\text{ltake } (\text{enat } (Suc \ ka)) \ nella = nella \longrightarrow$ 
 $\text{ltake } (\text{llength } (\text{lfilter } Pa \ (\text{ltake } (\text{enat } (Suc \ ka)) \ nella))) (\text{kfilter } Pa \ na \ nella) =$ 
 $\text{kfilter } Pa \ na \ (\text{ltake } (\text{enat } (Suc \ ka)) \ nella) \wedge$ 
 $\text{epred } (\text{llength } (\text{lfilter } Pa \ (\text{ltake } (\text{enat } (Suc \ ka)) \ nella))) \neq \infty$ 
  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  $\text{ltake } (\text{enat } (Suc \ ka)) \ nella \neq nella$ 
    then have  $\text{enat } (Suc \ ka) < \text{llength } nella \wedge \min (\text{enat } (Suc \ ka)) (\text{llength } nella) \neq \infty \vee$ 
 $\text{enat } (Suc \ ka) < \text{llength } nella \wedge \text{llength } (\text{lfilter } Pa \ (\text{ltake } (\text{enat } (Suc \ ka)) \ nella)) \neq eSuc \infty$ 
    by (metis (no-types) enat-ord-code(4) ltake-all min.orderE not-le-imp-less not-less-iff-gr-or-eq) }
  ultimately have  $\text{enat } (Suc \ ka) < \text{llength } nella \wedge \min (\text{enat } (Suc \ ka)) (\text{llength } nella) \neq \infty \vee$ 
 $\text{enat } (Suc \ ka) < \text{llength } nella \wedge \text{llength } (\text{lfilter } Pa \ (\text{ltake } (\text{enat } (Suc \ ka)) \ nella)) \neq eSuc \infty \vee$ 
 $\text{ltake } (\text{llength } (\text{lfilter } Pa \ (\text{ltake } (\text{enat } (Suc \ ka)) \ nella))) (\text{kfilter } Pa \ na \ nella) =$ 
 $\text{kfilter } Pa \ na \ (\text{ltake } (\text{enat } (Suc \ ka)) \ nella) \wedge$ 
 $\text{epred } (\text{llength } (\text{lfilter } Pa \ (\text{ltake } (\text{enat } (Suc \ ka)) \ nella))) \neq \infty$ 
    by fastforce }
  ultimately show  $\text{kfilter } Pa \ na \ (\text{ltake } (\text{enat } (Suc \ ka)) \ nella) =$ 
 $\text{ltake } (\text{enat } (Suc (\text{the-enat } (\text{epred } (\text{llength } (\text{kfilter } Pa \ na \ (\text{ltake } (\text{enat } (Suc \ ka)) \ nella))))))$ 
 $(\text{kfilter } Pa \ na \ nella)$ 
  using f6 f5 a4 a1
  kfilter-lappend-ltake[of  $(\text{enat } (Suc \ ka)) \ nella \ Pa \ na \ ]$ 
  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  $\implies$ 
         $\text{nfilter } P \text{ (ndropn (nnth (nkfilter } P \text{ 0 nell) (Suc 0)) nell) =}$ 
 $\text{nfilter } P \text{ (ndropn (Suc j) nell)}$ 
        by (metis NCons Suc.premis(2) Suc-le-D ndropn-Suc-NCons ndropn-is-NNil)
      have 2:  $P \text{ } x21 \wedge \neg \text{is-NNil } x22 \implies$ 
         $\text{nfilter } P \text{ (ndropn (nnth (nkfilter } P \text{ 0 nell) (Suc 0)) nell) =}$ 
 $\text{nfilter } P \text{ (ndropn (Suc j) nell)}$ 
        using Suc NCons
        proof simp
        assume a1:  $P \text{ } x21 \wedge \neg \text{is-NNil } x22$ 
        assume a2:  $\text{enat (Suc 0) } \leq \text{nlength (nfilter } P \text{ (NCons } x21 \text{ } x22))}$ 
        assume a3:  $\text{nell} = \text{NCons } x21 \text{ } x22$ 
        assume a4:  $\text{Suc } j \leq \text{nnth (nkfilter } P \text{ 0 (NCons } x21 \text{ } x22)) \text{ (Suc 0)}$ 
        have f5:  $\forall \text{ as } p \text{ a } n. ((\exists \text{ a. (a::'a) } \in \text{nset as } \wedge p \text{ a}) \vee \neg p \text{ a}) \vee$ 
 $\text{nkfilter } p \text{ n (NCons a as) = NNil n}$ 
        by (metis (no-types) nkfilter-NCons-a)
        have f6:  $\forall \text{ as } p \text{ n. } (\forall \text{ a. (a::'a) } \notin \text{nset as } \vee \neg p \text{ a}) \vee$ 
 $\text{nlength (nkfilter } p \text{ n as) = nlength (nfilter } p \text{ as)}$ 
        by (meson nkfilter-nlength)
        have f7:  $\forall p \text{ as. } (\forall \text{ a. (a::'a) } \notin \text{nset as } \vee \neg p \text{ a}) = (\forall \text{ a. a } \notin \text{nset as } \vee \neg p \text{ a})$ 
        by meson
        have f8:  $\text{nlength (nkfilter } P \text{ 0 (NCons } x21 \text{ } x22)) =}$ 
 $\text{nlength (nfilter } P \text{ (NCons } x21 \text{ } x22))}$ 
        by (meson a1 nell.set-intros(2) nkfilter-nlength)
        have f9:  $\forall p \text{ as. } (\forall \text{ a. (a::'a) } \notin \text{nset as } \vee \neg p \text{ a}) = (\forall \text{ a. a } \notin \text{nset as } \vee \neg p \text{ a})$ 
        by meson
        have f10:  $(\exists \text{ a. a } \in \text{nset } x22 \wedge P \text{ a}) \longrightarrow$ 
 $\text{nkfilter } P \text{ 0 (NCons } x21 \text{ } x22) = \text{NCons 0 (nkfilter } P \text{ (Suc 0) } x22)$ 
        using a1 by (simp add: Bex-def-raw)
        then have f11:  $(\exists \text{ a. a } \in \text{nset } x22 \wedge P \text{ a}) \longrightarrow$ 
 $\text{nnth (nkfilter } P \text{ (Suc 0) } x22) \text{ 0} - \text{Suc 0} = \text{nnth (nkfilter } P \text{ 0 } x22) \text{ 0}$ 
        using f9 f8 f7 a2 by (metis Suc-ile-eq iless-Suc-eq nkfilter-nnth-n-zero nlength-NCons)
        have f14:  $j < \text{nnth (nkfilter } P \text{ 0 (NCons } x21 \text{ } x22)) \text{ (Suc 0)}$ 

```

```

using a4 Suc-le-eq by blast
have  $\exists a. a \in \text{nset } x22 \wedge P a$ 
using f8 f5 a2 a1 by (metis Suc-ile-eq gr-implies-not-zero nlength-NNil)
then have  $(\exists a. a \in \text{nset } x22 \wedge P a) \wedge \text{nfilter } P (\text{ndropn } (\text{nnth } (\text{nkfilter } P 0 x22) 0) x22) =$ 
 $\text{nfilter } P (\text{ndropn } j x22)$ 
using f14 f11 a4
proof –
have  $j \leq \text{nnth } (\text{nkfilter } P 0 x22) 0$ 
using  $\langle \exists a. a \in \text{nset } x22 \wedge P a \rangle$  f10 f11 f14 by fastforce
then show ?thesis
by (simp add: Bex-def-raw  $\langle \exists a. a \in \text{nset } x22 \wedge P a \rangle$  nfilter-ndropns-nmap-help-0)
qed
then show  $\text{nfilter } P (\text{ndropn } (\text{nnth } (\text{nkfilter } P 0 (\text{NCons } x21 x22)) (\text{Suc } 0)) (\text{NCons } x21 x22)) =$ 
 $\text{nfilter } P (\text{ndropn } j x22)$ 
using f11 a4
by (metis One-nat-def Suc-le-D diff-Suc-1 f10 ndropn-Suc-NCons nnth-Suc-NCons)
qed
have  $\exists: \neg P x21 \wedge \neg \text{is-NNil } x22 \implies$ 
 $\text{nfilter } P (\text{ndropn } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) (\text{Suc } 0)) \text{ nell}) =$ 
 $\text{nfilter } P (\text{ndropn } (\text{Suc } j) \text{ nell})$ 
using Suc NCons
proof simp
assume a1:  $\bigwedge \text{nell. } [\exists a \in \text{nset } \text{nell. } P a; j \leq \text{nnth } (\text{nkfilter } P 0 \text{ nell}) (\text{Suc } 0);$ 
 $\text{nnth } (\text{nkfilter } P 0 \text{ nell}) 0 < j; \text{enat } (\text{Suc } 0) \leq \text{nlength } (\text{nfilter } P \text{ nell})] \implies$ 
 $\text{nfilter } P (\text{ndropn } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) (\text{Suc } 0)) \text{ nell}) =$ 
 $\text{nfilter } P (\text{ndropn } j \text{ nell})$ 
assume a2:  $\text{enat } (\text{Suc } 0) \leq \text{nlength } (\text{nfilter } P x22)$ 
assume a3:  $\text{Suc } j \leq \text{nnth } (\text{nkfilter } P (\text{Suc } 0) x22) (\text{Suc } 0)$ 
assume a4:  $\exists x \in \text{nset } x22. P x$ 
assume a5:  $\text{nnth } (\text{nkfilter } P (\text{Suc } 0) x22) 0 < \text{Suc } j$ 
have f12:  $\text{nnth } (\text{nkfilter } P 0 x22) (\text{Suc } 0) =$ 
 $(\text{nnth } (\text{nkfilter } P (\text{Suc } 0) x22) (\text{Suc } 0)) - (\text{Suc } 0)$ 
using nkfilter-nnth-n-zero[of x22 P Suc 0 Suc 0]
by (simp add: a2 a4 nkfilter-nlength)
then have f13:  $j \leq \text{nnth } (\text{nkfilter } P 0 x22) (\text{Suc } 0)$ 
using a3 by linarith
have f14:  $\text{enat } 0 \leq \text{nlength } (\text{nfilter } P x22)$ 
using a2 by (metis One-nat-def one-enat-def order.trans zero-enat-def zero-le-one)
have f15:  $\text{nlength } (\text{nkfilter } P (\text{Suc } 0) x22) = \text{nlength } (\text{nfilter } P x22)$ 
by (simp add: a4 nkfilter-nlength)
then have  $\text{Suc } 0 \leq \text{nnth } (\text{nkfilter } P (\text{Suc } 0) x22) 0$ 
by (simp add: a4 f14 nkfilter-lowerbound)
then have  $\text{Suc } (\text{nnth } (\text{nkfilter } P 0 x22) 0) \leq j$ 
by (metis a4 a5 add-Suc leD nkfilter-nleast not-less-eq-eq)
then have  $\text{nfilter } P (\text{ndropn } (\text{nnth } (\text{nkfilter } P 0 x22) (\text{Suc } 0)) x22) =$ 
 $\text{nfilter } P (\text{ndropn } j x22)$ 
by (simp add: Suc.IH Suc-le-lessD a2 a4 f13)
then show  $\text{nfilter } P (\text{ndropn } (\text{nnth } (\text{nkfilter } P (\text{Suc } 0) x22) (\text{Suc } 0)) (\text{NCons } x21 x22)) =$ 
 $\text{nfilter } P (\text{ndropn } j x22)$ 
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  $\exists x \in \text{nset } nell. P x$ 
 $j \leq \text{nnth}(\text{nkfilter } P \ 0 \ nell) \ (Suc \ i)$ 
 $\text{nnth} \ (\text{nkfilter } P \ 0 \ nell) \ i < j$ 
 $(Suc \ i) \leq \text{nlength}(\text{nfilter } P \ nell)$ 
shows  $(\text{nfilter } P \ (\text{ndropn} \ (\text{nnth} \ (\text{nkfilter } P \ 0 \ nell) \ (Suc \ i)) \ nell)) =$ 
 $(\text{nfilter } P \ (\text{ndropn } j \ nell))$ 
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  $\implies$ 
 $\text{nfilter } P \ (\text{ndropn} \ (\text{nnth} \ (\text{nkfilter } P \ 0 \ nell) \ (Suc \ i)) \ nell) =$ 
 $\text{nfilter } P \ (\text{ndropn} \ (Suc \ j) \ nell)$ 
by (metis NCons Suc.prem1(2) Suc-le-D ndropn-Suc-NCons ndropn-is-NNil)
have 2:  $\neg \text{is-NNil } x22 \wedge i = 0 \implies$ 
 $\text{nfilter } P \ (\text{ndropn} \ (\text{nnth} \ (\text{nkfilter } P \ 0 \ nell) \ (Suc \ i)) \ nell) =$ 
 $\text{nfilter } P \ (\text{ndropn} \ (Suc \ j) \ nell)$ 
using Suc nfilter-ndropns-nmap-help-1[of nell P ] by metis
have 3:  $P \ x21 \wedge \neg \text{is-NNil } x22 \wedge 0 < i \implies$ 
 $\text{nfilter } P \ (\text{ndropn} \ (\text{nnth} \ (\text{nkfilter } P \ 0 \ nell) \ (Suc \ i)) \ nell) = \text{nfilter } P \ (\text{ndropn} \ (Suc \ j) \ nell)$ 
using Suc NCons
proof simp
assume a1:  $\bigwedge nell \ i. [\exists a \in \text{nset } nell. P \ a; j \leq \text{nnth} \ (\text{nkfilter } P \ 0 \ nell) \ (Suc \ i);$ 
 $\text{nnth} \ (\text{nkfilter } P \ 0 \ nell) \ i < j; \text{enat} \ (Suc \ i) \leq \text{nlength} \ (\text{nfilter } P \ nell)] \implies$ 
 $\text{nfilter } P \ (\text{ndropn} \ (\text{nnth} \ (\text{nkfilter } P \ 0 \ nell) \ (Suc \ i)) \ nell) =$ 
 $\text{nfilter } P \ (\text{ndropn } j \ nell)$ 
assume a2:  $P \ x21 \wedge \neg \text{is-NNil } x22 \wedge 0 < i$ 
assume a3:  $\text{enat} \ (Suc \ i) \leq \text{nlength} \ (\text{nfilter } P \ (NCons \ x21 \ x22))$ 
assume a4:  $nell = NCons \ x21 \ x22$ 

```

```

assume a5:  $\text{Suc } j \leq \text{nnth } (\text{nkfilter } P \ 0 \ (\text{NCons } x21 \ x22)) \ (\text{Suc } i)$ 
assume a6:  $\text{nnth } (\text{nkfilter } P \ 0 \ (\text{NCons } x21 \ x22)) \ i < \text{Suc } j$ 
show  $\text{nfilter } P \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ (\text{NCons } x21 \ x22)) \ (\text{Suc } i)) \ (\text{NCons } x21 \ x22)) =$ 
 $\text{nfilter } P \ (\text{ndropn } j \ x22)$ 
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:  $\text{nlength } (\text{nkfilter } P \ 0 \ (\text{NCons } x21 \ x22)) = \text{nlength } (\text{nfilter } P \ (\text{NCons } x21 \ x22))$ 
    using a4 by (metis (no-types) Suc(2) nkfilter-nlength)
  have f2:  $\text{nkfilter } P \ 0 \ (\text{NCons } x21 \ x22) = \text{NCons } 0 \ (\text{nkfilter } P \ (\text{Suc } 0) \ x22)$ 
    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:  $\text{nnth } (\text{nkfilter } P \ 0 \ (\text{NCons } x21 \ x22)) \ (\text{Suc } i) =$ 
 $\text{nnth } ( (\text{nkfilter } P \ (\text{Suc } 0) \ x22)) \ (i)$ 
    by (simp add: f2)
  have f4:  $\text{enat } i \leq \text{nlength } (\text{nkfilter } P \ (\text{Suc } 0) \ x22)$ 
    by (metis Suc-ile-eq a3 f1 f2 iless-Suc-eq nlength-NCons)
  have f5:  $\text{nnth } ( (\text{nkfilter } P \ (\text{Suc } 0) \ x22)) \ (i) = (\text{Suc } 0) + \text{nnth } ( (\text{nkfilter } P \ 0 \ x22)) \ (i)$ 
    using nkfilter-nnth-n-zero[of x22 P i Suc 0]
    using a5 f0 f3 f4 by linarith
  have f6:  $\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ (\text{NCons } x21 \ x22)) \ (\text{Suc } i)) \ (\text{NCons } x21 \ x22) =$ 
 $\text{ndropn } ((\text{Suc } 0) + \text{nnth } ( (\text{nkfilter } P \ 0 \ x22)) \ (i)) \ (\text{NCons } x21 \ x22)$ 
    using f3 f5 by presburger
  have f7:  $\text{ndropn } ((\text{Suc } 0) + \text{nnth } ( (\text{nkfilter } P \ 0 \ x22)) \ (i)) \ (\text{NCons } x21 \ x22) =$ 
 $\text{ndropn } (\text{nnth } ( (\text{nkfilter } P \ 0 \ x22)) \ (i)) \ x22$ 
    using ndropn-Suc-NCons by auto
  have f8:  $j \leq \text{nnth } (\text{nkfilter } P \ 0 \ x22) \ (i)$ 
    using a5 f3 f5 by linarith
  have f11:  $\text{nnth } (\text{nkfilter } P \ 0 \ (\text{NCons } x21 \ x22)) \ i = \text{nnth } ( (\text{nkfilter } P \ (\text{Suc } 0) \ x22)) \ (i-1)$ 
    by (metis One-nat-def Suc-pred a2 f2 nnth-Suc-NCons)
  have f12:  $\text{enat } (i - 1) \leq \text{nlength } (\text{nkfilter } P \ (\text{Suc } 0) \ x22)$ 
    by (metis One-nat-def Suc-ile-eq Suc-pred a2 f4 less-imp-le)
  have f13:  $(\text{Suc } 0) \leq \text{nnth } ( (\text{nkfilter } P \ (\text{Suc } 0) \ x22)) \ (i-1)$ 
    by (meson f0 f12 nkfilter-lowerbound)
  have f14:  $\text{nnth } ( (\text{nkfilter } P \ (\text{Suc } 0) \ x22)) \ (i-1) = (\text{Suc } 0) + \text{nnth } ( (\text{nkfilter } P \ 0 \ x22)) \ (i-1)$ 
    using nkfilter-nnth-n-zero[of x22 P i-1 Suc 0] f12 f0 f13 by (metis le-add-diff-inverse)
  have f9:  $\text{nnth } (\text{nkfilter } P \ 0 \ x22) \ (i-1) < j$ 
    using a6 f11 f14 by linarith
  have f10:  $i \leq \text{nlength } (\text{nfilter } P \ x22)$ 
    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:  $\neg P \ x21 \wedge \neg \text{is-NNil } x22 \wedge 0 < i \implies$ 
 $\text{nfilter } P \ (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ (\text{Suc } i)) \ \text{nell}) =$ 
 $\text{nfilter } P \ (\text{ndropn } (\text{Suc } j) \ \text{nell})$ 
using Suc NCons
proof simp

```

```

assume a0:  $\neg P\ x21 \wedge \neg is\_NNil\ x22 \wedge 0 < i$ 
assume a2:  $\bigwedge nell\ i. \llbracket \exists a \in nset\ nell. P\ a; j \leq nnth\ (nkfilter\ P\ 0\ nell)\ (Suc\ i);$ 
 $nnth\ (nkfilter\ P\ 0\ nell)\ i < j; enat\ (Suc\ i) \leq nlength\ (nfilter\ P\ nell) \rrbracket \implies$ 
 $nfilter\ P\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ (Suc\ i))\ nell) =$ 
 $nfilter\ P\ (ndropn\ j\ nell)$ 
assume a1:  $\exists x \in nset\ x22. P\ x$ 
assume a4:  $Suc\ j \leq nnth\ (nkfilter\ P\ (Suc\ 0)\ x22)\ (Suc\ i)$ 
assume a5:  $nnth\ (nkfilter\ P\ (Suc\ 0)\ x22)\ i < Suc\ j$ 
assume a3:  $enat\ (Suc\ i) \leq nlength\ (nfilter\ P\ x22)$ 
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.premis(1) nkfilter-nlength)
have f2:  $nkfilter\ P\ 0\ (NCons\ x21\ x22) = (nkfilter\ P\ (Suc\ 0)\ x22)$ 
using NCons Suc.premis(2) 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:  $enat\ (Suc\ i) \leq nlength\ (nkfilter\ P\ (Suc\ 0)\ x22)$ 
using NCons Suc.premis(4) 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:  $j \leq nnth\ (nkfilter\ P\ 0\ x22)\ (Suc\ i)$ 
using a4 f5 by force
have f11:  $nnth\ (nkfilter\ P\ 0\ (NCons\ x21\ x22))\ i = nnth\ ((nkfilter\ P\ (Suc\ 0)\ x22))\ (i)$ 
by (simp add: f2)
have f12:  $enat\ (i) \leq nlength\ (nkfilter\ P\ (Suc\ 0)\ x22)$ 
using Suc-ile-eq f4 by auto
have f13:  $(Suc\ 0) \leq nnth\ ((nkfilter\ P\ (Suc\ 0)\ x22))\ (i)$ 
by (simp add: a1 f12 nkfilter-lowerbound)
have f14:  $nnth\ ((nkfilter\ P\ (Suc\ 0)\ x22))\ (i) = (Suc\ 0) + nnth\ ((nkfilter\ P\ 0\ x22))\ (i)$ 
by (metis a1 f12 f13 le-add-diff-inverse nkfilter-nnth-n-zero)
have f9:  $nnth\ (nkfilter\ P\ 0\ x22)\ (i) < j$ 
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 (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: $(\text{nfilter } P (\text{ndropn } (\text{nnth } (\text{nkfilter } P 0 (\text{ndropns nell})) (\text{Suc } i)) (\text{ndropns nell}))) =$

$(\text{nfilter } P (\text{ndropn } (\text{Suc } (\text{nnth } (\text{nkfilter } P 0 (\text{ndropns nell})) i)) (\text{ndropns nell})))$

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 $(\exists i. \text{enat } i \leq \text{nlength } \text{nell} \wedge \text{nnth } (\text{nfilter } P (\text{ndropns nell})) j = \text{ndropn } i \text{ nell})$

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 $nlength\ ((n nth\ (nfilter\ P\ (ndropns\ xs))\ j)) \leq nlength\ xs$
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 $(Suc\ k) \leq nlength\ nell$
 $\exists\ x \in nset\ (ndropns\ (ndropn\ (Suc\ k)\ nell)).\ P\ x$
 $\exists\ x \in nset\ (ntaken\ k\ (ndropns\ nell)).\ P\ x$
shows $(nfilter\ P\ (ndropns\ (ndropn\ (Suc\ k)\ nell))) =$
 $(ndropn\ (the-enat\ (eSuc\ (nlength\ (nfilter\ P\ (ntaken\ k\ (ndropns\ nell))))))\ (nfilter\ P\ (ndropns\ nell)))$
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 $k \leq nlength\ (nfilter\ P\ (ndropns\ nell))$
 $\exists\ x \in nset\ (ndropns\ nell).\ P\ x$
shows $ndropn\ k\ (nfilter\ P\ (ndropns\ nell)) =$
 $nfilter\ P\ (ndropns\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ (ndropns\ nell))\ k)\ nell))$
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 $\exists\ x \in nset\ nell.\ P\ x \wedge Q\ x$
shows $nlength\ (nfilter\ (\lambda x.\ P\ x \wedge Q\ x)\ nell) \leq nlength\ (nfilter\ P\ nell)$
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 nellx :: '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 nellx)))) nellya)
    then iterates Suc (ka + the-enat (epred (llength nellx)))
    else kfilter Pa (ka + the-enat (epred (llength nellx))) nellya = LCons b LNil
assume a2:  $\neg$  lnull nellya
assume a3: Pa (lhd nellya)
assume a4:  $\neg$  lnull (kfilter Pa (ka + the-enat (llength nellx)) (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-llnull-conv llength-LNil llength-eq-0 llist.set-sel(1))
then show lappend (kfilter Pa ka nellx) (kfilter Pa (ka + the-enat (llength nellx)) (ltl nellya)) = iterates
Suc ka
by meson
next
fix nellx :: '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 nellx)))) nellya)
    then iterates Suc (ka + the-enat (epred (llength nellx)))
    else kfilter Pa (ka + the-enat (epred (llength nellx))) nellya = LCons b LNil
assume a2:  $\neg$  lnull nellya
assume a3: Pa (lhd nellya)
assume a4:  $\neg$  lnull (kfilter Pa (ka + the-enat (llength nellx)) (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-llnull-conv)
then show lappend (kfilter Pa ka nellx) (kfilter Pa (ka + the-enat (llength nellx)) (ltl nellya)) =
    kfilter Pa ka nellx
by meson
qed
next
show nlast nellx = nfirst nelly  $\Longrightarrow$ 
    P (nfirst nelly)  $\Longrightarrow$ 
    nfinite nellx  $\Longrightarrow$ 
     $\neg$  is-NNil nelly  $\Longrightarrow$ 
     $\neg$  is-NNil (nkfilter P (k + the-enat (nlength nellx)) nelly)  $\Longrightarrow$ 
    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-llnull-conv split:if-split-asm)
show f1:  $\bigwedge$  nellx nelly P k x xa.
     $\neg$  lnull nellx  $\Longrightarrow$ 

```


$\neg \text{lnull nelly} \implies$
 $\text{llast nellx} = \text{lhs nelly} \implies$
 $P (\text{lhs nelly}) \implies$
 $\text{lfinite nellx} \implies$
 $\forall b. \text{nelly} \neq \text{LCons } b \text{ LNil} \implies$
 $\forall b. \text{kfilter } P (k + \text{the-enat } (\text{epred } (\text{llength nellx}))) \text{ nelly} \neq \text{LCons } b \text{ LNil} \implies$
 $x \in \text{lset nelly} \implies P x \implies \forall x \in \text{lset nellx}. \neg P x \implies P xa \implies xa \in \text{lset } (\text{ltl nelly}) \implies$
 $\text{kfilter } P k (\text{lappend nellx } (\text{ltl nelly})) = \text{iterates Suc } k$
by (metis in-lset-lappend-iff lappend-lbutlast-llast-id lbutlast-lfinite llist.set-intros(1))
next
show f2: $\bigwedge \text{nellx nelly } P k x xa xb.$
 $\neg \text{lnull nellx} \implies$
 $\neg \text{lnull nelly} \implies$
 $\text{llast nellx} = \text{lhs nelly} \implies$
 $P (\text{lhs nelly}) \implies$
 $\text{lfinite nellx} \implies$
 $\forall b. \text{nelly} \neq \text{LCons } b \text{ LNil} \implies$
 $\forall b. \text{kfilter } P (k + \text{the-enat } (\text{epred } (\text{llength nellx}))) \text{ nelly} \neq \text{LCons } b \text{ LNil} \implies$
 $x \in \text{lset nelly} \implies$
 $P x \implies$
 $xa \in \text{lset nellx} \implies$
 $P xa \implies$
 $P xb \implies$
 $xb \in \text{lset nellx} \implies$
 $\text{kfilter } P k (\text{lappend nellx } (\text{ltl nelly})) =$
 $\text{lappend } (\text{kfilter } P k \text{ nellx}) (\text{ltl } (\text{kfilter } P (k + \text{the-enat } (\text{epred } (\text{llength nellx}))) \text{ nelly}))$
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: $\neg \text{lnull nellya}$
assume a2: Pa (lhs nellya)
assume a3: llast nellxa = lhs nellya
assume a4: $\neg \text{lnull nellxa}$
assume a5: lfinite nellxa
have f6: nellya = LCons (lhs nellya) (ltl nellya)
using a1 **by** auto
have f7: LCons (lhs nellya) (ltl nellya) \neq LNil \wedge lhs (LCons (lhs nellya) (ltl nellya)) = lhs nellya \wedge
ltl (LCons (lhs nellya) (ltl nellya)) = ltl nellya
by auto
then have f8: kfilter Pa (the-enat (epred (llength nellxa)) + ka) (LCons (lhs 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: $\forall bs p n \text{bsa}. \neg \text{lfinite } (\text{bs}::'b \text{ llist}) \vee \text{kfilter } p n (\text{lappend bs bsa}) =$
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  $\bigwedge nellx\ nelly\ P\ k\ x\ xa\ xb.$ 
   $\neg lnull\ nellx \implies$ 
   $\neg lnull\ nelly \implies$ 
   $llast\ nellx = lhd\ nelly \implies$ 
   $P\ (lhd\ nelly) \implies$ 
   $lfinite\ nellx \implies$ 
   $\forall b. nelly \neq LCons\ b\ LNil \implies$ 
   $\forall b. kfilter\ P\ (k + the-enat\ (epred\ (llength\ nellx)))\ nelly \neq LCons\ b\ LNil \implies$ 
   $x \in lset\ nelly \implies$ 
   $P\ x \implies$ 
   $xa \in lset\ nellx \implies$ 
   $P\ xa \implies$ 
   $P\ xb \implies$ 
   $xb \in lset\ (ltl\ nelly) \implies$ 
   $kfilter\ P\ k\ (lappend\ nellx\ (ltl\ nelly)) =$ 
   $lappend\ (kfilter\ P\ k\ nellx)\ (ltl\ (kfilter\ P\ (k + the-enat\ (epred\ (llength\ nellx)))\ nelly))$ 
  by (simp add: f2)
qed
qed

```

lemma *nleast-conv*:

```

assumes  $\exists x \in nset\ nellx. P\ x$ 
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:  $\exists x \in \text{nset } \text{nellx}. P\ x$ 
    using assms nset-nlast by blast
  have 2:  $\exists x \in \text{nset } (\text{ntl } \text{nelly}). P\ x$ 
    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:  $\neg \text{is-NNil } (\text{nfilter } P\ \text{nelly}) \implies$ 
     $\text{nfilter } P\ (\text{nappend } \text{nellx } (\text{ntl } \text{nelly})) = \text{nappend } (\text{nfilter } P\ \text{nellx})\ (\text{nfilter } P\ (\text{ntl } \text{nelly}))$ 
    by (simp add: 1 2 assms(3) nappend-nfilter-nfinite)
  have 4:  $\text{is-NNil } (\text{nfilter } P\ \text{nelly}) \implies \text{nfilter } P\ \text{nellx} = \text{nappend } (\text{nfilter } P\ \text{nellx})\ (\text{nfilter } P\ (\text{ntl } \text{nelly}))$ 
    by (simp add: False)
  have 5:  $\text{nfilter } P\ (\text{nfuse } \text{nellx } \text{nelly}) = \text{nfilter } P\ (\text{nappend } \text{nellx } (\text{ntl } \text{nelly}))$ 
    unfolding nfuse-def1
    by (metis (full-types) False nellist.collapse(1) nfilter-NNil)
  have 6:  $(\text{ntl } (\text{nfilter } P\ \text{nelly})) = (\text{nfilter } P\ (\text{ntl } \text{nelly}))$ 
    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  $n \leq \text{nlength } \text{nellx}$ 
   $P\ (\text{nlast } (\text{ntaken } n\ \text{nellx}))$ 
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  $n \leq \text{nlength } \text{nellx}$ 
   $P\ (\text{nlast } (\text{ntaken } n\ \text{nellx}))$ 

```

shows $ntaken (the-enat (nlength(nfilter P (ntaken n nellx)))) (nfilter P nellx) =$
 $(nfilter P (ntaken n nellx))$
using *assms nfilter-nappend-ntaken* **by** $(metis nfinite-ntaken nset-nlast)$

lemma *nfilter-nlast*:

assumes $n \leq nlength\ nellx$
 $P (nlast (ntaken n nellx))$
shows $nlast(nfilter P (ntaken n nellx)) = (nlast(ntaken n nellx))$
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) ≤ 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 ≤ 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.premis(1) 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: ∧ nellx k. [[enat n ≤ nlength nellx; P (nlast (ntaken n nellx))]] ⇒
  nlast (nkfilter P k (ntaken n nellx)) = k + n
assume enat (Suc n) ≤ eSuc (nlength x22)
then have enat n < eSuc (nlength x22)
using Suc-ile-eq by blast
then have f3: enat n ≤ nlength x22
by (meson iless-Suc-eq)
have ∃ a. a ∈ nset (ntaken n x22) ∧ 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 $n \leq nlength\ nellx$
 $P \ (nlast\ (ntaken\ n\ nellx))$
shows $ndropn\ (the-enat\ (nlength\ (nkfilter\ P\ k\ (ntaken\ n\ nellx))))\ (nkfilter\ P\ k\ nellx) =$
 $(nkfilter\ P\ (k+n)\ (ndropn\ n\ nellx))$
proof –
have 1: $(nkfilter\ P\ k\ nellx) = nfuse\ (nkfilter\ P\ k\ (ntaken\ n\ nellx))\ (nkfilter\ P\ (k+n)\ (ndropn\ n\ nellx))$
by (*simp add: assms nkfilter-chop1*)
have 2: $nlast\ (nkfilter\ P\ k\ (ntaken\ n\ nellx)) = nfirst\ (nkfilter\ P\ (k+n)\ (ndropn\ n\ nellx))$
by (*metis assms(1) assms(2) ndropn-nfirst nkfilter-nfirst nkfilter-nlast ntaken-nlast*)
have 3: $ndropn\ (the-enat\ (nlength\ (nkfilter\ P\ k\ (ntaken\ n\ nellx))))$
 $(nfuse\ (nkfilter\ P\ k\ (ntaken\ n\ nellx))\ (nkfilter\ P\ (k+n)\ (ndropn\ n\ nellx))) =$
 $(nkfilter\ P\ (k+n)\ (ndropn\ n\ nellx))$
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 $n \leq nlength\ nellx$
 $P \ (nlast\ (ntaken\ n\ nellx))$
shows $ntaken\ (the-enat\ (nlength\ (nkfilter\ P\ k\ (ntaken\ n\ nellx))))\ (nkfilter\ P\ k\ nellx) =$
 $(nkfilter\ P\ k\ (ntaken\ n\ nellx))$

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: $\text{nlast}(\text{nkfilter } P \ k \ (\text{ntaken } n \ \text{nellx})) = \text{nfir}(\text{nkfilter } P \ (k+n) \ (\text{ndropn } n \ \text{nellx}))$
by (*metis assms(1) assms(2) ndropn-nfir nkfilter-nfir 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: $\text{ntaken } (\text{the-enat } (\text{nlength}(\text{nfilter } P \ (\text{ntaken } (j-i) \ (\text{ndropn } i \ \text{nellx}))))$
 $(\text{nfilter } P \ (\text{ndropn } i \ \text{nellx})) =$
 $\text{ntaken } (\text{the-enat } (\text{nlength}(\text{nfilter } P \ (\text{ntaken } (j-i) \ (\text{ndropn } i \ \text{nellx}))))$
 $(\text{ndropn } (\text{the-enat } (\text{nlength}(\text{nfilter } P \ (\text{ntaken } i \ \text{nellx})))) (\text{nfilter } P \ \text{nellx}))$

by (*simp add: assms(3) assms(4) nfilter-chop1-ndropn ntaken-nlast*)

have 4: $\text{ntaken } (\text{the-enat } (\text{nlength}(\text{nfilter } P \ (\text{ntaken } (j-i) \ (\text{ndropn } i \ \text{nellx}))))$
 $(\text{ndropn } (\text{the-enat } (\text{nlength}(\text{nfilter } P \ (\text{ntaken } i \ \text{nellx})))) (\text{nfilter } P \ \text{nellx})) =$
 $\text{nsubn } (\text{nfilter } P \ \text{nellx})$
 $(\text{the-enat } (\text{nlength}(\text{nfilter } P \ (\text{ntaken } i \ \text{nellx}))))$
 $((\text{the-enat } (\text{nlength}(\text{nfilter } P \ (\text{ntaken } (j-i) \ (\text{ndropn } i \ \text{nellx})))) +$
 $(\text{the-enat } (\text{nlength}(\text{nfilter } P \ (\text{ntaken } i \ \text{nellx}))))$

using *ntaken-ndropn* **by** *blast*

have 5: $((\text{the-enat } (\text{nlength}(\text{nfilter } P \ (\text{ntaken } (j-i) \ (\text{ndropn } i \ \text{nellx})))) +$
 $(\text{the-enat } (\text{nlength}(\text{nfilter } P \ (\text{ntaken } i \ \text{nellx})))) =$
 $((\text{the-enat } (\text{nlength}(\text{nfilter } P \ (\text{nsubn } \text{nellx } i \ j)))) +$
 $(\text{the-enat } (\text{nlength}(\text{nfilter } P \ (\text{ntaken } i \ \text{nellx}))))$

using 1 **by** *auto*

have 6: $\text{ntaken } j \ \text{nellx} = \text{nfuse } (\text{ntaken } i \ \text{nellx}) \ (\text{nsubn } \text{nellx } i \ j)$

using *nsubn-nfuse[of 0 i j nellx]*

by (*simp add: assms(1) assms(5) nsubn-def1*)

have 7: $\text{nlength } (\text{nfilter } P \ (\text{nfuse } (\text{ntaken } i \ \text{nellx}) \ (\text{nsubn } \text{nellx } i \ j))) =$

```

      (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)

end

```

```

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

```

4 Finite and Infinite ITL 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.

4.1 Types of Formulas

To mechanise the Finite and Infinite ITL semantics, the following type abbreviations are used:

type-synonym $'a \text{ intervals} = 'a \text{ nellist}$

type-synonym $('a, 'b) \text{ formfun} = 'a \text{ intervals} \Rightarrow 'b$

type-synonym $'a \text{ formula} = ('a, \text{bool}) \text{ formfun}$

type-synonym $('a, 'b) \text{ stfun} = 'a \Rightarrow 'b$

type-synonym $'a \text{ stpred} = ('a, \text{bool}) \text{ stfun}$

instance

$\text{fun} :: (\text{type}, \text{type}) \text{ world} \dots$

instance

$\text{prod} :: (\text{type}, \text{type}) \text{ world} \dots$

instance

$\text{sum} :: (\text{type}, \text{type}) \text{ world} \dots$

instance

$\text{nellist} :: (\text{type}) \text{ world} \dots$

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.

4.2 Semantics of ITL

The semantics of ITL is defined.

definition $\text{skip-d} :: ('a :: \text{world}) \text{ formula}$

where $\text{skip-d} \equiv (\lambda s. \text{nlength } s = (\text{enat } (\text{Suc } 0)))$

definition $\text{chop-d} :: ('a :: \text{world}) \text{ formula} \Rightarrow ('a :: \text{world}) \text{ formula} \Rightarrow ('a :: \text{world}) \text{ formula}$

where $\text{chop-d } F1 \ F2 \equiv$

$(\lambda s. \\ \quad (\exists n \leq \text{nlength } s. ((\text{ntaken } n \ s) \models F1) \wedge ((\text{ndropn } n \ s) \models F2)) \\ \quad \vee (\neg \text{nfinite } s \wedge (s \models F1)) \\)$

definition $\text{current-val-d} :: ('a :: \text{world}, 'b) \text{ stfun} \Rightarrow ('a, 'b) \text{ formfun}$

where $\text{current-val-d } f = (\lambda s. ((\text{nfirst } s) \models f))$

definition $\text{next-val-d} :: ('a :: \text{world}, 'b) \text{ stfun} \Rightarrow ('a, 'b) \text{ formfun}$

where $\text{next-val-d } f \equiv$

$(\lambda s. (\text{if } \text{nlength } s \neq (\text{enat } 0) \text{ then } ((\text{nnth } s \ 1) \models f) \text{ else } (\epsilon \ (x :: 'b). \ x = x)) \\)$

definition $\text{fin-val-d} :: ('a :: \text{world}, 'b) \text{ stfun} \Rightarrow ('a, 'b) \text{ formfun}$

where $\text{fin-val-d } f \equiv \lambda s. ((\text{if } \text{nfinite } s \text{ then } ((\text{nlast } s) \models f) \text{ else } (\epsilon \ (x :: 'b). \ x = x)))$

definition *penult-val-d* :: ('a::world,'b) stfun \Rightarrow ('a,'b) formfun

where *penult-val-d* $f \equiv$

```
( $\lambda$  s.
  (if nfinite s
    then (if nlength s  $\neq$  (enat 0)
      then ((nnth s (the-enat (epred(nlength s))))  $\models$  f)
      else ( $\epsilon$  (x::'b). x=x))
    else ( $\epsilon$  (x::'b). x=x)
  )
)
```

syntax

```
-skip-d      :: lift      ((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      ((TEMP -))
```

syntax (ASCII)

```
-skip-d      :: lift      ((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)
```

translations

```
-skip-d       $\equiv$  CONST skip-d
-chop-d       $\equiv$  CONST chop-d
-current-val-d  $\equiv$  CONST current-val-d
-next-val-d   $\equiv$  CONST next-val-d
-fin-val-d    $\equiv$  CONST fin-val-d
-penult-val-d  $\equiv$  CONST penult-val-d
TEMP F       $\rightarrow$  (F:: (- intervals)  $\Rightarrow$  -)
```

4.3 Abbreviations

Some standard temporal abbreviations, with their concrete syntax.

definition *infinite-d* :: ('a :: world) formula

where *infinite-d* \equiv LIFT(# True;# False)

syntax

```
-infinite-d :: lift      (inf)
```

syntax (ASCII)

```
-infinite-d :: lift      (inf)
```

translations

$-infinite-d \Rightarrow CONST\ infinite-d$

definition $finite-d :: ('a :: world)\ formula$

where $finite-d \equiv LIFT(\neg(inf))$

syntax

$-finite-d \quad :: lift \quad (finite)$

syntax (ASCII)

$-finite-d \quad :: lift \quad (finite)$

translations

$-finite-d \Rightarrow CONST\ finite-d$

definition $schop-d :: ('a :: world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$

where $schop-d\ F1\ F2 \equiv LIFT((F1 \wedge finite);F2)$

definition $sometimes-d :: ('a :: world)\ formula \Rightarrow 'a\ formula$

where $sometimes-d\ F \equiv LIFT(finite;F)$

definition $di-d :: ('a :: world)\ formula \Rightarrow 'a\ formula$

where $di-d\ F \equiv LIFT(F;\#True)$

definition $da-d :: ('a :: world)\ formula \Rightarrow 'a\ formula$

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

$-schop-d \quad :: [lift, lift] \Rightarrow lift\ ((- \curvearrowright -)\ [84,84]\ 83)$

$-sometimes-d \quad :: lift \Rightarrow lift\ ((\Diamond -)\ [88]\ 87)$

$-di-d \quad :: lift \Rightarrow lift\ ((di -)\ [88]\ 87)$

$-da-d \quad :: lift \Rightarrow lift\ ((da -)\ [88]\ 87)$

$-next-d \quad :: lift \Rightarrow lift\ ((\bigcirc -)\ [88]\ 87)$

$-prev-d \quad :: lift \Rightarrow lift\ ((prev -)\ [88]\ 87)$

syntax (ASCII)

$-schop-d \quad :: [lift, lift] \Rightarrow lift\ ((- \curvearrowright -)\ [84,84]\ 83)$

$-sometimes-d \quad :: lift \Rightarrow lift\ ((\langle \rangle -)\ [88]\ 87)$

$-di-d \quad :: lift \Rightarrow lift\ ((di -)\ [88]\ 87)$

$-da-d \quad :: lift \Rightarrow lift\ ((da -)\ [88]\ 87)$

$-next-d \quad :: lift \Rightarrow lift\ ((next -)\ [88]\ 87)$

$-prev-d \quad :: lift \Rightarrow lift\ ((prev -)\ [88]\ 87)$

translations

$-schop-d \quad \Rightarrow \text{CONST } schop-d$
 $-sometimes-d \Rightarrow \text{CONST } sometimes-d$
 $-di-d \quad \Rightarrow \text{CONST } di-d$
 $-da-d \quad \Rightarrow \text{CONST } da-d$
 $-next-d \quad \Rightarrow \text{CONST } next-d$
 $-prev-d \quad \Rightarrow \text{CONST } prev-d$

definition $df-d :: ('a::world) \text{ formula} \Rightarrow 'a \text{ formula}$
where $df-d F \equiv \text{LIFT}(F \frown \# \text{True})$

definition $sda-d :: ('a::world) \text{ formula} \Rightarrow 'a \text{ formula}$
where $sda-d F \equiv \text{LIFT}(\# \text{True} \frown (F \frown \# \text{True}))$

definition $always-d :: ('a::world) \text{ formula} \Rightarrow 'a \text{ formula}$
where $always-d F \equiv \text{LIFT}(\neg(\Diamond(\neg F)))$

definition $bi-d :: ('a::world) \text{ formula} \Rightarrow 'a \text{ formula}$
where $bi-d F \equiv \text{LIFT}(\neg(di(\neg F)))$

definition $ba-d :: ('a::world) \text{ formula} \Rightarrow 'a \text{ formula}$
where $ba-d F \equiv \text{LIFT}(\neg(da(\neg F)))$

definition $wnext-d :: ('a::world) \text{ formula} \Rightarrow 'a \text{ formula}$
where $wnext-d F \equiv \text{LIFT}(\neg(\bigcirc(\neg F)))$

definition $wprev-d :: ('a::world) \text{ formula} \Rightarrow 'a \text{ formula}$
where $wprev-d F \equiv \text{LIFT}(\neg(prev(\neg F)))$

definition $more-d :: ('a::world) \text{ formula}$
where $more-d \equiv \text{LIFT}(\bigcirc(\# \text{True}))$

syntax

$-df-d \quad :: \text{lift} \Rightarrow \text{lift} ((df \text{ -}) [88] \ 87)$
 $-sda-d \quad :: \text{lift} \Rightarrow \text{lift} ((sda \text{ -}) [88] \ 87)$
 $-always-d \quad :: \text{lift} \Rightarrow \text{lift} ((\Box \text{ -}) [88] \ 87)$
 $-bi-d \quad :: \text{lift} \Rightarrow \text{lift} ((bi \text{ -}) [88] \ 87)$
 $-ba-d \quad :: \text{lift} \Rightarrow \text{lift} ((ba \text{ -}) [88] \ 87)$
 $-wnext-d \quad :: \text{lift} \Rightarrow \text{lift} ((wnext \text{ -}) [88] \ 87)$
 $-wprev-d \quad :: \text{lift} \Rightarrow \text{lift} ((wprev \text{ -}) [88] \ 87)$
 $-more-d \quad :: \text{lift} \quad ((more))$

syntax (ASCII)

$-df-d \quad :: \text{lift} \Rightarrow \text{lift} ((df \text{ -}) [88] \ 87)$
 $-sda-d \quad :: \text{lift} \Rightarrow \text{lift} ((sda \text{ -}) [88] \ 87)$
 $-always-d \quad :: \text{lift} \Rightarrow \text{lift} ((\Box \text{ -}) [88] \ 87)$

$-bi-d \quad :: lift \Rightarrow lift ((bi -) [88] 87)$
 $-ba-d \quad :: lift \Rightarrow lift ((ba -) [88] 87)$
 $-wnext-d \quad :: lift \Rightarrow lift ((wnext -) [88] 87)$
 $-wprev-d \quad :: lift \Rightarrow lift ((wprev -) [88] 87)$
 $-more-d \quad :: lift \quad ((more))$

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

$-bf-d \quad \Rightarrow CONST bf-d$
 $-sba-d \quad \Rightarrow CONST sba-d$

$\text{-empty-d} \Rightarrow \text{CONST empty-d}$
 $\text{-fmore-d} \Rightarrow \text{CONST fmore-d}$
 $\text{-dm-d} \Rightarrow \text{CONST dm-d}$

definition $\text{bm-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{bm-d } F \equiv \text{LIFT}(\neg(\text{dm}(\neg F)))$

definition $\text{init-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{init-d } F \equiv \text{LIFT}((\text{empty} \wedge F); \# \text{True})$

definition $\text{fin-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{fin-d } F \equiv \text{LIFT}(\Box(\text{empty} \longrightarrow F))$

definition $\text{halt-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{halt-d } F \equiv \text{LIFT}(\Box(\text{empty} = F))$

definition $\text{initonly-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{initonly-d } F \equiv \text{LIFT}(\text{bi}(\text{empty} = F))$

definition $\text{keep-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{keep-d } F \equiv \text{LIFT}(\text{ba}(\text{skip} \longrightarrow F))$

definition $\text{yields-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{yields-d } F1 \ F2 \equiv \text{LIFT}(\neg(F1; (\neg F2)))$

definition $\text{syields-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{syields-d } F1 \ F2 \equiv \text{LIFT}(\neg(F1 \frown (\neg F2)))$

definition $\text{ifthenelse-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{ifthenelse-d } F \ G \ H \equiv \text{LIFT}((F \wedge G) \vee (\neg F \wedge H))$

syntax

$\text{-bm-d} \quad \quad \quad :: \text{lift} \Rightarrow \text{lift} \quad \quad \quad ((\text{bm } -) [88] \ 87)$
 $\text{-init-d} \quad \quad \quad :: \text{lift} \Rightarrow \text{lift} \quad \quad \quad ((\text{init } -) [88] \ 87)$
 $\text{-fin-d} \quad \quad \quad :: \text{lift} \Rightarrow \text{lift} \quad \quad \quad ((\text{fin } -) [88] \ 87)$
 $\text{-halt-d} \quad \quad \quad :: \text{lift} \Rightarrow \text{lift} \quad \quad \quad ((\text{halt } -) [88] \ 87)$
 $\text{-initonly-d} \quad :: \text{lift} \Rightarrow \text{lift} \quad \quad \quad ((\text{initonly } -) [88] \ 87)$
 $\text{-keep-d} \quad \quad \quad :: \text{lift} \Rightarrow \text{lift} \quad \quad \quad ((\text{keep } -) [88] \ 87)$
 $\text{-yields-d} \quad \quad :: [\text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((- \ \text{yields } -) [88, 88] \ 87)$
 $\text{-syields-d} \quad \quad :: [\text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((- \ \text{syields } -) [88, 88] \ 87)$
 $\text{-ifthenelse-d} :: [\text{lift}, \text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((\text{if}_i \text{ - then - else - }) [88, 88, 88] \ 87)$

syntax (ASCII)

$\text{-bm-d} \quad \quad \quad :: \text{lift} \Rightarrow \text{lift} \quad \quad \quad ((\text{bm } -) [88] \ 87)$
 $\text{-init-d} \quad \quad \quad :: \text{lift} \Rightarrow \text{lift} \quad \quad \quad ((\text{init } -) [88] \ 87)$
 $\text{-fin-d} \quad \quad \quad :: \text{lift} \Rightarrow \text{lift} \quad \quad \quad ((\text{fin } -) [88] \ 87)$
 $\text{-halt-d} \quad \quad \quad :: \text{lift} \Rightarrow \text{lift} \quad \quad \quad ((\text{halt } -) [88] \ 87)$

$-initonly-d :: lift \Rightarrow lift \quad ((initonly -) [88] 87)$
 $-keep-d :: lift \Rightarrow lift \quad ((keep -) [88] 87)$
 $-yields-d :: [lift, lift] \Rightarrow lift \quad ((- yields -) [88, 88] 87)$
 $-syields-d :: [lift, lift] \Rightarrow lift \quad ((- syields -) [88, 88] 87)$
 $-ifthenelse-d :: [lift, lift, lift] \Rightarrow lift \quad ((if_i - then - else -) [88, 88, 88] 87)$

translations

$-bm-d \quad \Rightarrow \text{CONST } bm-d$
 $-init-d \quad \Rightarrow \text{CONST } init-d$
 $-fin-d \quad \Rightarrow \text{CONST } fin-d$
 $-halt-d \quad \Rightarrow \text{CONST } halt-d$
 $-initonly-d \quad \Rightarrow \text{CONST } initonly-d$
 $-keep-d \quad \Rightarrow \text{CONST } keep-d$
 $-yields-d \quad \Rightarrow \text{CONST } yields-d$
 $-syields-d \quad \Rightarrow \text{CONST } syields-d$
 $-ifthenelse-d \Rightarrow \text{CONST } ifthenelse-d$

definition $sfin-d :: ('a::world) formula \Rightarrow 'a formula$
where $sfin-d F \equiv LIFT(\neg (fin (\neg F)))$

definition $ifthen-d :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a formula$
where $ifthen-d F G \equiv LIFT(if_i F then G else \#True)$

syntax

$-ifthen-d :: [lift, lift] \Rightarrow lift \quad ((if_i - then -) [88, 88] 87)$
 $-sfin-d \quad :: lift \Rightarrow lift \quad ((sfin -) [88] 87)$

syntax (ASCII)

$-ifthen-d :: [lift, lift] \Rightarrow lift \quad ((if_i - then -) [88, 88] 87)$
 $-sfin-d \quad :: lift \Rightarrow lift \quad ((sfin -) [88] 87)$

translations

$-ifthen-d \Rightarrow \text{CONST } ifthen-d$
 $-sfin-d \Rightarrow \text{CONST } sfin-d$

definition $next-assign-d :: ('a::world, 'b) stfun \Rightarrow ('a, 'b) formfun \Rightarrow 'a formula$
where $next-assign-d v e \equiv LIFT(v\$ = e)$

definition $prev-assign-d :: ('a::world, 'b) stfun \Rightarrow ('a, 'b) formfun \Rightarrow 'a formula$
where $prev-assign-d v e \equiv LIFT(finite \longrightarrow v! = e)$

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 $temporal-assign-d v e \equiv \lambda s. s \models finite \longrightarrow !v = e$

definition *gets-d* :: ('a::world,'b) stfun \Rightarrow ('a,'b) formfun \Rightarrow 'a formula
where *gets-d* v e $\equiv \lambda s. s \models \text{keep}(\text{temporal-assign-d } v \text{ } e)$

definition *stable-d* :: ('a::world,'b) stfun \Rightarrow 'a formula
where *stable-d* v $\equiv \lambda s. s \models \text{gets-d } v \text{ } (\text{current-val-d } v)$

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 $\equiv \lambda s. s \models (\text{temporal-assign-d } v \text{ } e) \wedge (\text{padded-d } v)$

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 ((- $<\sim$ -) [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 schop-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

4.4 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* :

$$(w \models F1 ; F2) =$$

$$\begin{aligned} & (\\ & \quad (\exists n \leq \text{nlength } w. ((\text{ntaken } n \ w) \models F1) \wedge ((\text{ndropn } n \ w) \models F2)) \\ & \quad \vee (\neg \text{nfinite } w \wedge (w \models F1)) \\ &) \end{aligned}$$

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 *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* :

$$(w \models F1 \frown F2) =$$

$$\begin{aligned} & (\\ & \quad (\exists n \leq \text{nlength } w. ((\text{ntaken } n \ w) \models F1) \wedge ((\text{ndropn } n \ w) \models F2)) \\ &) \end{aligned}$$

by (*auto simp add: itl-def chop-defs finite-defs*)

lemma *syields-defs* :

$$(w \models F1 \text{ syields } F2) =$$

$$(\forall n. ((\text{ntaken } n \ w) \models F1) \longrightarrow \text{enat } n \leq \text{nlength } w \longrightarrow ((\text{ndropn } n \ w \models F2)))$$

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 chop-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* :

($w \models \text{da } F$) =
($(\exists n \text{ na}. (n + \text{na} \leq \text{nlength } w \wedge ((\text{nsubn } w \ n \ (n + \text{na})) \models F)) \vee (\neg \text{nfinite } w \wedge ((\text{ndropn } n \ w) \models F)))$)

proof

(*auto simp add: itl-def chop-defs nsubn-def1*)

show $\bigwedge n \text{ na}.$

$\text{enat } n \leq \text{nlength } w \implies$
 $\text{enat } \text{na} \leq \text{nlength } w - \text{enat } n \implies$
 $F (\text{ntaken } \text{na} (\text{ndropn } n \ w)) \implies \exists n. (\exists \text{na}. \text{enat } (n + \text{na}) \leq \text{nlength } w \wedge F (\text{ntaken } \text{na} (\text{ndropn } n \ w)))$
 $\vee \neg \text{nfinite } w \wedge F (\text{ndropn } n \ w)$

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** $\text{na} :: \text{nat}$

assume *a1*: $\text{enat } (n + \text{na}) \leq \text{nlength } w$

assume *a2*: $F (\text{ntaken } \text{na} (\text{ndropn } n \ w))$

have $\text{enat } n \leq \text{nlength } w$

using *a1* **by** (*meson enat-ord-simps(1) le-iff-add order-subst2*)

then show $\exists n. \text{enat } n \leq \text{nlength } w \wedge$

$((\exists \text{na}. \text{enat } \text{na} \leq \text{nlength } w - \text{enat } n \wedge F (\text{ntaken } \text{na} (\text{ndropn } n \ w)))$
 $\vee \neg \text{nfinite } w \wedge F (\text{ndropn } n \ w))$

using *a2 a1* **by** (*metis (no-types) add-diff-cancel-left' enat-minus-mono1 idiff-enat-enat*)

next

show $\bigwedge n. \neg \text{nfinite } w \implies$

$F (\text{ndropn } n \ w) \implies$

$\exists n. \text{enat } n \leq \text{nlength } w \wedge$

$((\exists na. \text{ enat } na \leq \text{ nlength } w - \text{ enat } n \wedge F (\text{ ntaken } na (\text{ ndropn } n w))) \vee F (\text{ ndropn } n w))$
by (*meson enat-ile le-cases nfinite-conv-nlength-enat*)
qed

lemma *ba-defs* :

$(w \models \text{ ba } F) =$
 $(\forall n na. (\text{ enat } (n + na) \leq \text{ nlength } w \longrightarrow ((\text{ nsubn } w n (n+na)) \models F))$
 $\wedge (\text{ nfinite } w \vee ((\text{ ndropn } n w) \models F)))$

by (*simp add: ba-d-def da-defs*)

lemma *sda-defs* :

$(w \models \text{ sda } F) =$
 $(\exists n na. (n+na \leq \text{ nlength } w \wedge ((\text{ nsubn } w n (n+ na)) \models F)))$

proof

(*auto simp add: sda-d-def schop-defs nsubn-def1*)

show $\bigwedge n na.$

$\text{ enat } n \leq \text{ nlength } w \implies$
 $\text{ enat } na \leq \text{ nlength } w - \text{ enat } n \implies F (\text{ ntaken } na (\text{ ndropn } n w)) \implies$
 $\exists n na. \text{ enat } (n + na) \leq \text{ nlength } w \wedge F (\text{ ntaken } na (\text{ ndropn } n w))$
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*: $F (\text{ ntaken } na (\text{ ndropn } n w))$

assume *a2*: $\text{ enat } (n + na) \leq \text{ nlength } w$

have $\text{ enat } na \leq \text{ nlength } w - \text{ enat } n$

by (*metis a2 add-diff-cancel-left' enat-minus-mono1 idiff-enat-enat*)

then show $\exists n. \text{ enat } n \leq \text{ nlength } w \wedge$

$(\exists na. \text{ enat } na \leq \text{ nlength } w - \text{ enat } n \wedge F (\text{ ntaken } na (\text{ ndropn } n w)))$

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* :

$(w \models \text{ sba } F) =$
 $(\forall n na. n+na \leq \text{ nlength } w \longrightarrow ((\text{ nsubn } w n (n+na)) \models F))$

by (*simp add: sba-d-def sda-defs*)

lemma *next-defs* :

$(w \models \circ F) =$
 $(\text{ nlength } w \neq (\text{ enat } 0) \wedge ((\text{ ndropn } 1 w) \models F))$

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* :

$(w \models \text{ wnext } F) =$
 $(\text{ nlength } w = (\text{ enat } 0) \vee ((\text{ ndropn } 1 w) \models F))$

by (*simp add: wnext-d-def next-defs*)

lemma *prev-defs* :

($w \models \text{prev } F$) =
 (($\text{nlength } w \neq (\text{enat } 0) \wedge \text{nfinite } w \wedge$
 ($(\text{ntaken } (\text{the-enat}(\text{epred}(\text{nlength } w))) \text{ } w) \models F$) $\vee (\neg \text{nfinite } w \wedge (w \models F))$))

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* :

($w \models \text{wprev } F$) =
 (($\text{nfinite } w \longrightarrow (\text{nlength } w = (\text{enat } 0) \vee ((\text{ntaken } (\text{the-enat}(\text{epred}(\text{nlength } w))) \text{ } w) \models F))$)
 $\wedge (\text{nfinite } w \vee (w \models F))$)

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* :

($w \models \text{init } F$) = ($(\text{ntaken } 0 \text{ } w) \models F$)

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* :

($w \models \text{bi}(\text{empty} \longrightarrow F)$) = ($(\text{ntaken } 0 \text{ } w) \models F$)

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* :

$(w \models \text{fin } F) =$
 $((\neg \text{finite } w \wedge ((\text{ndropn } (\text{the-enat}(\text{nlength } w)) \ w) \models F)) \vee (\neg \neg \text{finite } w))$

by (*simp add: fin-d-def empty-defs always-defs*)

$(\text{metis add.right-neutral enat.distinct}(2) \text{ enat-add-sub-same le-iff-add } \neg \text{finite-nlength-enat}$
 $\text{nlength-eq-enat-nfiniteD the-enat.simps})$

lemma *finalt-defs* :

$(w \models \# \text{True}; (F \wedge \text{empty})) =$
 $((\neg \text{finite } w \wedge ((\text{ndropn } (\text{the-enat}(\text{nlength } w)) \ w) \models F)) \vee (\neg \neg \text{finite } w))$

by (*simp add: chop-defs empty-defs*)

$(\text{metis add.right-neutral enat.distinct}(2) \text{ enat-add-sub-same le-iff-add } \neg \text{finite-nlength-enat}$
 $\text{the-enat.simps})$

lemma *sfin-defs* :

$(w \models \text{sfin } F) = (\neg \text{finite } w \wedge ((\text{ndropn } (\text{the-enat}(\text{nlength } w)) \ w) \models F))$

by (*auto simp add: sfin-d-def fin-defs*)

lemma *halt-defs* :

$(w \models \text{halt}(F)) = (\forall n. n \leq \text{nlength } w \longrightarrow (\text{nlength } w = n) = ((\text{ndropn } n \ w) \models F))$

by (*simp add: halt-d-def empty-defs always-defs*)

$(\text{metis add.right-neutral dual-order.strict-iff-order enat-add-sub-same enat-ord-code}(4)$
 $\text{le-iff-add})$

lemma *initonly-defs* :

$(w \models \text{initonly}(F)) =$
 $((\forall n. n \leq \text{nlength } w \longrightarrow (n = 0) = ((\text{ntaken } n \ w) \models F)) \wedge$
 $(\neg \text{finite } w \vee (\text{nlength } w = 0) = F \ w)$
 $)$

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* :

$(s \models !v) =$

$((if\ nfinite\ s\ then\ (v\ (nlast\ s))\ else\ (\epsilon\ x.\ x=x))$
 $)$
by (*simp add: itl-def*)

lemma *penultval-defs* :

$(s \models v!) =$
 $(if\ nfinite\ s\ then$
 $(if\ nlength\ s \neq (enat\ 0)\ then\ (v\ (nnth\ s\ (the-enat(epred(nlength\ s))))))\ else\ (\epsilon\ x.\ x=x))$
 $else\ (\epsilon\ x.\ x=x)$
 $)$
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: $\bigwedge i. i < \text{nlength } s \longrightarrow v (\text{nfirst } (\text{nsubn } s \ i \ (\text{Suc } i))) = v (\text{nnth } s \ i)$
by (*simp add: nsubn-def1 ntaken-ndropn-nfirst*)
have 2: $(\forall i. \text{enat } i < \text{nlength } s \longrightarrow v (\text{nnth } s \ (\text{Suc } i)) = v (\text{nfirst } (\text{nsubn } s \ i \ (\text{Suc } i)))) =$
 $(\forall i. \text{enat } i < \text{nlength } s \longrightarrow v (\text{nnth } s \ (\text{Suc } i)) = v (\text{nnth } s \ i))$
by (*simp add: 1*)
have 3: $(\forall i. \text{enat } i < \text{nlength } s \longrightarrow v (\text{nnth } s \ (\text{Suc } i)) = v (\text{nnth } s \ i)) =$
 $(\forall i. i \leq \text{nlength } s \longrightarrow (v (\text{nnth } s \ i)) = (v (\text{nfirst } s)))$
using *stable-defs-help[of s v]* **by** *blast*
show $(\forall i. \text{enat } i < \text{nlength } s \longrightarrow v (\text{nnth } s \ (\text{Suc } i)) = v (\text{nfirst } (\text{nsubn } s \ i \ (\text{Suc } i)))) =$
 $(\forall i. \text{enat } i \leq \text{nlength } s \longrightarrow v (\text{nnth } s \ i) = v (\text{nfirst } s))$
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)
 $(\text{metis One-nat-def enat.simps}(3) \text{ enat-add-sub-same enat-ord.simps}(1) \text{ iless-Suc-eq le-iff-add}$
 $\text{less-imp-le one-enat-def plus-1-eSuc}(2),$
 $\text{meson iless-Suc-eq less-imp-le},$
 $\text{metis One-nat-def enat.simps}(3) \text{ enat-add-sub-same enat-ord.simps}(1) \text{ ile-eSuc nfinite-nlength-enat}$
 $\text{one-enat-def plus-1-eSuc}(2),$
 $\text{metis One-nat-def antisym-conv2 co.enat.sel}(2) \text{ diff-le-self enat-add-sub-same enat-ord-code}(4)$
 $\text{enat-ord.simps}(1) \text{ epred-enat iless-Suc-eq one-enat-def plus-1-eSuc}(2))$

qed

lemma *padded-temporal-assign-defs* :

$(s \models v < \sim e) =$
 $((s \models \text{padded } v) \wedge$
 $(\text{if } \text{nfinite } s \text{ then } (v (\text{nlast } s)) = e \text{ s else True}))$
 $)$

by (*auto simp add: padded-temp-assign-d-def padded-defs temporal-assign-defs*)

lemma *chop-nfuse-1* :

$(\exists \sigma 1 \sigma 2. \sigma = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge$
 $(\sigma 1 \models f) \wedge (\sigma 2 \models g) \wedge$
 $(\text{nlast } \sigma 1 = \text{nfirst } \sigma 2)) =$
 $((\exists i. 0 \leq i \wedge i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \ \sigma \models f) \wedge (\text{ndropn } i \ \sigma \models g)))$

by *auto*

$(\text{metis enat-le-plus-same}(1) \text{ nfuse-nlength ndropn-nfuse nfinite-conv-nlength-enat ntaken-nfuse}$
 $\text{the-enat.simps},$

metis nfuse-ntaken-ndropn ndropn-nfirst nfinite-ntaken ntaken-nlast)

lemma *chop-nfuse-2 :*

$(\exists \sigma 1 \sigma 2. \sigma = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge$
 $(\sigma 1 \in X) \wedge (\sigma 2 \in Y) \wedge$
 $(\text{nlast } \sigma 1 = \text{nfirst } \sigma 2)) =$
 $(\exists i. i \leq \text{nlength } \sigma \wedge (\text{ntaken } i \sigma) \in X \wedge (\text{ndropn } i \sigma) \in Y)$

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 schop-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-equiv-defs temporal-assign-defs gets-defs stable-defs padded-defs
padded-temporal-assign-defs

4.5 Soundness Axioms

4.5.1 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 $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge (\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge$
 $f (\text{ntaken } (\text{min } na n) \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \sigma))) \wedge h (\text{ndropn } n \sigma)) \vee$
 $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ntaken } n \sigma) \wedge g (\text{ndropn } n \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \sigma))$

proof *—*

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*


```

      h (ndropn na (ndropn n σ))
    using 9 by auto
  have 11: h (ndropn (na+n) σ)
    by (metis 10 add commute ndropn-ndropn)
  have 12: na+n ≤ nlength σ
    by (metis 10 6 Groups.add-ac(2) dual-order.strict-implies-order enat.simps(3)
      enat-add-sub-same enat-less-enat-plusI2 le-iff-add not-le-imp-less
      order.not-eq-order-implies-strict plus-enat-simps(1))
  have 13: g (ndropn n (ntaken (n+na) σ))
    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 ∧

```

$(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \ \sigma)))$
 $\wedge h (\text{ndropn } n \ \sigma)) \implies$
 $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $f (\text{ntaken } n \ \sigma) \wedge$
 $((\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \ \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n \ \sigma))))$
 \vee
 $\neg \text{nfinite } (\text{ndropn } n \ \sigma) \wedge g (\text{ndropn } n \ \sigma))) \vee$
 $\neg \text{nfinite } \sigma \wedge f \ \sigma)$
proof –
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: $na \leq n \wedge \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \ \sigma))$
 $n \ \sigma))$
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*:
 $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$

$f \text{ (ntaken } n \text{ } \sigma) \wedge$
 $((\exists na. \text{ enat } na \leq \text{ nlength } \sigma - \text{ enat } n \wedge g \text{ (ntaken } na \text{ (ndropn } n \text{ } \sigma))} \wedge h \text{ (ndropn } na \text{ (ndropn } n \text{ } \sigma))))$
 \vee
 $\neg \text{ nfinite (ndropn } n \text{ } \sigma) \wedge g \text{ (ndropn } n \text{ } \sigma))) \vee$
 $\neg \text{ nfinite } \sigma \wedge f \text{ } \sigma) =$
 $((\exists n. \text{ enat } n \leq \text{ nlength } \sigma \wedge$
 $(\exists na \leq n. \text{ enat } na \leq \text{ nlength } \sigma \wedge f \text{ (ntaken (min } na \text{ } n) \text{ } \sigma) \wedge g \text{ (ndropn } na \text{ (ntaken } n \text{ } \sigma))}$
 $\wedge h \text{ (ndropn } n \text{ } \sigma)) \vee$
 $\neg \text{ nfinite } \sigma \wedge ((\exists n. \text{ enat } n \leq \text{ nlength } \sigma \wedge f \text{ (ntaken } n \text{ } \sigma) \wedge g \text{ (ndropn } n \text{ } \sigma)) \vee \neg \text{ nfinite } \sigma \wedge f \text{ } \sigma))$
using *ChopAssocSemHelpa*[of $\sigma \text{ } f \text{ } g \text{ } h$] *ChopAssocSemHelpb*[of $\sigma \text{ } f \text{ } g \text{ } h$] **by** *blast*

lemma *ChopAssocSemHelp1*:

$((\sigma \models f ; (g ; h)) = ((\sigma \models (f;g);h))$

proof –

have $(\sigma \models f ; (g ; h)) = ((\exists n. \text{ enat } n \leq \text{ nlength } \sigma \wedge$
 $f \text{ (ntaken } n \text{ } \sigma) \wedge$
 $((\exists na. \text{ enat } na \leq \text{ nlength } \sigma - \text{ enat } n \wedge g \text{ (ntaken } na \text{ (ndropn } n \text{ } \sigma))} \wedge h \text{ (ndropn } na \text{ (ndropn } n \text{ } \sigma))))$
 \vee
 $\neg \text{ nfinite (ndropn } n \text{ } \sigma) \wedge g \text{ (ndropn } n \text{ } \sigma))) \vee$
 $\neg \text{ nfinite } \sigma \wedge f \text{ } \sigma)$

by (*simp add: chop-defs*)
also have ... =
 $((\exists n. \text{ enat } n \leq \text{ nlength } \sigma \wedge$
 $(\exists na \leq n. \text{ enat } na \leq \text{ nlength } \sigma \wedge f \text{ (ntaken (min } na \text{ } n) \text{ } \sigma) \wedge g \text{ (ndropn } na \text{ (ntaken } n \text{ } \sigma))}$
 $\wedge h \text{ (ndropn } n \text{ } \sigma)) \vee$
 $\neg \text{ nfinite } \sigma \wedge ((\exists n. \text{ enat } n \leq \text{ nlength } \sigma \wedge f \text{ (ntaken } n \text{ } \sigma) \wedge g \text{ (ndropn } n \text{ } \sigma)) \vee \neg \text{ nfinite } \sigma \wedge f \text{ } \sigma))$
using *ChopAssocSemHelp*[of $\sigma \text{ } f \text{ } g \text{ } h$] **by** *blast*

also have ... =

$(\sigma \models (f;g);h) \text{ by } (\text{simp add: chop-defs})$
finally show $(\sigma \models f ; (g ; h)) = (\sigma \models (f;g);h) \text{ .}$
qed

lemma *ChopAssocSem*:

$(\sigma \models f ; (g ; h) = (f;g);h)$

using *ChopAssocSemHelp1*[of $f \text{ } g \text{ } h \text{ } \sigma$] **by** *auto*

4.5.2 OrChopImp

lemma *OrChopImpSem*:

$(\sigma \models (f \vee g);h \longrightarrow f;h \vee g;h)$

by (*auto simp add: chop-defs*)

4.5.3 ChopOrImp

lemma *ChopOrImpSem*:

$(\sigma \models f;(g \vee h) \longrightarrow f;g \vee f;h)$

by (*auto simp add: chop-defs*)

4.5.4 EmptyChop

lemma *EmptyChopSem*:

$(\sigma \models \text{empty} ; f = f)$
by (*simp add: chop-defs empty-defs min-def*)
 $(\text{metis enat-0-iff}(1) \text{ ndropn-0 nlength-eq-enat-nfiniteD zero-le})$

4.5.5 ChopEmpty

lemma *ChopEmptySem*:
 $(\sigma \models f; \text{empty} = f)$
by (*simp add: chop-defs empty-defs min-def*)
 $(\text{metis cancel-comm-monoid-add-class.diff-cancel enat-diff-cancel-left idiff-enat-enat nfinite-nlength-enat ntaken-all order-refl zero-enat-def})$

4.5.6 StateImpBi

lemma *StateImpBiSem*:
 $(\sigma \models \text{init } f \longrightarrow \text{bi } (\text{init } f))$
by (*simp add: init-defs bi-defs*)

4.5.7 NextImpNotNextNot

lemma *NextImpNotNextNotSem*:
 $(\sigma \models \bigcirc f \longrightarrow \neg (\bigcirc (\neg f)))$
by (*simp add: next-defs*)

4.5.8 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

4.5.9 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  $\bigwedge n. \forall n. \text{ enat } n \leq \text{nlength } \sigma \longrightarrow f (\text{ndropn } n \sigma) \longrightarrow$ 
     $\text{nlength } \sigma - \text{ enat } n = \text{ enat } 0 \vee f (\text{ndropn } (\text{Suc } 0) (\text{ndropn } n \sigma)) \implies$ 
     $f \sigma \implies$ 
     $\text{ enat } n \leq \text{nlength } \sigma \implies$ 
     $f (\text{ndropn } n \sigma)$ 
  using box-induct-help-1 by blast
qed

```

4.6 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

```

4.7 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* ($(\exists \exists \exists \text{ -./ -}) [0,10] 10$)

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* $\equiv \text{LIFT}(\neg(\exists \exists x. \neg(F x)))$

syntax

-Aall :: [*idts*, *lift*] \Rightarrow *lift* ($(\exists \forall \forall \text{ -./ -}) [0,10] 10$)

translations

-Aall v A == *Aall v. A*

end

5 Finite and Infinite ITL: Axioms and Rules

```
theory ITL
imports
  Semantics
begin
```

The Finite and Infinite ITL axiom and proof rules are introduced (taken from [9]). The soundness of the rules and axioms are checked using the lemmas of Semantics.thy.

5.1 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)
```

5.2 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 :
   $\vdash f; \text{empty} = f$ 
using ChopEmptySem Valid-def by blast

lemma StateImpBi :
   $\vdash \text{init } f \longrightarrow \text{bi } (\text{init } f)$ 
using StateImpBiSem Valid-def by blast

lemma NextImpNotNextNot :
   $\vdash \bigcirc f \longrightarrow \neg (\bigcirc (\neg f))$ 
using NextImpNotNextNotSem Valid-def by blast

lemma BiBoxChopImpChop :
   $\vdash \text{bi } (f \longrightarrow f1) \wedge \Box (g \longrightarrow g1) \longrightarrow f; g \longrightarrow f1; g1$ 
using BiBoxChopImpChopSem Valid-def by blast

lemma BoxInduct :
   $\vdash \Box (f \longrightarrow \text{wnext } f) \wedge f \longrightarrow \Box f$ 
using BoxInductSem Valid-def by blast

```

5.3 Additional Lemmas

The following is again from [4, 2] but adapted for our need.

```

lemma int-eq-true:
assumes  $\vdash P$ 
shows  $\vdash P = \# \text{True}$ 
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*

5.4 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*:
 $((w) \models (\exists \exists x. F x)) =$
 $(\exists x (val :: 'a\ nelist). (\ (val = (nmap\ x\ w) \wedge ((w) \models F x))))$
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*

5.5 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*]

5.6 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*]

5.7 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]*

5.8 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]*

5.9 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-eqv-next*:
 $\vdash \text{skip} \longrightarrow !v = v\$$
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*:
 $\vdash \text{skip} \longrightarrow v! = \v
by (*simp add: Valid-def itl-defs current-val-d-def nlength-eq-enat-nfiniteD*)
 (*metis ndropn-0 ndropn-nfirst*)

end

6 Finite and Infinite ITL theorems using Weak Chop

theory *Theorems*
imports
ITL
begin

We give the proofs of a list of Finite and Infinite ITL theorems. These proofs and theorems are from [8] but adapted for infinite and finite intervals.

6.1 Propositional reasoning

This is a list of propositional logic theorems used in the proofs of the ITL theorems.

lemma *IfThenElseImp*:
 $\vdash (\text{if}_i \ g \ \text{then} \ f \ \text{else} \ f1) \longrightarrow ((g \longrightarrow f) \wedge (\neg g \longrightarrow f1))$
by (*simp add: itl-def Valid-def*)

lemma *Prop01*:
assumes $\vdash f \longrightarrow \neg g \vee h$
shows $\vdash g \wedge f \longrightarrow h$
using *assms* **by** *auto*

lemma *Prop02*:
assumes $\vdash f \longrightarrow g$
 $\vdash f1 \longrightarrow g$
shows $\vdash f \vee f1 \longrightarrow g$
using *assms* **by** *fastforce*

lemma *Prop03*:
assumes $\vdash f = (g \vee h)$

shows $\vdash h \longrightarrow f$
using *assms* **by** *auto*

lemma *Prop04*:
assumes $\vdash f = h$
 $\vdash f = h1$
shows $\vdash h1 = h$
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*)

6.2 State formulas

The *init* operator denotes state formulas, i.e., ITL formula that only constrain the first state of an interval.

lemma *Initprop* :

$\vdash ((\text{init } f) \wedge (\text{init } g)) = \text{init}(f \wedge g)$
 $\vdash (\neg (\text{init } f)) = \text{init } (\neg f)$
 $\vdash ((\text{init } f) \vee (\text{init } g)) = \text{init } (f \vee g)$
 $\vdash \text{init } \# \text{True}$
by (*auto simp: itl-defs*)

lemma *Finprop* :

$\vdash ((\# \text{True}; (f \wedge \text{empty})) \wedge (\# \text{True}; (g \wedge \text{empty}))) = (\# \text{True}; ((f \wedge g) \wedge \text{empty}))$
 $\vdash ((\# \text{True}; (f \wedge \text{empty})) \vee (\# \text{True}; (g \wedge \text{empty}))) = (\# \text{True}; ((f \vee g) \wedge \text{empty}))$
 $\vdash (\# \text{True}; ((\# \text{True}) \wedge \text{empty}))$
 $\vdash \text{finite} \longrightarrow (\neg (\# \text{True}; (f \wedge \text{empty}))) = (\# \text{True}; (\neg f \wedge \text{empty}))$
 $\vdash (\neg (\# \text{True}; (f \wedge \text{empty}))) = ((\# \text{True}; (\neg f \wedge \text{empty})) \wedge \text{finite})$
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)

6.3 finite and inf properties

lemma *EmptyImpFinite*:

$\vdash \text{empty} \longrightarrow \text{finite}$
using *nlength-eq-enat-nfiniteD* **by** (*auto simp add: itl-defs zero-enat-def*)

lemma *SkipChopFiniteImpFinite*:

$\vdash \text{skip}; \text{finite} \longrightarrow \text{finite}$
using *nfinite-ndropn nlength-eq-enat-nfiniteD*
by (*simp add: Valid-def itl-defs, force*)

lemma *FiniteChopSkipImpFinite*:

$\vdash \text{finite}; \text{skip} \longrightarrow \text{finite}$
using *nlength-eq-enat-nfiniteD*
by (*simp add: Valid-def itl-defs, force*)

lemma *FiniteChopSkipImpMore*:

$\vdash \text{finite}; \text{skip} \longrightarrow \text{more}$
using *nlength-eq-enat-nfiniteD one-enat-def*
by (*simp add: Valid-def itl-defs, force*)

lemma *FiniteAndMoreImpFiniteChopSkip*:

$\vdash \text{finite} \wedge \text{more} \longrightarrow \text{finite}; \text{skip}$
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:*

$\vdash \text{finite};\text{skip} = (\text{finite} \wedge \text{more})$

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 iless-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*)

6.4 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: $\vdash \text{bi } (f \wedge f1 \longrightarrow f) \longrightarrow (f \wedge f1); g \longrightarrow f; g$ **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:

$\vdash (f \wedge finite);g \longrightarrow \Diamond g$

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*:

$\vdash \bigcirc (\Diamond f) \longrightarrow \Diamond f$

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: $\vdash (\text{skip};\text{finite});f = \bigcirc(\Diamond f)$ **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: $\vdash \bigcirc (\Diamond (\neg f)) \longrightarrow \Diamond (\neg f)$ **by** (*rule NextDiamondImpDiamond*)
have 5: $\vdash \neg \neg (\Diamond (\neg f)) \longrightarrow \Diamond (\neg f)$ **by** *auto*
hence 6: $\vdash \bigcirc (\neg \neg (\Diamond (\neg f))) \longrightarrow \bigcirc (\Diamond (\neg f))$ **by** (*rule NextImpNext*)
have 7: $\vdash \bigcirc (\neg \neg (\Diamond (\neg f))) \longrightarrow \Diamond (\neg f)$ **using** 4 6 **by** *auto*
hence 8: $\vdash \bigcirc (\neg (\Box f)) \longrightarrow \Diamond (\neg f)$ **by** (*simp add: always-d-def*)
hence 9: $\vdash \neg (\Diamond (\neg f)) \longrightarrow \neg (\bigcirc (\neg (\Box f)))$ **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: $\vdash \Box(\neg g \longrightarrow \neg f) \longrightarrow (finite;(\neg g)) \longrightarrow (finite;(\neg f))$ **by** (*rule BoxChopImpChop*)

have 5: $\vdash (finite;(\neg g)) \longrightarrow (finite;(\neg f))$ **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: $\vdash \Box((\neg g) \longrightarrow \neg f) \longrightarrow (finite;(\neg g)) \longrightarrow (finite;(\neg f))$

by (*rule BoxChopImpChop*)

have 4: $\vdash \Box(f \longrightarrow g) \longrightarrow (finite;(\neg g)) \longrightarrow (finite;(\neg f))$

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*:

$\vdash \Diamond empty = finite$

proof –

have 1: $\vdash finite; empty = finite$ **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: $\vdash skip;f = skip;g$ **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 DiamondEmptyEqFinite 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: $\vdash \text{finite} \longrightarrow \neg(f \longrightarrow g)$ 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)
 qed

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: $\vdash \square(\text{empty} = f) = \square(\text{empty} = g)$ **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*)
qed

from 2 3 show ?thesis using MP by fastforce
qed

6.5 Further Properties Di and Bi

lemma *ImpDi*:

$\vdash f \longrightarrow di\ f$

proof –

have 1: $\vdash f; \text{empty} = f$ **by** (rule *ChopEmpty*)

have 2: $\vdash \text{empty} \longrightarrow \#True$ **by** auto

hence 3: $\vdash f; \text{empty} \longrightarrow f; \#True$ **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*:

$\vdash di\ (\text{init}\ w) = (\text{init}\ w)$

proof –

have 0: $\vdash (\text{init}\ (\neg w)) \longrightarrow bi\ (\text{init}\ (\neg w))$ **using** *StateImpBi* **by** fastforce

hence 1: $\vdash \neg(\text{init}\ w) \longrightarrow bi\ (\neg(\text{init}\ w))$ **using** *Initprop*(2) **by** (metis *inteq-reflection*)

hence 2: $\vdash (\neg(\text{init}\ w)) \longrightarrow \neg(di\ (\neg\neg(\text{init}\ w)))$ **by** (simp add: bi-d-def)

have 3: $\vdash (\neg(\text{init}\ w) \longrightarrow \neg(di\ (\neg\neg(\text{init}\ w)))) \longrightarrow$

$(di\ (\neg\neg(\text{init}\ w)) \longrightarrow (\text{init}\ w))$ **by** auto

have 4: $\vdash di\ (\neg\neg(\text{init}\ w)) \longrightarrow (\text{init}\ w)$ **using** 2 3 *MP* **by** blast

have 5: $\vdash (\text{init}\ w) \longrightarrow \neg\neg(\text{init}\ w)$ **by** auto

hence 6: $\vdash di\ (\text{init}\ w) \longrightarrow di\ (\neg\neg(\text{init}\ w))$ **by** (rule *DiImpDi*)

have 7: $\vdash di\ (\text{init}\ w) \longrightarrow (\text{init}\ w)$ **using** 6 4 **using** *lift-imp-trans* **by** metis

have 8: $\vdash (\text{init}\ w) \longrightarrow di\ (\text{init}\ w)$ **by** (rule *ImpDi*)

from 7 8 **show** ?thesis **by** fastforce

qed

lemma *StateChop*:

$\vdash (\text{init}\ w); f \longrightarrow (\text{init}\ w)$

by (metis *ChopAssoc Prop12 RightChopImpChop TrueW init-d-def int-simps*(13) *int-simps*(17) *inteq-reflection*)

lemma *StateChopExportA*:

$\vdash ((\text{init}\ w) \wedge f); g \longrightarrow (\text{init}\ w)$

using *DiState AndChopA StateChop* **by** fastforce

lemma *StateAndChop*:

$\vdash ((\text{init}\ w) \wedge f); g = ((\text{init}\ w) \wedge (f; g))$

by (simp add: *AndChopB StateAndChopImport StateChopExportA Prop11 Prop12*)

lemma *StateAndChopImpChopRule*:

assumes $\vdash (\text{init}\ w) \wedge f \longrightarrow f1$

shows $\vdash (\text{init}\ w) \wedge (f; g) \longrightarrow (f1; g)$

proof –

have 1: $\vdash (\text{init}\ w) \wedge f \longrightarrow f1$ **using** *assms* **by** auto

hence 2: $\vdash ((\text{init}\ w) \wedge f); g \longrightarrow f1; g$ **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 integ-reflection*)

have 3: $\vdash ((\text{empty} \vee \circ(\Box(\text{init } w))); \Box(\text{init } w)) =$
 $(\Box(\text{init } w) \vee \circ(\Box(\text{init } w); \Box(\text{init } w)))$

by (rule *EmptyOrNextChopEqv*)
 have 4: $\vdash (\Box (\text{init } w); \Box (\text{init } w)) =$
 $((\text{init } w) \wedge (\Box (\text{init } w) \vee \bigcirc(\Box (\text{init } w); \Box (\text{init } w))))$
 using 2 3 by fastforce
 have 5: $\vdash \neg (\Box (\text{init } w)) \longrightarrow \neg (\text{init } w) \vee \neg (\bigcirc(\Box (\text{init } w)))$
 by (rule *NotBoxImpNotOrNotNextBox*)
 have 6: $\vdash (\Box (\text{init } w); \Box (\text{init } w)) \wedge \neg (\Box (\text{init } w)) \longrightarrow$
 $\bigcirc(\Box (\text{init } w); \Box (\text{init } w)) \wedge \neg (\bigcirc(\Box (\text{init } w)))$
 using 4 5 by fastforce
 hence 7: $\vdash \Box (\text{init } w); \Box (\text{init } w) \wedge \text{finite} \longrightarrow \Box (\text{init } w)$
 by (rule *NextContra*)
 have 8: $\vdash \Box (\text{init } w); \Box (\text{init } w) \wedge \text{inf} \longrightarrow \Box (\text{init } w)$
 by (rule *BoxStateChopBoxAndInfImpBox*)
 have 9: $\vdash \Box (\text{init } w); \Box (\text{init } w) \wedge (\text{finite} \vee \text{inf}) \longrightarrow \Box (\text{init } w)$
 using 7 8 by fastforce
 hence 10: $\vdash \Box (\text{init } w); \Box (\text{init } w) \longrightarrow \Box (\text{init } w)$
 using *FiniteOrInfinite* by fastforce
 have 11: $\vdash \Box (\text{init } w) = ((\text{init } w) \wedge \Box (\text{init } w))$
 by (rule *BoxEqvAndBox*)
 have 12: $\vdash \text{empty} ; \Box (\text{init } w) = \Box (\text{init } w)$
 by (rule *EmptyChop*)
 have 13: $\vdash ((\text{init } w) \wedge \text{empty}) ; \Box (\text{init } w) = ((\text{init } w) \wedge (\text{empty} ; \Box (\text{init } w)))$
 by (rule *StateAndChop*)
 have 14: $\vdash \Box (\text{init } w) = ((\text{init } w) \wedge \text{empty}) ; \Box (\text{init } w)$
 using 11 12 13 by fastforce
 have 15: $\vdash (\text{init } w) \wedge \text{empty} \longrightarrow \Box (\text{init } w)$
 by (rule *StateAndEmptyImpBoxState*)
 hence 16: $\vdash ((\text{init } w) \wedge \text{empty}) ; \Box (\text{init } w) \longrightarrow \Box (\text{init } w); \Box (\text{init } w)$
 by (rule *LeftChopImpChop*)
 have 17: $\vdash \Box (\text{init } w) \longrightarrow \Box (\text{init } w); \Box (\text{init } w)$
 using 14 16 by fastforce
 from 10 17 show ?thesis by fastforce
 qed

lemma *NotBoxStateImpBoxYieldsNotBox*:

$\vdash \neg (\Box (\text{init } w)) \longrightarrow (\Box (\text{init } w)) \text{ yields } (\neg (\Box (\text{init } w)))$

proof –

have 1: $\vdash \Box (\text{init } w); \Box (\text{init } w) = \Box (\text{init } w)$ by (rule *BoxStateChopBoxEqvBox*)
 have 2: $\vdash \Box (\text{init } w) = (\neg \neg (\Box (\text{init } w)))$ by auto
 hence 3: $\vdash \Box (\text{init } w); \Box (\text{init } w) = \Box (\text{init } w); (\neg \neg (\Box (\text{init } w)))$ by (rule *RightChopEqvChop*)
 have 4: $\vdash \neg (\Box (\text{init } w)) \longrightarrow \neg (\Box (\text{init } w); (\neg \neg (\Box (\text{init } w))))$ using 1 3 by auto
 from 4 show ?thesis by (simp add: *yields-d-def*)
 qed

lemma *StateEqvBi*:

$\vdash (\text{init } w) = \text{bi } (\text{init } w)$

proof –

have 1: $\vdash (\text{init } w) \longrightarrow \text{bi } (\text{init } w)$ by (rule *StateImpBi*)
 have 2: $\vdash \text{bi } (\text{init } w) \longrightarrow (\text{init } w)$ 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*)

6.6 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*:
 $\vdash \Box (init \ w) = ba (\Box (init \ w))$
proof –
have 1: $\vdash (init \ w) = bi \ (init \ w)$ **by** (*rule StateEqvBi*)
hence 2: $\vdash \Box (init \ w) = \Box (bi \ (init \ w))$ **by** (*rule BoxEqvBox*)
have 3: $\vdash \Box (bi \ (init \ w)) = bi (\Box (init \ w))$ **by** (*rule BtBiEqvBiBt*)
have 4: $\vdash \Box (init \ w) = \Box (\Box (init \ w))$ **by** (*rule BoxEqvBoxBox*)
hence 5: $\vdash bi (\Box (init \ w)) = bi (\Box (\Box (init \ w)))$ **by** (*rule BiEqvBi*)
have 6: $\vdash ba (\Box (init \ w)) = bi (\Box (\Box (init \ w)))$ **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*:

$\vdash di(\circ f) = \circ (di\ f)$

proof –

have 1: $\vdash (\circ f); \#True = \circ(f; \#True)$ **by** (rule *NextChop*)
from 1 **show** ?thesis **by** (simp add: di-d-def)
qed

lemma *DiNextState*:

$\vdash di(\circ (init\ w)) = \circ (init\ w)$

proof –

have 1: $\vdash di(\circ (init\ w)) = \circ(di\ (init\ w))$ **by** (rule *DiNext*)
have 2: $\vdash di\ (init\ w) = (init\ w)$ **by** (rule *DiState*)
hence 3: $\vdash \circ(di\ (init\ w)) = \circ (init\ w)$ **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

6.7 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: $\vdash fmore ; (f \wedge finite) = \circ(finite ; (f \wedge finite))$

using *FmoreEqvSkipChopFinite* **by** (*metis NextChop inteq-reflection next-d-def*)

have 2: $\vdash \circ(finite ; (f \wedge finite)) \longrightarrow fmore$

by (*metis ChopAndB FiniteChopFiniteEqvFinite FmoreEqvSkipChopFinite RightChopImpChop inteq-reflection next-d-def*)

have 3: $\vdash (\circ finite ; (f \wedge finite)) \longrightarrow fmore$ **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*:

$\vdash fmore ; f = \circ(\diamond f)$

proof –

have 1: $\vdash fmore ; f = (\circ finite) ; f$

by (*simp add: FmoreEqvSkipChopFinite LeftChopEqvChop next-d-def*)

have 2: $\vdash (\circ finite) ; f = \circ(finite ; f)$

by (*rule NextChop*)

have 3: $\vdash fmore ; f = \circ(finite ; f)$

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: $\vdash fmore ; (\neg f) = \circ(\diamond (\neg f))$ **by** (*rule MoreChopEqvNextDiamond*)

have 2: $\vdash \circ(\diamond (\neg f)) = \circ(\neg(\Box f))$ **by** (*auto simp: always-d-def*)

have 3: $\vdash \circ(\neg(\Box f)) = (\neg (wnext(\Box f)))$ **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 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 *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

6.8 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

6.9 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

7 Finite and Infinite ITL theorems using strong chop

theory *SChopTheorems*

```

imports
  Theorems
begin

```

We give the proofs of a list of Finite and Infinite ITL theorems but now using the strong chop.

7.1 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*)

7.2 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* :

$\vdash ((\# \text{True} \frown (f \wedge \text{empty})) \wedge (\# \text{True} \frown (g \wedge \text{empty}))) = (\# \text{True} \frown ((f \wedge g) \wedge \text{empty}))$

$\vdash ((\# \text{True} \frown (f \wedge \text{empty})) \vee (\# \text{True} \frown (g \wedge \text{empty}))) = (\# \text{True} \frown ((f \vee g) \wedge \text{empty}))$

$\vdash \text{finite} \longrightarrow (\neg (\# \text{True} \frown (f \wedge \text{empty}))) = (\# \text{True} \frown (\neg f \wedge \text{empty}))$

$\vdash (\neg (\# \text{True} \frown (f \wedge \text{empty}))) = ((\# \text{True} \frown (\neg f \wedge \text{empty})) \vee \text{inf})$

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)

7.3 Basic Theorems

lemma *BfSChopImpSChop* :

$\vdash bf (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$

proof –

have 1: $\vdash g \longrightarrow g$ **by** *auto*

hence 2: $\vdash \Box (g \longrightarrow g)$ **by** (*rule BoxGen*)

have 3: $\vdash bf (f \longrightarrow f1) \wedge \Box (g \longrightarrow g) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (*rule BfBoxSChopImpSChop*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *BiImpBf*:

$\vdash bi f \longrightarrow bf f$

unfolding *bi-d-def bf-d-def di-d-def df-d-def schop-d-def*

by (*simp add: AndChopA*)

lemma *BiSChopImpSChop* :

$\vdash bi (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$

proof –

have 1: $\vdash g \longrightarrow g$

by *auto*

hence 2: $\vdash \Box (g \longrightarrow g)$

by (*rule BoxGen*)

have 3: $\vdash bi (f \longrightarrow f1) \wedge \Box (g \longrightarrow g) \longrightarrow f \frown g \longrightarrow f1 \frown g$

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: $\vdash bf (f \wedge f1 \longrightarrow f)$ **by** (*rule BfGen*)

have 3: $\vdash bf (f \wedge f1 \longrightarrow f) \longrightarrow (f \wedge f1) \frown g \longrightarrow f \frown g$ **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: $\vdash bf (f \wedge f1 \longrightarrow f1)$ **by** (*rule BfGen*)

have 3: $\vdash bf (f \wedge f1 \longrightarrow f1) \longrightarrow (f \wedge f1) \frown g \longrightarrow f1 \frown g$ **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: $\vdash skip \frown (f \frown g) = (skip \frown f) \frown g$ **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: $\vdash bf (g \longrightarrow g)$ **by** (*rule BfGen*)

have 3: $\vdash bf (f \longrightarrow f) \wedge \Box (g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **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: $\vdash bf (f \longrightarrow f1)$ **by** (*rule BfGen*)

have 3: $\vdash bf (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **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: $\vdash (g = g1) = (\vdash (g \longrightarrow g1) \wedge (\vdash g1 \longrightarrow g))$
by *fastforce*
have 1: $\vdash \Box (g = g1) = (\Box (g \longrightarrow g1) \wedge \Box (\vdash g1 \longrightarrow g))$
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: $\vdash \Box (\vdash g1 \longrightarrow g) \longrightarrow (f \frown g1) \longrightarrow (f \frown g)$
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: $\vdash f \frown g \longrightarrow (f \vee f1) \frown g$ **by** (*rule LeftSChopImpSChop*)

have 3: $\vdash f1 \longrightarrow f \vee f1$ **by** *auto*

hence 4: $\vdash f1 \frown g \longrightarrow (f \vee f1) \frown g$ **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 $\vdash f \frown g \longrightarrow (f1 \frown g) \vee (f2 \frown g)$

proof –

have 1: $\vdash f \longrightarrow f1 \vee f2$ **using** *assms* **by** *auto*

hence 2: $\vdash f \frown g \longrightarrow (f1 \vee f2) \frown g$ **by** (*rule LeftSChopImpSChop*)

have 3: $\vdash (f1 \vee f2) \frown g = (f1 \frown g \vee f2 \frown g)$ **by** (*rule OrSChopEqv*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *LeftSChopEqvSChop*:

assumes $\vdash f = f1$

shows $\vdash f \frown g = (f1 \frown g)$

proof –

have 1: $\vdash f = f1$ **using** *assms* **by** *auto*

hence 2: $\vdash f \longrightarrow f1$ **by** *auto*

hence 3: $\vdash f \frown g \longrightarrow f1 \frown g$ **by** (*rule LeftSChopImpSChop*)

have $\vdash f1 \longrightarrow f$ **using** 1 **by** *auto*

hence 4: $\vdash f1 \frown g \longrightarrow f \frown g$ **by** (*rule LeftSChopImpSChop*)

from 3 4 **show** *?thesis* **by** (*simp add: int-iffI*)

qed

lemma *OrSChopEqvRule*:

assumes $\vdash f = (f1 \vee f2)$

shows $\vdash f \frown g = ((f1 \frown g) \vee (f2 \frown g))$

proof –

have 1: $\vdash f = (f1 \vee f2)$ **using** *assms* **by** *auto*

hence 2: $\vdash f \frown g = ((f1 \vee f2) \frown g)$ **by** (*rule LeftSChopEqvSChop*)

have 3: $\vdash (f1 \vee f2) \frown g = (f1 \frown g \vee f2 \frown g)$ **by** (*rule OrSChopEqv*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *SChopOrImpRule*:

assumes $\vdash g \longrightarrow g1 \vee g2$

shows $\vdash f \frown g \longrightarrow (f \frown g1) \vee (f \frown g2)$

proof –

have 1: $\vdash g \longrightarrow g1 \vee g2$ **using** *assms* **by** *auto*
hence 2: $\vdash f \frown g \longrightarrow f \frown (g1 \vee g2)$ **by** (*rule RightSChopImpSChop*)
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 *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 $\vdash bf\ f = bf\ g$
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*:
 $\vdash bf\ f \wedge (f1 \frown g) \longrightarrow (f \wedge f1) \frown g$
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*:
 $\vdash bi\ f \wedge (f1 \frown g) \longrightarrow (f \wedge f1) \frown g$
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

7.4 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*:

$\vdash \text{bf } f \wedge \text{finite} \longrightarrow f$

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 $\vdash bf\ f = bf\ g$
proof –
have 1: $\vdash finite \longrightarrow (f = g)$ **using** *assms* **by** *blast*
have 2: $\vdash finite \longrightarrow (f \longrightarrow g)$ **using** 1 **by** *auto*
have 3: $\vdash bf\ f \longrightarrow bf\ g$ **by** (*simp add: 2 FiniteImpBfImpBfRule*)
have 4: $\vdash finite \longrightarrow (g \longrightarrow f)$ **using** 1 **by** *auto*
have 5: $\vdash bf\ g \longrightarrow bf\ f$ **by** (*simp add: 4 FiniteImpBfImpBfRule*)
from 3 5 **show** *?thesis* **by** *fastforce*
qed

lemma *IfSChopEqvRule*:

assumes $\vdash f = if_i\ (init\ w)\ then\ f1\ else\ f2$
shows $\vdash f \frown g = if_i\ (init\ w)\ then\ (f1 \frown g)\ else\ (f2 \frown g)$
proof –
have 1: $\vdash f = if_i\ (init\ w)\ then\ f1\ else\ f2$
using *assms* **by** *auto*
hence 2: $\vdash f = (((init\ w) \wedge f1) \vee ((init\ (\neg w)) \wedge f2))$
unfolding *ifthenelse-d-def* **by** (*metis Initprop(2) int-eq*)
hence 3: $\vdash f \frown g = (((init\ w) \wedge f1) \frown g \vee ((init\ (\neg w)) \wedge f2) \frown g)$
by (*rule OrSChopEqvRule*)
have 4: $\vdash ((init\ w) \wedge f1) \frown g = ((init\ w) \wedge (f1 \frown g))$
by (*rule StateAndSChop*)
have 5: $\vdash ((init\ (\neg w)) \wedge f2) \frown g = ((init\ (\neg w)) \wedge (f2 \frown g))$
by (*rule StateAndSChop*)
have 6: $\vdash f \frown g = (((init\ w) \wedge f1 \frown g) \vee ((init\ (\neg w)) \wedge f2 \frown g))$
using 3 4 5 **by** *fastforce*
from 6 **show** *?thesis* **unfolding** *ifthenelse-d-def* **by** (*metis Initprop(2) inteq-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*:

$\vdash (empty \vee f) \frown g = (g \vee (f \frown g))$
proof –
have 1: $\vdash (empty \vee f) \frown g = ((empty \frown g) \vee (f \frown g))$ **by** (*rule OrSChopEqv*)
have 2: $\vdash empty \frown g = g$ **by** (*rule EmptySChop*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextSChopEqv*:

$\vdash (empty \vee \circ f) \frown g = (g \vee \circ (f \frown g))$
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:*

$\vdash \Box (\text{init } w) \frown \Box (\text{init } w) \wedge \text{inf} \longrightarrow \Box (\text{init } w)$

by (*metis AndChopA BoxStateChopBoxEqvBox OrFiniteInf Prop03 int-eq lift-imp-trans schop-d-def*)

lemma *BoxStateSChopBoxEqvBox:*

$\vdash \Box (\text{init } w) \frown \Box (\text{init } w) = \Box (\text{init } w)$

proof –

have 1: $\vdash (\Box (\text{init } w)) = ((\text{init } w) \wedge (\text{empty} \vee \bigcirc(\Box (\text{init } w))))$

by (*rule BoxEqvAndEmptyOrNextBox*)

hence 2: $\vdash (\Box (\text{init } w) \frown \Box (\text{init } w)) =$

$((\text{init } w) \wedge ((\text{empty} \vee \bigcirc(\Box (\text{init } w))) \frown \Box (\text{init } w)))$

by (*metis StateAndSChop inteq-reflection*)

have 3: $\vdash ((\text{empty} \vee \bigcirc(\Box (\text{init } w))) \frown \Box (\text{init } w)) =$

$(\Box (\text{init } w) \vee \bigcirc(\Box (\text{init } w) \frown \Box (\text{init } w)))$

by (*rule EmptyOrNextSChopEqv*)

have 4: $\vdash (\Box (\text{init } w) \frown \Box (\text{init } w)) =$

$((\text{init } w) \wedge (\Box (\text{init } w) \vee \bigcirc(\Box (\text{init } w) \frown \Box (\text{init } w))))$

using 2 3 by *fastforce*

have 5: $\vdash \neg (\Box (\text{init } w)) \longrightarrow \neg (\text{init } w) \vee \neg (\bigcirc(\Box (\text{init } w)))$

by (*rule NotBoxImpNotOrNotNextBox*)

have 6: $\vdash (\Box (\text{init } w) \frown \Box (\text{init } w)) \wedge \neg (\Box (\text{init } w)) \longrightarrow$

$\bigcirc(\Box (\text{init } w) \frown \Box (\text{init } w)) \wedge \neg (\bigcirc(\Box (\text{init } w)))$

using 4 5 by *fastforce*

hence 7: $\vdash \Box (\text{init } w) \frown \Box (\text{init } w) \wedge \text{finite} \longrightarrow \Box (\text{init } w)$

by (*rule NextContra*)

have 8: $\vdash \Box (\text{init } w) \frown \Box (\text{init } w) \wedge \text{inf} \longrightarrow \Box (\text{init } w)$

by (*rule BoxStateSChopBoxAndInfImpBox*)

have 9: $\vdash \Box (\text{init } w) \frown \Box (\text{init } w) \wedge (\text{finite} \vee \text{inf}) \longrightarrow \Box (\text{init } w)$

using 7 8 by *fastforce*

hence 10: $\vdash \Box (\text{init } w) \frown \Box (\text{init } w) \longrightarrow \Box (\text{init } w)$

using *FiniteOrInfinite* **by** *fastforce*

have 11: $\vdash \Box (\text{init } w) = ((\text{init } w) \wedge \Box (\text{init } w))$

by (*rule BoxEqvAndBox*)

have 12: $\vdash \text{empty} \frown \Box (\text{init } w) = \Box (\text{init } w)$

by (*rule EmptySChop*)

have 13: $\vdash ((\text{init } w) \wedge \text{empty}) \frown \Box (\text{init } w) = ((\text{init } w) \wedge (\text{empty} \frown \Box (\text{init } w)))$

by (*rule StateAndSChop*)

have 14: $\vdash \Box (\text{init } w) = ((\text{init } w) \wedge \text{empty}) \frown \Box (\text{init } w)$

using 11 12 13 by *fastforce*

have 15: $\vdash (\text{init } w) \wedge \text{empty} \longrightarrow \Box (\text{init } w)$

by (*rule StateAndEmptyImpBoxState*)

hence 16: $\vdash ((\text{init } w) \wedge \text{empty}) \frown \Box (\text{init } w) \longrightarrow \Box (\text{init } w) \frown \Box (\text{init } w)$

by (*rule LeftSChopImpSChop*)

have 17: $\vdash \Box (\text{init } w) \longrightarrow \Box (\text{init } w) \frown \Box (\text{init } w)$

using 14 16 by *fastforce*

from 10 17 show ?thesis by fastforce
qed

lemma *NotBoxStateImpBoxSYieldsNotBox*:

$\vdash \neg(\Box(\text{init } w)) \longrightarrow (\Box(\text{init } w) \text{ syields } (\neg(\Box(\text{init } w))))$

proof –

have 1: $\vdash \Box(\text{init } w) \frown \Box(\text{init } w) = \Box(\text{init } w)$ **by** (rule *BoxStateSChopBoxEqvBox*)

have 2: $\vdash \Box(\text{init } w) = (\neg \neg(\Box(\text{init } w)))$ **by** auto

hence 3: $\vdash \Box(\text{init } w) \frown \Box(\text{init } w) = \Box(\text{init } w) \frown (\neg \neg(\Box(\text{init } w)))$ **by** (rule *RightSChopEqvSChop*)

have 4: $\vdash \neg(\Box(\text{init } w)) \longrightarrow \neg(\Box(\text{init } w) \frown (\neg \neg(\Box(\text{init } w))))$ **using** 1 3 **by** auto

from 4 **show** ?thesis **by** (simp add: syields-d-def)

qed

lemma *StateEqvBf*:

$\vdash (\text{init } w) = \text{bf } (\text{init } w)$

proof –

have 1: $\vdash (\text{init } w) \longrightarrow \text{bf } (\text{init } w)$ **by** (rule *StateImpBf*)

have 2: $\vdash \text{bf } (\text{init } w) \wedge \text{finite} \longrightarrow (\text{init } w)$ **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*:

$\vdash \# \text{True} \frown f = \Diamond f$

using *DiamondSChopdef* **by** fastforce

lemma *BfAndEqvBfAndBf*:

$\vdash \text{bf}(f \wedge g) = (\text{bf } f \wedge \text{bf } g)$

proof –

have 1: $\vdash f \wedge g \longrightarrow f$ **by** auto

have 2: $\vdash \text{bf}(f \wedge g) \longrightarrow \text{bf } f$ **by** (simp add: 1 *BfImpBfRule*)

have 3: $\vdash f \wedge g \longrightarrow g$ **by** auto

have 4: $\vdash \text{bf}(f \wedge g) \longrightarrow \text{bf } g$ **by** (simp add: 3 *BfImpBfRule*)

have 5: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ **by** auto

have 6: $\vdash \text{bf } f \longrightarrow \text{bf } (g \longrightarrow f \wedge g)$ **by** (simp add: 5 *BfImpBfRule*)

have 7: $\vdash \text{bf } (g \longrightarrow f \wedge g) \longrightarrow (\text{bf } g \longrightarrow \text{bf}(f \wedge g))$ **by** (simp add: *BfImpDist*)

have 8: $\vdash \text{bf } f \wedge \text{bf } g \longrightarrow \text{bf } (f \wedge g)$ **using** 6 7 **by** fastforce

from 2 4 8 **show** ?thesis **by** fastforce

qed

lemma *BfEqvBfImpAndBfImp*:

$\vdash \text{bf}(f = g) = (\text{bf } (f \longrightarrow g) \wedge \text{bf } (g \longrightarrow f))$

proof –

have 1: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** auto

have 2: $\vdash \text{bf}(f = g) = \text{bf}((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** (simp add: 1 *BfEqvBf*)

have 3: $\vdash \text{bf}((f \longrightarrow g) \wedge (g \longrightarrow f)) = (\text{bf } (f \longrightarrow g) \wedge \text{bf } (g \longrightarrow f))$ **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: $\vdash bf(f = f1) = (bf(f \longrightarrow f1) \wedge bf(f1 \longrightarrow f))$ **by** (*simp add: BfEqvBfImpAndBfImp*)

have 2: $\vdash bf(f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (*simp add: BfSChopImpSChop*)

have 3: $\vdash bf(f1 \longrightarrow f) \longrightarrow f1 \frown g \longrightarrow f \frown g$ **by** (*simp add: BfSChopImpSChop*)

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

qed

lemma *BfEqvDfEqvDf*:

$\vdash bf(f = g) \longrightarrow (df f = df g)$

proof –

have 1: $\vdash bf(f = g) \longrightarrow (f \frown \#True) = (g \frown \#True)$

using *BfEqvImpSChopEqvSChop* **by** *fastforce*

from 1 **show** *?thesis* **by** (*simp add: df-d-def*)

qed

lemma *FiniteImpEqvDfImpRule*:

assumes $\vdash finite \longrightarrow f = g$

shows $\vdash df f = df g$

proof –

have 1: $\vdash finite \longrightarrow f = g$ **using** *assms* **by** *auto*

have 2: $\vdash bf(f = g)$ **using** 1 **by** (*simp add: FiniteBfGen*)

have 3: $\vdash bf(f = g) \longrightarrow (df f = df g)$ **by** (*simp add: BfEqvDfEqvDf*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *DfEmpty*:

$\vdash df\ empty$

proof –

have 1: $\vdash \#True$ **by** *auto*

have 2: $\vdash empty \frown \#True = \#True$ **by** (*rule EmptySChop*)

have 3: $\vdash empty \frown \#True$ **using** 1 2 **by** *auto*

from 3 **show** *?thesis* **by** (*simp add: df-d-def*)

qed

lemma *BfImpDf*:

$\vdash bf f \longrightarrow df f$

proof –

have 1: $\vdash f \longrightarrow (empty \longrightarrow f)$ **by** *auto*

have 2: $\vdash bf f \longrightarrow bf(empty \longrightarrow f)$ **by** (*simp add: 1 BfImpBfRule*)

have 3: $\vdash bf(empty \longrightarrow f) \longrightarrow df\ empty \longrightarrow df f$ **by** (*simp add: BfImpDfImpDf*)

have 4: $\vdash bf f \longrightarrow df\ empty \longrightarrow df f$ **using** 2 3 *lift-imp-trans* **by** *blast*

have 5: $\vdash df\ empty$ **by** (*simp add: DfEmpty*)
from 4 5 **show** ?thesis **by** fastforce
qed

7.5 Properties of SDa and SBa

lemma *SDaEqvDtDf*:

$\vdash sda\ f = \Diamond (df\ f)$

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*:

$\vdash sda\ f = df\ (\Diamond f)$

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*:

$\vdash sba\ f = bf\ (\Box f)$

proof –

have 1: $\vdash sda\ (\neg f) = df\ (\Diamond (\neg f))$ **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: $\vdash sda\ (\neg f) = df\ (\neg (\Box f))$ **using** 1 3 **by** fastforce
hence 5: $\vdash (\neg (sda\ (\neg f))) = (\neg (df\ (\neg (\Box f))))$ **by** auto
hence 6: $\vdash (\neg (sda\ (\neg f))) = bf\ (\Box f)$ **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*:

$\vdash sba\ f \wedge finite \longrightarrow \Box\ f$

proof –

have 1: $\vdash sba\ f = bf(\Box\ f)$ **by** (rule *SBaEqvBfBt*)
have 2: $\vdash bf(\Box\ f) \wedge finite \longrightarrow \Box\ f$ **by** (rule *BfElim*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *DiamondImpSDa*:

$\vdash \Diamond\ f \wedge finite \longrightarrow sda\ f$

by (metis *AndFiniteImpDf SDaEqvDfDt inteq-reflection*)

lemma *DfImpSDa*:

$\vdash df\ f \longrightarrow sda\ f$

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*:

$\vdash sba\ f \wedge finite \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown (f \wedge g1)$

proof –

have 1: $\vdash sba\ f \longrightarrow bf\ f$ **by** (rule *SBaImpBf*)
have 2: $\vdash bf\ f \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown g1$ **by** (rule *BfAndSChopImport*)
have 3: $\vdash sba\ f \wedge finite \longrightarrow \Box\ f$ **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*:

$\vdash ba\ f \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown (f \wedge g1)$

proof –

have 1: $\vdash ba\ f \longrightarrow bi\ f$ **by** (rule *BaImpBi*)
have 2: $\vdash bi\ f \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown g1$ **by** (rule *BiAndSChopImport*)
have 3: $\vdash ba\ f \longrightarrow \Box\ f$ **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*:

$\vdash ((init \ w) \wedge \text{empty}) \frown f = ((init \ w) \wedge f)$

proof –

have 1: $\vdash ((init \ w) \wedge \text{empty}) \frown f = ((init \ w) \wedge \text{empty} \frown f)$ **by** (rule *StateAndSChop*)

have 2: $\vdash \text{empty} \frown f = f$ **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:

$\vdash ((init\ w) \longrightarrow (f\ syields\ g)) = ((init\ w) \wedge f)\ syields\ g$

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 $\vdash (\text{init } w) \longrightarrow f$

shows $\vdash (\text{init } w) \longrightarrow \text{bf } f$

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: $\vdash \text{df } (\neg f) \longrightarrow \text{df } (\neg (\text{init } w))$ **by** (*rule DfImpDf*)

hence 4: $\vdash \text{df } (\neg f) \longrightarrow \text{df } (\text{init } (\neg w))$ **by** (*metis Initprop(2) inteq-reflection*)

have 5: $\vdash \text{df } (\text{init } (\neg w)) = (\text{init } (\neg w))$ **by** (*rule DfState*)

have 6: $\vdash \text{df } (\neg f) \longrightarrow \neg (\text{init } w)$ **using** 4 5 **using** *Initprop(2)* **by** *fastforce*

hence 7: $\vdash (\text{init } w) \longrightarrow \neg (\text{df } (\neg f))$ **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

7.6 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*:
 $\vdash \text{sf}in\ w \longrightarrow \Box(\text{sf}in\ w)$
by (*metis* (*mono-tags*, *lifting*) *DiamondFin* *always-d-def* *intI* *int-eq* *int-simps*(4) *sf}in-d-def* *unl-lift2*)

lemma *SFinAndSChopImport*:
 $\vdash (\text{sf}in\ w) \wedge (f \frown g) \longrightarrow f \frown ((\text{sf}in\ w) \wedge g)$
proof –
have 1: $\vdash \text{sf}in\ w \longrightarrow \Box(\text{sf}in\ w)$ **by** (*rule* *SFinImpBox*)
hence 2: $\vdash \text{sf}in\ w \wedge (f \frown g) \longrightarrow \Box(\text{sf}in\ w) \wedge (f \frown g)$ **by** *auto*
have 3: $\vdash \Box(\text{sf}in\ w) \wedge (f \frown g) \longrightarrow f \frown ((\text{sf}in\ w) \wedge g)$ **using** *BoxAndSChopImport* **by** *blast*
from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *SFinAndSChop*:
 $\vdash (f \frown (g \wedge \text{sf}in\ w)) = (\text{sf}in\ w \wedge f \frown g)$
using *SFinAndSChopImport* *SChopSFinExportA* *SChopAndA* *SChopAndCommute*
by *fastforce*

lemma *SChopAndEmptyEqvEmptySChopEmpty*:
 $\vdash ((f \frown g) \wedge \text{empty}) = (f \wedge \text{empty}) \frown (g \wedge \text{empty})$
by (*auto* *simp*: *itl-defs* *zero-enat-def*)

lemma *SFinAndEmpty*:
 $\vdash ((\text{sf}in\ w) \wedge \text{empty}) = (w \wedge \text{empty})$
proof –
have 1: $\vdash ((\text{sf}in\ w) \wedge \text{empty}) = (\#True \frown (w \wedge \text{empty}) \wedge \text{empty})$
using *SFinEqvTrueSChopAndEmpty* **by** *fastforce*
have 2: $\vdash (\#True \frown (w \wedge \text{empty}) \wedge \text{empty}) = ((\#True \wedge \text{empty}) \frown (w \wedge \text{empty}))$
by (*simp* *add*: *FiniteChopAndEmptyEqvChopAndEmpty* *schop-d-def*)
have 3: $\vdash (\#True \wedge \text{empty}) \frown (w \wedge \text{empty}) = (\text{empty} \frown (w \wedge \text{empty}))$
using *LeftSChopEqvSChop* **by** *fastforce*
have 4: $\vdash (\text{empty} \frown (w \wedge \text{empty})) = (w \wedge \text{empty})$
using *EmptySChop* **by** *blast*
from 1 2 3 4 **show** *?thesis* **by** *fastforce*
qed

lemma *AndSFinEqvSChopAndEmpty*:
 $\vdash ((f \wedge \text{finite}) \wedge \text{sf}in\ g) = f \frown (g \wedge \text{empty})$
proof –
have 1: $\vdash ((f \wedge \text{finite}) \wedge \text{sf}in\ g) = (f \frown \text{empty} \wedge \text{sf}in\ g)$
using *SChopEmpty*
by (*metis* (*no-types*, *lifting*) *DiamondEmptyEqvFinite* *FiniteImpAnd* *Prop10* *SChopImpDiamond* *inteq-reflection* *lift-and-com*)
have 2: $\vdash (\text{sf}in\ g \wedge f \frown \text{empty}) = (f \frown (\text{empty} \wedge \text{sf}in\ g))$
using *SFinAndSChop* **by** *fastforce*
have 3: $\vdash (\text{empty} \wedge \text{sf}in\ g) = (\text{sf}in\ g \wedge \text{empty})$
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*:
 $\vdash ((f \wedge finite) \wedge sfin\ (init\ w)) = f \frown ((init\ w) \wedge empty)$
using *AndSFinEqvSChopAndEmpty* **by** *blast*

lemma *DiamondEqvEmptyOrNextDiamond*:

$\vdash \Diamond f = (f \vee \bigcirc(\Diamond f))$

proof –

have 1: $\vdash \Box (\neg f) = ((\neg f) \wedge wnext(\Box (\neg f)))$
by (*simp* *add*: *BoxEqvAndWnextBox*)
have 2: $\vdash (\neg \Diamond f) = ((\neg f) \wedge wnext(\Box (\neg f)))$
using 1 **by** (*simp* *add*: *always-d-def*)
have 3: $\vdash \Diamond f = (f \vee \neg(wnext(\Box (\neg f))))$
using 2 **by** *auto*
have 4: $\vdash (\neg(wnext(\Box (\neg f)))) = \bigcirc(\neg\Box(\neg f))$
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*:

$\vdash sfin\ (init\ w) = (((init\ w) \wedge empty) \vee \bigcirc(sfin\ (init\ w)))$

proof –

have 01: $\vdash sfin\ (init\ w) = \#True \frown ((init\ w) \wedge empty)$
by (*simp* *add*: *SFinEqvTrueSChopAndEmpty*)
have 02: $\vdash \#True \frown ((init\ w) \wedge empty) = \Diamond((init\ w) \wedge empty)$
by (*simp* *add*: *TrueSChopEqvDiamond*)
have 03: $\vdash \Diamond((init\ w) \wedge empty) = (((init\ w) \wedge empty) \vee \bigcirc(sfin\ (init\ w)))$
using *DiamondEqvEmptyOrNextDiamond* 02 01 **by** (*metis* *inteq-reflection*)
from 01 02 03 **show** *?thesis* **by** *fastforce*
qed

lemma *SFinSChopEqvOr*:

$\vdash (sfin\ (init\ w)) \frown f = (((init\ w) \wedge f) \vee \bigcirc((sfin\ (init\ w)) \frown f))$

proof –

have 1: $\vdash sfin\ (init\ w) = (((init\ w) \wedge empty) \vee \bigcirc(sfin\ (init\ w)))$
by (*rule* *SFinStateEqvStateAndEmptyOrNextSFinState*)
hence 2: $\vdash (sfin\ (init\ w)) \frown f = (((init\ w) \wedge empty) \vee \bigcirc(sfin\ (init\ w))) \frown f$
by (*rule* *LeftSChopEqvSChop*)

have 3: $\vdash (((init\ w) \wedge empty) \vee \bigcirc (sfin\ (init\ w))) \frown f$
 $= (((init\ w) \wedge empty) \frown f \vee (\bigcirc (sfin\ (init\ w))) \frown f)$
by (rule OrSCHopEqv)
have 4: $\vdash ((init\ w) \wedge empty) \frown f = ((init\ w) \wedge f)$
by (rule StateAndEmptySCHop)
have 5: $\vdash (\bigcirc (sfin\ (init\ w))) \frown f = \bigcirc((sfin\ (init\ w)) \frown f)$
by (rule NextSCHop)
from 2 3 4 5 **show** ?thesis **by** fastforce
qed

lemma SFinSCHopEqvDiamond:

$\vdash (\bigcirc (sfin\ (init\ w))) \frown f = \Diamond ((init\ w) \wedge f)$

proof –

have 1: $\vdash (\bigcirc (sfin\ (init\ w))) = (\#True \frown ((init\ w) \wedge empty))$
by (simp add: SFinEqvTrueSCHopAndEmpty)
hence 2: $\vdash (\bigcirc (sfin\ (init\ w))) \frown f = (\#True \frown ((init\ w) \wedge empty)) \frown f$
by (rule LeftSCHopEqvSCHop)
have 3: $\vdash \#True \frown ((init\ w) \wedge empty) \frown f = (\#True \frown ((init\ w) \wedge empty)) \frown f$
by (rule SCHopAssoc)
have 4: $\vdash \#True \frown ((init\ w) \wedge empty) \frown f = \Diamond ((init\ w) \wedge empty) \frown f$
using TrueSCHopEqvDiamond **by** blast
have 5: $\vdash ((init\ w) \wedge empty) \frown f = ((init\ w) \wedge f)$
using StateAndEmptySCHop **by** blast
hence 6: $\vdash \Diamond ((init\ w) \wedge empty) \frown f = \Diamond ((init\ w) \wedge f)$
by (rule DiamondEqvDiamond)
from 2 3 4 6 **show** ?thesis **by** fastforce
qed

lemma SFinSYields:

$\vdash (\bigcirc (sfin\ (init\ w)))\ syields\ (init\ w)$

proof –

have 1: $\vdash (\bigcirc (sfin\ (init\ w))) \frown (\neg (init\ w)) = \Diamond ((init\ w) \wedge \neg (init\ w))$
by (rule SFinSCHopEqvDiamond)
have 2: $\vdash \neg (\Diamond ((init\ w) \wedge \neg (init\ w)))$
by (rule NotDiamondAndNot)
have 3: $\vdash \neg ((\bigcirc (sfin\ (init\ w))) \frown (\neg (init\ w)))$
using 1 2 **by** fastforce
from 3 **show** ?thesis **by** (simp add: syields-d-def)
qed

lemma SFinEqvFinAndFinite:

$\vdash (finite \wedge fin\ f) = sfin\ f$

by (metis AndFinEqvChopAndEmpty DiamondSCHopdef SFinEqvTrueSCHopAndEmpty int-simps(20) inteq-reflection sometimes-d-def)

lemma AndFiniteImpAndSFinStateOrSFinNotState:

$\vdash f \wedge finite \longrightarrow (f \wedge sfin\ (init\ w)) \vee (f \wedge sfin\ (\neg (init\ w)))$

proof –

have 1: $\vdash (\neg fin\ (init\ (\neg w))) \wedge finite = (fin\ (init\ (\neg \neg w))) \wedge finite$
using FinNotStateEqvNotFinState **by** blast

have 2: $\vdash (fin (init (\neg \neg w)) \wedge finite) = (fin (init (w)) \wedge finite)$
by *simp*
have 3: $\vdash f \wedge finite \longrightarrow (f \wedge finite) \wedge fin (init w) \vee (f \wedge finite) \wedge fin (\neg init w)$
by (*simp add: ImpAndFinStateOrFinNotState*)
have 4: $\vdash (finite \wedge fin (init w)) = sfin(init w)$
using *SFinEqvFinAndFinite*[of *LIFT*(*init w*)] **by** *fastforce*
have 5: $\vdash ((f \wedge finite) \wedge fin (init w)) = (f \wedge sfin (init w))$
using 4 **by** *auto*
have 6: $\vdash (finite \wedge fin (\neg(init w))) = sfin(\neg(init w))$
using *SFinEqvFinAndFinite*[of *LIFT*($\neg(init w)$)] **by** *fastforce*
have 7: $\vdash ((f \wedge finite) \wedge fin (\neg (init w))) = (f \wedge sfin (\neg (init w)))$
using 6 **by** *auto*
show *?thesis*
using 3 5 7 **by** *fastforce*
qed

lemma *AndSFinSChopEqvStateAndSChop*:

$\vdash (f \wedge sfin (init w)) \frown g = f \frown ((init w) \wedge g)$

proof –

have 1: $\vdash (sfin (init w)) syields (init w)$
by (*rule SFinSYields*)
have 2: $\vdash f \wedge sfin (init w) \longrightarrow sfin (init w)$
by *auto*
hence 3: $\vdash (sfin (init w)) syields (init w) \longrightarrow (f \wedge sfin (init w)) syields (init w)$
using *LeftSYieldsImpSYields* **by** *metis*
have 4: $\vdash (f \wedge sfin (init w)) syields (init w)$
using 1 3 *MP* **by** *fastforce*
have 5: $\vdash (f \wedge sfin (init w)) \frown g \wedge (f \wedge sfin (init w)) syields (init w) \longrightarrow (f \wedge sfin (init w)) \frown (g \wedge (init w))$
by (*rule SChopAndSYieldsImp*)
have 6: $\vdash (f \wedge sfin (init w)) \frown g \longrightarrow (f \wedge sfin (init w)) \frown (g \wedge (init w))$
using 4 5 **by** *fastforce*
have 7: $\vdash (f \wedge sfin (init w)) \frown (g \wedge (init w)) \longrightarrow f \frown (g \wedge (init w))$
by (*rule AndSChopA*)
have 8: $\vdash g \wedge (init w) \longrightarrow (init w) \wedge g$
by *auto*
hence 9: $\vdash f \frown (g \wedge (init w)) \longrightarrow f \frown ((init w) \wedge g)$
by (*rule RightSChopImpSChop*)
have 10: $\vdash (f \wedge sfin (init w)) \frown g \longrightarrow f \frown ((init w) \wedge g)$
using 6 7 9 **by** *fastforce*
have 11: $\vdash (f \wedge finite) \longrightarrow (f \wedge sfin (init w)) \vee (f \wedge sfin (\neg (init w)))$
using *AndFiniteImpAndSFinStateOrSFinNotState* **by** *blast*
hence 12: $\vdash f \frown ((init w) \wedge g) \longrightarrow ((f \wedge sfin (init w)) \vee (f \wedge sfin (\neg (init w)))) \frown ((init w) \wedge g)$
by (*metis FiniteImp LeftChopImpChop integ-reflection chop-d-def*)
have 13: $\vdash ((f \wedge sfin (init w)) \vee (f \wedge sfin (\neg (init w)))) \frown ((init w) \wedge g)$
 $=$
 $((f \wedge sfin (init w)) \frown ((init w) \wedge g) \vee (f \wedge sfin (\neg (init w))) \frown ((init w) \wedge g))$
by (*rule OrSChopEqv*)

have 14: $\vdash (f \wedge \text{sfm } (\text{init } (\neg w))) \frown ((\text{init } w) \wedge g) \longrightarrow \Diamond ((\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g))$
using *SFinSChopEqvDiamond*
by (*metis SChopImpSChop Prop12 int-iffD1 inteq-reflection lift-and-com*)
have 141: $\vdash \neg (\Diamond ((\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g))) \longrightarrow$
 $\neg ((f \wedge \text{sfm } (\text{init } (\neg w))) \frown ((\text{init } w) \wedge g))$
using 14 by fastforce
have 150: $\vdash ((\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g)) = \#False$
using *Initprop(2)* **by fastforce**
have 15: $\vdash \neg (\Diamond ((\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g)))$
by (*metis 150 NotDiamondAndNot int-eq int-simps(21)*)
have 151: $\vdash \neg ((f \wedge \text{sfm } (\text{init } (\neg w))) \frown ((\text{init } w) \wedge g))$
using 15 141 by fastforce
have 1511: $\vdash (f \wedge \text{sfm } (\neg (\text{init } w))) \frown ((\text{init } w) \wedge g) \longrightarrow \#False$
using 151 by (*metis Initprop(2) int-simps(14) inteq-reflection*)
have 152: $\vdash (f \wedge \text{sfm } (\text{init } w)) \frown ((\text{init } w) \wedge g) \vee (f \wedge \text{sfm } (\neg (\text{init } w))) \frown ((\text{init } w) \wedge g) \longrightarrow$
 $(f \wedge \text{sfm } (\text{init } w)) \frown ((\text{init } w) \wedge g)$
using 1511 by fastforce
have 16: $\vdash f \frown ((\text{init } w) \wedge g) \longrightarrow (f \wedge \text{sfm } (\text{init } w)) \frown ((\text{init } w) \wedge g)$
using 12 13 152 by fastforce
have 17: $\vdash (f \wedge \text{sfm } (\text{init } w)) \frown ((\text{init } w) \wedge g) \longrightarrow (f \wedge \text{sfm } (\text{init } w)) \frown g$
by (*rule SChopAndB*)
have 18: $\vdash f \frown ((\text{init } w) \wedge g) \longrightarrow (f \wedge \text{sfm } (\text{init } w)) \frown g$
using 16 17 by fastforce
from 10 18 show ?thesis by fastforce
qed

lemma DfAndSFinEqvSChopState:

$\vdash df (f \wedge \text{sfm } (\text{init } w)) = f \frown (\text{init } w)$

proof –

have 1: $\vdash (f \wedge \text{sfm } (\text{init } w)) \frown \#True = f \frown ((\text{init } w) \wedge \#True)$

by (*rule AndSFinSChopEqvStateAndSChop*)

have 2: $\vdash ((\text{init } w) \wedge \#True) = (\text{init } w)$

by auto

hence 3: $\vdash (f \frown ((\text{init } w) \wedge \#True)) = (f \frown (\text{init } w))$

by (*rule RightSChopEqvSChop*)

have 4: $\vdash (f \wedge \text{sfm } (\text{init } w)) \frown \#True = f \frown (\text{init } w)$

using 1 3 by auto

from 4 show ?thesis by (simp add: df-d-def)

qed

lemma SFinNotStateEqvNotSFinState:

$\vdash \text{finite} \longrightarrow (\neg (\text{sfm } (\text{init } w))) = (\text{sfm } (\text{init } (\neg w)))$

using *SFinEqvTrueSChopAndEmpty*

by (*metis Initprop(2) SFinprop(3) int-eq*)

lemma BfImpSFinEqvSYieldsState:

$\vdash bf (f \longrightarrow \text{sfm } (\text{init } w)) = f \text{ syields } (\text{init } w)$

proof –

have 1: $\vdash df (f \wedge \text{sfm } (\text{init } (\neg w))) = f \frown (\text{init } (\neg w))$

by (*rule DfAndSFinEqvSChopState*)

have 2: $\vdash \text{finite} \longrightarrow (f \wedge \text{sfm}(\text{init } (\neg w))) = (f \wedge \neg(\text{sfm}(\text{init } w)))$
using *SFinNotStateEqvNotSFinState* **by** *fastforce*
have 3: $\vdash (f \wedge \neg(\text{sfm}(\text{init } w))) = (\neg(f \longrightarrow \text{sfm}(\text{init } w)))$
by *auto*
have 4: $\vdash \text{finite} \longrightarrow (f \wedge \text{sfm}(\text{init } (\neg w))) = (\neg(f \longrightarrow \text{sfm}(\text{init } w)))$
using 2 3 **by** *fastforce*
hence 5: $\vdash \text{df } (f \wedge \text{sfm}(\text{init } (\neg w))) = \text{df } (\neg(f \longrightarrow \text{sfm}(\text{init } w)))$
by (*metis DfEqvNotBfNot FiniteImpAnd df-d-def inteq-reflection schop-d-def*)
have 6: $\vdash \text{df } (\neg(f \longrightarrow \text{sfm}(\text{init } w))) = (\neg(\text{bf } (f \longrightarrow \text{sfm}(\text{init } w))))$
by (*rule DfNotEqvNotBf*)
have 7: $\vdash \neg(\text{bf } (f \longrightarrow \text{sfm}(\text{init } w))) = f \frown (\text{init } (\neg w))$
using 1 5 6 *Initprop* **by** *fastforce*
hence 8: $\vdash \text{bf } (f \longrightarrow \text{sfm}(\text{init } w)) = (\neg(f \frown (\neg(\text{init } w))))$
by (*metis Initprop(2) int-eq int-simps(7)*)
from 8 **show** ?thesis **by** (*simp add: syields-d-def*)
qed

lemma *StateImpSYields*:

assumes $\vdash (\text{init } w) \wedge f \longrightarrow \text{sfm}(\text{init } w1)$
shows $\vdash (\text{init } w) \longrightarrow (f \text{ syields } (\text{init } w1))$
proof –
have 1: $\vdash (\text{init } w) \wedge f \longrightarrow \text{sfm}(\text{init } w1)$ **using** *assms* **by** *auto*
hence 2: $\vdash (\text{init } w) \longrightarrow (f \longrightarrow \text{sfm}(\text{init } w1))$ **by** *auto*
hence 3: $\vdash (\text{init } w) \longrightarrow \text{bf } (f \longrightarrow \text{sfm}(\text{init } w1))$ **using** *StateImpBfGen* **by** *auto*
have 4: $\vdash \text{bf } (f \longrightarrow \text{sfm}(\text{init } w1)) = f \text{ syields } (\text{init } w1)$ **by** (*rule BfImpSFinEqvSYieldsState*)
from 3 4 **show** ?thesis **by** *fastforce*
qed

lemma *StateAndSYieldsImpSYields*:

assumes $\vdash (\text{init } w) \wedge f \longrightarrow f1$
shows $\vdash (\text{init } w) \wedge (f1 \text{ syields } g) \longrightarrow (f \text{ syields } g)$
proof –
have 1: $\vdash (\text{init } w) \wedge f \longrightarrow f1$ **using** *assms* **by** *auto*
hence 2: $\vdash (\text{init } w) \wedge (f \frown (\neg g)) \longrightarrow f1 \frown (\neg g)$ **by** (*rule StateAndSChopImpSChopRule*)
hence 3: $\vdash (\text{init } w) \wedge \neg(f1 \frown (\neg g)) \longrightarrow \neg(f \frown (\neg g))$ **by** *auto*
from 3 **show** ?thesis **by** (*simp add: syields-d-def*)
qed

lemma *AndSYieldsA*:

$\vdash f \text{ syields } g \longrightarrow (f \wedge f1) \text{ syields } g$
proof –
have 1: $\vdash f \wedge f1 \longrightarrow f$ **by** *auto*
from 1 **show** ?thesis **by** (*rule LeftSYieldsImpSYields*)
qed

lemma *AndSYieldsB*:

$\vdash f1 \text{ syields } g \longrightarrow (f \wedge f1) \text{ syields } g$
proof –
have 1: $\vdash f \wedge f1 \longrightarrow f1$ **by** *auto*
from 1 **show** ?thesis **by** (*rule LeftSYieldsImpSYields*)
qed

qed

lemma *RightSYieldsImpSYields*:

assumes $\vdash g \longrightarrow g1$

shows $\vdash (f \text{ syields } g) \longrightarrow (f \text{ syields } g1)$

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*:

$\vdash (f \text{ syields } g) \wedge (f1 \text{ syields } g1) \longrightarrow (f \wedge f1) \text{ syields } (g \wedge g1)$

proof –

have 1: $\vdash f \text{ syields } g \longrightarrow (f \wedge f1) \text{ syields } g$

by (rule *AndSYieldsA*)

have 2: $\vdash f1 \text{ syields } g1 \longrightarrow (f \wedge f1) \text{ syields } g1$

by (rule *AndSYieldsB*)

have 3: $\vdash ((f \wedge f1) \text{ syields } g \wedge (f \wedge f1) \text{ syields } g1) = (f \wedge f1) \text{ syields } (g \wedge g1)$

by (rule *SYieldsAndSYieldsEqvSYieldsAnd*)

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

qed

lemma *SYieldsSYieldsEqvSChopSYields*:

$\vdash f \text{ syields } (g \text{ syields } h) = (f \frown g) \text{ syields } h$

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: $\vdash f \frown (\neg (g \text{ syields } h)) = (f \frown g) \frown (\neg h)$ **by** (simp add: syields-d-def)

hence 7: $\vdash (\neg (f \frown (\neg (g \text{ syields } h)))) = (\neg ((f \frown g) \frown (\neg h)))$ **by** auto

from 7 **show** ?thesis **by** (simp add: syields-d-def)

qed

lemma *EmptyYields*:

$\vdash \text{empty} \text{ syields } f = f$

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*:

$\vdash (\circ f) \text{ syields } g = \text{wnext } (f \text{ syields } g)$

proof –

have 1: $\vdash (\bigcirc f) \frown (\neg g) = \bigcirc(f \frown (\neg g))$ **by** (rule *NextSChop*)
hence 2: $\vdash (\neg ((\bigcirc f) \frown (\neg g))) = (\neg (\bigcirc(f \frown (\neg g))))$ **by** *auto*
hence 3: $\vdash (\bigcirc f) \text{ syields } g = (\neg (\bigcirc(f \frown (\neg g))))$ **by** (simp add: *syields-d-def*)
have 4: $\vdash (\neg (\bigcirc(f \frown (\neg g)))) = \text{wnext } (\neg (f \frown (\neg g)))$ **by** (auto simp: *wnext-d-def*)
have 5: $\vdash (\bigcirc f) \text{ syields } g = \text{wnext } (\neg (f \frown (\neg g)))$ **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: $\vdash \bigcirc(\# \text{True} \frown (f \wedge \text{finite})) \longrightarrow \text{fmore}$

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: $\vdash \text{more} \frown f = (\bigcirc \# \text{True}) \frown f$ **by** (*simp add: more-d-def*)

have 2: $\vdash (\bigcirc \# \text{True}) \frown f = \bigcirc(\# \text{True} \frown f)$ **by** (*rule NextSChop*)

have 3: $\vdash \text{more} \frown f = \bigcirc(\# \text{True} \frown f)$ **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*:
 $\vdash \text{sda } (\neg f) = (\neg (\text{sba } f))$
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: $\vdash \neg \neg f \longrightarrow \neg (sba (\neg f) \wedge finite)$ **by** *fastforce*
have 3: $\vdash f = (\neg \neg f)$ **by** *simp*
have 4: $\vdash sda f = (\neg (sba (\neg f)))$ **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*:
 $\vdash sba\ (f = g) \longrightarrow (sba\ f = sba\ g)$
proof –
have 1: $\vdash sba\ (f \longrightarrow g) \longrightarrow sba\ f \longrightarrow sba\ g$
by (rule *SBaImpDist*)
have 2: $\vdash sba\ (g \longrightarrow f) \longrightarrow sba\ g \longrightarrow sba\ f$
by (rule *SBaImpDist*)
have 3: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$
by *auto*
hence 31: $\vdash sba(f = g) = sba\ ((f \longrightarrow g) \wedge (g \longrightarrow f))$
using *inteq-reflection* **by** *force*
have 4: $\vdash sba\ ((f \longrightarrow g) \wedge (g \longrightarrow f)) = (sba((f \longrightarrow g)) \wedge sba((g \longrightarrow f)))$
by (rule *SBaAndEqv*)
have 5: $\vdash ((sba\ f \longrightarrow sba\ g) \wedge (sba\ g \longrightarrow sba\ f)) = (sba\ f = sba\ g)$
by *auto*
from 1 2 31 4 5 **show** *?thesis* **by** *fastforce*
qed

lemma *SBaImpSBa*:
assumes $\vdash f \longrightarrow g$
shows $\vdash sba\ f \longrightarrow sba\ g$
using *SBaGen SBaImpDist MP assms* **by** *metis*

lemma *SBaEqvSBa*:
assumes $\vdash f = g$
shows $\vdash sba\ f = sba\ g$
using *SBaGen SBaImpSBaEqvSBa MP assms* **by** *metis*

lemma *SDaImpSDa*:
assumes $\vdash f \longrightarrow g$
shows $\vdash sda\ f \longrightarrow sda\ g$
using *assms* **by** (*metis SDaEqvDtDf DfAndB DiamondImpDiamond inteq-reflection Prop10*)

lemma *SDaEqvSDaSDa*:
 $\vdash sda\ f = sda\ (sda\ f)$
proof –
have 1: $\vdash sda\ f = \Diamond(df\ f)$
by (rule *SDaEqvDtDf*)
have 2: $\vdash df\ f = (df\ (df\ f))$
by (rule *DfEqvDfDf*)
hence 3: $\vdash \Diamond(df\ f) = \Diamond(df\ (df\ f))$
by (rule *DiamondEqvDiamond*)
have 4: $\vdash \Diamond(df\ f) = \Diamond(\Diamond(df\ (df\ f)))$

using *DiamondEqvDiamondDiamond DfEqvDfDf* using 3 by *fastforce*
 have 5: $\vdash \Diamond (df (df f)) = df (\Diamond (df f))$
 by (rule *DtDfEqvDfDt*)
 hence 6: $\vdash \Diamond (\Diamond (df (df f))) = \Diamond (df (\Diamond (df f)))$
 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: $\vdash \text{ba } (g \longrightarrow g1) \longrightarrow \Box(g \longrightarrow g1)$ **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*:

$\vdash (f \frown f1) \wedge \text{sba } g \wedge \text{finite} \longrightarrow (f \wedge g) \frown (f1 \wedge g)$

proof –
have 1: $\vdash \text{sba } g \wedge \text{finite} \wedge (f \frown f1) \longrightarrow (g \wedge f) \frown (g \wedge f1)$ **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*:

$\vdash (f \frown f1) \wedge \text{ba } g \longrightarrow (f \wedge g) \frown (f1 \wedge g)$

proof –
have 1: $\vdash \text{ba } g \wedge (f \frown f1) \longrightarrow (g \wedge f) \frown (g \wedge f1)$ **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*:

$\vdash \text{ba } f \wedge g \frown g1 \longrightarrow (f \wedge g) \frown g1$

by (meson *BaAndSChopImport SChopAndB lift-imp-trans*)

lemma *BaAndSChopImportB*:

$\vdash \text{ba } f \wedge g \frown g1 \longrightarrow (f \wedge g) \frown (\text{ba } f \wedge g1)$

proof –
have 1: $\vdash \text{ba } f = \text{ba } (\text{ba } f)$
by (simp add: *BaEqvBaBa*)
have 2: $\vdash \text{ba } (\text{ba } f) \wedge g \frown g1 \longrightarrow g \frown (\text{ba } f \wedge g1)$
by (metis *AndSChopB BaAndSChopImport lift-imp-trans*)
have 3: $\vdash \text{ba } f \wedge g \frown (\text{ba } f \wedge g1) \longrightarrow (f \wedge g) \frown (\text{ba } f \wedge g1)$
by (simp add: *BaAndSChopImportA*)
from 1 2 3 **show** ?thesis **by** fastforce
qed

lemma *SBaImpSBaImpSBaAnd*:

$\vdash \text{sba } h \longrightarrow \text{sba}(g \longrightarrow \text{sba } h \wedge g)$

proof –
have 1: $\vdash \text{sba } h \longrightarrow (g \longrightarrow \text{sba } h \wedge g)$ **by** fastforce
hence 2: $\vdash \text{sba}(\text{sba } h) \longrightarrow \text{sba}(g \longrightarrow \text{sba } h \wedge g)$ **by** (rule *SBaImpSBa*)
have 3: $\vdash \text{sba } h = \text{sba}(\text{sba } h)$ **by** (rule *SBaEqvSBaSBa*)
from 2 3 **show** ?thesis **by** fastforce
qed

lemma *SBaSChopImpSChopSBa*:

$\vdash \text{sba } f \wedge \text{finite} \longrightarrow g \frown g1 \longrightarrow g \frown ((\text{sba } f) \wedge g1)$

proof –

have 1: $\vdash \text{ sba } f \longrightarrow \text{ sba } (g1 \longrightarrow (\text{ sba } f) \wedge g1)$
by (rule *SBaImpSBaImpSBaAnd*)
have 2: $\vdash \text{ sba } (g1 \longrightarrow \text{ sba } f \wedge g1) \wedge \text{ finite } \longrightarrow g \frown g1 \longrightarrow g \frown (\text{ sba } f \wedge g1)$
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*:

$\vdash \Diamond (sfin\ f) \longrightarrow sfin\ f$

proof –

have 1: $\vdash sfin\ f = \#True \frown (f \wedge empty)$

by (rule *SFinEqvTrueSChopAndEmpty*)

hence 2: $\vdash \Diamond (sfin\ f) = \#True \frown (\#True \frown (f \wedge empty))$

by (metis *DiamondSChopdef* inteq-reflection)

have 3: $\vdash \#True \frown (\#True \frown (f \wedge empty)) = (\#True \frown \#True) \frown (f \wedge empty)$

by (rule *SChopAssoc*)

have 4: $\vdash (\#True \frown \#True) \frown (f \wedge empty) \longrightarrow \#True \frown (f \wedge empty)$

using 1 2 3

by (metis *SChopImpDiamond* *TrueEqvTrueSChopTrue* inteq-reflection)

from 1 2 3 4 show ?thesis by fastforce

qed

lemma *SChopSFinImpSFin*:

$\vdash f \frown sfin\ (init\ w) \longrightarrow sfin\ (init\ w)$

proof –

have 1: $\vdash f \frown sfin\ (init\ w) \longrightarrow \Diamond (sfin\ (init\ w))$ by (rule *SChopImpDiamond*)

have 2: $\vdash \Diamond (sfin\ (init\ w)) \longrightarrow sfin\ (init\ w)$ by (rule *DiamondSFinImpSFin*)

from 1 2 show ?thesis using lift-imp-trans by blast

qed

lemma *SFinImpSYieldsSFin*:

$\vdash sfin\ (init\ w) \longrightarrow f\ syields\ (sfin\ (init\ w))$

proof –

have 1: $\vdash f \frown (sfin\ (init\ (\neg w))) \longrightarrow (sfin\ (init\ (\neg w)))$

by (simp add: *SChopSFinImpSFin*)

have 2: $\vdash finite \longrightarrow (\neg (sfin\ (init\ w))) = (sfin\ (init\ (\neg w)))$

using *SFinNotStateEqvNotSFinState* by fastforce

hence 3: $\vdash finite \longrightarrow f \frown (\neg (sfin\ (init\ w))) = f \frown (sfin\ (init\ (\neg w)))$

using *FiniteRightSChopEqvSChop* by blast

have 4: $\vdash f \frown (\neg (sfin\ (init\ w))) \wedge finite \longrightarrow (\neg (sfin\ (init\ w)))$

using 1 2 3 by fastforce

hence 5: $\vdash sfin\ (init\ w) \longrightarrow \neg (f \frown (\neg (sfin\ (init\ w))))$

by (metis (no-types, lifting) *DiamondFin* *SChopImpDiamond* int-simps(32) int-simps(4) inteq-reflection sfin-d-def)

from 5 show ?thesis by (simp add: *syields-d-def*)

qed

lemma *SChopAndSFin*:

$\vdash ((f \frown g) \wedge (sfin\ (init\ w))) = f \frown (g \wedge (sfin\ (init\ w)))$

proof –

have 1: $\vdash sfin\ (init\ w) \longrightarrow f\ syields\ (sfin\ (init\ w))$

by (rule *SFinImpSYieldsSFin*)

have 2: $\vdash (f \frown g) \wedge (sfin\ (init\ w)) \longrightarrow (f \frown g) \wedge f\ syields\ (sfin\ (init\ w))$

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*:

$\vdash (f \frown g \wedge \neg (\text{sfm } (\text{init } w)) \wedge \text{finite}) = f \frown (g \wedge \neg (\text{sfm } (\text{init } w)) \wedge \text{finite})$
proof –
have 1: $\vdash (f \frown g \wedge \text{sfm } (\text{init } (\neg w))) = f \frown (g \wedge \text{sfm } (\text{init } (\neg w)))$
by (*rule SChopAndSFin*)
have 2: $\vdash (\text{sfm } (\text{init } (\neg w)) \wedge \text{finite}) = (\neg (\text{sfm } (\text{init } w)) \wedge \text{finite})$
using *SFinNotStateEqvNotSFinState* **by** *fastforce*
hence 3: $\vdash (g \wedge \text{sfm } (\text{init } (\neg w))) = (g \wedge \neg (\text{sfm } (\text{init } w)) \wedge \text{finite})$
using *DiamondEmptyEqvFinite SChopAndB SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond*
by *fastforce*
hence 4: $\vdash f \frown (g \wedge \text{sfm } (\text{init } (\neg w))) = f \frown (g \wedge \neg (\text{sfm } (\text{init } w)) \wedge \text{finite})$
using *RightSChopEqvSChop* **by** *blast*
from 1 2 4 **show** ?thesis
by (*metis* *DiamondEmptyEqvFinite Prop10 SChopAndB SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond*
inteq-reflection)
qed

lemma *SFinSChopChain*:

$\vdash (((\text{init } w) \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w1))) \frown$
 $((\text{init } w1) \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w2)))$
 $\wedge \text{finite}$
 $\longrightarrow (((\text{init } w) \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w2)))$
proof –
have 01: $\vdash (\text{init } w \wedge \text{finite} \wedge$
 $((\text{init } w \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w1)) \wedge \text{finite}); (\text{init } w1 \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w2))) =$
 $(\text{init } w \wedge \text{empty}) \frown (((\text{init } w \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w1)) \wedge \text{finite});$
 $(\text{init } w1 \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w2)) \wedge \text{finite})$
by (*meson Prop04 SChopAndCommute StateAndEmptySChop*)
have 02: $\vdash (\text{finite} \wedge \text{init } w) = (\text{init } w \wedge \text{empty}) \frown \text{finite}$
by (*metis StateAndEmptySChop inteq-reflection lift-and-com*)
have 03: $\vdash \text{init } w \wedge \text{finite} \wedge$
 $((\text{init } w \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w1)) \wedge \text{finite});$
 $(\text{init } w1 \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w2)) \longrightarrow \text{finite} \wedge \text{init } w$
by *force*
have 04: $\vdash (\text{init } w \wedge \text{finite} \wedge (\text{init } w \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w1)));$
 $(\text{init } w1 \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w2)) =$
 $(\text{init } w \wedge (\text{finite} \wedge (\text{init } w \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w1)));$
 $(\text{init } w1 \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w2)))$
by (*simp* *add: StateAndChop*)
have 05: $\vdash \text{init } w \wedge \text{finite} \wedge ((\text{init } w \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w1)) \wedge \text{finite});$

$$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \longrightarrow$$

$$(init\ w \wedge finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)));$$

$$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2))$$
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: $\vdash (init\ w) \wedge f \wedge finite \longrightarrow sfin\ (init\ w1)$
using *assms* **by** *auto*
have 21: $\vdash (init\ w \wedge f \wedge finite); (f1 \wedge finite) = ((init\ w \wedge f); f1 \wedge finite)$
by (*metis ChopAndFiniteDist StateAndChop StateAndSChop inteq-reflection schop-d-def*)
have 22: $\vdash (init\ w \wedge f \wedge finite) = ((init\ w \wedge f \wedge finite) \wedge sfin\ (init\ w1))$
using 2 *Prop10* **by** *blast*
hence 3: $\vdash ((init\ w) \wedge f) \frown (f1 \wedge finite) \longrightarrow (sfin\ (init\ w1)) \frown (f1 \wedge finite)$
using 21 22
by (*metis (no-types, opaque-lifting) 2 ChopEmpty EmptySChop SChopAssoc StateAndSChop StateAndSChopImpSChopRule int-eq schop-d-def*)
have 4: $\vdash (sfin\ (init\ w1)) \frown (f1 \wedge finite) = \Diamond((init\ w1) \wedge f1 \wedge finite)$
by (*rule SFinSChopEqvDiamond*)
have 5: $\vdash (init\ w1) \wedge f1 \wedge finite \longrightarrow sfin\ (init\ w2)$
using *assms* **by** *auto*
hence 6: $\vdash \Diamond((init\ w1) \wedge f1 \wedge finite) \longrightarrow \Diamond(sfin\ (init\ w2))$
by (*rule DiamondImpDiamond*)
have 7: $\vdash \Diamond(sfin\ (init\ w2)) \longrightarrow sfin\ (init\ w2)$
using *DiamondSFinImpSFin* **by** *blast*
from 1 3 4 6 7 **show** *?thesis* **by** *fastforce*
qed

lemma *SChopRep*:

assumes $\vdash (init\ w) \wedge f \wedge finite \longrightarrow f1 \wedge sfin\ (init\ w1)$
 $\vdash (init\ w1) \wedge g \wedge finite \longrightarrow g1$
shows $\vdash (init\ w) \wedge ((f \frown g) \wedge finite) \longrightarrow (f1 \frown g1)$
proof –
have 1: $\vdash (init\ w) \wedge f \wedge finite \longrightarrow (f1 \wedge sfin\ (init\ w1))$
using *assms* **by** *auto*
hence 2: $\vdash (init\ w) \wedge (f \frown (g \wedge finite)) \longrightarrow (f1 \wedge sfin\ (init\ w1)) \frown (g \wedge finite)$
by (*metis DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty Prop12 SChopSFinExportA SFinEqvTrueSChopAndEmpty StateAndChopImpChopRule StateAndEmptySChop TrueSChopEqvDiamond inteq-reflection schop-d-def*)
have 3: $\vdash (f1 \wedge sfin\ (init\ w1)) \frown (g \wedge finite) = f1 \frown ((init\ w1) \wedge (g \wedge finite))$
using *AndSFinSChopEqvStateAndSChop* **by** *blast*
have 31: $\vdash (init\ w) \wedge ((f \frown g) \wedge finite) \longrightarrow f1 \frown ((init\ w1) \wedge (g \wedge finite))$
using 2 3 **by** (*metis ChopEmpty SChopAssoc inteq-reflection schop-d-def*)
have 4: $\vdash (init\ w1) \wedge (g \wedge finite) \longrightarrow g1$
using *assms* **by** *auto*
hence 5: $\vdash f1 \frown ((init\ w1) \wedge (g \wedge finite)) \longrightarrow f1 \frown g1$
using *RightSChopImpSChop* **by** *blast*
show *?thesis* **using** 31 5 **by** *fastforce*
qed

lemma *SChopRepAndSFin*:

assumes $\vdash (init\ w) \wedge f \wedge finite \longrightarrow f1 \wedge sfin\ (init\ w1)$

$\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1 \wedge \text{sfin } (\text{init } w2)$
shows $\vdash (\text{init } w) \wedge (f \frown g) \wedge \text{finite} \longrightarrow (f1 \frown g1) \wedge \text{sfin } (\text{init } w2)$
proof –
have 1: $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow f1 \wedge \text{sfin } (\text{init } w1)$
using *assms* **by** *auto*
have 2: $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1 \wedge \text{sfin } (\text{init } w2)$
using *assms* **by** *auto*
have 3: $\vdash (\text{init } w) \wedge (f \frown g) \wedge \text{finite} \longrightarrow f1 \frown (g1 \wedge \text{sfin } (\text{init } w2))$
using 1 2 **by** (*rule* *SChopRep*)
have 4: $\vdash f1 \frown (g1 \wedge \text{sfin } (\text{init } w2)) \longrightarrow f1 \frown g1$
by (*rule* *SChopAndA*)
have 5: $\vdash f1 \frown (g1 \wedge \text{sfin } (\text{init } w2)) \longrightarrow f1 \frown \text{sfin } (\text{init } w2)$
by (*rule* *SChopAndB*)
have 6: $\vdash f1 \frown \text{sfin } (\text{init } w2) \longrightarrow \text{sfin } (\text{init } w2)$
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 $\vdash f \wedge \neg g \longrightarrow (more \frown (f \wedge \neg g))$

shows $\vdash f \wedge finite \longrightarrow g$

proof –

have 1: $\vdash f \wedge \neg g \longrightarrow (more \frown (f \wedge \neg g))$ **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 $\vdash f \wedge finite \longrightarrow more \frown f$

shows $\vdash finite \longrightarrow \neg f$

proof –

have 1: $\vdash f \wedge finite \longrightarrow more \frown f$

using *assms* **by** *auto*

hence 11: $\vdash \Diamond (f \wedge finite) \longrightarrow \Diamond (more \frown f)$

using *DiamondImpDiamond* **by** *blast*

have 12: $\vdash \Diamond (more \frown f) = \#True \frown (more \frown f)$

by (*simp add: DiamondSChopdef*)

have 13: $\vdash \#True \frown (more \frown f) = (\#True \frown more) \frown f$

by (*rule SChopAssoc*)

have 14: $\vdash \Diamond (more \frown f) = more \frown f$

using 12 13 **by** (*metis TrueSChopMoreEqvMore inteq-reflection*)

have 2: $\vdash more \frown f = \bigcirc(\Diamond f)$

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 $\vdash (f \wedge \neg g) \wedge finite \longrightarrow (more \frown (f \wedge \neg g))$

shows $\vdash f \wedge finite \longrightarrow g$

proof –

have 1: $\vdash (f \wedge \neg g) \wedge finite \longrightarrow (more \frown (f \wedge \neg g))$ **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 $\vdash f \longrightarrow g \frown f$

$\vdash g \longrightarrow f\text{more}$

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** (*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 $\vdash f \wedge \neg g \longrightarrow h \frown f \wedge \neg (h \frown g)$

$\vdash h \longrightarrow f\text{more}$

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** (*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

7.7 Properties of Halt

lemma *HaltSChopEqv*:

$\vdash ((\text{halt } (init\ w)) \frown f) = (\text{if}_i\ (init\ w)\ \text{then } (f)\ \text{else } (\bigcirc(\text{halt } (init\ w)) \frown f))$
proof –
have 1: $\vdash \text{halt}(init\ w) =$
 $(\text{if}_i\ (init\ w)\ \text{then } \text{empty}\ \text{else } (\bigcirc(\text{halt } (init\ w))))$
by (rule *HaltStateEqvIfStateThenEmptyElseNext*)
hence 2: $\vdash ((\text{halt}(init\ w)) \frown f) =$
 $(\text{if}_i\ (init\ w)\ \text{then } (\text{empty} \frown f)\ \text{else } (\bigcirc(\text{halt } (init\ w)) \frown f))$
by (rule *IfSChopEqvRule*)
have 3: $\vdash \text{empty} \frown f = f$
by (rule *EmptySChop*)
have 4: $\vdash (\bigcirc(\text{halt } (init\ w))) \frown f = \bigcirc(\text{halt } (init\ w) \frown f)$
by (rule *NextSChop*)
from 2 3 4 **show** ?thesis **by** (metis *inteq-reflection*)
qed

lemma *AndHaltSChopImp*:

$\vdash init\ w \wedge (\text{halt } (init\ w) \frown f) \longrightarrow f$
proof –
have 1: $\vdash \text{halt } (init\ w) \frown f = \text{if}_i\ (init\ w)\ \text{then } f\ \text{else } (\bigcirc(\text{halt } (init\ w) \frown f))$
by (rule *HaltSChopEqv*)
have 2: $\vdash init\ w \wedge \text{if}_i\ (init\ w)\ \text{then } f\ \text{else } (\bigcirc(\text{halt } (init\ w) \frown f)) \longrightarrow f$
by (auto simp: *ifthenelse-d-def*)
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *NotAndHaltSChopImpNext*:

$\vdash \neg (init\ w) \wedge (\text{halt } (init\ w) \frown f) \longrightarrow \bigcirc(\text{halt } (init\ w) \frown f)$
proof –
have 1: $\vdash \text{halt } (init\ w) \frown f = \text{if}_i\ (init\ w)\ \text{then } f\ \text{else } (\bigcirc(\text{halt } (init\ w) \frown f))$
by (rule *HaltSChopEqv*)
have 2: $\vdash \neg (init\ w) \wedge \text{if}_i\ (init\ w)\ \text{then } f\ \text{else } (\bigcirc(\text{halt } (init\ w) \frown f)) \longrightarrow$
 $\bigcirc(\text{halt } (init\ w) \frown f)$
by (auto simp: *ifthenelse-d-def*)
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *NotAndHaltSChopImpSkipSYields*:

$\vdash \neg (init\ w) \wedge (\text{halt } (init\ w) \frown f) \longrightarrow \text{skip } syields\ (\text{halt } (init\ w) \frown f)$
proof –
have 1: $\vdash \neg (init\ w) \wedge (\text{halt } (init\ w) \frown f) \longrightarrow \bigcirc(\text{halt } (init\ w) \frown f)$
by (rule *NotAndHaltSChopImpNext*)
have 2: $\vdash \bigcirc(\text{halt } (init\ w) \frown f) \longrightarrow \text{skip } syields\ (\text{halt } (init\ w) \frown f)$
by (rule *NextImpSkipSYields*)
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *SChopAndEmptyEqvSChopAndEmpty*:

$\vdash ((\#True \frown (f \wedge \text{empty})) \wedge g) = (g \frown (f \wedge \text{empty}))$

proof –
have 1: $\vdash (\# \text{True} \neg (f \wedge \text{empty})) \wedge g \longrightarrow g \neg (f \wedge \text{empty})$
by (*simp add: FiniteChopAndEmptyEqvChopAndEmpty int-iffD1 schop-d-def*)
have 2: $\vdash g \neg (f \wedge \text{empty}) \longrightarrow (\# \text{True} \neg (f \wedge \text{empty})) \wedge g$
by (*metis AndSFinEqvSChopAndEmpty Prop12 SFinEqvTrueSChopAndEmpty int-iffD1 inteq-reflection*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *NotSChopSkipEqvFmoreAndNotSChopSkip*:

$\vdash (\neg f) \neg \text{skip} = (f \text{more} \wedge \neg (f \neg \text{skip}))$

proof –

have 1: $\vdash (\neg f) \neg \text{skip} = ((\neg f \wedge \text{finite}); \text{skip})$

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: $\vdash (\text{more} \wedge \neg (f; \text{skip}) \wedge \text{finite}) = (\text{more} \wedge \neg (f \neg \text{skip} \vee (f \wedge \text{inf})) \wedge \text{finite})$

by (*metis ChopEmpty ChopSChopdef inteq-reflection*)

have 8: $\vdash (\text{more} \wedge \neg (f \neg \text{skip} \vee (f \wedge \text{inf})) \wedge \text{finite}) =$
 $(\text{more} \wedge \neg (f \neg \text{skip}) \wedge \neg (f \wedge \text{inf}) \wedge \text{finite})$

by auto

have 9: $\vdash (\neg (f \wedge \text{inf}) \wedge \text{finite}) = \text{finite}$

unfolding *finite-d-def* **by** force

have 10: $\vdash (\text{more} \wedge \neg (f \neg \text{skip}) \wedge \neg (f \wedge \text{inf}) \wedge \text{finite}) =$
 $(\text{more} \wedge \neg (f \neg \text{skip}) \wedge \text{finite})$

using 9 **by** fastforce

have 11: $\vdash (\text{more} \wedge \neg (f \neg \text{skip}) \wedge \text{finite}) = (f \text{more} \wedge \neg (f \neg \text{skip}))$

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*:

$\vdash \text{halt} (\text{init } w) \neg f \wedge \text{finite} \longrightarrow \neg (\text{halt} (\text{init } w) \neg (\neg f))$

proof –

have 1: $\vdash \text{halt} (\text{init } w) \neg f = \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\circ (\text{halt} (\text{init } w) \neg f))$

by (*rule HaltSChopEqv*)

have 2: $\vdash \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\circ (\text{halt} (\text{init } w) \neg f)) \longrightarrow$
 $(((\text{init } w) \longrightarrow f) \wedge (\neg (\text{init } w) \longrightarrow (\circ (\text{halt} (\text{init } w) \neg f))))$

by (*rule IfThenElseImp*)

have 3: $\vdash \text{halt} (\text{init } w) \neg (\neg f) =$

$\text{if}_i (\text{init } w) \text{ then } (\neg f) \text{ else } (\circ (\text{halt} (\text{init } w) \neg (\neg f)))$

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: $\vdash \text{halt } (\text{init } w) \frown f \wedge \text{halt } (\text{init } w) \frown (\neg f) \longrightarrow$
 $(((\text{init } w) \longrightarrow f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt } (\text{init } w) \frown f)))) \wedge$
 $(((\text{init } w) \longrightarrow \neg f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt } (\text{init } w) \frown (\neg f)))))$
using 1 2 3 4 **by** *fastforce*
have 6: $\vdash (((\text{init } w) \longrightarrow f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt } (\text{init } w) \frown f)))) \wedge$
 $(((\text{init } w) \longrightarrow \neg f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt } (\text{init } w) \frown (\neg f))))) \longrightarrow$
 $(\bigcirc(\text{halt } (\text{init } w) \frown f)) \wedge (\bigcirc(\text{halt } (\text{init } w) \frown (\neg f)))$
by *auto*
have 7: $\vdash \text{halt } (\text{init } w) \frown f \wedge \text{halt } (\text{init } w) \frown (\neg f) \longrightarrow$
 $(\bigcirc(\text{halt } (\text{init } w) \frown f)) \wedge (\bigcirc(\text{halt } (\text{init } w) \frown (\neg f)))$
using 5 6 *lift-imp-trans* **by** *blast*
have 8: $\vdash ((\bigcirc(\text{halt } (\text{init } w) \frown f)) \wedge (\bigcirc(\text{halt } (\text{init } w) \frown (\neg f)))) =$
 $\bigcirc(\text{halt } (\text{init } w) \frown f \wedge \text{halt } (\text{init } w) \frown (\neg f))$
using *NextAndEqvNextAndNext* **by** *fastforce*
have 9: $\vdash \text{halt } (\text{init } w) \frown f \wedge \text{halt } (\text{init } w) \frown (\neg f) \longrightarrow$
 $\bigcirc(\text{halt } (\text{init } w) \frown f \wedge \text{halt } (\text{init } w) \frown (\neg f))$
using 7 8 **by** *fastforce*
hence 10: $\vdash \text{finite} \longrightarrow \neg(\text{halt } (\text{init } w) \frown f \wedge \text{halt } (\text{init } w) \frown (\neg f))$
using *NextLoop* **by** *blast*
from 10 **show** ?thesis **by** *auto*
qed

lemma *HaltSChopImpHaltSYields*:

$\vdash \text{halt } (\text{init } w) \frown f \wedge \text{finite} \longrightarrow (\text{halt } (\text{init } w)) \text{ syields } f$

proof –

have 1: $\vdash \text{halt } (\text{init } w) \frown f \wedge \text{finite} \longrightarrow \neg (\text{halt } (\text{init } w) \frown (\neg f))$

by (rule *HaltSChopImpNotHaltSChopNot*)

from 1 **show** ?thesis **by** (simp add: *syields-d-def*)

qed

lemma *HaltSChopAnd*:

$\vdash (\text{halt } (\text{init } w)) \frown f \wedge (\text{halt } (\text{init } w)) \frown g \wedge \text{finite} \longrightarrow (\text{halt } (\text{init } w)) \frown (f \wedge g)$

proof –

have 1: $\vdash (\text{halt } (\text{init } w)) \frown g \wedge \text{finite} \longrightarrow (\text{halt } (\text{init } w)) \text{ syields } g$

by (rule *HaltSChopImpHaltSYields*)

hence 2: $\vdash (\text{halt } (\text{init } w)) \frown f \wedge (\text{halt } (\text{init } w)) \frown g \wedge \text{finite} \longrightarrow$
 $(\text{halt } (\text{init } w)) \frown f \wedge (\text{halt } (\text{init } w)) \text{ syields } g$

by *auto*

have 3: $\vdash (\text{halt } (\text{init } w)) \frown f \wedge (\text{halt } (\text{init } w)) \text{ syields } g \longrightarrow$
 $(\text{halt } (\text{init } w)) \frown (f \wedge g)$

by (rule *SChopAndSYieldsImp*)

from 2 3 **show** ?thesis **by** *fastforce*

qed

lemma *HaltAndSChopAndHaltSChopImpHaltAndSChopAnd*:

$\vdash (\text{halt } (\text{init } w) \wedge f) \frown f1 \wedge (\text{halt } (\text{init } w) \frown g) \wedge \text{finite}$
 $\longrightarrow (\text{halt } (\text{init } w) \wedge f) \frown (f1 \wedge g)$

proof –

have 1: $\vdash f1 \longrightarrow \neg g \vee (f1 \wedge g)$
by *auto*
hence 2: $\vdash (\text{halt } (init\ w) \wedge f) \frown f1 \longrightarrow$
 $(\text{halt } (init\ w) \wedge f) \frown (\neg g) \vee ((\text{halt } (init\ w) \wedge f) \frown (f1 \wedge g))$
by (*rule SChopOrImpRule*)
have 3: $\vdash (\text{halt } (init\ w) \wedge f) \frown (\neg g) \longrightarrow \text{halt } (init\ w) \frown (\neg g)$
by (*rule AndSChopA*)
have 31: $\vdash (\text{halt } (init\ w) \wedge f) \frown f1 \longrightarrow$
 $\text{halt } (init\ w) \frown (\neg g) \vee ((\text{halt } (init\ w) \wedge f) \frown (f1 \wedge g))$
using 23 **by** *fastforce*
have 4: $\vdash \text{halt } (init\ w) \frown g \wedge \text{finite} \longrightarrow \neg (\text{halt } (init\ w) \frown (\neg g))$
by (*rule HaltSChopImpNotHaltSChopNot*)
hence 41: $\vdash (\text{halt } (init\ w) \frown (\neg g)) \wedge \text{finite} \longrightarrow \neg (\text{halt } (init\ w) \frown g)$
by *auto*
have 42: $\vdash (\text{halt } (init\ w) \wedge f) \frown f1 \wedge \text{finite} \longrightarrow$
 $\neg (\text{halt } (init\ w) \frown g) \vee ((\text{halt } (init\ w) \wedge f) \frown (f1 \wedge g))$
using 31 41 **by** *fastforce*
from 42 **show** *?thesis* **by** *auto*
qed

lemma *HaltImpBoxSYields*:

$\vdash (\text{halt } (init\ w)) \frown f \wedge \text{finite} \longrightarrow (\Box(\neg (init\ w))) \text{ syields } ((\text{halt } (init\ w)) \frown f)$

proof –

have 1: $\vdash (\Box(\neg (init\ w))) \frown (\neg (\text{halt } (init\ w) \frown f)) \longrightarrow \text{df } (\Box(\neg (init\ w)))$
by (*rule SChopImpDf*)
have 2: $\vdash \Box(\neg (init\ w)) \longrightarrow \neg (init\ w)$
by (*rule BoxElim*)
hence 3: $\vdash \text{df } (\Box(\neg (init\ w))) \longrightarrow \text{df } (\neg (init\ w))$
by (*rule DfImpDf*)
have 4: $\vdash \text{df } (init\ (\neg w)) = (init\ (\neg w))$
by (*rule DfState*)
have 41: $\vdash (init\ (\neg w)) = (\neg (init\ w))$
using *Initprop(2)* **by** *fastforce*
have 42: $\vdash \text{df } (\neg (init\ w)) = (\neg (init\ w))$
using 4 41 **by** (*metis integ-reflection*)
have 5: $\vdash ((\Box(\neg (init\ w))) \frown (\neg (\text{halt } (init\ w) \frown f))) \longrightarrow \neg (init\ w)$
using 1 2 42 **using** 3 **by** *fastforce*
hence 51: $\vdash (\text{halt } (init\ w) \frown f) \wedge ((\Box(\neg (init\ w))) \frown (\neg (\text{halt } (init\ w) \frown f))) \longrightarrow$
 $(\text{halt } (init\ w) \frown f) \wedge \neg (init\ w)$
by *fastforce*
have 6: $\vdash \text{halt } (init\ w) \frown f = \text{if}_i (init\ w) \text{ then } f \text{ else } (\Box(\text{halt } (init\ w) \frown f))$
by (*rule HaltSChopEqv*)
hence 61: $\vdash (\text{halt } (init\ w) \frown f \wedge \neg (init\ w)) =$
 $((\text{if}_i (init\ w) \text{ then } f \text{ else } (\Box(\text{halt } (init\ w) \frown f))) \wedge \neg (init\ w))$
using 6 **by** *auto*
have 62: $\vdash (\text{if}_i (init\ w) \text{ then } f \text{ else } (\Box(\text{halt } (init\ w) \frown f))) \wedge$
 $\neg (init\ w) \longrightarrow (\Box(\text{halt } (init\ w) \frown f))$
by (*auto simp: ifthenelse-d-def*)
have 63: $\vdash \text{halt } (init\ w) \frown f \wedge \neg (init\ w) \longrightarrow (\Box(\text{halt } (init\ w) \frown f))$

using 61 62 by fastforce
 have 7: $\vdash (\text{halt } (\text{init } w) \frown f) \wedge (\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)) \longrightarrow$
 $\quad \circ((\text{halt } (\text{init } w)) \frown f)$
 using 51 63 using lift-imp-trans by blast
 have 8: $\vdash \Box(\neg (\text{init } w)) \longrightarrow \text{empty} \vee \circ(\Box(\neg (\text{init } w)))$
 by (metis BoxImpYields WeakNextBoxImpMoreYields WnextEqvEmptyOrNext fmore-d-def int-eq)
 hence 9: $\vdash ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))) \longrightarrow$
 $\quad \neg (\text{halt } (\text{init } w) \frown f) \vee \circ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)))$
 by (rule EmptyOrNextSChopImpRule)
 hence 10: $\vdash ((\text{halt } (\text{init } w)) \frown f) \wedge (\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)) \longrightarrow$
 $\quad \circ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)))$
 by fastforce
 have 11: $\vdash (\text{halt } (\text{init } w)) \frown f \wedge (\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)) \longrightarrow$
 $\quad \circ((\text{halt } (\text{init } w)) \frown f) \wedge \circ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)))$
 using 7 10 by fastforce
 have 12: $\vdash \circ((\text{halt } (\text{init } w)) \frown f) \wedge \circ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)))$
 $\quad \longrightarrow \circ(((\text{halt } (\text{init } w)) \frown f) \wedge ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))))$
 using NextAndEqvNextAndNext by fastforce
 have 13: $\vdash (\text{halt } (\text{init } w)) \frown f \wedge (\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)) \longrightarrow$
 $\quad \circ(((\text{halt } (\text{init } w)) \frown f) \wedge ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))))$
 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

7.8 Properties of Groups of strong chops

lemma *NestedSChopImpSChop*:

assumes $\vdash \text{init } w \wedge f \longrightarrow g \frown (\text{init } w1 \wedge f1)$
 $\quad \vdash \text{init } w1 \wedge f1 \longrightarrow g1 \frown (\text{init } w2 \wedge f2)$
 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

8 Finite and Infinite ITL theorems using Weak Chop

theory *Chopstar*
 imports

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.

8.1 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 \rightleftharpoons *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 \rightleftharpoons *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

$-len-d \quad :: nat \Rightarrow lift \quad ((len -) [88] 87)$
 $-wpowerstar-d \quad :: lift \Rightarrow lift \quad ((wpowerstar -) [85] 85)$
 $-fpowerstar-d \quad :: lift \Rightarrow lift \quad ((fpowerstar -) [85] 85)$

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 $while\text{-}d :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a formula$
where $while\text{-}d F G \equiv LIFT((F \wedge G)^* \wedge (fin ((\neg F))))$

syntax

$\text{-}while\text{-}d :: [lift, lift] \Rightarrow lift ((while - do -) [88,88] 87)$

syntax (ASCII)

$\text{-}while\text{-}d :: [lift, lift] \Rightarrow lift ((while - do -) [88,88] 87)$

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

$\text{-}swhile\text{-}d :: [lift, lift] \Rightarrow lift ((swhile - do -) [88,88] 87)$
 $\text{-}repeat\text{-}d :: [lift, lift] \Rightarrow lift ((repeat - until -) [88,88] 87)$

syntax (ASCII)

$\text{-}swhile\text{-}d :: [lift, lift] \Rightarrow lift ((swhile - do -) [88,88] 87)$
 $\text{-}repeat\text{-}d :: [lift, lift] \Rightarrow lift ((repeat - until -) [88,88] 87)$

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

$\text{-}srepeat\text{-}d :: [lift, lift] \Rightarrow lift ((srepeat - until -) [88,88] 87)$

syntax (ASCII)

$\text{-}srepeat\text{-}d :: [lift, lift] \Rightarrow lift ((srepeat - until -) [88,88] 87)$

translations

$\text{-}srepeat\text{-}d \Rightarrow CONST srepeat\text{-}d$

definition $aschopstar\text{-}d :: ('a::world) formula \Rightarrow 'a formula$
where $aschopstar\text{-}d F \equiv$

$\lambda s. (\exists (l:: nat nellist).$
 $(n nth\ l\ 0) = 0 \wedge n finite\ l \wedge n idx\ l \wedge$
 $(enat\ (n last\ l)) = (n length\ s) \wedge n finite\ s \wedge$

$$\begin{aligned}
& (\forall (i::nat) . ((enat\ i) < (nlength\ l)) \longrightarrow \\
& \quad ((nsubn\ s\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))\) \models F)) \\
&)
\end{aligned}$$

syntax

-aschopstar-d :: *lift* \Rightarrow *lift* ((*aschopstar -*) [85] 85)

syntax (*ASCII*)

-aschopstar-d :: *lift* \Rightarrow *lift* ((*aschopstar -*) [85] 85)

translations

-aschopstar-d \Rightarrow *CONST aschopstar-d*

lemma *FChopSem-var* [*mono*]:

$(w \models f;g) =$
 $((\exists n. enat\ n \leq nlength\ w \wedge f\ ((ntaken\ n\ w)) \wedge g\ (ndropn\ n\ w)) \vee$
 $(\neg nfinite\ w \wedge f\ w)))$

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 :: *lift* \Rightarrow *lift* ((*istar -*) [85] 85)

syntax (*ASCII*)

-istar-d :: *lift* \Rightarrow *lift* ((*istar -*) [85] 85)

translations

-istar-d \Rightarrow *CONST istar-d*

8.2 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 \wedge 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 \wedge 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 $(\exists (l:: \text{nat nellist}).$

$(\text{nlength } l) = (\text{Suc } n) \wedge \text{nfinite } l \wedge \text{nidx } l \wedge (\text{nnth } l 0) = 0 \wedge$

$(\text{enat } (\text{nlast } l)) = (\text{nlength } \sigma) \wedge \text{nfinite } \sigma \wedge$

$(\forall (i::\text{nat}). i < (\text{nlength } l) \longrightarrow$

$((\text{nsbn } \sigma (\text{nnth } l i) (\text{nnth } l (\text{Suc } i))) \models f)$

$)$

$)$

shows $(\exists k. 0 \leq k \wedge k \leq \text{nlength } \sigma \wedge 0 < k \wedge (\text{nsbn } \sigma 0 k \models f) \wedge$

$(\exists ls. \text{nfinite } ls \wedge (\text{nlength } ls) = n \wedge \text{nidx } ls \wedge (\text{nnth } ls 0) = 0 \wedge$

$(\text{enat } (\text{nlast } ls)) = (\text{nlength } (\text{ndropn } k \sigma)) \wedge \text{nfinite}(\text{ndropn } k \sigma) \wedge$

$(\forall (i::\text{nat}). i < (\text{nlength } ls) \longrightarrow$

$((\text{nsbn } (\text{ndropn } k \sigma) (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) \models f)$

$))$

$)$

proof –

obtain l **where** $1: (\text{nlength } l) = (\text{Suc } n) \wedge \text{nfinite } l \wedge \text{nidx } l \wedge (\text{nnth } l 0) = 0 \wedge$

$(\text{enat } (\text{nlast } l)) = (\text{nlength } \sigma) \wedge \text{nfinite } \sigma \wedge$

$(\forall (i::\text{nat}). i < (\text{nlength } l) \longrightarrow$

$((\text{nsbn } \sigma (\text{nnth } l i) (\text{nnth } l (\text{Suc } i))) \models f)$

$)$

using *assms* **by** *auto*

have $2: \text{nlength } l > 0$

using 1 **by** (*simp add: enat-0-iff(1)*)

have $3: l = \text{NCons } (\text{nnth } l 0) (\text{ndropn } 1 l)$

using 2 **by** (*simp add: ndropn-Suc-conv-ndropn zero-enat-def*)

have $4: \text{nlength } (\text{ndropn } 1 l) = n$

by (*simp add: 1*)

have $5: 0 \leq (\text{nnth } l 0)$

by *simp*

have $6: (\text{nnth } l 0) \leq \text{nlength } \sigma$

using 1 **using** *i0-lb zero-enat-def* **by** *presburger*

have $7: (\text{nnth } l 0) = 0$

using 1 **by** *blast*

have $71: (\text{nnth } (\text{ndropn } 1 l) 0) = (\text{nnth } l 1)$

by *auto*

have $8: \text{nnth } l 0 < \text{nnth } (\text{ndropn } 1 l) 0$

by (*metis 1 71 One-nat-def eSuc-enat enat-ord-simps(2) ileI1 nidx-gr-first zero-less-Suc*)

have $9: \text{nidx } (\text{ndropn } 1 l)$

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 $10: (\text{enat } (\text{nlast } (\text{ndropn } 1 l))) = (\text{nlength } \sigma)$

using $1 3$ **by** (*metis nlast-NCons*)

have $101: \bigwedge j. j \leq \text{nlength } (\text{ndropn } 1 l) \longrightarrow$

$\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l 1)) (\text{ndropn } 1 l)) j =$

$(\text{nnth } l (\text{Suc } j)) - (\text{nnth } l 1)$

by *simp*

have $102: \bigwedge j. j \leq \text{nlength } (\text{ndropn } 1 l) \longrightarrow$

$(\text{nnth } l 1) \leq (\text{nnth } l (\text{Suc } j))$

by (*simp add: 1 nidx-less-eq*)

have 103: $\bigwedge j. j < \text{nlength } (\text{ndropn } 1 \ l) \longrightarrow$
 $(\text{nnth } l \ (\text{Suc } j)) - (\text{nnth } l \ 1) <$
 $(\text{nnth } l \ (\text{Suc } (\text{Suc } j))) - (\text{nnth } l \ 1)$
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* σ 0 (*nnth* *l* 1) $\models f$)
by (*metis 1 One-nat-def enat-ord-simps*(2) *zero-less-Suc*)
have 15: ($\forall (i::\text{nat}). i < \text{nlength } (\text{ndropn } 1 \ l) \longrightarrow$
 $(\text{nsubn } \sigma \ (\text{nnth } (\text{ndropn } 1 \ l) \ i) \ (\text{nnth } (\text{ndropn } 1 \ l) \ (\text{Suc } i)) \models f)$)
using 1 3 *eSuc-enat ileI1 illess-Suc-eq ndropn-nnth nlength-NCons plus-1-eq-Suc* **by** *metis*

have 16: ($\forall (i::\text{nat}). i < \text{nlength } (\text{nmap } (\lambda x. x - (\text{nnth } l \ 1)) (\text{ndropn } 1 \ l)) \longrightarrow$
 $(\text{nsubn } \sigma \ ((\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l \ 1)) (\text{ndropn } 1 \ l)) \ i) + (\text{nnth } l \ 1))$
 $((\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l \ 1)) (\text{ndropn } 1 \ l)) (\text{Suc } i)) + (\text{nnth } l \ 1)) \models f)$)
using 102 12 15 4 **by** *force*
have 17: ($\forall (i::\text{nat}). i < \text{nlength } (\text{nmap } (\lambda x. x - (\text{nnth } l \ 1)) (\text{ndropn } 1 \ l)) \longrightarrow$
 $((\text{nsubn } (\text{ndropn } (\text{nnth } l \ 1) \ \sigma)$
 $(\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l \ 1)) (\text{ndropn } 1 \ l)) \ i)$
 $(\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } l \ 1)) (\text{ndropn } 1 \ l)) (\text{Suc } i))) \models f)$)
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: ($\exists \text{ ls. } \text{nfinite } \text{ls} \wedge (\text{nlength } \text{ls}) = n \wedge \text{nidx } \text{ls} \wedge (\text{nnth } \text{ls } 0) = 0 \wedge$
 $(\text{enat } (\text{nlast } \text{ls})) = (\text{nlength } (\text{ndropn } (\text{nnth } l \ 1) \ \sigma)) \wedge \text{nfinite}(\text{ndropn } (\text{nnth } l \ 1) \ \sigma) \wedge$
 $(\forall (i::\text{nat}). i < (\text{nlength } \text{ls}) \longrightarrow$
 $((\text{nsubn } (\text{ndropn } (\text{nnth } l \ 1) \ \sigma) \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i))) \models f)$
 $)$)
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 ($\exists k. 0 \leq k \wedge k \leq \text{nlength } \sigma \wedge 0 < k \wedge$
 $(\text{nsubn } \sigma \ 0 \ k \models f) \wedge$
 $(\exists \text{ ls. } \text{nfinite } \text{ls} \wedge (\text{nlength } \text{ls}) = n \wedge \text{nidx } \text{ls} \wedge (\text{nnth } \text{ls } 0) = 0 \wedge$
 $(\text{enat } (\text{nlast } \text{ls})) = (\text{nlength } (\text{ndropn } k \ \sigma)) \wedge \text{nfinite}(\text{ndropn } k \ \sigma) \wedge$

```

    (∀ (i::nat). i < (nlength ls) →
      ((nsubn (ndropn k σ) (nnth ls i) (nnth ls (Suc i)) ) ⊨ f)
    ))
  )
shows (∃ (l:: nat nellist).
  (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)
    )
  )
proof –
obtain k where 1: 0 ≤ k ∧ k ≤ nlength σ ∧ k > 0 ∧
  (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)
    ))
  )
using assms by auto
have 2: (∃ 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)
  ))
  )
using 1 by auto
obtain ls where 3: 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)
  )
  )
using 2 by auto
have 4: nidx (nmap (λx. x + k) ls)
  using 3
  by (simp add: nidx-expand)
have 41: ∧j. j ≤ nlength (nmap (λx. x + k) ls) →
  0 < (nnth (nmap (λx. x + k) ls) j)
  by (simp add: 1)
have 5: nidx (NCons 0 (nmap (λx. x + k) ls))

  using 3 4 nidx-expand[of ls] nidx-expand[of (nmap (λx. x + k) ls)]
  nidx-expand[of (NCons 0 (nmap (λx. x + k) 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: (nlength ((NCons 0 (nmap (λx. x + k) ls)))) = (Suc n)
  by (simp add: 3 eSuc-enat)
have 7: (enat (nlast ((NCons 0 (nmap (λx. x + k) ls))))) = (nlength σ)
  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: (nsubn σ 0 k ⊨ f)

```

using 1 **by** *auto*

have 9: $(\forall (i::nat). \ i < (nlength\ ls) \longrightarrow$
 $((nsubn\ (ndropn\ k\ \sigma)\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f)$
 $)$

using 3 **by** *auto*

have 10: $(\forall (i::nat). \ i < (nlength\ ls) \longrightarrow$
 $((nsubn\ \sigma\ ((nnth\ ls\ (i))+k)\ ((nnth\ ls\ ((i)+1))+k)) \models f)$
 $)$

by (*metis* 3 *Suc-ile-eq* *add.commute* *add.right-neutral* *nidx-less* *nsubn-ndropn* *plus-1-eq-Suc*)

have 11: $(\forall (i::nat). \ i < nlength\ (((nmap\ (\lambda x. \ x + k)\ ls)))) \longrightarrow$
 $(nsubn\ \sigma\ (nnth\ (((nmap\ (\lambda x. \ x + k)\ ls))))\ i)\ (nnth\ (((nmap\ (\lambda x. \ x + k)\ ls))))\ (Suc\ i) \models f)$
by (*metis* 10 *add.commute* *eSuc-enat* *ileI1* *nlength-nmap* *nnth-nmap* *order-less-imp-le* *plus-1-eq-Suc*)

have 12: $(\forall i. \ (0 < i \wedge i < 1 + (nlength\ (nmap\ (\lambda x. \ x + k)\ ls))) \longrightarrow$
 $((nsubn\ \sigma\ ((nnth\ (nmap\ (\lambda x. \ x + k)\ ls)\ (i-1)))$
 $((nnth\ (nmap\ (\lambda x. \ x + k)\ ls)\ ((i)))) \models f)$
 $)$

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: $(\forall (i::nat). \ i < nlength\ ((NCons\ 0\ (nmap\ (\lambda x. \ x + k)\ ls)))) \longrightarrow$
 $(nsubn\ \sigma\ (nnth\ (((NCons\ 0\ (nmap\ (\lambda x. \ x + k)\ ls))))\ i)$
 $(nnth\ (((NCons\ 0\ (nmap\ (\lambda x. \ x + k)\ ls))))\ (Suc\ i) \models f)$

using 12

using 1 11 3 6 *less-Suc-eq-0-disj* **by** *auto*

have 14: $(nnth\ (NCons\ 0\ (nmap\ (\lambda x. \ x + k)\ ls))\ 0) = 0$
by *simp*

have 15: $(nnth\ (NCons\ 0\ (nmap\ (\lambda x. \ x + k)\ ls))\ 1) = k$
using 4 **by** (*simp* *add*: 3)

have 16: $(nsubn\ \sigma\ (nnth\ (NCons\ 0\ (nmap\ (\lambda x. \ x + k)\ ls))\ 0)\ k \models f)$
by (*simp* *add*: 8)

have 17: $nfinite\ (NCons\ 0\ (nmap\ (\lambda x. \ x + k)\ ls))$
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-eqv-sem*:

$(\ (\sigma \models (\exists n. \ (wpower\ ((f \wedge more) \wedge finite)\ n))) =$
 $((\sigma \models empty) \vee (\ (\sigma \models ((f \wedge more) \wedge finite); (\exists n. \ (wpower\ ((f \wedge more) \wedge finite)\ n))))))$

using *wpowersem* **by** *fastforce*

lemma *aschopstar-eqv-wpower-chop-help*:

$(\ \sigma \models wpower\ ((f \wedge more) \wedge finite)\ n) =$
 $(\exists (l::nat\ nellist).$

$(nlength\ l) = n) \wedge nfinite\ l \wedge nidx\ l \wedge (nnth\ l\ 0) = 0 \wedge$
 $(enat\ (nlast\ l)) = (nlength\ \sigma) \wedge nfinite\ \sigma \wedge$

$$\begin{aligned}
& (\forall (i::nat). i < (nlength\ l) \longrightarrow \\
& \quad ((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models f) \\
&)
\end{aligned}$$

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 ($\sigma \models wpower\ ((f \wedge more) \wedge finite)\ (Suc\ n) =$

$(\sigma \models (((f \wedge more) \wedge finite);(wpower\ ((f \wedge more) \wedge finite)\ n)))$)

by *simp*

also have ... =

$(\exists\ k. 0 \leq k \wedge k \leq nlength\ (\sigma) \wedge k > 0 \wedge$

$(ntaken\ k\ (\sigma) \models f) \wedge$

$(ndropn\ k\ (\sigma) \models wpower\ ((f \wedge more) \wedge finite)\ n)$

)

using *enat-0-iff(1) le-zero-eq* **by** (*auto simp add: more-defs chop-defs finite-defs*)

also have ... =

$(\exists\ k. 0 \leq k \wedge k \leq nlength\ (\sigma) \wedge k > 0 \wedge$

$(nsubn\ \sigma\ 0\ k \models f) \wedge$

$(ndropn\ k\ (\sigma) \models wpower\ ((f \wedge more) \wedge finite)\ n)$

)

by (*metis One-nat-def Suc-diff-1 Suc-diff-Suc diff-add ndropn-0 ntaken-ndropn*)

also have ... =

$(\exists\ k. 0 \leq k \wedge k \leq nlength\ (\sigma) \wedge k > 0 \wedge$

$(nsubn\ \sigma\ 0\ k \models f) \wedge$

$(\exists\ l. nfinite\ l \wedge (nlength\ l) = n \wedge nidx\ l \wedge (nnth\ l\ 0) = 0 \wedge$

$(enat\ (nlast\ l)) = (nlength\ (ndropn\ k\ \sigma)) \wedge nfinite\ (ndropn\ k\ \sigma) \wedge$

$(\forall (i::nat). i < (nlength\ l) \longrightarrow$

$((nsubn\ (ndropn\ k\ \sigma)\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models f)$

))

)

using *Suc.hyps* **by** *blast*

also have ... =

$(\exists\ (l::nat\ nellist).$

$(nlength\ l) = (Suc\ n) \wedge nfinite\ l \wedge nidx\ l \wedge (nnth\ l\ 0) = 0 \wedge$

$(enat\ (nlast\ l)) = (nlength\ \sigma) \wedge nfinite\ \sigma \wedge$

$(\forall (i::nat). i < (nlength\ l) \longrightarrow$

$((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models f)$

)

)

```

using aschopstar-wpower-chain-a [of  $n \ \sigma \ f$ ]
      aschopstar-wpower-chain-b [of  $\sigma \ f \ n$ ] by auto
finally show  $(\sigma \models \text{wpower } ((f \wedge \text{more}) \wedge \text{finite}) (\text{Suc } n)) =$ 
       $(\exists (l:: \text{nat nellist}).$ 
       $(\text{nlength } l) = (\text{Suc } n) \wedge \text{nfinite } l \wedge \text{nidx } l \wedge (\text{nnth } l \ 0) = 0 \wedge$ 
       $(\text{enat } (\text{nlast } l)) = (\text{nlength } \sigma) \wedge \text{nfinite } \sigma \wedge$ 
       $(\forall (i:: \text{nat}). i < (\text{nlength } l) \longrightarrow$ 
       $((\text{nsubn } \sigma (\text{nnth } l \ i) (\text{nnth } l (\text{Suc } i))) \models f)$ 
       $)$ 
       $)$  .

```

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*:

```

 $\vdash (\text{aschopstar } f) = (\text{schopstar } f)$ 
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*:

$(\sigma \models f^* = (\text{empty} \vee (f \wedge \text{more}); f^*))$

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*:

$(w \models (\text{istar } F)) \Longrightarrow$

$((w \models \text{empty} \vee (F; \text{istar } F)) \Longrightarrow P) \Longrightarrow P$

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 $(\bigwedge s. (s \models \text{empty}) \Longrightarrow P s)$

$(\bigwedge s. (s \models (F;((\text{istar } F) \wedge P))) \Longrightarrow P s)$

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 $(\bigwedge s. (s \models \text{empty}) \Longrightarrow P s)$

$(\bigwedge s. (s \models (F;P)) \Longrightarrow P s)$

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 $\vdash empty \vee f; g \longrightarrow g$

shows $\vdash (istar\ f) \longrightarrow g$

using *assms* *IStarWeakInductSem*[of g] **unfolding** *Valid-def*

by (*metis* *intensional-rews*(3))

8.3 Helper lemmas

lemma *AndEmptyChopAndEmptyEqvAndEmpty*:

$\vdash (f \wedge empty);(f \wedge empty) = (f \wedge empty)$

proof –

have 1: $\vdash (f \wedge empty);(f \wedge empty) \longrightarrow (f \wedge empty)$

by (*metis* *ChopAndB* *ChopEmpty* *int-eq*)

have 2: $\vdash (f \wedge empty) \longrightarrow (f \wedge empty);(f \wedge empty)$

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

8.4 Properties of Chopstar and Chopplus

lemma *FPowerstardef*:

$\vdash fpowerstar\ f = (\exists\ n. fpower\ f\ n)$

by (*simp* *add*: *fpowerstar-d-def*)

lemma *Powerstardef*:

$\vdash powerstar\ f = (fpowerstar\ f);(empty \vee (f \wedge inf))$

by (*simp* *add*: *fpowerstar-d-def* *powerstar-d-def*)

lemma *Chopstardef*:

$\vdash chopstar\ f = powerstar\ (f \wedge more)$

by (*simp* *add*: *chopstar-d-def*)

lemma *SChopstardef*:

$\vdash schopstar\ f = fpowerstar\ (f \wedge more)$

by (*simp* *add*: *s chopstar-d-def*)

lemma *WPowerEqRule*:

assumes $\vdash f = g$

shows $\vdash wpower\ f\ n = wpower\ g\ n$

using *assms*

by (*metis* *int-eq* *int-simps*(20))

lemma *WPowerCommute*:

```

  ⊢ (f) ; (wpower f n) = (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*:

```

  ⊢ (f ∧ finite) ; fpower f n = fpower f n; (f ∧ finite)
unfolding fpower-d-def using WPowerCommute[of LIFT (f ∧ finite)]
by blast

```

lemma *WPowerChopInductL*:

```

  assumes ⊢ g ∨ f; h ⟶ h
  shows ⊢ (wpower f n); g ⟶ 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 ⊢ g ∨ f; h ⟶ h
  shows ⊢ (fpower f n); g ⟶ h
unfolding fpower-d-def using assms WPowerChopInductL[of g LIFT (f ∧ finite) h n]
using AndChopA by fastforce

```

lemma *FPowerChopInductFiniteL*:

```

  assumes ⊢ g ∨ (f ∧ finite); h ⟶ h
  shows ⊢ (fpower f n); g ⟶ h
unfolding fpower-d-def using assms WPowerChopInductL[of g LIFT (f ∧ finite) h n]
by blast

```

lemma *WPowerChopInductMoreL*:

```

  assumes ⊢ g ∨ (f ∧ more); h ⟶ h
  shows ⊢ (wpower f n); g ⟶ 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 $\vdash (f \wedge \text{finite}); (\text{fpower } f \ n; ((f \wedge \text{inf}); g)) = (\text{fpower } f \ (\text{Suc } n); (f \wedge \text{inf}); g)$
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 $\vdash ((\text{fpower } f \ n); ((f \wedge \text{more}) \wedge \text{inf}); g) \longrightarrow h$
using *FPowerChopInductInfL*
by (*metis AndMoreAndInfEqvAndInf assms inteq-reflection*)

lemma *WPowerChopInductR*:
assumes $\vdash g \vee h; f \longrightarrow h$
shows $\vdash g; (\text{wpower } f \ n) \longrightarrow h$
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 $\vdash g; (\text{fpower } f \ n) \longrightarrow h$
unfolding *fpower-d-def* **using** *assms WPowerChopInductR[of g h LIFT (f ∧ finite) n]*
using *ChopAndA* **by** *fastforce*

lemma *FpowerChopInductInfR*:
assumes $\vdash g \vee h; f \longrightarrow h$
shows $\vdash g; ((\text{fpower } f \ n); (f \wedge \text{inf})) \longrightarrow h$
using *assms*
by (*metis ChopAndA ChopAssoc FPowerChopInductR LeftChopImpChop Prop05 int-iffD1 lift-imp-trans*)

lemma *WPowerStarCommute*:
 $\vdash f; (\exists n. \text{wpower } f \ n) = (\exists n. \text{wpower } f \ n); f$
proof –
have 1: $\vdash f; (\exists n. \text{wpower } f \ n) = (\exists n. f; \text{wpower } f \ n)$
by (*metis ChopExist Prop11*)
have 2: $\vdash (\exists n. f; \text{wpower } f \ n) = (\exists n. (\text{wpower } f \ n); f)$
using *WPowerCommute* **by** (*metis ExEqvRule*)

have $\exists: \vdash (\exists n. (wpower\ f\ n); f) = (\exists n. (wpower\ f\ n)); f$
by (*simp add: ExistChop*)
from 1 2 3 **show** ?thesis **by** fastforce
qed

lemma *FPowerStarCommute*:
 $\vdash (f \wedge finite); (\exists n. fpower\ f\ n) = (\exists n. fpower\ f\ n); (f \wedge finite)$
unfolding *fpower-d-def* **using** *WPowerStarCommute*[of *LIFT* ($f \wedge finite$)]
by blast

lemma *WPowerSucAndEmptyEqvAndEmpty*:
 $\vdash (wpower\ (f \wedge empty)\ (Suc\ n)) = (f \wedge empty)$
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*:
 $\vdash (fpower\ (f \wedge empty)\ (Suc\ n)) = (f \wedge empty)$
unfolding *fpower-d-def* **using** *WPowerSucAndEmptyEqvAndEmpty*[of *LIFT* ($f \wedge finite$) *n*]
WPowerEqRule[of *LIFT* ($(f \wedge empty) \wedge finite$) *LIFT* ($(f \wedge finite) \wedge empty$) (*Suc n*)]
by (*meson EmptyImpFinite Prop01 Prop04 Prop05 Prop10 WPowerEqRule WPowerSucAndEmptyEqvAndEmpty*)

lemma *WPowerOr*:
 $\vdash (wpower\ (f \vee g)\ (Suc\ n)) = ((f; wpower\ (f \vee g)\ n) \vee (g; wpower\ (f \vee g)\ n))$
by (*simp add: OrChopEqv*)

lemma *FPowerOr*:
 $\vdash (fpower\ (f \vee g)\ (Suc\ n)) = ((f \wedge finite); fpower\ (f \vee g)\ n) \vee ((g \wedge finite); fpower\ (f \vee g)\ n)$
by (*simp add: FiniteOr OrChopEqvRule fpower-d-def*)

lemma *WPowerEmptyOrMore*:
 $\vdash (wpower\ ((f \wedge empty) \vee (f \wedge more))\ (Suc\ n)) = ((f \wedge empty); (wpower\ ((f \wedge empty) \vee (f \wedge more))\ n) \vee (f \wedge more); (wpower\ ((f \wedge empty) \vee (f \wedge more))\ n))$
using *WPowerOr* **by** blast

lemma *FPowerEmptyOrMore*:
 $\vdash (fpower\ ((f \wedge empty) \vee (f \wedge more))\ (Suc\ n)) = ((f \wedge empty); (fpower\ ((f \wedge empty) \vee (f \wedge more))\ n) \vee (f \wedge fmore); (fpower\ ((f \wedge empty) \vee (f \wedge more))\ n))$
using *FPowerOr*[of *LIFT* ($f \wedge empty$) *LIFT* ($f \wedge more$) *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 inteq-reflection*)
qed

lemma *WPowerstarEqv* :
 $\vdash (\text{wpowerstar } f) = (\text{empty} \vee f; (\text{wpowerstar } f))$
using *WPowerstarEqvSem* **by** *blast*

lemma *FPowerstarEqv* :
 $\vdash (\text{fpowerstar } f) = (\text{empty} \vee (f \wedge \text{finite}); (\text{fpowerstar } f))$
by (*simp add: fpowersem fpowerstar-d-def*)

lemma *SChopstarEqv* :
 $\vdash (\text{schopstar } f) = (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{finite}); (\text{schopstar } f))$
by (*simp add: FPowerstarEqv chopstar-d-def*)

lemma *WPowerstar-more-absorb*:
 $\vdash (\text{wpowerstar } (f \wedge \text{more})) = (\text{wpowerstar } f)$
proof –
have 1: $\vdash (\text{wpowerstar } (f \wedge \text{more})) \longrightarrow (\text{wpowerstar } f)$
using *WPowerstarInductL*[*of LIFT empty LIFT f ∧ more LIFT (wpowerstar f)*]
by (*metis AndChopA ChopEmpty Prop02 Prop05 WPowerChopInductL WPowerstarEqv int-iffD2 inteq-reflection wpow-0*)
have 2: $\vdash \text{empty} \longrightarrow \text{wpowerstar } (f \wedge \text{more})$
using *WPowerstarEqv*[*of LIFT f ∧ more*] **by** *fastforce*
have 20: $\vdash (f \wedge \text{more}); \text{wpowerstar } (f \wedge \text{more}) \longrightarrow \text{wpowerstar } (f \wedge \text{more})$
by (*meson Prop03 WPowerstarEqv*)
have 21: $\vdash (f \wedge \text{empty}); \text{wpowerstar } (f \wedge \text{more}) \longrightarrow \text{wpowerstar } (f \wedge \text{more})$
by (*metis AndChopB EmptyChop int-eq*)
have 22: $\vdash ((f \wedge \text{more}) \vee (f \wedge \text{empty})); \text{wpowerstar } (f \wedge \text{more}) =$
 $((f \wedge \text{more}); \text{wpowerstar } (f \wedge \text{more}) \vee (f \wedge \text{empty}); \text{wpowerstar } (f \wedge \text{more}))$
by (*simp add: OrChopEqv*)
have 23: $\vdash ((f \wedge \text{more}) \vee (f \wedge \text{empty})) = f$
unfolding *empty-d-def* **by** *fastforce*
have 3: $\vdash f; \text{wpowerstar } (f \wedge \text{more}) \longrightarrow \text{wpowerstar } (f \wedge \text{more})$
by (*metis 20 21 22 23 Prop02 inteq-reflection*)
have 4: $\vdash \text{empty} \vee f; \text{wpowerstar } (f \wedge \text{more}) \longrightarrow \text{wpowerstar } (f \wedge \text{more})$
using 2 3 **by** *fastforce*
have 5: $\vdash (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } (f \wedge \text{more}))$
using 4 *WPowerstarInductL*[*of LIFT empty f LIFT (wpowerstar (f ∧ more))*]
by (*metis ChopEmpty inteq-reflection*)
show *?thesis* **using** 1 5 **by** *fastforce*
qed

lemma *FPowerstar-more-absorb*:

$\vdash (f\text{powerstar } (f \wedge \text{more})) = (f\text{powerstar } f)$
proof –
have 1: $\vdash ((f \wedge \text{more}) \wedge \text{finite}) = ((f \wedge \text{finite}) \wedge \text{more})$
by *fastforce*
show *?thesis*
using *FPowerstar-WPowerstar[of f] FPowerstar-WPowerstar[of LIFT (f \wedge more)]*
WPowerstar-more-absorb[of LIFT (f \wedge 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

8.5 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*:

$\vdash (f) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f)$

by (*metis SChopstar-WPowerstar WPowerstar-1L inteq-reflection chop-d-def*)

lemma *SChopstarMore-1L*:

$\vdash (f \wedge \text{more}) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f)$

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 chop-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 chopstar-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 chop-d-def chopstar-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 chop-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 *more g*]
SChopstar-and-more[of *f*] **by** (*metis int-eq*)

lemma *WPowerstar-induct-lvar-eq*:
assumes $\vdash f;g = g$
shows $\vdash (wpowerstar\ f);g \longrightarrow g$
using *assms*
using *WPowerstar-induct-lvar int-iffD1* **by** *blast*

lemma *SChopstar-induct-lvar-eq*:
assumes $\vdash (f) \frown g = g$
shows $\vdash (schopstar\ f) \frown g \longrightarrow g$
using *assms*
using *SChopstar-induct-lvar int-iffD1* **by** *blast*

lemma *SChopstarMore-induct-lvar-eq*:
assumes $\vdash (f \wedge more) \frown g = g$
shows $\vdash (schopstar\ f) \frown g \longrightarrow g$
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 $\vdash (wpowerstar\ f);g = g$
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 $\vdash (schopstar\ f) \frown g = g$
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 $\vdash (f \wedge more) \frown g = g$
shows $\vdash (schopstar\ f) \frown g = g$
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 $\vdash empty \vee f ; g \longrightarrow g$
shows $\vdash (wpowerstar\ f) \longrightarrow g$
using *assms*
by (*metis ChopEmpty WPowerstarInductL inteq-reflection*)

lemma *SChopstar-induct-lvar-empty*:
assumes $\vdash empty \vee (f) \frown g \longrightarrow g$
shows $\vdash (schopstar\ f) \longrightarrow g$
using *assms*

by (*metis FPowerstar-WPowerstar FPowerstar-more-absorb WPowerstar-induct-lvar-empty inteq-reflection*
schop-d-def schopstar-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 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 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 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*:

$\vdash (wpowerstar (wpowerstar f)) = (wpowerstar f)$

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 $\vdash f \frown g \longrightarrow (\text{schopstar } h)$
using *assms*
by (*metis AndSChopA Prop10 SChopAndB SChopstarMore-1L SChopstarMore-induct-lvar inteq-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 chopstar-d-def)

lemma *WPowerstar-1R*:

$\vdash (\text{wpowerstar } f) ; f \longrightarrow (\text{wpowerstar } f)$

by (*simp add: Chop-WPowerstar-Closure WPowerstar-ext*)

lemma *SChopstar-1R*:

$\vdash (\text{schopstar } f) \frown (f \wedge \text{finite}) \longrightarrow (\text{schopstar } f)$

by (*simp add: SChop-SChopstar-Closure SChopstar-ext*)

lemma *SChopstarMore-1R*:

$\vdash (\text{schopstar } f) \frown (f \wedge \text{fmore}) \longrightarrow (\text{schopstar } f)$

by (*simp add: SChop-SChopstar-Closure SChopstarMore-ext fmore-d-def*)

lemma *WPowerstar-unfoldR*:

$\vdash \text{empty} \vee (\text{wpowerstar } f) ; f \longrightarrow (\text{wpowerstar } f)$

by (*meson Prop02 WPowerstar-1R WPowerstar-imp-empty*)

lemma *SChopstar-unfoldR*:

$\vdash \text{empty} \vee (\text{schopstar } f) \frown (f \wedge \text{finite}) \longrightarrow (\text{schopstar } f)$

by (*meson Prop02 SChopstar-1R SChopstar-imp-empty*)

lemma *SChopstarMore-unfoldR*:

$\vdash \text{empty} \vee (\text{schopstar } f) \frown (f \wedge \text{fmore}) \longrightarrow (\text{schopstar } f)$

by (*meson Prop02 SChopstarMore-1R SChopstar-imp-empty*)

lemma *WPowerstar-sim1*:

assumes $\vdash f ; h \longrightarrow h ; g$

shows $\vdash (\text{wpowerstar } f) ; h \longrightarrow h ; (\text{wpowerstar } g)$

proof –

have 1: $\vdash (f ; h) ; (\text{wpowerstar } g) \longrightarrow (h ; g) ; (\text{wpowerstar } g)$

by (*simp add: LeftChopImpChop assms*)

have 2: $\vdash (h ; g) ; (\text{wpowerstar } g) \longrightarrow h ; (\text{wpowerstar } g)$

by (*metis ChopAssoc RightChopImpChop WPowerstar-1L inteq-reflection*)

have 3: $\vdash (f ; h) ; (\text{wpowerstar } g) \longrightarrow h ; (\text{wpowerstar } g)$

using 1 2 *lift-imp-trans* **by** *blast*

have 4: $\vdash (\text{wpowerstar } g) = (\text{empty} \vee g ; (\text{wpowerstar } g))$

by (*simp add: WPowerstarEqv*)

have 41: $\vdash h ; (\text{empty} \vee g ; (\text{wpowerstar } g)) = (h ; \text{empty} \vee h ; (g ; (\text{wpowerstar } g)))$

by (*simp add: ChopOrEqv*)

have 42: $\vdash h ; (g ; (\text{wpowerstar } g)) = (h ; g) ; (\text{wpowerstar } g)$

by (*simp add: ChopAssoc*)

have 5: $\vdash h ; (\text{wpowerstar } g) = (h \vee (h ; g) ; (\text{wpowerstar } g))$

by (*metis 4 41 42 ChopEmpty inteq-reflection*)

have 6: $\vdash h \longrightarrow h ; (\text{wpowerstar } g)$

using 5 **by** *fastforce*

have 7: $\vdash h \vee (f ; h) ; (\text{wpowerstar } g) \longrightarrow h ; (\text{wpowerstar } g)$

```

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  $\wedge$  more h LIFT g  $\wedge$  more]
by (metis SChopstar-and-more int-eq)

```

```

lemma WPowerstar-Quasicom-m-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 (wpowerstar (f ∨ g))*]
by (*metis 0 assms int-eq*)
qed

lemma *SChopstar-Quasicommm-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-Quasicommmore-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-Quasicommm-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 $\vdash ((schopstar\ g) \frown ((f) \wedge finite) \longrightarrow (f) \frown (schopstar\ (f \vee g)))$

shows $\vdash ((g) \frown ((f) \wedge finite) \longrightarrow (f) \frown (schopstar\ (f \vee g)))$

proof –

have 1: $\vdash (g) \frown ((f) \wedge finite) \longrightarrow (schopstar\ g) \frown ((f) \wedge finite)$

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 $\vdash ((schopstar\ g) \frown ((f \wedge more) \wedge finite) \longrightarrow (f \wedge more) \frown (schopstar\ (f \vee g)))$

shows $\vdash ((g \wedge more) \frown ((f \wedge more) \wedge finite) \longrightarrow (f \wedge more) \frown (schopstar\ (f \vee g)))$

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*:

$(\vdash (g; f \longrightarrow f ; (wpowerstar\ (f \vee g)))) =$

$(\vdash ((wpowerstar\ g); f \longrightarrow f ; (wpowerstar\ (f \vee g))))$

using WPowerstar-Quasicommm-varA WPowerstar-Quasicommm-varB **by** blast

lemma *SChopstar-Quasicommm-var*:

$(\vdash ((g) \frown ((f) \wedge finite) \longrightarrow (f) \frown (schopstar\ (f \vee g)))) =$

$(\vdash ((schopstar\ g) \frown ((f) \wedge finite) \longrightarrow (f) \frown (schopstar\ (f \vee g))))$

using SChopstar-Quasicommm-varA SChopstar-Quasicommm-varB **by** blast

lemma *SChopstar-QuasicommmMore-var*:

$(\vdash ((g \wedge more) \frown ((f \wedge more) \wedge finite) \longrightarrow (f \wedge more) \frown (schopstar\ (f \vee g)))) =$

$(\vdash ((schopstar\ g) \frown ((f \wedge more) \wedge finite) \longrightarrow (f \wedge more) \frown (schopstar\ (f \vee g))))$

using SChopstar-QuasicommmMore-varA SChopstar-QuasicommmMore-varB **by** blast

lemma *WPowerstar-slide1*:

$\vdash (wpowerstar\ (f ; g); f \longrightarrow f ; (wpowerstar\ (g; f)))$

by (simp add: ChopAssoc WPowerstar-sim1 int-iffD2)

lemma *SChopstar-slide1*:

$\vdash (schopstar\ (f \frown g) \frown (f \wedge finite) \longrightarrow f \frown (schopstar\ (g \frown f)))$

using SChopstar-sim1

by (metis SChopAssoc int-iffD2)

lemma *WPowerstar-slide1-var1*:

$\vdash (wpowerstar f);f \longrightarrow f;(wpowerstar f)$
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*:

$\vdash (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{finite}) \vee (\text{schopstar } f) \neg (\text{schopstar } f)) \longrightarrow (\text{schopstar } f)$
using *SChopstar-rtc1*[of *LIFT* $f \wedge \text{more}$] *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*:

$(\vdash \text{empty} \vee (f) \neg (f \wedge \text{finite}) \longrightarrow (f \wedge \text{finite})) = (\vdash (f \wedge \text{finite}) = (\text{schopstar } f))$

proof –

have 1: $(\vdash \text{empty} \vee (f) \neg (f \wedge \text{finite}) \longrightarrow (f \wedge \text{finite})) \implies (\vdash (f \wedge \text{finite}) = (\text{schopstar } f))$

using *SChopstar-induct-lvar-empty*[of f *LIFT* $f \wedge \text{finite}$]

by (*simp* *add*: *Prop11 SChopstar-ext*)

have 2: $(\vdash (f \wedge \text{finite}) = (\text{schopstar } f)) \implies (\vdash \text{empty} \vee (f) \neg (f \wedge \text{finite}) \longrightarrow (f \wedge \text{finite}))$

by (*metis* *Prop02 SChopstar-1L SChopstar-imp-empty inteq-reflection*)

show *?thesis*

by (*metis* 1 2 *inteq-reflection*)

qed

lemma *SChopstarMore-rtc2*:

$(\vdash \text{empty} \vee (f \wedge \text{more}) \neg ((f \wedge \text{more}) \wedge \text{finite}) \longrightarrow ((f \wedge \text{more}) \wedge \text{finite})) =$
 $(\vdash ((f \wedge \text{more}) \wedge \text{finite}) = (\text{schopstar } f))$

using *SChopstar-rtc2*[of *LIFT* $f \wedge \text{more}$] *SChopstar-and-more*[of f]

by (*metis* *inteq-reflection*)

lemma *SChopstarMore-rtc2-alt*:

$(\vdash \text{empty} \vee (f \wedge \text{more}) \neg (f \wedge \text{finite}) \longrightarrow (f \wedge \text{finite})) = (\vdash (f \wedge \text{finite}) = (\text{schopstar } f))$

proof –

have 1: $(\vdash \text{empty} \vee (f \wedge \text{more}) \neg (f \wedge \text{finite}) \longrightarrow (f \wedge \text{finite})) \implies (\vdash (f \wedge \text{finite}) = (\text{schopstar } f))$

using *SChopstarMore-induct-lvar-empty*[of f *LIFT* $f \wedge \text{finite}$]

by (*simp* *add*: *SChopstar-ext int-iffI*)

have 2: $(\vdash (f \wedge \text{finite}) = (\text{schopstar } f)) \implies (\vdash \text{empty} \vee (f \wedge \text{more}) \neg (f \wedge \text{finite}) \longrightarrow (f \wedge \text{finite}))$

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 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*:

$\vdash (\text{wpowerstar } f); (\text{wpowerstar } g) \longrightarrow (\text{wpowerstar } (f \vee g))$
by (*metis* *Chop-WPowerstar-Closure* *WPowerstar-subdist* *WPowerstar-swap* *inteq-reflection*)

lemma *SChopstar-subdist-var-3*:

$\vdash (\text{schopstar } f) \frown (\text{schopstar } g) \longrightarrow (\text{schopstar } (f \vee g))$
by (*metis* *SChop-SChopstar-Closure* *SChopstar-subdist* *SChopstar-swap* *inteq-reflection*)

lemma *WPowerstar-denest*:

$\vdash (\text{wpowerstar } (f \vee g)) = (\text{wpowerstar } ((\text{wpowerstar } f); (\text{wpowerstar } g)))$

proof –

have 1: $\vdash f \longrightarrow (\text{wpowerstar } f)$
by (*simp* *add*: *WPowerstar-ext*)
have 2: $\vdash (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f); (\text{wpowerstar } g)$
by (*metis* *ChopEmpty* *RightChopImpChop* *WPowerstar-imp-empty* *int-eq*)
have 3: $\vdash f \longrightarrow (\text{wpowerstar } f); (\text{wpowerstar } g)$
using 1 2 **by** *fastforce*
have 4: $\vdash g \longrightarrow (\text{wpowerstar } g)$
by (*simp* *add*: *WPowerstar-ext*)
have 5: $\vdash (\text{wpowerstar } g) \longrightarrow (\text{wpowerstar } f); (\text{wpowerstar } g)$
by (*meson* *EmptyChop* *LeftChopImpChop* *Prop11* *WPowerstar-imp-empty* *lift-imp-trans*)
have 6: $\vdash g \longrightarrow (\text{wpowerstar } f); (\text{wpowerstar } g)$
using 4 5 **by** *fastforce*
have 7: $\vdash f \vee g \longrightarrow (\text{wpowerstar } f); (\text{wpowerstar } g)$
using 3 6 **by** *fastforce*
have 9: $\vdash (\text{wpowerstar } (f \vee g)) \longrightarrow (\text{wpowerstar } ((\text{wpowerstar } f); (\text{wpowerstar } g)))$
using 7 *WPowerstar-iso* **by** *blast*
have 10: $\vdash (\text{wpowerstar } f); (\text{wpowerstar } g) \longrightarrow (\text{wpowerstar } (f \vee g))$
by (*simp* *add*: *WPowerstar-subdist-var-3*)
have 11: $\vdash (\text{wpowerstar } ((\text{wpowerstar } f); (\text{wpowerstar } g))) \longrightarrow (\text{wpowerstar } (f \vee g))$
by (*simp* *add*: 10 *Chop-WPowerstar-Closure* *WPowerstar-induct-lvar-star*)
show ?thesis **using** 11 9 *int-iffI* **by** *blast*
qed

lemma *SChopstar-denest*:

$\vdash (\text{schopstar } (f \vee g)) = (\text{schopstar } ((\text{schopstar } f) \frown (\text{schopstar } g)))$

proof –

have 1: $\vdash (f \wedge \text{finite}) \longrightarrow (\text{schopstar } f)$
by (*simp* *add*: *SChopstar-ext*)
have 2: $\vdash (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)$
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: $\vdash (f \wedge \text{finite}) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)$
using 1 2 **by** *fastforce*

have 4: $\vdash (g \wedge \text{finite}) \longrightarrow (\text{schopstar } g)$
by (*simp add: SChopstar-ext*)
have 5: $\vdash (\text{schopstar } g) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)$
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
schop-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:*

$\vdash (wpowerstar\ f); (wpowerstar\ (g; (wpowerstar\ f))) =$
 $(wpowerstar\ g); (wpowerstar\ (f; (wpowerstar\ g)))$
by (*metis WPowerstar-denest-var WPowerstar-swap int-eq*)

lemma *SChopstar-denest-var-5:*

$\vdash (schopstar\ f) \frown (schopstar\ (g \frown (schopstar\ f))) = (schopstar\ g) \frown (schopstar\ (f \frown (schopstar\ g)))$
by (*metis SChopstar-denest-var SChopstar-swap inteq-reflection*)

lemma *SChopstarMore-denest-var-5:*

$\vdash (schopstar\ f) \frown (schopstar\ ((g \wedge more) \frown (schopstar\ f))) = (schopstar\ g) \frown (schopstar\ (f \frown (schopstar\ g)))$

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:*

$\vdash ((wpowerstar\ f); (wpowerstar\ g)); (wpowerstar\ (f \vee g)) = (wpowerstar\ (f \vee g))$
by (*metis ChopAssoc WPowerstar-denest WPowerstar-denest-var-3 inteq-reflection*)

lemma *SChopstar-denest-var-6:*

$\vdash ((schopstar\ f) \frown (schopstar\ g)) \frown (schopstar\ (f \vee g)) = (schopstar\ (f \vee g))$
by (*metis SChopAssoc SChopstar-denest SChopstar-denest-var-3 inteq-reflection*)

lemma *WPowerstar-denest-var-7:*

$\vdash (wpowerstar\ (f \vee g)) ; ((wpowerstar\ f); (wpowerstar\ g)) = (wpowerstar\ (f \vee g))$

proof –

have 1: $\vdash (wpowerstar\ (f \vee g)) ; ((wpowerstar\ f); (wpowerstar\ g)) \longrightarrow$
 $(wpowerstar\ (f \vee g)) ; (wpowerstar\ ((wpowerstar\ f); (wpowerstar\ g)))$

by (*simp add: RightChopImpChop WPowerstar-ext*)

have 2: $\vdash (wpowerstar\ (f \vee g)) ; ((wpowerstar\ f); (wpowerstar\ g)) \longrightarrow (wpowerstar\ (f \vee g))$

by (*simp add: Chop-WPowerstar-Closure WPowerstar-subdist-var-3*)

have 3: $\vdash empty \longrightarrow (wpowerstar\ (f \vee g)) ; ((wpowerstar\ f); (wpowerstar\ g))$

by (*metis ChopEmpty ChopImpChop WPowerstar-imp-empty inteq-reflection*)

have 4: $\vdash (f \vee g) ; ((wpowerstar\ (f \vee g)) ; ((wpowerstar\ f); (wpowerstar\ g))) \longrightarrow$

$((wpowerstar\ (f \vee g)) ; ((wpowerstar\ f); (wpowerstar\ g)))$

by (*metis ChopAssoc LeftChopImpChop WPowerstar-1L inteq-reflection*)

have 5: $\vdash empty \vee (f \vee g) ; ((wpowerstar\ (f \vee g)) ; ((wpowerstar\ f); (wpowerstar\ g))) \longrightarrow$

$((wpowerstar\ (f \vee g)) ; ((wpowerstar\ f); (wpowerstar\ g)))$

using 3 4 *Prop02* **by** *blast*

have 6: $\vdash (wpowerstar\ (f \vee g)) \longrightarrow ((wpowerstar\ (f \vee g)) ; ((wpowerstar\ f); (wpowerstar\ g)))$

using 5 *WPowerstar-induct-lvar-empty* **by** *blast*

show *?thesis* **using** 2 6 *int-iffI* **by** *blast*

qed

lemma *SChopstar-denest-var-7:*

$\vdash (schopstar\ (f \vee g)) \frown ((schopstar\ f) \frown (schopstar\ g)) = (schopstar\ (f \vee g))$

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: $\vdash (\text{schopstar } (f \vee g)) \frown ((\text{schopstar } f) \frown (\text{schopstar } g)) \longrightarrow (\text{schopstar } (f \vee g))$
by (*simp add: SChop-SChopstar-Closure SChopstar-subdist-var-3*)
have 3: $\vdash \text{empty} \longrightarrow (\text{schopstar } (f \vee g)) \frown ((\text{schopstar } f) \frown (\text{schopstar } g))$
by (*metis EmptySChop SChopImpSChop SChopstar-imp-empty inteq-reflection*)
have 4: $\vdash (f \vee g) \frown ((\text{schopstar } (f \vee g)) \frown ((\text{schopstar } f) \frown (\text{schopstar } g))) \longrightarrow$
 $((\text{schopstar } (f \vee g)) \frown ((\text{schopstar } f) \frown (\text{schopstar } g)))$
by (*metis LeftSChopImpSChop SChopAssoc SChopstar-1L inteq-reflection*)
have 5: $\vdash \text{empty} \vee (f \vee g) \frown ((\text{schopstar } (f \vee g)) \frown ((\text{schopstar } f) \frown (\text{schopstar } g))) \longrightarrow$
 $((\text{schopstar } (f \vee g)) \frown ((\text{schopstar } f) \frown (\text{schopstar } g)))$
using 3 4 *Prop02* **by** *blast*
have 6: $\vdash (\text{schopstar } (f \vee g)) \longrightarrow ((\text{schopstar } (f \vee g)) \frown ((\text{schopstar } f) \frown (\text{schopstar } g)))$
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:*

$\vdash ((\text{schopstar } f) \frown (\text{schopstar } g)) \frown (\text{schopstar } ((\text{schopstar } f) \frown (\text{schopstar } g))) =$
 $(\text{schopstar } ((\text{schopstar } f) \frown (\text{schopstar } g)))$
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:*

$\vdash (\text{schopstar } ((\text{schopstar } f) \frown (\text{schopstar } g))) \frown ((\text{schopstar } f) \frown (\text{schopstar } g)) =$
 $(\text{schopstar } ((\text{schopstar } f) \frown (\text{schopstar } g)))$
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: $(\vdash g ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f);(\text{wpowerstar } g)) \Longrightarrow$
 $(\vdash (\text{wpowerstar } g) ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f);(\text{wpowerstar } g))$
by (*metis Prop02 WPowerstar-imp-empty WPowerstar-rtc2 WPowerstar-sim1 WPowerstar-trans in-*
teq-reflection)
have 2: $(\vdash (\text{wpowerstar } g) ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f);(\text{wpowerstar } g)) \Longrightarrow$
 $(\vdash g ; (\text{wpowerstar } f) \longrightarrow (\text{wpowerstar } f);(\text{wpowerstar } g))$

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: $(\vdash g \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)) \implies$

$(\vdash (\text{schopstar } g) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g))$

using *SChopstar-sim1*[of *g LIFT (schopstar f) LIFT (schopstar g)*]

by (metis Prop10 SChopstar-finite SChopstar-invol inteq-reflection)

have 2: $(\vdash (\text{schopstar } g) \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g)) \implies$

$(\vdash g \frown (\text{schopstar } f) \longrightarrow (\text{schopstar } f) \frown (\text{schopstar } g))$

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: $\vdash (wpowerstar (f \vee g)) \longrightarrow (wpowerstar f) ; (wpowerstar g)$
by (*metis* 5 *WPowerstar-denest inteq-reflection*)
have 7: $\vdash (wpowerstar f) ; (wpowerstar g) \longrightarrow (wpowerstar (f \vee g))$
by (*simp* *add: WPowerstar-subdist-var-3*)
show ?thesis
by (*simp* *add: 6 7 int-iffI*)
qed

lemma *SChopstar-church-rosser*:

assumes $\vdash (schopstar g) \frown (schopstar f) \longrightarrow (schopstar f) \frown (schopstar g)$
shows $\vdash (schopstar (f \vee g)) = (schopstar f) \frown (schopstar g)$
proof –
have 0: $\vdash ((schopstar f) \frown (schopstar g)) \frown ((schopstar f)) \longrightarrow$
 $((schopstar f) \frown (schopstar f)) \frown ((schopstar g))$
by (*metis* *RightSChopImpSChop SChopAssoc assms inteq-reflection*)
have 01: $\vdash (((schopstar f) \frown (schopstar g)) \frown ((schopstar f))) \frown (schopstar g) \longrightarrow$
 $((((schopstar f) \frown (schopstar f)) \frown ((schopstar g))) \frown (schopstar g))$
by (*simp* *add: 0 LeftSChopImpSChop*)
have 1: $\vdash ((schopstar f) \frown (schopstar g)) \frown ((schopstar f) \frown (schopstar g)) \longrightarrow$
 $((schopstar f) \frown (schopstar f)) \frown ((schopstar g) \frown (schopstar g))$
by (*metis* 01 *SChopAssoc inteq-reflection*)
have 2: $\vdash ((schopstar f) \frown (schopstar g)) \frown ((schopstar f) \frown (schopstar g)) \longrightarrow$
 $((schopstar f) \frown (schopstar g))$
by (*metis* (*no-types, lifting*) 1 *SChopstar-denest SChopstar-denest-var-3 int-eq int-simps(27)*)
have 3: $\vdash empty \longrightarrow ((schopstar f) \frown (schopstar g))$
by (*meson* *EmptySChop Prop11 SChopImpSChop SChopstar-imp-empty lift-imp-trans*)
have 4: $\vdash empty \vee ((schopstar f) \frown (schopstar g)) \frown ((schopstar f) \frown (schopstar g)) \longrightarrow$
 $((schopstar f) \frown (schopstar g))$
using 2 3 *Prop02* **by** *blast*
have 5: $\vdash (schopstar ((schopstar f) \frown (schopstar g))) \longrightarrow ((schopstar f) \frown (schopstar g))$
using 4 *SChopstar-induct-lvar-empty* **by** *blast*
have 6: $\vdash (schopstar (f \vee g)) \longrightarrow (schopstar f) \frown (schopstar g)$
by (*metis* 5 *SChopstar-denest inteq-reflection*)
have 7: $\vdash (schopstar f) \frown (schopstar g) \longrightarrow (schopstar (f \vee g))$
by (*simp* *add: SChopstar-subdist-var-3*)
show ?thesis
by (*simp* *add: 6 7 int-iffI*)
qed

lemma *WPowerstar-church-rosser-var*:

assumes $\vdash g ; (wpowerstar f) \longrightarrow (wpowerstar f) ; (wpowerstar g)$
shows $\vdash (wpowerstar (f \vee g)) = (wpowerstar f) ; (wpowerstar g)$
using *assms*
by (*simp* *add: WPowerstar-church-rosser WPowerstar-confluence*)

lemma *SChopstar-church-rosser-var*:

assumes $\vdash g \frown (schopstar f) \longrightarrow (schopstar f) \frown (schopstar g)$
shows $\vdash (schopstar (f \vee g)) = (schopstar f) \frown (schopstar g)$
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* $g \wedge \text{more}$ f]
SChopstar-and-more[of g]
by (*metis* *inteq-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*) *AndEmptyChopAndEmptyEqvAndEmptyChopImpChopChopOrEqvChopOrImp*
FiniteAndEmptyEqvEmptyProp02WPowerstarEqvWPowerstar-1LWPowerstar-imp-emptyint-iffIinteq-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* *EmptyImpFiniteProp12SChopstar-WPowerstarWPowerstar-sup-id-star1inteq-reflectionschop-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* *ChopEmptyChopImpChopWPowerstar-1Rint-eqint-iffD2int-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* $(f \wedge \text{finite})$] *SChopstar-WPowerstar*[of f]
by (*metis* *EmptyImpFiniteProp10Prop12SChopstar-finiteinteq-reflectionschop-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*) *ChopAssocChopEmptyChopOrEqvEmptyOrChopEqvinteq-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 *inteq-reflectionwpowerstar-unfoldl-eq*)
have 4: $\vdash (\text{wpowerstar } f) \longrightarrow (\text{empty} \vee (\text{wpowerstar } f);f)$
by (*metis* 2 3 *WPowerstar-induct-lvar-emptyint-eqint-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 *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*:

$\vdash (\text{empty} \vee (f) \frown ((\text{s chopstar } (g \frown f)) \frown (g \wedge \text{finite}))) = (\text{s chopstar } (f \frown g))$

proof –

have 1: $\vdash (\text{s chopstar } (f \frown g)) = (\text{empty} \vee (\text{s chopstar } (f \frown g)) \frown (f \frown (g \wedge \text{finite})))$

by (metis FiniteImportSChopRight SChopAndCommute SChopstar-unfoldr-eq inteq-reflection)

have 2: $\vdash (\text{s chopstar } (f \frown g)) \frown (f \frown (g \wedge \text{finite})) \longrightarrow (f) \frown ((\text{s chopstar } (g \frown f)) \frown (g \wedge \text{finite}))$

using SChopstar-slide1[of f g]

by (metis (no-types, opaque-lifting) ChopAndFiniteDist ChopEmpty LeftSChopImpSChop SChopAssoc inteq-reflection schop-d-def)

have 3: $\vdash (\text{s chopstar } (f \frown g)) \longrightarrow (\text{empty} \vee (f) \frown ((\text{s chopstar } (g \frown f)) \frown (g \wedge \text{finite})))$

using 1 2 by fastforce

have 4: $\vdash (f) \frown ((\text{s chopstar } (g \frown f)) \frown (g \wedge \text{finite})) \longrightarrow (f \frown g) \frown (\text{s chopstar } (f \frown g))$

using SChopstar-slide1[of g f]

by (metis RightSChopImpSChop SChopAssoc inteq-reflection)

have 5: $\vdash (\text{empty} \vee (f) \frown ((\text{s chopstar } (g \frown f)) \frown (g \wedge \text{finite}))) \longrightarrow$
 $\text{empty} \vee (f \frown g) \frown (\text{s chopstar } (f \frown g))$

using 4 by fastforce

have 6: $\vdash (\text{empty} \vee (f \frown g) \frown (\text{s chopstar } (f \frown g))) \longrightarrow (\text{s chopstar } (f \frown g))$

by (meson Prop02 SChopstar-1L SChopstar-imp-empty)

show ?thesis

by (meson 3 5 6 int-iffI lift-imp-trans)

qed

lemma *WPowerstar-slide* :

$\vdash (\text{w powerstar } (f;g));f = f;(\text{w powerstar } (g;f))$

proof –

have 1: $\vdash f;(\text{w powerstar } (g;f)) = f;(\text{empty} \vee g;((\text{w powerstar } (f;g));f))$

by (metis RightChopEqvChop WPowerstar-star-prod-unfold inteq-reflection)

have 2: $\vdash f;(\text{empty} \vee g;((\text{w powerstar } (f;g));f)) =$
 $(f;\text{empty} \vee f;(g;((\text{w powerstar } (f;g));f))))$

by (simp add: ChopOrEqv)

have 3: $\vdash f;\text{empty} = \text{empty};f$

by (metis ChopEmpty EmptyChop int-eq)

have 4: $\vdash f;(g;((\text{w powerstar } (f;g));f)) =$
 $((f;g);(\text{w powerstar } (f;g)));f$

by (metis ChopAssoc int-eq)

have 5: $\vdash (f;\text{empty} \vee f;(g;((\text{w powerstar } (f;g));f)))) =$
 $(\text{empty};f \vee ((f;g);(\text{w powerstar } (f;g)));f)$

using 3 4 by fastforce

have 6: $\vdash (\text{empty};f \vee ((f;g);(\text{w powerstar } (f;g)));f) =$
 $(\text{empty} \vee ((f;g);(\text{w powerstar } (f;g))));f$

by (meson OrChopEqv Prop11)

have 7: $\vdash f;(\text{empty} \vee g;((\text{w powerstar } (f;g));f)) =$
 $(\text{empty} \vee (f;g);(\text{w powerstar } (f;g)));f$

by (metis 2 3 4 6 inteq-reflection)

have 8: $\vdash (\text{empty} \vee (f;g);(\text{w powerstar } (f;g))) = (\text{w powerstar } (f;g))$

by (simp add: w powerstar-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 chop-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 integ-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: $\vdash (\text{schopstar } f) \frown (\text{empty} \vee (g \frown (\text{schopstar } f)) \frown (\text{schopstar } (g \frown (\text{schopstar } f)))) =$

$(schopstar\ f) \frown (empty \vee g \frown (schopstar\ (f \vee g)))$
by (*metis 1 2 SChopAssoc inteq-reflection*)
have 4: $\vdash (schopstar\ f) \frown (empty \vee g \frown (schopstar\ (f \vee g))) =$
 $((schopstar\ f) \frown empty \vee (schopstar\ f) \frown (g \frown (schopstar\ (f \vee g))))$
using *SChopOrEqv* **by** *blast*
have 5: $\vdash (schopstar\ f) \frown empty = (schopstar\ f)$
by (*metis ChopEmpty Prop10 SChopstar-finite inteq-reflection schop-d-def*)
have 6: $\vdash ((schopstar\ f) \frown empty \vee (schopstar\ f) \frown (g \frown (schopstar\ (f \vee g)))) =$
 $((schopstar\ f) \vee (schopstar\ f) \frown (g \frown (schopstar\ (f \vee g))))$
using 5 **by** *auto*
show *?thesis*
by (*metis 1 2 3 4 6 int-eq*)
qed

lemma *SChopstarMore-or-unfold*:

$\vdash ((schopstar\ f) \vee (schopstar\ f) \frown ((g \wedge more) \frown (schopstar\ (f \vee g)))) = (schopstar\ (f \vee g))$
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*:

$\vdash (wpowerstar\ (f \vee g));h =$
 $(wpowerstar\ f);(g;((wpowerstar\ (f \vee g));h) \vee h)$
proof –
have 1: $\vdash (wpowerstar\ (f \vee g));h =$
 $((wpowerstar\ f) \vee (wpowerstar\ f);(g;(wpowerstar\ (f \vee g))));h$
using *WPowerstar-or-unfold*[of *f g*] **by** (*metis LeftChopEqvChop int-eq*)
have 2: $\vdash ((wpowerstar\ f) \vee (wpowerstar\ f);(g;(wpowerstar\ (f \vee g))));h =$
 $((wpowerstar\ f);h \vee ((wpowerstar\ f);(g;(wpowerstar\ (f \vee g))));h)$
by (*simp add: OrChopEqv*)
have 3: $\vdash ((wpowerstar\ f);(g;(wpowerstar\ (f \vee g))));h =$
 $(wpowerstar\ f);(g;((wpowerstar\ (f \vee g));h))$
by (*metis ChopAssoc inteq-reflection*)
have 3: $\vdash ((wpowerstar\ f);h \vee ((wpowerstar\ f);(g;(wpowerstar\ (f \vee g)))));h =$
 $(wpowerstar\ f);(h \vee g;((wpowerstar\ (f \vee g));h))$
by (*metis 3 ChopOrEqv inteq-reflection*)
have 4: $\vdash (h \vee g;((wpowerstar\ (f \vee g));h)) =$
 $(g;((wpowerstar\ (f \vee g));h) \vee h)$
by *fastforce*
show *?thesis*
by (*metis 1 2 3 4 int-eq*)
qed

lemma *SChopstar-troeger*:

$\vdash (schopstar\ (f \vee g)) \frown (h) =$
 $(schopstar\ f) \frown (g \frown ((schopstar\ (f \vee g)) \frown (h)) \vee (h))$
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: $\vdash f \wedge (\text{empty} \wedge (\text{schopstar } (f \frown f)) \vee f \wedge (\text{schopstar } (f \frown f))) =$
 $f \wedge ((\text{schopstar } (f \frown f)) \vee f \wedge (\text{schopstar } (f \frown f)))$
 using 2 RightSCHopEqvSCHop by blast
 have 4: $\vdash f \wedge ((\text{schopstar } (f \frown f)) \vee f \wedge (\text{schopstar } (f \frown f))) \longrightarrow$
 $(\text{schopstar } (f \frown f)) \vee f \wedge (\text{schopstar } (f \frown f))$
 using SCHopAssoc SCHopOrEqv SCHopstar-unfoldl-eq by fastforce
 have 41: $\vdash ((\text{schopstar } (f \frown f)) \vee f \wedge (\text{schopstar } (f \frown f))) =$
 $((\text{empty} \vee f) \wedge (\text{schopstar } (f \frown f)))$
 by (metis 2 OrSCHopEqv inteq-reflection)
 have 5: $\vdash f \wedge ((\text{empty} \vee f) \wedge (\text{schopstar } (f \frown f))) \longrightarrow$
 $((\text{empty} \vee f) \wedge (\text{schopstar } (f \frown f)))$
 by (metis 4 41 int-eq)
 have 6: $\vdash \text{empty} \longrightarrow ((\text{empty} \vee f) \wedge (\text{schopstar } (f \frown f)))$
 using EmptyOrSCHopEqv SCHopstar-imp-empty by fastforce
 have 7: $\vdash \text{empty} \vee f \wedge ((\text{empty} \vee f) \wedge (\text{schopstar } (f \frown f))) \longrightarrow$
 $((\text{empty} \vee f) \wedge (\text{schopstar } (f \frown f)))$
 using 5 6 Prop02 by blast
 have 8: $\vdash (\text{schopstar } f) \longrightarrow ((\text{empty} \vee f) \wedge (\text{schopstar } (f \frown f)))$
 using 7 SCHopstar-induct-lvar-empty by blast
 have 9: $\vdash ((\text{empty} \vee f) \wedge (\text{schopstar } (f \frown f))) \longrightarrow (\text{empty} \vee f) \wedge (\text{schopstar } f)$
 using SCHopstar-square[of f]
 using RightSCHopImpSCHop by blast
 have 10: $\vdash (\text{empty} \vee f) \wedge (\text{schopstar } f) \longrightarrow (\text{schopstar } f)$
 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:

$\vdash (\text{empty} \vee (f \wedge \text{more})) \wedge (\text{schopstar } (f \frown f)) = (\text{schopstar } f)$

proof –

have 1: $\vdash (f \wedge \text{more}) \wedge ((\text{empty} \vee (f \wedge \text{more})) \wedge (\text{schopstar } (f \frown f))) =$
 $(f \wedge \text{more}) \wedge (\text{empty} \wedge (\text{schopstar } (f \frown f)) \vee (f \wedge \text{more}) \wedge (\text{schopstar } (f \frown f)))$
 by (simp add: OrSCHopEqv RightSCHopEqvSCHop)
 have 2: $\vdash (\text{empty} \wedge (\text{schopstar } (f \frown f)) \vee (f \wedge \text{more}) \wedge (\text{schopstar } (f \frown f))) =$
 $((\text{schopstar } (f \frown f)) \vee (f \wedge \text{more}) \wedge (\text{schopstar } (f \frown f)))$
 by (meson EmptyOrSCHopEqv OrSCHopEqv Prop04)
 have 3: $\vdash (f \wedge \text{more}) \wedge (\text{empty} \wedge (\text{schopstar } (f \frown f)) \vee (f \wedge \text{more}) \wedge (\text{schopstar } (f \frown f))) =$
 $(f \wedge \text{more}) \wedge ((\text{schopstar } (f \frown f)) \vee (f \wedge \text{more}) \wedge (\text{schopstar } (f \frown f)))$
 using 2 RightSCHopEqvSCHop by blast
 have 30: $\vdash ((f \wedge \text{more}) \wedge (f \wedge \text{more})) \longrightarrow f \frown f$
 by (metis Prop11 Prop12 SCHopImpSCHop lift-and-com)
 have 31: $\vdash ((f \wedge \text{more}) \wedge (f \wedge \text{more})) \wedge (\text{schopstar } (f \frown f)) \longrightarrow (\text{schopstar } (f \frown f))$
 by (metis 30 AndSCHopA Prop05 Prop10 SCHopstar-unfoldl-eq int-eq lift-and-com)
 have 4: $\vdash (f \wedge \text{more}) \wedge ((\text{schopstar } (f \frown f)) \vee (f \wedge \text{more}) \wedge (\text{schopstar } (f \frown f))) \longrightarrow$
 $(f \wedge \text{more}) \wedge (\text{schopstar } (f \frown f)) \vee (\text{schopstar } (f \frown f))$
 using 31 SCHopAssoc SCHopOrImp by fastforce
 have 5: $\vdash (f \wedge \text{more}) \wedge ((\text{empty} \vee (f \wedge \text{more})) \wedge (\text{schopstar } (f \frown f))) \longrightarrow$
 $((\text{empty} \vee (f \wedge \text{more})) \wedge (\text{schopstar } (f \frown f)))$

```

  by (metis (no-types, opaque-lifting) 4 EmptyOrSChopEqv Prop11 int-simps(33)
    inteq-reflection lift-and-com lift-imp-trans)
have 6:  $\vdash \text{empty} \longrightarrow ((\text{empty} \vee (f \wedge \text{more})) \frown (\text{s chopstar } (f \frown f)))$ 
  using EmptyOrSChopEqv SChopstar-unfoldl-eq by fastforce
have 7:  $\vdash \text{empty} \vee (f \wedge \text{more}) \frown ((\text{empty} \vee (f \wedge \text{more})) \frown (\text{s chopstar } (f \frown f))) \longrightarrow$ 
   $((\text{empty} \vee (f \wedge \text{more})) \frown (\text{s chopstar } (f \frown f)))$ 
  using 5 6 Prop02 by blast
have 8:  $\vdash (\text{s chopstar } f) \longrightarrow ((\text{empty} \vee (f \wedge \text{more})) \frown (\text{s chopstar } (f \frown f)))$ 
  using 7 SChopstarMore-induct-lvar-empty by blast
have 9:  $\vdash ((\text{empty} \vee (f \wedge \text{more})) \frown (\text{s chopstar } (f \frown f))) \longrightarrow (\text{empty} \vee (f \wedge \text{more})) \frown (\text{s chopstar } f)$ 
  using SChopstar-square[of f]
  using RightSChopImpSChop by blast
have 10:  $\vdash (\text{empty} \vee (f \wedge \text{more})) \frown (\text{s chopstar } f) \longrightarrow (\text{s chopstar } f)$ 
  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  $\vdash (\text{w powerstar } f);f = f$ 
proof -
  have 1:  $\vdash f \vee f;f \longrightarrow f$ 
    using assms by fastforce
  have 2:  $\vdash (\text{w powerstar } f);f \longrightarrow f$ 
    by (simp add: WPowerstar-inductL-var-equiv assms)
  have 3:  $\vdash f \longrightarrow (\text{w powerstar } f);f$ 
    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  $\vdash (\text{s chopstar } f) \frown (f \wedge \text{finite}) = (f \wedge \text{finite})$ 
proof -
  have 1:  $\vdash f \vee f \frown f \longrightarrow f$ 
    using assms by fastforce
  have 2:  $\vdash (\text{s chopstar } f) \frown (f \wedge \text{finite}) \longrightarrow f \wedge \text{finite}$ 
    by (metis Prop12 SChopAndA SChopstar-1R SChopstar-finite SChopstar-induct-lvar assms lift-imp-trans)
  have 3:  $\vdash f \wedge \text{finite} \longrightarrow (\text{s chopstar } f) \frown (f \wedge \text{finite})$ 
    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  $\vdash (\text{w powerstar } f);f = f$ 
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:  $\vdash h \longrightarrow (wpowerstar\ g) ; h$ 
  by (metis AndChopB EmptyChop Prop10 WPowerstar-imp-empty inteq-reflection)
have 5:  $\vdash h \vee (wpowerstar\ g) ; (h ; f) \longrightarrow (wpowerstar\ g) ; h$ 
  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  $\vdash g \frown schopstar\ f \longrightarrow h$ 
proof -
have 1:  $\vdash g \wedge finite \longrightarrow h$ 
  using assms by fastforce
have 2:  $\vdash h \frown f \longrightarrow h$ 
  using assms by fastforce
have 3:  $\vdash g \wedge finite \vee h \frown f \longrightarrow h$ 
  using 1 2 by fastforce
have 4:  $\vdash (g \wedge finite);fpowerstar\ f \longrightarrow h$ 
  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  $\vdash h \frown (s chopstar\ f) \longrightarrow (s chopstar\ g) \frown h$ 
proof -
have 1:  $\vdash (s chopstar\ g) \frown (h \frown (f)) \longrightarrow (s chopstar\ g) \frown (g \frown h)$ 
  by (simp add: RightSChopImpSChop assms)
have 2:  $\vdash (s chopstar\ g) \frown (g \frown h) \longrightarrow (s chopstar\ g) \frown h$ 
  by (metis (no-types, lifting) ChopAssoc LeftChopImpChop Prop10 SChopstar-1L SChopstar-finite
    SChopstar-WPowerstar WPowerstar-slide-var inteq-reflection schop-d-def)
have 3:  $\vdash (s chopstar\ g) \frown (h \frown (f)) \longrightarrow (s chopstar\ g) \frown h$ 
  using 1 2 lift-imp-trans by blast
have 4:  $\vdash h \longrightarrow (s chopstar\ g) \frown h$ 
  by (meson EmptySChop LeftSChopImpSChop Prop11 SChopstar-imp-empty lift-imp-trans)
have 5:  $\vdash h \vee (s chopstar\ g) \frown (h \frown (f)) \longrightarrow (s chopstar\ g) \frown h$ 
  using 3 4 Prop02 by blast
show ?thesis
  by (metis 5 SChopAssoc SChopstarInductR inteq-reflection)
qed

```

```

lemma SChopImpFinite:
 $\vdash f \longrightarrow finite \implies \vdash g \frown f \longrightarrow finite$ 
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: $(\vdash g ; f \longrightarrow g) \Longrightarrow (\vdash g ; (\text{wpowerstar } f) \longrightarrow g)$
by (*simp add: WPowerstar-inductr-var*)
have 2: $(\vdash g ; (\text{wpowerstar } f) \longrightarrow g) \Longrightarrow (\vdash g ; f \longrightarrow g)$
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: $(\vdash g \frown (f \wedge \text{finite}) \longrightarrow g) \Longrightarrow (\vdash g \frown (\text{schopstar } f) \longrightarrow g)$

by (*simp add: SChopstar-inductr-finite-var*)

have 2: $(\vdash g \frown (\text{schopstar } f) \longrightarrow g) \Longrightarrow (\vdash g \frown (f \wedge \text{finite}) \longrightarrow g)$

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[of g LIFT f \wedge more]

SChopstar-and-more[of *f*]
by (*metis integ-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 1: $\vdash (\text{schopstar } f) \frown g = (g \vee (\text{schopstar } f) \frown (f \frown g))$
by (*metis (no-types, lifting) ChopAssoc EmptyOrSChopEqv Prop10 SChopstar-1L SChopstar-finite SChopstar-unfoldr-eq SChopstar-WPowerstar WPowerstar-slide-var integ-reflection lift-imp-trans chop-d-def*)
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
using 1 2 3 **by** *fastforce*
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 schop-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) schop-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* ($f;(\text{wpowerstar } (f \vee g)) \vee g$) *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: $\vdash g;(\text{wpowerstar } f) = (g; \text{empty} \vee g;f \vee g;(f;(\text{wpowerstar } f)))$

by (*metis 8 ChopOrEqv int-eq*)

show *?thesis*

using 7 9 **by** *fastforce*

qed
qed

lemma *SChopstar-lazy-comm*:

$(\vdash g \frown (f \wedge \text{finite}) \longrightarrow f \frown (\text{schopstar } (f \vee g)) \vee g) =$
 $(\vdash g \frown (\text{schopstar } f) \longrightarrow f \frown (\text{schopstar } (f \vee g)) \vee g)$

proof

assume 1: $\vdash g \frown (f \wedge \text{finite}) \longrightarrow f \frown (\text{schopstar } (f \vee g)) \vee g$

show $\vdash g \frown (\text{schopstar } f) \longrightarrow f \frown (\text{schopstar } (f \vee g)) \vee g$

proof –

have 2: $\vdash (f \frown (\text{schopstar } (f \vee g)) \vee g) \frown (f \wedge \text{finite}) =$
 $((f \frown (\text{schopstar } (f \vee g))) \frown (f \wedge \text{finite}) \vee g \frown (f \wedge \text{finite}))$

by (*simp add: OrSChopEqv*)

have 3: $\vdash ((f \frown (\text{schopstar } (f \vee g))) \frown (f \wedge \text{finite}) \vee g \frown (f \wedge \text{finite})) \longrightarrow$
 $((f \frown (\text{schopstar } (f \vee g))) \frown (f \wedge \text{finite}) \vee (f \frown (\text{schopstar } (f \vee g))) \vee g)$

using 1

using *SChopAndA* **by** *fastforce*

have 4: $\vdash (f \frown (\text{schopstar } (f \vee g))) \frown (f \wedge \text{finite}) \longrightarrow (f \frown (\text{schopstar } (f \vee g)))$

using *SChopstar-subdist-var-1*

by (*metis Prop11 RightSChopImpSChop SChopAssoc SChop-SChopstar-Closure SChopstardef lift-imp-trans*)

have 5: $\vdash ((f \frown (\text{schopstar } (f \vee g))) \frown (f \wedge \text{finite}) \vee (f \frown (\text{schopstar } (f \vee g))) \vee g) \longrightarrow$
 $((f \frown (\text{schopstar } (f \vee g))) \vee g)$

using 4 **by** *fastforce*

have 6: $\vdash g \vee ((f \frown (\text{schopstar } (f \vee g))) \vee g) \longrightarrow ((f \frown (\text{schopstar } (f \vee g))) \vee g)$

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: $\vdash g \frown (\text{schopstar } f) \longrightarrow f \frown (\text{schopstar } (f \vee g)) \vee g$

show $\vdash g \frown (f \wedge \text{finite}) \longrightarrow f \frown (\text{schopstar } (f \vee g)) \vee g$

proof –

have 8: $\vdash (\text{schopstar } f) = (\text{empty} \vee (f \wedge \text{finite}) \vee f \frown (\text{schopstar } f))$

by (*meson AndSChopA Prop02 Prop05 Prop08 SChopstarMore-unfoldl-eq SChopstar-1L SChopstar-ext*
SChopstar-imp-empty int-iffD2 int-iffI)

have 9: $\vdash g \frown (\text{schopstar } f) = (g \frown \text{empty} \vee g \frown (f \wedge \text{finite}) \vee g \frown (f \frown (\text{schopstar } f)))$

by (*metis 8 SChopOrEqv inteq-reflection*)

show *?thesis*

using 7 9 **by** *fastforce*

qed

qed

8.6 WPowerstar

lemma *WPowerstar-and-inf*:

$\vdash (\text{wpowerstar } (f \wedge \text{inf})) = (\text{empty} \vee (f \wedge \text{inf}))$

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-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 $\vdash f \longrightarrow (g \wedge more); f$
shows $\vdash f \wedge finite \longrightarrow wpowerstar\ g$
proof –
have 14: $\vdash wpowerstar\ g = (empty \vee (g \wedge more)); (wpowerstar\ g)$
using *WPowerstar-more-absorb[of g] WPowerstarEqv[of LIFT (g \wedge more)]*
by (*metis int-eq*)
have 15: $\vdash (\neg(wpowerstar\ g)) = (more \wedge \neg((g \wedge more); wpowerstar\ g))$
using 14 **unfolding** *empty-d-def* **by** *fastforce*
have 20: $\vdash f \wedge \neg(wpowerstar\ g) \longrightarrow (g \wedge more); f \wedge \neg((g \wedge more); wpowerstar\ g)$

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 wpowerstar g LIFT (g \wedge 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 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 wpowerstar g LIFT (g \wedge 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 schopstar-d-def*)

lemma WPowerstar-skip-finite:
 $\vdash (\text{wpowerstar } \text{skip}) = \text{finite}$
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 schop-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*:

$\vdash (f;f^*) = (f \vee (f \wedge \text{more}); (f;f^*))$
by (*metis ChopAssoc ChopplusCommute ChopstarEqv EmptyOrChopEqv inteq-reflection*)

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 ∧ 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 ∧ finite)*] 1
by (*metis inteq-reflection schop-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 integ-reflection schop-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 integ-reflection schop-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*:

$\vdash ((init\ w) \wedge empty) = (w \wedge empty)$

proof –

have 1: $\vdash ((init\ w) \wedge empty) = ((w \wedge empty); \#True \wedge empty)$

by (*metis init-d-def int-eq lift-and-com*)

have 2: $\vdash ((w \wedge empty); \#True \wedge empty) = (w \wedge empty); (\#True \wedge empty)$

by (*meson AndChopA ChopAndA ChopAndEmptyEqvEmptyChopEmpty lift-imp-trans Prop11 Prop12*)

have 3: $\vdash (w \wedge empty); (\#True \wedge empty) = (w \wedge empty); empty$

using *RightChopEqvChop* **by** *fastforce*

have 4: $\vdash (w \wedge empty); empty = (w \wedge empty)$

using *ChopEmpty* **by** *blast*

from 1 2 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *InitAndNotBoxInitImpNotEmpty*:

$\vdash init\ w \wedge \neg(\Box (init\ w)) \longrightarrow \neg\ empty$

proof –

have 1: $\vdash ((init\ w) \wedge empty) = (w \wedge empty)$

by (*rule InitAndEmptyEqvAndEmpty*)

have 2: $\vdash (\neg(\Box (init\ w)) \wedge empty) = (\Diamond (\neg (init\ w)) \wedge empty)$

by (*auto simp: always-d-def*)

have 3: $\vdash (\Diamond (\neg (init\ w)) \wedge empty) = (\neg (init\ w) \wedge empty)$

by (*simp add: DiamondAndEmptyEqvAndEmpty*)

have 4: $\vdash (\neg (init\ w)) = (init\ (\neg\ w))$

using *Initprop(2)* **by** *blast*

have 5: $\vdash (\neg (init\ w) \wedge empty) = (\neg\ w \wedge empty)$

using 4 *InitAndEmptyEqvAndEmpty* **by** (*metis inteq-reflection*)

have 6: $\vdash (\neg(\Box (init\ w)) \wedge empty) = (\neg\ w \wedge empty)$

using 2 3 5 **by** *fastforce*

have 7: $\vdash \neg(init\ w \wedge \neg(\Box (init\ w)) \wedge empty)$

using 1 6 **by** *fastforce*

from 7 **show** *?thesis* **by** *auto*

qed

lemma *BoxImpTrueChopAndEmpty*:

$\vdash \Box\ f \longrightarrow \#True; (f \wedge empty)$

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 $\vdash f \longrightarrow \text{empty} \vee ((\Box(\text{init } w) \wedge \text{more}) \frown f)$
shows $\vdash (\text{init } w \wedge f) \wedge \text{finite} \longrightarrow \Box(\text{init } w)$
using *assms* **by** (*simp add: CSImpBox schop-d-def*)

lemma ChopstarInductR:

assumes $\vdash g \vee h; f \longrightarrow h$
shows $\vdash g; (\text{chopstar } f) \longrightarrow h$
by (*metis Chopstar-WPowerstar WPowerstarInductR WPowerstar-more-absorb assms int-eq*)

lemma BoxWPowerstarEqvBox:

$\vdash (\text{init } w \wedge \text{wpowerstar } (\Box(\text{init } w))) = \Box(\text{init } w)$

proof –

have 1: $\vdash \Box(\text{init } w); \Box(\text{init } w) \longrightarrow \Box(\text{init } w)$
by (*simp add: BoxStateChopBoxEqvBox int-iffD1*)
have 2: $\vdash (\text{init } w \wedge \text{empty}) \longrightarrow \Box(\text{init } w)$
by (*simp add: StateAndEmptyImpBoxState*)
have 3: $\vdash (\text{init } w \wedge \text{empty}) \vee \Box(\text{init } w); \Box(\text{init } w) \longrightarrow \Box(\text{init } w)$
using 1 2 by fastforce
have 4: $\vdash (\text{init } w \wedge \text{empty}); \text{wpowerstar } (\Box(\text{init } w)) \longrightarrow \Box(\text{init } w)$
using 3 WPowerstarInductR by blast
have 5: $\vdash \text{init } w \wedge \text{wpowerstar } (\Box(\text{init } w)) \longrightarrow \Box(\text{init } w)$
using 4 StateAndEmptyChop by fastforce
have 11: $\vdash \Box(\text{init } w) \longrightarrow (\text{init } w)$
using BoxElim by blast
have 12: $\vdash \Box(\text{init } w) \longrightarrow \text{wpowerstar } (\Box(\text{init } w))$
by (*simp add: WPowerstar-ext*)
have 13: $\vdash \Box(\text{init } w) \longrightarrow \text{init } w \wedge \text{wpowerstar } (\Box(\text{init } w))$
using 11 12 by fastforce
from 5 13 show ?thesis by fastforce
qed

lemma BoxCSEqvBox:

$\vdash (\text{init } w \wedge (\Box(\text{init } w))^*) = \Box(\text{init } w)$

by (*metis BoxWPowerstarEqvBox Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)

lemma BoxSCSEqvBox:

$\vdash (\text{init } w \wedge \text{schopstar } (\Box(\text{init } w))) = (\Box(\text{init } w) \wedge \text{finite})$

proof –

have 1: $\vdash \Box(\text{init } w) \frown (\Box(\text{init } w) \wedge \text{finite}) \longrightarrow \Box(\text{init } w) \wedge \text{finite}$
by (*metis BoxStateAndChopEqvChop FiniteChopFiniteEqvFinite int-iffD2 inteq-reflection schop-d-def*)
have 2: $\vdash (\text{init } w \wedge \text{empty}) \longrightarrow \Box(\text{init } w) \wedge \text{finite}$
using EmptyImpFinite StateAndEmptyImpBoxState by fastforce
have 3: $\vdash (\text{init } w \wedge \text{empty}) \vee \Box(\text{init } w) \frown (\Box(\text{init } w) \wedge \text{finite}) \longrightarrow \Box(\text{init } w) \wedge \text{finite}$

using 1 2 by fastforce
 have 4: $\vdash (\text{init } w \wedge \text{empty}) \frown \text{schopstar } (\Box (\text{init } w)) \longrightarrow \Box (\text{init } w) \wedge \text{finite}$
 using *SChopstarInductR 3*
 by (metis *Prop12 SChopImpFinite SChopstar-finite SChopstar-finite-absorb inteq-reflection*)
 have 5: $\vdash \text{init } w \wedge \text{schopstar } (\Box (\text{init } w)) \longrightarrow \Box (\text{init } w) \wedge \text{finite}$
 using 4 *StateAndEmptySChop* by fastforce
 have 11: $\vdash \Box (\text{init } w) \longrightarrow (\text{init } w)$
 using *BoxElim* by blast
 have 12: $\vdash \Box (\text{init } w) \wedge \text{finite} \longrightarrow \text{schopstar } (\Box (\text{init } w))$
 by (metis *SChopstar-WPowerstar WPowerstar-ext inteq-reflection*)
 have 13: $\vdash \Box (\text{init } w) \wedge \text{finite} \longrightarrow \text{init } w \wedge \text{schopstar } (\Box (\text{init } w))$
 using 11 12 by fastforce
 from 5 13 show ?thesis by fastforce
 qed

lemma *WpowerstarAndMoreEqvAndMoreChop*:

$\vdash (\text{wpowerstar } f \wedge \text{more}) = (f \wedge \text{more}); \text{wpowerstar } f$
proof –
 have 1: $\vdash (\text{empty} \vee (f \wedge \text{more}); \text{wpowerstar } f) \wedge \text{more} \longrightarrow (f \wedge \text{more}); \text{wpowerstar } f$
 by (auto simp: *empty-d-def*)
 have 2: $\vdash \text{wpowerstar } f = (\text{empty} \vee (f \wedge \text{more}); \text{wpowerstar } f)$
 by (metis *ChopstarEqv Chopstar-WPowerstar WPowerstar-more-absorb inteq-reflection*)
 have 3: $\vdash \text{wpowerstar } f \wedge \text{more} \longrightarrow (f \wedge \text{more}); \text{wpowerstar } f$
 using 1 2 by fastforce
 have 4: $\vdash (f \wedge \text{more}); \text{wpowerstar } f \longrightarrow \text{wpowerstar } f$
 using 2 by fastforce
 have 5: $\vdash (f \wedge \text{more}) \longrightarrow \text{more}$
 by auto
 hence 6: $\vdash (f \wedge \text{more}); \text{wpowerstar } f \longrightarrow \text{more}$
 by (rule *LeftChopImpMoreRule*)
 have 7: $\vdash (f \wedge \text{more}); \text{wpowerstar } f \longrightarrow \text{wpowerstar } f \wedge \text{more}$
 using 4 6 by fastforce
 from 3 7 show ?thesis by fastforce
 qed

lemma *CSAndMoreEqvAndMoreChop*:

$\vdash (f^* \wedge \text{more}) = (f \wedge \text{more}); f^*$
 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* $(\Box(\text{init } w) \wedge \text{wpowerstar } f)$ *LIFT* $(\Box(\text{init } w) \wedge f)$]

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: $\vdash \text{ba } (f \wedge \text{more} \longrightarrow g \wedge \text{more}) \longrightarrow$

$(f \wedge \text{more}); \text{wpowerstar } f \longrightarrow (g \wedge \text{more}); \text{wpowerstar } f$

by (rule BaLeftChopImpChop)

have 7: $\vdash \text{ba } (f \longrightarrow g) \wedge (f \wedge \text{more}); \text{wpowerstar } f \longrightarrow (g \wedge \text{more}); \text{wpowerstar } f$

using 5 6 by fastforce

have 8: $\vdash (g \wedge \text{more}); \text{wpowerstar } f \wedge \neg((g \wedge \text{more}); \text{wpowerstar } g)$

$\longrightarrow (g \wedge \text{more}); (\text{wpowerstar } f \wedge \neg(\text{wpowerstar } g))$

by (rule ChopAndNotChopImp)

have 9: $\vdash (g \wedge \text{more}); (\text{wpowerstar } f \wedge \neg(\text{wpowerstar } g)) \longrightarrow \text{more}; (\text{wpowerstar } f \wedge \neg(\text{wpowerstar } g))$

by (rule AndChopB)

have 10: $\vdash \text{ba } (f \longrightarrow g) \longrightarrow \text{more}; (\text{wpowerstar } f \wedge \neg(\text{wpowerstar } g)) \longrightarrow$

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 \wedge finite) LIFT (g \wedge finite)]
 SChopstar-WPowerstar[of f] SChopstar-WPowerstar[of g]
 SBa-Ba[of LIFT (f \longrightarrow g)] 1 2 by fastforce
 qed

lemma BaWPowerstarEqvWPowerstar:

\vdash ba (f = g) \wedge finite \longrightarrow (wpowerstar f = wpowerstar 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: $\vdash ba (f \longrightarrow g) \wedge finite \longrightarrow (wpowerstar f \longrightarrow wpowerstar g)$
 by (*rule* *BaWPowerstarImpWPowerstar*)
have 3: $\vdash ba (g \longrightarrow f) \wedge finite \longrightarrow (wpowerstar g \longrightarrow wpowerstar f)$
 by (*rule* *BaWPowerstarImpWPowerstar*)
have 4: $\vdash ba (f = g) \wedge finite \longrightarrow (wpowerstar f \longrightarrow wpowerstar g) \wedge (wpowerstar g \longrightarrow wpowerstar f)$
 using 1 2 3 by *fastforce*
have 5: $\vdash ((wpowerstar f \longrightarrow wpowerstar g) \wedge (wpowerstar g \longrightarrow wpowerstar f)) = (wpowerstar f = wpowerstar g)$
 by *auto*
from 4 5 **show** *?thesis* by *auto*
qed

lemma *BaCSEqvCS*:

$\vdash ba (f = g) \wedge finite \longrightarrow (f^* = g^*)$
 by (*metis* *BaWPowerstarEqvWPowerstar Chopstar-WPowerstar WPowerstar-more-absorb int-eq*)

lemma *SBaSCSEqvSCS*:

$\vdash sba (f = g) \longrightarrow (schopstar f = schopstar g)$
proof –
have 0: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$
 by *fastforce*
have 1: $\vdash sba (f = g) = (sba (f \longrightarrow g) \wedge sba (g \longrightarrow f))$
 by (*metis* 0 *SBaAndEqv int-eq*)
show *?thesis* using *SBaSCSImpSCS[of f g]* *SBaSCSImpSCS[of g f]*
 1 by *fastforce*
qed

lemma *BaAndWPowerstarImport*:

$\vdash ba f \wedge wpowerstar g \wedge finite \longrightarrow wpowerstar (f \wedge g)$
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: $\vdash ba (g \longrightarrow f \wedge g) \wedge finite \longrightarrow wpowerstar g \longrightarrow wpowerstar (f \wedge g)$
 by (*rule* *BaWPowerstarImpWPowerstar*)
from 2 3 **show** *?thesis* by *fastforce*
qed

lemma *BaAndCSImport*:

$\vdash ba f \wedge g^* \wedge finite \longrightarrow (f \wedge g)^*$
 by (*metis* *BaAndWPowerstarImport Chopstar-WPowerstar WPowerstar-more-absorb int-eq*)

lemma *SBaAndSCSImpSCS*:

$\vdash sba f \wedge schopstar g \longrightarrow schopstar (f \wedge g)$
proof –
have 1: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ by *auto*
hence 2: $\vdash sba f \longrightarrow sba (g \longrightarrow f \wedge g)$ by (*rule* *SBaImpSBa*)
have 3: $\vdash sba (g \longrightarrow f \wedge g) \longrightarrow schopstar g \longrightarrow schopstar (f \wedge g)$ by (*rule* *SBaSCSImpSCS*)

from 2 3 show ?thesis by fastforce
qed

8.7 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 chop-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*:

$\vdash \text{len } k ; \# \text{False} \longrightarrow \# \text{False}$

by (*metis AndInfEqvChopFalse ChopImpChop InfEqvNotFinite int-iffD1 int-simps(21) inteq-reflection len-k-finite*)

lemma *LenSChopFalse*:

$\vdash \text{len } k \frown \# \text{False} \longrightarrow \# \text{False}$

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: $\vdash (\text{empty} \wedge \text{skip}; f) \longrightarrow \# \text{False}$

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: $\vdash (\text{len } k \wedge \text{len } (\text{Suc } k); f) \longrightarrow \text{len } k; \# \text{False}$

by (*metis 1 2 3 4 RightChopImpChop inteq-reflection*)

have 6: $\vdash \text{len } k ; \# \text{False} \longrightarrow \# \text{False}$

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*:
 $\vdash \neg(\text{len } k \wedge \text{len } (\text{Suc } k) \neg f)$
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*)

8.8 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*:

$\vdash (if_i (init\ w) \text{ then } (f;g) \text{ else } empty \wedge finite) =$
 $(if_i (init\ w) \text{ then } ((f \wedge finite);(g \wedge finite)) \text{ else } empty)$

proof –

have 1: $\vdash (if_i (init\ w) \text{ then } (f;g) \text{ else } empty \wedge finite) =$
 $(((init\ w \wedge (f;g)) \vee (\neg(init\ w) \wedge empty)) \wedge finite)$
by (*auto simp: ifthenelse-d-def*)
have 2: $\vdash (((init\ w \wedge (f;g)) \vee (\neg(init\ w) \wedge empty)) \wedge finite) =$
 $(((init\ w \wedge (f;g) \wedge finite) \vee (\neg(init\ w) \wedge empty \wedge finite)))$
by *auto*
have 3: $\vdash (init\ w \wedge (f;g) \wedge finite) = (init\ w \wedge (f \wedge finite);(g \wedge finite))$
using *ChopAndFiniteDist* **by** *fastforce*
have 4: $\vdash (\neg(init\ w) \wedge empty \wedge finite) = (\neg(init\ w) \wedge empty)$
using *FiniteAndEmptyEqvEmpty* **by** *auto*
have 5: $\vdash (((init\ w \wedge (f;g) \wedge finite) \vee (\neg(init\ w) \wedge empty \wedge finite))) =$
 $((init\ w \wedge (f \wedge finite);(g \wedge finite)) \vee (\neg(init\ w) \wedge empty))$
by (*metis 2 3 4 inteq-reflection*)
have 6: $\vdash ((init\ w \wedge (f \wedge finite);(g \wedge finite)) \vee (\neg(init\ w) \wedge empty)) =$
 $(if_i (init\ w) \text{ then } ((f \wedge finite);(g \wedge finite)) \text{ else } empty)$
by (*auto simp: ifthenelse-d-def*)
from 1 2 5 6 **show** ?thesis **by** (*metis inteq-reflection*)
qed

lemma *WhileEqvIf*:

$\vdash ((while\ (init\ w) \text{ do } f) \wedge finite) =$
 $(if_i (init\ w) \text{ then } (f; (while\ (init\ w) \text{ do } f)) \text{ else } empty \wedge finite)$

proof –

have 1: $\vdash (while\ (init\ w) \text{ do } f \wedge finite) =$
 $(((((init\ w) \wedge f)^* \wedge fin\ (\neg\ (init\ w))) \wedge finite)$
by (*simp add: while-d-def*)
have 2: $\vdash (init\ w \wedge f)^* = (empty \vee ((init\ w \wedge f); (init\ w \wedge f)^*))$
by (*metis CSEqvOrChopCSB ChopplusCommutate int-eq*)
have 21: $\vdash (((init\ w) \wedge f)^* \wedge fin\ (\neg\ (init\ w)) \wedge finite) =$
 $((empty \vee ((init\ w \wedge f); (init\ w \wedge f)^*)) \wedge fin\ (\neg\ (init\ w)) \wedge finite)$
using 2 **by** *fastforce*
have 22: $\vdash ((empty \vee ((init\ w \wedge f); (init\ w \wedge f)^*)) \wedge fin\ (\neg\ (init\ w)) \wedge finite) =$
 $((empty \wedge fin\ (\neg\ (init\ w)) \wedge finite) \vee$
 $((init\ w \wedge f); (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w)) \wedge finite))$
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: $\vdash ((\text{empty} \wedge \text{fin } (\neg(\text{init } w)) \wedge \text{finite}) \vee$
 $(((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))$
using 23 33 by auto
have 4: $\vdash (\text{init } w \wedge f); (\text{init } w \wedge f)^* = (\text{init } w \wedge (f; (\text{init } w \wedge f)^*))$
by (rule StateAndChop)
have 41: $\vdash (((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$
 $(\text{init } w \wedge (f; (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})$
using 4 by auto
have 42: $\vdash (\text{init } w \wedge ((f; (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})) =$
 $(\text{init } w \wedge ((f; (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))$
using Initprop(2) by (metis StateAndEmptyChop int-eq)
have 5: $\vdash ((f; ((\text{init } w \wedge f)^*)) \wedge (\text{fin } (\neg (\text{init } w))) \wedge \text{finite})$
 $= ((f \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge (\text{fin } (\neg (\text{init } w))) \wedge \text{finite}))$
using ChopAndFin by fastforce
hence 49: $\vdash (\text{init } w \wedge (f; (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$
 $(\text{init } w \wedge (f \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge (\text{fin } (\neg (\text{init } w))) \wedge \text{finite}))$
using 42 by fastforce
have 50: $\vdash (((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$
 $(\text{init } w \wedge (f \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge (\text{fin } (\neg (\text{init } w))) \wedge \text{finite}))$
by (meson 41 49 Prop04 lift-and-com)
have 51: $\vdash (\text{init } w \wedge (f \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge (\text{fin } (\neg (\text{init } w))) \wedge \text{finite})) =$
 $(\text{init } w \wedge (f \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge (\text{fin } (\neg (\text{init } w))) \wedge \text{finite}))$
by (metis (no-types) EmptyChop Initprop(2) integ-reflection)
have 52: $\vdash (((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$
 $(\text{init } w \wedge ((f \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})))$
using 50 51 by fastforce
have 53: $\vdash ((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((f \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})))$
using 52 34 by auto
have 6: $\vdash ((f \wedge \text{finite}); (((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite})) =$
 $(f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})$
by (simp add: while-d-def)
have 61: $\vdash (\text{init } w \wedge ((f \wedge \text{finite}); (((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite}))) =$
 $(\text{init } w \wedge ((f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})))$
using 6 by auto
have 62: $\vdash ((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((f \wedge \text{finite}); (((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite})))$
 $= ((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}))))$
using 61 by fastforce
have 7: $\vdash (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})$
 $= (((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge (f ; \text{while } (\text{init } w) \text{ do } f) \wedge \text{finite})))$
proof –
have $\vdash (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) =$

$(\neg \text{init } w \wedge \text{empty} \vee (\text{init } w \wedge f); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg \text{init } w) \wedge \text{finite})$
by (metis 1 23 34 inteq-reflection)
then show ?thesis
by (metis 21 22 34 52 ChopAndFiniteDist int-eq)
qed
have 71: $\vdash ((\text{if}_i (\text{init } w) \text{ then } (f; (\text{while } (\text{init } w) \text{ do } f)) \text{ else } \text{empty}) \wedge \text{finite}) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f; \text{while } (\text{init } w) \text{ do } f)) \wedge \text{finite}))$
using FiniteAndEmptyEqvEmpty **by** (auto simp: ifthenelse-d-def)
from 7 71 **show** ?thesis **by** fastforce
qed

lemma WPower-import-finite:
 $\vdash (\text{wpower } f \ k \wedge \text{finite}) = \text{wpower } (f \wedge \text{finite}) \ k$
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:
 $\vdash (\text{wpowerstar } f \wedge \text{finite}) = (\text{wpowerstar } (f \wedge \text{finite}))$
proof –
have 1: $\vdash (\text{wpowerstar } (f \wedge \text{finite})) \longrightarrow (\text{wpowerstar } f \wedge \text{finite})$
by (metis OrFiniteInf Prop12 SChopstar-finite SChopstar-WPowerstar WPowerstar-subdist inteq-reflection)
have 2: $\vdash \text{wpowerstar } f \wedge \text{finite} \longrightarrow (\text{wpowerstar } (f \wedge \text{finite}))$
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:
 $\vdash (\text{chopstar } f \wedge \text{finite}) = (\text{chopstar } (f \wedge \text{finite}))$
using Chopstar-WPowerstar[of f] Chopstar-WPowerstar[of LIFT (f \wedge 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* ($g \wedge f$)] 2 **by** *fastforce*

qed

lemma *SWhileEqvIf*:

$\vdash \text{swhile } (\text{init } w) \text{ do } f = \text{if}_i (\text{init } w) \text{ then } (f \frown (\text{swhile } (\text{init } w) \text{ do } f)) \text{ else } \text{empty}$

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* ($\text{init } w \wedge f$)] 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 swwhile-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: $\vdash (\text{if}_i (\text{init } w) \text{ then } ((f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } (f \wedge \text{finite}))) \text{ else } \text{empty} \wedge \text{finite}) =$
 $(\text{if}_i (\text{init } w) \text{ then } (f \frown (\text{swhile } (\text{init } w) \text{ do } f)) \text{ else } \text{empty})$

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: $\vdash (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}); h =$
 $\quad \text{if}_i (\text{init } w) \text{ then } ((g \wedge \text{finite}); ((\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}); h)) \text{ else } h$
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: $\vdash \text{if}_i (\text{init } w) \text{ then } ((g \wedge \text{finite}); ((\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}); h)) \text{ else } h =$
 $\quad \text{if}_i (\text{init } w) \text{ then } ((g \wedge \text{finite}); f) \text{ else } h$
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: $\vdash ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite});$
 $((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})))$
by (metis 61 7 81 84 inteq-reflection)
have 10: $\vdash ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$
 $((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite}$
by auto
hence 11: $\vdash ((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite});$
 $((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite})))$

$$(((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w))) \wedge finite)))$$
 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*:

$\vdash\ swhile\ (init\ w)\ do\ f = ((\neg (init\ w) \wedge empty) \vee (init\ w \wedge (f \wedge more) \frown swhile\ (init\ w)\ do\ f))$
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: $\vdash\ (f \wedge \neg (while\ (init\ w)\ do\ g \wedge finite)) =$

$$(f \wedge \neg(\neg (init\ w) \wedge empty) \vee$$

$$(init\ w \wedge ((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite))))$$

 by fastforce
 have 33: $\vdash\ (f \wedge \neg(\neg (init\ w) \wedge empty) \vee$

$$(init\ w \wedge ((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite)))) =$$

$$(f \wedge \neg(\neg (init\ w) \wedge empty) \wedge$$

$$\neg(init\ w \wedge ((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite))))$$

 by auto
 have 34: $\vdash\ (f \wedge \neg(\neg (init\ w) \wedge empty) \wedge$

$$\neg((init\ w) \wedge (((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite)))) =$$

$(f \wedge ((init\ w) \vee more) \wedge$
 $(\neg(init\ w) \vee \neg(((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite))))$
by (*auto simp: empty-d-def*)
have 35: $\vdash (f \wedge ((init\ w) \vee more) \wedge$
 $(\neg(init\ w) \vee \neg(((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite)))) =$
 $((f \wedge (init\ w) \wedge \neg(((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite))) \vee$
 $(f \wedge (init\ w) \wedge \neg(init\ w)) \vee$
 $(f \wedge more \wedge \neg(((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite))) \vee$
 $(f \wedge more \wedge \neg(init\ w)))$
by *auto*
have 36: $\vdash (f \wedge \neg (while\ (init\ w)\ do\ g \wedge finite)) =$
 $((f \wedge (init\ w) \wedge \neg(((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite))) \vee$
 $(f \wedge (init\ w) \wedge \neg(init\ w)) \vee$
 $(f \wedge more \wedge \neg(((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite))) \vee$
 $(f \wedge more \wedge \neg(init\ w)))$
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: $\vdash (f \wedge more \wedge \neg(((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite))) \longrightarrow$
 $((g \wedge more) \wedge finite); f \wedge$
 $\neg(((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite)))$
using 1 2 **by** (*auto simp: empty-d-def Valid-def*)
have 39: $\vdash (f \wedge (init\ w) \wedge \neg(((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite))) \longrightarrow$
 $((g \wedge more) \wedge finite); f \wedge$
 $\neg(((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite)))$
using 2 **by** *auto*
have 40: $\vdash ((f \wedge (init\ w) \wedge \neg(((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite))) \vee$
 $(f \wedge (init\ w) \wedge \neg(init\ w)) \vee$
 $(f \wedge more \wedge \neg(((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite))) \vee$
 $(f \wedge more \wedge \neg(init\ w))) \longrightarrow$
 $((g \wedge more) \wedge finite); f \wedge$
 $\neg(((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite)))$
using 39 38 37 38 **by** *fastforce*
have 4: $\vdash f \wedge \neg (while\ (init\ w)\ do\ g \wedge finite) \longrightarrow$
 $((g \wedge more) \wedge finite); f \wedge$
 $\neg(((g \wedge more) \wedge finite); (while\ (init\ w)\ do\ g \wedge finite)))$
by (*meson 36 40 Prop11 lift-imp-trans*)
have 50: $\vdash g \wedge more \longrightarrow more$
by *auto*
have 5: $\vdash (g \wedge more) \wedge finite \longrightarrow more$
by (*simp add: 50 Prop05 Prop07 finite-d-def*)
have 6: $\vdash f \wedge finite \longrightarrow (while\ (init\ w)\ do\ g \wedge finite)$
using 4 5 *ChopContraB* **by** *blast*
from 6 **show** ?thesis **by** (*simp add: Prop12*)
qed

lemma *SWhileIntro*:
assumes $\vdash \neg (init\ w) \wedge f \longrightarrow empty$
 $\vdash init\ w \wedge f \longrightarrow (g \wedge more) \frown f$
shows $\vdash f \wedge finite \longrightarrow swhile\ (init\ w)\ do\ g$

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* ($\text{init } w$) g]
While-import-finite[of *LIFT* ($\text{init } w$) 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: $\vdash (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) \wedge \neg g \longrightarrow$
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})) \wedge \neg g$
using 11 21 **by** *fastforce*

have 3: $\vdash (\text{init } w) \wedge (((f \wedge \text{more}) \wedge \text{finite}); g) \longrightarrow g$
using *assms* **by** *blast*

hence 31: $\vdash \neg g \longrightarrow \neg((\text{init } w) \wedge (((f \wedge \text{more}) \wedge \text{finite}); g))$
by *fastforce*

have 32: $\vdash (\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})) \wedge \neg g \longrightarrow$
 $((((f \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})) \wedge$
 $\neg (((f \wedge \text{more}) \wedge \text{finite}); g)) \wedge \neg g$

using 31 **by** *fastforce*

have 4: $\vdash (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) \wedge \neg g \longrightarrow$
 $((f \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})) \wedge$

$\neg ((f \wedge \text{more}) \wedge \text{finite}); g)$
by (*meson 23 32 Prop12 lift-imp-trans*)
have 5: $\vdash (f \wedge \text{more}) \wedge \text{finite} \longrightarrow \text{more}$
by *auto*
from 4 5 **show** *?thesis* **using**
ChopContraB[of LIFT(while (init w) do f \wedge finite) LIFT(g) LIFT(((f \wedge more) \wedge finite))]
by *auto*
qed

lemma *SWhileElim*:

assumes $\vdash \neg (\text{init } w) \wedge \text{empty} \longrightarrow g$
 $\vdash \text{init } w \wedge (f \wedge \text{more}) \wedge g \longrightarrow g$
shows $\vdash \text{swhile } (\text{init } w) \text{ do } f \longrightarrow g$
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*:

$\vdash \text{ba } (f \longrightarrow g) \wedge \text{finite} \longrightarrow (\text{while } (\text{init } w) \text{ do } f) \longrightarrow (\text{while } (\text{init } w) \text{ do } g)$
proof –
have 1: $\vdash (f \longrightarrow g) \longrightarrow ((\text{init } w \wedge f) \longrightarrow (\text{init } w \wedge g))$
by *auto*
hence 2: $\vdash \text{ba } (f \longrightarrow g) \longrightarrow \text{ba } ((\text{init } w \wedge f) \longrightarrow (\text{init } w \wedge g))$
using *BaImpBa* **by** *blast*
have 3: $\vdash \text{ba } ((\text{init } w \wedge f) \longrightarrow (\text{init } w \wedge g)) \wedge \text{finite} \longrightarrow ((\text{init } w \wedge f)^* \longrightarrow (\text{init } w \wedge g)^*)$
by (*rule BaCSImpCS*)
have 4: $\vdash \text{ba } (f \longrightarrow g) \wedge \text{finite} \longrightarrow ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)))$
 $\longrightarrow (\text{init } w \wedge g)^* \wedge \text{fin } (\neg (\text{init } w)))$
using 2 3 **by** *fastforce*
from 4 **show** *?thesis* **by** (*simp add: while-d-def*)
qed

lemma *SBaSWhileImpSWhile*:

$\vdash \text{sba } (f \longrightarrow g) \longrightarrow (\text{swhile } (\text{init } w) \text{ do } f) \longrightarrow (\text{swhile } (\text{init } w) \text{ do } g)$
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

```

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.

9 Omega and variants

9.1 Definitions

9.1.1 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*)
 $(\text{metis enat-0-iff}(1) \text{ linorder-not-less ndropn-all neq0-conv}$
 $\text{ nfinite-nlength-enat nfinite-ntaken ntaken-all order-le-less})$

coinductive *omega-d* :: ('a::world) formula \Rightarrow 'a formula

for *F* **where**

$(s \models ((F \wedge \text{more}) \wedge \text{finite});(\text{omega-d } F)) \Longrightarrow (s \models (\text{omega-d } F))$

syntax

-omega-d :: lift \Rightarrow lift $((-\omega) [85] 85)$

syntax (*ASCII*)

-omega-d :: lift \Rightarrow lift $((\text{omega } -) [85] 85)$

translations

-omega-d \rightleftharpoons *CONST omega-d*

lemma *OmegaIntros*:

$\vdash ((f \wedge \text{more}) \wedge \text{finite});(\text{omega } f) \longrightarrow (\text{omega } f)$

using *omega-d.intros* **using** *unl-lift2* **by** *blast*

lemma *OmegaCases*:

$(w \models (\text{omega } F)) \Longrightarrow$

$(w \models ((F \wedge \text{more}) \wedge \text{finite});(\text{omega } F) \Longrightarrow P) \Longrightarrow P$

using *omega-d.cases*[*of F w P*] **by** *auto*

lemma *OmegaUnrollSem*:

$(s \models (\text{omega } f)) = (s \models ((f \wedge \text{more}) \wedge \text{finite});(\text{omega } f))$

using *omega-d.cases*[*of f*]

by (*metis omega-d.simps*)

lemma *OmegaCoinductSem*:

assumes $\bigwedge x. X \ x \Longrightarrow x \models ((F \wedge \text{more}) \wedge \text{finite});(X \ \vee \ \text{omega } F)$

shows $x \models X \longrightarrow \text{omega } F$

using *assms omega-d.coinduct*[*of X x F*]

by (*auto simp add: chop-defs*)

lemma *OmegaWeakCoinductSem*:

assumes $\bigwedge x. X \ x \Longrightarrow x \models ((F \wedge \text{more}) \wedge \text{finite});X$

shows $x \models X \longrightarrow \text{omega } F$

using *assms omega-d.coinduct*[*of X x F*]

by (*auto simp add: chop-defs*)

lemma *OmegaUnroll*:

$\vdash f^\omega = ((f \wedge \text{more}) \wedge \text{finite});f^\omega$

using *OmegaUnrollSem unl-lift2* **by** *blast*

lemma *OmegaCoinduct*:
assumes $\vdash X \longrightarrow ((F \wedge \text{more}) \wedge \text{finite}); (X \vee (\text{omega } F))$
shows $\vdash X \longrightarrow (\text{omega } F)$
using *assms OmegaCoinductSem*[of $X F$]
by (*simp add: Valid-def*)

lemma *OmegaWeakCoinduct*:
assumes $\vdash X \longrightarrow ((F \wedge \text{more}) \wedge \text{finite}); X$
shows $\vdash X \longrightarrow (\text{omega } F)$
using *assms OmegaWeakCoinductSem*[of $X F$]
by (*simp add: Valid-def*)

9.1.2 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: $\bigwedge j. (\text{nnth } l \ j) < (\text{nnth } l \ (\text{Suc } j))$
by (*metis assms(1) assms(2) linorder-le-cases nfinite-ntaken nidx-expand ntaken-all*)
have 2: $\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } (\text{NCons } x \ \text{nell}) \implies \text{False}$
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

definition *aomega-d* :: $('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$
where *aomega-d* $F \equiv$
 $(\lambda s.$
 $(\exists (l::\text{nat nellist}).$
 $\neg \text{nfinite } l \wedge (\text{nnth } l \ 0) = 0 \wedge \text{nidx } l \wedge$
 $(\forall i. (\text{nnth } l \ i) \leq \text{nlength } s) \wedge$

$(\forall i. ((nsubn\ s\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i))) \models F))$
 $)$
 $)$

syntax

-aomega-d :: *lift* \Rightarrow *lift* ((*aomega* -) [85] 85)

syntax (*ASCII*)

-aomega-d :: *lift* \Rightarrow *lift* ((*aomega* -) [85] 85)

translations

-aomega-d \Rightarrow *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*: $\neg nfinite\ l \wedge (nnth\ l\ 0) = 0 \wedge nidx\ l \wedge$
 $(\forall i. (nnth\ l\ i) \leq nlength\ \sigma) \wedge (\forall i. f\ ((nsubn\ \sigma\ (nnth\ l\ i)\ (nnth\ l\ (Suc\ i)))))$

using *assms* **by** *auto*

have *2*: *l* = *NCons* (*nnth* *l* 0) (*ndropn* 1 *l*)

by (*metis* 1 *One-nat-def* *gr-zeroI* *ndropn-0* *ndropn-Suc-conv-ndropn* *nlength-eq-enat-nfiniteD*
zero-enat-def)

have *3*: (*nnth* *l* 0) = 0

using 1 **by** *blast*

have *4*: (*nnth* *l* 0) < (*nnth* *l* 1)

using 1

by (*metis* *One-nat-def* *Suc-ile-eq* *enat-defs*(1) *nidx-gr-first* *nlength-eq-enat-nfiniteD*
not-gr-zero *zero-less-one*)

have *5*: *nidx* (*ndropn* 1 *l*)

using 1 *nidx-expand*[*of* *l*] *nidx-expand*[*of* (*ndropn* 1 *l*)]

by (*metis* *dual-order.order-iff-strict* *enat-defs*(1) *ndropn-all* *ndropn-nnth* *nfinite-ndropn-b*
nlength-NNil *nlength-eq-enat-nfiniteD* *not-le-imp-less* *plus-1-eq-Suc*)

have *6*: *f* ((*nsubn* σ (*nnth* *l* 0) (*nnth* *l* 1)))

by (*metis* 1 *One-nat-def*)

have *7*: $(\forall i. f\ ((nsubn\ \sigma\ (nnth\ (ndropn\ 1\ l)\ i)\ (nnth\ (ndropn\ 1\ l)\ (Suc\ i))))$

by (*simp* *add*: 1)

have 8: $f \ (\ (ntaken \ (nnth \ l \ 1) \ \sigma))$
by (metis 1 4 One-nat-def Suc-diff-1 Suc-diff-Suc ndropn-0 nsubn-def1)
have 81: $\neg \ nfinite \ (nmap \ (\lambda \ e. \ e - (nnth \ l \ 1)) \ (ndropn \ 1 \ l))$
by (simp add: 1)
have 82: $\bigwedge j. \ 0 < j \longrightarrow (nnth \ l \ 1) < nnth \ (ndropn \ 1 \ l) \ j$
by (metis 1 Suc-diff-1 enat-defs(1) linorder-le-cases ndropn-all ndropn-nnth nfinite-ndropn-b
nidx-less nlength-NNil nlength-eq-enat-nfiniteD plus-1-eq-Suc)
have 83: $\bigwedge j. \ nnth \ (nmap \ (\lambda e. \ e - nnth \ l \ 1) \ (ndropn \ 1 \ l)) \ j =$
 $(nnth \ l \ (Suc \ j)) - (nnth \ l \ 1)$
by (metis 81 le-cases ndropn-nnth nfinite-ntaken nlength-nmap nnth-nmap ntaken-all plus-1-eq-Suc)
have 84: $\bigwedge j. \ nnth \ (nmap \ (\lambda e. \ e - nnth \ l \ 1) \ (ndropn \ 1 \ l)) \ (Suc \ j) =$
 $(nnth \ l \ (Suc \ (Suc \ j))) - (nnth \ l \ 1)$
using 83 **by** blast
have 840: $\bigwedge j. \ (nnth \ l \ 1) \leq (nnth \ l \ (Suc \ j))$
by (metis 1 diff-add-zero diff-is-0-eq linorder-le-cases nfinite-ntaken nidx-less-eq ntaken-all
plus-1-eq-Suc)
have 85: $\bigwedge j. \ (nnth \ l \ (Suc \ j)) - (nnth \ l \ 1) < (nnth \ l \ (Suc \ (Suc \ j))) - (nnth \ l \ 1)$
by (metis 1 840 diff-less-mono linorder-le-cases nfinite-ntaken nidx-expand ntaken-all)
have 86: $(\forall i. \ enat \ (Suc \ i) \leq nlength \ (nmap \ (\lambda e. \ e - nnth \ l \ 1) \ (ndropn \ 1 \ l)) \longrightarrow$
 $nnth \ (nmap \ (\lambda e. \ e - nnth \ l \ 1) \ (ndropn \ 1 \ l)) \ i <$
 $nnth \ (nmap \ (\lambda e. \ e - nnth \ l \ 1) \ (ndropn \ 1 \ l)) \ (Suc \ i))$
using 83 85 **by** presburger
have 9: $nidx \ (nmap \ (\lambda \ e. \ e - (nnth \ l \ 1)) \ (ndropn \ 1 \ l))$
using nidx-expand[of (nmap (λ e. e - (nnth l 1)) (ndropn 1 l))]
using 83 85 **by** presburger
have 91: $\bigwedge j. \ (nnth \ (nmap \ (\lambda e. \ e + (nnth \ l \ 1)) \ (nmap \ (\lambda \ e. \ e - (nnth \ l \ 1)) \ (ndropn \ 1 \ l))) \ j) =$
 $(nnth \ l \ (Suc \ j))$
by (metis 81 82 83 One-nat-def add-Suc diff-Suc-1 diff-Suc-eq-diff-pred
diff-add diff-is-0-eq le-refl linorder-le-cases ndropn-nnth nfinite-ntaken nlength-nmap
nnth-nmap not-le-imp-less ntaken-all order-less-imp-le plus-1-eq-Suc)
have 10: $(\forall i. \ f \ (\ (nsubn \ \sigma$
 $(nnth \ (nmap \ (\lambda e. \ e + (nnth \ l \ 1)) \ (nmap \ (\lambda \ e. \ e - (nnth \ l \ 1)) \ (ndropn \ 1 \ l))) \ i)$
 $(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: $(\forall i. \ f \ (\ (nsubn \ \sigma$
 $(\ (nnth \ (nmap \ (\lambda \ e. \ e - (nnth \ l \ 1)) \ (ndropn \ 1 \ l)) \ i) \ + \ (nnth \ l \ 1) \)$
 $(\ (nnth \ (nmap \ (\lambda \ e. \ e - (nnth \ l \ 1)) \ (ndropn \ 1 \ l)) \ (Suc \ i)) \ + \ (nnth \ l \ 1) \)$
 $)) \)$
using 7 83 840 **by** fastforce
have 12: $(\forall i. \ f \ (\ (nsubn \ (ndropn \ (nnth \ l \ 1) \ \sigma)$
 $(\ (nnth \ (nmap \ (\lambda \ e. \ e - (nnth \ l \ 1)) \ (ndropn \ 1 \ l)) \ i) \)$
 $(\ (nnth \ (nmap \ (\lambda \ e. \ e - (nnth \ l \ 1)) \ (ndropn \ 1 \ l)) \ (Suc \ i)) \)$
 $)) \)$
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: $\neg \ nfinite \ (nmap \ (\lambda \ e. \ e - (nnth \ l \ 1)) \ (ndropn \ 1 \ l)) \wedge$
 $nnth \ (nmap \ (\lambda \ e. \ e - (nnth \ l \ 1)) \ (ndropn \ 1 \ l)) \ 0 = 0 \wedge$
 $nidx \ (nmap \ (\lambda \ e. \ e - (nnth \ l \ 1)) \ (ndropn \ 1 \ l)) \wedge$
 $(\forall i. \ f \ (\ (nsubn \ (ndropn \ (nnth \ l \ 1) \ \sigma)$

```

      ( (nnth (nmap (λ e. e - (nnth l 1)) (ndropn 1 l)) i) )
      ( (nnth (nmap (λ e. e - (nnth l 1)) (ndropn 1 l)) (Suc i)) )
    ) ) )
  using 12 121 81 9 by blast
have 123: (∀ i. (nnth (nmap (λ e. e - (nnth l 1)) (ndropn 1 l)) i) ≤ nlength (ndropn (nnth l 1) σ))
  by (meson 1 enat-ile infinite-nidx-imp-infinite-interval le-cases nfinite-ndropn-b
    nlength-eq-enat-nfiniteD)
have 13: (∃ ls.
  ¬ nfinite ls ∧ (nnth ls 0) = 0 ∧ nidx ls ∧
  (∀ i. (nnth ls i) ≤ nlength (ndropn (nnth l 1) σ)) ∧
  (∀ i. f ( (nsubn (ndropn (nnth l 1) σ) (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 (∃ n. enat n ≤ nlength σ ∧
  f ( (ntaken n σ)) ∧
  0 < n ∧
  enat 0 < nlength σ ∧
  (∃ l.
    ¬ nfinite l ∧ (nnth l 0) = 0 ∧ nidx l ∧
    (∀ i. (nnth l i) ≤ nlength σ - enat n) ∧
    (∀ i. f ( (nsubn (ndropn n σ) (nnth l i) (nnth l (Suc i))))))
  )
)
shows (∃ l. ¬ nfinite l ∧ (nnth l 0) = 0 ∧ nidx l ∧
  (∀ i. (nnth l i) ≤ nlength σ) ∧
  (∀ i. f ( (nsubn σ (nnth l i) (nnth l (Suc i))))))
proof -
obtain n where 1: enat n ≤ nlength σ ∧ f ( (ntaken n σ)) ∧
  0 < n ∧ enat 0 < nlength σ ∧
  (∃ l.
    ¬ nfinite l ∧ (nnth l 0) = 0 ∧ nidx l ∧
    (∀ i. (nnth l i) ≤ nlength σ - enat n) ∧
    (∀ i. f ( (nsubn (ndropn n σ) (nnth l i) (nnth l (Suc i))))))
  )
  using assms by auto
have 2: (∃ l.
  ¬ nfinite l ∧ (nnth l 0) = 0 ∧ nidx l ∧
  (∀ i. (nnth l i) ≤ nlength (ndropn n σ)) ∧
  (∀ i. f ( (nsubn (ndropn n σ) (nnth l i) (nnth l (Suc i))))))
)
  using 1 by auto
obtain l where 3: ¬ nfinite l ∧ (nnth l 0) = 0 ∧ nidx l ∧
  (∀ i. (nnth l i) ≤ nlength (ndropn n σ)) ∧
  (∀ i. f ( (nsubn (ndropn n σ) (nnth l i) (nnth l (Suc i))))))

```

```

using 2 by auto
have 4:  $\text{nidx } (nmap (\lambda e. e + n) l)$ 
  using 3
  by (simp add: Suc-ile-eq nidx-expand)
have 42:  $\bigwedge j. \text{nnth } (NCons\ 0\ (nmap (\lambda e. e + n) l))\ j =$ 
  (case j of 0  $\Rightarrow$  0 |
    (Suc k  $\Rightarrow$  nnth (nmap (λe. e + n) l) k))
  by (simp add: nnth-NCons)
have 43:  $\bigwedge j. \text{nnth } (NCons\ 0\ (nmap (\lambda e. e + n) l))\ (Suc\ j) =$ 
   $\text{nnth } (nmap (\lambda e. e + n) l)\ j$ 
  by simp
have 44:  $0 < \text{nnth } (nmap (\lambda e. e + n) l)\ 0$ 
  using 1 enat-defs(1) by auto
have 45:  $\bigwedge j. \text{nnth } (nmap (\lambda e. e + n) l)\ j < \text{nnth } (nmap (\lambda e. e + n) l)\ (Suc\ j)$ 
  by (metis 3 4 add.right-neutral le-cases nfinite-ntaken nidx-less nlength-nmap ntaken-all)
have 46:  $(\forall i. \text{enat } (Suc\ i) \leq \text{nlength } (NCons\ 0\ (nmap (\lambda e. e + n) l))) \longrightarrow$ 
   $\text{nnth } (NCons\ 0\ (nmap (\lambda e. e + n) l))\ i <$ 
   $\text{nnth } (NCons\ 0\ (nmap (\lambda e. e + n) l))\ (Suc\ i)$ 
  by (metis 43 44 45 lessI less-Suc-eq-0-disj nnth-0)
have 5:  $\text{nidx } (NCons\ 0\ (nmap (\lambda e. e + n) l))$ 
  using 46 nidx-expand by blast
have 6:  $(\text{ntaken } n\ \sigma) =$ 
   $(\text{nsubn } \sigma\ (\text{nnth } (NCons\ 0\ (nmap (\lambda e. e + n) l))\ 0)$ 
   $\quad (\text{nnth } (NCons\ 0\ (nmap (\lambda e. e + n) l))\ (Suc\ 0)))$ 
  by (metis 3 43 44 One-nat-def Suc-diff-1 Suc-diff-Suc add-0 ndropn-0 nnth-0 nnth-nmap
    nsubn-def1 zero-enat-def zero-le)
have 7:  $(\forall i. f\ (\text{nsubn } (ndropn\ n\ \sigma)\ (\text{nnth } l\ i)\ (\text{nnth } l\ (Suc\ i))))$ 
  using 3 by auto
have 8:  $(\forall i. f\ (\text{nsubn } \sigma\ ((\text{nnth } l\ i) + n)\ ((\text{nnth } l\ (Suc\ i)) + n)))$ 
  using 7
  by (simp add: add commute ndropn-ndropn nsubn-def1)
have 9:  $(\forall i. f\ (\text{nsubn } \sigma\ (\text{nnth } (nmap (\lambda e. e + n) l)\ i)$ 
   $\quad (\text{nnth } (nmap (\lambda e. e + n) l)\ (Suc\ i))))$ 
  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:  $f\ (\text{nsubn } \sigma\ (\text{nnth } (NCons\ 0\ (nmap (\lambda e. e + n) l))\ 0)$ 
   $\quad (\text{nnth } (NCons\ 0\ (nmap (\lambda e. e + n) l))\ (Suc\ 0))))$ 
  using 1 6 by auto
have 11:  $(\forall i. i > 0 \longrightarrow$ 
   $f\ (\text{nsubn } \sigma\ (\text{nnth } (NCons\ 0\ (nmap (\lambda e. e + n) l))\ i)$ 
   $\quad (\text{nnth } (NCons\ 0\ (nmap (\lambda e. e + n) l))\ (Suc\ i))))$ 
  by (metis 9 Suc-diff-1 nnth-Suc-NCons)
have 12:  $(\forall i.$ 
   $f\ (\text{nsubn } \sigma\ (\text{nnth } (NCons\ 0\ (nmap (\lambda e. e + n) l))\ i)$ 
   $\quad (\text{nnth } (NCons\ 0\ (nmap (\lambda e. e + n) l))\ (Suc\ i))))$ 
  using 10 11 neq0-conv by blast
have 13:  $\forall i. \text{nnth } (NCons\ 0\ (nmap (\lambda e. e + n) l))\ i \leq \text{nlength } \sigma$ 
  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*:

$$\begin{aligned}
& (\exists l. \neg \text{nfinite } l \wedge (\text{nnth } l \ 0) = 0 \wedge \text{nidx } l \wedge \\
& \quad (\forall i. (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge \\
& \quad (\forall i. f \ (\ (\text{nsubn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)))))) \\
& = \\
& (\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge \\
& \quad f \ ((\text{ntaken } n \ \sigma)) \wedge \\
& \quad 0 < n \wedge \\
& \quad \text{enat } 0 < \text{nlength } \sigma \wedge \\
& \quad (\exists l. \\
& \quad \quad \neg \text{nfinite } l \wedge (\text{nnth } l \ 0) = 0 \wedge \text{nidx } l \wedge \\
& \quad \quad (\forall i. (\text{nnth } l \ i) \leq \text{nlength } \sigma - \text{enat } n) \wedge \\
& \quad \quad (\forall i. f \ (\ (\text{nsubn } (\text{ndropn } n \ \sigma) \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)))))) \\
& \quad) \\
&)
\end{aligned}$$

using *aomega-unroll-chain-a*[of σ f] *aomega-unroll-chain-b*[of σ f]
by *blast*

lemma *aomega-unroll-sem*:

$(\sigma \models ((f \wedge \text{more}) \wedge \text{finite}); (\text{aomega } f) = (\text{aomega } f))$

proof

(simp add: itl-defs zero-enat-def aomega-d-def)

show $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $f \ (\text{ntaken } n \ \sigma) \wedge$
 $0 < n \wedge$
 $\text{enat } 0 < \text{nlength } \sigma \wedge$
 $(\exists l. \neg \text{nfinite } l \wedge \text{nnth } l \ 0 = 0 \wedge \text{nidx } l \wedge$
 $(\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma - \text{enat } n) \wedge$
 $(\forall i. f \ (\text{nsubn } (\text{ndropn } n \ \sigma) \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)))))) \vee$
 $\neg \text{nfinite } \sigma \wedge f \ \sigma \wedge \text{enat } 0 < \text{nlength } \sigma \wedge \text{nfinite } \sigma) =$
 $(\exists l. \neg \text{nfinite } l \wedge \text{nnth } l \ 0 = 0 \wedge \text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. f \ (\text{nsubn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))))))$

using *aomega-unroll-chain*[of σ f] **by** *blast*

qed

lemma *AOmegaUnroll*:

$\vdash (\text{aomega } f) = ((f \wedge \text{more}) \wedge \text{finite}); (\text{aomega } f)$

unfolding *Valid-def*

using *aomega-unroll-sem* **by** *auto*

lemma *AOmegaInductSem-help*:

$(\sigma \models \text{inf } \wedge g \wedge \Box(g \longrightarrow ((f \wedge \text{more}) \wedge \text{finite}); g)) =$
 $(\neg \text{nfinite } \sigma \wedge g \ \sigma \wedge$
 $(\forall n. g \ (\ (\text{ndropn } n \ \sigma)) \longrightarrow$
 $(\exists na. f \ (\ (\text{nsubn } \sigma \ n \ (\text{na} + n)))) \wedge$

$0 < na \wedge g \ (\ (ndropn \ (n + na) \ \sigma))))$

)

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 σ = 0
| *cpoint* *f g* (Suc *n*) σ =
 $(\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 σ) = 0
by *simp*

lemma *cpoint-expand-1*:
(cpoint f g 1 σ) =
(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*:
(cpoint f g (Suc n) σ) =
(SOME x. ($\exists \ m. f \ (\ (nsubn \ \sigma \ (cpoint \ f \ g \ n \ \sigma) \ (m+(cpoint \ f \ g \ n \ \sigma))$)
 $\wedge \ m > 0 \wedge g \ ((ndropn \ (m+(cpoint \ f \ g \ n \ \sigma)) \ \sigma))$
 $\wedge \ x=m+(cpoint \ f \ g \ n \ \sigma))$
 $)$
by (*simp add: itl-defs*)

lemma *cpoint-0*:
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 $g \ ((ndropn \ (cpoint \ f \ g \ i \ \sigma) \ \sigma))$
proof
(induct i)
case 0
then show ?case **by** (*simp add: assms*)

```

next
case (Suc i)
then show ?case
proof -
  have 1:  $g ((ndropn (cpoint f g i \sigma) \sigma))$ 
  by (simp add: Suc.hyps)
  have 2:  $g ((ndropn (cpoint f g i \sigma) \sigma)) \longrightarrow$ 
     $(\exists m. f ( (nsubn \sigma (cpoint f g i \sigma) (m+(cpoint f g i \sigma)))) \wedge$ 
     $0 < m \wedge g ((ndropn (m+(cpoint f g i \sigma)) \sigma)))$ 
  using assms by blast
  have 3:  $(\exists m. f ( (nsubn \sigma (cpoint f g i \sigma) (m+(cpoint f g i \sigma)))) \wedge$ 
     $0 < m \wedge g ( (ndropn (m+(cpoint f g i \sigma)) \sigma)))$ 
  using 1 2 by auto
  have 4:  $(cpoint f g (Suc i) \sigma) =$ 
     $(SOME x. (\exists m. f ( (nsubn \sigma (cpoint f g i \sigma) (m+(cpoint f g i \sigma))))$ 
     $\wedge m > 0 \wedge g ((ndropn (m+(cpoint f g i \sigma)) \sigma))$ 
     $\wedge x = m + (cpoint f g i \sigma)))$ 
  by simp
  have 5:  $g ((ndropn ((cpoint f g (Suc i) \sigma) \sigma))$ 
  using 3 4 someI-ex[of  $\lambda x. (\exists m. f ( (nsubn \sigma (cpoint f g i \sigma) (m+(cpoint f g i \sigma))))$ 
     $\wedge m > 0 \wedge g ((ndropn (m+(cpoint f g i \sigma)) \sigma))$ 
     $\wedge x = m + (cpoint f g i \sigma))]$  by auto
  from 5 show ?thesis by auto
qed
qed

```

lemma cpoint-1:

```

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 (  $g ((ndropn (cpoint f g i \sigma) \sigma))$ 
 $\implies g ((ndropn (cpoint f g (Suc i) \sigma) \sigma))$  )

```

using assms cpoint-0 by blast

lemma cpoint-2:

```

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  $f ( (nsubn \sigma (cpoint f g i \sigma) (cpoint f g (Suc i) \sigma)))$ 

```

proof

(induct i)

case 0

then show ?case

proof –

have 1: $g ((ndropn\ 0\ \sigma))$
using *assms cpoint-0 cpoint-expand-0* **by** *force*
have 2: $(\exists m. f ((nsbn\ \sigma\ (cpoint\ f\ g\ 0\ \sigma)\ (m+(cpoint\ f\ g\ 0\ \sigma)))) \wedge$
 $0 < m \wedge g ((ndropn\ (m+(cpoint\ f\ g\ 0\ \sigma))\ \sigma))$
using *assms 1* **by** *(metis cpoint-expand-0)*
have 3: $(cpoint\ f\ g\ 1\ \sigma) =$
 $(SOME\ x. (\exists m. f ((nsbn\ \sigma\ (cpoint\ f\ g\ 0\ \sigma)\ (m+(cpoint\ f\ g\ 0\ \sigma))))$
 $\wedge m > 0 \wedge g ((ndropn\ (m+(cpoint\ f\ g\ 0\ \sigma))\ \sigma))$
 $\wedge x = m + (cpoint\ f\ g\ 0\ \sigma))$
 $)$

by *simp*

have 4: $f ((nsbn\ \sigma\ (cpoint\ f\ g\ 0\ \sigma)\ ((cpoint\ f\ g\ 1\ \sigma))))$
using 2 3 *someI-ex[of $\lambda x. (\exists m. f ((nsbn\ \sigma\ (cpoint\ f\ g\ 0\ \sigma)$*
 $(m+(cpoint\ f\ g\ 0\ \sigma))))$
 $\wedge m > 0 \wedge g ((ndropn\ (m+(cpoint\ f\ g\ 0\ \sigma))\ \sigma))$
 $\wedge x = m + (cpoint\ f\ g\ 0\ \sigma)]$ **by** *auto*

from 4 **show** *?thesis* **by** *auto*

qed

next

case *(Suc i)*

then show *?case*

proof –

have n1: $g ((ndropn\ (cpoint\ f\ g\ (Suc\ i)\ \sigma)\ \sigma))$
using *assms cpoint-0* **by** *blast*
have n2: $(\exists m. f ((nsbn\ \sigma\ (cpoint\ f\ g\ (Suc\ i)\ \sigma)\ (m+(cpoint\ f\ g\ (Suc\ i)\ \sigma)))) \wedge$
 $0 < m \wedge g ((ndropn\ (m+(cpoint\ f\ g\ (Suc\ i)\ \sigma))\ \sigma))$
using *assms n1* **by** *auto*
have n3: $(cpoint\ f\ g\ (Suc\ (Suc\ i))\ \sigma) =$
 $(SOME\ x. (\exists m. f ((nsbn\ \sigma\ (cpoint\ f\ g\ (Suc\ i)\ \sigma)\ (m+(cpoint\ f\ g\ (Suc\ i)\ \sigma))))$
 $\wedge m > 0 \wedge g ((ndropn\ (m+(cpoint\ f\ g\ (Suc\ i)\ \sigma))\ \sigma))$
 $\wedge x = m + (cpoint\ f\ g\ (Suc\ i)\ \sigma))$
 $)$

using *cpoint-expand-n* **by** *blast*

have n4: $f ((nsbn\ \sigma\ (cpoint\ f\ g\ (Suc\ i)\ \sigma)\ ((cpoint\ f\ g\ (Suc\ (Suc\ i))\ \sigma))))$
using n2 n3 *someI-ex[of $\lambda x. (\exists m. f ((nsbn\ \sigma\ (cpoint\ f\ g\ (Suc\ i)\ \sigma)$*
 $(m+(cpoint\ f\ g\ (Suc\ i)\ \sigma))))$
 $\wedge m > 0 \wedge g ((ndropn\ (m+(cpoint\ f\ g\ (Suc\ i)\ \sigma))\ \sigma))$
 $\wedge x = m + (cpoint\ f\ g\ (Suc\ i)\ \sigma)]$ **by** *auto*

from n4 **show** *?thesis* **by** *auto*

qed

qed

lemma *cpoint-3a:*

$m > 0 \wedge x = m + (cpoint\ f\ g\ (Suc\ i)\ \sigma) \implies (cpoint\ f\ g\ (Suc\ i)\ \sigma) < x$

by *auto*

lemma *cpoint-3:*

```

assumes  $\neg \text{nfinit} \sigma \wedge g \ \sigma \wedge$ 
            $(\forall k. g \ ( \text{ndropn } k \ \sigma)) \longrightarrow$ 
            $(\exists m. f \ ( \text{nsubn } \sigma \ k \ (m+k))) \wedge$ 
            $0 < m \wedge g \ ( \text{ndropn } (m+k) \ \sigma))$ 

shows  $(\text{cpoint } f \ g \ i \ \sigma) < (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma)$ 

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:  $(\text{cpoint } f \ g \ 0 \ \sigma) < (\text{cpoint } f \ g \ 1 \ \sigma)$ 
    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:  $(\text{cpoint } f \ g \ (\text{Suc } (\text{Suc } i)) \ \sigma) =$ 
            $(\text{SOME } x. (\exists m. f \ ( \text{nsubn } \sigma \ (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma) \ (m+(\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma))))$ 
            $\wedge m > 0 \wedge g \ ((\text{ndropn } (m+(\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma)) \ \sigma))$ 
            $\wedge x = m + (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma))$ 
           )
    using cpoint-expand-n by blast
  have n4:  $(\exists m. f \ ( \text{nsubn } \sigma \ (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma) \ (m+(\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma))))$ 
            $\wedge m > 0 \wedge g \ ((\text{ndropn } (m+(\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma)) \ \sigma))$ 
            $\wedge (\text{cpoint } f \ g \ (\text{Suc } (\text{Suc } i)) \ \sigma) = m + (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma)$ 
    using n2 n3 someI-ex[of  $\lambda x. (\exists m. f \ ( \text{nsubn } \sigma \ (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma) \ (m+(\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma))))$ 
            $\wedge m > 0 \wedge g \ ((\text{ndropn } (m+(\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma)) \ \sigma))$ 
            $\wedge (\text{cpoint } f \ g \ (\text{Suc } (\text{Suc } i)) \ \sigma) = m + (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma)$ ]

```



```

       $\wedge x=m+(cpoint\ f\ g\ (Suc\ i)\ \sigma))\ ]\ \mathbf{by}\ auto$ 
have n5:  $(cpoint\ f\ g\ (Suc\ i)\ \sigma) < (cpoint\ f\ g\ (Suc\ (Suc\ i))\ \sigma)$ 
using n4 using cpoint-3a by blast
from n5 show ?thesis by auto
qed
qed

```

```

primcorec cpl :: ('a::world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  nat  $\Rightarrow$  'a intervals  $\Rightarrow$  nat nellist
where  $cpl\ f\ g\ x\ \sigma = NCons\ (cpoint\ f\ g\ x\ \sigma)\ (cpl\ f\ g\ (Suc\ x)\ \sigma)$ 

```

```

lemma
   $nnth\ (cpl\ f\ g\ 0\ \sigma)\ 0 = 0$ 
by (metis cpl.disc-iff cpl.simps(2) cpoint-expand-0 nhd-conv-nnth)

```

```

lemma
   $nnth\ (cpl\ f\ g\ 0\ \sigma)\ 1 = cpoint\ f\ g\ 1\ \sigma$ 
by (metis One-nat-def cpl.code cpl.disc-iff cpl.simps(2) nhd-conv-nnth nnth-Suc-NCons)

```

```

lemma nnth-cpl:
   $nnth\ (cpl\ f\ g\ x\ \sigma)\ i = cpoint\ f\ g\ (x+i)\ \sigma$ 
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: $(\neg nfinite\ w \wedge g\ w \wedge$
 $(\forall k. g ((ndropn\ k\ w)) \longrightarrow$
 $(\exists m. f ((nsubn\ w\ k\ (m+k))) \wedge$
 $0 < m \wedge g ((ndropn\ (m+k)\ w)))) \longrightarrow$
 $\neg nfinite\ w \wedge \neg nfinite\ (cpl\ f\ g\ 0\ w) \wedge nnth\ (cpl\ f\ g\ 0\ w)\ 0 = 0 \wedge$
 $nidx\ (cpl\ f\ g\ 0\ w) \wedge$
 $(\forall i. (nnth\ (cpl\ f\ g\ 0\ w)\ i) \leq nlength\ w) \wedge$
 $(\forall i. f ((nsubn\ w\ (nnth\ (cpl\ f\ g\ 0\ w)\ i)$
 $(nnth\ (cpl\ f\ g\ 0\ w)\ (Suc\ i))))))$
using 2 3 31
by (*simp add: cpl-infinite nnth-0-conv-nhd*)
have 5: $(\neg nfinite\ w \wedge g\ w \wedge$
 $(\forall k. g ((ndropn\ k\ w)) \longrightarrow$
 $(\exists m. f ((nsubn\ w\ k\ (m+k))) \wedge$
 $0 < m \wedge g ((ndropn\ (m+k)\ w)))) \longrightarrow$
 $\neg nfinite\ w \wedge$
 $(\exists (ls). \neg nfinite\ ls \wedge nnth\ ls\ 0 = 0 \wedge nidx\ ls \wedge$
 $(\forall i. (nnth\ ls\ i) \leq nlength\ w) \wedge$
 $(\forall i. f ((nsubn\ w\ (nnth\ ls\ i)$
 $(nnth\ ls\ (Suc\ i)))))) \wedge$
 $ls = (cpl\ f\ g\ 0\ w))$
using 4 by *auto*
have 6: $(\neg nfinite\ w \wedge g\ w \wedge$
 $(\forall k. g ((ndropn\ k\ w)) \longrightarrow$
 $(\exists m. f ((nsubn\ w\ k\ (m+k))) \wedge$

```

      0 < m ∧ g ( ( ndropn (m+k) w)))))) → (w ⊨ aomega f)
    using 3 5 unfolding aomega-d-def by blast
    have 7: (w ⊨ (inf ∧ g ∧ □(g → ((f ∧ more) ∧ finite);g)) → (aomega f) )
    using 6 1 by auto
    from 7 show ?thesis by blast
qed

```

lemma *AOmegaInduct*:

```

  ⊢ (inf ∧ g ∧ □(g → ((f ∧ more) ∧ finite);g)) → aomega f
unfolding Valid-def using AOmegaInductSem[of g f] by blast

```

lemma *AOmegaWeakCoInduct*:

```

assumes ⊢ g → ((f ∧ more) ∧ finite);g
shows   ⊢ inf ∧ g → aomega f
proof -
  have 1: ⊢ □(g → ((f ∧ more) ∧ finite);g)
    using assms by (simp add: BoxGen)
  show ?thesis using assms AOmegaInduct[of g f]
    using 1 by fastforce
qed

```

9.1.3 Weak Omega

lemma *ChopSemMono* [mono]:

```

(w ⊨ f ;g) =
  ( (∃ n. enat n ≤ nlength w ∧ f ( (ntaken n w)) ∧ g (ndropn n w)) ∨
    (¬nfinite w ∧ f w))
by (simp add: itl-defs)

```

coinductive *womega-d* :: ('a::world) formula ⇒ 'a formula

for *F* **where**

```

(s ⊨ F;(womega-d F)) ⇒ (s ⊨ (womega-d F))

```

syntax

```

-womega-d :: lift ⇒ lift      ((-W) [85] 85)

```

syntax (*ASCII*)

```

-womega-d :: lift ⇒ lift      ((womega -) [85] 85)

```

translations

```

-womega-d      ⇒ CONST womega-d

```

lemma *WOmegaIntros*:

```

  ⊢ f;(womega f) → (womega f)
using womega-d.intros using unl-lift2 by blast

```

lemma *WOmegaCases*:
 $(w \models (\text{womega } F)) \implies$
 $(w \models F; (\text{womega } F) \implies P) \implies P$
using *womega-d.cases*[*of F w P*] **by** *auto*

lemma *WOmegaUnrollSem*:
 $(s \models (\text{womega } f)) = (s \models f; (\text{womega } f))$
using *womega-d.cases*[*of f*]
by (*metis womega-d.simps*)

lemma *WOmegaUnroll*:
 $\vdash (\text{womega } f) = f; (\text{womega } f)$
using *WOmegaUnrollSem*
using *unl-lift2* **by** *blast*

lemma *WOmegaCoinductSem*:
assumes $\bigwedge x. X \ x \implies x \models F; (X \vee \text{womega } F)$
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 $\vdash X \longrightarrow F; (X \vee (\text{womega } F))$
shows $\vdash X \longrightarrow (\text{womega } F)$
using *assms WOmegaCoinductSem*[*of X F*]
by (*simp add: Valid-def*)

lemma *WOmegaWeakCoinductSem*:
assumes $\bigwedge x. x \models X \implies x \models F; X$
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 $\vdash X \longrightarrow F; X$
shows $\vdash X \longrightarrow (\text{womega } F)$
using *assms WOmegaWeakCoinductSem*[*of X F*]
by (*simp add: Valid-def*)

lemma *Omega-WOmega*:
 $\vdash (\text{omega } f) = ((\text{womega } ((f \wedge \text{more}) \wedge \text{finite})))$
proof –
have 3: $\vdash ((\text{womega } ((f \wedge \text{more}) \wedge \text{finite}))) \longrightarrow (\text{omega } f)$
using *OmegaWeakCoinduct*[*of LIFT (womega ((f \wedge more) \wedge finite)) f*]
by (*simp add: WOmegaUnroll int-iffD1*)
have 4: $\vdash (\text{omega } f) \longrightarrow ((f \wedge \text{more}) \wedge \text{finite}); (\text{omega } f)$
by (*simp add: OmegaUnroll int-iffD1*)
have 5: $\vdash (\text{omega } f) \longrightarrow (\text{womega } ((f \wedge \text{more}) \wedge \text{finite}))$
using *WOmegaWeakCoinduct*[*of LIFT (omega f) LIFT (f \wedge more) \wedge finite*]

4 **by** *blast*
show *?thesis*
by (*simp add: 3 5 int-iffI*)
qed

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:*

$\vdash (g \wedge \Box(g \longrightarrow ((f \wedge \text{more}) \wedge \text{finite});g))$
 \longrightarrow
 $((f \wedge \text{more}) \wedge \text{finite}); ((g \wedge \Box(g \longrightarrow ((f \wedge \text{more}) \wedge \text{finite});g)))$

using *WOmegaInducthelp* **by** *blast*

lemma *WOmegaInduct:*

$\vdash (g \wedge \Box(g \longrightarrow f;g)) \longrightarrow \text{womega } f$

by (*simp add: WOmegaInducthelp WOmegaWeakCoinduct*)

lemma *OmegaInduct:*

$\vdash (g \wedge \Box(g \longrightarrow ((f \wedge \text{more}) \wedge \text{finite});g)) \longrightarrow \text{omega } f$

by (*simp add: OmegaInducthelp OmegaWeakCoinduct*)

lemma *WOmega-coinduct:*

assumes $\vdash g \longrightarrow h \vee f;g$

shows $\vdash g \longrightarrow (\text{womega } f) \vee (\text{wpowerstar } f);h$

proof –

have 1: $\vdash (\text{wpowerstar } f);h = (\text{empty} \vee f;(\text{wpowerstar } f));h$
by (*simp add: LeftChopEqvChop WPowerstarEqv*)
have 2: $\vdash (\text{empty} \vee f;(\text{wpowerstar } f));h = (h \vee (f;(\text{wpowerstar } f));h)$
by (*simp add: EmptyOrChopEqv*)
have 3: $\vdash (\neg(h \vee (f;(\text{wpowerstar } f));h)) = (\neg h \wedge \neg(f;(\text{wpowerstar } f);h))$

```

  by (metis ChopAssoc int-eq int-simps(14) int-simps(33))
have 31:  $\vdash (\neg (wpowerstar f;h)) = (\neg h \wedge \neg(f;((wpowerstar f);h)))$ 
  by (metis 1 2 3 int-eq)
have 32:  $\vdash ((\neg h \wedge \neg(f;((wpowerstar f);h))) \wedge (h \vee f;g)) \longrightarrow$ 
   $(\neg(f;((wpowerstar f);h)) \wedge f;g)$ 
  by force
have 33:  $\vdash g \wedge \neg (wpowerstar f;h) \longrightarrow ((\neg h \wedge \neg(f;((wpowerstar f);h))) \wedge (h \vee f;g))$ 
  using assms 31
  by (metis Prop10 Prop12 int-eq int-iffD2 lift-and-com)
have 4:  $\vdash g \wedge \neg (wpowerstar f;h) \longrightarrow \neg(f;((wpowerstar f);h)) \wedge f;g$ 
  using assms 31 32 33
  using lift-imp-trans by blast
have 5:  $\vdash \neg(f;((wpowerstar f);h)) \wedge f;g \longrightarrow f;(g \wedge \neg wpowerstar f;h)$ 
  by (metis ChopAndNotChopImp int-eq lift-and-com)
have 6:  $\vdash g \wedge \neg (wpowerstar f;h) \longrightarrow f;(g \wedge \neg wpowerstar f;h)$ 
  using 4 5 lift-imp-trans by blast
have 7:  $\vdash g \wedge \neg (wpowerstar f;h) \longrightarrow (womega f)$ 
  using WOmegaWeakCoinduct[of LIFT  $g \wedge \neg (wpowerstar f;h)$   $f$ ] 6
  by blast
show ?thesis using 7 by fastforce
qed

```

lemma *Omega-coinduct*:

```

assumes  $\vdash g \longrightarrow h \vee (f \wedge more) \frown g$ 
shows  $\vdash g \longrightarrow (omega f) \vee (schopstar f) \frown h$ 
using assms WOmega-coinduct[of  $g h$  LIFT  $((f \wedge more) \wedge finite)$  ]
Omega-WOmega[of  $f$ ] SChopstar-WPowerstar[of LIFT  $(f \wedge more)$  ]
by (metis Prop10 SChopstar-finite SChopstar-WPowerstar-more integ-reflection schop-d-def)

```

9.2 Omega algebra

lemma *WOmega-coinduct-var1*:

```

assumes  $\vdash f \longrightarrow empty \vee g;f$ 
shows  $\vdash f \longrightarrow (womega g) \vee (wpowerstar g)$ 
using assms WOmega-coinduct[of  $f$  LIFT  $empty g$ ]
by (metis ChopEmpty integ-reflection)

```

lemma *Omega-coinduct-var1*:

```

assumes  $\vdash f \longrightarrow empty \vee (g \wedge more) \frown f$ 
shows  $\vdash f \longrightarrow (omega g) \vee (schopstar g)$ 
using assms
Omega-WOmega[of  $g$ ] SChopstar-WPowerstar[of LIFT  $(g \wedge more)$  ] SChopstar-WPowerstar-more[of  $g$ ]
WOmega-coinduct-var1[of  $f$  LIFT  $((g \wedge more) \wedge finite)$ ]
by (metis integ-reflection schop-d-def)

```

lemma *WOmega-coinduct-var2*:

```

assumes  $\vdash f \longrightarrow g;f$ 
shows  $\vdash f \longrightarrow (womega g)$ 
using assms WOmegaWeakCoinduct by blast

```

lemma *Omega-coinduct-var2*:
assumes $\vdash f \longrightarrow (g \wedge \text{more}) \frown f$
shows $\vdash f \longrightarrow (\text{omega } g)$
using *assms*
Omega-WOmega[of *g*] *SChopstar-WPowerstar*[of *LIFT (g ∧ more)*] *SChopstar-WPowerstar-more*[of *g*]
WOmega-coinduct-var2[of *f LIFT ((g ∧ more) ∧ finite)*]
by (*metis int-eq schop-d-def*)

lemma *WOmega-coinduct-eq*:
assumes $\vdash f = (h \vee g; f)$
shows $\vdash f \longrightarrow (\text{womega } g) \vee (\text{wpowerstar } g); h$
using *assms*
by (*simp add: Prop11 WOmega-coinduct*)

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*
Omega-WOmega[of *g*] *SChopstar-WPowerstar*[of *LIFT (g ∧ more)*] *SChopstar-WPowerstar-more*[of *g*]
WOmega-coinduct-eq[of *f h LIFT ((g ∧ more) ∧ finite)*]
SChopstar-finite[of *g*]
unfolding *schop-d-def*
by (*metis Prop10 inteq-reflection*)

lemma *WOmega-coinduct-eq-var1*:
assumes $\vdash f = (\text{empty} \vee g; f)$
shows $\vdash f \longrightarrow (\text{womega } g) \vee (\text{wpowerstar } g)$
using *assms*
using *WOmega-coinduct-var1 int-iffD1* **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*
Omega-WOmega[of *g*] *SChopstar-WPowerstar*[of *LIFT (g ∧ more)*] *SChopstar-WPowerstar-more*[of *g*]
WOmega-coinduct-eq-var1[of *f LIFT ((g ∧ more) ∧ finite)*]
by (*metis int-eq schop-d-def*)

lemma *WOmega-coinduct-eq-var2*:
assumes $\vdash f = g; f$
shows $\vdash f \longrightarrow (\text{womega } g)$
using *assms WOmega-coinduct-var2*
using *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*
Omega-WOmega[of *g*] *SChopstar-WPowerstar*[of *LIFT (g ∧ more)*] *SChopstar-WPowerstar-more*[of *g*]

WOmega-coinduct-eq-var2[of *f* *LIFT* $((g \wedge \text{more}) \wedge \text{finite})$]
by (*metis int-eq schop-d-def*)

lemma *WOmega-subdist*:

$\vdash (\text{womega } f) \longrightarrow (\text{womega } (f \vee g))$

proof –

have 1: $\vdash (\text{womega } f) \longrightarrow ((f \vee g)); (\text{womega } f)$

by (*metis (mono-tags, lifting) OrChopEqv WOmegaUnroll intI inteq-reflection unl-lift2*)

have 2: $\vdash (\text{womega } f) \longrightarrow ((f \vee g)); (\text{womega } f)$

by (*simp add: 1 Prop12*)

show *?thesis* **using** *WOmega-coinduct-var2 2* **by** *auto*

qed

lemma *Omega-subdist*:

$\vdash (\text{omega } f) \longrightarrow (\text{omega } (f \vee g))$

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 *f*] *Omega-WOmega*[of *LIFT* $(f \vee g)$]

WOmega-subdist[of *LIFT* $((f \wedge \text{more}) \wedge \text{finite})$ *LIFT* $((g \wedge \text{more}) \wedge \text{finite})$] 1

by (*metis int-eq*)

qed

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 LeftChopImpChop WOmegaUnroll WOmegaWeakCoinduct inteq-reflection*)

lemma *Omega-iso*:

assumes $\vdash f \longrightarrow g$

shows $\vdash (\text{omega } f) \longrightarrow (\text{omega } g)$

using *assms*

Omega-WOmega[of *f*] *Omega-WOmega*[of *g*]

WOmega-iso[of *LIFT* $((f \wedge \text{more}) \wedge \text{finite})$ *LIFT* $((g \wedge \text{more}) \wedge \text{finite})$]

by *fastforce*

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*:

$\vdash (\text{omega } f) \vee (\text{omega } g) \longrightarrow (\text{omega } (f \vee g))$

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 f] Omega-WOmega[of g] Omega-WOmega[of LIFT (f ∨ g)]
  WOmega-subdist-var[of LIFT ((f ∧ more) ∧ finite) LIFT ((g ∧ more) ∧ finite)] 1
by (metis int-eq)
qed

lemma WOmega-zero:
  ⊢ ¬(womega #False)
by (metis AndInfChopEqvAndInf InfEqvNotFinite WOmegaUnroll int-iffD1 int-simps(14) int-simps(21)
    inteq-reflection)

lemma Omega-zero:
  ⊢ ¬(omega #False)
using Omega-WOmega[of LIFT #False] WOmega-zero by fastforce

lemma WOmega-star-1:
  ⊢ (wpowerstar f);(womega f) = (womega f)
proof -
  have 1: ⊢ f; (womega f) → (womega f)
    by (simp add: WOmegaUnroll int-iffD2)
  have 2: ⊢ (wpowerstar f);(womega f) → (womega f)
    by (simp add: 1 WPowerstar-inductL-var-equiv)
  have 3: ⊢ (womega f) → (wpowerstar f);(womega f)
    by (meson Prop11 WOmegaUnroll WPowerstar-induct-lvar-eq2)
  show ?thesis
  by (simp add: 2 3 int-iffI)
qed

lemma Omega-star-1:
  ⊢ (schopstar f) ∧ (omega f) = (omega f)
using Omega-WOmega[of f]
WOmega-star-1[of LIFT ((f ∧ more) ∧ finite)]
SChopstar-WPowerstar[of LIFT (f ∧ more) ]
SChopstar-WPowerstar-more[of f] SChopstar-finite[of f]
unfolding schop-d-def
by (metis Prop10 inteq-reflection)

lemma WOmega-max-element:
  ⊢ f → (womega empty)
by (simp add: EmptyChop WOmegaWeakCoinduct int-iffD2)

lemma Omega-empty-zero:
  ⊢ ¬(omega empty)
  using Omega-WOmega[of LIFT empty] WOmega-zero
unfolding empty-d-def by fastforce

lemma WOmega-star-3:
  ⊢ (womega (wpowerstar f))

```

```

proof –
  have 1:  $\vdash \text{empty} \longrightarrow (\text{wpowerstar } f)$ 
    using WPowerstar-imp-empty by auto
  have 2:  $\vdash (\text{womega } \text{empty}) \longrightarrow (\text{womega } (\text{wpowerstar } f))$ 
    by (simp add: 1 WOmega-iso)
  show ?thesis
  by (meson 2 MP WOmega-max-element)
qed

lemma WOmega-1:
   $\vdash (\text{womega } f); g \longrightarrow (\text{womega } f)$ 
proof –
  have 1:  $\vdash (\text{womega } f); g \longrightarrow f; ((\text{womega } f); g)$ 
    by (metis ChopAssoc WOmegaUnroll int-eq int-iffD1)
  show ?thesis using WOmega-coinduct-var2[of LIFT (womega f); g f]
    using 1 by auto
qed

lemma Omega-1:
   $\vdash (\text{omega } f); g \longrightarrow (\text{omega } f)$ 
using Omega-WOmega[of f]
WOmega-1[of LIFT ((f  $\wedge$  more)  $\wedge$  finite) g]
by (metis int-eq)

lemma WOmega-sup-id:
assumes  $\vdash \text{empty} \longrightarrow g$ 
shows  $\vdash (\text{womega } f); g = (\text{womega } f)$ 
by (meson ChopEmpty Prop11 RightChopImpChop WOmega-1 assms lift-imp-trans)

lemma Omega-sup-id:
assumes  $\vdash \text{empty} \longrightarrow g$ 
shows  $\vdash (\text{omega } f); g = (\text{omega } f)$ 
using assms
by (metis Omega-WOmega WOmega-sup-id int-eq)

lemma WOmega-top:
   $\vdash (\text{womega } f); (\text{womega } \text{empty}) = (\text{womega } f)$ 
by (simp add: WOmega-max-element WOmega-sup-id)

lemma supid-WOmega:
assumes  $\vdash \text{empty} \longrightarrow f$ 
shows  $\vdash (\text{womega } f) = (\text{womega } \text{empty})$ 
using assms
by (simp add: Prop11 WOmega-iso WOmega-max-element)

lemma WOmega-simulation:
assumes  $\vdash h; f \longrightarrow g; h$ 
shows  $\vdash h; (\text{womega } f) \longrightarrow (\text{womega } g)$ 

```

proof –
have 1: $\vdash h ; (\text{womega } f) = h ; (f ; (\text{womega } f))$
by (*simp add: RightChopEqvChop WOmegaUnroll*)
have 2: $\vdash h ; (f ; (\text{womega } f)) \longrightarrow$
 $g ; (h ; (\text{womega } f))$
by (*meson ChopAssoc LeftChopImpChop Prop11 assms lift-imp-trans*)
have 3: $\vdash h ; (\text{womega } f) \longrightarrow g ; (h ; (\text{womega } f))$
by (*metis 1 2 int-eq*)
show ?thesis
using WOmega-coinduct-var2[of LIFT($h ; (\text{womega } f)$) g] 3 **by** blast
qed

lemma Omega-simulation:
assumes $\vdash (h) \frown (f \wedge \text{more}) \longrightarrow (g \wedge \text{more}) \frown (h)$
shows $\vdash (h) \frown (\text{omega } f) \longrightarrow (\text{omega } g)$
using assms
Omega-WOmega[of f] Omega-WOmega[of g]
WOmega-simulation[of LIFT($(h) \wedge \text{finite}$) LIFT($(f \wedge \text{more}) \wedge \text{finite}$) LIFT($(g \wedge \text{more}) \wedge \text{finite}$)]
by (*metis (no-types, opaque-lifting) ChopEmpty LeftSChopImpSChop SChopAssoc int-eq schop-d-def*)

lemma WOmega-WOmega:
 $\vdash (\text{womega } (\text{womega } f)) \longrightarrow (\text{womega } f)$
by (*metis WOmegaUnroll WOmega-1 int-eq*)

lemma WOmega-Wagner-1:
 $\vdash (\text{womega } (f ; \text{wpowerstar } f)) = (\text{womega } f)$
proof –
have 1: $\vdash (\text{womega } (f ; \text{wpowerstar } f)) =$
 $((f ; (\text{wpowerstar } f)) ; (f ; (\text{wpowerstar } f))) ; (\text{womega } (f ; \text{wpowerstar } f))$
by (*metis ChopAssoc WOmegaUnroll int-eq*)
have 2: $\vdash ((f ; (\text{wpowerstar } f)) ; (f ; (\text{wpowerstar } f))) ; (\text{womega } (f ; \text{wpowerstar } f)) =$
 $f ; (\text{womega } (f ; \text{wpowerstar } f))$
by (*metis (no-types, lifting) ChopAssoc WOmegaUnroll WPowerstar-slide-var WPowerstar-trans-eq inteq-reflection*)
have 3: $\vdash (\text{womega } (f ; \text{wpowerstar } f)) \longrightarrow (\text{womega } f)$
by (*metis 1 2 WOmega-coinduct-eq-var2 int-eq*)
have 4: $\vdash (\text{womega } f) \longrightarrow (\text{womega } (f ; \text{wpowerstar } f))$
by (*metis ChopAssoc WOmegaUnroll WOmega-coinduct-eq-var2 WOmega-star-1 int-eq*)
show ?thesis **by** (*simp add: 3 4 int-iffI*)
qed

lemma Omega-Wagner-1:
 $\vdash (\text{omega } (\text{s chopstar } f)) = (\text{omega } f)$
using Omega-WOmega[of f]
WOmega-Wagner-1[of LIFT($(f \wedge \text{more}) \wedge \text{finite}$)]
SChopstar-WPowerstar[of LIFT($f \wedge \text{more}$)]
SChopstar-WPowerstar-more[of f]
by (*metis AndMoreAndFiniteEqvAndFmore Omega-WOmega SCSAndMoreEqvAndFMoreSChop int-eq schop-d-def*)

lemma *Omega-Wagner-1-var2*:

$\vdash (\text{omega } ((f \wedge \text{more}) \neg \text{schopstar } f)) = (\text{omega } f)$

by (*metis* (*no-types*, *lifting*) *LeftSChopImpMoreRule* *Omega-WOmega* *Omega-Wagner-1* *OrFiniteInf* *Prop01* *Prop05* *Prop10* *Prop11* *SCSAndMoreEqvAndMoreSChop* *int-iffD1* *inteq-reflection*)

lemma *WOmega-Wagner-2-var*:

$\vdash f;(\text{womega } (g;f)) \longrightarrow (\text{womega } (f;g))$

proof –

have 1: $\vdash f;(\text{womega } (g;f)) \longrightarrow f;(g;f)$

by *auto*

show *?thesis*

by (*meson* *ChopAssoc* *Prop11* *WOmega-simulation*)

qed

lemma *Omega-Wagner-2-var*:

$\vdash (f \wedge \text{more}) \neg (\text{omega } ((g \wedge \text{more}) \neg (f \wedge \text{more}))) \longrightarrow (\text{omega } ((f \wedge \text{more}) \neg (g \wedge \text{more})))$

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*:

$\vdash (\text{womega } (f;g)) = f;(\text{womega } (g;f))$

proof –

have 1: $\vdash f;(\text{womega } (g;f)) \longrightarrow (\text{womega } (f;g))$

by (*simp* *add*: *WOmega-Wagner-2-var*)

have 2: $\vdash (\text{womega } (f;g)) = (f;g);(\text{womega } (f;g))$

by (*simp* *add*: *WOmegaUnroll*)

have 3: $\vdash (\text{womega } (f;g)) \longrightarrow f;(\text{womega } (g;f))$

by (*metis* 2 *ChopAssocSem* *RightChopImpChop* *Valid-def* *WOmega-Wagner-2-var* *inteq-reflection*)

show *?thesis*

using 1 3 *int-iffI* **by** *blast*

qed

lemma *Omega-Wagner-2*:

$\vdash (\text{omega } ((f \wedge \text{more}) \neg (g \wedge \text{more}))) = (f \wedge \text{more}) \neg (\text{omega } ((g \wedge \text{more}) \neg (f \wedge \text{more})))$

proof –

have 1: $\vdash (f \wedge \text{more}) \neg (\text{omega } ((g \wedge \text{more}) \neg (f \wedge \text{more}))) \longrightarrow$
 $(\text{omega } ((f \wedge \text{more}) \neg (g \wedge \text{more})))$

by (*simp* *add*: *Omega-Wagner-2-var*)

have 2: $\vdash (\text{omega } ((f \wedge \text{more}) \neg (g \wedge \text{more}))) = ((f \wedge \text{more}) \neg (g \wedge \text{more})) \neg (\text{omega } ((f \wedge \text{more}) \neg (g \wedge \text{more})))$

by (*metis* *AndMoreSChopAndMoreEqvAndMoreSChop* *OmegaUnroll* *int-eq* *schop-d-def*)

have 3: $\vdash (\text{omega } ((f \wedge \text{more}) \neg (g \wedge \text{more}))) \longrightarrow (f \wedge \text{more}) \neg (\text{omega } ((g \wedge \text{more}) \neg (f \wedge \text{more})))$

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 *Omega-Wagner-2-var1:*

$\vdash (\text{omega } ((f) \neg (g \wedge \text{more}))) = (f) \neg (\text{omega } ((g \wedge \text{more}) \neg (f)))$

proof –

have 1: $\vdash (f) \neg (\text{omega } ((g \wedge \text{more}) \neg (f))) \longrightarrow (\text{omega } ((f) \neg (g \wedge \text{more})))$

using *Omega-simulation*[of *LIFT* ($g \wedge \text{more}$) $\neg f$ *LIFT* $f \neg (g \wedge \text{more})$]

by (*metis* (*no-types*, *lifting*) *AndMoreSChopAndMoreEqvAndMoreSChop Prop10 SChopAndB SChopAssoc*

SChopMoreImpMore int-iffD1 inteq-reflection lift-imp-trans)

have 2: $\vdash (\text{omega } ((f) \neg (g \wedge \text{more}))) = ((f) \neg (g \wedge \text{more})) \neg (\text{omega } ((f) \neg (g \wedge \text{more})))$

by (*metis* *AndSChopA OmegaUnroll SChopstarMore-induct-lvar-eq SChopstar-inductL-var-equiv int-eq int-iffI schop-d-def*)

have 3: $\vdash (\text{omega } ((f) \neg (g \wedge \text{more}))) \longrightarrow (f) \neg (\text{omega } ((g \wedge \text{more}) \neg (f)))$

using 2 *Omega-coinduct-eq-var2*[of *LIFT* ($g \wedge \text{more}$) $\neg (\text{omega } ((f) \neg (g \wedge \text{more})))$ *LIFT* ($(g \wedge \text{more}) \neg (f)$)]

by (*metis* (*no-types*, *lifting*) *AndMoreSChopAndMoreEqvAndMoreSChop RightSChopImpSChop SChopAssoc 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 SChopAndA SChopAssoc 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 schop-d-def*)

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 $\vdash (f ; (\text{womega } (f \vee g)) \vee h) = (\text{womega } (f \vee g))$

shows $\vdash (\text{womega } (f \vee g)) = ((\text{womega } f) \vee (\text{wpowerstar } f); h)$

proof –

have 1: $\vdash (\text{womega } (f \vee g)) \longrightarrow ((\text{womega } f) \vee (\text{wpowerstar } f); h)$

using *WOmega-coinduct*[of *LIFT* ($\text{womega } (f \vee g)$) h f] *assms*

by *fastforce*

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** *blast*

qed

lemma *Omega-Wagner-3:*

assumes $\vdash ((f \wedge \text{more}) \frown (\text{omega } (f \vee g)) \vee (h)) = (\text{omega } (f \vee g))$

shows $\vdash (\text{omega } (f \vee g)) = ((\text{omega } f) \vee (\text{schopstar } f) \frown (h))$

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 *assms*

WOmega-Wagner-3[of LIFT ((f ∧ more) ∧ finite) LIFT ((g ∧ more) ∧ finite) h]

Omega-WOmega[of f] Omega-WOmega[of LIFT (f ∨ g)]

SChopstar-WPowerstar[of LIFT (f ∧ more)]

SChopstar-WPowerstar-more[of f]

unfolding *schop-d-def*

by (*metis 1 Prop10 SChopstar-finite inteq-reflection*)

qed

lemma *WOmega-Wagner-1-var:*

$\vdash (\text{womega } ((\text{wpowerstar } f);f)) = (\text{womega } f)$

by (*metis WOmega-Wagner-1 WOmega-Wagner-2 WOmega-star-1 int-eq*)

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:*

$\vdash (\text{wpowerstar } (\text{womega } f)) = (\text{empty} \vee (\text{womega } f))$

proof –

have 1: $\vdash (\text{wpowerstar } (\text{womega } f)) = (\text{empty} \vee (\text{womega } f);(\text{wpowerstar } (\text{womega } f)))$

by (*simp add: WPowerstarEqv*)

have 2: $\vdash (\text{empty} \vee (\text{womega } f);(\text{wpowerstar } (\text{womega } f))) \longrightarrow (\text{empty} \vee (\text{womega } f);(\text{womega } \text{empty}))$

by (*metis 1 WOmega-sup-id WOmega-top WPowerstar-imp-empty int-eq int-iffD2*)

have 3: $\vdash (\text{wpowerstar } (\text{womega } f)) \longrightarrow (\text{empty} \vee (\text{womega } f))$

by (*metis 2 WOmega-top WPowerstarEqv inteq-reflection*)

have 4: $\vdash (\text{empty} \vee (\text{womega } f)) \longrightarrow (\text{wpowerstar } (\text{womega } f))$

by (*meson Prop02 WPowerstar-ext WPowerstar-imp-empty*)

show *?thesis*

using 3 4 *int-iffI* **by** *blast*

qed

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*:

$\vdash ((w\Omega f) \vee ((w\text{powerstar } f);g); (w\Omega (f \vee g))) = (w\Omega (f \vee g))$

proof –

have 1: $\vdash (w\Omega (f \vee g)) = (f;(w\Omega (f \vee g)) \vee g;(w\Omega (f \vee g)))$

using *WOmegaUnroll*[of *LIFT* ($f \vee g$)] **by** (*metis OrChopEqv int-eq*)

have 2: $\vdash ((w\text{powerstar } f);g); (w\Omega (f \vee g)) = (w\text{powerstar } f);(g; (w\Omega (f \vee g)))$

by (*meson ChopAssoc int-iffD1 int-iffD2 int-iffI*)

show *?thesis* **using** 1 2 *WOmega-Wagner-3*[of f g *LIFT* $g;(w\Omega (f \vee g))$]]

by (*metis int-eq*)

qed

lemma *Omega-or-unfold*:

$\vdash ((\Omega f) \vee ((\text{schopstar } f) \frown (g \wedge \text{more})) \frown (\Omega (f \vee g))) = (\Omega (f \vee g))$

proof –

have 0: $\vdash (((f \vee g) \wedge \text{more}) \wedge \text{finite}) = (((f \wedge \text{more}) \wedge \text{finite}) \vee ((g \wedge \text{more}) \wedge \text{finite}))$

by *fastforce*

show *?thesis* **using** *WOmega-or-unfold*[of *LIFT* $((f \wedge \text{more}) \wedge \text{finite})$ *LIFT* $((g \wedge \text{more}) \wedge \text{finite})$] *Omega-WOmega*[of f] *Omega-WOmega*[of *LIFT* $(f \vee g)$]

SChopstar-WPowerstar[of *LIFT* $(f \wedge \text{more})$] *SChopstar-WPowerstar-more*[of f]

by (*metis (no-types, lifting) 0 Prop10 ChopAssoc SChopstar-finite SChopAssoc inteq-reflection schop-d-def*)

qed

lemma *WOmega-or-unfold-coinduct*:

$\vdash (w\Omega (f \vee g)) \longrightarrow (w\Omega (((w\text{powerstar } f);g) \)) \vee (w\text{powerstar } ((w\text{powerstar } f);g \));(w\Omega f)$

using *WOmega-or-unfold*[of f g]

WOmega-coinduct-eq[of *LIFT* $(w\Omega (f \vee g))$ *LIFT* $(w\Omega f)$ *LIFT* $(w\text{powerstar } f);g$]

using *WOmega-coinduct int-iffD2* **by** *blast*

lemma *Omega-or-unfold-coinduct*:

$\vdash (\Omega (f \vee g)) \longrightarrow (\Omega (((\text{schopstar } f) \frown (g \wedge \text{more})) \)) \vee (\text{schopstar } ((\text{schopstar } f) \frown (g \wedge \text{more}) \)) \frown (\Omega f)$

using *Omega-or-unfold*[of f g]

Omega-coinduct-eq[of *LIFT* $(\Omega (f \vee g))$ *LIFT* (Ωf) *LIFT* $(\text{schopstar } f) \frown (g \wedge \text{more})$]

by (*metis Prop10 Prop12 RightSChopImpMoreRule int-eq int-iffD2 lift-and-com*)

lemma *WOmega-or-unfold-induct*:

$\vdash (w\text{powerstar } ((w\text{powerstar } f);g));(w\Omega f) \longrightarrow (w\Omega (f \vee g))$

using *WOmega-or-unfold*[of f g]

using *WPowerstar-induct-leq* **by** *blast*

lemma *Omega-or-unfold-induct*:

$\vdash (\text{schopstar } ((\text{schopstar } f) \frown (g \wedge \text{more}))) \frown (\Omega f) \longrightarrow (\Omega (f \vee g))$

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*:

$\vdash (\text{womega } (f;(\text{wpowerstar } g)) \longrightarrow (\text{womega } (f \vee g)))$

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: $\vdash (f;(\text{wpowerstar } g)) \longrightarrow (f \vee g);(\text{wpowerstar } (f \vee g))$

using 01 02 *ChopImpChop* **by** *blast*

have 1: $\vdash (\text{womega } (f;(\text{wpowerstar } g))) \longrightarrow$
 $(\text{womega } ((f \vee g);(\text{wpowerstar } (f \vee g))))$

by (*simp add: 03 WOmega-iso*)

show *?thesis* **using** *WOmega-Wagner-1*

by (*metis 1 int-eq*)

qed

lemma *Omega-Wagner-1-gen*:

$\vdash (\text{omega } ((f \wedge \text{more}) \neg (\text{schopstar } g))) \longrightarrow (\text{omega } (f \vee g))$

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) \neg (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 AndMoreSChopAndMoreEqvAndMoreSChop Prop10 SChopImpFinite SChopstar-finite*
inteq-reflection schop-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*:

$\vdash (\text{womega } ((\text{wpowerstar } f);g)) \longrightarrow (\text{womega } (f \vee g))$
proof –
have 1: $\vdash (\text{womega } ((\text{wpowerstar } f);g)) =$
 $(\text{wpowerstar } f);(\text{womega } (g;(\text{wpowerstar } f)))$
by (*simp add: WOmega-Wagner-2*)
have 2: $\vdash (\text{womega } (g;(\text{wpowerstar } f))) \longrightarrow (\text{womega } (g \vee f))$
using *WOmega-Wagner-1-gen[of g f]* **by** *blast*
have 5: $\vdash (\text{wpowerstar } f);(\text{womega } (g;(\text{wpowerstar } f)))$
 $\longrightarrow (\text{wpowerstar } f);(\text{womega } (f \vee g))$
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:*

$\vdash (\text{omega } ((\text{schopstar } f) \frown (g \wedge \text{more}))) \longrightarrow (\text{omega } (f \vee g))$
proof –
have 1: $\vdash (\text{omega } ((\text{schopstar } f) \frown (g \wedge \text{more}))) =$
 $(\text{schopstar } f) \frown (\text{omega } ((g \wedge \text{more}) \frown (\text{schopstar } f)))$
by (*simp add: Omega-Wagner-2-var1*)
have 2: $\vdash (\text{omega } ((g \wedge \text{more}) \frown (\text{schopstar } f))) \longrightarrow (\text{omega } (g \vee f))$
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: $\vdash (\text{schopstar } f) \frown (\text{omega } ((g \wedge \text{more}) \frown (\text{schopstar } f)))$
 $\longrightarrow (\text{schopstar } f) \frown (\text{omega } (f \vee g))$
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: $\vdash (\text{schopstar } f) \frown (\text{omega } (f \vee g)) \longrightarrow (\text{schopstar } (f \vee g)) \frown (\text{omega } (f \vee g))$
by (*simp add: 6 LeftSChopImpSChop*)
have 8: $\vdash (\text{schopstar } (f \vee g)) \frown (\text{omega } (f \vee g)) \longrightarrow (\text{omega } (f \vee g))$
by (*simp add: Omega-star-1 int-iffD1*)
show *?thesis*
using 1 5 7 8 **by** *fastforce*
qed

lemma *WOmega-Denest:*

$\vdash (\text{womega } (f \vee g)) =$
 $((\text{womega } ((\text{wpowerstar } f);g)) \vee$
 $(\text{wpowerstar } ((\text{wpowerstar } f);g));(\text{womega } f))$

proof –

have 1: $\vdash (\text{womega } (f \vee g)) \longrightarrow (\text{womega } (((\text{wpowerstar } f); g) \vee (\text{wpowerstar } ((\text{wpowerstar } f); g)); (\text{womega } f)))$
by (*simp add: WOmega-or-unfold-coinduct*)
have 2: $\vdash (\text{womega } ((\text{wpowerstar } f); g)) \longrightarrow (\text{womega } (f \vee g))$
by (*simp add: WOmega-Wagner-1-var-gen*)
have 3: $\vdash (\text{wpowerstar } ((\text{wpowerstar } f); g)); (\text{womega } f) \longrightarrow (\text{womega } (f \vee g))$
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: $\vdash (\text{omega } ((\text{schopstar } f) \neg (g \wedge \text{more}))) \longrightarrow (\text{omega } (f \vee g))$
by (*simp add: Omega-Wagner-1-var-gen*)
have 3: $\vdash (\text{schopstar } ((\text{schopstar } f) \neg (g \wedge \text{more}))) \neg (\text{omega } f) \longrightarrow (\text{omega } (f \vee g))$
by (*simp add: Omega-or-unfold-induct*)
show ?thesis
by (*meson 1 2 3 Prop02 int-iffI*)
qed

lemma *WOmega-or-refine*:

assumes $\vdash g; f \longrightarrow f; (\text{wpowerstar } (f \vee g))$
shows $\vdash (\text{womega } (f \vee g)) = ((\text{womega } f) \vee (\text{wpowerstar } f); (\text{womega } g))$
proof –
have 1: $\vdash (\text{wpowerstar } g); f \longrightarrow f; (\text{wpowerstar } (f \vee g))$
using *assms WPowerstar-Quasicomm-var* **by** *blast*
have 2: $\vdash (\text{womega } (f \vee g)) = ((\text{womega } g) \vee ((\text{wpowerstar } g); f); (\text{womega } (f \vee g)))$
by (*metis WOmega-swap WOmega-or-unfold inteq-reflection*)
have 3: $\vdash ((\text{womega } g) \vee ((\text{wpowerstar } g); f); (\text{womega } (f \vee g))) \longrightarrow (\text{womega } g) \vee (f; (\text{wpowerstar } (f \vee g)); (\text{womega } (f \vee g)))$
by (*metis 1 2 AndChopB Prop08 Prop10 int-iffD2 inteq-reflection*)
have 40: $\vdash (\text{womega } g) \longrightarrow f; (\text{womega } (f \vee g)) \vee (\text{womega } g)$
by (*simp add: Prop05*)
have 41: $\vdash (f; (\text{wpowerstar } (f \vee g)); (\text{womega } (f \vee g))) \longrightarrow f; (\text{womega } (f \vee g)) \vee (\text{womega } g)$
by (*metis (mono-tags, lifting) ChopAssoc WOmega-star-1 intI inteq-reflection unl-lift2*)
have 4: $\vdash (\text{womega } g) \vee (f; (\text{wpowerstar } (f \vee g)); (\text{womega } (f \vee g))) \longrightarrow f; (\text{womega } (f \vee g)) \vee (\text{womega } g)$
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 WOmega-coinduct WOmega-star-1 ChopAssoc inteq-reflection*)
have 6: $\vdash ((\text{womega } f) \vee (\text{wpowerstar } f); (\text{womega } g)) \longrightarrow$

$(w\text{omega } (f \vee g)) \vee (w\text{powerstar } (f \vee g)); (w\text{omega } (f \vee g))$
by (*metis ChopImpChop Prop02 Prop05 WOmega-star-1 WOmega-subdist WOmega-swap WPowerstar-subdist int-eq*)
have 7: $\vdash ((w\text{omega } f) \vee (w\text{powerstar } f)); (w\text{omega } g)) \longrightarrow (w\text{omega } (f \vee g))$
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 $\vdash g;f \longrightarrow f;(\text{wpowerstar } (f \vee g))$

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 $\vdash (g \wedge \text{more}) \frown ((f \wedge \text{more}) \wedge \text{finite}) \longrightarrow (f \wedge \text{more}) \frown (\text{schopstar } (f \vee g))$

shows $(\vdash (\text{omega } (f \vee g)) = \#False) = (\vdash ((\text{omega } f) \vee (\text{omega } g)) = \#False)$

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 chop-d-def*)

lemma *WOmega-and-more-imp-more:*

$\vdash (\text{womega } (f \wedge \text{more})) \longrightarrow \text{more}$

by (*metis ChopImpDi DiAndB DiMoreEquMore 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
by (*metis ChopAssoc WOmegaUnroll int-eq wpow-Suc wpower-len*)
qed

lemma *omega-len*:

$\vdash \text{omega skip} = (\text{len } k) \frown \text{omega skip}$
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: $\vdash \text{len } \text{Suc } k = \text{len } k \frown \text{skip}$
by (*simp add: len-k-schop*)
have 2: $\vdash \text{len } \text{Suc } k \frown \text{omega skip} = \text{len } k \frown (\text{skip} \frown \text{omega skip})$
by (*metis 1 SChopAssoc inteq-reflection*)
have 3: $\vdash \text{skip} \frown \text{omega skip} = \text{omega skip}$
by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite NextSChopdef Omega-WOmega*
Omega-Wagner-1 Prop10 SkipChopFiniteImpFinite WOmega-Wagner-2 inteq-reflection next-d-def
s chopstar-skip-finite)
show ?thesis
by (*metis 2 3 Suc inteq-reflection*)
qed
qed

lemma *womega-exist-len*:

$\vdash (\text{womega skip}) = (\exists k. \text{len } k; (\text{womega skip}))$
proof –
have 1: $\vdash (\text{womega skip}) = (\text{wpowerstar skip}); (\text{womega skip})$
by (*simp add: WOmegaIntros WOmega-star-1 WPowerstar-induct-lvar int-iffD2 int-iffI*)
have 2: $\vdash (\text{wpowerstar skip}) = (\exists k. \text{len } k)$
unfolding *wpowerstar-d-def*
by (*simp add: ExEqvRule wpower-len*)
have 3: $\vdash (\exists k. \text{len } k); (\text{womega skip}) = (\exists k. \text{len } k; (\text{womega skip}))$
by (*metis ExistChop int-eq*)
show ?thesis **using** 1 2 3
by (*metis int-eq*)
qed

lemma *omega-exist-len*:

$\vdash (\text{omega skip}) = (\exists k. \text{len } k \frown (\text{omega skip}))$

proof –

have 1: $\vdash (\text{omega skip}) = (\text{schopstar skip}) \frown (\text{omega skip})$

by (*meson Omega-star-1 Prop11*)

have 2: $\vdash (\text{schopstar skip}) = (\exists k. \text{len } k)$

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:
 $\vdash (\text{womega fmore}) \longrightarrow \text{inf}$
 using wpowerstar-skip-fmore
 by (metis WOmega-Wagner-1 int-eq womega-skip-inf)

lemma omega-fmore-inf:
 $\vdash (\text{omega fmore}) \longrightarrow \text{inf}$
 by (metis AndMoreAndFiniteEqvAndFmore FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite
 FmoreEqvSkipChopFinite Omega-Wagner-1-var2 Prop10 SCSAndMoreEqvAndFmoreSChop
 SkipChopFiniteImpFinite inteq-reflection omega-skip-inf schopstar-skip-finite)

lemma womega-and-fmore-inf:
 $\vdash (\text{womega } (f \wedge \text{fmore})) \longrightarrow \text{inf}$
 by (meson Prop12 WOmega-iso int-iffD1 lift-and-com lift-imp-trans womega-fmore-inf)

lemma omega-and-fmore-inf:
 $\vdash (\text{omega } (f \wedge \text{fmore})) \longrightarrow \text{inf}$
 by (meson Prop12 Omega-iso int-iffD1 lift-and-com lift-imp-trans omega-fmore-inf)

lemma womega-and-empty-init:
 $\vdash (\text{womega } (f \wedge \text{empty})) = \text{init } f$
proof–
 have 1: $\vdash (\text{womega } (f \wedge \text{empty})) \longrightarrow \text{init } f$
 by (metis InitAndEmptyEqvAndEmpty StateChopExportA WOmegaUnroll inteq-reflection)
 have 2: $\vdash \text{init } f \longrightarrow (\text{womega } (f \wedge \text{empty}))$
 unfolding init-d-def using WOmega-coinduct-eq-var2[of LIFT (empty \wedge f);#True LIFT (f \wedge empty)]
 by (metis (no-types, lifting) AndChopCommute AndEmptyChopAndEmptyEqvAndEmpty ChopAssoc inteq-reflection)
 show ?thesis
 by (simp add: 1 2 int-iffI)
 qed

lemma omega-and-empty:
 $\vdash \neg(\text{omega } (f \wedge \text{empty}))$
proof –
 have 1: $\vdash (\text{omega } (f \wedge \text{empty})) = ((f \wedge \text{empty}) \wedge \text{more}) \frown (\text{omega } (f \wedge \text{empty}))$
 by (simp add: OmegaUnroll schop-d-def)
 have 2: $\vdash ((f \wedge \text{empty}) \wedge \text{more}) = \#False$
 unfolding empty-d-def by auto

show *?thesis*
by (*metis 1 2 NotDfFalse SChopImpDf int-simps(14) inteq-reflection lift-imp-trans*)
qed

lemma *EmptyOrMoreSplit*:
 $\vdash f = ((f \wedge \text{empty}) \vee (f \wedge \text{more}))$
unfolding *empty-d-def* **by** *auto*

lemma *womega-split-empty-more*:
 $\vdash (\text{womega } f) = ((\text{womega } (f \wedge \text{more})) \vee \text{wpowerstar } f; \text{init } f)$
proof –
have 1: $\vdash (\text{womega } ((f \wedge \text{empty}) \vee (f \wedge \text{more}))) =$
 $(\text{womega } (\text{wpowerstar } (f \wedge \text{empty}); (f \wedge \text{more}))) \vee$
 $\text{wpowerstar } (\text{wpowerstar } (f \wedge \text{empty}); (f \wedge \text{more})); \text{womega } (f \wedge \text{empty}))$
using *WOmega-Denest[of LIFT f \wedge empty LIFT f \wedge more]* **by** *blast*
have 2: $\vdash (\text{womega } (\text{wpowerstar } (f \wedge \text{empty}); (f \wedge \text{more}))) = (\text{womega } (f \wedge \text{more}))$
by (*metis EmptyChop WPowerstar-and-empty int-eq*)
have 3: $\vdash \text{wpowerstar } (\text{wpowerstar } (f \wedge \text{empty}); (f \wedge \text{more})) = \text{wpowerstar } f$
by (*metis EmptyChop WPowerstar-and-empty WPowerstar-more-absorb int-eq*)
show *?thesis*
by (*metis 1 2 3 EmptyOrMoreSplit inteq-reflection womega-and-empty-init*)
qed

lemma *omega-split-empty-more*:
 $\vdash (\text{omega } f) = ((\text{omega } (f \wedge \text{more})))$
by (*metis FPowerstar-WPowerstar FPowerstar-more-absorb Omega-Wagner-1 SChopstar-WPowerstar int-eq*)

lemma *omega-split-empty-more-var*:
 $\vdash (\text{omega } (f \wedge \text{more})) = (\text{omega } (f \wedge \text{fmore}))$
by (*metis AndMoreSChopEqvAndFmoreChop ChopEmpty EmptySChop Omega-Wagner-2-var1 inteq-reflection*)

lemma *omega-imp-inf*:
 $\vdash (\text{omega } f) \longrightarrow \text{inf}$
by (*metis int-eq omega-and-fmore-inf omega-split-empty-more omega-split-empty-more-var*)

lemma *inf-imp-omega-skip*:
 $\vdash \text{inf} \longrightarrow (\text{omega } \text{skip})$
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*
using 3 *OmegaWeakCoinduct[of LIFT inf LIFT skip]* **by** *blast*
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 chopstar-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 AndInfChopEqvAndInf WOmegaUnroll int-eq*)

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: $\vdash (\text{womega } ((f \wedge \text{finite}) \vee (f \wedge \text{inf}))) =$
 $(\text{womega } (\text{wpowerstar } (f \wedge \text{finite});(f \wedge \text{inf})) \vee$
 $\text{wpowerstar } (\text{wpowerstar } (f \wedge \text{finite});(f \wedge \text{inf}));\text{womega } (f \wedge \text{finite}))$
using *WOmega-Denest[of LIFT f \wedge finite LIFT f \wedge inf]* **by** *blast*
have 2: $\vdash (\text{womega } (\text{wpowerstar } (f \wedge \text{finite});(f \wedge \text{inf}))) = \text{wpowerstar } (f \wedge \text{finite});(f \wedge \text{inf})$
by (*metis ChopAndInf int-eq womega-and-inf*)

have 3: $\vdash \text{wpowerstar } (\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf})) = (\text{empty} \vee (\text{wpowerstar } (f \wedge \text{finite}); (f \wedge \text{inf})))$
by (*metis ChopAndInf WPowerstar-and-inf int-eq*)
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 *WPowerstar-SChopstar*:

$\vdash (\text{wpowerstar } f) = (\text{schopstar } f); (\text{empty} \vee (f \wedge \text{inf}))$
by (*metis SChopstar-WPowerstar WPowerstar-and-inf WPowerstar-chop-and-finite-inf int-eq*)

lemma *WOmega-Omega*:

$\vdash (\text{womega } f) = ((\text{schopstar } f); ((\text{init } f) \vee (f \wedge \text{inf})) \vee (\text{omega } f))$

proof –

have 1: $\vdash (\text{womega } f) = ((\text{womega } (f \wedge \text{more})) \vee \text{wpowerstar } f; \text{init } f)$
by (*simp add: womega-split-empty-more*)
have 2: $\vdash (\text{womega } (f \wedge \text{more})) = (\text{wpowerstar } ((f \wedge \text{more}) \wedge \text{finite}); ((f \wedge \text{more}) \wedge \text{inf}) \vee$
 $(\text{womega } ((f \wedge \text{more}) \wedge \text{finite})))$
by (*simp add: womega-split-finite-inf*)
have 3: $\vdash ((\text{womega } (f \wedge \text{more})) \vee \text{wpowerstar } f; \text{init } f) =$
 $((\text{wpowerstar } f; \text{init } f \vee$
 $(\text{wpowerstar } ((f \wedge \text{more}) \wedge \text{finite}); ((f \wedge \text{more}) \wedge \text{inf}) \vee$
 $(\text{womega } ((f \wedge \text{more}) \wedge \text{finite}))))$
using 1 2 **by** *fastforce*
have 4: $\vdash \text{wpowerstar } f; \text{init } f = (\text{schopstar } f); (\text{empty} \vee (f \wedge \text{inf}); \text{init } f)$
by (*metis ChopAssoc WPowerstar-SChopstar inteq-reflection*)
have 5: $\vdash ((\text{empty} \vee (f \wedge \text{inf}); \text{init } f) = ((\text{init } f) \vee (f \wedge \text{inf}))$
by (*meson AndInfChopEqvAndInf EmptyOrChopEqv Prop06*)
have 6: $\vdash (\text{wpowerstar } ((f \wedge \text{more}) \wedge \text{finite}); ((f \wedge \text{more}) \wedge \text{inf})) = (\text{schopstar } f); (f \wedge \text{inf})$
by (*meson AndMoreAndInfEqvAndInf ChopEqvChop Prop04 SChopstar-WPowerstar SChopstar-and-more*)
have 7: $\vdash (\text{womega } ((f \wedge \text{more}) \wedge \text{finite})) = (\text{omega } f)$
by (*meson Omega-WOmega int-iffD1 int-iffD2 int-iffI*)

have 8: $\vdash (\text{wpowerstar } f; \text{init } f \vee$
 $(\text{wpowerstar } ((f \wedge \text{more}) \wedge \text{finite})); ((f \wedge \text{more}) \wedge \text{inf}) \vee$
 $(\text{womega } ((f \wedge \text{more}) \wedge \text{finite}))) =$
 $((\text{schopstar } f); ((\text{init } f) \vee (f \wedge \text{inf})) \vee$
 $(\text{schopstar } f); (f \wedge \text{inf}) \vee$
 $(\text{omega } f))$
by (*metis* 3 4 5 6 7 *int-eq*)
have 90: $\vdash (\text{schopstar } f); ((\text{init } f) \vee (f \wedge \text{inf})) =$
 $((\text{schopstar } f); (\text{init } f) \vee (\text{schopstar } f); (f \wedge \text{inf}))$
by (*simp* *add: ChopOrEqv*)
have 91: $\vdash ((\text{schopstar } f); ((\text{init } f) \vee (f \wedge \text{inf})) \vee$
 $(\text{schopstar } f); (f \wedge \text{inf})) =$
 $((\text{schopstar } f); ((\text{init } f) \vee (f \wedge \text{inf})))$
using 90 **by** *fastforce*
have 9: $\vdash ((\text{schopstar } f); ((\text{init } f) \vee (f \wedge \text{inf})) \vee$
 $(\text{schopstar } f); (f \wedge \text{inf}) \vee$
 $(\text{omega } f)) =$
 $((\text{schopstar } f); ((\text{init } f) \vee (f \wedge \text{inf})) \vee (\text{omega } f))$
using 91 **by** *fastforce*
show *?thesis*
by (*metis* 1 3 8 9 *int-eq*)
qed

lemma *womega-and-more-inf:*

$\vdash (\text{womega } (f \wedge \text{more})) \longrightarrow \text{inf}$
using *womega-split-finite-inf* [of *LIFT* ($f \wedge \text{more}$)]
by (*metis* (*no-types*, *lifting*) *AndMoreAndFiniteEqvAndFmore ChopAndInf Prop02 Prop12 int-eq int-iffD1*
womega-and-fmore-inf)

lemma *womega-fmore-inf-eq:*

$\vdash \text{womega } \text{fmore} = \text{inf}$
by (*metis* (*no-types*, *lifting*) *ChopAndA ChopAndInf MoreAndInfEqvInf MoreEqvSkipSChopTrue SChopAs-*
soc
TrueChopMoreEqvMore TrueEqvTrueSChopTrue WOmega-simulation fmore-d-def int-eq int-iffI schop-d-def
womega-and-inf womega-fmore-inf)

lemma *womega-skip-inf-eq:*

$\vdash \text{womega } \text{skip} = \text{inf}$
by (*metis* *WOmega-Wagner-1 int-eq womega-fmore-inf-eq wpowerstar-skip-fmore*)

lemma *womega-more-inf:*

$\vdash \text{womega } \text{more} \longrightarrow \text{inf}$
by (*metis* *MoreAndInfEqvInf Prop02 WOmega-star-1 fmore-d-def inteq-reflection womega-fmore-inf-eq*
womega-skip-inf womega-skip-inf-eq womega-split-finite-inf)

lemma *womega-more-inf-eq:*

$\vdash \text{womega } \text{more} = \text{inf}$
by (*metis* *OrFiniteInf WOmega-subdist fmore-d-def int-eq int-iffI womega-fmore-inf-eq womega-more-inf*)

```

lemma womega-true:
  ⊢ womega # True
by (meson MP TrueEqvTrueChopTrue TrueW WOmega-coinduct-eq-var2)

```

9.3 Properties of Omega

```

lemma NotOmegaInf:
  ⊢ ¬(omega (inf))
proof –
  have 1: ⊢ omega (inf) = ((inf ∧ more) ∧ finite);omega inf
    by (simp add: OmegaUnroll)
  have 2: ⊢ ((inf ∧ more) ∧ finite) = #False
    unfolding finite-d-def by auto
  have 3: ⊢ #False;omega 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:
  ⊢ inf ⟶ omega (len(Suc n))
proof –
  have 1: ⊢ inf ⟶ ((len Suc n ∧ more) ∧ finite);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 blast
qed

```

```

lemma OmegaLenPlusOneEqvInf:
  ⊢ omega (len(Suc n)) = inf
  by (simp add: InfImpOmegaLenPlusOne int-iffI omega-imp-inf)

```

```

lemma OmegaSkipEqvInf:
  ⊢ omega skip = inf
proof –
  have 1: ⊢ skip = (len 1)
    by (simp add: len-d-def wpower-d-def)
    (metis ChopEmpty inteq-reflection)
  have 2: ⊢ omega skip = omega (len 1)
    using 1 by (metis OmegaUnroll inteq-reflection)
from 2 show ?thesis using OmegaLenPlusOneEqvInf by fastforce
qed

```

```

lemma InfImpOmegaTrue:
  ⊢ inf ⟶ omega # True
proof –
  have 1: ⊢ inf ⟶ ((# True ∧ more) ∧ finite);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** *blast*
qed

lemma *OmegaTrueEqvInf*:
 $\vdash \text{omega} \# \text{True} = \text{inf}$
by (*simp add: InfImpOmegaTrue int-iffI omega-imp-inf*)

lemma *InfImpOmegaMore*:
 $\vdash \text{inf} \longrightarrow \text{omega more}$
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** *blast*
qed

lemma *OmegaMoreEqvInf*:
 $\vdash \text{omega more} = \text{inf}$
by (*simp add: InfImpOmegaMore int-iffI omega-imp-inf*)

lemma *InfImpOmegaFinite*:
 $\vdash \text{inf} \longrightarrow \text{omega finite}$
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** *blast*
qed

lemma *OmegaFiniteEqvInf*:
 $\vdash \text{omega finite} = \text{inf}$
by (*simp add: InfImpOmegaFinite int-iffI omega-imp-inf*)

lemma *BoxStateAndInfImpWOmegaBoxState*:
 $\vdash \Box(\text{init } w) \longrightarrow \text{womega } (\Box(\text{init } w))$
using *WOmegaWeakCoinduct[of LIFT ($\Box(\text{init } w)$) LIFT $\Box(\text{init } w)$]*
by (*simp add: BoxStateChopBoxEqvBox int-iffD2*)

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 ($\text{more} \wedge \text{finite}$) LIFT inf]*
by (*metis AndMoreAndFiniteEqvAndFmore fmore-d-def int-eq*)
have 2: $\vdash (\text{more} \wedge \text{finite}); \text{inf} = \text{inf}$

by (metis *FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite FmoreEqvSkipChopFinite*
 Ω FiniteEqvInf Ω Unroll Prop10 SkipChopFiniteImpFinite fmore-d-def int-eq)
have 3: $\vdash \Box (init\ w) \wedge inf \longrightarrow ((\Box (init\ w) \wedge more) \wedge finite); (\Box (init\ w) \wedge inf)$
by (metis 1 2 Prop11 int-eq)
show ?thesis using Ω WeakCoinduct[of LIFT $(\Box (init\ w) \wedge inf)$ LIFT $\Box (init\ w)$] 3 by blast
qed

lemma *WOmegaBoxStateAndMoreImpBoxState:*

$\vdash w\Omega (\Box (init\ w) \wedge more) \longrightarrow \Box (init\ w)$
proof –
have 1: $\vdash w\Omega (\Box (init\ w) \wedge more) \longrightarrow init\ w$
by (metis AndChopA AndChopB BoxWPowerstarEqvBox StateChopExportA WOmegaUnroll WOmega-star-1
WPowerstar-more-absorb int-eq int-iffI)
have 2: $\vdash w\Omega (\Box (init\ w) \wedge more) \longrightarrow (\Box (init\ w) \wedge more); w\Omega (\Box (init\ w) \wedge more)$
by (simp add: WOmegaUnroll int-iffD1)
have 21: $\vdash ((\Box (init\ w) \wedge more)) \longrightarrow \bigcirc (\Box (init\ w))$
by (metis BoxImpNowAndWeakNext Prop01 Prop12 WnextEqvEmptyOrNext empty-d-def inteq-reflection
lift-and-com)
have 22: $\vdash w\Omega (\Box (init\ w) \wedge more) \longrightarrow more$
by (simp add: WOmega-and-more-imp-more)
have 23: $\vdash \bigcirc (\Box (init\ w)) \longrightarrow \bigcirc (w\Omega (\Box (init\ w) \wedge more))$
by (simp add: NextImpNext WPowerstar-ext)
have 24: $\vdash w\Omega (\Box (init\ w) \wedge more) \longrightarrow (\bigcirc (w\Omega (\Box (init\ w) \wedge more))); (w\Omega (\Box (init\ w) \wedge more))$
by (metis 21 23 LeftChopImpChop WOmegaUnroll WPowerstar-more-absorb int-eq lift-imp-trans)
have 10: $\vdash more \wedge w\Omega (\Box (init\ w) \wedge more) \longrightarrow \bigcirc (w\Omega (\Box (init\ w) \wedge more))$
using WOmega-star-1[of LIFT $(\Box (init\ w) \wedge more)$]
by (metis 22 24 NextChop Prop10 inteq-reflection lift-and-com)
show ?thesis using BoxIntro[of LIFT $w\Omega (\Box (init\ w) \wedge more)$ LIFT $(init\ w)$]
using 1 10 by blast
qed

lemma *OmegaBoxStateImpBoxState:*

$\vdash \Omega (\Box (init\ w)) \longrightarrow \Box (init\ w) \wedge inf$
proof –
have 1: $\vdash \Omega (\Box (init\ w)) \longrightarrow init\ w$
by (metis AndMoreAndFiniteEqvAndFmore BoxElim OmegaUnroll Omega-iso StateChopExportA inteq-reflection
lift-imp-trans)
have 2: $\vdash \Omega (\Box (init\ w)) \longrightarrow ((\Box (init\ w) \wedge more) \wedge finite); (\Omega (\Box (init\ w)))$
by (simp add: OmegaUnroll int-iffD1)
have 21: $\vdash ((\Box (init\ w) \wedge more) \wedge finite) \longrightarrow \bigcirc (\Box (init\ w))$
by (metis BoxImpNowAndWeakNext Prop01 Prop05 Prop12 WnextEqvEmptyOrNext empty-d-def
inteq-reflection lift-and-com)
have 22: $\vdash finite = (empty \vee fmore)$
by (metis EmptyOrMoreSplit FiniteAndEmptyEqvEmpty fmore-d-def inteq-reflection lift-and-com)
have 23: $\vdash (\Box (init\ w) \wedge finite) = ((\Box (init\ w) \wedge empty) \vee (\Box (init\ w) \wedge fmore))$

```

using 22 by fastforce
have 24:  $\vdash (\Box (init\ w) \wedge empty) = (init\ w \wedge empty)$ 
using BoxEqvAndBox StateAndEmptyImpBoxState by fastforce
have 25:  $\vdash \bigcirc(\Box (init\ w)) \wedge fmore \longrightarrow \bigcirc((init\ w) \wedge empty) \vee (\Box (init\ w) \wedge fmore)$ 
using 23 24 by (metis FmoreEqvSkipChopFinite NextAndEqvNextAndNext SkipChopEqvNext int-iffD2
inteq-reflection)
have 26:  $\bigwedge g. \vdash (\bigcirc((init\ w) \wedge empty) \vee (\Box (init\ w) \wedge fmore));g =$ 
 $(\bigcirc(init\ w \wedge g) \vee \bigcirc((\Box (init\ w) \wedge fmore);g))$ 
proof -
fix g :: 'a nelist  $\Rightarrow$  bool
have f1:  $\vdash \bigcirc (init\ w \wedge empty \vee \Box (init\ w) \wedge fmore);g =$ 
 $\bigcirc ((init\ w \wedge empty \vee \Box (init\ w) \wedge fmore);g)$ 
by (meson NextChop int-eq)
have  $\vdash skip;((init\ w \wedge empty \vee \Box (init\ w) \wedge fmore);g) =$ 
 $(skip;(init\ w \wedge g) \vee skip;((\Box (init\ w) \wedge fmore);g))$ 
by (metis ChopOrEqvRule OrChopEqv StateAndEmptyChop int-eq)
then show  $\vdash \bigcirc (init\ w \wedge empty \vee \Box (init\ w) \wedge fmore);g =$ 
 $(\bigcirc (init\ w \wedge g) \vee \bigcirc ((\Box (init\ w) \wedge fmore);g))$ 
by (metis (no-types, opaque-lifting) ChopAssoc Prop04 next-d-def)
qed
have 27:  $\vdash (\Box (init\ w) \wedge fmore) = (\Box (init\ w) \wedge skip);(\Box (init\ w) \wedge finite)$ 
by (metis BoxStateAndChopEqvChop FmoreEqvSkipChopFinite inteq-reflection)
have 28:  $\vdash (init\ w \wedge empty \vee \Box (init\ w) \wedge fmore) = (\Box (init\ w) \wedge finite)$ 
by (metis 23 24 int-eq)
have 29:  $\vdash (skip;(\Box (init\ w) \wedge finite));\omega (\Box (init\ w)) =$ 
 $(skip;(init\ w \wedge \omega (\Box (init\ w))) \vee skip;((\Box (init\ w) \wedge fmore);\omega (\Box (init\ w))))$ 
by (metis 26 28 SkipChopEqvNext int-eq)
have 3:  $\vdash (\Box (init\ w) \wedge fmore);(\omega (\Box (init\ w))) \longrightarrow$ 
 $(\bigcirc (init\ w \wedge \omega (\Box (init\ w))) \vee \bigcirc ((\Box (init\ w) \wedge fmore);\omega (\Box (init\ w))))$ 
by (metis 27 29 AndChopB LeftChopImpChop inteq-reflection next-d-def)
have 4:  $\vdash (\bigcirc (init\ w \wedge \omega (\Box (init\ w))) \vee \bigcirc ((\Box (init\ w) \wedge fmore);\omega (\Box (init\ w)))) \longrightarrow$ 
 $\bigcirc (\omega (\Box (init\ w)))$ 
by (metis 1 AndMoreAndFiniteEqvAndFmore ChopAndB OmegaUnroll Prop02 Prop10 int-eq lift-and-com
next-d-def)
have 5:  $\vdash \omega (\Box (init\ w)) \longrightarrow \bigcirc (\omega (\Box (init\ w)))$ 
by (metis 3 4 AndMoreAndFiniteEqvAndFmore OmegaUnroll inteq-reflection lift-imp-trans)
from 1 5 show ?thesis using BoxIntro
by (metis Prop01 Prop05 Prop12 omega-imp-inf)
qed

```

lemma OmegaIntro:

assumes $\vdash h \longrightarrow (f \wedge fmore);h$

shows $\vdash h \longrightarrow \omega f$

using *assms*

by (metis AndMoreAndFiniteEqvAndFmore Omega-coinduct-var2 int-eq schop-d-def)

lemma WOmegaEqvRule:

assumes $\vdash f = g$

shows $\vdash w\omega f = w\omega g$

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: $\vdash \text{ba } (f \longrightarrow g) \wedge (f); \text{womega } f \longrightarrow ((f \longrightarrow g) \wedge (f)); ((f \longrightarrow g) \wedge \text{womega } f)$

using *BaAndChopImport*[*of LIFT (f → g) LIFT (f) LIFT womega f*]

by *blast*

have 2: $\vdash (f \longrightarrow g) \wedge (f) \longrightarrow (g)$

by *auto*

have 3: $\vdash (f \longrightarrow g) \wedge \text{womega } f \longrightarrow \text{womega } f$

by *auto*

have 4: $\vdash \text{ba } (f \longrightarrow g) \wedge (f); \text{womega } f \longrightarrow (g); \text{womega } f$

using 1 2 3

by (*metis (no-types, lifting) AndChopB ChopAndB Prop10 int-eq lift-imp-trans*)

have 5: $\vdash \text{ba } (f \longrightarrow g) \wedge (f); \text{womega } f \longrightarrow ((f \longrightarrow g) \wedge (f)); (\text{ba } (f \longrightarrow g) \wedge \text{womega } f)$

using *BaAndChopImportB* **by** *blast*

have 6: $\vdash ((f \longrightarrow g) \wedge (f)); (\text{ba } (f \longrightarrow g) \wedge \text{womega } f) \longrightarrow$

$(g); (\text{ba } (f \longrightarrow g) \wedge \text{womega } f)$

using 2 *LeftChopImpChop* **by** *blast*

have 61: $\vdash \text{womega } f = (f); \text{womega } f$

by (*simp add: WOmegaUnroll*)

have 62: $\vdash (\text{ba } (f \longrightarrow g) \wedge \text{womega } f) \longrightarrow (\text{ba } (f \longrightarrow g) \wedge (f); \text{womega } f)$

by (*metis 61 ChopEmpty Prop11 int-eq*)

have 7: $\vdash (\text{ba } (f \longrightarrow g) \wedge \text{womega } f) \longrightarrow (g); (\text{ba } (f \longrightarrow g) \wedge \text{womega } f)$

using 62 5 6

by (*meson lift-imp-trans*)

have 8: $\vdash (\text{ba } (f \longrightarrow g) \wedge \text{womega } f) \longrightarrow \text{womega } g$


```

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*)

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: $\vdash ((womega f \longrightarrow womega g) \wedge (womega g \longrightarrow womega f)) = (womega f = womega g)$ **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*:

$\vdash ba f \wedge \omega g \longrightarrow \omega (f \wedge g)$
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* *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:
  ⊢ omega f → aomega f
using AOmegaWeakCoInduct[of LIFT omega f f]
by (metis AndMoreAndFiniteEqvAndFmore InfAndChop OmegaIntros OmegaUnroll Omega-fmore-absorb
    Prop10
      int-eq omega-and-fmore-inf)

```

```

lemma AOmegaImpOmega:
  ⊢ aomega f → omega f
using OmegaWeakCoInduct[of LIFT aomega f f]
by (simp add: AOmegaUnroll OmegaIntro int-iffD1)

```

```

lemma OmegaEqvAOmega:
  ⊢ omega f = aomega f
using OmegaImpAOmega AOmegaImpOmega
by (metis int-iffI)

```

end

10 Projection operator

```

theory Projection
imports Omega
begin

```

This theory introduces the projection operator `fproj` [5]. 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 [7].

10.1 Definitions

```

primcorec pfilt :: 'a intervals ⇒ nat nellist ⇒ 'a intervals
where
  pfilt σ ls = (case ls of (NNil n) ⇒ (NNil (nnth σ n)) |
    (NCons n ls1) ⇒ (NCons (nnth σ n) (pfilt σ ls1)))

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

primcorec lsum :: 'a intervals intervals ⇒ nat ⇒ nat nellist
where
  lsum nells a = (case nells of (NNil nell) ⇒ (NNil (a+(the-enat (nlength nell)))) |
    (NCons nell nells1) ⇒ (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 ⇒ nat nellist

```

where $\text{addzero } ls = (\text{if } \text{nlength } ls = 0 \text{ then}$
 $(\text{if } \text{nfirst } ls = 0 \text{ then } ls \text{ else } (NCons \ 0 \ ls)) \text{ else } (NCons \ 0 \ ls))$

definition $\text{powerinterval} :: ('a :: \text{world}) \text{ formula} \Rightarrow 'a \text{ intervals} \Rightarrow \text{nat nellist} \Rightarrow \text{bool}$
where $\text{powerinterval } F \ \sigma \ l =$
 $(\forall \ (i :: \text{nat}). (\text{enat } i) < \text{nlength } l \longrightarrow ((\text{nsubn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i))) \models F))$

definition $\text{cppl} :: ('a :: \text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ intervals} \Rightarrow \text{nat nellist}$
where $\text{cppl } f \ g \ \sigma = (\epsilon \ ls. \ \text{nidx } ls \wedge (\text{nnth } ls \ 0) = 0 \wedge \text{powerinterval } f \ \sigma \ ls \wedge$
 $((\text{nfinite } ls \wedge \text{nfinite } \sigma \wedge (\text{nlast } ls) = \text{the-enat}(\text{nlength } \sigma)) \vee$
 $(\neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma)) \wedge$
 $((\text{pfilt } \sigma \ ls) \models g))$

primcorec $\text{lcpl} :: ('a :: \text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ intervals} \Rightarrow \text{nat nellist} \Rightarrow \text{nat intervals intervals}$
where $\text{lcpl } f \ g \ \sigma \ ls =$
 $(\text{case } ls \text{ of } (NNil \ x) \Rightarrow (NNil \ (\text{nmap } (\lambda l. l+x) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ x)))) \mid$
 $(NCons \ x \ ls1) \Rightarrow$
 $(\text{case } ls1 \text{ of } (NNil \ y) \Rightarrow (NNil \ (\text{nmap } (\lambda l. l+x) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ x \ y)))) \mid$
 $(NCons \ y \ ls2) \Rightarrow (NCons \ (\text{nmap } (\lambda l. l+x) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ x \ y))) (\text{lcpl } f \ g \ \sigma \ ls1))))$

simps-of-case $\text{lcpl-code} \ [\text{code}, \text{simp}, \text{nitpick-simp}]: \text{lcpl.code}$

definition $\text{fprojection-d} :: ('a :: \text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{fprojection-d } F \ G \equiv \lambda \sigma. (\exists \ ls. \ \text{nidx } ls \wedge (\text{nnth } ls \ 0) = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge$
 $(\text{nlast } ls) = \text{the-enat}(\text{nlength } \sigma) \wedge$
 $\text{powerinterval } F \ \sigma \ ls \wedge ((\text{pfilt } \sigma \ ls) \models G)$
 $)$

definition $\text{oprojection-d} :: ('a :: \text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{oprojection-d } F \ G \equiv \lambda \sigma. (\exists \ ls. \ \text{nidx } ls \wedge (\text{nnth } ls \ 0) = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$
 $\text{powerinterval } F \ \sigma \ ls \wedge ((\text{pfilt } \sigma \ ls) \models G)$
 $)$

syntax

$\text{-fprojection-d} \quad :: [\text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((- \text{fproj } -) \ [84, 84] \ 83)$
 $\text{-oprojection-d} \quad :: [\text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((- \text{oproj } -) \ [84, 84] \ 83)$

syntax (ASCII)

$\text{-fprojection-d} \quad :: [\text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((- \text{fproj } -) \ [84, 84] \ 83)$
 $\text{-oprojection-d} \quad :: [\text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((- \text{oproj } -) \ [84, 84] \ 83)$

translations

$\text{-fprojection-d} \quad \Rightarrow \text{CONST } \text{fprojection-d}$
 $\text{-oprojection-d} \quad \Rightarrow \text{CONST } \text{oprojection-d}$

definition $\text{ufprojection-d} :: ('a :: \text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{ufprojection-d } F \ G \equiv \text{LIFT}(\neg(F \text{ fproj } (\neg G)))$

definition $uoprojection-d :: ('a :: world) formula \Rightarrow 'a formula \Rightarrow 'a formula$
where $uoprojection-d F G \equiv LIFT(\neg(F oproj (\neg G)))$

syntax

$-ufprojection-d \quad :: [lift, lift] \Rightarrow lift \quad ((- ufproj -) [84, 84] 83)$

syntax

$-uoprojection-d \quad :: [lift, lift] \Rightarrow lift \quad ((- uoproj -) [84, 84] 83)$

syntax (*ASCII*)

$-ufprojection-d \quad :: [lift, lift] \Rightarrow lift \quad ((- ufproj -) [84, 84] 83)$

$-uoprojection-d \quad :: [lift, lift] \Rightarrow lift \quad ((- uoproj -) [84, 84] 83)$

translations

$-ufprojection-d \quad \Rightarrow CONST ufprojection-d$

$-uoprojection-d \quad \Rightarrow CONST uoprojection-d$

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* $\equiv (\forall \text{ nell} \in \text{nset nells. } 0 < \text{nlength nell})$

10.2 Lemmas

10.2.1 Misc

lemma *all-nfinite-nnth-a:*

assumes $\forall i \leq \text{nlength nells. nfinite (nnth nells i)}$

shows *all-nfinite nells*

using *assms*

by (*metis in-nset-conv-nnth*)

lemma *all-nfinite-nnth-b:*

assumes *all-nfinite nells*

shows $\forall i \leq \text{nlength nells. nfinite (nnth nells i)}$

using *assms*

by (*meson in-nset-conv-nnth*)

lemma *all-gr-zero-nnth-a:*

assumes $\forall i \leq \text{nlength nells. } 0 < \text{nlength (nnth nells i)}$

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 $\forall i \leq \text{nlength nells. } 0 < \text{nlength (nnth nells i)}$

using *assms*

by (*meson in-nset-conv-nnth*)

lemma *all-nfinite-nnth-ntaken:*

assumes *all-nfinite nells*

$n \leq \text{nlength nells}$

shows *all-nfinite (ntaken n nells)*

using *assms*

by (*meson nset-ntaken subsetD*)

lemma *all-nfinite-nnth-ndropn:*

assumes *all-nfinite nells*

$n \leq \text{nlength nells}$

shows *all-nfinite (ndropn n nells)*

using *assms*

by (*metis nset-ndropn subsetD*)

lemma *all-nfinite-nnth-nsubn:*

assumes *all-nfinite nells*

$n \leq \text{nlength nells}$

$k \leq n$

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*

$n \leq \text{nlength } \text{nells}$

$k \leq n$

shows *all-gr-zero (nsubn nells k n)*

using *assms*

by (*metis in-mono nset-ndropn nset-ntaken nsubn-def1*)

10.2.2 pfilt Lemmas

lemma *pfilt-nmap*:

$\text{pfilt } \text{nell } \text{ls} = \text{nmap } (\lambda x. (\text{nnth } \text{nell } x)) \text{ ls}$

proof (*coinduction arbitrary: nell ls*)

case (*Eq-nellist nell1*)

then show *?case* **by** *simp (metis nellist.case-eq-if)*

qed

lemma *pfilt-nlength*:

$\text{nlength}(\text{pfilt } \text{nell } \text{ls}) = \text{nlength } \text{ls}$

by (*simp add: pfilt-nmap*)

lemma *pfilt-nnth*:

assumes $i \leq \text{nlength } (\text{pfilt } \text{nell } \text{ls})$

shows $(\text{nnth } (\text{pfilt } \text{nell } \text{ls}) \ i) = (\text{nnth } \text{nell } (\text{nnth } \text{ls } i))$

using *assms*

by (*simp add: pfilt-nmap*)

lemma *pfilt-expand*:

$(\text{nell1} = (\text{pfilt } \text{nell } \text{ls})) =$

$(\text{nlength } \text{nell1} = \text{nlength } \text{ls} \wedge$

$(\forall (i::\text{nat}). i \leq \text{nlength } \text{nell1} \longrightarrow (\text{nnth } \text{nell1 } i) = (\text{nnth } \text{nell } (\text{nnth } \text{ls } i))))$

by (*metis nellist-eq-nnth-eq pfilt-nlength pfilt-nnth*)

lemma *pfilt-nfuse*:

$pfilt\ nell\ (nfuse\ ls1\ ls2) = (nfuse\ (pfilt\ nell\ ls1)\ (pfilt\ nell\ ls2))$

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 $nlength\ (pfilt\ (nfuse\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ndropn\ (nnth\ ls\ n)\ nell))\ ls) =$

$nlength\ (nfuse\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))$

$(pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))))$

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*

$(nnth\ ls\ 0) = 0$

$(nlast\ ls) = the-enat(nlength\ nell)$

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: $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))\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\)))\ i =$
 $(if\ i \leq nlength\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))\ then\ nnth\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))\ i$
 $else\ nnth\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\))\ (i - (the-enat\ (nlength\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))\))))$

using 1 8 **by** (*meson* *nfuse-nnth* *not-le-imp-less*)

have 11: $nlength\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls)) = n$

using *assms* **by** (*simp* *add: pfilt-nlength*)

have 12: $i \leq n \longrightarrow nnth\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))\ i =$
 $nnth\ (ntaken\ (nnth\ ls\ n)\ nell)\ (nnth\ (ntaken\ n\ ls)\ i)$

by (*simp* *add: 11* *pfilt-nnth*)

have 13: $i \leq n \longrightarrow nnth\ (ntaken\ (nnth\ ls\ n)\ nell)\ (nnth\ (ntaken\ n\ ls)\ i) =$
 $nnth\ (ntaken\ (nnth\ ls\ n)\ nell)\ (nnth\ ls\ i)$

by (*simp* *add: ntaken-nnth*)

have 15: $i \leq n \longrightarrow nnth\ (ntaken\ (nnth\ ls\ n)\ nell)\ (nnth\ ls\ i) = nnth\ nell\ (nnth\ ls\ i)$

using *assms* **by** (*simp* *add: nidx-less-eq* *ntaken-nnth*)

have 16: $nnth\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\))\ (i - (the-enat\ (nlength\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))\)))) =$
 $nnth\ (pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))\))\ (i - n)$

```

by (simp add: 11)
have 17:  $\text{nnth } (p\text{filt } (\text{ndropn } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } n)) (\text{ndropn } n \text{ ls} )) (i - n) =$   

 $\text{nnth } (\text{ndropn } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } n)) (\text{ndropn } n \text{ ls} )) (i - n))$   

using assms  $p\text{filt-nnth}$ [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-b-alt*:

assumes *nidx ls*

$$n\text{nth } ls \ 0 = 0$$
$$\neg nfinite\ nell$$
$$\neg nfinite\ ls$$
$$i \leq nlength \text{ } ls$$
$$n \leq nlength \text{ } ls$$

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 ≤ nlength(nfuse (pfilt (ntaken (nnth ls n) nell) (ntaken n ls))
    (pfilt (ndropn (nnth ls n) nell) (nmap (λx. 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*)

$$\text{have } \mathcal{B}: n_{\text{first}}(\text{pfilt } (\text{ndropn } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } n)) (\text{ndropn } n \text{ ls}))) = \\ \text{nnth } (\text{pfilt } (\text{ndropn } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } n)) (\text{ndropn } n \text{ ls}))) 0$$

by (*metis ndropn-0 ndropn-nfirst*)

$$\text{have } 4: \text{nnth } (\text{pfilt } (\text{ndropn } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } n)) (\text{ndropn } n \text{ ls}))) 0 = \text{nnth } (\text{ndropn } (\text{nnth } \text{ls } n) \text{ nell}) (\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } n)) (\text{ndropn } n \text{ ls})) 0)$$

```

    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))
    (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))
    then nnth (pfilt (ntaken (nnth ls n) nell) (ntaken n ls)) i
    else nnth (pfilt (ndropn (nnth ls n) nell) (nmap ( $\lambda x. x - (nnth\ ls\ n)$ ) (ndropn n ls) ))
    (i - (the-enat (nlength (pfilt (ntaken (nnth ls n) nell) (ntaken n ls))))))
    using 1 8 by (meson nfuse-nnth not-le-imp-less)
have 11: nlength (pfilt (ntaken (nnth ls n) nell) (ntaken n ls)) = n
    using assms by (simp add: pfilt-nlength)
have 12:  $i \leq n \longrightarrow nnth$  (pfilt (ntaken (nnth ls n) nell) (ntaken n ls)) i =
    nnth (ntaken (nnth ls n) nell) (nnth (ntaken n ls) i)
    by (simp add: 11 pfilt-nnth)
have 13:  $i \leq n \longrightarrow nnth$  (ntaken (nnth ls n) nell) (nnth (ntaken n ls) i) =
    nnth (ntaken (nnth ls n) nell) (nnth ls i)
    by (simp add: ntaken-nnth)
have 15:  $i \leq n \longrightarrow nnth$  (ntaken (nnth ls n) nell) (nnth ls i) = nnth nell (nnth ls i)
    using assms by (simp add: nidx-less-eq ntaken-nnth)
have 16: nnth (pfilt (ndropn (nnth ls n) nell) (nmap ( $\lambda x. x - (nnth\ ls\ n)$ ) (ndropn n ls) ))
    (i - (the-enat (nlength (pfilt (ntaken (nnth ls n) nell) (ntaken n ls)))))) =
    nnth (pfilt (ndropn (nnth ls n) nell) (nmap ( $\lambda x. x - (nnth\ ls\ n)$ ) (ndropn n ls) ))
    (i - n)
    by (simp add: 11)
have 17: nnth (pfilt (ndropn (nnth ls n) nell) (nmap ( $\lambda x. x - (nnth\ ls\ n)$ ) (ndropn n ls) )) (i - n) =
    nnth (ndropn (nnth ls n) nell) (nnth (nmap ( $\lambda x. x - (nnth\ ls\ n)$ ) (ndropn n ls) ) (i - n))
    using assms pfilt-nnth[of i-n (ndropn (nnth ls n) nell)
    (nmap ( $\lambda x. x - (nnth\ ls\ n)$ ) (ndropn n ls) ) ]
    by (metis enat-minus-mono1 idiff-enat-enat ndropn-nlength nlength-nmap pfilt-nlength)
have 18:  $i > n \longrightarrow nnth$  (nmap ( $\lambda x. x - (nnth\ ls\ n)$ ) (ndropn n ls) ) (i - n) =
    nnth ls (n+(i-n)) - (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:  $i > n \longrightarrow nnth$  (ndropn (nnth ls n) nell) (nnth (nmap ( $\lambda x. x - (nnth\ ls\ n)$ ) (ndropn n ls) ) (i
-n))
    = nnth (ndropn (nnth ls n) nell) ((nnth ls i) - (nnth ls n))
    by (simp add: 18)
have 20 :  $i > n \longrightarrow nnth$  (ndropn (nnth ls n) nell) ((nnth ls i) - (nnth ls n)) =

```

$$nnth\ nell\ ((nnth\ ls\ n) + ((nnth\ ls\ i) - (nnth\ ls\ n)))$$
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:*

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 $nnth\ (pfilt\ (nfuse\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ndropn\ (nnth\ ls\ n)\ nell))\ ls)\ i =$
 $nnth\ (nfuse\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))$
 $(pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))))\ i$
using *assms nfuse-pfilt-nnth-a[of ls nell i n]*
 $nfuse-pfilt-nnth-b[of\ ls\ nell\ i\ n]$
by *simp*

lemma *nfuse-pfilt-nnth-alt:*

assumes *nidx ls*
 $nnth\ ls\ 0 = 0$
 $\neg nfinite\ nell$
 $\neg nfinite\ ls$
 $i \leq nlength\ ls$
 $n \leq nlength\ ls$
shows $nnth\ (pfilt\ (nfuse\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ndropn\ (nnth\ ls\ n)\ nell))\ ls)\ i =$
 $nnth\ (nfuse\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))$
 $(pfilt\ (ndropn\ (nnth\ ls\ n)\ nell)\ (nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))))\ i$
using *assms nfuse-pfilt-nnth-a-alt[of ls nell i n]*
 $nfuse-pfilt-nnth-b-alt[of\ ls\ nell\ i\ n]$
by *simp*

lemma *nfuse-pfilt:*

assumes *nidx ls*
 $nnth\ ls\ 0 = 0$
 $(nlast\ ls) = the-enat(nlength\ nell)$
nfinite nell
nfinite ls
 $n \leq nlength\ ls$
shows $pfilt\ (nfuse\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ndropn\ (nnth\ ls\ n)\ nell))\ ls =$
 $nfuse\ (pfilt\ (ntaken\ (nnth\ ls\ n)\ nell)\ (ntaken\ n\ ls))$

$(\text{pfilt } (\text{ndropn } (\text{nnth } ls \ n) \ nell) \ (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \))$
using *assms nfuse-pfilt-nlength nfuse-pfilt-nnth*
nellist-eq-nnth-eq **by** (*metis pfilt-nlength*)

lemma *nfuse-pfilt-alt:*

assumes *nidx ls*
 $\text{nnth } ls \ 0 = 0$
 $\neg \text{nfinite } nell$
 $\neg \text{nfinite } ls$
 $n \leq \text{nlength } ls$
shows $\text{pfilt } (\text{nfuse } (\text{ntaken } (\text{nnth } ls \ n) \ nell) \ (\text{ndropn } (\text{nnth } ls \ n) \ nell)) \ ls =$
 $\text{nfuse } (\text{pfilt } (\text{ntaken } (\text{nnth } ls \ n) \ nell) \ (\text{ntaken } n \ ls))$
 $(\text{pfilt } (\text{ndropn } (\text{nnth } ls \ n) \ nell) \ (\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls) \))$
using *assms nfuse-pfilt-nlength nfuse-pfilt-nnth-alt*
nellist-eq-nnth-eq **by** (*metis pfilt-nlength*)

lemma *nfuse-pfilt-a:*

assumes *nidx ls1*
 $\text{nfinite } ls1$
 $\text{nnth } ls1 \ 0 = 0$
nidx ls2
 $\text{nfinite } ls2$
 $\text{nnth } ls2 \ 0 = \text{nlast } ls1$
 $(\text{nlast } ls2) = \text{the-enat}(\text{nlength } nell)$
 $\text{nfinite } nell$
shows $\text{pfilt } (\text{nfuse } (\text{ntaken } (\text{nlast } ls1) \ nell) \ (\text{ndropn } (\text{nfirst } ls2) \ nell)) \ (\text{nfuse } ls1 \ ls2) =$
 $\text{nfuse } (\text{pfilt } (\text{ntaken } (\text{nlast } ls1) \ nell) \ ls1)$
 $(\text{pfilt } (\text{ndropn } (\text{nfirst } ls2) \ nell) \ (\text{nmap } (\lambda x. x - (\text{nfirst } ls2)) \ ls2))$

proof –

have *0*: $\exists ll. (\text{enat } ll) = \text{nlength } ls1$
using *assms(2) nfinite-nlength-enat* **by** *fastforce*
obtain *ll* **where** *01*: $(\text{enat } ll) = \text{nlength } ls1$
using *0* **by** *auto*
have *1*: $\text{nlast } ls1 = \text{nfirst } ls2$
using *assms* **by** (*metis nlast-NNil ntaken-0 ntaken-nlast*)
have *2*: *nidx* $(\text{nfuse } ls1 \ ls2)$
using *assms nidx-nfuse* **by** *blast*
have *20*: $\text{nnth } (\text{nfuse } ls1 \ ls2) \ 0 = 0$
by (*metis 1 assms(3) nfuse-nnth zero-enat-def zero-le*)
have *3*: $\text{nlast}(\text{nfuse } ls1 \ ls2) = \text{the-enat}(\text{nlength } nell)$
using *assms*
by (*simp add: 1 nlast-nfuse*)
have *4*: $ll \leq \text{nlength } (\text{nfuse } ls1 \ ls2)$
by (*simp add: 01 nfuse-nlength*)
have *5*: $\text{pfilt } (\text{nfuse } (\text{ntaken } (\text{nnth } (\text{nfuse } ls1 \ ls2) \ ll) \ nell)$
 $(\text{ndropn } (\text{nnth } (\text{nfuse } ls1 \ ls2) \ ll) \ nell))$
 $(\text{nfuse } ls1 \ ls2)$
 $=$
 $\text{nfuse } (\text{pfilt } (\text{ntaken } (\text{nnth } (\text{nfuse } ls1 \ ls2) \ ll) \ nell) \ (\text{ntaken } ll \ (\text{nfuse } ls1 \ ls2)))$
 $(\text{pfilt } (\text{ndropn } (\text{nnth } (\text{nfuse } ls1 \ ls2) \ ll) \ nell)$

```

      (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 *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: $\exists ll. (enat ll) = nlength ls1$

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 4: $ll \leq nlength (nfuse ls1 ls2)$

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 4 nfuse-pfilt-alt[of (nfuse ls1 ls2) nell ll] 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 *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: $\text{nlength (ntaken } n \text{ (pfilt } \sigma \text{ } ls))} = \text{nlength (pfilt } \sigma \text{ (ntaken } n \text{ } ls))}$

by (*simp add: assms pfilt-nlength*)

have 2: $\forall (i::\text{nat}). i \leq \text{nlength (ntaken } n \text{ (pfilt } \sigma \text{ } ls))} \longrightarrow$

$(\text{nnth (ntaken } n \text{ (pfilt } \sigma \text{ } ls)) } i) = (\text{nnth (pfilt } \sigma \text{ (ntaken } n \text{ } ls)) } i)$

by (*simp add: pfilt-nmap*)

show ?thesis **using** 1 2 *nellist-eq-nnth-eq* **by** blast

qed

lemma *pfilt-ntaken-nidx*:

assumes $n \leq \text{nlength } nell$

$\text{nidx } ls$

$\text{nnth } ls \ 0 = 0$

$\text{nfinite } ls$

$\text{nlast } ls = n$

shows $\text{pfilt } nell \text{ } ls = \text{pfilt (ntaken } n \text{ } nell) \text{ } ls$

proof –

have 1: $\text{nlength (pfilt } nell \text{ } ls) = \text{nlength (pfilt (ntaken } n \text{ } nell) \text{ } ls)}$

by (*simp add: pfilt-nlength*)

have 2: $\forall (i::\text{nat}). i \leq \text{nlength (pfilt } nell \text{ } ls) \longrightarrow$

$(\text{nnth (pfilt } nell \text{ } ls) } i) = (\text{nnth (pfilt (ntaken } n \text{ } nell) \text{ } ls) } i)$

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 $(\text{enat } n) \leq \text{nlength } nell$

$\text{nidx } ls$

$\text{nnth } ls \text{ } 0 = 0$

shows $\text{nlength } (\text{pfilt } nell (\text{nmap } (\lambda x. x + n) \text{ } ls)) = \text{nlength } (\text{pfilt } (\text{ndropn } n \text{ } nell) \text{ } ls)$

using assms **by** (simp add: pfilt-nlength)

lemma pfilt-ndropn-nidx-nnth:

assumes $(\text{enat } n) \leq \text{nlength } nell$

$\text{nidx } ls$

$\text{nnth } ls \text{ } 0 = 0$

$i \leq \text{nlength } ls$

shows $\text{nnth } (\text{pfilt } nell (\text{nmap } (\lambda x. x + n) \text{ } ls)) \text{ } i = \text{nnth } (\text{pfilt } (\text{ndropn } n \text{ } nell) \text{ } ls) \text{ } i$

proof –

have 1: $\text{nnth } (\text{pfilt } nell (\text{nmap } (\lambda x. x + n) \text{ } ls)) \text{ } i = \text{nnth } nell (\text{nnth } (\text{nmap } (\lambda x. x + n) \text{ } ls) \text{ } i)$

by (simp add: assms pfilt-nnth pfilt-nlength)

have 2: $\text{nnth } nell (\text{nnth } (\text{nmap } (\lambda x. x + n) \text{ } ls) \text{ } i) = (\text{nnth } nell ((\text{nnth } ls \text{ } i) + n))$

by (simp add: assms)

have 3: $\text{nnth } (\text{pfilt } (\text{ndropn } n \text{ } nell) \text{ } ls) \text{ } i = \text{nnth } (\text{ndropn } n \text{ } nell) (\text{nnth } ls \text{ } i)$

by (simp add: assms pfilt-nnth pfilt-nlength)

have 4: $\text{nnth } (\text{ndropn } n \text{ } nell) (\text{nnth } ls \text{ } i) = (\text{nnth } nell ((\text{nnth } ls \text{ } i) + n))$

by (metis ntaken-ndropn-nlast ntaken-nlast)

show ?thesis

using 1 2 3 4 **by** presburger

qed

lemma pfilt-ndropn-nidx:

assumes $(\text{enat } n) \leq \text{nlength } nell$

$\text{nidx } ls$

$\text{nnth } ls \text{ } 0 = 0$

shows $\text{pfilt } nell (\text{nmap } (\lambda x. x + n) \text{ } ls) = \text{pfilt } (\text{ndropn } n \text{ } nell) \text{ } ls$

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:*

(*nnth* (*pfilt* (*pfilt nell ls1*) *ls2*) *k*) = (*nnth nell* (*nnth ls1* (*nnth 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:*

(*pfilt* (*pfilt nell ls1*) *ls2*) = (*pfilt nell* (*nmap* ($\lambda x. (nnth ls1 x)$) *ls2*))

by (*simp add: pfilt-expand pfilt-nlength pfilt-pfilt*)

lemma *pfilt-nmap-pfilt:*

(*pfilt* (*pfilt nell ls1*) *ls2*) = *pfilt nell* (*pfilt ls1 ls2*)

by (*metis pfilt-nmap pfilt-pfilt-nmap*)

lemma *pfilt-nsubn:*

assumes $k \leq n$

$n \leq nlength\ ls$

shows (*nsubn* (*pfilt nell ls*) *k n*) = (*pfilt nell* (*nsubn ls k n*))

using *assms*

by (*simp add: ndropn-nmap nsubn-def1 pfilt-nmap*)

lemma *pfilt-nfusecat-nmap:*

(*pfilt nell* (*nfusecat lls*)) = (*nfusecat* (*nmap* ($\lambda ls. (pfilt nell ls)$) *lls*))

proof –

have 1: *pfilt nell* (*nfusecat lls*) = *nmap* (*nnth nell*) (*nfusecat lls*)

using *pfilt-nmap[of nell (nfusecat lls)]* **by** *blast*

have 2: (*nfusecat* (*nmap* ($\lambda ls. (pfilt nell ls)$) *lls*)) =

(*nfusecat* (*nmap* (*nmap* (*nnth nell*)) *lls*))

using *pfilt-nmap* **by** *metis*

have 3: *nmap* (*nnth nell*) (*nfusecat lls*) = (*nfusecat* (*nmap* (*nmap* (*nnth nell*)) *lls*))

by (*simp add: nmap-nfusecat*)

show *?thesis*

by (*simp add: 1 2 3*)

qed

10.2.3 powerinterval lemmas

lemma *powerinterval-splita0:*

assumes *nidx ls*

nnth ls 0 = 0

$n \leq nlength\ ls$

nfinite ls

nfinite σ

nlast ls = the-enat(nlength σ)

i < nlength(ntaken n ls)

shows (*nsubn* (*ntaken* (*nnth ls n*) *σ*) (*nnth* (*ntaken n ls*) *i*) (*nnth* (*ntaken n ls*) (*Suc i*))) =
(*nsubn* *σ* (*nnth ls i*) (*nnth ls* (*Suc i*)))

proof –

have 01: $(\text{nnth } ls \ n) \leq \text{nlength } \sigma$
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: $(\text{nnth } (\text{ntaken } n \ ls) \ i) \leq (\text{nnth } (\text{ntaken } n \ ls) \ (\text{Suc } i))$
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: $(\text{nnth } (\text{ntaken } n \ ls) \ (\text{Suc } i)) \leq (\text{nnth } ls \ n)$
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-splita0-alt:*

assumes *nidx ls*
 $\text{nnth } ls \ 0 = 0$
 $n \leq \text{nlength } ls$
 $\neg \text{nfinite } ls$
 $\neg \text{nfinite } \sigma$
 $i < \text{nlength}(\text{ntaken } n \ ls)$
shows $(\text{nsubn } (\text{ntaken } (\text{nnth } ls \ n) \ \sigma) \ (\text{nnth } (\text{ntaken } n \ ls) \ i) \ (\text{nnth } (\text{ntaken } n \ ls) \ (\text{Suc } i))) =$
 $(\text{nsubn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))$

proof –

have 01: $(\text{nnth } ls \ n) \leq \text{nlength } \sigma$
by (*meson* *assms*(5) *enat-less-imp-le less-enatE nfinite-conv-nlength-enat*)
have 02: $(\text{nnth } (\text{ntaken } n \ ls) \ i) \leq (\text{nnth } (\text{ntaken } n \ ls) \ (\text{Suc } i))$
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: $(\text{nnth } (\text{ntaken } n \ ls) \ (\text{Suc } i)) \leq (\text{nnth } ls \ n)$
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*
 $\text{nnth } ls \ 0 = 0$
 $\text{nfinite } ls$
 $n \leq \text{nlength } ls$
 $\text{nfinite } \sigma$
 $\text{nlast } ls = \text{the-enat}(\text{nlength } \sigma)$
 $\text{powerinterval } f \ \sigma \ ls$
shows $\text{powerinterval } f \ (\text{ntaken } (\text{nnth } ls \ n) \ \sigma) \ (\text{ntaken } n \ ls)$

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: $\text{powerinterval } f \text{ (ntaken (nnth ls n) } \sigma \text{) (ntaken n ls) =}$
 $(\forall i. i < \text{nlength(ntaken n ls)} \longrightarrow$
 $((\text{nsbn (ntaken (nnth ls n) } \sigma \text{) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i))) } \models f))$
using *powerinterval-def* **by** *blast*
have 2: $(\forall i. i < \text{nlength(ntaken n ls)} \longrightarrow$
 $((\text{nsbn (ntaken (nnth ls n) } \sigma \text{) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i))) } \models f))$
proof
fix i
show $i < \text{nlength(ntaken n ls)} \longrightarrow$
 $((\text{nsbn (ntaken (nnth ls n) } \sigma \text{) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i))) } \models f)$
proof –
have 21: $i < n \longrightarrow ((\text{nsbn } \sigma \text{ (nnth ls i) (nnth ls (Suc i))) } \models f)$
using *0 assms less-le-trans* **by** *(metis enat-ord-simps(2))*
have 22: $i < \text{nlength(ntaken n ls)} \longrightarrow$
 $((\text{nsbn } \sigma \text{ (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i))) } \models f)$
by *(simp add: 21 assms(4) ntaken-nnth)*
have 23: $i < \text{nlength(ntaken n ls)} \longrightarrow$
 $((\text{nsbn (ntaken (nnth ls n) } \sigma \text{) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i))) } \models f)$
using *22 assms powerinterval-split0*
using *21 by fastforce*
show *?thesis* **using** *23 by blast*
qed
qed
show *?thesis* **using** *1 2 by blast*
qed

lemma *powerinterval-split0-alt:*

assumes nidx ls
 $\text{nnth ls 0} = 0$
 $\neg \text{nfinite ls}$
 $n \leq \text{nlength ls}$
 $\neg \text{nfinite } \sigma$
 $\text{powerinterval } f \text{ } \sigma \text{ } \text{ls}$
shows $\text{powerinterval } f \text{ (ntaken (nnth ls n) } \sigma \text{) (ntaken n ls)}$
proof –
have 0: $(\forall i. i < \text{nlength ls} \longrightarrow ((\text{nsbn } \sigma \text{ (nnth ls i) (nnth ls (Suc i))) } \models f))$
using *assms by (simp add: powerinterval-def)*
have 1: $\text{powerinterval } f \text{ (ntaken (nnth ls n) } \sigma \text{) (ntaken n ls) =}$
 $(\forall i. i < \text{nlength(ntaken n ls)} \longrightarrow$
 $((\text{nsbn (ntaken (nnth ls n) } \sigma \text{) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i))) } \models f))$
using *powerinterval-def* **by** *blast*
have 2: $(\forall i. i < \text{nlength(ntaken n ls)} \longrightarrow$
 $((\text{nsbn (ntaken (nnth ls n) } \sigma \text{) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i))) } \models f))$
proof
fix i
show $i < \text{nlength(ntaken n ls)} \longrightarrow$
 $((\text{nsbn (ntaken (nnth ls n) } \sigma \text{) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i))) } \models f)$
proof –
have 21: $i < n \longrightarrow ((\text{nsbn } \sigma \text{ (nnth ls i) (nnth ls (Suc i))) } \models f)$
using *0 assms less-le-trans* **by** *(metis enat-ord-simps(2))*

```

have 22: i<nlength(ntaken n ls) →
  ((nsubn σ (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) ) ⊨ f)
  by (simp add: 21 assms(4) ntaken-nnth)
have 23: i<nlength(ntaken n ls) →
  ((nsubn (ntaken (nnth ls n) σ) (nnth (ntaken n ls) i) (nnth (ntaken n ls) (Suc i)) ) ⊨ f)
  using 22 assms powerinterval-split0-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: $i < (nlength\ ls - n) \longrightarrow (nnth\ ls\ n) \leq (nnth\ ls\ (n + i))$
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 10: $i < (nlength\ ls - n) \longrightarrow$

$$\begin{aligned} & (nsubn\ (ndropn\ (nnth\ ls\ n)\ \sigma) \\ & \quad ((nnth\ ls\ (n + i)) - (nnth\ ls\ n)) \\ & \quad ((nnth\ ls\ (n + (Suc\ i))) - (nnth\ ls\ n)) \\ &) = \\ & (nsubn\ \sigma\ (nnth\ ls\ (i+n))\ (nnth\ ls\ ((Suc\ i)+n))\) \end{aligned}$$

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*
 $nnth\ ls\ 0 = 0$
 $\neg nfinite\ ls$
 $n \leq nlength\ ls$
 $\neg nfinite\ \sigma$
 $i < (nlength\ ls - n)$
shows $(nsubn\ (ndropn\ (nnth\ ls\ n)\ \sigma)$

$$\begin{aligned} & \quad (nnth\ (((nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))))\ i) \\ & \quad (nnth\ (((nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))))\ (Suc\ i)) \\ &) = \\ & (nsubn\ \sigma\ (nnth\ ls\ (i+n))\ (nnth\ ls\ ((Suc\ i)+n))\) \end{aligned}$$

proof –

have 1: $nlength\ (((nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls)))) = (nlength\ ls) - n$
by (*simp add: assms*)
have 2: $i < (nlength\ ls - n) \longrightarrow$

$$\begin{aligned} & (nnth\ (((nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))))\ i) = \\ & (nnth\ ls\ (n + i)) - (nnth\ ls\ n) \end{aligned}$$

by *simp*
have 3: $i < (nlength\ ls - n) \longrightarrow$

$$\begin{aligned} & (nnth\ (((nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))))\ (Suc\ i)) = \\ & (nnth\ ls\ (n + (Suc\ i))) - (nnth\ ls\ n) \end{aligned}$$

using *assms* **by** (*metis eSuc-enat ileI1 ndropn-nlength ndropn-nnth nnth-nmap*)
have 4: $i < (nlength\ ls - n) \longrightarrow$

$$\begin{aligned} & (nsubn\ (ndropn\ (nnth\ ls\ n)\ \sigma) \\ & \quad (nnth\ (((nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))))\ i) \\ & \quad (nnth\ (((nmap\ (\lambda x. x - (nnth\ ls\ n))\ (ndropn\ n\ ls))))\ (Suc\ i)) \\ &) = \\ & (nsubn\ (ndropn\ (nnth\ ls\ n)\ \sigma) \\ & \quad ((nnth\ ls\ (n + i)) - (nnth\ ls\ n)) \\ & \quad ((nnth\ ls\ (n + (Suc\ i))) - (nnth\ ls\ n)) \\ &) \end{aligned}$$

by (simp add: 2 3)
 have 5: $i < (\text{nlength } ls - n) \longrightarrow (\text{nnth } ls (n + i)) < (\text{nnth } ls (n + (\text{Suc } i)))$
 using assms
 by (metis Suc-ile-eq add.commute add-Suc-right enat-min nidx-expand plus-enat-simps(1))
 have 6: $i < (\text{nlength } ls - n) \longrightarrow (\text{nnth } ls n) \leq (\text{nnth } ls (n + i))$
 using assms
 by (metis add.commute enat-min le-add1 nidx-less-eq order-less-imp-le plus-enat-simps(1))
 have 7: $i < (\text{nlength } ls - n) \longrightarrow (\text{nnth } ls n) \leq (\text{nnth } ls (n + (\text{Suc } i)))$
 using 5 6 by linarith
 have 8: $i < (\text{nlength } ls - n) \longrightarrow (\text{nnth } ls (n + i)) - (\text{nnth } ls n) < (\text{nnth } ls (n + (\text{Suc } i))) - (\text{nnth } ls n)$
 using 5 6 diff-less-mono by blast
 have 9: $i < (\text{nlength } ls - n) \longrightarrow (\text{nnth } ls (n + (\text{Suc } i))) - (\text{nnth } ls n) \leq \text{nlength } \sigma - (\text{nnth } ls n)$
 using assms
 by (simp add: nfinite-conv-nlength-enat)
 have 10: $i < (\text{nlength } ls - n) \longrightarrow$
 $(\text{nsubn } (\text{ndropn } (\text{nnth } ls n) \sigma)$
 $((\text{nnth } ls (n + i)) - (\text{nnth } ls n))$
 $((\text{nnth } ls (n + (\text{Suc } i))) - (\text{nnth } ls n))$
 $) =$
 $(\text{nsubn } \sigma (\text{nnth } ls (i+n)) (\text{nnth } ls ((\text{Suc } i)+n)))$
 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*
 $\text{nnth } ls 0 = 0$
 nfinite *ls*
 $n \leq \text{nlength } ls$
 nfinite σ
 $\text{nlast } ls = \text{the-enat}(\text{nlength } \sigma)$
 powerinterval *f* σ *ls*
 shows *powerinterval* *f* $(\text{ndropn } (\text{nnth } ls n) \sigma) ((\text{nmap } (\lambda x. x - (\text{nnth } ls n)) (\text{ndropn } n \text{ } ls)))$
 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: $\text{powerinterval } f (\text{ndropn } (\text{nnth } ls n) \sigma) ((\text{nmap } (\lambda x. x - (\text{nnth } ls n)) (\text{ndropn } n \text{ } ls))) =$
 $(\forall i. i < \text{nlength } (((\text{nmap } (\lambda x. x - (\text{nnth } ls n)) (\text{ndropn } n \text{ } ls)))) \longrightarrow$
 $((\text{nsubn } (\text{ndropn } (\text{nnth } ls n) \sigma)$
 $(\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls n)) (\text{ndropn } n \text{ } ls)))) i)$
 $(\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls n)) (\text{ndropn } n \text{ } ls)))) (\text{Suc } i))$
 $) \models f))$
 by (simp add: powerinterval-def)
 have 2: $(\forall i. i < \text{nlength } (((\text{nmap } (\lambda x. x - (\text{nnth } ls n)) (\text{ndropn } n \text{ } ls)))) \longrightarrow$
 $((\text{nsubn } (\text{ndropn } (\text{nnth } ls n) \sigma)$
 $(\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls n)) (\text{ndropn } n \text{ } ls)))) i)$
 $(\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls n)) (\text{ndropn } n \text{ } ls)))) (\text{Suc } i))$
 $) \models f))$
 proof

```

fix i
show  $i < \text{nlength}(((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls)))) \longrightarrow$ 
  ( (  $\text{nsbn } (\text{ndropn } (\text{nnth } ls \ n) \ \sigma)$ 
    (  $\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls)))) \ i$ 
      (  $\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls)))) \ (\text{Suc } i)$ 
        )  $\models f$  )
  )
proof –
  have 21:  $i < (\text{nlength } ls) - n \longrightarrow$ 
    (  $(\text{nsbn } \sigma \ (\text{nnth } ls \ (i+n)) \ (\text{nnth } ls \ ((\text{Suc } i)+n))) \models f$  )
    using 0 assms(4) enat-min by auto
  show ?thesis
  using 21 assms powerinterval-splitb0 by fastforce
qed
qed
show ?thesis using 1 2 by blast
qed

lemma powerinterval-splitb-alt:
assumes nidx ls
   $\text{nnth } ls \ 0 = 0$ 
   $\neg \text{nfinite } ls$ 
   $n \leq \text{nlength } ls$ 
   $\neg \text{nfinite } \sigma$ 
  powerinterval f  $\sigma$  ls
shows powerinterval f (  $\text{ndropn } (\text{nnth } ls \ n) \ \sigma$  ) (  $(\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls))$  )
proof –
have 0:  $(\forall \ i. \ i < \text{nlength } ls \longrightarrow ( (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) \models f ))$ 
  using assms by ( simp add: powerinterval-def )
have 1:  $\text{powerinterval } f \ (\text{ndropn } (\text{nnth } ls \ n) \ \sigma) \ ((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls))) =$ 
  (  $\forall \ i. \ i < \text{nlength } (((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls)))) \longrightarrow$ 
    ( (  $\text{nsbn } (\text{ndropn } (\text{nnth } ls \ n) \ \sigma)$ 
      (  $\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls)))) \ i$ 
        (  $\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls)))) \ (\text{Suc } i)$ 
          )  $\models f$  )
      )
    )
  )
  by ( simp add: powerinterval-def )
have 2:  $(\forall \ i. \ i < \text{nlength}(((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls)))) \longrightarrow$ 
  ( (  $\text{nsbn } (\text{ndropn } (\text{nnth } ls \ n) \ \sigma)$ 
    (  $\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls)))) \ i$ 
      (  $\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls)))) \ (\text{Suc } i)$ 
        )  $\models f$  )
    )
  )
proof
fix i
show  $i < \text{nlength}(((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls)))) \longrightarrow$ 
  ( (  $\text{nsbn } (\text{ndropn } (\text{nnth } ls \ n) \ \sigma)$ 
    (  $\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls)))) \ i$ 
      (  $\text{nnth } (((\text{nmap } (\lambda x. x - (\text{nnth } ls \ n)) \ (\text{ndropn } n \ ls)))) \ (\text{Suc } i)$ 
        )  $\models f$  )
    )
  )
proof –
  have 21:  $i < (\text{nlength } ls) - n \longrightarrow$ 
    (  $(\text{nsbn } \sigma \ (\text{nnth } ls \ (i+n)) \ (\text{nnth } ls \ ((\text{Suc } i)+n))) \models f$  )

```

```

    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 ≤ 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) ∧
     powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x - (nnth ls n)) (ndropn n ls))))
proof -
  have 1: powerinterval f σ ls ⇒
    powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls)
  by (simp add: assms powerinterval-splita)
  have 2: powerinterval f σ ls ⇒
    powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x - (nnth ls n)) (ndropn n ls)))
  by (simp add: assms powerinterval-splitb)
  have 3: powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls) =
    (∀ i. i < n → ( (nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f ))
  using assms by (simp add: powerinterval-def powerinterval-splita0)
  have 4: powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x - (nnth ls n)) (ndropn n ls))) =
    (∀ i. i < (nlength ls) - n → ((nsubn σ (nnth ls (i+n)) (nnth ls ((Suc i)+n)) ) ⊨ f ))
  unfolding powerinterval-def using assms powerinterval-splitb0[of ls n σ] by (simp)
  have 5: (∀ i. i < (nlength ls) - n → ((nsubn σ (nnth ls (i+n)) (nnth ls ((Suc i)+n)) ) ⊨ f )) =
    (∀ i. n ≤ i ∧ i < (nlength ls) → ((nsubn σ (nnth ls i) (nnth ls (Suc i)) ) ⊨ f ))
  using assms enat-min by auto
  (metis diff-add diff-less-mono enat-ord-simps(2) idiff-enat-enat nfinite-nlength-enat)
  have 6: (powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls) ∧
    powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) ⇒
    powerinterval f σ 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
    nnth ls 0 = 0
    ¬nfinite ls
    n ≤ nlength ls
    ¬nfinite σ

```


shows $\text{powerinterval } f \ \sigma \ \text{ls} =$
 $(\text{powerinterval } f \ (\text{ntaken} \ (\text{nnth } \text{ls } n) \ \sigma) \ (\text{ntaken } n \ \text{ls}) \wedge$
 $\text{powerinterval } f \ (\text{ndropn} \ (\text{nnth } \text{ls } n) \ \sigma) \ ((\text{nmap} \ (\lambda x. x - (\text{nnth } \text{ls } n)) \ (\text{ndropn } n \ \text{ls})))$
proof –
have 1: $\text{powerinterval } f \ \sigma \ \text{ls} \implies$
 $\text{powerinterval } f \ (\text{ntaken} \ (\text{nnth } \text{ls } n) \ \sigma) \ (\text{ntaken } n \ \text{ls})$
by (*simp add: assms powerinterval-splita-alt*)
have 2: $\text{powerinterval } f \ \sigma \ \text{ls} \implies$
 $\text{powerinterval } f \ (\text{ndropn} \ (\text{nnth } \text{ls } n) \ \sigma) \ ((\text{nmap} \ (\lambda x. x - (\text{nnth } \text{ls } n)) \ (\text{ndropn } n \ \text{ls})))$
by (*simp add: assms powerinterval-splitb-alt*)
have 3: $\text{powerinterval } f \ (\text{ntaken} \ (\text{nnth } \text{ls } n) \ \sigma) \ (\text{ntaken } n \ \text{ls}) =$
 $(\forall i. i < n \longrightarrow (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)) \) \models f))$
using *assms by (simp add: powerinterval-def powerinterval-splita0-alt)*
have 4: $\text{powerinterval } f \ (\text{ndropn} \ (\text{nnth } \text{ls } n) \ \sigma) \ ((\text{nmap} \ (\lambda x. x - (\text{nnth } \text{ls } n)) \ (\text{ndropn } n \ \text{ls}))) =$
 $(\forall i. i < (\text{nlength } \text{ls}) - n \longrightarrow ((\text{nsbn } \sigma \ (\text{nnth } \text{ls } (i+n)) \ (\text{nnth } \text{ls } ((\text{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 } \text{ls}) - n \longrightarrow ((\text{nsbn } \sigma \ (\text{nnth } \text{ls } (i+n)) \ (\text{nnth } \text{ls } ((\text{Suc } i)+n)) \) \models f)) =$
 $(\forall i. n \leq i \wedge i < (\text{nlength } \text{ls}) \longrightarrow ((\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{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: $(\text{powerinterval } f \ (\text{ntaken} \ (\text{nnth } \text{ls } n) \ \sigma) \ (\text{ntaken } n \ \text{ls}) \wedge$
 $\text{powerinterval } f \ (\text{ndropn} \ (\text{nnth } \text{ls } n) \ \sigma) \ ((\text{nmap} \ (\lambda x. x - (\text{nnth } \text{ls } n)) \ (\text{ndropn } n \ \text{ls})))) \implies$
 $\text{powerinterval } f \ \sigma \ \text{ls}$
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-nfuse:*

assumes *nidx ls1*
 $\text{nnth } \text{ls1 } 0 = 0$
nidx ls2
 $\text{nnth } \text{ls2 } 0 = 0$
nfinite ls1
 $\text{nlast } \text{ls1} = \text{cp}$
 $\text{cp} \leq \text{nlength } \sigma$
nfinite ls2
nfinite σ
 $\text{nlast } \text{ls2} = \text{the-enat}(\text{nlength } \sigma) - \text{cp}$
shows $\text{powerinterval } f \ \sigma \ (\text{nfuse } \text{ls1} \ (\text{nmap} \ (\lambda x. x + \text{cp}) \ \text{ls2})) =$
 $(\text{powerinterval } f \ (\text{ntaken } \text{cp } \sigma) \ \text{ls1} \wedge$
 $\text{powerinterval } f \ (\text{ndropn } \text{cp } \sigma) \ \text{ls2})$

proof –

have 1: $\text{nidx} \ (\text{nfuse } \text{ls1} \ (\text{nmap} \ (\lambda x. x + \text{cp}) \ \text{ls2}))$
using *assms nidx-nfuse[of ls1 (nmap (λx. x + cp) ls2)]*
using *Suc-ile-eq nidx-expand zero-enat-def by force*
have 2: $\text{cp} = (\text{nnth } \text{ls1} \ (\text{the-enat} \ (\text{nlength } \text{ls1})))$

```

using assms using nnth-nlast by blast
have 3: nlength ls1 ≤ nlength (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2))
  by (simp add: nfuse-nlength)
have 30: nfinite (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2))
  using assms
by (metis nfinite-conv-nlength-enat nfuse-nlength nlength-nmap plus-enat-simps(1))
have 31: nfirst (nmap ( $\lambda x. x + cp$ ) ls2) = nnth (nmap ( $\lambda x. x + cp$ ) ls2) 0
  by (metis ndropn-0 ndropn-nfirst)
have 32: nnth (nmap ( $\lambda x. x + cp$ ) ls2) 0 = cp
  using assms nnth-nmap[of 0 ls2 ( $\lambda x. x + cp$ )]
  using zero-enat-def by auto
have 33: nlast ls1 = nfirst (nmap ( $\lambda x. x + cp$ ) ls2)
  by (simp add: 31 32 assms(6))
have 34: nlast (nmap ( $\lambda x. x + cp$ ) ls2) = the-enat(nlength  $\sigma$ )
  using assms
  by (metis diff-add enat.simps(3) enat-ord-simps(1) enat-the-enat nfinite-conv-nlength-enat nlast-nmap)
have 4: nlast (nfuse ls1 (nmap ( $\lambda x. x + cp$ ) ls2)) = the-enat(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:  $\bigwedge j. j \leq \text{nlength } ls2 \longrightarrow$ 
   $\text{nnth } (\text{nmap } (\lambda x. x - cp) (\text{nmap } (\lambda x. x + cp) ls2)) j = \text{nnth } ls2 j$ 
  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)) ≤ 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 ( $\lambda x. x - \text{nnth } (\text{nfuse } ls1 (\text{nmap } (\lambda x. x + cp) ls2)) (\text{the-enat } (\text{nlength } ls1))$ )) (the-enat (nlength ls1)))

```

```

      (ndropn (the-enat (nlength ls1)) (nfuse ls1 (nmap (λx. x + cp) ls2))))))
using 72 73 30 74 76 75 4 using powerinterval-split[of (nfuse ls1 (nmap (λx. x+ cp) ls2))
  (the-enat (nlength ls1)) σ f]
by fastforce
have 81: powerinterval f (ntaken (nnth (nfuse ls1 (nmap (λx. x + cp) ls2)) (the-enat (nlength ls1))) σ)
  (ntaken (the-enat (nlength ls1)) (nfuse ls1 (nmap (λx. x + cp) ls2))) =
    powerinterval f (ntaken cp σ) ls1
using 5 70 by auto
have 82: powerinterval f (ndropn (nnth (nfuse ls1 (nmap (λx. x + cp) ls2)) (the-enat (nlength ls1))) σ)
  (nmap (λx. x - nnth (nfuse ls1 (nmap (λx. x + cp) ls2)) (the-enat (nlength ls1)))
    (ndropn (the-enat (nlength ls1)) (nfuse ls1 (nmap (λx. x + cp) ls2)))) =
    powerinterval f (ndropn cp σ) 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
          ¬nfinite ls2
          ¬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: ¬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:  $nlast\ ls1 = nfirst\ (nmap\ (\lambda x. x + cp)\ ls2)$ 
  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:  $ntaken\ (the-enat\ (nlength\ ls1))\ (nfuse\ ls1\ (nmap\ (\lambda x::nat. x + cp)\ ls2)) = ls1$ 
  by (simp add: 7 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: 7 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 7 assms(2) nfuse-nnth zero-enat-def zero-le)
have 74:  $enat\ (the-enat\ (nlength\ ls1)) \leq nlength\ (nfuse\ ls1\ (nmap\ (\lambda x. x + cp)\ ls2))$ 
  by (metis 70 assms(5) enat.simps(3) enat-the-enat min-def nfinite-nlength-enat ntaken-nlength
    order-eq-refl)
have 76:  $\neg nfinite\ \sigma$ 
  by (simp add: assms(9))
have 80:  $powerinterval\ f\ \sigma\ (nfuse\ ls1\ (nmap\ (\lambda x. x + cp)\ ls2)) =$ 
   $(powerinterval\ f\ (ntaken\ (nnth\ (nfuse\ ls1\ (nmap\ (\lambda x. x + cp)\ ls2))\ (the-enat\ (nlength\ ls1))))\ \sigma)$ 
   $(ntaken\ (the-enat\ (nlength\ ls1))\ (nfuse\ ls1\ (nmap\ (\lambda x. x + cp)\ ls2))) \wedge$ 
   $powerinterval\ f\ (ndropn\ (nnth\ (nfuse\ ls1\ (nmap\ (\lambda x. x + cp)\ ls2))\ (the-enat\ (nlength\ ls1))))\ \sigma)$ 
   $(nmap\ (\lambda x::nat. x - nnth\ (nfuse\ ls1\ (nmap\ (\lambda x. x + cp)\ ls2))\ (the-enat\ (nlength\ ls1))))$ 
   $(ndropn\ (the-enat\ (nlength\ ls1))\ (nfuse\ ls1\ (nmap\ (\lambda x. x + cp)\ ls2))))$ 
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

```

10.2.4 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 $nidx\ (cppl\ f\ g\ \sigma) \wedge nnth\ (cppl\ f\ g\ \sigma)\ 0 = 0 \wedge powerinterval\ f\ \sigma\ (cppl\ f\ g\ \sigma) \wedge$
 $((nfinite\ (cppl\ f\ g\ \sigma) \wedge nfinite\ \sigma \wedge nlast\ (cppl\ f\ g\ \sigma) = the-enat(nlength\ \sigma)) \vee$

$(\neg \text{nfinite } (\text{cppl } f \ g \ \sigma) \wedge \neg \text{nfinite } \sigma)) \wedge ((\text{pfilt } \sigma \ (\text{cppl } f \ g \ \sigma)) \models g)$
proof –
have 0: $\text{cppl } f \ g \ \sigma = (\epsilon \text{ ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \text{powerinterval } f \ \sigma \ \text{ls} \wedge$
 $((\text{nfinite } \text{ls} \wedge \text{nfinite } \sigma \wedge \text{nlast } \text{ls} = \text{the-enat}(\text{nlength } \sigma)) \vee$
 $(\neg \text{nfinite } \text{ls} \wedge \neg \text{nfinite } \sigma)) \wedge ((\text{pfilt } \sigma \ \text{ls}) \models g))$
using *cppl-def* **by** *blast*
have 1: $(\exists \text{ ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \text{powerinterval } f \ \sigma \ \text{ls} \wedge$
 $((\text{nfinite } \text{ls} \wedge \text{nfinite } \sigma \wedge \text{nlast } \text{ls} = \text{the-enat}(\text{nlength } \sigma)) \vee$
 $(\neg \text{nfinite } \text{ls} \wedge \neg \text{nfinite } \sigma)) \wedge ((\text{pfilt } \sigma \ \text{ls}) \models g))$
using *assms* **by** *auto*
have 2: $\text{nidx } (\text{cppl } f \ g \ \sigma) \wedge \text{nnth } (\text{cppl } f \ g \ \sigma) \ 0 = 0 \wedge \text{powerinterval } f \ \sigma \ (\text{cppl } f \ g \ \sigma) \wedge$
 $((\text{nfinite } (\text{cppl } f \ g \ \sigma) \wedge \text{nfinite } \sigma \wedge \text{nlast } (\text{cppl } f \ g \ \sigma) = \text{the-enat}(\text{nlength } \sigma)) \vee$
 $(\neg \text{nfinite } (\text{cppl } f \ g \ \sigma) \wedge \neg \text{nfinite } \sigma)) \wedge ((\text{pfilt } \sigma \ (\text{cppl } f \ g \ \sigma)) \models g)$
using 0 1
someI-ex[*of* $\lambda \text{ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \text{powerinterval } f \ \sigma \ \text{ls} \wedge$
 $((\text{nfinite } \text{ls} \wedge \text{nfinite } \sigma \wedge \text{nlast } \text{ls} = \text{the-enat}(\text{nlength } \sigma)) \vee$
 $(\neg \text{nfinite } \text{ls} \wedge \neg \text{nfinite } \sigma)) \wedge$
 $((\text{pfilt } \sigma \ \text{ls}) \models g)]$ **by** *simp*
show *?thesis*
using 2 **by** *blast*
qed

lemma *cppl-fprojection*:

$(\sigma \models f \ \text{fproj } g) =$
 $(\text{nidx } (\text{cppl } f \ g \ \sigma) \wedge \text{nnth } (\text{cppl } f \ g \ \sigma) \ 0 = 0 \wedge \text{powerinterval } f \ \sigma \ (\text{cppl } f \ g \ \sigma) \wedge$
 $\text{nfinite } (\text{cppl } f \ g \ \sigma) \wedge \text{nfinite } \sigma \wedge$
 $\text{nlast } (\text{cppl } f \ g \ \sigma) = \text{the-enat}(\text{nlength } \sigma) \wedge g \ (\text{pfilt } \sigma \ (\text{cppl } f \ g \ \sigma)))$
using *cppl-expand*[*of* $f \ \sigma \ g$]
by (*simp* *add*: *fprojection-d-def*)
 $(\text{metis } \text{nfinite-conv-nlength-enat } \text{the-enat.simps})$

lemma *cppl-oprojection*:

$(\sigma \models f \ \text{oproj } g) =$
 $(\text{nidx } (\text{cppl } f \ g \ \sigma) \wedge \text{nnth } (\text{cppl } f \ g \ \sigma) \ 0 = 0 \wedge \text{powerinterval } f \ \sigma \ (\text{cppl } f \ g \ \sigma) \wedge$
 $\neg \text{nfinite } (\text{cppl } f \ g \ \sigma) \wedge \neg \text{nfinite } \sigma \wedge g \ (\text{pfilt } \sigma \ (\text{cppl } f \ g \ \sigma)))$
using *cppl-expand* **by** (*simp* *add*: *oprojection-d-def*, *blast*)

lemma *cppl-empty*:

assumes $\text{nlength } \sigma = 0$
 $(\sigma \models f \ \text{fproj } g)$
shows $(\text{cppl } f \ g \ \sigma) = (\text{NNil } 0)$
using *assms* *cppl-fprojection*[*of* $f \ g \ \sigma$]
by (*metis* *gr-zeroI* *less-numeral-extra*(3) *ndropn-0* *ndropn-nlast* *nfinite-conv-nlength-enat*
nidx-less-last-1 *nnth-nlast* *the-enat.simps* *the-enat-0*)

lemma *cppl-empty-a*:

assumes $\text{nlength } \sigma = 0$
 $(\text{cppl } f \ g \ \sigma) = \text{NNil } 0$

```

      g(pfilt  $\sigma$  (NNil 0))
shows   ( $\sigma \models f$  fproj g)
proof -
  have 1: nidx (NNil 0)
    by (simp add: nidx-def)
  have 10: nnth (NNil 0) 0 = 0
    by (simp add: nnth-NNil)
  have 11: nfinite (NNil 0)
    by simp
  have 12: nfinite  $\sigma$ 
    by (simp add: assms(1) nlength-eq-enat-nfiniteD zero-enat-def)
  have 2: powerinterval f  $\sigma$  (NNil 0)
    by (simp add: powerinterval-def)
  have 3: nlast (NNil 0) = 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  $\sigma > 0$ 
      ( $\sigma \models f$  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  $\sigma > 0$ 
      ( $\sigma \models f$  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  $\sigma > 0$ 
      ( $\sigma \models f$  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  $\sigma > 0$ 
      ( $\sigma \models f$  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 $\text{nlast } (\text{cppl } f \ g \ \sigma) = \text{the-enat}(\text{nlength } \sigma)$

using *assms cppl-fprojection* by *blast*

lemma *cppl-sub-more*:

assumes $n < k$

$k \leq \text{nlength } \sigma$

$((\text{nsubn } \sigma \ n \ k) \models f \text{ fproj } g)$

shows $\text{nlength}(\text{cppl } f \ g \ (\text{nsubn } \sigma \ n \ k)) > 0$

using *assms cppl-fprojection*[of $f \ g \ (\text{nsubn } \sigma \ n \ k)$]

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*

$(\text{metis enat-minus-mono1 idiff-enat-enat min-def nfinite-conv-nlength-enat nidx-less-last-1}$
 $\text{nnth-nlast order-less-imp-le the-enat.simps})$

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: (nnth (cppl f g (nsubn σ x1a (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 (nsubn σ x1a x1a) \models f fproj g
shows nfirst((nmap (λx. x + x1a) (cppl f g (nsubn σ x1a x1a)))) = x1a
proof –
have 1: nfirst (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 ((nsubn σ x (nfirst ls)) \models f fproj g)
 $x < \text{nfirst } ls$
 $\text{nfirst } ls \leq \text{nlength } \sigma$
shows nlast((nmap (λy. y + x) (cppl f g (nsubn σ x (nfirst ls))))) = nfirst ls
proof –
have 01: (nidx (cppl f g (nsubn σ x (nfirst ls))) \wedge
nnth (cppl f g (nsubn σ x (nfirst ls))) 0 = 0 \wedge
powerinterval f (nsubn σ x (nfirst ls)) (cppl f g (nsubn σ x (nfirst ls))) \wedge
nfinite (cppl f g (nsubn σ x (nfirst ls))) \wedge nfinite (nsubn σ x (nfirst ls)) \wedge
nlast (cppl f g (nsubn σ x (nfirst ls))) = the-enat(nlength (nsubn σ x (nfirst ls))) \wedge
g (pfilt (nsubn σ x (nfirst ls)) (cppl f g (nsubn σ x (nfirst ls)))))
using cppl-fprojection assms **by** (simp add: cppl-fprojection)
have 02: nlast (cppl f g (nsubn σ x (nfirst ls))) =
the-enat(nlength (nsubn σ x (nfirst ls)))
using 01 **by** auto
have 04: $x < \text{nfirst } ls$
using assms **by** blast
have 05: nlength (nsubn σ x (nfirst ls)) = (nfirst ls) - x
using assms **by** (metis enat-minus-mono1 idiff-enat-enat min.orderE nsubn-nlength)
have 06: nlast((nmap (λy. y + x) (cppl f g (nsubn σ x (nfirst ls))))) =
((nfirst ls) - x) + x
using 01 05 **by** auto
show ?thesis **using** 04 06 **by** auto
qed

lemma cppl-nlast-i:

assumes ((nsubn σ (nnth ls i) (nnth ls (Suc i))) \models f fproj g)
 $(\text{nnth } ls \ i) < (\text{nnth } ls \ (\text{Suc } i))$
 $(\text{nnth } ls \ (\text{Suc } i)) \leq \text{nlength } \sigma$
shows nlast((nmap (λx. x + (nnth ls i)) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))))) =
(nnth ls (Suc i))
proof –
have 01: (nidx (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) \wedge


```

    nnth (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) 0 = 0 ∧
    powerinterval f (nsubn σ (nnth ls i) (nnth ls (Suc i)))
      (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ∧
    nfinite (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ∧
    nfinite (nsubn σ (nnth ls i) (nnth ls (Suc i))) ∧
    nlast (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) =
      the-enat(nlength (nsubn σ (nnth ls i) (nnth ls (Suc i)))) ∧
    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 by (simp add: cppl-fprojection)
have 02: nlast (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))) =
  the-enat(nlength (nsubn σ (nnth ls i) (nnth ls (Suc i))))
  using 01 by auto
have 04: (nnth ls i) < (nnth ls (Suc i))
  by (simp add: assms)
have 05: nlength (nsubn σ (nnth ls i) (nnth ls (Suc i))) = (nnth ls (Suc i)) - (nnth ls i)
  by (metis assms(3) enat-minus-mono1 idiff-enat-enat min.orderE nsubn-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

```

10.2.5 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

```

```

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.premis(1) Suc.premis(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
  case False
  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.premis(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  $nfinite\ ls \longleftrightarrow nfinite(lcppl\ f\ g\ \sigma\ ls)$  (is ?lhs  $\longleftrightarrow$  ?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 σ 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
qed

lemma lcppl-nlength-alt:
  assumes  $\neg nfinite\ ls$ 

```

```

      nidx ls
shows   nlength(lcppl f g σ ls) = (epred (nlength ls))
proof -
have 1: ¬ nfinite (lcppl f g σ ls)
  using assms(1) assms(2) lcppl-nfinite by blast
have 2: epred (nlength ls) = ∞
  using assms
  by (metis enat2-cases epred-Infty nlength-eq-enat-nfiniteD)
have 3: nlength (lcppl f g σ ls) = ∞
  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 σ 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 σ
    nlast ls = the-enat(nlength σ)
    nlength ls > 0
  shows   nlast (lcppl f g σ ls) =
    (nmap (λx. x + (nnth ls (the-enat (epred(nlength ls)))))
      (cppl f g (nsbn σ (nnth ls (the-enat (epred (nlength ls)))))
        (nnth ls (the-enat (nlength ls)))))
proof -
have 1: nlast (lcppl f g σ ls) = (nnth (lcppl f g σ ls) ((the-enat (nlength (lcppl f g σ ls)))))
  using assms lcppl-nfinite nnth-nlast by blast
have 2: nlast (lcppl f g σ ls) =
  (nmap (λx. x + (nnth ls ((the-enat (nlength (lcppl f g σ ls)))))
    (cppl f g (nsbn σ (nnth ls ((the-enat (nlength (lcppl f g σ ls)))))
      (nnth ls (Suc (the-enat (nlength (lcppl f g σ ls)))))
    )))
  using assms lcppl-nnth[of ls ((the-enat (nlength (lcppl f g σ ls)))] f g σ]
    lcppl-nlength[of ls f g σ]
  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 (λx. x + (nnth ls (the-enat (nlength (lcppl f g σ ls)))))
  (cppl f g (nsbn σ (nnth ls ((the-enat (nlength (lcppl f g σ ls)))))
    (nnth ls (Suc (the-enat (nlength (lcppl f g σ ls)))))
  ))) =

```

```

    (nmap (λx. x+ (nnth ls ((the-enat (epred(nlength ls))))))
      (cppl f g (nsubn σ (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 σ
    nlast ls = the-enat(nlength σ)
    ((nsubn σ (nnth ls (the-enat (epred(nlength ls)))) ) (nnth ls (the-enat(nlength ls)))) ⊨ f fproj g)
    nlength ls > 0
  shows nlast (nlast (lcppl f g σ ls)) = the-enat(nlength σ)
proof -
  have 2: (nlast (lcppl f g σ ls)) =
    (nmap (λx. x+ (nnth ls (the-enat (epred(nlength ls))))))
      (cppl f g (nsubn σ (nnth ls (the-enat (epred (nlength ls))))))
        (nnth ls (the-enat (nlength ls))))
  using assms lcppl-nlast[of ls σ f g] by blast
  have 3: nlast (nmap (λx. x+ (nnth ls (the-enat (epred(nlength ls))))))
    (cppl f g (nsubn σ (nnth ls (the-enat (epred (nlength ls))))))
      (nnth ls (the-enat (nlength ls)))) =
    (λx. x+ (nnth ls (the-enat (epred(nlength ls))))))
      (nlast (cppl f g (nsubn σ (nnth ls (the-enat (epred (nlength ls))))))
        (nnth ls (the-enat (nlength ls))))
  using nnth-nmap assms by (simp add: cppl-fprojection)
  have 4: nnth ls (the-enat (epred (nlength ls))) < nnth ls (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 (nnth ls (Suc (the-enat (epred (nlength ls))))) ≤ nlength σ
  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 (nmap (λx. x+ (nnth ls (the-enat (epred(nlength ls))))))
    (cppl f g (nsubn σ (nnth ls (the-enat (epred (nlength ls))))))
      (nnth ls (the-enat (nlength ls)))) =
    nnth ls (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 ls (Suc (the-enat (epred (nlength ls)))) = the-enat(nlength σ)
  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 nid $x$   $ls$ 
    nfinite  $ls$ 
    nfinite  $\sigma$ 
    nlast  $ls = the-enat(nlength\ \sigma)$ 
    nlength  $ls = 0$ 
  shows (nnth (lcppl  $f\ g\ \sigma\ ls$ ) 0) =
    (nmap ( $\lambda x. x + (nnth\ ls\ 0)$ ) (cppl  $f\ g\ (nsubn\ \sigma\ (nnth\ ls\ 0)\ (nnth\ ls\ 0))$ ))
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  $ls = the-enat(nlength\ \sigma)$ 
    ( $\forall i. i < nlength\ ls \longrightarrow ((nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$ )
    nlength  $ls > 0$ 
  shows nfirst(nfirst((lcppl  $f\ g\ \sigma\ ls$ ))) = nfirst  $ls$ 
proof -
  have 01: (nfirst((lcppl  $f\ g\ \sigma\ ls$ ))) =
    (nmap ( $\lambda x. x + (nnth\ ls\ 0)$ ) (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 ( $\lambda x. x + (nnth\ ls\ 0)$ ) (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 $x$   $ls$ 
     $\neg nfinite\ ls$ 

```

$\neg nfinite\ \sigma$
 $(\forall i. i < nlength\ ls \longrightarrow ((nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g))$
shows $nfirst(nfirst((lcppl\ f\ g\ \sigma\ ls))) = nfirst\ ls$
proof –
have 01: $(nfirst((lcppl\ f\ g\ \sigma\ ls))) =$
 $(nmap\ (\lambda x. x + (nnth\ ls\ 0))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ 0)\ (nnth\ ls\ (Suc\ 0))))$
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: $nfirst((nmap\ (\lambda x. x + (nnth\ ls\ 0))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ 0)\ (nnth\ ls\ (Suc\ 0)))))) =$
 $(nnth\ ls\ 0)$
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*
 $nidx\ ls$
 $nfinite\ \sigma$
 $nlast\ ls = the-enat(nlength\ \sigma)$
 $((nlength\ ls = 0 \wedge ((nsubn\ \sigma\ (nnth\ ls\ 0)\ (nnth\ ls\ 0)) \models f\ fproj\ g)) \vee$
 $(nlength\ ls > 0 \wedge (\forall i. i < nlength\ ls \longrightarrow ((nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g))))$
shows $nlastnfirst\ (lcppl\ f\ g\ \sigma\ ls)$
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 $nlast\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i) = nfirst\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ (Suc\ i))$
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: $(\forall i. i < nlength\ ls \longrightarrow ((nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g))$
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*)

```

      iless-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 iless-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 (λx::nat. x + nnth ls i) (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i))))) =
  (nnth ls (Suc i))
  using assms 2 3 4 5 6 cppl-nlast-i[of f g σ ls i]
  using 71 72 by blast
have 9: nfirst (nmap (λx::nat. x + nnth ls (Suc i))
  (cppl f g (nsubn σ (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 ¬nfinite ls
        nidx ls
        ¬nfinite σ
        (∀ i. i < nlength ls → ((nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ f fproj g))
shows   nlastnfirst (lcppl f g σ ls)
proof (auto simp add: nlastnfirst-def1)
fix i
assume a: enat (Suc i) ≤ nlength (lcppl f g σ ls)
show   nlast (nnth (lcppl f g σ ls) i) = nfirst (nnth (lcppl f g σ ls) (Suc i))
proof -
  have 2: (∀ i. i < nlength ls → ((nsubn σ (nnth ls i) (nnth ls (Suc i))) ⊨ 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 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 4 assms(1) eSuc-enat ileI1 nlength-eq-enat-nfiniteD order-neq-le-trans)
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 8: nlast (nmap (λx::nat. x + nnth ls i) (cppl f g (nsubn σ (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 (λx::nat. x + nnth ls (Suc i))
  (cppl f g (nsubn σ (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: $\text{nlength } ls = 0 \longrightarrow \text{nlength}(\text{nfusecat } (\text{lcpl } f \ g \ \sigma \ ls)) = 0$
 using 2 3 by auto
have 5: $\text{nfinite } (\text{lcpl } f \ g \ \sigma \ ls)$
 using $\text{assms}(1) \ \text{assms}(2) \ \text{lcpl-nfinite}$ by blast
have 6: $\text{all-nfinite } (\text{lcpl } f \ g \ \sigma \ ls)$
by (metis (no-types, lifting) 20 all-nfinite-nnth-a $\text{assms}(1) \ \text{assms}(2) \ \text{assms}(5) \ \text{co.enat.exhaust-sel}$
 $\text{cppl-fprojection} \ \text{iless-Suc-eq} \ \text{lcpl-nlength} \ \text{lcpl-nnth} \ \text{nfinite-nmap} \ \text{nnth-NNil}$
 $\text{not-less-iff-gr-or-eq}$)
have 7: $\text{nlength } (\text{lcpl } f \ g \ \sigma \ ls) = (\text{epred}(\text{nlength } ls))$
 by (metis $\text{assms}(1) \ \text{assms}(2) \ \text{assms}(5) \ \text{epred-0} \ \text{lcpl-nlength} \ \text{lcpl-nlength-zero}$)
have 8: $\text{nlength } ls > 0 \longrightarrow$
 $\text{nlength}(\text{nfusecat } (\text{lcpl } f \ g \ \sigma \ ls)) =$
 $(\sum k=0..(\text{the-enat } (\text{epred}(\text{nlength } ls))). ((\text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k))))$
 using $\text{nfusecat-nlength-nfinite[of } (\text{lcpl } f \ g \ \sigma \ ls)] \ 5 \ 6 \ 7 \ 1$ by presburger
show ?thesis
 using 4 8 by blast
qed

lemma $\text{lcpl-nfusecat-nidx}$:

assumes $\text{nidx } ls$
 $\text{nnth } ls \ 0 = 0$
 $\text{nfinite } ls$
 $\text{nfinite } \sigma$
 $\text{nlast } ls = \text{the-enat}(\text{nlength } \sigma)$
 $(\forall i < \text{nlength } ls. (\text{nsubn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) \models f \ \text{fproj } \ g)$
 $\text{nlength } ls > 0$
shows $\text{nidx } (\text{nfusecat } (\text{lcpl } f \ g \ \sigma \ ls))$
proof –
have 0: $\text{nlength } \sigma > 0 \longrightarrow \text{nlength } ls > 0$
 using assms by auto
have 2: $\text{nlength } \sigma > 0 \longrightarrow \text{nlastnfirst } (\text{lcpl } f \ g \ \sigma \ ls)$
 using $\text{assms} \ \text{lcpl-nfusecat-nlastnfirst}$ 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 $\text{assms} \ \text{cppl-fprojection}$ by auto
have 4: $(\forall i < \text{nlength } ls.$
 $\text{nidx } (\text{nmap } (\lambda x. x + (\text{nnth } ls \ i)) \ (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))))$
 using 3 by (simp add: $\text{Suc-ile-eq} \ \text{nidx-expand}$)
have 5: $\text{nlength } \sigma > 0 \longrightarrow$
 $(\forall i < \text{nlength } ls. \text{nidx } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ i))$
 using assms by (simp add: 4 lcpl-nnth)
have 6: $\text{nlength } \sigma > 0 \longrightarrow \text{nlength } (\text{lcpl } f \ g \ \sigma \ ls) = \text{epred}(\text{nlength } ls)$
 using $\text{assms} \ \text{lcpl-nlength}$ by blast
have 7: $\text{nlength } \sigma > 0 \longrightarrow$
 $(\forall i \leq \text{nlength } ls.$
 $\text{nidx } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ i))$
 using 5 assms
by (metis 6 $\text{Extended-Nat.eSuc-mono} \ \text{antisym-conv2} \ \text{co.enat.exhaust-sel} \ \text{enat-the-enat} \ \text{epred-simps}(5)$
 $\text{i0-ne-infinity} \ \text{iless-Suc-eq} \ \text{lcpl-nfinite} \ \text{nnth-beyond} \ \text{nnth-nlast}$)
have 68: $\bigwedge i. i < \text{nlength } ls \implies 0 < \text{nlength } (\text{nsubn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) \implies$

$(nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models (f\ fproj\ g) \implies$
 $0 < nlength\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$
using *cppl-more* **by** *blast*
have 69: $\bigwedge i. i < nlength\ ls \implies 0 < nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
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: $nlength\ \sigma > 0 \longrightarrow (\forall i < nlength\ ls. (0::nat) < nlength\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i))$
using *lcppl-nnth[of ls - f g σ] 68 69 assms(1) assms(6)* **by** *auto*
have 700: $nlength\ \sigma > 0 \longrightarrow (\forall lx \in nset\ (lcppl\ f\ g\ \sigma\ ls). (0::nat) < nlength\ lx)$
using 70 *assms*
by $(metis\ co-enat.exhaust-sel\ iless-Suc-eq\ in-nset-conv-nnth\ lcppl-nlength\ less-numeral-extra(3))$
have 71: $nlength\ \sigma > 0 \longrightarrow (\forall lx \in nset\ (lcppl\ f\ g\ \sigma\ ls). nfinite\ lx)$
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: $nlength\ \sigma > 0 \longrightarrow$
 $nidx\ (nfusecat\ (\ (lcppl\ f\ g\ \sigma\ ls)))$
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*
 $nnth\ ls\ 0 = 0$
 $\neg nfinite\ ls$
 $\neg nfinite\ \sigma$
 $(\forall i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$
shows $nidx\ (nfusecat\ (\ (lcppl\ f\ g\ \sigma\ ls)))$

proof –

have 2: $nlastnfirst\ (lcppl\ f\ g\ \sigma\ ls)$
using *assms lcppl-nfusecat-nlastnfirst-alt* **by** *blast*
have 3: $(\forall i < nlength\ ls. nidx\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$
using *assms cppl-fprojection* **by** *auto*
have 4: $(\forall i < nlength\ ls.$
 $nidx\ (nmap\ (\lambda x. x + (nnth\ ls\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))$
using 3 **by** $(simp\ add:\ Suc-ile-eq\ nidx-expand)$
have 5: $(\forall i < nlength\ ls. nidx\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i))$
using *assms* **by** $(simp\ add:\ 4\ lcppl-nnth)$
have 7: $(\forall i < nlength\ ls.$
 $nidx\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i))$
using 5 **by** *simp*
have 60: $\bigwedge j. (nsubn\ \sigma\ (nnth\ ls\ j)\ (nnth\ ls\ (Suc\ j))) \models (f\ fproj\ g)$
by $(metis\ assms(3)\ assms(5)\ leI\ nfinite-ntaken\ ntaken-all)$
have 61: $\bigwedge j. nnth\ ls\ j < nnth\ ls\ (Suc\ j)$
by $(metis\ Suc-ile-eq\ assms(1)\ assms(3)\ nfinite-ntaken\ nidx-expand\ not-le-imp-less\ ntaken-all)$
have 62: $\bigwedge j. enat\ (nnth\ ls\ (Suc\ j)) \leq nlength\ \sigma$

by (meson assms(4) enat-iless enat-less-imp-le nfinite-conv-nlength-enat)
 have 63: $\bigwedge j. \text{nlast } (\text{nmap } (\lambda x::\text{nat}. x + \text{nnth } ls \ j) \ (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ j) \ (\text{nnth } ls \ (\text{Suc } j)))) = \text{nnth } ls \ (\text{Suc } j)$
 using 60 61 62 cppl-nlast-i by blast
 have 68: $\bigwedge i. i < \text{nlength } ls \implies 0 < \text{nlength } (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) \implies$
 $(\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) \models (f \ \text{fproj } \ g) \implies$
 $0 < \text{nlength } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))))$
 using cppl-more by blast
 have 69: $\bigwedge i. i < \text{nlength } ls \implies 0 < \text{nlength } (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)))$
 using assms unfolding nid_x-expand using 61 62 nsbn-nlength-gr-one by blast
 have 700: $\text{nlength } \sigma > 0 \longrightarrow (\forall i < \text{nlength } ls. (0::\text{enat}) < \text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ i))$
 using lcpl-nnth[of $ls - f \ g \ \sigma$] 68 69 assms by (simp)
 have 701: $\text{nlength } \sigma > 0 \longrightarrow (\forall lx \in \text{nset } (\text{lcpl } f \ g \ \sigma \ ls). (0::\text{enat}) < \text{nlength } lx)$
 using 700 assms
 by (metis co.enat.exhaust-sel illess-Suc-eq in-nset-conv-nnth lcpl-nlength-alt
 nlength-eq-enat-nfiniteD zero-enat-def)
 have 70: $(\forall lx \in \text{nset } (\text{lcpl } f \ g \ \sigma \ ls). (0::\text{enat}) < \text{nlength } lx)$
 using 701 assms nfinite-conv-nlength-enat zero-enat-def by fastforce
 have 71: $(\forall lx \in \text{nset } (\text{lcpl } f \ g \ \sigma \ ls). \text{nfinite } lx)$
 using assms
 by (metis (no-types, lifting) co.enat.exhaust-sel cppl-fprojection illess-Suc-eq in-nset-conv-nnth
 lcpl-nlength-alt lcpl-nnth nfinite-nmap nlength-eq-enat-nfiniteD zero-enat-def)
 have 8: nid_x (nfusecat ((lcpl $f \ g \ \sigma \ ls$)))
 using assms nid_x-nfusecat[of ((lcpl $f \ g \ \sigma \ ls$))] 70 71
 by (metis 2 7 co.enat.exhaust-sel illess-Suc-eq lcpl-nlength-alt
 nlength-eq-enat-nfiniteD zero-enat-def)
 from 8 show ?thesis using assms by blast
 qed

lemma lcpl-nlength-all-nfinite:

assumes nid_x ls

$\text{nnth } ls \ 0 = 0$

$\text{nfinite } ls$

$\text{nfinite } \sigma$

$\text{nlast } ls = \text{the-enat}(\text{nlength } \sigma)$

$(\forall i < \text{nlength } ls. (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i))) \models f \ \text{fproj } \ g)$
 $\text{nlength } \sigma > 0$

shows $(\forall j \leq \text{nlength } (\text{lcpl } f \ g \ \sigma \ ls). \text{nfinite}(\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ j))$

proof

fix j

show $j \leq \text{nlength } (\text{lcpl } f \ g \ \sigma \ ls) \longrightarrow \text{nfinite } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ j)$

proof –

have 1: $\text{nlength } \sigma > 0 \longrightarrow \text{nlastnfirst } (\text{lcpl } f \ g \ \sigma \ ls)$

using assms

by (metis enat.distinct(2) enat-the-enat i0-less lcpl-nfusecat-nlastnfirst
 nfinite-conv-nlength-enat nnth-nlast the-enat-0)

have 2: $\text{nlength } \sigma > 0 \longrightarrow \text{nlength } ls > 0$

using assms

by (metis enat.distinct(2) enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0)

have 3: $\text{nlength } \sigma > 0 \longrightarrow j \leq \text{nlength } (\text{lcpl } f \ g \ \sigma \ ls) \longrightarrow$

```

      (nnth (lcppl f g σ ls) j) =
      (nmap (λx. x+ (nnth ls j)) (cppl f g (nsubn σ (nnth ls j) (nnth ls (Suc j)))))
using assms lcppl-nnth[of ls j f g σ]
by (metis 2 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength not-gr-zero)
have 4: nlength σ > 0  $\longrightarrow$  j ≤ nlength (lcppl f g σ ls)  $\longrightarrow$ 
      nfinite (nmap (λx. x+ (nnth ls j)) (cppl f g (nsubn σ (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 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  $(\forall j \leq \text{nlength } (lcppl \ f \ g \ \sigma \ ls). \text{nlength}(\text{nnth } (lcppl \ f \ g \ \sigma \ ls) \ j) > 0)$ 
proof
  fix j
  show j ≤ nlength (lcppl f g σ ls)  $\longrightarrow$  0 < nlength (nnth (lcppl f g σ ls) j)
  proof –
  have 1: nlength σ > 0  $\longrightarrow$  nlastnfirst (lcppl f g σ 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 σ > 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 σ > 0  $\longrightarrow$  j ≤ nlength (lcppl f g σ ls)  $\longrightarrow$ 
      (nnth (lcppl f g σ ls) j) =
      (nmap (λx. x+ (nnth ls j)) (cppl f g (nsubn σ (nnth ls j) (nnth ls (Suc j)))))
    using assms lcppl-nnth[of ls j f g σ]
    by (metis 2 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength not-gr-zero)
  have 4: nlength σ > 0  $\longrightarrow$  j ≤ nlength (lcppl f g σ ls)  $\longrightarrow$ 
      nlength (nmap (λx. x+ (nnth ls j)) (cppl f g (nsubn σ (nnth ls j) (nnth ls (Suc j))))) > 0
    using assms 2 cppl-more[of (nsubn σ (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 σ*

$(\forall i < \text{nlength } ls. (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) \models f \text{ fproj } g)$

shows $(\forall j \leq \text{nlength } (\text{lcppl } f g \sigma ls). \text{nfinite}(\text{nnth } (\text{lcppl } f g \sigma ls) j))$

proof

fix *j*

show $j \leq \text{nlength } (\text{lcppl } f g \sigma ls) \longrightarrow \text{nfinite } (\text{nnth } (\text{lcppl } f g \sigma ls) j)$

proof –

have 1: *nlastnfirst (lcppl f g σ 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: $j \leq \text{nlength } (\text{lcppl } f g \sigma ls) \longrightarrow$

$\text{nfinite } (\text{nmap } (\lambda x. x + (\text{nnth } ls j)) (\text{cppl } f g (\text{nsubn } \sigma (\text{nnth } ls j) (\text{nnth } ls (\text{Suc } j))))))$

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*

nnth ls 0 = 0

\neg *nfinite ls*

\neg *nfinite σ*

$(\forall i < \text{nlength } ls. (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) \models f \text{ fproj } g)$

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: *nlastnfirst (lcppl f g σ 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*

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*
 $nnth\ ls\ 0 = 0$
 $\neg finite\ ls$
 $\neg finite\ \sigma$
 $nlast\ ls = the-enat(nlength\ \sigma)$
 $(\forall\ i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$
 $nlength\ \sigma > 0$
 $j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls)$
shows $nlast(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j) = (nnth\ ls\ (Suc\ j))$

proof –

have 0: $nlength\ \sigma > 0 \longrightarrow nlength\ ls > 0$
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: $nlength\ \sigma > 0 \longrightarrow$
 $j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls) \longrightarrow$
 $nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j =$
 $(nmap\ (\lambda x. x + (nnth\ ls\ j))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ j)\ (nnth\ ls\ (Suc\ j)))) =$
using *assms lcppl-nnth[of ls j f g sigma]*
by (*metis 0 co.enat.exhaust-sel iless-Suc-eq lcppl-nlength not-gr-zero*)
have 2: $nlength\ \sigma > 0 \longrightarrow$
 $j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls) \longrightarrow (nnth\ ls\ (Suc\ j)) \leq nlength\ \sigma$
using *assms*
by (*metis dual-order.eq-iff enat.simps(3) enat-ord-simps(1) enat-the-enat nfinite-conv-nlength-enat nidx-all-le-nlast nnth-beyond not-le-imp-less*)
have 3: $nlength\ \sigma > 0 \longrightarrow$
 $j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls) \longrightarrow$
 $nlast\ (nmap\ (\lambda x. x + (nnth\ ls\ j))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ j)\ (nnth\ ls\ (Suc\ j)))) =$
 $(nnth\ ls\ (Suc\ j))$
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 nidx-expand*)
show *?thesis* **using** 1 3 *assms*
by *presburger*
qed

lemma *lcppl-nlast-nnth-alt*:

assumes *nidx ls*
 $nnth\ ls\ 0 = 0$
 $\neg finite\ ls$
 $\neg finite\ \sigma$
 $(\forall\ i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$
 $j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls)$
shows $nlast(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j) = (nnth\ ls\ (Suc\ j))$

proof –

have 1:

$$j \leq \text{nlength } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \longrightarrow \\ \text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ j = \\ (\text{nmap } (\lambda x. x + (\text{nnth } \text{ls } j)) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } \text{ls } j) \ (\text{nnth } \text{ls } (\text{Suc } j)) \)))$$

using *assms lcpl-nnth[of ls j f g σ]* **by** (*metis leI nfinite-ntaken ntaken-all*)

have 2:

$$j \leq \text{nlength } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \longrightarrow (\text{nnth } \text{ls } (\text{Suc } j)) \leq \text{nlength } \sigma$$

using *assms*

by (*simp add: nfinite-conv-nlength-enat*)

have 3: $j \leq \text{nlength } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \longrightarrow$

$$\text{nlast } (\text{nmap } (\lambda x. x + (\text{nnth } \text{ls } j)) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } \text{ls } j) \ (\text{nnth } \text{ls } (\text{Suc } j)) \)))) = \\ (\text{nnth } \text{ls } (\text{Suc } j))$$

using *assms cppl-nlast-i[of f g σ ls j]*

by (*metis 2 co.enat.exhaust-sel eSuc-enat ileI1 iless-Suc-eq lcpl-nlength-alt*
nidx-expand nlength-eq-enat-nfiniteD zero-enat-def)

show *?thesis* **using** 1 3 *assms*

by *presburger*

qed

lemma *lcpl-nfusecat-pfilt-fpower-help:*

assumes *nidx ls*

nnth ls 0 = 0

nfinite ls

nfinite σ

nlast ls = the-enat(nlength σ)

$(\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$

shows $(\forall i < \text{nlength } \text{ls}. g \ (\text{pfilt } \sigma \ (\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ i)) \)$

proof –

have 1: $(\forall i < \text{nlength } \text{ls}. g \ (\text{pfilt } (\text{nsubn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i))) \\ (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)) \)) \))$

using *assms cppl-fprojection* **by** *blast*

have 2: $\text{nlength } \sigma > 0 \longrightarrow$

$$(\forall i < \text{nlength } \text{ls}. \\ (\text{pfilt } \sigma \ (\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ i)) = \\ (\text{pfilt } \sigma \ (\text{nmap } (\lambda x. x + (\text{nnth } \text{ls } i)) (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)) \)))) \))$$

using *assms* **by** (*simp add: lcpl-nnth*)

have 3: $\text{nlength } \sigma > 0 \longrightarrow$

$$(\forall i < \text{nlength } \text{ls}. \\ \text{nlength } (\text{pfilt } \sigma \ (\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ i)) = \\ \text{nlength } (\text{pfilt } (\text{nsubn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i))) \\ (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)) \)) \)) \\)$$

by (*simp add: 2 pfilt-nlength*)

have 4: $\text{nlength } \sigma > 0 \longrightarrow$

$$(\forall i < \text{nlength } \text{ls}. \\ (\forall j \leq \text{nlength } (\text{pfilt } \sigma \ (\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ i)).$$

$$\begin{aligned}
& (nnth (pfilt \sigma (nmap (\lambda x. x + (nnth ls i)) \\
& \quad (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)) \\
&) \\
&) \\
& \text{using } nnth-nmap \text{ pfilt-nmap by (metis 2 nlength-nmap)} \\
\text{have 5: } nlength \sigma > 0 \longrightarrow \\
& (\forall i < nlength ls. \\
& (\forall j \leq nlength (pfilt \sigma (nnth (lcppl f g \sigma ls) i)). \\
& \quad (nnth (pfilt (nsubn \sigma (nnth ls i) (nnth ls (Suc i)))) \\
& \quad \quad (cppl f g (nsubn \sigma (nnth ls i) (nnth ls (Suc i)))) j) = \\
& \quad (nnth (nsubn \sigma (nnth ls i) (nnth ls (Suc i))) \\
& \quad \quad (nnth (cppl f g (nsubn \sigma (nnth ls i) (nnth ls (Suc i)))) j)) \\
&)) \\
& \text{by (simp add: 3 pfilt-nnth)} \\
\text{have 6: } nlength \sigma > 0 \longrightarrow \\
& (\forall i < nlength ls. \\
& \quad (nnth ls (Suc i)) \leq nlength \sigma \\
&) \\
& \text{using } asms \\
& \text{by (metis eSuc-enat enat-le-plus-same(2) enat-ord-simps(1) gen-nlength-def ileI1} \\
& \quad nfinite-conv-nlength-enat nidx-less-eq nlength-code nnth-nlast the-enat.simps)} \\
\text{have 7: } nlength \sigma > 0 \longrightarrow \\
& (\forall i < nlength ls. \\
& \quad (nnth ls i) \leq (nnth ls (Suc i)) \\
&) \\
& \text{using } asms \\
& \text{by (metis eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc)} \\
\text{have 70: } \bigwedge i j . i < nlength ls \implies (nnth ls i) < (nnth ls (Suc i)) \implies \\
& enat (nnth ls (Suc i)) \leq nlength \sigma \implies \\
& (nsubn \sigma (nnth ls i) (nnth ls (Suc i))) \models (f fproj g) \implies \\
& enat j < nlength (cppl f g (nsubn \sigma (nnth ls i) (nnth ls (Suc i)))) \implies \\
& 0 \leq nnth (cppl f g (nsubn \sigma (nnth ls i) (nnth ls (Suc i)))) j \wedge \\
& nnth (cppl f g (nsubn \sigma (nnth ls i) (nnth ls (Suc i)))) j \leq (nnth ls (Suc i)) - (nnth ls i) \\
& \text{using cppl-bounds by blast} \\
\text{have 71: } \bigwedge i . i < nlength ls \implies (nsubn \sigma (nnth ls i) (nnth ls (Suc i))) \models (f fproj g) \implies \\
& nnth ls i < nnth ls (Suc i) \implies \\
& enat (nnth ls (Suc i)) \leq nlength \sigma \implies \\
& nlast (nmap (\lambda x. x + nnth ls i) (cppl f g (nsubn \sigma (nnth ls i) (nnth ls (Suc i))))) = \\
& nnth ls (Suc i) \\
& \text{using cppl-nlast-i by blast} \\
\text{have 8: } nlength \sigma > 0 \longrightarrow \\
& (\forall i < nlength ls. \\
& (\forall j \leq nlength (pfilt \sigma (nnth (lcppl f g \sigma ls) i)). \\
& \quad (nnth (cppl f g (nsubn \sigma (nnth ls i) (nnth ls (Suc i)))) j) \leq \\
& \quad (nnth ls (Suc i)) - (nnth ls i) \\
&)) \\
& \text{using 6 asms 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)
\end{aligned}$$

$ntaken-nlast)$
have 9: $nlength\ \sigma > 0 \longrightarrow$
 $(\forall\ i < nlength\ ls.$
 $(\forall\ j \leq nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$
 $(nnth\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
 $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ j)) =$
 $(nnth\ \sigma\ ((nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ j) + (nnth\ ls\ i)))$
 $))$
using 6 7 8
by (*simp add: add.commute nsubn-def1 ntaken-nnth*)
have 10: $nlength\ \sigma > 0 \longrightarrow$
 $(\forall\ i < nlength\ ls.$
 $(\forall\ j \leq nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$
 $(pfilt\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
 $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ j) =$
 $(pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i))$
 $))$
using 2 3 4 9 **by** (*simp add: pfilt-expand pfilt-nlength pfilt-nnth*)
show ?thesis **using** 1 10 *assms* **by** *fastforce*
qed

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:  $\bigwedge i\ j. i < nlength\ ls \implies (nnth\ ls\ i) < (nnth\ ls\ (Suc\ i)) \implies$ 
   $enat\ (nnth\ ls\ (Suc\ i)) \leq nlength\ \sigma \implies$ 
   $(nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models (f\ fproj\ g) \implies$ 
   $enat\ j < nlength\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \implies$ 
   $0 \leq nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ j \wedge$ 
   $nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ j \leq (nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i)$ 
using cppl-bounds by blast
have 71:  $\bigwedge i. i < nlength\ ls \implies (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models (f\ fproj\ g) \implies$ 
   $nnth\ ls\ i < nnth\ ls\ (Suc\ i) \implies$ 
   $enat\ (nnth\ ls\ (Suc\ i)) \leq nlength\ \sigma \implies$ 
   $nlast\ (nmap\ (\lambda x. x + nnth\ ls\ i)\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) =$ 
   $nnth\ ls\ (Suc\ i)$ 
using cppl-nlast-i by blast
have 8: ( $\forall i < nlength\ ls.$ 
  ( $\forall j \leq nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$ 
    ( $nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$ ))  $j \leq$ 
    ( $nnth\ ls\ (Suc\ i) - (nnth\ ls\ i)$ 
  ))
using assms 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: ( $\forall i < nlength\ ls.$ 
  ( $\forall j \leq nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$ 
    ( $nnth\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$ 
      ( $nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$ ))  $j$ ) =
    ( $nnth\ \sigma\ ((nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$ ))  $j$ ) + (nnth\ ls\ i))
  ))
using 6 7 8 by (simp add: add.commute nsubn-def1 ntaken-nnth)
have 10: ( $\forall i < nlength\ ls.$ 
  ( $\forall j \leq nlength\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)).$ 
    ( $pfilt\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$ 

```

```

      (cppl f g (nsubn σ (nnth ls i) (nnth ls (Suc i)) )) ) =
    (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

```

10.2.6 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) ⟷ is-NNil nells ∧ 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) ⟷

*(∃ nell nells'. nells = (NCons nell nells') ∧ b = a+(the-enat (nlength nell)) ∧
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 ∧ nlength(nfirst nells) = 0  $\longrightarrow$ 
  nlength (addzero (lsum nells 0)) = 0)
 $\wedge$ 
(nlength nells = 0 ∧ 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 ≤ 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 ≤ nlength (nells::'a nelist nelist) + (1::enat)
      using Suc.prem1(2) Suc-ile-eq by auto
    have 3: (∑ k::nat = 0::nat..i - (1::nat). nlength (nnth nells k)) =
      (∑ k::nat = 1::nat..i. nlength (nnth nells (k - (1::nat))))
      using 1 2 Suc.hyps by blast
    have 4: (∑ k::nat = 0::nat..Suc i - (1::nat). nlength (nnth nells k)) =
      (∑ k::nat = 0::nat..i. nlength (nnth nells k))
      by simp
    have 5: (∑ k::nat = 0::nat..i. nlength (nnth nells k)) =
      nlength (nnth nells i) + (∑ k::nat = 0::nat..(i-1). nlength (nnth nells k))
      by (simp add: Suc)
    have 6: nlength (nnth nells i) + (∑ k::nat = 1::nat..i. nlength (nnth 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: $\text{nnth } (\text{lsum nells } a) (\text{Suc } i) = \text{nnth } (\text{lsum } x22 (a + (\text{the-enat } (\text{nlength } x21)))) i$
by (*simp add: NCons*)

have 2: $\text{enat } (i::\text{nat}) \leq \text{nlength } x22$

using *NCons Suc.premis Suc-ile-eq* **by** *auto*

have 20: *all-nfinite* $x22$

by (*simp add: NCons Suc.premis(2)*)

have 3: $\text{nnth } (\text{lsum } x22 (a + (\text{the-enat } (\text{nlength } x21)))) i =$

$(a + (\text{the-enat } (\text{nlength } x21))) +$
 $(\sum k::\text{nat} = 0::\text{nat}..i. (\text{the-enat } (\text{nlength } (\text{nnth } x22 k))))$

using *Suc.hyps[of x22 (a + (the-enat (nlength x21))) 2 20* **by** *blast*

have 4: $a + (\sum k::\text{nat} = 0::\text{nat}.. \text{Suc } i. (\text{the-enat } (\text{nlength } (\text{nnth nells } k)))) =$
 $a + (\text{the-enat } ((\text{nlength } (\text{nnth nells } 0)))) +$

$(\sum k::\text{nat} = 1::\text{nat}.. \text{Suc } i. (\text{the-enat } (\text{nlength } (\text{nnth nells } k))))$

by (*simp add: sum.atLeast-Suc-atMost*)

have 5: $(\text{nlength } (\text{nnth nells } 0)) = \text{nlength } x21$

by (*simp add: NCons*)

have 6: $(\sum k::\text{nat} = 1::\text{nat}.. \text{Suc } i. (\text{the-enat } (\text{nlength } (\text{nnth nells } k)))) =$
 $(\sum k::\text{nat} = 0::\text{nat}.. i. (\text{the-enat } (\text{nlength } (\text{nnth nells } (k+1)))))$

using *sum.shift-bounds-cl-nat-ivl[of $\lambda k. (\text{the-enat } (\text{nlength } (\text{nnth nells } k)))$ 0 1 i]*

by *simp*

have 7: $(\sum k::\text{nat} = 0::\text{nat}.. i. (\text{the-enat } (\text{nlength } (\text{nnth nells } (k+1))))) =$
 $(\sum k::\text{nat} = 0::\text{nat}.. i. (\text{the-enat } (\text{nlength } (\text{nnth } x22 (k)))))$

using *NCons* **by** *auto*

show *?thesis*

using 1 3 4 5 6 7 **by** *presburger*

qed

qed

qed

lemma *lsum-addzero-nnth*:

assumes $i \leq \text{nlength } (\text{addzero } (\text{lsum nells } 0))$
 $\text{nfinite } (\text{nfirst nells})$

shows $(\text{nlength nells} = 0 \wedge \text{nlength}(\text{nfirst nells}) = 0 \longrightarrow$
 $\text{nnth } (\text{addzero } (\text{lsum nells } 0)) i = (\text{nnth } (\text{lsum nells } 0) i))$

\wedge

$(\text{nlength nells} = 0 \wedge \text{nlength}(\text{nfirst nells}) > 0 \longrightarrow$
 $\text{nnth } (\text{addzero } (\text{lsum nells } 0)) i = (\text{nnth } (\text{NCons } 0 (\text{lsum nells } 0))) i)$

\wedge

$(\text{nlength nells} > 0 \longrightarrow$
 $\text{nnth } (\text{addzero } (\text{lsum nells } 0)) i = (\text{nnth } (\text{NCons } 0 (\text{lsum nells } 0)) i))$

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*
 $i \leq nlength\ nells$
shows $(\sum k = 0..i. (nlength\ (nnth\ nells\ k))) < \infty$
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: $(\sum k = 0..Suc\ i. nlength\ (nnth\ nells\ k)) =$
 $nlength\ (nnth\ nells\ 0) + (\sum k = 1..Suc\ i. nlength\ (nnth\ nells\ k))$
by (*metis* *One-nat-def* *sum.atLeast0-atMost-Suc-shift* *sum.atLeast-Suc-atMost-Suc-shift*)
have 2: $(\sum k = 1..Suc\ i. nlength\ (nnth\ nells\ k)) =$

```

      (∑ k = 0.. i. nlength (nnth nells (k+1)))
    using sum.shift-bounds-cl-nat-ivl[of λk . (nlength (nnth nells k)) 0 1 i]
    by (metis One-nat-def Suc-eq-plus1 )
  have 3: (∑ k = 0.. i. nlength (nnth nells (k+1))) = (∑ k = 0.. i. nlength (nnth x22 k))
    using NCons by auto
  have 4: all-nfinite x22
    by (simp add: NCons Suc.premis(1))
  have 5: i ≤ nlength x22
    using NCons Suc.premis(2) Suc-ile-eq by auto
  have 6: (∑ k = 0.. i. nlength (nnth x22 k)) < ∞
    using Suc.IH[of x22] 4 5 by blast
  have 7: nlength (nnth nells 0) < ∞
    by (metis Suc.premis(1) all-nfinite-nnth-b enat.distinct(2) enat-ord-simps(4)
        nfinite-nlength-enat zero-enat-def zero-le)
  have 8: nlength (nnth nells 0) + (∑ k = 0.. i. nlength (nnth x22 k)) < ∞
    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 (∑ k = 0..i. the-enat(nlength(nnth nells k))) = the-enat(∑ k = 0..i. (nlength(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: (∑ k = 0..Suc i. the-enat (nlength (nnth nells k))) =
        the-enat(nlength (nnth nells 0)) + (∑ k = 1..Suc i. the-enat (nlength (nnth nells k)))
        by (simp add: sum.atLeast-Suc-atMost)
      have 2: the-enat(nlength (nnth nells 0)) = the-enat(nlength x21)

```

```

using NCons by simp
have 3:  $(\sum k = 1..Suc\ i. the-enat\ (nlength\ (nnth\ nells\ k))) =$ 
 $(\sum k = 0..i. the-enat\ (nlength\ (nnth\ nells\ (k+1))))$ 
using sum.shift-bounds-cl-nat-ivl[of  $\lambda k. the-enat\ (nlength\ (nnth\ nells\ k))\ 0\ 1\ i$ ]
by (metis One-nat-def Suc-eq-plus1)
have 4:  $(\sum k = 0..i. the-enat\ (nlength\ (nnth\ nells\ (k+1)))) =$ 
 $(\sum k = 0..i. the-enat\ (nlength\ (nnth\ x22\ k)))$ 
using NCons by auto
have 5: all-nfinite x22
by (simp add: NCons Suc.prems(1))
have 6:  $i \leq nlength\ x22$ 
using NCons Suc.prems(2) Suc-ile-eq by auto
have 7:  $(\sum k = 0..i. the-enat\ (nlength\ (nnth\ x22\ k))) =$ 
 $the-enat(\sum k = 0..i. (nlength\ (nnth\ x22\ k)))$ 
using Suc.IH[of x22] 5 6 by blast
have 8:  $the-enat(\sum k = 0..i. (nlength\ (nnth\ x22\ k))) =$ 
 $the-enat(\sum k = 0..i. (nlength\ (nnth\ nells\ (k+1))))$ 
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.prems(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  $i < \text{nlength nells}$ 
    all-gr-zero nells
    all-nfinite nells
     $a < \infty$ 
  shows  $\text{nnth (lsum nells } a) i < \text{nnth (lsum nells } a) (\text{Suc } i)$ 
proof -
  have 1:  $\text{nnth (lsum nells } a) i = a + (\sum k::\text{nat} = 0..i. (\text{the-enat (nlength (nnth nells } k))))$ 
    using assms less-imp-le-nat lsum-nnth by (metis order-less-imp-le )
  have 2:  $\text{nnth (lsum nells } a) (\text{Suc } i) = a + (\sum k::\text{nat} = 0..(\text{Suc } i). (\text{the-enat (nlength (nnth nells } k))))$ 
    using assms Suc-ile-eq lsum-nnth by blast
  have 3:  $(\sum k::\text{nat} = 0..(\text{Suc } i). (\text{the-enat (nlength (nnth nells } k)))) =$ 
     $(\sum k::\text{nat} = 0..i. (\text{the-enat (nlength (nnth nells } k)))) + (\text{the-enat (nlength (nnth nells (Suc } i))))$ 
    using sum.atLeast0-atMost-Suc by blast
  have 4:  $\text{nlength (nnth nells (Suc } i)) > 0$ 
    using assms by (metis eSuc-enat ileI1 in-nset-conv-nnth)
  have 5:  $(\sum k::\text{nat} = 0..i. (\text{the-enat (nlength (nnth nells } k)))) < \infty$ 
    using lsum-nnth-nfinite[of i nells] assms order.order-iff-strict by blast
  have 6:  $\text{nlength (nnth nells (Suc } i)) < \infty$ 
    using assms
    by (metis all-nfinite-nnth-b eSuc-enat enat.distinct(2) enat-ord-simps(4) ileI1 nfinite-nlength-enat)
  have 7:  $a + (\sum k::\text{nat} = 0..i. (\text{the-enat (nlength (nnth nells } k)))) <$ 
     $a + (\sum k::\text{nat} = 0..(\text{Suc } i). (\text{the-enat (nlength (nnth nells } k))))$ 
    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  $i < \text{nlength (addzero (lsum nells } 0))$ 
    all-gr-zero nells
    all-nfinite nells
  shows  $\text{nnth (addzero (lsum nells } 0)) i < \text{nnth (addzero (lsum nells } 0)) (\text{Suc } i)$ 
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: 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: 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: $\bigwedge j. j \leq nlength (nmap (nnth nell1) (lsum (NNil nell1) (0::nat))) \longrightarrow$
*nnth ((nmap (nnth (nfuse nell nell1)) (lsum (NNil nell1) (the-enat (nlength nell)))) j =
nnth ((nmap (nnth nell1) (lsum (NNil nell1) (0::nat)))) j*

by *(metis assms(1) assms(4) lsum-NNil ndropn-nfuse ndropn-nnth nellist.simps(14) plus-nat.add-0)*

have 5: *nmap (nnth (nfuse nell nell1)) (lsum (NNil nell1) (the-enat (nlength nell))) =
(nmap (nnth nell1) (lsum (NNil nell1) (0::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 *(pfilt (nfuse nell (nfusecat nells)) (lsum nells (the-enat (nlength nell)))) =
(pfilt (nfusecat nells) (lsum nells 0))*

proof –

have 1: *(pfilt (nfuse nell (nfusecat nells)) (lsum nells (the-enat (nlength nell)))) =
nmap (nnth (nfuse nell (nfusecat nells))) (lsum nells (the-enat (nlength nell)))*

unfolding *pfilt-nmap* **by** *simp*

have 2: *(pfilt (nfusecat nells) (lsum nells 0))) = nmap (nnth (nfusecat nells)) (lsum nells (0::nat))*

unfolding *pfilt-nmap* **by** *simp*

have 3: *nlength (nmap (nnth (nfuse nell (nfusecat nells))) (lsum nells (the-enat (nlength nell)))) =
nlength (nmap (nnth (nfusecat nells)) (lsum nells (0::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 $\text{nnth-nmap}[of\ j\ (\text{lsum } \text{nells } (\text{the-enat } (\text{nlength } \text{nell}))) (\text{nnth } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells})))]$
 $a\ 3\ \text{by}\ \text{auto}$
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 $\text{lsum-nnth}[of\ j\ \text{nells } (\text{the-enat } (\text{nlength } \text{nell}))]$
by $(\text{metis } a\ \text{assms}(3)\ \text{lsum-nlength } \text{nlength-nmap})$
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 $\text{nnth-nmap}[of\ j\ (\text{lsum } \text{nells } (0::\text{nat})) (\text{nnth } (\text{nfusecat } \text{nells}))]$ **using** $a\ \text{by}\ \text{auto}$
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 $(\text{metis } (\text{no-types}, \text{lifting})\ 6\ a\ \text{add-0 } \text{add-diff-cancel-left}'\ \text{assms}(3)\ \text{lsum-nlength}$
 $\text{lsum-nnth } \text{nlength-nmap})$
have 9: $\text{nlast } \text{nell} = \text{nfirst } (\text{nfusecat } \text{nells})$
by $(\text{metis } \text{assms}(1)\ \text{nfirst-nfusecat-nfirst } \text{nlastnfirst-LCons})$
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 $(\text{metis } a\ \text{lsum-nlength } \text{nlength-nmap})$
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 $\text{nfusecat-nlength-nfinite}[of\ \text{nells}]$
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 $\text{nfuse-nlength}[of\ \text{nell } \text{nfusecat } \text{nells}]$ **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 $(\text{metis } \text{bot-nat-0.extremum } \text{canonically-ordered-monoid-add-class.lessE}$
 $\text{enat-le-plus-same}(1)\ \text{sum.ub-add-nat})$
have 141: $j = \text{the-enat } (\text{nlength } \text{nells}) \implies$
 $(\sum k::\text{nat} = 0::\text{nat}..j. (\text{nlength } (\text{nnth } \text{nells } k))) \leq$

```

      (∑ i::nat = 0::nat..the-enat (nlength nells). nlength (nnth nells i))
    by blast
  have 142: (∑ k::nat = 0::nat..j. (nlength (nnth nells k))) ≤
    (∑ i::nat = 0::nat..the-enat (nlength nells). nlength (nnth nells i))
    using 14 141 101 True nfinite-nlength-enat by fastforce
  have 143: enat (the-enat (nlength nell)) = nlength nell
    by (simp add: assms(5) enat-the-enat nfinite-nlength-enat)
  have 144: enat (the-enat (∑ k = 0..j. nlength (nnth nells k))) =
    (∑ k = 0..j. nlength (nnth nells k))
    by (metis 11 142 True assms(1) assms(3) dual-order.refl enat-ord-code(4)
      enat-the-enat leD nfinite-nlength-enat nfusecat-nlength-nfinite
      nlastnfirst-LCons sum-finite)
  have 143: enat (the-enat (nlength nell) + (∑ k = 0..j. the-enat (nlength (nnth nells k)))) =
    nlength nell + (∑ k = 0..j. nlength (nnth nells k))
    using 13
    by (metis 143 144 plus-enat-simps(1))
  have 15: (the-enat (nlength nell) + (∑ k::nat = 0::nat..j. the-enat (nlength (nnth nells k)))) ≤
    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
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 $(\text{pfilt } (\text{nfusecat } (N\text{Cons } \text{nell } \text{nells})) (\text{addzero } (\text{lsum } (N\text{Cons } \text{nell } \text{nells}) 0))) =$
 $(N\text{Cons } (\text{nfirst } \text{nell}) (N\text{Cons } (\text{nlast } \text{nell}) (\text{pfilt } (\text{nfusecat } \text{nells}) (\text{lsum } \text{nells } 0))))$
proof –
have 1: $\text{nfusecat } (N\text{Cons } \text{nell } \text{nells}) = \text{nfuse } \text{nell } (\text{nfusecat } \text{nells})$
by *simp*
have 2: $(\text{pfilt } (\text{nfusecat } (N\text{Cons } \text{nell } \text{nells})) (\text{addzero } (\text{lsum } (N\text{Cons } \text{nell } \text{nells}) 0))) =$
 $(\text{pfilt } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells})) (\text{addzero } (\text{lsum } (N\text{Cons } \text{nell } \text{nells}) 0)))$
using 1 **by** *simp*
have 3: $\text{addzero } (\text{lsum } (N\text{Cons } \text{nell } \text{nells}) 0) =$
 $(N\text{Cons } 0 (N\text{Cons } (\text{the-enat } (\text{nlength } \text{nell})) (\text{lsum } \text{nells } (\text{the-enat } (\text{nlength } \text{nell}))))))$
using *lsum-addzero-NCons* **by** *auto*
have 4: $(\text{pfilt } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells})) (\text{addzero } (\text{lsum } (N\text{Cons } \text{nell } \text{nells}) 0))) =$
 $(\text{pfilt } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells})) (N\text{Cons } 0 (N\text{Cons } (\text{the-enat } (\text{nlength } \text{nell})) (\text{lsum } \text{nells } (\text{the-enat } (\text{nlength } \text{nell}))))))$
using 3 **by** *auto*
have 5: $(\text{pfilt } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells})) (N\text{Cons } 0 (N\text{Cons } (\text{the-enat } (\text{nlength } \text{nell})) (\text{lsum } \text{nells } (\text{the-enat } (\text{nlength } \text{nell})))))) =$
 $(N\text{Cons } (\text{nnth } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells})) 0)$
 $(N\text{Cons } (\text{nnth } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells})) (\text{the-enat } (\text{nlength } \text{nell}))$
 $(\text{pfilt } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells})) (\text{lsum } \text{nells } (\text{the-enat } (\text{nlength } \text{nell}))))))$
by *simp*
have 6: $(\text{nnth } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells})) 0) = (\text{nnth } \text{nell } 0)$
using *assms*
by *(metis nfirsr-nfusecat-nfirsr nfuse-nnth nlastnfirsr-LCons zero-enat-def zero-le)*
have 7: $(\text{nnth } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells})) (\text{the-enat } (\text{nlength } \text{nell}))) =$
 $(\text{nnth } \text{nell } (\text{the-enat } (\text{nlength } \text{nell})))$
using *assms*
by *(metis ndropn-nfirsr ndropn-nfuse nfirsr-nfusecat-nfirsr nlastnfirsr-LCons nnth-nlast)*
have 8: $\text{nidx } (\text{addzero } (\text{lsum } \text{nells } 0))$
using *assms lsum-addzero-nidx* **by** *blast*
have 9: $(\text{pfilt } (\text{nfuse } \text{nell } (\text{nfusecat } \text{nells})) (\text{lsum } \text{nells } (\text{the-enat } (\text{nlength } \text{nell})))) =$
 $(\text{pfilt } (\text{nfusecat } \text{nells}) (\text{lsum } \text{nells } 0))$
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 *nlastnfirsr* $(N\text{Cons } \text{nell } \text{nells})$

all-gr-zero nells

all-nfinite nells

nlength nell > 0

nfinite nell

shows $(\text{pfilt } (\text{nfusecat } (N\text{Cons } \text{nell } \text{nells})) ((\text{lsum } (N\text{Cons } \text{nell } \text{nells}) 0))) =$
 $(N\text{Cons } (\text{nlast } \text{nell}) (\text{pfilt } (\text{nfusecat } \text{nells}) (\text{lsum } \text{nells } 0)))$

using *assms*

pfilt-nfusecat-lsum[of nell nells]

by *(metis lsum-addzero-NCons nellist.inject(2) pfilt-code(2))*

lemma *pfilt-nfusecat-lsum-2:*

assumes *nlastnfirst* (*nells*)

all-gr-zero *nells*

all-nfinite *nells*

$j \leq \text{nlength } nells$

shows $(\text{nnth } (\text{pfilt } (\text{nfusecat } (nells)) ((\text{lsum } (nells) 0))) j = \text{nlast}(\text{nnth } nells j)$

proof –

have 1: $(\text{pfilt } (\text{nfusecat } (nells)) ((\text{lsum } (nells) 0))) =$

$\text{nmap } (\text{nnth } (\text{nfusecat } nells)) (\text{lsum } nells (0::\text{nat}))$

using *pfilt-nmap*[of $(\text{nfusecat } (nells)) (\text{lsum } (nells) 0)$] **by** *simp*

have 2: $(\text{nnth } (\text{pfilt } (\text{nfusecat } (nells)) ((\text{lsum } (nells) 0))) j =$

$\text{nnth } (\text{nfusecat } nells) (\text{nnth } (\text{lsum } nells (0::\text{nat})) j)$

using 1 *nnth-nmap*[of $j ((\text{lsum } (nells) 0)) (\text{nnth } (\text{nfusecat } nells))$]]

by (*simp add: assms(4) lsum-nlength*)

have 3: $(\text{nnth } (\text{lsum } nells (0::\text{nat})) j) = (\sum k::\text{nat} = 0::\text{nat}..j. \text{the-enat } (\text{nlength } (\text{nnth } nells k))))$

by (*metis add-0 assms(3) assms(4) lsum-nnth*)

have 4: $(\sum k::\text{nat} = 0::\text{nat}..j. \text{the-enat } (\text{nlength } (\text{nnth } nells k)))) =$

$\text{the-enat } ((\sum k::\text{nat} = 0::\text{nat}..j. (\text{nlength } (\text{nnth } nells k))))$

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: $\text{nnth } (\text{nfusecat } nells) (\text{the-enat } ((\sum k::\text{nat} = 0::\text{nat}..j. (\text{nlength } (\text{nnth } nells k)))))) =$
 $\text{nlast}(\text{nnth } nells j)$

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*

$j \leq \text{nlength } (\text{addzero } (\text{lsum } nells 0))$

shows $(j=0 \longrightarrow (\text{nnth } (\text{pfilt } (\text{nfusecat } (nells)) (\text{addzero}(\text{lsum } (nells) 0))) j = \text{nfirst}(\text{nfirst } nells))$

\wedge

$(j>0 \longrightarrow (\text{nnth } (\text{pfilt } (\text{nfusecat } (nells)) (\text{addzero}(\text{lsum } (nells) 0))) j = \text{nlast}(\text{nnth } nells (j-1)))$

using *assms*

proof –

have 1: $j \leq \text{nlength } (\text{addzero } (\text{lsum } nells 0)) \wedge j=0 \longrightarrow$

$(\text{nnth } (\text{pfilt } (\text{nfusecat } (nells)) (\text{addzero}(\text{lsum } (nells) 0))) j = \text{nfirst}(\text{nfirst } nells)$

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:  $j \leq \text{nlength} (\text{addzero} (\text{lsum} \text{ nells } 0)) \wedge j > 0 \longrightarrow$ 
  ( $\text{nnth} (\text{pfilt} (\text{nfusecat} \text{ nells}) (\text{addzero} (\text{lsum} (\text{nells} \ 0)))) \ j =$ 
  ( $\text{nnth} (\text{nfusecat} \text{ nells}) (\text{nnth} (\text{addzero} (\text{lsum} (\text{nells} \ 0))) \ j)$ )
  by (simp add: pfilt-nlength pfilt-nnth)
have 3:  $j \leq \text{nlength} (\text{addzero} (\text{lsum} \text{ nells } 0)) \wedge j > 0 \longrightarrow$ 
   $\text{nnth} (\text{addzero} (\text{lsum} (\text{nells} \ 0))) \ j = \text{nnth} (\text{lsum} \text{ nells } 0) \ (j-1)$ 
  by (metis Suc-diff-1 addzero-def enat-0-iff(1) le-zero-eq nat-less-le nnth-Suc-NCons)
have 20:  $j \leq \text{nlength} (\text{addzero} (\text{lsum} \text{ nells } 0)) \wedge j > 0 \longrightarrow \text{enat} ((j::\text{nat}) - (1::\text{nat})) \leq \text{nlength} \text{ nells}$ 
  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:  $j \leq \text{nlength} (\text{addzero} (\text{lsum} \text{ nells } 0)) \wedge j > 0 \longrightarrow$ 
  ( $\text{nnth} (\text{nfusecat} \text{ nells}) (\text{nnth} (\text{lsum} \text{ nells } 0) \ (j-1))) = \text{nlast} (\text{nnth} \text{ nells} \ (j-1))$ 
  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
 $1 \leq i$
 $i \leq \text{nlength } nells$
 $j \leq \text{nlength}(\text{nnth } nells \ i)$
nfinite nells
all-gr-zero nells
shows $(\text{nnth } ((\text{lsum } nells \ 0)) \ (i-1) + j \leq \text{nlength } (\text{nfusecat } nells))$
proof –
have 0: $i-1 \leq \text{nlength } nells$
by (*metis Suc-ile-eq assms(3) assms(4) le-add-diff-inverse order-less-imp-le plus-1-eq-Suc*)
have 1: $(\text{nnth } ((\text{lsum } nells \ 0)) \ (i-1) = (\sum k::\text{nat} = 0..(i-1). \text{the-enat}(\text{nlength}(\text{nnth } nells \ k))))$
using 0 *lsum-nnth[of i-1 nells 0] assms by presburger*
have 2: $\text{nlength } (\text{nfusecat } nells) = (\sum k::\text{nat} = 0..(\text{the-enat}(\text{nlength } nells)). \text{nlength}(\text{nnth } nells \ k))$
using *assms nfusecat-nlength-nfinite by metis*
have 20: $(\sum k::\text{nat} = 0..(i). \text{the-enat}(\text{nlength}(\text{nnth } nells \ k))) =$
 $(\sum k = 0..i-1. \text{the-enat } (\text{nlength } (\text{nnth } nells \ k)))) +$
 $\text{the-enat } (\text{nlength } (\text{nnth } nells \ (\text{Suc } (i-1))))$
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*
 $(\text{metis } (\text{no-types, lifting}) \ \text{all-nfinite-nnth-b enat-ord-simps(1) enat-the-enat infinity-ileE}$
 $\text{ndropn-eq-NNil ndropn-nlast})$
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 $\lambda k. \text{the-enat}(\text{nlength}(\text{nnth } nells \ k)) (\text{the-enat}(\text{nlength } nells)) - i$] by fastforce*
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 (the-enat(nlength nells))] assms*
by (*metis 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
 $1 \leq i$
 $i \leq \text{nlength } nells$
 $j \leq \text{nlength}(\text{nnth } nells \ i)$

$\neg nfinite\ nells$
 $all-gr-zero\ nells$
shows $nnth\ ((lsum\ nells\ 0))\ (i-1) + j \leq nlength\ (nfusecat\ nells)$
proof –
have 1: $\neg nfinite\ (nfusecat\ nells)$
using *assms* **by** (*metis* *nfusecat-nfinite*)
show *?thesis*
by (*meson* 1 *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

$(enat\ i) \leq nlength\ (addzero\ (lsum\ nells\ 0))$

nfinite nells

shows $(nnth\ (addzero\ (lsum\ nells\ 0))\ i) \leq nlength\ (nfusecat\ nells)$

using *assms*

proof –

have 2: $nlength\ (nfusecat\ nells) = (\sum i = 0..the-enat\ (nlength\ nells). nlength\ (nnth\ nells\ i))$

using *assms* *nfusecat-nlength-nfinite* **by** *blast*

have 3: $nlength\ nells > 0 \implies nlength\ (lsum\ nells\ 0) > 0$

by (*simp* *add: lsum-nlength*)

have 4: $nlength\ nells > 0 \implies (nnth\ (addzero\ (lsum\ nells\ 0))\ i) = nnth\ (NCons\ 0\ (lsum\ nells\ 0))\ i$

using *assms* **unfolding** *addzero-def* **using** 3 **by** *simp*

have 5: $nlength\ nells > 0 \wedge i=0 \implies (nnth\ (addzero\ (lsum\ nells\ 0))\ i) = 0$

using 4 **by** *auto*

have 6: $nlength\ nells > 0 \wedge i>0 \implies (nnth\ (addzero\ (lsum\ nells\ 0))\ i) = nnth\ (lsum\ nells\ 0)\ (i-1)$

by (*metis* 4 *Suc-diff-1 nnth-Suc-NCons*)

have 7: $nlength\ nells > 0 \wedge i>0 \implies nnth\ (lsum\ nells\ 0)\ (i-1) =$

$(\sum k = 0..(i-1). the-enat\ (nlength\ (nnth\ nells\ k)))$

using *assms* *lsum-nnth[of i-1 nells 0]*

by (*metis* *Suc-diff-1 Suc-ile-eq add-0 addzero-def i0-less iless-Suc-eq lsum-nlength nlength-NCons*)

have 8: $nlength\ nells > 0 \wedge i>0 \implies$

$(\sum k = 0..(i-1). the-enat\ (nlength\ (nnth\ nells\ k))) =$

$the-enat(\sum k = 0..(i-1). (nlength\ (nnth\ nells\ k)))$

using *assms* *sum-the-enat[of nells i-1]*

by (*metis* *Suc-diff-1 Suc-ile-eq addzero-def i0-less iless-Suc-eq lsum-nlength nlength-NCons*)

have 9: $nlength\ nells > 0 \wedge i>0 \implies nnth\ (lsum\ nells\ 0)\ (i-1) \leq nlength\ (nfusecat\ nells)$

using *assms* *nfusecat-nlength-b[of nells i 0] pfilt-nfusecat-lsum-4[of nells]*

3 4 6 *lsum-nlength[of nells]* **unfolding** *addzero-def* **by** *simp*

(*metis* *iless-Suc-eq order.order-iff-strict the-enat.simps zero-enat-def zero-le*)

have 10: $nlength\ nells = 0 \wedge i = 0 \implies nnth\ (NCons\ 0\ (lsum\ nells\ 0))\ i \leq nlength\ (nfusecat\ nells)$

using *assms* *zero-enat-def* **by** *auto*

have 11: $nlength\ nells = 0 \wedge i = 1 \implies nnth\ (NCons\ 0\ (lsum\ nells\ 0))\ i \leq nlength\ (nfusecat\ nells)$

using *assms*

by (*metis* *addzero-def lsum-nlength nlength-NCons not-one-le-zero one-eSuc one-enat-def*

pfilt-nfusecat-lsum-4 the-enat.simps)

have 12: $nfinite\ (nfirst\ nells)$

by (metis assms(3) assms(5) nconcat-expand nfinite-nappend nfinite-nconcat)
 have 13: nlength (nfirst nells) > 0
 using assms
 by (metis in-nset-conv-nnth nlast-NNil ntaken-0 ntaken-nlast zero-enat-def zero-le)
 have 14: nlength nells = 0 \implies $i \leq 1$
 using assms lsum-addzero-nlength[of nells]
 by (metis 12 13 enat-ord-simps(1) one-enat-def)
 have 15: nnth (NCons 0 (lsum nells 0)) $i \leq$ nlength (nfusecat nells)
 by (metis 10 11 14 4 5 6 9 One-nat-def Suc-leI dual-order.antisym not-gr-zero
 zero-enat-def zero-le)
 show ?thesis
 by (metis 12 13 15 assms(4) gr-zeroI lsum-addzero-nnth)
 qed

lemma pfilt-nfusecat-lsum-5-alt:

assumes nlastnfirst (nells)
 all-gr-zero nells
 all-nfinite nells
 (enat i) \leq nlength (addzero (lsum nells 0))
 \neg nfinite nells
 shows (nnth (addzero (lsum nells 0)) i) \leq 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 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:  $(\sum k::nat = 0::nat..i. the-enat (nlength (nnth nells k))) =$ 
     $the-enat (nlength (nnth nells 0)) +$ 
     $(\sum k::nat = 1::nat..i. the-enat (nlength (nnth nells k)))$ 
    by (simp add: sum.atLeast-Suc-atMost)
  have 13:  $(nlength (nnth nells 0)) = nlength x21$ 
    by (simp add: NCons)
  have 14:  $nlength x21 \leq$ 
     $the-enat (nlength x21) +$ 
     $(\sum k::nat = 1::nat..i. the-enat (nlength (nnth nells k))) + j$ 
    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.premis(1) nfirst-nfusecat-nfirst nlastnfirst-LCons)
have 16:  $enat (nnth (addzero (lsum nells 0)) (Suc i) + j) \leq nlength (nfuse x21 (nfusecat x22))$ 
  by (metis 8 9 NCons Suc.premis(1) Suc.premis(2) Suc.premis(3) Suc.premis(4) Suc.premis(5)
    add-diff-cancel-left' le-add1 nfusecat-NCons nfusecat-nlength-b nfusecat-nlength-b-alt
    plus-1-eq-Suc)
have 17:  $nlength x21 \leq enat (nnth (addzero (lsum nells (0::nat))) (Suc i) + j)$ 
  using 10 11 8 9 by presburger
have 18:  $nnth (nfuse x21 (nfusecat x22)) (nnth (addzero (lsum nells (0::nat))) (Suc i) + j) =$ 
   $nnth (nfusecat x22) (nnth (addzero (lsum nells (0::nat))) (Suc i) + j - (the-enat(nlength x21)))$ 
  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:  $i=0 \implies nnth (addzero (lsum nells (0::nat))) (Suc i) = (the-enat(nlength x21))$ 
  using 8 NCons by auto
have 20:  $i=0 \implies nnth (nfusecat x22) 0 = nnth (nnth x22 i) 0$ 
  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 nnth-0 ntaken-0 ntaken-nlast order-less-imp-le zero-enat-def)
have 21:  $i=0 \implies ?thesis$ 
  by (metis 1 18 19 2 3 4 5 NCons Suc.IH Suc.premis(5) 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:  $0 < i \implies (\sum k::nat = 0::nat..i. the-enat (nlength (nnth nells k))) =$ 
   $(the-enat(nlength x21)) + (\sum k::nat = 0::nat..(i-1). the-enat (nlength (nnth x22 k)))$ 
proof -
  assume a1:  $0 < i$ 
  have 23:  $(\sum k::nat = 0::nat..i. the-enat (nlength (nnth nells k))) =$ 
     $the-enat(nlength x21) + (\sum k::nat = 1::nat..i. the-enat (nlength (nnth nells k)))$ 
    by (simp add: NCons sum.atLeast-Suc-atMost)
  have 24:  $(\sum k::nat = 1::nat..i. the-enat (nlength (nnth nells k))) =$ 
     $(\sum k::nat = 0::nat..(i-1). the-enat (nlength (nnth nells (k+1))))$ 
    using sum.shift-bounds-cl-nat-ivl[of  $\lambda k. the-enat (nlength (nnth nells k)) 0 1 i-1$ ]
    by (simp add: a1)
  have 25:  $(\sum k::nat = 0::nat..(i-1). the-enat (nlength (nnth nells (k+1)))) =$ 

```



```

      (∑ 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 gr0I infinity-ileE less-numeral-extra(3)
      ndropn-eq-NNil ndropn-nlast nellist.set-intros(1) nlast-NNil zero-enat-def)
  next
  case (NCons x21 x22)
  then show ?thesis by simp
    (metis 2 3 gr0I 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 ⇒ (∑ 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 ⇒
      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 ⇒ ?thesis
  using 18 28 NCons by fastforce
show ?thesis
  using 21 29 by blast
qed
qed
qed

```

lemma *lcpl-lsum-less-th-equal*:

```

  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
    i < nlength (addzero (lsum (lcpl f g σ ls) 0))

```

shows $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)) \ (\text{Suc } i)) \leq$
 $\text{nlength } (\text{nfusecat } (\text{lcpl } f \ g \ \sigma \ ls))$
proof –
have 1: $\text{nlastnfirst } (\text{lcpl } f \ g \ \sigma \ ls)$
using *assms*
by $(\text{metis } \text{enat.distinct}(2) \ \text{enat-the-enat } i0\text{-less } \text{lcpl-nfusecat-nlastnfirst}$
 $\text{nfinite-conv-nlength-enat } \text{nnth-nlast } \text{the-enat-0})$
have 2: $\text{all-gr-zero } (\text{lcpl } f \ g \ \sigma \ ls)$
using *assms* $\text{all-gr-zero-nnth-a } \text{lcpl-nlength-all-gr-zero}$ **by** *blast*
have 3: $\text{all-nfinite } (\text{lcpl } f \ g \ \sigma \ ls)$
using *assms* $\text{all-nfinite-nnth-a } \text{lcpl-nlength-all-nfinite}$ **by** *blast*
have 4: $\text{enat } (\text{Suc } (i::\text{nat})) \leq \text{nlength } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ (0::\text{nat})))$
using *assms* **using** *Suc-ile-eq* **by** *blast*
have 5: $\text{nfinite } (\text{lcpl } f \ g \ \sigma \ ls)$
using *assms* **using** *lcpl-nfinite* **by** *blast*
show ?thesis **using**
 $\text{pfilt-nfusecat-lsum-5}[of \ (\text{lcpl } f \ g \ \sigma \ ls) \ \text{Suc } i]$
using 1 2 3 4 5 **by** *blast*
qed

lemma *lcpl-lsum-less-th-equal-alt:*

assumes $\text{nidx } ls$
 $\text{nnth } ls \ 0 = 0$
 $\neg \text{nfinite } ls$
 $\neg \text{nfinite } \sigma$
 $(\forall \ i < \text{nlength } ls. (\text{nsubn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)) \) \models f \ \text{fproj } g)$
 $i < \text{nlength } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0))$
shows $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)) \ (\text{Suc } i)) \leq$
 $\text{nlength } (\text{nfusecat } (\text{lcpl } f \ g \ \sigma \ ls))$
proof –
have 1: $\text{nlastnfirst } (\text{lcpl } f \ g \ \sigma \ ls)$
using *assms*
using *lcpl-nfusecat-nlastnfirst-alt* **by** *blast*
have 2: $\text{all-gr-zero } (\text{lcpl } f \ g \ \sigma \ ls)$
using *assms* $\text{all-gr-zero-nnth-a } \text{lcpl-nlength-all-gr-zero-alt}$ **by** *blast*
have 3: $\text{all-nfinite } (\text{lcpl } f \ g \ \sigma \ ls)$
using *assms* $\text{all-nfinite-nnth-a } \text{lcpl-nlength-all-nfinite-alt}$ **by** *blast*
have 4: $\text{enat } (\text{Suc } (i::\text{nat})) \leq \text{nlength } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ (0::\text{nat})))$
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) *lcpl-nfinite* **by** *blast*
qed

lemma *lcpl-lsum-nlength:*

assumes $\text{nidx } ls$
 $\text{nnth } ls \ 0 = 0$
 $\text{nfinite } ls$
 $\text{nfinite } \sigma$
 $\text{nlast } ls = (\text{the-enat } (\text{nlength } \sigma))$

$(\forall i < \text{nlength } ls. (\text{nsubn } \sigma (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) \models f \text{ fproj } g)$
 $\text{nlength } \sigma > 0$
shows $\text{nlength } (\text{addzero } (\text{lsum } ((\text{lcpl } f \ g \ \sigma \ ls)) \ 0)) = \text{nlength } ls$
proof –
have 1: $\text{nlength } \sigma > 0 \longrightarrow \text{nlength } (\text{lcpl } f \ g \ \sigma \ ls) = \text{nlength } ls - 1$
using *assms*
by (*metis epred-0 epred-conv-minus i0-less lcpl-nlength lcpl-nlength-zero*)
have 2: $\text{nlength } \sigma > 0 \longrightarrow \text{nlength } ls > 0$
using *assms*
by (*metis enat.distinct(2) enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0*)
have 3: $\text{nlength } \sigma > 0 \longrightarrow \text{nlength } (\text{nfirst } (\text{lcpl } f \ g \ \sigma \ ls)) > 0$
using *assms*
by (*metis lcpl-nlength-all-gr-zero ndropn-0 ndropn-nfirst nfinite-nlength-enat the-enat.simps zero-enat-def zero-le*)
have 30: $\text{nlastnfirst } (\text{lcpl } f \ g \ \sigma \ ls)$
using 2 *assms* *lcpl-nfusecat-nlastnfirst* **by** *blast*
have 31: $\text{nfinite } (\text{nfirst } (\text{lcpl } f \ g \ \sigma \ ls))$
using *assms*
by (*metis all-nfinite-nnth-a lcpl-nfinite lcpl-nlength-all-nfinite nconcat-expand nfinite-nappend nfinite-nconcat*)
have 4: $\text{nlength } \sigma > 0 \longrightarrow \text{nlength } (\text{lcpl } f \ g \ \sigma \ ls) = 0 \longrightarrow$
 $\text{nlength } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)) = \text{nlength } ls$
using 1 2 3 *assms* *lsum-addzero-nlength[of (lcpl f g σ ls)] 30 31*
by (*metis co.enat.exhaust-sel i0-less lcpl-nlength one-eSuc*)
have 5: $\text{nlength } \sigma > 0 \longrightarrow \text{nlength } (\text{lcpl } f \ g \ \sigma \ ls) > 0 \longrightarrow$
 $\text{nlength } (\text{addzero } (\text{lsum } ((\text{lcpl } f \ g \ \sigma \ ls)) \ 0)) = \text{nlength } ls$
using *assms*
by (*metis 2 4 addzero-def co.enat.exhaust-sel lcpl-nlength lcpl-nlength-zero lsum-nlength nlength-NCons*)
show ?thesis **using** 4 5
using *assms(7) gr-zeroI* **by** *blast*
qed

lemma *lcpl-lsum-nlength-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 $\text{nlength } (\text{addzero } (\text{lsum } ((\text{lcpl } f \ g \ \sigma \ ls)) \ 0)) = \text{nlength } ls$
proof –
have 1: $\text{nlength } (\text{lcpl } f \ g \ \sigma \ ls) = \text{nlength } ls - 1$
using *assms*
by (*simp add: epred-conv-minus lcpl-nlength-alt*)
have 2: $\text{nlength } ls > 0$
using *assms*
by (*simp add: nfinite-conv-nlength-enat*)
have 3: $\text{nlength } (\text{nfirst } (\text{lcpl } f \ g \ \sigma \ ls)) > 0$
using *assms* *lcpl-nlength-all-gr-zero-alt[of ls σ 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 $(j=0 \longrightarrow$
 $(nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0))))\ j =$
 $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 1: $nlastnfirst\ (lcppl\ f\ g\ \sigma\ ls)$
using *assms*
using *lcppl-nfusecat-nlastnfirst-alt* **by** *blast*
have 2: $(\forall\ j \leq nlength\ (lcppl\ f\ g\ \sigma\ ls). nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ j) > 0)$
using *assms lcppl-nlength-all-gr-zero-alt* **by** *blast*
have 3: $nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) = nlength\ (lcppl\ f\ g\ \sigma\ ls) + 1$
using *assms*
by (*simp add: lcppl-lsum-nlength-alt lcppl-nlength-alt nfinite-conv-nlength-enat*)
have 4: $(nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0))))\ 0 =$
 $nfirst(nfirst\ (lcppl\ f\ g\ \sigma\ ls))$
by (*metis lsum-addzero-nfirst nfirst-nfusecat-nfirst nlast-NNil ntaken-0 ntaken-nlast pfilt-nnth zero-enat-def zero-le*)
have 5: $j \leq nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) \wedge j > 0 \longrightarrow$
 $(nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls)))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0))))\ (j))$
 $= nlast(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ (j-1))$
using *assms pfilt-nfusecat-lsum-3[of (lcppl f g σ ls) j]*
using 1 2 *all-gr-zero-nnth-a all-nfinite-nnth-a lcppl-nlength-all-nfinite-alt* **by** *blast*
from 4 5 **show** *?thesis* **using** *assms*
by *blast*
qed

lemma *lcppl-lsum-nnth-a:*

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)$
 $nlength\ \sigma > 0$
 $j \leq nlength\ ls$
shows $(nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls))))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0)))\ j) =$
 $(nnth\ ls\ j)$
proof –
have 1: $nlength\ \sigma > 0 \longrightarrow nlength\ ls > 0$
using *assms*
by (*metis enat.distinct(2) enat-the-enat nfinite-nlength-enat nnth-nlast the-enat-0 zero-less-iff-neq-zero*)
have 2: $nlength\ \sigma > 0 \longrightarrow nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) = nlength\ ls$
by (*simp add: assms lcppl-lsum-nlength*)
have 3: $nlength\ \sigma > 0 \longrightarrow$
 $j \leq nlength\ ls \longrightarrow$
 $(j = 0 \longrightarrow$
 $(nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls))))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0)))\ j) =$
 $nfirst(nfirst\ (lcppl\ f\ g\ \sigma\ ls))\)$
 \wedge
 $(j > 0 \longrightarrow (nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls))))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0)))\ j) =$
 $nlast(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ (j-1)))$
by (*metis 2 assms lcppl-lsum-nnth*)
have 4: $nlength\ \sigma > 0 \longrightarrow$
 $j \leq nlength\ ls \wedge j = 0 \longrightarrow nfirst(nfirst\ (lcppl\ f\ g\ \sigma\ ls)) = (nnth\ ls\ j)$
using *assms lcppl-nfirst[of ls sigma f g]*
by (*metis 1 nlast-NNil ntaken-0 ntaken-nlast*)
have 5: $nlength\ \sigma > 0 \longrightarrow$
 $j \leq nlength\ ls \wedge j > 0 \longrightarrow j-1 \leq nlength\ (lcppl\ f\ g\ \sigma\ ls)$
using *assms*
by (*metis 1 Suc-diff-1 Suc-ile-eq co.enat.exhaust-sel i0-less iless-Suc-eq lcppl-nlength*)
have 6: $nlength\ \sigma > 0 \longrightarrow$
 $j \leq nlength\ ls \wedge j > 0 \longrightarrow nlast(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ (j-1)) = (nnth\ ls\ j)$
using *lcppl-nlast-nnth assms*
by (*metis 5 Suc-diff-1 enat.distinct(2) enat-the-enat nfinite-conv-nlength-enat*)
show ?thesis
using 3 4 6 **using** *assms*
by (*metis not-gr-zero*)
qed

lemma *lcppl-lsum-nnth-a-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)$
 $j \leq nlength\ ls$
shows $(nnth\ (pfilt\ (nfusecat\ ((lcppl\ f\ g\ \sigma\ ls))))\ (addzero(lsum\ ((lcppl\ f\ g\ \sigma\ ls))\ 0)))\ j) =$
 $(nnth\ ls\ j)$

proof –

have 2: $nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) = nlength\ ls$

by (simp add: assms lcppl-lsum-nlength-alt)
 have 3: $j \leq \text{nlength } ls \longrightarrow$
 ($j=0 \longrightarrow$
 $(\text{nnth } (\text{pfilt } (\text{nfusecat } ((\text{lcppl } f \ g \ \sigma \ ls)))) \ (\text{addzero}(\text{lsum } ((\text{lcppl } f \ g \ \sigma \ ls)) \ 0))) \ j) =$
 $\text{nfirst}(\text{nfirst } (\text{lcppl } f \ g \ \sigma \ ls)) \)$
 \wedge
 ($j>0 \longrightarrow (\text{nnth } (\text{pfilt } (\text{nfusecat } ((\text{lcppl } f \ g \ \sigma \ ls)))) \ (\text{addzero}(\text{lsum } ((\text{lcppl } f \ g \ \sigma \ ls)) \ 0))) \ j) =$
 $\text{nlast}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ (j-1))$)
 by (metis 2 assms lcppl-lsum-nnth-alt)
 have 4: $j \leq \text{nlength } ls \wedge j=0 \longrightarrow \text{nfirst}(\text{nfirst } (\text{lcppl } f \ g \ \sigma \ ls)) = (\text{nnth } ls \ j)$
 using assms lcppl-nfirst-alt[of $ls \ \sigma \ f \ g$] by (metis ndropn-0 ndropn-nfirst)
 have 5: $j \leq \text{nlength } ls \wedge j>0 \longrightarrow j-1 \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls)$
 using assms
 by (meson enat-ile lcppl-nfinite linorder-le-cases nfinite-conv-nlength-enat)
 have 6: $j \leq \text{nlength } ls \wedge j>0 \longrightarrow \text{nlast}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ (j-1)) = (\text{nnth } ls \ j)$
 using lcppl-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 $\text{nidx } ls$
 $\text{nnth } ls \ 0 = 0$
 $\text{nfinite } ls$
 $\text{nfinite } \sigma$
 $\text{nlast } ls = \text{the-enat } (\text{nlength } \sigma)$
 $(\forall \ i < \text{nlength } ls. (\text{nsubn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)) \) \models f \ \text{fproj } \ g)$
 $\text{nlength } \sigma > 0$
 shows $(\text{pfilt } (\text{nfusecat } ((\text{lcppl } f \ g \ \sigma \ ls)))) \ (\text{addzero}(\text{lsum } ((\text{lcppl } f \ g \ \sigma \ ls)) \ 0))) = ls$
 using assms
 proof –
 have 0: $\text{nlength } \sigma > 0 \longrightarrow \text{nlength } ls > 0$
 using assms
 by (metis enat.distinct(2) enat-the-enat gr-zeroI nfinite-conv-nlength-enat nnth-nlast the-enat-0)
 have 1: $\text{nlength } \sigma > 0 \longrightarrow$
 $\text{nlength } (\text{pfilt } (\text{nfusecat } ((\text{lcppl } f \ g \ \sigma \ ls)))) \ (\text{addzero}(\text{lsum } ((\text{lcppl } f \ g \ \sigma \ ls)) \ 0))) = \text{nlength } ls$
 using assms
 by (simp add: lcppl-lsum-nlength pfilt-nlength)
 have 2: $\text{nlength } \sigma > 0 \longrightarrow$
 $(\forall \ j. j \leq \text{nlength } ls \longrightarrow$
 $(\text{nnth } (\text{pfilt } (\text{nfusecat } ((\text{lcppl } f \ g \ \sigma \ ls)))) \ (\text{addzero}(\text{lsum } ((\text{lcppl } f \ g \ \sigma \ ls)) \ 0))) \ j) =$
 $(\text{nnth } ls \ j))$
 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 $nidx\ ls$
 $nnth\ ls\ 0 = 0$
 $nfinite\ ls$
 $nfinite\ \sigma$
 $nlast\ ls = the-enat\ (nlength\ \sigma)$
 $(\forall\ i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$
 $nlength\ \sigma > 0$
shows $powerinterval\ g\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))$
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: $(\forall\ i < nlength\ ls.$
 $g\ (pfilt\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))$
using $assms\ cppl-fprojection$ **by** $auto$
have 02: $nlength\ \sigma > 0 \longrightarrow (\forall\ i < nlength\ ls. g\ (pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)))$
using 0 $assms\ lcppl-nfusecat-pfilt-fpower-help$
by $(metis\ enat.distinct(2)\ enat-the-enat\ nfinite-conv-nlength-enat)$
have 03 : $powerinterval\ g\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) =$
 $(\forall\ i < nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)).$
 $g\ (nsubn\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))$
 $(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ i)$
 $(nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ (Suc\ i))$
 $))$
by $(simp\ add: powerinterval-def)$
have 04: $nlength\ \sigma > 0 \longrightarrow nlength\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) = nlength\ ls$
using $assms\ lcppl-lsum-nlength$ **by** $blast$
have 05: $nlength\ \sigma > 0 \longrightarrow$
 $(\forall\ i < nlength\ ls.$
 $(pfilt\ \sigma\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ i)) =$
 $(nsubn\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls))))$

$(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0)) \ i)$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0)) \ (\text{Suc } i))$
 $)$

proof –

have 06: $\text{nlength } \sigma > 0 \longrightarrow$

$(\forall \ i < \text{nlength } \text{ls}. \text{nlength } (\text{pfilt } \sigma \ (\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ i)) =$
 $\text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ i))$

using *pfilt-nlength* **by** *blast*

have 07: $\text{nlength } \sigma > 0 \longrightarrow$

$(\forall \ i < \text{nlength } \text{ls}. \text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ i) =$
 $\text{nlength } (\text{nmap } (\lambda x. x + (\text{nnth } \text{ls } i)) \ (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)) \)))$

using *assms* **by** (*simp add: lcpl-nnth*)

have 08: $\text{nlength } \sigma > 0 \longrightarrow$

$(\forall \ i < \text{nlength } \text{ls}.$
 $\text{nlength } (\text{nmap } (\lambda x. x + (\text{nnth } \text{ls } i)) \ (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)) \))) =$
 $\text{nlength } (\text{cppl } f \ g \ (\text{nsubn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)) \)))$

using *nlength-nmap* **by** *blast*

have 09: $\text{nlength } \sigma > 0 \longrightarrow$

$(\forall \ i < \text{nlength } \text{ls}.$
 $\text{nlength } (\text{nsubn } (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcpl } f \ g \ \sigma \ \text{ls})))$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0)) \ i)$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0)) \ (\text{Suc } i))$
 $) =$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0)) \ (\text{Suc } i)) -$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0)) \ i)$

using *assms*

by (*metis 04 enat-minus-mono1 idiff-enat-enat lcpl-lsum-less-th-equal min.orderE*
nsubn-nlength pfilt-nlength)

have 10: $\text{nlength } \sigma > 0 \longrightarrow$

$(\forall \ i < \text{nlength } \text{ls}. (\text{nnth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0)) \ (\text{Suc } i)) =$
 $(\text{nnth } (\text{NCons } 0 \ (\text{lsum } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0)) \ (\text{Suc } i)))$

using 04 *addzero-def* **by** *auto*

have 11: $\text{nlength } \sigma > 0 \longrightarrow$

$(\forall \ i < \text{nlength } \text{ls}. (\text{nnth } (\text{NCons } 0 \ (\text{lsum } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ 0)) \ (\text{Suc } i)) =$
 $(\sum k::\text{nat} = 0..i). \text{nlength}(\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ k))$

using 04

proof *simp-all*

assume $\text{nlength } \sigma \neq (0::\text{enat}) \longrightarrow \text{nlength } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ (0::\text{nat}))) = \text{nlength } \text{ls}$

show $\text{nlength } \sigma \neq (0::\text{enat}) \longrightarrow$

$(\forall \ i::\text{nat}. \text{enat } i < \text{nlength } \text{ls} \longrightarrow \text{enat } (\text{nnth } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ (0::\text{nat})) \ i) =$
 $(\sum k::\text{nat} = 0::\text{nat}..i. \text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ k)))$

proof

assume $\text{nlength } \sigma \neq (0::\text{enat})$

show $(\forall \ i::\text{nat}. \text{enat } i < \text{nlength } \text{ls} \longrightarrow$

$\text{enat } (\text{nnth } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ (0::\text{nat})) \ i) =$
 $(\sum k::\text{nat} = 0::\text{nat}..i. \text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ k)))$

```

proof
  fix  $i$ 
  show  $i < \text{nlength } ls \longrightarrow$ 
     $\text{enat } (\text{nnth } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ (0::\text{nat})) \ i) =$ 
     $(\sum k::\text{nat} = 0::\text{nat}..i. \text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k))$ 
  proof
    assume  $a: i < \text{nlength } ls$ 
    show  $\text{enat } (\text{nnth } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0) \ i) =$ 
       $(\sum k = 0..i. \text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k))$ 
    proof –
      have 110:  $\text{all-nfinite } (\text{lcpl } f \ g \ \sigma \ ls)$ 
        using  $\text{all-nfinite-nnth-a assms lcpl-nlength-all-nfinite}$  by  $\text{blast}$ 
      have 111:  $\text{nnth } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0) \ i =$ 
         $(\sum k = 0..i. \text{the-enat } (\text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k)))$ 
      using  $\text{lsum-nnth[of } i \ (\text{lcpl } f \ g \ \sigma \ ls) \ 0] \ a \ \text{assms}$ 
      by  $(\text{metis } 0 \ 110 \ \text{add-cancel-right-left co.enat.exhaust-sel iless-Suc-eq}$ 
         $\text{lcpl-nlength not-gr-zero})$ 
      have 112:  $(\sum k = 0..i. \text{the-enat } (\text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k))) =$ 
         $\text{the-enat}(\sum k = 0..i. (\text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k)))$ 
        using  $\text{sum-the-enat[of } (\text{lcpl } f \ g \ \sigma \ ls) \ i] \ \text{assms}$ 
        by  $(\text{metis } 0 \ 110 \ a \ \text{co.enat.exhaust-sel gr-implies-not-zero iless-Suc-eq}$ 
           $\text{lcpl-nlength})$ 
      have 113:  $(\sum k = 0..i. (\text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k))) < \infty$ 
        using  $\text{sum-finite[of } (\text{lcpl } f \ g \ \sigma \ ls) \ i] \ 0 \ 110 \ \text{assms } a$ 
        by  $(\text{metis co.enat.exhaust-sel gr-implies-not-zero iless-Suc-eq lcpl-nlength})$ 
      have 114:  $\text{enat } (\text{the-enat}(\sum k = 0..i. (\text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k)))) =$ 
         $(\sum k = 0..i. (\text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k)))$ 
        using 113  $\text{enat-ord-simps(4) enat-the-enat}$  by  $\text{blast}$ 
      show  $?thesis$  using 111 112 114 by  $\text{presburger}$ 
    qed
  qed
qed
qed
qed
have 12:  $\text{nlength } \sigma > 0 \longrightarrow$ 
   $(\forall i < \text{nlength } ls. (\text{nnth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)) \ i) =$ 
   $(\text{nnth } (\text{NCons } 0 \ (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)) \ i) )$ 
  using 04  $\text{addzero-def}$  by  $\text{auto}$ 
have 121:  $\text{nlength } \sigma > 0 \longrightarrow (\text{nnth } (\text{NCons } 0 \ (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)) \ 0) = 0$ 
  by  $\text{simp}$ 
have 13:  $\text{nlength } \sigma > 0 \longrightarrow$ 
   $(\forall i < \text{nlength } ls. (\text{nnth } (\text{NCons } 0 \ (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)) \ i) =$ 
   $(\text{if } i = 0 \text{ then } 0$ 
   $\text{else } (\sum k::\text{nat} = 0..(i-1). \text{nlength}(\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k)) )$ 
  using 11
  by  $(\text{metis } 121 \ \text{Suc-diff-1} \ \text{Suc-ile-eq not-gr-zero order-less-imp-le zero-enat-def})$ 
have 14:  $\text{nlength } \sigma > 0 \longrightarrow$ 
   $(\forall i < \text{nlength } ls.$ 
   $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)) \ (\text{Suc } i)) -$ 

```

$(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i) =$
 $(\sum k::\text{nat} = 0..(i). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ k)) -$
 $(\text{if } i = 0 \text{ then } 0 \text{ else}$
 $(\sum k::\text{nat} = 0..(i-1). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ k)) \) \)$
using 10 11 12 13 **by** *simp (metis One-nat-def idiff-enat-enat not-gr-zero)*
have 15: $\text{nlength } \sigma > 0 \longrightarrow$
 $(\forall i < \text{nlength } ls.$
 $(\sum k::\text{nat} = 0..(i). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ k)) -$
 $(\text{if } i = 0 \text{ then } 0 \text{ else}$
 $(\sum k::\text{nat} = 0..(i-1). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ k)) \) =$
 $(\text{if } i = 0 \text{ then } \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ 0) \text{ else}$
 $(\sum k::\text{nat} = 0..(i). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ k)) -$
 $(\sum k::\text{nat} = 0..(i-1). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ k)) \) \)$
by (*simp add: Nitpick.case-nat-unfold*)
have 16: $\text{nlength } \sigma > 0 \longrightarrow$
 $(\forall i < \text{nlength } ls.$
 $(\text{if } i = 0 \text{ then } \text{nlength } (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ 0) \text{ else}$
 $(\sum k::\text{nat} = 0..(i). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ k)) -$
 $(\sum k::\text{nat} = 0..(i-1). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ k)) \) =$
 $(\text{if } i = 0 \text{ then } \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ 0) \text{ else}$
 $\text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i) \) \)$
using 13 *sum.cl-ivl-Suc[of $\lambda k. \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ k) \ 0]$*
by (*metis Suc-diff-1 enat.distinct(2) enat-add-sub-same less-nat-zero-code not-gr-zero*)
have 17: $\text{nlength } \sigma > 0 \longrightarrow$
 $(\forall i < \text{nlength } ls.$
 $(\text{if } i = 0 \text{ then } \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ 0) \text{ else}$
 $\text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i) \) =$
 $\text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i))$
by (*simp add: Nitpick.case-nat-unfold*)
have 18: $\text{nlength } \sigma > 0 \longrightarrow$
 $(\forall i < \text{nlength } ls. \text{nlength } (\text{pfilt } \sigma \ (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i)) =$
 $\text{nlength } (\text{nsbn } (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))))$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i)$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ (\text{Suc } i))$
 $) \)$
by (*simp add: 09 14 15 16 pfilt-nlength*)
have 19: $\text{nlength } \sigma > 0 \longrightarrow$
 $(\forall i < \text{nlength } ls.$
 $(\forall j \leq \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i).$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ (\text{Suc } i)) \leq$
 $\text{nlength } (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))))$
by (*simp add: 04 assms lcppl-lsum-less-th-equal pfilt-nlength*)
have 22: $\text{nlength } \sigma > 0 \longrightarrow \text{nlastnfirst } (\text{lcppl } f \ g \ \sigma \ ls)$
using 0 *assms*
by (*simp add: enat-the-enat lcppl-nfusecat-nlastnfirst nfinite-conv-nlength-enat*)
have 23: $\text{nlength } \sigma > 0 \longrightarrow (\forall j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ j) > 0)$
using *assms*
by (*metis enat.distinct(2) enat-the-enat lcppl-nlength-all-gr-zero nfinite-conv-nlength-enat*)
have 190: $\text{nlength } \sigma > 0 \longrightarrow$

```

    (∀ i < nlength ls.
      (∀ j ≤ nlength (nnth (lcppl f g σ ls) i).
        (nnth (addzero (lsum (lcppl f g σ ls) 0)) i) ≤
        (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i))
      ))
  using 0 04 06 09 18 23 lsum-nlength[of (lcppl f g σ ls) 0] lcppl-nlength[of ls f g σ] assms
  by simp
  (metis (no-types, lifting) co.enat.exhaust-sel diff-is-0-eq' iless-Suc-eq le-cases zero-enat-def)
have 20: nlength σ > 0 ⟶
  (∀ i < nlength ls.
    (∀ j ≤ nlength (nnth (lcppl f g σ ls) i).
      (nnth (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))
      ) j) =
      (nnth (pfilt σ (nfusecat (lcppl f g σ ls)))
        ((nnth (addzero (lsum (lcppl f g σ ls) 0)) i) + j) ) ) )
  using nsubn-nnth[of (pfilt σ (nfusecat (lcppl f g σ ls))) ] by simp
  (metis 06 09 18 assms(7) enat-ord-simps(1) min.orderE)
have 21: nlength σ > 0 ⟶
  (∀ i < nlength ls.
    (∀ j ≤ nlength (nnth (lcppl f g σ ls) i).
      (nnth (pfilt σ (nfusecat (lcppl f g σ ls)))
        ((nnth (addzero (lsum (lcppl f g σ ls) 0)) i) + j) ) =
      (nnth σ (nnth (nfusecat (lcppl f g σ ls))
        ((nnth (addzero (lsum (lcppl f g σ ls) 0)) i) + j) )) ) )
  using pfilt-nlength[of σ (nfusecat (lcppl f g σ ls)) ]
  pfilt-nnth[of - σ (nfusecat (lcppl f g σ ls)) ]
  by (metis nnth-0 pfilt-code(2) pfilt-pfilt)
have 24: nlength σ > 0 ⟶
  (∀ i ≤ nlength (lcppl f g σ ls).
    (∀ j ≤ nlength (nnth (lcppl f g σ ls) i).
      ( (nnth (nfusecat (lcppl f g σ ls)) ((nnth (addzero (lsum (lcppl f g σ ls) 0)) i) + j) )) =
      ( (nnth (nnth (lcppl f g σ ls) i) j) ) ) )

  using assms lsum-nfusecat-nnth-lsum-nnth[of (lcppl f g σ ls)] lcppl-nlength[of ls f g σ]
  0 06 09 18 22 23
  by (metis all-gr-zero-nnth-a all-nfinite-nnth-a lcppl-nlength-all-nfinite)
have 241: nlength σ > 0 ⟶ nlength (lcppl f g σ ls) = nlength ls - 1
  using 0 assms
  by (simp add: epred-conv-minus lcppl-nlength)
have 25: nlength σ > 0 ⟶
  (∀ i < nlength ls.
    (∀ j ≤ nlength (nnth (lcppl f g σ ls) i).
      (nnth σ (nnth (nfusecat (lcppl f g σ ls))
        ((nnth (addzero (lsum (lcppl f g σ ls) 0)) i) + j) )) =
      (nnth (pfilt σ (nnth (lcppl f g σ ls) i) j) ) )
  using 0 04 24 lsum-nlength[of (lcppl f g σ ls) 0] pfilt-nlength[of σ ] pfilt-nnth
  by (metis addzero-def i0-less iless-Suc-eq nlength-NCons)
have 26: nlength σ > 0 ⟶

```

$(\forall i < \text{nlength } ls.$
 $(\forall j \leq \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i).$
 $(\text{nnth } (\text{pfilt } \sigma \ (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i)) \ j) =$
 $(\text{nnth } (\text{nsbn } (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))))$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i)$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ (\text{Suc } i))$
 $) \ 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
 $\text{nnth } ls \ 0 = 0$
 $\neg \text{nfinite } ls$
 $\neg \text{nfinite } \sigma$
 $(\forall i < \text{nlength } ls. (\text{nsbn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)) \) \models f \ \text{fproj } g)$
 shows $\text{powerinterval } g \ (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))) \ (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0))$

proof –

have 01: $(\forall i < \text{nlength } ls.$
 $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* *cppl-fprojection* by auto
 have 02: $(\forall i < \text{nlength } ls. g \ (\text{pfilt } \sigma \ (\text{nnth } (\text{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 : $\text{powerinterval } g \ (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))) \ (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) =$
 $(\forall i < \text{nlength } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)).$
 $g \ (\text{nsbn } (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))))$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i)$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ (\text{Suc } i))$
 $)$
 by (simp add: *powerinterval-def*)
 have 04: $\text{nlength } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) = \text{nlength } ls$
 using *assms* *lcppl-lsum-nlength-alt* by blast
 have 05: $(\forall i < \text{nlength } ls.$
 $(\text{pfilt } \sigma \ (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i)) =$
 $(\text{nsbn } (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))))$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i)$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ (\text{Suc } i))$
 $)$

proof –

have 06:
 $(\forall i < \text{nlength } ls. \text{nlength } (\text{pfilt } \sigma \ (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i)) =$
 $\text{nlength } (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i))$
 using *pfilt-nlength* by blast

```

have 07: (∀ 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: (∀ 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 σ (nnth ls i) (nnth ls (Suc i)) )))
  using nlength-nmap by blast
have 09: (∀ i<nlength ls.
  nlength (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))
  ) =
  (nnth (addzero (lsum (lcppl f g σ ls) 0)) (Suc i)) -
  (nnth (addzero (lsum (lcppl f g σ 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( $\sum k = 0..i. \text{ nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k))) =$ 
    ( $\sum k = 0..i. \text{ nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k)))$ 
    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: ( $\forall i < \text{nlength } ls. (\text{nnth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)) \ i) =$ 
  ( $\text{nnth } (\text{NCons } 0 \ (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)) \ i)$ )
  using 04 addzero-def by auto
have 121: ( $\text{nnth } (\text{NCons } 0 \ (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)) \ 0) = 0$ 
  by simp
have 13: ( $\forall i < \text{nlength } ls. (\text{nnth } (\text{NCons } 0 \ (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)) \ i) =$ 
  (if  $i = 0$  then  $0$ 
    else ( $\sum k::\text{nat} = 0..(i-1). \text{ nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k))$ ))
  using 11
  by (metis 121 Suc-diff-1 Suc-ile-eq not-gr-zero order-less-imp-le zero-enat-def)
have 14: ( $\forall i < \text{nlength } ls. (\text{nnth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)) \ (\text{Suc } i)) -$ 
  ( $\text{nnth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)) \ i) =$ 
  ( $\sum k::\text{nat} = 0..(i). \text{ nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k)) -$ 
  (if  $i = 0$  then  $0$  else
    ( $\sum k::\text{nat} = 0..(i-1). \text{ nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k))$ ))
  using 10 11 12 13 by simp (metis One-nat-def idiff-enat-enat not-gr-zero)
have 15: ( $\forall i < \text{nlength } ls. (\sum k::\text{nat} = 0..(i). \text{ nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k)) -$ 
  (if  $i = 0$  then  $0$  else
    ( $\sum k::\text{nat} = 0..(i-1). \text{ nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k))$ ) =
  (if  $i = 0$  then  $\text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)$  else
    ( $\sum k::\text{nat} = 0..(i). \text{ nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k)) -$ 
    ( $\sum k::\text{nat} = 0..(i-1). \text{ nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k))$ ))
  by (simp add: Nitpick.case-nat-unfold)
have 16: ( $\forall i < \text{nlength } ls. (\text{if } i = 0 \text{ then } \text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ 0) \text{ else}$ 
  ( $\sum k::\text{nat} = 0..(i). \text{ nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k)) -$ 
  ( $\sum k::\text{nat} = 0..(i-1). \text{ nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k))$ ) =
  (if  $i = 0$  then  $\text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ 0)$  else
     $\text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ i)$ )
  using 13 sum.cl-ivl-Suc[of  $\lambda k. \text{ nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ k) \ 0]$ 
  by (metis Suc-diff-1 enat.distinct(2) enat-add-sub-same less-nat-zero-code not-gr-zero)
have 17: ( $\forall i < \text{nlength } ls. (\text{if } i = 0 \text{ then } \text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ 0) \text{ else}$ 
   $\text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ i)$ ) =
   $\text{nlength } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ ls) \ i)$ )
  by (simp add: Nitpick.case-nat-unfold)

```

have 18: $(\forall i < \text{nlength } ls. \text{nlength } (\text{pfilt } \sigma (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i)) =$
 $\text{nlength } (\text{nsbn } (\text{pfilt } \sigma (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))))$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i)$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ (\text{Suc } i))$
 $)$
by (*simp add: 09 14 15 16 pfilt-nlength*)
have 19: $(\forall i < \text{nlength } ls.$
 $(\forall j \leq \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i).$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ (\text{Suc } i)) \leq$
 $\text{nlength } (\text{pfilt } \sigma (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))))$
by (*simp add: 04 assms lcppl-lsum-less-th-equal-alt pfilt-nlength*)
have 22: $\text{nlastnfirst } (\text{lcppl } f \ g \ \sigma \ ls)$
using *assms lcppl-nfusecat-nlastnfirst-alt by blast*
have 23: $(\forall j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ j) > 0)$
using *assms lcppl-nlength-all-gr-zero-alt by blast*
have 190: $(\forall i < \text{nlength } ls.$
 $(\forall j \leq \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i).$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i) \leq$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ (\text{Suc } i))$
 $)$
using *04 06 09 18 23 lsum-nlength[of (lcppl f g σ ls) 0] lcppl-nlength-alt[of ls f g σ] 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: $(\forall i < \text{nlength } ls.$
 $(\forall j \leq \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i).$
 $(\text{nnth } (\text{nsbn } (\text{pfilt } \sigma (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))))$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i)$
 $(\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ (\text{Suc } i))$
 $) \ j) =$
 $(\text{nnth } (\text{pfilt } \sigma (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))))$
 $((\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i) + j) \) \)$
using *nsbn-nnth[of (pfilt σ (nfusecat (lcppl f g σ ls)))] by simp*
(metis 06 09 18 enat-ord-simps(1) min.orderE)
have 21: $(\forall i < \text{nlength } ls.$
 $(\forall j \leq \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i).$
 $(\text{nnth } (\text{pfilt } \sigma (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))))$
 $((\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 σ (nfusecat (lcppl f g σ ls))]]*
pfilt-nnth[of - σ (nfusecat (lcppl f g σ ls))]]
by (*metis enat-ord-code(4) enat-the-enat not-le-imp-less ntaken-all ntaken-nlast order-less-imp-le*)
have 24: $(\forall i \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls).$
 $(\forall j \leq \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i).$
 $(\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 σ ls)] 04 22 23*
all-gr-zero-nnth-a all-nfinite-nnth-a lcppl-nlength-all-nfinite-alt by blast
have 241: $\text{nlength } (\text{lcppl } f \ g \ \sigma \ ls) = \text{nlength } ls - 1$


```

using assms by (simp add: epred-conv-minus lcppl-nlength-alt)
have 25: ( $\forall i < \text{nlength } ls.$ 
  ( $\forall j \leq \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i).$ 
    ( $\text{nnth } \sigma \ (\text{nnth } (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls))$ 
      ( $((\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i) + j) \ )) =$ 
      ( $\text{nnth } (\text{pfilt } \sigma \ (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i)) \ j) \ ))$ 
    using 04 24 lsum-nlength[of (lcppl f g σ ls) 0] pfilt-nlength[of σ ] pfilt-nnth
    assms(3)
    by (metis 241 enat2-cases enat-ord-code(3) epred-Infty epred-conv-minus
      nfinite-conv-nlength-enat)
have 26: ( $\forall i < \text{nlength } ls.$ 
  ( $\forall j \leq \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i).$ 
    ( $\text{nnth } (\text{pfilt } \sigma \ (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i)) \ j) =$ 
    ( $\text{nnth } (\text{nsubn } (\text{pfilt } \sigma \ (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls)))$ 
      ( $\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ i)$ 
      ( $\text{nnth } (\text{addzero } (\text{lsum } (\text{lcppl } f \ g \ \sigma \ ls) \ 0)) \ (\text{Suc } i)$ 
        )  $j) \ ))$ 
    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 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)
  ( $\forall i < \text{nlength } ls. (\text{nsubn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)) \ ) \models f \ fproj \ g$ )
  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:  $\forall l x \in \text{nset } (\text{lcppl } f \ g \ \sigma \ ls). \ 0 < \text{nlength } l x$ 
  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:  $\forall lx \in nset (lcppl\ f\ g\ \sigma\ ls). \text{nfiniteness}\ lx$ 
  by (metis (no-types, lifting) 0 assms(1) assms(3) assms(6) co.enat.exhaust-sel
    cppl-fprojection i0-less iless-Suc-eq in-nset-conv-nnth lcppl-nlength lcppl-nnth nfiniteness-nmap)
have 141:  $\forall i \leq nlength (lcppl\ f\ g\ \sigma\ ls). 0 < nlength (nnth (lcppl\ f\ g\ \sigma\ ls)\ i)$ 
  using 12 assms
  by (metis enat.distinct(2) enat-the-enat lcppl-nlength-all-gr-zero nfiniteness-conv-nlength-enat)
have 142:  $\forall i \leq nlength (lcppl\ f\ g\ \sigma\ ls). \text{nfiniteness}\ (nnth (lcppl\ f\ g\ \sigma\ ls)\ i)$ 
  using 13 assms
  by (meson in-nset-conv-nnth)
have 15:  $\text{nfiniteness}\ (nfusecat (lcppl\ f\ g\ \sigma\ ls))$ 
  using nfusecat-nfiniteness[of (lcppl f g σ ls)] 10 11 12 13 by blast
have 2:  $nlength\ (\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls))) =$ 
   $(the-enat\ (\sum k::nat = 0..(the-enat(nlength\ (lcppl\ f\ g\ \sigma\ ls))).\ nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k)))$ 
  using 0 assms
  by (metis 10 11 13 15 nfiniteness-conv-nlength-enat nfusecat-nlength-nfiniteness the-enat.simps)
have 3:
   $nlast(\text{addzero}\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) = nlast(\text{addzero}\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))$ 
  using lsum-addzero-nlast
  using assms(1) assms(3) lcppl-nfiniteness by blast
have 4:  $nlast(\text{addzero}\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) =$ 
   $(\sum k::nat = 0::nat..the-enat\ (nlength\ (lcppl\ f\ g\ \sigma\ ls))).\ the-enat\ (nlength\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k)))$ 
  using assms lsum-nlast[of (lcppl f g σ ls) 0] 10 13 plus-nat.add-0 by presburger
have 5:  $(\sum k::nat = 0..the-enat\ (nlength\ (lcppl\ f\ g\ \sigma\ ls))).\ the-enat\ (nlength\ (nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k))) =$ 
   $(the-enat\ (\sum k::nat = 0..the-enat(nlength\ (lcppl\ f\ g\ \sigma\ ls))).\ nlength(nnth\ (lcppl\ f\ g\ \sigma\ ls)\ k)))$ 
  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 nfiniteness-conv-nlength-enat)
qed

```

10.3 Soundness of Projection Axioms

lemma PJ00sem:

$\sigma \models \neg(f\ fproj\ inf)$

by (auto simp add: fprojection-d-def infinite-defs nfiniteness-conv-nlength-enat pfilt-nlength)

lemma OPJ00sem:

$\sigma \models \neg(f\ oproj\ finite)$

by (auto simp add: oprojection-d-def finite-defs nfiniteness-conv-nlength-enat pfilt-nlength)

lemma PJ01sem:

$\sigma \models (f\ fproj\ g) \longrightarrow finite$

by (auto simp add: fprojection-d-def finite-defs nfiniteness-conv-nlength-enat pfilt-nlength)

lemma OPJ01sem:

$\sigma \models (f\ oproj\ g) \longrightarrow inf$

by (auto simp add: oprojection-d-def infinite-defs nfiniteness-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*)

10.3.1 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*

10.3.2 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*

10.3.3 PJ3

lemma *PJ3help*:

assumes *nfinite* σ

shows $nsubn\ \sigma\ 0\ (the-enat\ (nlength\ \sigma)) = \sigma$

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 $f\ \sigma \wedge 0 < nlength\ \sigma \wedge nfinite\ \sigma$

shows $(\exists l. nidx\ l \wedge nnth\ l\ 0 = 0 \wedge nfinite\ l \wedge nfinite\ \sigma \wedge$

$nlast\ l = (the-enat\ (nlength\ \sigma)) \wedge$

$(\forall i < nlength\ l. f\ (nsubn\ \sigma\ (nnth\ l\ i))\ (nnth\ l\ (Suc\ i))) \wedge$

$(\exists \sigma 1. nlength\ \sigma 1 = nlength\ l \wedge$

$(\forall i \leq nlength\ \sigma 1. nnth\ \sigma 1\ i = nnth\ \sigma\ (nnth\ l\ i)) \wedge nlength\ \sigma 1 = Suc\ 0))$

proof –

have 1: $nidx\ (NCons\ 0\ (NNil\ (the-enat\ (nlength\ \sigma))))$

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: $nnth\ (NCons\ 0\ (NNil\ (the-enat\ (nlength\ \sigma))))\ ((the-enat\ (nlength\ (NCons\ 0\ (NNil\ (nlength\ \sigma))))))$

=

$(the-enat\ (nlength\ \sigma))$

by (*metis nfinite-NCons nfinite-NNil nlast-NCons nlast-NNil nlength-NCons nlength-NNil nnth-nlast*)

have 3: $(\forall i < nlength\ (NCons\ 0\ (NNil\ (the-enat\ (nlength\ \sigma))))).$

$f\ (nsubn\ \sigma\ (nnth\ (NCons\ 0\ (NNil\ (the-enat\ (nlength\ \sigma))))\ i)$

$(nnth\ (NCons\ 0\ (NNil\ (the-enat\ (nlength\ \sigma))))\ (Suc\ i))$

using *assms PJ3help*

by (*metis enat-0-iff(1) iless-eSuc0 nlength-NCons nlength-NNil nnth-0 nnth-NNil nnth-Suc-NCons*)

have 4: $nlength\ (NCons\ (nnth\ \sigma\ 0)\ (NNil\ (nnth\ \sigma\ (the-enat\ (nlength\ \sigma)))) =$

$nlength\ (NCons\ 0\ (NNil\ (the-enat\ (nlength\ \sigma))))$

by *simp*

have 5: $(\forall i \leq nlength\ (NCons\ (nnth\ \sigma\ 0)\ (NNil\ (nnth\ \sigma\ (the-enat\ (nlength\ \sigma))))).$

$nnth\ (NCons\ (nnth\ \sigma\ 0)\ (NNil\ (nnth\ \sigma\ (the-enat\ (nlength\ \sigma))))\ i =$

$nnth\ \sigma\ (nnth\ (NCons\ 0\ (NNil\ (the-enat\ (nlength\ \sigma))))\ i)$

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: $nlength\ (NCons\ (nnth\ \sigma\ 0)\ (NNil\ (nnth\ \sigma\ (the-enat\ (nlength\ \sigma)))) = Suc\ 0$

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: $nnth\ (NCons\ 0\ (NNil\ (the-enat\ (nlength\ \sigma))))\ 0 = 0$

using *nnth-0 by simp*

show *?thesis*

using 1 2 3 4 5 6 7 8 9 *assms by blast*

qed

lemma *PJ3sem*:

$(\sigma \models f \text{ fproj } \text{skip}) = (f \wedge \text{more} \wedge \text{finite})$

proof –

have 1: $(\sigma \models f \text{ fproj } \text{skip}) \implies (\sigma \models f \wedge \text{more} \wedge \text{finite})$

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: $(\sigma \models f \wedge \text{more} \wedge \text{finite}) \implies (\sigma \models f \text{ fproj } \text{skip})$

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*

10.3.4 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: $(\exists \text{ls}. \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \text{nfinite } \text{ls} \wedge \text{nfinite } \sigma \wedge$

$\text{nlast } \text{ls} = (\text{the-enat } (\text{nlength } \sigma)) \wedge$

$\text{powerinterval } f \sigma \text{ls} \wedge$

$(\exists n \leq \text{nlength } (\text{pfilt } \sigma \text{ls}). g (\text{ntaken } n (\text{pfilt } \sigma \text{ls})) \wedge h (\text{ndropn } n (\text{pfilt } \sigma \text{ls}))))$

using *assms* **unfolding** *chop-defs fprojection-d-def*

by (*metis nfinite-conv-nlength-enat pfilt-nlength the-enat.simps*)

obtain *ls* **where** 3: $\text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \text{nfinite } \text{ls} \wedge \text{nfinite } \sigma \wedge$

$\text{nlast } \text{ls} = (\text{the-enat } (\text{nlength } \sigma)) \wedge$

$\text{powerinterval } f \sigma \text{ls} \wedge$

$(\exists n \leq \text{nlength } (\text{pfilt } \sigma \text{ls}). g (\text{ntaken } n (\text{pfilt } \sigma \text{ls})) \wedge h (\text{ndropn } n (\text{pfilt } \sigma \text{ls})))$

using 2 **by** *auto*

have 4: $\text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0$

using 3 **by** *auto*

have 5: $\text{powerinterval } f \sigma \text{ls}$

using 3 **by** *auto*

have 6: $\text{nlast } \text{ls} = \text{the-enat } (\text{nlength } \sigma)$

using 3 **by** *auto*

have 7: $(\exists n \leq \text{nlength } (\text{pfilt } \sigma \text{ls}). g (\text{ntaken } n (\text{pfilt } \sigma \text{ls})) \wedge h (\text{ndropn } n (\text{pfilt } \sigma \text{ls})))$

using 3 **by** *auto*

obtain *n* **where** 8: $n \leq \text{nlength } (\text{pfilt } \sigma \text{ls}) \wedge g (\text{ntaken } n (\text{pfilt } \sigma \text{ls})) \wedge h (\text{ndropn } n (\text{pfilt } \sigma \text{ls}))$

using 7 **by** *auto*

have 9: $n \leq \text{nlength } (\text{pfilt } \sigma \text{ls})$

```

using 8 by auto
have 10: g (ntaken n (pfilt σ ls))
using 8 by auto
have 11: h (ndropn n (pfilt σ ls))
using 8 by auto
have 12: nidx (ntaken n ls) ∧ nnth (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: nidx (ndropn n ls) ∧ 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 (λx. x - (nnth ls n)) (ndropn n ls))) = ndropn n (nmap (λx. x - nnth ls n) ls)
  using ndropn-nmap[of n (λx. x - (nnth ls n)) ls] by presburger
have 132: nnth (ndropn n (nmap (λx. x - nnth ls n) ls)) 0 = 0
  by (metis 13 131 diff-self-eq-0 nnth-nmap zero-enat-def zero-le)
have 133:  $\bigwedge j. \text{enat } j \leq \text{nlength } ls \implies \text{nnth } (nmap (\lambda x. x - \text{nnth } ls \ n) \ ls) \ j = \text{nnth } ls \ j - \text{nnth } ls \ n$ 
  using nnth-nmap by blast
have 134: nlength (ndropn n (nmap (λx. x - nnth ls n) ls)) = nlength ls - enat n
  by simp
have 135:  $\bigwedge j. j \leq \text{nlength } ls - \text{enat } n \implies$ 
  nnth (ndropn n (nmap (λx. x - nnth ls n) ls)) j = nnth ls (n+ j) - nnth ls n
  by (metis 131 ndropn-nlength ndropn-nnth nnth-nmap)
have 136:  $\bigwedge j. (\text{Suc } j) \leq \text{nlength } ls - \text{enat } n \implies$ 
  nnth (ndropn n (nmap (λx. x - nnth ls n) ls)) (Suc j) = nnth ls (n+ (Suc j)) - nnth ls n
  using 135 by blast
have 137:  $\bigwedge j. (\text{Suc } j) \leq \text{nlength } ls - \text{enat } n \implies$ 
  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 (λx. x - nnth ls n) ls))
  by (metis 134 135 137 Suc-ile-eq dual-order.strict-iff-order nidx-expand)
have 15: g (pfilt σ (ntaken n ls))
  by (metis 10 9 pfilt-nlength pfilt-ntaken)
have 16: h (pfilt σ (ndropn n ls))
  by (metis 8 pfilt-ndropn pfilt-nlength)
have 17: g (pfilt (ntaken (nnth ls n) σ) (ntaken n ls))
  by (metis 12 15 nfinite-ntaken nle-le ntaken-all ntaken-nlast pfilt-ntaken-nidx)
have 170: nmap (λx. nnth σ (nnth ls n + x)) (nmap (λx. x - nnth ls n) (ndropn n ls)) =
  nmap ((λx. nnth σ (nnth ls n + x)) ∘ (λx. x - nnth ls n)) (ndropn n ls)
  using nellist.map-comp by blast
have 171: nmap ((λx. nnth σ (nnth ls n + x)) ∘ (λx. x - nnth ls n)) (ndropn n ls) =
  nmap (nnth σ) (ndropn n ls)
proof -
  have 172: nlength (nmap ((λx. nnth σ (nnth ls n + x)) ∘ (λx. x - nnth ls n)) (ndropn n ls)) =
    nlength (nmap (nnth σ) (ndropn n ls))
  by auto
  have 173:  $\bigwedge j. j \leq \text{nlength } (nmap (\text{nnth } \sigma) (\text{ndropn } n \ ls)) \implies$ 
    nnth (nmap ((λx. nnth σ (nnth ls n + x)) ∘ (λx. x - nnth ls n)) (ndropn n ls)) j =
    nnth (nmap (nnth σ) (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)  $\sigma$ ) ((nmap ( $\lambda x. x - (nnth\ ls\ n)$ ) (ndropn n ls)))]
    pfilt-nmap[of  $\sigma$  (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  $\sigma$  f]
  by (metis 3 9 pfilt-nlength)
have 21: (nnth ls n)  $\leq$  nlength  $\sigma$ 
  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 ( $\lambda x. x - (nnth\ ls\ n)$ ) (ndropn n ls)))
  (the-enat(nlength ((nmap ( $\lambda x. x - (nnth\ ls\ n)$ ) (ndropn n ls))))) =
  (the-enat(nlength  $\sigma$ )) - (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 ( $\lambda x. x - (nnth\ ls\ n)$ ) (ndropn n ls))) = (the-enat(nlength  $\sigma$ )) - (nnth ls n)
  by (metis 23 3 nfinite-ndropn-a nfinite-nmap nnth-nlast)
have 224: (nmap ( $\lambda x. x + (nnth\ ls\ n)$ ) ((nmap ( $\lambda x. x - (nnth\ ls\ n)$ ) (ndropn n ls)))) = ndropn n ls
  proof -
    have 2241: nlength (nmap ( $\lambda x. x + (nnth\ ls\ n)$ ) ((nmap ( $\lambda x. x - (nnth\ ls\ n)$ ) (ndropn n ls)))) =
      nlength (ndropn n ls)
    by auto
    have 2242:  $\bigwedge j. j \leq nlength\ (ndropn\ n\ ls) \longrightarrow$ 
      nnth (nmap ( $\lambda x. x + (nnth\ ls\ n)$ ) ((nmap ( $\lambda x. x - (nnth\ ls\ n)$ ) (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 nellist-eq-nnth-eq
    by metis
  qed
have 24: ls = nfuse (ntaken n ls)
  (nmap ( $\lambda x. x + (nnth\ ls\ n)$ ) ((nmap ( $\lambda x. x - (nnth\ ls\ n)$ ) (ndropn n ls))))
  using nfuse-ntaken-ndropn
  by (metis 224 9 pfilt-nlength)
have 241: nfinite (ntaken n ls)

```

by *simp*
 have 242: $\text{nfinite } (\text{ntaken } (\text{nnth } ls \ n) \ \sigma)$
 by *simp*
 have 243: $\text{nlast } (\text{ntaken } n \ ls) = (\text{the-enat}(\text{nlength } (\text{ntaken } (\text{nnth } ls \ n) \ \sigma)))$
 by (*simp add: 21 222*)
 have 25: $(\exists ls1. \text{nidx } ls1 \wedge \text{nnth } ls1 \ 0 = 0 \wedge \text{nfinite } ls1 \wedge \text{nfinite } (\text{ntaken } (\text{nnth } ls \ n) \ \sigma) \wedge$
 $\text{nlast } ls1 = (\text{the-enat}(\text{nlength } (\text{ntaken } (\text{nnth } ls \ n) \ \sigma))) \wedge$
 $\text{powerinterval } f \ (\text{ntaken } (\text{nnth } ls \ n) \ \sigma) \ ls1 \wedge g \ (\text{pfilt } (\text{ntaken } (\text{nnth } ls \ n) \ \sigma) \ ls1))$
 using 12 17 19 241 242 243 by *blast*
 have 251: $\text{nfinite } (\text{ndropn } (\text{nnth } ls \ n) \ \sigma)$
 by (*simp add: 3*)
 have 252: $\text{nlast } (\text{nmap } (\lambda x. x - \text{nnth } ls \ n) \ (\text{ndropn } n \ ls)) = \text{the-enat}(\text{nlength } (\text{ndropn } (\text{nnth } ls \ n) \ \sigma))$
 by (*metis 223 3 idiff-enat-enat ndropn-nlength nfinite-nlength-enat the-enat.simps*)
 have 26: $(\exists ls2. \text{nidx } ls2 \wedge \text{nnth } ls2 \ 0 = 0 \wedge \text{nfinite } ls2 \wedge \text{nfinite } (\text{ndropn } (\text{nnth } ls \ n) \ \sigma) \wedge$
 $\text{nlast } ls2 = \text{the-enat}(\text{nlength } (\text{ndropn } (\text{nnth } ls \ n) \ \sigma)) \wedge$
 $\text{powerinterval } f \ (\text{ndropn } (\text{nnth } ls \ n) \ \sigma) \ ls2 \wedge h \ (\text{pfilt } (\text{ndropn } (\text{nnth } ls \ n) \ \sigma) \ ls2))$
 by (*metis 131 132 14 18 20 252 3 nfinite-ndropn-a nfinite-nmap*)
 have 27: $(\text{ntaken } (\text{nnth } ls \ n) \ \sigma) \models f \text{ fproj } g$
 by (*metis 25 fprojection-d-def*)
 have 28: $(\text{ndropn } (\text{nnth } ls \ n) \ \sigma) \models f \text{ fproj } h$
 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 $(\sigma \models f \text{ oproj } ((g \wedge \text{finite}); h))$
 shows $(\sigma \models (f \text{ fproj } g) ; (f \text{ oproj } h))$
 proof –
 have 1: $(\sigma \models f \text{ oproj } ((g \wedge \text{finite}); h))$
 using *assms* by *auto*
 have 2: $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$
 $\text{powerinterval } f \ \sigma \ ls \wedge$
 $(\exists n \leq \text{nlength } (\text{pfilt } \sigma \ ls). g \ (\text{ntaken } n \ (\text{pfilt } \sigma \ ls)) \wedge h \ (\text{ndropn } n \ (\text{pfilt } \sigma \ ls))))$
 using *assms* **unfolding** *chop-defs oprojection-d-def finite-defs*
 by *auto*
 obtain *ls* **where** 3: $\text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$
 $\text{powerinterval } f \ \sigma \ ls \wedge$
 $(\exists n \leq \text{nlength } (\text{pfilt } \sigma \ ls). g \ (\text{ntaken } n \ (\text{pfilt } \sigma \ ls)) \wedge h \ (\text{ndropn } n \ (\text{pfilt } \sigma \ ls)))$
 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 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 (ntaken n (pfilt σ ls))
  using 8 by auto
have 11: h (ndropn n (pfilt σ ls))
  using 8 by auto
have 12: nidx (ntaken n ls) ∧ 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) ∧ 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 (λx. x - (nnth ls n)) (ndropn n ls))) = ndropn n (nmap (λx. x - nnth ls n) ls)
  using ndropn-nmap[of n (λx. x - (nnth ls n)) ls] by presburger
have 132: nnth (ndropn n (nmap (λx. x - nnth ls n) ls)) 0 = 0
  by (metis 13 131 diff-self-eq-0 nnth-nmap zero-enat-def zero-le)
have 133:  $\bigwedge j. \text{enat } j \leq \text{nlength } ls \implies \text{nnth } (nmap (\lambda x. x - \text{nnth } ls \ n) \ ls) \ j = \text{nnth } ls \ j - \text{nnth } ls \ n$ 
  using nnth-nmap by blast
have 134: nlength (ndropn n (nmap (λx. x - nnth ls n) ls)) = nlength ls - enat n
  by simp
have 135:  $\bigwedge j. j \leq \text{nlength } ls - \text{enat } n \implies$ 
  nnth (ndropn n (nmap (λx. x - nnth ls n) ls)) j = nnth ls (n+ j) - nnth ls n
  by (metis 131 ndropn-nlength ndropn-nnth nnth-nmap)
have 136:  $\bigwedge j. (\text{Suc } j) \leq \text{nlength } ls - \text{enat } n \implies$ 
  nnth (ndropn n (nmap (λx. x - nnth ls n) ls)) (Suc j) = nnth ls (n+ (Suc j)) - nnth ls n
  using 135 by blast
have 137:  $\bigwedge j. (\text{Suc } j) \leq \text{nlength } ls - \text{enat } n \implies$ 
  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 (λx. x - nnth ls n) ls))
  by (metis 134 135 137 Suc-ile-eq dual-order.order-iff-strict nidx-expand)
have 15: g (pfilt σ (ntaken n ls))
  by (metis 10 9 pfilt-nlength pfilt-ntaken)
have 16: h (pfilt σ (ndropn n ls))
  by (metis 8 pfilt-ndropn pfilt-nlength)
have 17: g (pfilt (ntaken (nnth ls n) σ) (ntaken n ls))
  by (metis 12 15 nfinite-ntaken nle-le ntaken-all ntaken-nlast pfilt-ntaken-nidx)
have 170: nmap (λx. nnth σ (nnth ls n + x)) (nmap (λx. x - nnth ls n) (ndropn n ls)) =
  nmap ((λx. nnth σ (nnth ls n + x)) ∘ (λx. x - nnth ls n)) (ndropn n ls)
  using nellist.map-comp by blast
have 171: nmap ((λx. nnth σ (nnth ls n + x)) ∘ (λx. x - nnth ls n)) (ndropn n ls) =
  nmap (nnth σ) (ndropn n ls)
proof -
  have 172: nlength (nmap ((λx. nnth σ (nnth ls n + x)) ∘ (λx. x - nnth ls n)) (ndropn n ls)) =
    nlength (nmap (nnth σ) (ndropn n ls))
    by auto
  have 173:  $\bigwedge j. j \leq \text{nlength } (nmap (\text{nnth } \sigma) (\text{ndropn } n \ ls)) \implies$ 
    nnth (nmap ((λx. nnth σ (nnth ls n + x)) ∘ (λx. x - nnth ls n)) (ndropn n ls)) j =
    nnth (nmap (nnth σ) (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) σ) ((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) =
  (pfilt σ (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) σ) ((nmap (λx. x - (nnth ls n)) (ndropn n ls))))
  using 16 171 by auto
have 19: powerinterval f (ntaken (nnth ls n) σ) (ntaken n ls)
  by (metis 3 8 pfilt-nlength powerinterval-splita-alt)
have 20: powerinterval f (ndropn (nnth ls n) σ) ((nmap (λx. x - (nnth ls n)) (ndropn n ls)))
  using powerinterval-split[of ls n σ f]
  by (metis 3 8 pfilt-nlength powerinterval-splitb-alt)
have 21: (nnth ls n) ≤ nlength σ
  by (metis 3 linorder-le-cases nfinite-ntaken ntaken-all)
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 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:  $\bigwedge j. j \leq nlength (ndropn n ls) \longrightarrow$ 
      nnth (nmap (λx. x + (nnth ls n)) ((nmap (λx. x - (nnth ls n)) (ndropn n ls)))) j =
      nnth (ndropn n ls) j
    by (metis 13 132 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 = 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 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 21 222 3
  by (metis min.orderE nfinite-ntaken ntaken-nlength the-enat.simps)
have 26: (∃ ls2. nidx ls2 ∧ nnth ls2 0 = 0 ∧ ¬nfinite ls2 ∧ ¬nfinite (ndropn (nnth ls n) σ) ∧
  powerinterval f (ndropn (nnth ls n) σ) ls2 ∧ h (pfilt (ndropn (nnth ls n) σ) ls2))
  using assms 14 18 20 3 132 131 by (metis nfinite-ndropn nfinite-nmap)
have 27: (ntaken (nnth ls n) σ) ⊨ f fproj g
  by (metis 25 fprojection-d-def)
have 28: (ndropn (nnth ls n) σ) ⊨ f oproj h
  by (metis 26 oprojection-d-def)

```

show ?thesis
 using 21 27 28 chop-defs by auto
 qed

lemma PJ4semchainb:

assumes $(\sigma \models (f \text{ fproj } g) ; (f \text{ fproj } h))$

shows $(\sigma \models f \text{ fproj } (g;h))$

proof –

have 1: $(\exists n \leq \text{nlength } \sigma.$

$(\exists \text{ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \text{nfinite } \text{ls} \wedge \text{nfinite } (\text{ntaken } n \sigma) \wedge$
 $\text{nlast } \text{ls} = (\text{the-enat}(\text{nlength } (\text{ntaken } n \sigma)))) \wedge$

$\text{powerinterval } f (\text{ntaken } n \sigma) \text{ls} \wedge g (\text{pfilt } (\text{ntaken } n \sigma) \text{ls})) \wedge$
 $(\exists \text{ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \text{nfinite } \text{ls} \wedge \text{nfinite } (\text{ndropn } n \sigma) \wedge$
 $\text{nlast } \text{ls} = (\text{the-enat}(\text{nlength } (\text{ndropn } n \sigma)))) \wedge$
 $\text{powerinterval } f (\text{ndropn } n \sigma) \text{ls} \wedge h (\text{pfilt } (\text{ndropn } n \sigma) \text{ls}))$

using assms unfolding chop-defs fprojection-d-def by simp blast

obtain cp where 2: $\text{cp} \leq \text{nlength } \sigma \wedge$

$(\exists \text{ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \text{nfinite } \text{ls} \wedge \text{nfinite } (\text{ntaken } \text{cp } \sigma) \wedge$
 $\text{nlast } \text{ls} = (\text{the-enat}(\text{nlength } (\text{ntaken } \text{cp } \sigma)))) \wedge$
 $\text{powerinterval } f (\text{ntaken } \text{cp } \sigma) \text{ls} \wedge g (\text{pfilt } (\text{ntaken } \text{cp } \sigma) \text{ls})) \wedge$
 $(\exists \text{ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \text{nfinite } \text{ls} \wedge \text{nfinite } (\text{ndropn } \text{cp } \sigma) \wedge$
 $\text{nlast } \text{ls} = (\text{the-enat}(\text{nlength } (\text{ndropn } \text{cp } \sigma)))) \wedge$
 $\text{powerinterval } f (\text{ndropn } \text{cp } \sigma) \text{ls} \wedge h (\text{pfilt } (\text{ndropn } \text{cp } \sigma) \text{ls}))$

using 1 by auto

have 3: $\text{cp} \leq \text{nlength } \sigma$

using 2 by auto

have 4: $(\exists \text{ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \text{nfinite } \text{ls} \wedge \text{nfinite } (\text{ntaken } \text{cp } \sigma) \wedge$
 $\text{nlast } \text{ls} = (\text{the-enat}(\text{nlength } (\text{ntaken } \text{cp } \sigma)))) \wedge$
 $\text{powerinterval } f (\text{ntaken } \text{cp } \sigma) \text{ls} \wedge g (\text{pfilt } (\text{ntaken } \text{cp } \sigma) \text{ls}))$

using 2 by auto

obtain ls1 where 5: $\text{nidx } \text{ls1} \wedge \text{nnth } \text{ls1 } 0 = 0 \wedge \text{nfinite } \text{ls1} \wedge \text{nfinite } (\text{ntaken } \text{cp } \sigma) \wedge$
 $\text{nlast } \text{ls1} = (\text{the-enat}(\text{nlength } (\text{ntaken } \text{cp } \sigma)))) \wedge$
 $\text{powerinterval } f (\text{ntaken } \text{cp } \sigma) \text{ls1} \wedge g (\text{pfilt } (\text{ntaken } \text{cp } \sigma) \text{ls1})$

using 4 by auto

have 6: $\text{nidx } \text{ls1} \wedge \text{nnth } \text{ls1 } 0 = 0$

using 5 by auto

have 61: $\text{nfinite } \text{ls1} \wedge \text{nfinite } (\text{ntaken } \text{cp } \sigma)$

using 5 by auto

have 7: $\text{nlast } \text{ls1} = \text{the-enat}(\text{nlength } (\text{ntaken } \text{cp } \sigma))$

using 5 by auto

have 8: $\text{powerinterval } f (\text{ntaken } \text{cp } \sigma) \text{ls1}$

using 5 by auto

have 9: $g (\text{pfilt } (\text{ntaken } \text{cp } \sigma) \text{ls1})$

using 5 by auto

have 10: $(\exists \text{ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \text{nfinite } \text{ls} \wedge \text{nfinite } (\text{ndropn } \text{cp } \sigma) \wedge$
 $\text{nlast } \text{ls} = (\text{the-enat}(\text{nlength } (\text{ndropn } \text{cp } \sigma)))) \wedge$
 $\text{powerinterval } f (\text{ndropn } \text{cp } \sigma) \text{ls} \wedge h (\text{pfilt } (\text{ndropn } \text{cp } \sigma) \text{ls}))$

using 2 by auto

obtain ls2 where 11: $\text{nidx } \text{ls2} \wedge \text{nnth } \text{ls2 } 0 = 0 \wedge \text{nfinite } \text{ls2} \wedge \text{nfinite } (\text{ndropn } \text{cp } \sigma) \wedge$
 $\text{nlast } \text{ls2} = (\text{the-enat}(\text{nlength } (\text{ndropn } \text{cp } \sigma)))) \wedge$

```

      powerinterval f (ndropn cp σ) ls2 ∧ h (pfilt (ndropn cp σ) ls2)
    using 10 by auto
  have 12: nidx ls2 ∧ nnth ls2 0 = 0
    using 11 by auto
  have 13: nlast ls2 = the-enat(nlength (ndropn cp σ))
    using 11 by auto
  have 14: powerinterval f (ndropn cp σ) ls2
    using 11 by auto
  have 15: h (pfilt (ndropn cp σ) ls2)
    using 11 by auto
  have 150: nlast ls1 = cp
    by (simp add: 2 7)
  have 151: nfirst (nmap (λx. 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 (λx. x+ cp) ls2)) 0 = 0
    by (metis 150 151 5 nfuse-nnth zero-enat-def zero-le)
  have 153: nidx ls1 ∧ nnth ls1 0 = 0 ∧ nfinite ls1 ∧ nlast ls1 = cp
    by (simp add: 150 5)
  have 154: nidx ls2 ∧ nnth ls2 0 = 0 ∧ nfinite ls2 ∧ nfinite σ ∧ nlast ls2 = the-enat (nlength σ) - cp
    by (metis 11 idiff-enat-enat ndropn-nlength nfinite-conv-nlength-enat nfinite-ndropn the-enat.simps)
  have 16: nidx (nfuse ls1 (nmap (λx. x+ cp) ls2)) ∧ nnth (nfuse ls1 (nmap (λx. x+ cp) ls2)) 0 = 0
    using nidx-nfuse-nidx[of ls1 ls2 cp σ ] 153 154 3 by fastforce
  have 17: nlast ls1 = nfirst (nmap (λx. x+ cp) ls2)
    by (simp add: 150 151)
  have 18: nnth (nfuse ls1 (nmap (λx. x+ cp) ls2)) (the-enat(nlength ls1)) = cp
    by (metis 151 17 5 ntaken-nfuse ntaken-nlast)
  have 19: nlast (nfuse ls1 (nmap (λx. x+ cp) ls2)) = (the-enat (nlength σ))
    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: powerinterval f σ (nfuse ls1 (nmap (λx. x+ cp) ls2))
    using powerinterval-nfuse[of ls1 (ls2) cp σ f] 11 150 154 3 5 by fastforce
  have 21: σ = nfuse (ntaken cp σ) (ndropn cp σ)
    by (simp add: 2 nfuse-ntaken-ndropn)
  have 22: nnth ((nfuse ls1 (nmap (λx. x+ cp) ls2))) (the-enat(nlength ls1)) = cp
    using 18 by blast
  have 23: (ntaken (the-enat(nlength ls1)) (pfilt σ (nfuse ls1 (nmap (λx. x+ cp) ls2))) ) =
    (pfilt (ntaken cp σ) 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 (ntaken (the-enat(nlength ls1)) (pfilt σ (nfuse ls1 (nmap (λx. x+ cp) ls2))) )
    by (simp add: 23 9)
  have 25: (ndropn (the-enat(nlength ls1)) (pfilt σ (nfuse ls1 (nmap (λx. x+ cp) ls2))) ) =
    (pfilt (ndropn cp σ) ls2)
    using pfilt-ndropn[of (the-enat(nlength ls1)) (nfuse ls1 (nmap (λx. x+ cp) ls2)) σ]
      pfilt-ndropn-nidx[of cp σ 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: nlength ls1 ≤ nlength (pfilt σ (nfuse ls1 (nmap (λx. x+ cp) ls2)))
    by (simp add: nfuse-nlength pfilt-nlength)

```

have 27: $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge$
 $\text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$
 $\text{powerinterval } f \ \sigma \ ls \wedge$
 $(\exists n \leq \text{nlength } (\text{pfilt } \sigma \ ls). \ g \ (\text{ntaken } n \ (\text{pfilt } \sigma \ ls)) \wedge h \ (\text{ndropn } n \ (\text{pfilt } \sigma \ ls))))$
using 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: $cp \leq \text{nlength } \sigma \wedge$

$(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } (\text{ntaken } cp \ \sigma) \wedge$

$\text{nlast } ls = (\text{the-enat}(\text{nlength } (\text{ntaken } cp \ \sigma)))) \wedge$

$\text{powerinterval } f \ (\text{ntaken } cp \ \sigma) \ ls \wedge g \ (\text{pfilt } (\text{ntaken } cp \ \sigma) \ ls)) \wedge$

$(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } (\text{ndropn } cp \ \sigma) \wedge$

$\text{powerinterval } f \ (\text{ndropn } cp \ \sigma) \ ls \wedge h \ (\text{pfilt } (\text{ndropn } cp \ \sigma) \ ls))))$

using 1 **by** auto

have 3: $cp \leq \text{nlength } \sigma$

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 $(\text{simp add: 2 } \gamma)$
have 151: $\text{nfirst } (\text{nmap } (\lambda x. x + cp) \ ls2) = cp$
by $(\text{metis 11 } \text{NNil-eq-nmap-conv } \text{nlast-NNil } \text{ntake-eq-NNil-iff } \text{ntake-nmap } \text{ntaken-0 } \text{ntaken-nlast}$
 $\text{plus-nat.add-0})$
have 152: $\text{nnth } (\text{nfuse } ls1 \ (\text{nmap } (\lambda x. x + cp) \ ls2)) \ 0 = 0$
by $(\text{metis 150 151 5 } \text{nfuse-nnth } \text{zero-enat-def } \text{zero-le})$
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: $\text{nidx } (\text{nfuse } ls1 \ (\text{nmap } (\lambda x. x + cp) \ ls2)) \wedge \text{nnth } (\text{nfuse } ls1 \ (\text{nmap } (\lambda x. x + cp) \ ls2)) \ 0 = 0$
using $\text{nidx-nfuse-nidx-infinite[of } ls1 \ ls2 \ cp \ \sigma \] \ 150 \ 154 \ 5$ **by fastforce**
have 17: $\text{nlast } ls1 = \text{nfirst } (\text{nmap } (\lambda x. x + cp) \ ls2)$
by $(\text{simp add: 150 151})$
have 18: $\text{nnth } (\text{nfuse } ls1 \ (\text{nmap } (\lambda x. x + cp) \ ls2)) \ (\text{the-enat}(\text{nlength } ls1)) = cp$
by $(\text{metis 150 17 5 } \text{ntaken-nfuse } \text{ntaken-nlast})$
have 20: $\text{powerinterval } f \ \sigma \ (\text{nfuse } ls1 \ (\text{nmap } (\lambda x. x + cp) \ ls2))$
using $\text{powerinterval-nfuse-alt[of } ls1 \ (ls2) \ cp \ \sigma \ f]$
using 14 154 3 5 by fastforce
have 21: $\sigma = \text{nfuse } (\text{ntaken } cp \ \sigma) \ (\text{ndropn } cp \ \sigma)$
by $(\text{simp add: 2 nfuse-ntaken-ndropn})$
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 $(\text{metis 150 17 2 5 } \text{enat.distinct(2) } \text{enat-le-plus-same(1) } \text{enat-the-enat}$
 $\text{nfinite-conv-nlength-enat } \text{nfuse-nlength } \text{ntaken-nfuse } \text{pfilt-ntaken } \text{pfilt-ntaken-nidx})$
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 $(\text{metis 12 17 2 23 5 } \text{enat-ord-code(4) } \text{enat-the-enat } \text{min-def } \text{ndropn-nfuse } \text{nle-le}$
 $\text{ntaken-nlength } \text{order-less-imp-le } \text{pfilt-nlength})$
have 26: $\text{nlength } ls1 \leq \text{nlength } (\text{pfilt } \sigma \ (\text{nfuse } ls1 \ (\text{nmap } (\lambda x. x + cp) \ ls2)))$
by $(\text{simp add: nfuse-nlength pfilt-nlength})$

have 27: $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls \ 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$
 $\text{powerinterval } f \ \sigma \ ls \wedge$
 $(\exists n \leq \text{nlength } (pfilt \ \sigma \ ls). \ g \ (\text{ntaken } n \ (pfilt \ \sigma \ ls)) \wedge h \ (\text{ndropn } n \ (pfilt \ \sigma \ ls))))$
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

10.3.5 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)

10.3.6 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  $nidx\ ls$ 
           $nnth\ ls\ 0 = 0$ 
           $\neg nfinite\ ls$ 
           $\neg nfinite\ \sigma$ 
shows  $(\forall i. 0 \leq i \wedge i < nlength\ ls \longrightarrow nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ )$ 
         $= (nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i))$ 
proof
  fix  $i$ 
  show  $0 \leq i \wedge i < nlength\ ls \longrightarrow$ 
         $nlength\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ) = nnth\ ls\ (Suc\ i) - nnth\ ls\ i$ 
  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  $(metis\ One-nat-def\ Suc-ile-eq\ assms(1)\ assms(5)\ diff-add\ nidx-expand\ order-less-imp-le$ 
     $plus-1-eq-Suc)$ 
qed
qed

```

lemma *OPJ6help2*:

```

assumes  $nidx\ ls$ 
           $nnth\ ls\ 0 = 0$ 
           $\neg nfinite\ ls$ 
           $\neg nfinite\ \sigma$ 
           $(\forall i < nlength\ ls. nnth\ ls\ (Suc\ i) - nnth\ ls\ i = Suc\ 0)$ 
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 nidx-expand order-less-imp-le
    plus-1-eq-Suc)
qed
qed

```

lemma *PJ6help3*:

```

assumes nidx ls
        nnth ls 0 = 0
        nfinite ls
        nfinite σ
        nlast ls = the-enat(nlength σ)
        (∀ i ≤ nlength ls. nnth ls i = i)
shows   (∀ i < nlength ls. nnth ls (Suc i) - nnth ls i = Suc 0)
proof
  fix i
  show i < nlength ls ⟶ nnth ls (Suc i) - nnth ls i = 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 nidx ls
        nnth ls 0 = 0
        ¬nfinite ls
        ¬nfinite σ
        (∀ i ≤ nlength ls. nnth ls i = i)
shows   (∀ i < nlength ls. nnth ls (Suc i) - nnth ls i = Suc 0)
proof
  fix i
  show i < nlength ls ⟶ nnth ls (Suc i) - nnth ls i = 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 nfinite σ
shows   (∃ ls. nidx ls ∧ nnth ls 0 = 0 ∧ nfinite ls ∧ nfinite σ ∧
        ls = ntaken (the-enat (nlength σ)) (niterates Suc 0) ∧
        nlast ls = the-enat(nlength σ) ∧
        (∀ i ≤ nlength ls. nnth ls i = i) ∧
        (∃ σ1. nlength σ1 = nlength ls ∧ (∀ i ≤ nlength σ1. nnth σ1 i = nnth σ (i)) ∧ σ1 = σ))

```

proof –

have 1: $\text{nidx } (\text{ntaken } (\text{the-enat}(\text{nlength } \sigma)) (\text{niterates } \text{Suc } 0))$
using $\text{nidx-ntaken-niterates-Suc}$ **by** presburger
have 2: $\text{nnth } (\text{ntaken } (\text{the-enat } (\text{nlength } \sigma)) (\text{niterates } \text{Suc } 0)) \ 0 = 0$
by $(\text{simp add: ntaken-nnth})$
have 3: $\text{nlast } (\text{ntaken } (\text{the-enat } (\text{nlength } \sigma)) (\text{niterates } \text{Suc } 0)) = \text{the-enat}(\text{nlength } \sigma)$
by $(\text{simp add: ntaken-nlast})$
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 $(\text{simp add: ntaken-nnth})$
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** $(\text{simp add: enat-the-enat nfinite-nlength-enat})$
show $?thesis$
using $1 \ 2 \ 3 \ 4 \ 5 \ \text{assms } \text{nfinite-ntaken}$ **by** blast
qed

lemma $OPJ6\text{help4}$:

assumes $\neg \text{nfinite } \sigma$

shows $(\exists \text{ls}. \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \neg \text{nfinite } \text{ls} \wedge \neg \text{nfinite } \sigma \wedge$
 $\text{ls} = (\text{niterates } \text{Suc } 0) \wedge$
 $(\forall i \leq \text{nlength } \text{ls}. \text{nnth } \text{ls } i = i) \wedge$
 $(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } \text{ls} \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 \ i = \text{nnth } \sigma \ (i)) \wedge \sigma 1 = \sigma))$

proof –

have 1: $\text{nidx } ((\text{niterates } \text{Suc } 0))$
using $\text{nidx-expand nnth-niterates-Suc}$ **by** presburger
have 2: $\text{nnth } ((\text{niterates } \text{Suc } 0)) \ 0 = 0$
by (simp)
have 4: $(\forall i \leq \text{nlength } ((\text{niterates } \text{Suc } 0)). \text{nnth } ((\text{niterates } \text{Suc } 0)) \ i = i)$
by (simp)
have 5: $(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ((\text{niterates } \text{Suc } 0)) \wedge$
 $(\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 \ i = \text{nnth } \sigma \ i) \wedge \sigma 1 = \sigma)$
using assms **by** $(\text{simp add: nfinite-conv-nlength-enat})$
show $?thesis$
using $1 \ 2 \ 4 \ 5 \ \text{assms}$ **by** blast
qed

lemma $PJ6\text{help5}$:

assumes $\text{nfinite } \sigma$

shows $(\exists \text{ls}. \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \text{nfinite } \text{ls} \wedge \text{nfinite } \sigma \wedge$
 $\text{nlast } \text{ls} = \text{the-enat}(\text{nlength } \sigma) \wedge$
 $(\forall i < \text{nlength } \text{ls}. \text{nlength } (\text{nsubn } \sigma \ (\text{nnth } \text{ls } i)) \ (\text{nnth } \text{ls } (\text{Suc } i)) = \text{Suc } 0) \wedge$
 $(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } \text{ls} \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 \ i = \text{nnth } \sigma \ (\text{nnth } \text{ls } i)) \wedge g \ \sigma 1))$
 $= g \ \sigma$

proof –

have $(\exists \text{ls}. \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \text{nfinite } \text{ls} \wedge \text{nfinite } \sigma \wedge$
 $\text{nlast } \text{ls} = \text{the-enat } (\text{nlength } \sigma) \wedge$
 $(\forall i < \text{nlength } \text{ls}. \text{nlength } (\text{nsubn } \sigma \ (\text{nnth } \text{ls } i)) \ (\text{nnth } \text{ls } (\text{Suc } i)) = \text{Suc } 0) \wedge$

$(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 i = \text{nnth } \sigma (\text{nnth } ls i)) \wedge g \sigma 1))$
 $=$
 $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge$
 $\text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$
 $(\forall i < \text{nlength } ls. \text{nnth } ls (\text{Suc } i) - \text{nnth } ls i = \text{Suc } 0) \wedge$
 $(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 i = \text{nnth } \sigma (\text{nnth } ls i)) \wedge g \sigma 1))$
using *PJ6help1* [*of* - σ] **assms by auto**
also have ... =
 $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge$
 $\text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$
 $(\forall i \leq \text{nlength } ls. \text{nnth } ls i = i) \wedge$
 $(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 i = \text{nnth } \sigma (\text{nnth } ls i)) \wedge g \sigma 1))$

using *PJ6help2* *PJ6help3* **by blast**
also have ... =
 $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge$
 $\text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$
 $(\forall i \leq \text{nlength } ls. \text{nnth } ls i = i) \wedge$
 $(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 i = \text{nnth } \sigma (i)) \wedge g \sigma 1))$

using *assms* **by auto** (*metis*, *metis*)
also have ... =
 $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge$
 $\text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$
 $(\forall i \leq \text{nlength } ls. \text{nnth } ls i = i) \wedge$
 $(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 i = \text{nnth } \sigma (i)) \wedge \sigma 1 = \sigma \wedge g \sigma))$
by auto
 $(\text{metis dual-order.refl enat.distinct}(2) \text{ enat-the-enat nellist-eq-nnth-eq nfinite-nlength-enat}$
 $\text{nnth-nlast})$
also have ... =
 $g \sigma$

using *PJ6help4* **assms by auto**
finally show $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge$
 $\text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$
 $(\forall i < \text{nlength } ls. \text{nlength } (\text{nsbn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) = \text{Suc } 0) \wedge$
 $(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 i = \text{nnth } \sigma (\text{nnth } ls i)) \wedge g \sigma 1))$
 $= g \sigma$
 \cdot
qed

lemma *OPJ6help5*:

assumes $\neg \text{nfinite } \sigma$

shows $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$
 $(\forall i < \text{nlength } ls. \text{nlength } (\text{nsbn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) = \text{Suc } 0) \wedge$
 $(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 i = \text{nnth } \sigma (\text{nnth } ls i)) \wedge g \sigma 1))$
 $= g \sigma$

proof –

have $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$
 $(\forall i < \text{nlength } ls. \text{nlength } (\text{nsbn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) = \text{Suc } 0) \wedge$

$(\exists \sigma 1. \text{ nlength } \sigma 1 = \text{ nlength } ls \wedge (\forall i \leq \text{ nlength } \sigma 1. \text{ nnth } \sigma 1 i = \text{ nnth } \sigma (\text{ nnth } ls i)) \wedge g \sigma 1))$
 $=$
 $(\exists ls. \text{ nid } ls \wedge \text{ nnth } ls 0 = 0 \wedge \neg \text{ nfinite } ls \wedge \neg \text{ nfinite } \sigma \wedge$
 $(\forall i < \text{ nlength } ls. \text{ nnth } ls (\text{ Suc } i) - \text{ nnth } ls i = \text{ Suc } 0) \wedge$
 $(\exists \sigma 1. \text{ nlength } \sigma 1 = \text{ nlength } ls \wedge (\forall i \leq \text{ nlength } \sigma 1. \text{ nnth } \sigma 1 i = \text{ nnth } \sigma (\text{ nnth } ls i)) \wedge g \sigma 1))$
using *OPJ6help1*[of - σ] **assms by auto**
also have ... =
 $(\exists ls. \text{ nid } ls \wedge \text{ nnth } ls 0 = 0 \wedge \neg \text{ nfinite } ls \wedge \neg \text{ nfinite } \sigma \wedge$
 $(\forall i \leq \text{ nlength } ls. \text{ nnth } ls i = i) \wedge$
 $(\exists \sigma 1. \text{ nlength } \sigma 1 = \text{ nlength } ls \wedge (\forall i \leq \text{ nlength } \sigma 1. \text{ nnth } \sigma 1 i = \text{ nnth } \sigma (\text{ nnth } ls i)) \wedge g \sigma 1))$

using *OPJ6help2*[of - σ] *OPJ6help3*[of - σ] **by blast**
also have ... =
 $(\exists ls. \text{ nid } ls \wedge \text{ nnth } ls 0 = 0 \wedge \neg \text{ nfinite } ls \wedge \neg \text{ nfinite } \sigma \wedge$
 $(\forall i \leq \text{ nlength } ls. \text{ nnth } ls i = i) \wedge$
 $(\exists \sigma 1. \text{ nlength } \sigma 1 = \text{ nlength } ls \wedge (\forall i \leq \text{ nlength } \sigma 1. \text{ nnth } \sigma 1 i = \text{ nnth } \sigma (i)) \wedge g \sigma 1))$

using *assms by auto* (*metis*, *metis*)
also have ... =
 $(\exists ls. \text{ nid } ls \wedge \text{ nnth } ls 0 = 0 \wedge \neg \text{ nfinite } ls \wedge \neg \text{ nfinite } \sigma \wedge$
 $(\forall i \leq \text{ nlength } ls. \text{ nnth } ls i = i) \wedge$
 $(\exists \sigma 1. \text{ nlength } \sigma 1 = \text{ nlength } ls \wedge (\forall i \leq \text{ nlength } \sigma 1. \text{ nnth } \sigma 1 i = \text{ nnth } \sigma (i)) \wedge \sigma 1 = \sigma \wedge g \sigma))$

by auto (*metis* *OPJ6help4* *nellist-eq-nnth-eq*)
also have ... =
 $g \sigma$

using *OPJ6help4* **assms by auto**
finally show $(\exists ls. \text{ nid } ls \wedge \text{ nnth } ls 0 = 0 \wedge \neg \text{ nfinite } ls \wedge \neg \text{ nfinite } \sigma \wedge$
 $(\forall i < \text{ nlength } ls. \text{ nlength } (\text{ nsubn } \sigma (\text{ nnth } ls i) (\text{ nnth } ls (\text{ Suc } i))) = \text{ Suc } 0) \wedge$
 $(\exists \sigma 1. \text{ nlength } \sigma 1 = \text{ nlength } ls \wedge (\forall i \leq \text{ nlength } \sigma 1. \text{ nnth } \sigma 1 i = \text{ nnth } \sigma (\text{ nnth } ls i)) \wedge g \sigma 1))$
 $= g \sigma$
 \cdot
qed

lemma *PJ6sem*:

$(\sigma \models \text{ skip } \text{ fproj } g = (g \wedge \text{ finite}))$

proof –

have 1: $(\sigma \models \text{ skip } \text{ fproj } g = (g \wedge \text{ finite})) =$

$((\exists l. \text{ nid } l \wedge \text{ nnth } l 0 = 0 \wedge \text{ nfinite } l \wedge \text{ nfinite } \sigma \wedge \text{ nlast } l = \text{ the-enat}(\text{ nlength } \sigma) \wedge$
 $(\forall i < \text{ nlength } l. (\text{ nsubn } \sigma (\text{ nnth } l i) (\text{ nnth } l (\text{ Suc } i))) \models \text{ skip}) \wedge g (\text{ pfilt } \sigma l)) =$
 $(g \sigma \wedge \text{ nfinite } \sigma))$

by (*simp add: fprojection-d-def powerinterval-def finite-defs*)

have 2: $(\exists ls. \text{ nid } ls \wedge \text{ nnth } ls 0 = 0 \wedge \text{ nfinite } ls \wedge \text{ nfinite } \sigma \wedge \text{ nlast } ls = \text{ the-enat}(\text{ nlength } \sigma) \wedge$
 $(\forall i < \text{ nlength } ls. (\text{ nsubn } \sigma (\text{ nnth } ls i) (\text{ nnth } ls (\text{ Suc } i))) \models \text{ skip}) \wedge g (\text{ pfilt } \sigma ls)) =$
 $(\exists \sigma 1. \text{ nid } ls \wedge \text{ nnth } ls 0 = 0 \wedge \text{ nfinite } ls \wedge \text{ nfinite } \sigma \wedge \text{ nlast } ls = \text{ the-enat}(\text{ nlength } \sigma) \wedge$
 $(\forall i < \text{ nlength } ls. (\text{ nsubn } \sigma (\text{ nnth } ls i) (\text{ nnth } ls (\text{ Suc } i))) \models \text{ skip}) \wedge$
 $\text{ nlength } \sigma 1 = \text{ nlength } ls \wedge$
 $(\forall i \leq \text{ nlength } \sigma 1. (\text{ nnth } \sigma 1 i) = (\text{ nnth } \sigma (\text{ nnth } ls i))) \wedge$
 $g \sigma 1)$

```

using pfilt-expand[of -  $\sigma$  ]
by auto (blast,metis)
have 3: ( $\exists$  ls  $\sigma 1$  .  $\text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge$ 
 $\text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$ 
 $(\forall i < \text{nlength } ls. (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) \models \text{skip}) \wedge$ 
 $\text{nlength } \sigma 1 = \text{nlength } ls \wedge$ 
 $(\forall i \leq \text{nlength } \sigma 1. (\text{nnth } \sigma 1 i) = (\text{nnth } \sigma (\text{nnth } ls i))) \wedge$ 
 $g \sigma 1) =$ 
 $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge \text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$ 
 $(\forall i < \text{nlength } ls. \text{nlength } (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) = \text{Suc } 0) \wedge$ 
 $(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 i = \text{nnth } \sigma (\text{nnth } ls i)) \wedge g \sigma 1))$ 
by (simp add: skip-defs)
have 4: ( $\exists$  ls.  $\text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \text{nfinite } ls \wedge \text{nfinite } \sigma \wedge \text{nlast } ls = \text{the-enat}(\text{nlength } \sigma) \wedge$ 
 $(\forall i < \text{nlength } ls. \text{nlength } (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) = \text{Suc } 0) \wedge$ 
 $(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 i = \text{nnth } \sigma (\text{nnth } ls i)) \wedge g \sigma 1)) =$ 
 $((g \sigma) \wedge \text{nfinite } \sigma)$ 
using PJ6help5[of  $\sigma$  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: ( $\sigma \models \text{skip } \text{oproj } g = (g \wedge \text{inf})) =$ 
 $((\exists ls. \text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$ 
 $(\forall i < \text{nlength } ls. (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) \models \text{skip}) \wedge g (\text{pfilt } \sigma ls)) =$ 
 $(g \sigma \wedge \neg \text{nfinite } \sigma))$ 

```

by (simp add: oprojection-d-def powerinterval-def infinite-defs)

```

have 2: ( $\exists$  ls.  $\text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$ 
 $(\forall i < \text{nlength } ls. (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) \models \text{skip}) \wedge g (\text{pfilt } \sigma ls)) =$ 
 $(\exists ls \sigma 1. \text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$ 
 $(\forall i < \text{nlength } ls. (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) \models \text{skip}) \wedge$ 
 $\text{nlength } \sigma 1 = \text{nlength } ls \wedge$ 
 $(\forall i \leq \text{nlength } \sigma 1. (\text{nnth } \sigma 1 i) = (\text{nnth } \sigma (\text{nnth } ls i))) \wedge$ 
 $g \sigma 1)$ 

```

using pfilt-expand[of - σ] **by** auto (blast,metis)

```

have 3: ( $\exists$  ls  $\sigma 1$  .  $\text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$ 
 $(\forall i < \text{nlength } ls. (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) \models \text{skip}) \wedge$ 
 $\text{nlength } \sigma 1 = \text{nlength } ls \wedge$ 
 $(\forall i \leq \text{nlength } \sigma 1. (\text{nnth } \sigma 1 i) = (\text{nnth } \sigma (\text{nnth } ls i))) \wedge$ 
 $g \sigma 1) =$ 
 $(\exists ls. \text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$ 
 $(\forall i < \text{nlength } ls. \text{nlength } (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) = \text{Suc } 0) \wedge$ 
 $(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 i = \text{nnth } \sigma (\text{nnth } ls i)) \wedge g \sigma 1))$ 

```

by (simp add: skip-defs)

```

have 4: ( $\exists$  ls.  $\text{nidx } ls \wedge \text{nnth } ls 0 = 0 \wedge \neg \text{nfinite } ls \wedge \neg \text{nfinite } \sigma \wedge$ 
 $(\forall i < \text{nlength } ls. \text{nlength } (\text{nsubn } \sigma (\text{nnth } ls i) (\text{nnth } ls (\text{Suc } i))) = \text{Suc } 0) \wedge$ 
 $(\exists \sigma 1. \text{nlength } \sigma 1 = \text{nlength } ls \wedge (\forall i \leq \text{nlength } \sigma 1. \text{nnth } \sigma 1 i = \text{nnth } \sigma (\text{nnth } ls i)) \wedge g \sigma 1)) =$ 
 $((g \sigma) \wedge \neg \text{nfinite } \sigma)$ 

```

```

    using OPJ6help5[of  $\sigma$   $g$ ] by auto
  from 1 2 3 4 show ?thesis
  by simp
qed

```

10.3.7 PJ7

```

lemma PJemptyImp:
  assumes 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 ( $\sigma \models f \text{ fproj } (g \text{ fproj } h) = (f \text{ fproj } g) \text{ fproj } h$ )
  proof -
    have 1: ( $\sigma \models f \text{ fproj } (g \text{ fproj } h) = (g \text{ fproj } h)$ )
      using PJemptyImp assms by blast
    have 2: ( $\sigma \models (g \text{ fproj } h) = h$ )
      using PJemptyImp assms by blast
    have 3: ( $\sigma \models (f \text{ fproj } g) \text{ fproj } h = h$ )
      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  $ls = \text{the-enat}(\text{nlength } \sigma)$ 
    ( $\forall i < \text{nlength } ls. (\text{nsubn } \sigma (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) \models f \text{ fproj } g$ )
    nlength  $\sigma > 0$ 
  shows nidx ( $\text{nfusecat } ( \text{lcpl } f \ g \ \sigma \ ls))) \wedge \text{nnth } (\text{nfusecat } ( \text{lcpl } f \ g \ \sigma \ ls))) \ 0 = 0$ 
  proof -
    have 0: nlength  $\sigma > 0 \longrightarrow \text{nlength } ls > 0$ 
      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 ( $\text{lcpl } f \ g \ \sigma \ ls$ )
      using assms lcpl-nfusecat-nlastnfirst[of  $ls \ \sigma \ f \ g$ ]
      by (metis 0)
    have 1: nfirst ( $\text{nfusecat } ( \text{lcpl } f \ g \ \sigma \ ls))) = 0$ 

```

using *assms 0 01 02 lcppl-nfirst*[of *ls σ f g*] **by** (*metis nfirst-nfusecat-nfirst*)
from 1 0 **show** ?thesis **using** *assms lcppl-nfusecat-nidx*[of *ls σ 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: $nlastnfirst\ (lcppl\ f\ g\ \sigma\ ls)$
using *assms lcppl-nfusecat-nlastnfirst-alt*[of *ls σ f g*]
by (*metis*)
have 1: $nfirst\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)) = 0$
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*
 $nnth\ ls\ 0 = 0$
 $nfinite\ ls$
 $nfinite\ \sigma$
 $nlast\ ls = the-enat(nlength\ \sigma)$
 $(\forall\ i < nlength\ ls. (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$
 $nlength\ \sigma > 0$
shows $powerinterval\ f\ \sigma\ (nfusecat\ (\ lcppl\ f\ g\ \sigma\ ls)))$
proof –
have 1: $(\forall\ i < nlength\ ls.$
 $powerinterval\ f\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
 $(cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$
using *assms cppl-fprojection* **by** *blast*
have 2: $\forall\ ia < nlength\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$
 $f\ (nsubn\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$
 $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ ia)$
 $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ (Suc\ ia))$

```

    )
  using 1 by (simp add: powerinterval-def)
have 3:  $\forall i < \text{nlength } ls.$ 
   $\forall ia < \text{nlength } (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ )).$ 
   $(nsubn\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ )$ 
     $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ))\ ia)$ 
     $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ))\ (Suc\ ia))$ 
   $) =$ 
   $(nsubn\ \sigma$ 
     $(nnth\ (nmap\ (\lambda x. x + (nnth\ ls\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ )))\ ia)$ 
     $(nnth\ (nmap\ (\lambda x. x + (nnth\ ls\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ )))\ (Suc\ ia))$ 
   $)$ 
proof auto
  fix i
  fix ia
  assume a0:  $\text{enat } i < \text{nlength } ls$ 
  assume a1:  $\text{enat } ia < \text{nlength } (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$ 
  show  $nsubn\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$ 
     $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ ia)$ 
     $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ (Suc\ ia)) =$ 
     $nsubn\ \sigma\ (nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ ia + nnth\ ls\ i)$ 
     $(nnth\ (nmap\ (\lambda x. x + nnth\ ls\ i)\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ (Suc\ ia))$ 
proof -
  have 300:  $0 < \text{nlength } \sigma$ 
  using assms(7) by auto
  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(6))
  have 303:  $\text{the-enat } (\text{nlength } (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) = (nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i)$ 
  by (simp add: PJ6help1 a0 assms(1) assms(2) assms(3) assms(4) assms(5))
  have 304:  $\text{nlast } (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) = (nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i)$ 
  using a0 a1 cppl-fprojection[of f g (nsubn σ (nnth ls i) (nnth ls (Suc i)))]
  using 302 303 by presburger
  have 305:  $(Suc\ ia) \leq \text{nlength } (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$ 
  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:  $\text{nlast } (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) =$ 
     $nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$ 
     $(\text{the-enat}(\text{nlength } (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))))$ 
  using 302 cppl-fprojection nnth-nlast by auto
  have 308:  $nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ (Suc\ ia) \leq$ 
     $\text{nlast } (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$ 
  by (metis 302 305 cppl-fprojection nidx-all-le-nlast)
  have 310:  $\text{enat } (nnth\ ls\ (Suc\ i)) \leq \text{nlength } \sigma$ 
  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: $\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)))) \ (\text{Suc } ia) \leq \text{nnth } \text{ls } (\text{Suc } i) - \text{nnth } \text{ls } i$
using *assms cppl-bounds*[of $(\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)) \ \sigma \ f \ g \ \text{Suc } ia$]
PJ6help1[of $\text{ls } \sigma$] *a0 a1 cppl-fprojection*[of $f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)))$]
by (*metis 303 308*)
have 311: $\text{nsbn } (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)))$
 $(\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)))) \ ia)$
 $(\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)))) \ (\text{Suc } ia)) =$
 $\text{nsbn } \sigma \ (\text{nnth } \text{ls } i + \text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)))) \ ia)$
 $(\text{nnth } \text{ls } i + \text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)))) \ (\text{Suc } ia))$
using *nsbn-nsbn-1*[of $(\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)))) \ ia)$
 $(\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)))) \ (\text{Suc } ia)) \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)) \ \sigma$]
using 30 301 306 310 *order-less-imp-le* **by** *blast*
have 312: $(\text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)))) \ ia + \text{nnth } \text{ls } i =$
 $(\text{nnth } \text{ls } i + \text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)))) \ ia)$
by *simp*
have 313: $(\text{nnth } \text{ls } i + \text{nnth } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)))) \ (\text{Suc } ia) =$
 $(\text{nnth } (\text{nmap } (\lambda x. x + \text{nnth } \text{ls } i) \ (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)))) \ (\text{Suc } ia))$
using 305 **by** *force*
show *?thesis*
using 311 312 313 **by** *presburger*
qed
qed
have 4: $\forall i < \text{nlength } \text{ls}.$
 $\forall ia < \text{nlength } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i))))).$
 $f \ (\text{nsbn } \sigma \ (\text{nnth } (\text{nmap } (\lambda x. x + (\text{nnth } \text{ls } i)) \ (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)))) \ ia)$
 $(\text{nnth } (\text{nmap } (\lambda x. x + (\text{nnth } \text{ls } i)) \ (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } i) \ (\text{nnth } \text{ls } (\text{Suc } i)))) \ (\text{Suc } ia))$
 $)$
using 2 3 **by** *auto*
have 5: $\text{nlength}(\text{lcpl } f \ g \ \sigma \ \text{ls}) = \text{nlength } \text{ls} - 1$
using *assms lcpl-nlength lcpl-nlength-zero*
by (*metis epred-0 epred-conv-minus i0-less*)
have 6: $\text{nlength } \sigma > 0 \longrightarrow \text{nlength } \text{ls} > 0$
using *assms*
by (*metis enat.distinct(2) enat-the-enat nfinite-nlength-enat nnth-nlast the-enat-0 zero-less-iff-neq-zero*)
have 7: $\forall i < \text{nlength } \text{ls}.$
 $\forall ia < \text{nlength } ((\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ i)).$
 $f \ (\text{nsbn } \sigma \ (\text{nnth } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ i) \ ia)$
 $(\text{nnth } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ i) \ (\text{Suc } ia))$
 $)$
using *assms 4 lcpl-nnth* **by** (*metis nlength-nmap*)
have 8: $\text{nlength } \sigma > 0 \longrightarrow$
 $(\forall i \leq \text{nlength}(\text{lcpl } f \ g \ \sigma \ \text{ls}).$
 $\forall ia < \text{nlength } ((\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ i)).$
 $f \ (\text{nsbn } \sigma \ (\text{nnth } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ i) \ ia)$
 $(\text{nnth } (\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ i) \ (\text{Suc } ia))$
 $))$
using 5 6 7 *assms* **by** (*metis co.enat.exhaust-sel i0-less illess-Suc-eq lcpl-nlength*)
have 9: $\text{nlength } \sigma > 0 \longrightarrow$
 $\text{powerinterval } f \ \sigma \ (\text{nfusecat } ((\text{lcpl } f \ g \ \sigma \ \text{ls}))) =$
 $(\forall i < \text{nlength } (\text{nfusecat } (\text{lcpl } f \ g \ \sigma \ \text{ls})).$

$f (nsubn \sigma (nnth (nfusecat (lcppl f g \sigma ls)) i) (nnth (nfusecat (lcppl f g \sigma ls)) (Suc i)))$
by (*simp add: powerinterval-def*)
have 10: $nlength \sigma > 0 \longrightarrow nlastnfirst (lcppl f g \sigma ls)$
using 6 *assms lcppl-nfusecat-nlastnfirst*
by (*metis nfinite-conv-nlength-enat*)
have 11: $nlength \sigma > 0 \longrightarrow$
 $(\forall j \leq nlength (lcppl f g \sigma ls). nlength (nnth (lcppl f g \sigma ls) j) > 0)$

using *assms*
by (*metis enat.distinct(2) enat-the-enat lcppl-nlength-all-gr-zero nfinite-conv-nlength-enat*)
have 110: *all-gr-zero* (*lcppl f g \sigma ls*)
using 11 *all-gr-zero-nnth-a assms(7)* **by** *blast*
have 111: *all-nfinite* (*lcppl f g \sigma ls*)
using *all-nfinite-nnth-a assms lcppl-nlength-all-nfinite* **by** *blast*
have 12: $nlength \sigma > 0 \longrightarrow$
 $(\forall i \leq nlength (lcppl f g \sigma ls).$
 $(\forall ia < nlength ((nnth (lcppl f g \sigma ls) i)).$
 $f (nsubn \sigma (nnth (nnth (lcppl f g \sigma ls) i) ia)$
 $(nnth (nnth (lcppl f g \sigma ls) i) (Suc ia))$
 $))) =$
 $(\forall j < nlength (nfusecat (lcppl f g \sigma ls)).$
 $f (nsubn \sigma (nnth (nfusecat (lcppl f g \sigma ls)) j)$
 $(nnth (nfusecat (lcppl f g \sigma ls)) (Suc j))$
 $))$

using *nfusecat-split-nsubn[of (lcppl f g \sigma ls) f \sigma]* 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*
 $nnth ls 0 = 0$
 $\neg nfinite ls$
 $\neg nfinite \sigma$
 $(\forall i < nlength ls. (nsubn \sigma (nnth ls i) (nnth ls (Suc i))) \models f fproj g)$
shows *powerinterval f \sigma (nfusecat ((lcppl f g \sigma ls)))*
proof –
have 1: $(\forall i < nlength ls.$
 $powerinterval f (nsubn \sigma (nnth ls i) (nnth ls (Suc i)))$
 $(cppl f g (nsubn \sigma (nnth ls i) (nnth ls (Suc i)))))$
using *assms cppl-fprojection* **by** *blast*
have 2: $\forall i < nlength ls.$
 $\forall ia < nlength (cppl f g (nsubn \sigma (nnth ls i) (nnth ls (Suc i)))).$
 $f (nsubn (nsubn \sigma (nnth ls i) (nnth ls (Suc i)))$
 $(nnth (cppl f g (nsubn \sigma (nnth ls i) (nnth ls (Suc i)))) ia)$
 $(nnth (cppl f g (nsubn \sigma (nnth ls i) (nnth ls (Suc i)))) (Suc ia))$
 $)$

```

using 1 by (simp add: powerinterval-def)
have 3:  $\forall i < \text{nlength } ls.$ 
   $\forall ia < \text{nlength } (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ )).$ 
   $(nsubn\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ )$ 
     $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ))\ ia)$ 
     $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ ))\ (Suc\ ia))$ 
   $) =$ 
   $(nsubn\ \sigma$ 
     $(nnth\ (nmap\ (\lambda x. x + (nnth\ ls\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ )))\ ia)$ 
     $(nnth\ (nmap\ (\lambda x. x + (nnth\ ls\ i))\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))\ )))\ (Suc\ ia))$ 
   $)$ 
proof auto
fix i
fix ia
assume a0:  $\text{enat } i < \text{nlength } ls$ 
assume a1:  $\text{enat } ia < \text{nlength } (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$ 
show  $nsubn\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))$ 
   $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ ia)$ 
   $(nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ (Suc\ ia)) =$ 
   $nsubn\ \sigma\ (nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))\ ia + nnth\ ls\ i)$ 
   $(nnth\ (nmap\ (\lambda x. x + nnth\ ls\ i)\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))))\ (Suc\ ia))$ 
proof –
have 300:  $0 < \text{nlength } \sigma$ 
  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:  $\text{the-enat } (\text{nlength } (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) = (nnth\ ls\ (Suc\ i)) - (nnth\ ls\ i)$ 
  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:  $(Suc\ ia) \leq \text{nlength } (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$ 
  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:  $nlast\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) =$ 
   $nnth\ (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))))$ 
   $(\text{the-enat}(\text{nlength } (cppl\ f\ g\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))))$ 
  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:  $\text{enat } (nnth\ ls\ (Suc\ i)) \leq \text{nlength } \sigma$ 
  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)) σ f g Suc ia]

```

```

    PJ6help1[of ls  $\sigma$  ] a0 a1 cppl-fprojection[of f g (nsubn  $\sigma$  (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  $\sigma$  (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:  $(\forall j \leq \text{nlength } (\text{lcppl } f \ g \ \sigma \ ls). \text{nlength}(\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ j) > 0)$ 
  using assms lcppl-nlength-all-gr-zero-alt by blast
have 110: all-gr-zero (lcppl f g σ ls)
  using 11 all-gr-zero-nnth-a assms by blast
have 111: all-nfinite (lcppl f g σ ls)
  using all-nfinite-nnth-a assms lcppl-nlength-all-nfinite-alt by blast
have 12:  $(\forall i \leq \text{nlength}(\text{lcppl } f \ g \ \sigma \ ls).$ 
   $(\forall ia < \text{nlength } ((\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i)).$ 
   $f \ (\text{nsubn } \sigma \ (\text{nnth } (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i) \ ia)$ 
   $(\text{nnth } (\text{nnth } (\text{lcppl } f \ g \ \sigma \ ls) \ i) \ (\text{Suc } ia))$ 
   $))) =$ 
   $(\forall j < \text{nlength } (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls)).$ 
   $f \ (\text{nsubn } \sigma \ (\text{nnth } (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls)) \ j)$ 
   $(\text{nnth } (\text{nfusecat } (\text{lcppl } f \ g \ \sigma \ ls)) \ (\text{Suc } j))$ 
   $))$ 

  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)
   $(\forall i < \text{nlength } ls. (\text{nsubn } \sigma \ (\text{nnth } ls \ i) \ (\text{nnth } ls \ (\text{Suc } i)) \ ) \models f \ \text{fproj } g)$ 
  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))

```

$= (\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ (\text{the-enat}(\text{nlength } \text{ls}) - 1))$
using *assms*
by (*metis 0 epred-enat lcpl-nfinite lcpl-nlength nfinite-nlength-enat nnth-nlast the-enat.simps*)
have 3: $\text{nlength } \sigma > 0 \longrightarrow$
 $(\text{nnth } (\text{lcpl } f \ g \ \sigma \ \text{ls}) \ (\text{the-enat}(\text{nlength } \text{ls}) - 1)) =$
 $(\text{nmap } (\lambda x. x + (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))$
 $(\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))$
 $(\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))) \))$

using *assms 0 lcpl-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: $\text{nlength } \sigma > 0 \longrightarrow$
 $\text{nlast} ((\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1))$
 $(\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))) \)) =$
 $\text{the-enat}(\text{nlength } ((\text{nsbn } \sigma \ (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1))$
 $(\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))) \))$
using *0 assms cppl-fprojection[of f g*
 $(\text{nsbn } \sigma \ (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)) \ (\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))))]$
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: $\text{nlength } \sigma > 0 \longrightarrow$
 $\text{the-enat}(\text{nlength } ((\text{nsbn } \sigma \ (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1))$
 $(\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: $\text{nfinite} \ (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1))$
 $(\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))))$
by (*metis 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: $\text{nlength } \sigma > 0 \longrightarrow$
 $\text{nlast} ((\text{nmap } (\lambda x. x + (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))$
 $(\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))$
 $(\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))) \)) =$
 $(\lambda x. x + (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))$
 $(\text{nlast } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1))$
 $(\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))) \))$
using *nlast-nmap[of (cppl f g (nsbn 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: $\text{nlength } \sigma > 0 \longrightarrow$
 $(\lambda x. x + (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1)))$
 $(\text{nlast } (\text{cppl } f \ g \ (\text{nsbn } \sigma \ (\text{nnth } \text{ls } (\text{the-enat}(\text{nlength } \text{ls}) - 1))$
 $(\text{nnth } \text{ls } (\text{Suc } (\text{the-enat}(\text{nlength } \text{ls}) - 1))) \)) =$

```

      (λx. x + (nnth ls (the-enat(nlength ls) - 1)))
      ((nnth ls (Suc (the-enat(nlength ls) - 1))) - (nnth ls (the-enat(nlength ls) - 1)))
    using 4 5 by auto
  have 8: nlength σ > 0 ⟶
    (λx. x + (nnth ls (the-enat(nlength ls) - 1)))
    ((nnth ls (Suc (the-enat(nlength ls) - 1))) - (nnth ls (the-enat(nlength ls) - 1))) =
    ((nnth ls (Suc (the-enat(nlength ls) - 1))) -
    (nnth ls (the-enat(nlength ls) - 1))) + (nnth ls (the-enat(nlength ls) - 1))
  by (simp add: shift-def)
  have 9: nlength σ > 0 ⟶
    ((nnth ls (Suc (the-enat(nlength ls) - 1))) -
    (nnth ls (the-enat(nlength ls) - 1))) + (nnth ls (the-enat(nlength ls) - 1))
    = (nnth ls (Suc (the-enat(nlength 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: nlength σ > 0 ⟶ (nnth ls (Suc (the-enat(nlength ls) - 1))) = nlength σ
  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
    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 ((pfilt σ (nfusecat (lcppl f g σ ls))) ⊨ g fproj h)
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: all-gr-zero (lcppl f g σ ls)
  using all-gr-zero-nnth-a assms lcppl-nlength-all-gr-zero by blast
  have 01: all-nfinite (lcppl f g σ ls)
  using all-nfinite-nnth-a assms lcppl-nlength-all-nfinite by blast
  have 02: nnth (addzero (lsum (lcppl f g σ ls) 0)) 0 = 0
  using 0 assms
  by (metis addzero-def lcppl-lsum-nlength less-numeral-extra(3) nnth-0)
  have 1: nlength σ > 0 ⟶ nidx (addzero (lsum (lcppl f g σ ls) 0)) ∧ nnth (addzero (lsum (lcppl f g σ ls)
    0)) 0 = 0
  using assms lsum-addzero-nidx[of (lcppl f g σ ls)] lcppl-nlength-all-gr-zero[of ls σ f g]
  using 00 01 02 by blast

```

have 2: $nlength\ \sigma > 0 \longrightarrow$
 $nlast\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) =$
 $the-enat(nlength\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls))))$
using *assms lcppl-lsum-nlength[of ls σ f g] lcppl-nfusecat-pfilt-nlength[of ls σ f g]*
by *fastforce*
have 3: $nlength\ \sigma > 0 \longrightarrow$
 $powerinterval\ g\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))$
by (*simp add: assms lcppl-nfusecat-pfilt-powerinterval*)
have 4: $nlength\ \sigma > 0 \longrightarrow$
 $h\ (pfilt\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)))$
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*
 $nnth\ ls\ 0 = 0$
 $\neg nfinite\ ls$
 $\neg nfinite\ \sigma$
 $(\forall\ i < nlength\ ls.\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g)$
 $h\ (pfilt\ \sigma\ ls)$
shows $((pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls))) \models g\ oproj\ h)$
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: $nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ 0 = 0$
using *assms*
by (*metis addzero-def lcppl-nfinite nfinite-lsum-conv-b nlength-eq-enat-nfiniteD nnth-0 zero-enat-def*)
have 1: $nidx\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)) \wedge nnth\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))\ 0 = 0$
using *assms lsum-addzero-nidx 00 01 02* **by** *blast*
have 3: $powerinterval\ g\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))$
by (*simp add: assms lcppl-nfusecat-pfilt-powerinterval-alt*)
have 4: $h\ (pfilt\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0)))$
by (*simp add: assms lcppl-pfilt-nfusecat-lsum-alt pfilt-nmap-pfilt*)
have 5: $\neg nfinite\ (addzero\ (lsum\ (lcppl\ f\ g\ \sigma\ ls)\ 0))$
using *assms*
by (*simp add: lcppl-lsum-nlength-alt nfinite-conv-nlength-enat*)
have 6: $\neg nfinite\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls))$
using *assms 00 01 lcppl-nfinite lcppl-nfusecat-nlastnfirst-alt nfusecat-nfinite* **by** *blast*
have 7: $\neg nfinite\ (pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)))$
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 $nlength\ \sigma > 0$
 $(\sigma \models (f\ fproj\ g)\ fproj\ h)$
shows $(\sigma \models f\ fproj\ (g\ fproj\ h))$
proof –
have 1: $nlength\ \sigma > 0$
using *assms* **by** *auto*
have 2: $(\exists\ ls.\ nidx\ ls \wedge\ nnth\ ls\ 0 = 0 \wedge\ nfinite\ ls \wedge\ nfinite\ \sigma \wedge\ nlast\ ls = the-enat(nlength\ \sigma) \wedge$
 $powerinterval\ (LIFT(f\ fproj\ g))\ \sigma\ ls \wedge$
 $h\ (pfilt\ \sigma\ ls))$
using *assms* **using** *cppl-fprojection*[of $LIFT(f\ fproj\ g)\ h\ \sigma$] **by** *blast*
obtain ls **where** 3: $nidx\ ls \wedge\ nnth\ ls\ 0 = 0 \wedge\ nfinite\ ls \wedge\ nfinite\ \sigma \wedge$
 $nlast\ ls = the-enat(nlength\ \sigma) \wedge$
 $powerinterval\ (LIFT(f\ fproj\ g))\ \sigma\ ls \wedge$
 $h\ (pfilt\ \sigma\ ls)$
using 2 **by** *blast*
have 4: $nidx\ ls \wedge\ nnth\ ls\ 0 = 0 \wedge\ nfinite\ ls \wedge\ nfinite\ \sigma \wedge\ nlast\ ls = the-enat(nlength\ \sigma) \wedge$
 $(\forall\ i < nlength\ ls.\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g) \wedge$
 $h\ (pfilt\ \sigma\ ls)$
using 3 **by** (*simp* *add: powerinterval-def*)
have 6: $nlength\ ls > 0$
using 1 4
by (*metis* *PJ6help4* *nnth-nlast* *the-enat-0* *zero-less-iff-neq-zero*)
have 7: $nidx\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)) \wedge\ nnth\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls))\ 0 = 0$
by (*metis* (*no-types*, *lifting*) 1 4 *PJ7helpchain1a-help-1*)
have 8: $powerinterval\ f\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls))$
using 4 6 *PJ7helpchain1a-help-2* *assms* **by** *auto*
have 9: $nlast\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls)) = the-enat(nlength\ \sigma)$
by (*metis* 1 4 *PJ7helpchain1a-help-3* *the-enat.simps*)
have 10: $((pfilt\ \sigma\ (nfusecat\ (lcppl\ f\ g\ \sigma\ ls))) \models g\ fproj\ h)$
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 $(\sigma \models (f\ fproj\ g)\ oproj\ h)$

shows $(\sigma \models f\ oproj\ (g\ oproj\ h))$

proof –

have 2: $(\exists\ ls.\ nidx\ ls \wedge\ nnth\ ls\ 0 = 0 \wedge\ \neg nfinite\ ls \wedge\ \neg nfinite\ \sigma \wedge$
 $powerinterval\ (LIFT(f\ fproj\ g))\ \sigma\ ls \wedge$
 $h\ (pfilt\ \sigma\ ls))$

using *assms* **using** *cppl-oprojection*[of $LIFT(f\ fproj\ g)\ h\ \sigma$] **by** *blast*

obtain ls **where** 3: $nidx\ ls \wedge\ nnth\ ls\ 0 = 0 \wedge\ \neg nfinite\ ls \wedge\ \neg nfinite\ \sigma \wedge$
 $powerinterval\ (LIFT(f\ fproj\ g))\ \sigma\ ls \wedge$
 $h\ (pfilt\ \sigma\ ls)$

using 2 **by** *blast*

have 4: $nidx\ ls \wedge\ nnth\ ls\ 0 = 0 \wedge\ \neg nfinite\ ls \wedge\ \neg nfinite\ \sigma \wedge$
 $(\forall\ i < nlength\ ls.\ (nsubn\ \sigma\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \models f\ fproj\ g) \wedge$
 $h\ (pfilt\ \sigma\ ls)$

```

using 3 by (simp add: powerinterval-def)
have 6: nlength ls > 0
  by (metis 4 gr-zeroI nlength-eq-enat-nfiniteD zero-enat-def)
have 7: nidx (nfusecat (lcppl f g σ ls)) ∧ nnth (nfusecat (lcppl f g σ ls)) 0 = 0
  using 4 PJ7helpchain1a-help-1-alt
  by (metis 6 OPJ6help4)
have 8: powerinterval f σ (nfusecat (lcppl f g σ ls))
  using 4 PJ7helpchain1a-help-2-alt by blast
have 10: ((pfilt σ (nfusecat (lcppl f g σ ls))) ⊨ g 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 σ > 0
  (σ ⊨ f fproj (g fproj h))
shows (σ ⊨ (f fproj g) fproj h)
proof –
  have 1: nlength σ > 0
  using assms by auto
  have 2: (∃ ls. nidx ls ∧ nnth ls 0 = 0 ∧ nfinite ls ∧ nfinite σ ∧ nlast ls = the-enat(nlength σ) ∧
    powerinterval f σ ls ∧
    ((pfilt σ ls) ⊨ g fproj h) )
  using assms cppl-fprojection by blast
  obtain ls where 3: nidx ls ∧ nnth ls 0 = 0 ∧ nfinite ls ∧ nfinite σ ∧ nlast ls = the-enat(nlength σ) ∧
    powerinterval f σ ls ∧
    ((pfilt σ ls) ⊨ g fproj h)
  using 2 by blast
  have 4: (∃ lsa. nidx lsa ∧ nnth lsa 0 = 0 ∧ nfinite lsa ∧ nfinite (pfilt σ ls) ∧
    nlast lsa = the-enat(nlength(pfilt σ ls)) ∧
    powerinterval g (pfilt σ ls) lsa ∧
    ((pfilt (pfilt σ ls) lsa) ⊨ h))
  using 3 using cppl-fprojection by blast
  obtain lsa where 5: nidx lsa ∧ nnth lsa 0 = 0 ∧ nfinite lsa ∧ nfinite (pfilt σ ls) ∧
    nlast lsa = the-enat(nlength(pfilt σ ls)) ∧
    powerinterval g (pfilt σ ls) lsa ∧
    ((pfilt (pfilt σ ls) lsa) ⊨ 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: ( $\forall i < nlength\ lsa. (nnth\ (pfilt\ ls\ lsa)\ i) = (nnth\ ls\ (nnth\ lsa\ i))$ )
  by (simp add: pfilt-nmap)
have 114: ( $\forall i < nlength\ lsa. (nnth\ (pfilt\ ls\ lsa)\ (Suc\ i)) = (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))$ )
  by (metis 112 eSuc-enat ileI1 pfilt-nnth)
have 1141:  $\bigwedge j. j \leq nlength\ lsa \longrightarrow nnth\ lsa\ j \leq nlength\ ls$ 
  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: ( $\forall i < nlength\ lsa. (nnth\ ls\ (nnth\ lsa\ i)) < (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))$ )
  by (metis 1141 3 5 eSuc-enat ileI1 less-imp-Suc-add nidx-expand nidx-less)
have 12: nidx (pfilt ls lsa)  $\wedge$  nnth (pfilt ls lsa) 0 = 0
  by (simp add: 111 112 115 Suc-ile-eq nidx-expand pfilt-nnth)
have 2021: ( $\forall i < nlength\ lsa. nfinite\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$ )
  by (simp add: nsubn-def1)
have 2022:  $\bigwedge i. i < nlength\ lsa \implies (nnth\ ls\ (nnth\ lsa\ i)) \leq nlength\ \sigma$ 
  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:  $\bigwedge j. j \leq nlength\ lsa \longrightarrow nnth\ lsa\ j \leq nlength\ ls$ 
  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: ( $\forall i < nlength\ lsa. nlength\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))) =$ 
  ( $nnth\ ls\ (nnth\ lsa\ (Suc\ i)) - nnth\ ls\ (nnth\ lsa\ i)$ )
  by (metis 112 113 114 12 3 5 9 PJ6help1 le-add1 le-add-same-cancel1
    nfinite-conv-nlength-enat)
have 2041: ( $\forall i < nlength\ lsa. (nnth\ lsa\ (Suc\ i)) \leq nlength\ ls$ )
  by (metis 2023 eSuc-enat ileI1)
have 204: ( $\forall i < nlength\ lsa. nlength\ (nmap\ (\lambda x. x - (nnth\ ls\ (nnth\ lsa\ i)))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i)))) =$ 
  ( $nnth\ lsa\ (Suc\ i) - nnth\ lsa\ i$ )
  by (simp add: 3 5 PJ6help1 pfilt-nlength)
have 205: ( $\forall i < nlength\ lsa. (\forall j \leq (nnth\ lsa\ (Suc\ i)) - (nnth\ lsa\ i). (nnth\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i)))\ j) =$ 
  ( $nnth\ ls\ ((nnth\ lsa\ i) + j)$ )
  )
  )
  using 2041 5 by (simp add: nsubn-def1 ntaken-nnth)
have 2060: ( $\forall i < nlength\ lsa. (nnth\ lsa\ i) \leq (nnth\ lsa\ (Suc\ i))$ )
  using 5 by (metis eSuc-enat ileI1 le-add2 nidx-less-eq plus-1-eq-Suc)
have 206: ( $\forall i < nlength\ lsa. (\forall j \leq (nnth\ lsa\ (Suc\ i)) - (nnth\ lsa\ i). (nnth\ ls\ (nnth\ lsa\ i)) \leq (nnth\ ls\ ((nnth\ lsa\ i) + j))$ )
  )

```

```

    )
  )
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: (∀ i < nlength lsa.
  (∀ j ≤ (nnth lsa (Suc i)) - (nnth lsa i).
    (nnth (nmap (λx. x - (nnth ls (nnth lsa i)))
      (nsubn 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)))
    (nsubn 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)))
    (nsubn 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: nsubn-def1 ntaken-ndropn-nlast)
have 210: (∀ i < nlength lsa.
  nidx (nmap (λx. x - (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i))) ) ∧
  nnth (nmap (λx. x - (nnth ls (nnth lsa i))) (nsubn ls (nnth lsa i) (nnth lsa (Suc i))) ) 0 = 0
  )
using 3 5 7 nidx-expand nidx-shiftn nidx-nsubn 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. (nsubn 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. (nsubn 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 (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ∧
        nlast ls1 = the-enat(nlength (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ) ∧
        powerinterval f (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ls1 ∧
        ((pfilt (nsubn 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 (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ∧
    nlast ls1 = the-enat(nlength (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ) ∧
    powerinterval f (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))) ) ls1 ∧
    ((pfilt (nsubn 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 $\exists ls1. nidx\ ls1 \wedge$

$nnth\ ls1\ 0 = 0 \wedge$

$nfinite\ ls1 \wedge$

$nfinite\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))) \wedge$

$nlast\ ls1 = the-enat\ (nlength\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i)))) \wedge$

$powerinterval\ f\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))\ ls1 \wedge$

$g\ (pfilt\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))\ ls1)$

proof –

have $fg1: nnth\ (nmap\ (\lambda x. x - (nnth\ ls\ (nnth\ lsa\ i)))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))))\ 0 = 0$

using 210 *a* **by** *blast*

have $fg2: nidx\ (nmap\ (\lambda x. x - (nnth\ ls\ (nnth\ lsa\ i)))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))))$

using 210 *a* **by** *blast*

have $fg3: nfinite\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$

using 2021 *a* **by** *blast*

have $fg4: nlast\ (nmap\ (\lambda x. x - (nnth\ ls\ (nnth\ lsa\ i)))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i)))) =$
 $the-enat\ (nlength\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$

using 203 209 *a* *the-enat.simps* **by** *presburger*

have $fg5: powerinterval\ f\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$
 $(nmap\ (\lambda x. x - (nnth\ ls\ (nnth\ lsa\ i)))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))))$

proof (*auto simp add: powerinterval-def*)

fix $ia::nat$

assume $aa: enat\ ia < nlength\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i)))$

show $f\ (nsubn\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$

$(nnth\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i)))\ ia - nnth\ ls\ (nnth\ lsa\ i))$

$(nnth\ (nmap\ (\lambda x. x - nnth\ ls\ (nnth\ lsa\ i))\ (nsubn\ ls\ (nnth\ lsa\ i)\ (nnth\ lsa\ (Suc\ i))))$

$(Suc\ ia)))$

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 $f2: nsubn\ (nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i))\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))))$

$(nnth\ ls\ (nnth\ lsa\ i + ia) - nnth\ ls\ (nnth\ lsa\ i))$

$(nnth\ ls\ (nnth\ lsa\ i + Suc\ ia) - nnth\ ls\ (nnth\ lsa\ i)) =$

$nsubn\ \sigma\ (nnth\ ls\ (nnth\ lsa\ i + ia))\ (nnth\ ls\ (nnth\ lsa\ i + Suc\ ia))$

proof –

have $f3: (nnth\ ls\ (nnth\ lsa\ i + ia) - nnth\ ls\ (nnth\ lsa\ i)) \leq$

$nnth\ ls\ (nnth\ lsa\ i + Suc\ ia) - nnth\ ls\ (nnth\ lsa\ i)$

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 $f4: nnth\ ls\ (nnth\ lsa\ i) \leq nnth\ ls\ (nnth\ lsa\ (Suc\ i))$

using 115 *a* *dual-order.order-iff-strict* **by** *blast*

have $f5: enat\ (nnth\ ls\ (nnth\ lsa\ (Suc\ i))) \leq nlength\ \sigma$

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 $f6: nnth\ ls\ (nnth\ lsa\ i + Suc\ ia) - nnth\ ls\ (nnth\ lsa\ i) \leq$

$nnth\ ls\ (nnth\ lsa\ (Suc\ i)) - nnth\ ls\ (nnth\ lsa\ i)$

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)))  $\sigma$  ]
by (metis 206 Suc-leI a f1 f3 f4 f5 f6 le-add-diff-inverse order-less-imp-le)
qed
have f7: (nnth (nmap ( $\lambda x. x - \text{nnth ls (nnth lsa i)}$ )
      (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  $\sigma$  (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  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
      (nmap ( $\lambda x. x - \text{nnth ls (nnth lsa i)}$ ) (nsubn ls (nnth lsa i) (nnth lsa (Suc i))) ))
proof -
have g1: (pfilt (nsubn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
      (nmap ( $\lambda x. x - \text{nnth ls (nnth lsa i)}$ ) (nsubn ls (nnth lsa i) (nnth lsa (Suc i))) )) =
      nmap (nnth (nsubn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
      (nmap ( $\lambda x. x - \text{nnth ls (nnth lsa i)}$ ) (nsubn ls (nnth lsa i) (nnth lsa (Suc i)))))
using pfilt-nmap[of (nsubn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
      (nmap ( $\lambda x. x - \text{nnth ls (nnth lsa i)}$ ) (nsubn ls (nnth lsa i) (nnth lsa (Suc i))) )]]
by blast
have g2: ... =
      nmap (((nnth (nsubn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))  $\circ$ 
      ( $\lambda x. x - \text{nnth ls (nnth lsa i)}$ )))
      (nsubn ls (nnth lsa i) (nnth lsa (Suc i)))
using nellist.map-comp by blast
have g3: ... =
      (nsubn (nmap (((nnth (nsubn  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))  $\circ$ 
      ( $\lambda x. x - \text{nnth ls (nnth lsa i)}$ ))) ls)
      (nnth lsa i) (nnth lsa (Suc i)))
unfolding nsubn-def1 by (simp add: ndropn-nmap)
have g4:  $\bigwedge y. (((nnth (nsubn \sigma (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))) \circ$ 
      ( $\lambda x. x - \text{nnth ls (nnth lsa i)}$ ))) y =
      nnth (nsubn  $\sigma$  (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  $\sigma$  (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))  $\circ$ 
      ( $\lambda x. x - \text{nnth ls (nnth lsa i)}$ ))) ls)
      (nnth lsa i) (nnth lsa (Suc i))) =
      (nsubn (nmap ( $\lambda y. \text{nnth (nsubn } \sigma (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))$ 
      (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 ( $\lambda y. \text{nnth (nsubn } \sigma (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i)))))$ 

```

$$(y - \text{nnth } ls (\text{nnth } lsa \ i)) \text{ } ls \ j = \\ \text{nnth } (nmap (\text{nnth } \sigma) \text{ } ls) \ j$$

proof

fix $j::nat$

assume $aaa: \text{nnth } lsa \ i \leq j \wedge j \leq \text{nnth } lsa \ (Suc \ i)$

show $\text{nnth } (nmap (\lambda y. \text{nnth } (nsubn \ \sigma \ (\text{nnth } ls \ (\text{nnth } lsa \ i)) \ (\text{nnth } ls \ (\text{nnth } lsa \ (Suc \ i)))) \\ (y - \text{nnth } ls \ (\text{nnth } lsa \ i)) \text{ } ls) \ j = \text{nnth } (nmap (\text{nnth } \sigma) \text{ } ls) \ j$

proof –

have $g8: \text{nnth } (nmap (\text{nnth } \sigma) \text{ } ls) \ j = \text{nnth } \sigma \ (\text{nnth } ls \ j)$

by (*metis nfinite-ntaken nlast-nmap ntaken-nlast ntaken-nmap*)

have $g9: \text{nnth } (nmap (\lambda y. \text{nnth } (nsubn \ \sigma \ (\text{nnth } ls \ (\text{nnth } lsa \ i)) \ (\text{nnth } ls \ (\text{nnth } lsa \ (Suc \ i)))) \\ (y - \text{nnth } ls \ (\text{nnth } lsa \ i)) \text{ } ls) \ j = \\ (\lambda y. \text{nnth } (nsubn \ \sigma \ (\text{nnth } ls \ (\text{nnth } lsa \ i)) \ (\text{nnth } ls \ (\text{nnth } lsa \ (Suc \ i)))) \\ (y - \text{nnth } ls \ (\text{nnth } lsa \ i)) \text{ } (\text{nnth } ls \ j)$

by (*metis (no-types, lifting) nfinite-ntaken nlast-nmap ntaken-nlast ntaken-nmap*)

have $g10: \dots = \\ \text{nnth } (nsubn \ \sigma \ (\text{nnth } ls \ (\text{nnth } lsa \ i)) \ (\text{nnth } ls \ (\text{nnth } lsa \ (Suc \ i)))) \\ (\text{nnth } ls \ j - \text{nnth } ls \ (\text{nnth } lsa \ i))$

by *auto*

have $g11: \dots = \text{nnth } \sigma \ ((\text{nnth } ls \ (\text{nnth } lsa \ i)) + (\text{nnth } ls \ j - \text{nnth } ls \ (\text{nnth } lsa \ i))) \\ \text{using } nsubn\text{-nnth}[of \ \sigma \ (\text{nnth } ls \ (\text{nnth } lsa \ i)) \ (\text{nnth } ls \ (\text{nnth } lsa \ (Suc \ i))) \\ (\text{nnth } ls \ j - \text{nnth } ls \ (\text{nnth } lsa \ i))]$

by (*simp add: 2041 3 a aaa diff-le-mono nidx-less-eq*)

have $g12: \dots = \text{nnth } \sigma \ (\text{nnth } ls \ j)$

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 \ (nsubn \ \sigma \ (\text{nnth } ls \ (\text{nnth } lsa \ i)) \ (\text{nnth } ls \ (\text{nnth } lsa \ (Suc \ i)))) \\ (nmap \ (\lambda x. \ x - (\text{nnth } ls \ (\text{nnth } lsa \ i)) \ (nsubn \ ls \ (\text{nnth } lsa \ i) \ (\text{nnth } lsa \ (Suc \ i)) \))) = \\ (nsubn \ (pfilt \ \sigma \ ls) \ (\text{nnth } lsa \ i) \ (\text{nnth } lsa \ (Suc \ i)) \) \\ \text{by } (simp \ add: \ 2041 \ 2060 \ a \ g2 \ g3 \ g5 \ g6 \ nsubn\text{-eq} \ pfilt\text{-nmap})$

show *?thesis* **using** *5 a* **by** (*simp add: g13 powerinterval-def*)

qed

show *?thesis*

using $203 \ 2041 \ 2060 \ 209 \ 210 \ a \ fg3 \ fg5 \ fg6 \ nsubn\text{-nfinite}$ **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: $(\exists \text{ ls. } \text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \neg \text{nfinite } \text{ls} \wedge \neg \text{nfinite } \sigma \wedge$
 $\text{powerinterval } f \sigma \text{ ls} \wedge$
 $((\text{pfilt } \sigma \text{ ls}) \models g \text{ oproj } h))$
using *assms cppl-oprojection* **by** *blast*
obtain *ls* **where** 3: $\text{nidx } \text{ls} \wedge \text{nnth } \text{ls } 0 = 0 \wedge \neg \text{nfinite } \text{ls} \wedge \neg \text{nfinite } \sigma \wedge$
 $\text{powerinterval } f \sigma \text{ ls} \wedge$
 $((\text{pfilt } \sigma \text{ ls}) \models g \text{ oproj } h)$
using 2 **by** *blast*
have 4: $(\exists \text{ lsa. } \text{nidx } \text{lsa} \wedge \text{nnth } \text{lsa } 0 = 0 \wedge \neg \text{nfinite } \text{lsa} \wedge \neg \text{nfinite } (\text{pfilt } \sigma \text{ ls}) \wedge$
 $\text{powerinterval } g (\text{pfilt } \sigma \text{ ls}) \text{ lsa} \wedge$
 $((\text{pfilt } (\text{pfilt } \sigma \text{ ls}) \text{ lsa}) \models h))$
using 3 **using** *cppl-oprojection* **by** *blast*
obtain *lsa* **where** 5: $\text{nidx } \text{lsa} \wedge \text{nnth } \text{lsa } 0 = 0 \wedge \neg \text{nfinite } \text{lsa} \wedge \neg \text{nfinite } (\text{pfilt } \sigma \text{ ls}) \wedge$
 $\text{powerinterval } g (\text{pfilt } \sigma \text{ ls}) \text{ lsa} \wedge$
 $((\text{pfilt } (\text{pfilt } \sigma \text{ ls}) \text{ lsa}) \models h)$
using 4 **by** *blast*
have 7: $\text{nlength}(\text{pfilt } \sigma \text{ ls}) = \text{nlength } \text{ls}$
by (*simp add: pfilt-nlength*)
have 8: $(\text{pfilt } (\text{pfilt } \sigma \text{ ls}) \text{ lsa}) = (\text{pfilt } \sigma (\text{pfilt } \text{ls } \text{lsa}))$
using *pfilt-nmap-pfilt* **by** *blast*
have 11: $\text{nlength } (\text{pfilt } \text{ls } \text{lsa}) > 0$
by (*metis 5 gr-zeroI nlength-eq-enat-nfiniteD pfilt-nlength zero-enat-def*)
have 111: $\text{nnth } (\text{pfilt } \text{ls } \text{lsa}) 0 = 0$
by (*metis 3 5 pfilt-nnth zero-enat-def zero-le*)
have 112: $\text{nlength}(\text{pfilt } \text{ls } \text{lsa}) = \text{nlength } \text{lsa}$
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 (*meson 3 enat-ile linorder-le-cases nfinite-conv-nlength-enat*)
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: $(\forall i < \text{nlength } \text{lsa. } \text{nfinite } (\text{nsubn } \sigma (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)) (\text{nnth } \text{ls } (\text{nnth } \text{lsa } (\text{Suc } i))))$
by (*simp add: nsubn-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 (*meson 3 enat-ile linorder-le-cases nfinite-conv-nlength-enat*)
have 203: $(\forall i < \text{nlength } \text{lsa. } \text{nlength } (\text{nsubn } \sigma (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)) (\text{nnth } \text{ls } (\text{nnth } \text{lsa } (\text{Suc } i)))) =$
 $(\text{nnth } \text{ls } (\text{nnth } \text{lsa } (\text{Suc } i))) - (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)))$
by (*metis 112 113 114 12 3 5 OPJ6help1 le-add1 le-add-same-cancel1 nfinite-nmap*
 pfilt-nmap)
have 2041: $(\forall i < \text{nlength } \text{lsa. } (\text{nnth } \text{lsa } (\text{Suc } i)) \leq \text{nlength } \text{ls})$
by (*metis 1141 eSuc-enat ileI1*)
have 204: $(\forall i < \text{nlength } \text{lsa. } (\text{nnth } \text{lsa } (\text{Suc } i)) \leq \text{nlength } \text{ls})$

$$\text{nlength } (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i))) (\text{nsbn } \text{ls } (\text{nnth } \text{lsa } i) (\text{nnth } \text{lsa } (\text{Suc } i)))) =$$

$$(\text{nnth } \text{lsa } (\text{Suc } i)) - (\text{nnth } \text{lsa } i)$$

by (*simp add: 3 5 OPJ6help1*)

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 (*simp add: nfinite-conv-nlength-enat nidx-less-eq*)

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: $(\forall i < \text{nlength } \text{lsa}.$

$$\text{nlast } (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)))$$

$$(\text{nsbn } \text{ls } (\text{nnth } \text{lsa } i) (\text{nnth } \text{lsa } (\text{Suc } i)))) =$$

$$(\text{nnth } \text{ls } (\text{nnth } \text{lsa } (\text{Suc } i))) - (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i))$$

$$)$$

using 204 207 5 2060 by (*simp add: nsbn-def1 ntaken-ndropn-nlast*)

have 210: $(\forall i < \text{nlength } \text{lsa}.$

$$\text{nidx } (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i))) (\text{nsbn } \text{ls } (\text{nnth } \text{lsa } i) (\text{nnth } \text{lsa } (\text{Suc } i)))) \wedge$$

$$\text{nnth } (\text{nmap } (\lambda x. x - (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i))) (\text{nsbn } \text{ls } (\text{nnth } \text{lsa } i) (\text{nnth } \text{lsa } (\text{Suc } i)))) 0 = 0$$

$$)$$

using 3 5 7 nidx-expand nidx-shiftn 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) =*

$$(\forall i < \text{nlength } \text{lsa}. (\text{nsbn } \sigma (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)) (\text{nnth } \text{ls } (\text{nnth } \text{lsa } (\text{Suc } i)))) \models f \text{ fproj } g)$$

unfolding *powerinterval-def by* (*auto simp add: Suc-ile-eq pfilt-nlength pfilt-nnth*)

have 202: $(\forall i < \text{nlength } \text{lsa}. (\text{nsbn } \sigma (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)) (\text{nnth } \text{ls } (\text{nnth } \text{lsa } (\text{Suc } i)))) \models f \text{ fproj}$

$$g) =$$

$$(\forall i < \text{nlength } \text{lsa}.$$

$$(\exists \text{ls1}. \text{nidx } \text{ls1} \wedge \text{nnth } \text{ls1 } 0 = 0 \wedge \text{nfinite } \text{ls1} \wedge$$

$$\text{nfinite } (\text{nsbn } \sigma (\text{nnth } \text{ls } (\text{nnth } \text{lsa } i)) (\text{nnth } \text{ls } (\text{nnth } \text{lsa } (\text{Suc } i)))) \wedge$$

```

    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)) ≤

```

$$\text{nnth ls (nnth lsa i + Suc ia) - nnth ls (nnth lsa i)}$$
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: $\text{nnth ls (nnth lsa i)} \leq \text{nnth ls (nnth lsa (Suc i))}$
using 115 a dual-order.order-iff-strict **by** blast

have f5: $\text{enat (nnth ls (nnth lsa (Suc i)))} \leq \text{nlength } \sigma$
by (metis 2022 5 a eSuc-enat ileI1 nfinite-conv-nlength-enat order-le-imp-less-or-eq)

have f6: $\text{nnth ls (nnth lsa i + Suc ia) - nnth ls (nnth lsa i)} \leq$
 $\text{nnth ls (nnth lsa (Suc i)) - nnth ls (nnth lsa i)}$
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-nsbn-1[of (nnth ls (nnth lsa i + ia) - nnth ls (nnth lsa i))
(nnnth ls (nnth lsa i + Suc ia) - nnth ls (nnth lsa i))
(nnnth 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: $(\text{nnth (nmap } (\lambda x. x - \text{nnth ls (nnth lsa i)})$
 $(\text{nsubn ls (nnth lsa i) (nnth lsa (Suc i))})) (\text{Suc ia}) =$
 $(\text{nnth (nsubn ls (nnth lsa i) (nnth lsa (Suc i))}) (\text{Suc ia}) - \text{nnth ls (nnth lsa i)})$
by (metis aa eSuc-enat ileI1 nnth-nmap)

have f8: $f (\text{nsubn } \sigma (\text{nnth ls ((nnth lsa i) + ia)}) ((\text{nnth ls ((nnth lsa i) + (Suc ia))})))$
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: $g (\text{pfilt (nsubn } \sigma (\text{nnth ls (nnth lsa i)}) (\text{nnth ls (nnth lsa (Suc i))}))$
 $(\text{nmap } (\lambda x. x - (\text{nnth ls (nnth lsa i)})) (\text{nsubn ls (nnth lsa i) (nnth lsa (Suc i))})))$

proof –

have g1: $(\text{pfilt (nsubn } \sigma (\text{nnth ls (nnth lsa i)}) (\text{nnth ls (nnth lsa (Suc i))}))$
 $(\text{nmap } (\lambda x. x - (\text{nnth ls (nnth lsa i)})) (\text{nsubn ls (nnth lsa i) (nnth lsa (Suc i))}))) =$
 $\text{nmap (nnth (nsubn } \sigma (\text{nnth ls (nnth lsa i)}) (\text{nnth ls (nnth lsa (Suc i))}))}$
 $(\text{nmap } (\lambda x. x - \text{nnth ls (nnth lsa i)}) (\text{nsubn ls (nnth lsa i) (nnth lsa (Suc i))}))$
using pfilt-nmap[of (nsubn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
(nmap $(\lambda x. x - (\text{nnth ls (nnth lsa i)})) (\text{nsubn ls (nnth lsa i) (nnth lsa (Suc i))}))$]
by blast

have g2: ... =
 $\text{nmap } (((\text{nnth (nsubn } \sigma (\text{nnth ls (nnth lsa i)}) (\text{nnth ls (nnth lsa (Suc i))})) \circ$
 $(\lambda x. x - (\text{nnth ls (nnth lsa i)})))$
 $(\text{nsubn ls (nnth lsa i) (nnth lsa (Suc i))}))$
using nellist.map-comp **by** blast

have g3: ... =
 $(\text{nsubn (nmap } (((\text{nnth (nsubn } \sigma (\text{nnth ls (nnth lsa i)}) (\text{nnth ls (nnth lsa (Suc i))})) \circ$
 $(\lambda x. x - (\text{nnth ls (nnth lsa i)}))) \text{ ls}$
 $(\text{nnth lsa i) (nnth lsa (Suc i))}))$
unfolding nsubn-def1 **by** (simp add: ndropn-nmap)

have g4: $\bigwedge y. (((\text{nnth (nsubn } \sigma (\text{nnth ls (nnth lsa i)}) (\text{nnth ls (nnth lsa (Suc i))})) \circ$
 $(\lambda x. x - (\text{nnth ls (nnth lsa i)}))) y =$
 $\text{nnth (nsubn } \sigma (\text{nnth ls (nnth lsa i)}) (\text{nnth ls (nnth lsa (Suc i))}))$

```

      (y - nnth ls (nnth lsa i))
    by simp
  have g5: (nsbn (nmap (((nnth (nsbn 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))) =
    (nsbn (nmap (λy. nnth (nsbn 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:  $\bigwedge j. (nnth lsa i \leq j \wedge j \leq nnth lsa (Suc i)) \longrightarrow$ 
    nnth (nmap (λy. nnth (nsbn 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 (λy. nnth (nsbn 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 -
    have g8: nnth (nmap (nnth σ) ls) j = nnth σ (nnth ls j)
    by (metis nfinite-ntaken nlast-nmap ntaken-nlast ntaken-nmap)
    have g9: nnth (nmap (λy. nnth (nsbn σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
    (y - nnth ls (nnth lsa i))) ls) j =
    (λy. nnth (nsbn σ (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 σ (nnth ls (nnth lsa i)) (nnth ls (nnth lsa (Suc i))))
    (nnth ls j - nnth ls (nnth lsa i))
    by auto
    have g11: ... = nnth σ ((nnth ls (nnth lsa i)) + (nnth ls j - nnth ls (nnth lsa i)))
    using nsbn-nnth[of σ (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: ... = nnth σ (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 (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)))) =
    (nsbn (pfilt σ ls) (nnth lsa i) (nnth lsa (Suc i)))
    by (simp add: 2041 2060 a g2 g3 g5 g6 nsbn-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 nsbn-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*:

$(\sigma \models f \text{ fproj } (g \text{ fproj } h) = (f \text{ fproj } g) \text{ fproj } h)$

proof –

have 1: $nlength \sigma > 0 \longrightarrow (\sigma \models f \text{ fproj } (g \text{ fproj } h) = (f \text{ fproj } g) \text{ fproj } h)$

using *PJ7helpchain1a PJ7helpchain1b unl-lift2* **by** *blast*

have 2: $nlength \sigma = 0 \longrightarrow (\sigma \models f \text{ fproj } (g \text{ fproj } h) = (f \text{ fproj } g) \text{ fproj } h)$

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*

10.3.8 PJ8

lemma *PJ8semhelp*:

assumes *nidx ls*

$nnth \text{ ls } 0 = 0$

nfinite ls

nfinite σ

$nlast \text{ ls} = the-enat(nlength \sigma)$

$(\forall n \text{ na. } na + n \leq nlength \sigma \longrightarrow f (\text{nsbn } \sigma \text{ } n \text{ } (n + na)) \longrightarrow g (\text{nsbn } \sigma \text{ } n \text{ } (n + na)))$

shows

$(\forall i < nlength \text{ ls. } f (\text{nsbn } \sigma \text{ } (nnth \text{ ls } i) \text{ } (nnth \text{ ls } (Suc \text{ } i))) \longrightarrow g (\text{nsbn } \sigma \text{ } (nnth \text{ ls } i) \text{ } (nnth \text{ ls } (Suc \text{ } i))))$

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*

$nnth \text{ ls } 0 = 0$

$\neg nfinite \text{ ls}$

$\neg nfinite \sigma$

$(\forall n \text{ na. } na + n \leq nlength \sigma \longrightarrow f (\text{nsbn } \sigma \text{ } n \text{ } (n + na)) \longrightarrow g (\text{nsbn } \sigma \text{ } n \text{ } (n + na)))$

shows

$(\forall i < nlength \text{ ls. } f (\text{nsbn } \sigma \text{ } (nnth \text{ ls } i) \text{ } (nnth \text{ ls } (Suc \text{ } i))) \longrightarrow g (\text{nsbn } \sigma \text{ } (nnth \text{ ls } i) \text{ } (nnth \text{ ls } (Suc \text{ } i))))$

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*)

10.3.9 PJ9

lemma *PJ9sem*:

$(\sigma \models f \text{ ufproj } (g \longrightarrow h) \longrightarrow f \text{ fproj } g \longrightarrow f \text{ fproj } h)$

by (*simp add: ufprojection-d-def fprojection-d-def*)

(*metis less-numeral-extra*(3))

lemma *OPJ9sem*:

$(\sigma \models f \text{ uoproj } (g \longrightarrow h) \longrightarrow f \text{ oproj } g \longrightarrow f \text{ oproj } h)$

by (*simp add: uoprojection-d-def oprojection-d-def*)

(*metis less-numeral-extra*(3))

10.4 Axioms

lemma *FBpGen*:

assumes $\vdash f$

shows $\vdash \text{fbp } f$

using *assms*

by (*simp add: fbp-d-def ufprojection-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 uoprojection-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*:
 $\vdash \text{skip fproj } g = (g \wedge \text{finite})$
using *PJ6sem Valid-def* **by** *blast*

lemma *OPJ6*:
 $\vdash \text{skip oproj } g = (g \wedge \text{inf})$
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*

10.5 Theorems

10.5.1 Projection

lemma *FPowerFProjLen*:

$\vdash f \text{ fproj } \text{len } n = \text{fpower } (f \wedge \text{more}) \ n$

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 inteq-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:*

$\vdash f \text{ oproj } ((\text{init } w) \wedge \bigcirc g) = ((\text{init } w) \wedge (f \wedge \text{more} \wedge \text{finite}); (f \text{ oproj } g))$
proof –
have 1: $\vdash (\text{skip} \wedge \text{finite}) = \text{skip}$
using *itl-defs(1) itl-defs(5) nlength-eq-enat-nfiniteD by fastforce*
have 2: $\vdash (f \wedge \text{more} \wedge \text{finite}); (f \text{ oproj } g) = f \text{ oproj } \bigcirc g$
using *PJ3 OPJ4 unfolding next-d-def by (metis 1 inteq-reflection)*
have 3: $\vdash f \text{ oproj } ((\text{init } w)) \longrightarrow \text{init } w$
by (*simp add: OPJ5*)
have 4: $\vdash (\text{init } w \wedge f \text{ oproj } \bigcirc g) = f \text{ oproj } ((\text{init } w) \wedge \bigcirc g)$
by (*simp add: StateImportOProj*)
have 5: $\vdash f \text{ oproj } ((\text{init } w) \wedge \bigcirc g) \longrightarrow ((\text{init } w) \wedge (f \wedge \text{more} \wedge \text{finite}); (f \text{ oproj } g))$
using 2 3 *OProjAndImp by fastforce*
from 5 4 **show** *?thesis using 2 by fastforce*
qed

lemma *FProjNext:*

$\vdash f \text{ fproj } \bigcirc g = (f \wedge \text{more} \wedge \text{finite}); (f \text{ fproj } g)$
by (*metis PJ3 PJ4 inteq-reflection next-d-def*)

lemma *OProjNext:*

$\vdash f \text{ oproj } \bigcirc g = (f \wedge \text{more} \wedge \text{finite}); (f \text{ oproj } g)$
by (*metis DiamondEmptyEqvFinite DiamondEqvEmptyOrNextDiamond FiniteChopEqvDiamond*
FiniteChopSkipEqvSkipChopFinite NowImpDiamond OPJ4 PJ3 Prop05 Prop10 SkipChopEqvNext
inteq-reflection)

lemma *FProjWnext:*

$\vdash f \text{ fproj } (wnext \ g) = (\text{empty} \vee (f \wedge \text{more} \wedge \text{finite}); (f \text{ fproj } g))$
proof –
have 1: $\vdash f \text{ fproj } (wnext \ g) = f \text{ fproj } (\text{empty} \vee \bigcirc g)$

by (*simp add: RightFProjEqvFProj WnextEqvEmptyOrNext*)
have 2: $\vdash f \text{ fproj } (\text{empty} \vee \circ g) = (\text{empty} \vee f \text{ fproj } (\circ g))$
using *PJ1 PJ2* **by** *fastforce*
have 3: $\vdash f \text{ fproj } (\circ g) = (f \wedge \text{more} \wedge \text{finite}); (f \text{ fproj } g)$
by (*metis PJ3 PJ4 inteq-reflection next-d-def*)
show *?thesis*
using 1 2 3 **by** *fastforce*
qed

lemma *OProjWnext*:

$\vdash f \text{ oproj } (\text{wnext } g) = ((f \wedge \text{more} \wedge \text{finite}); (f \text{ oproj } g))$
proof –
have 1: $\vdash f \text{ oproj } (\text{wnext } g) = f \text{ oproj } (\text{empty} \vee \circ g)$
by (*simp add: RightOProjEqvOProj WnextEqvEmptyOrNext*)
have 2: $\vdash f \text{ oproj } (\text{empty} \vee \circ g) = (f \text{ oproj } (\circ g))$
using *OPJ2[of f] OPJ1[of f LIFT empty LIFT \circ g]* **by** *fastforce*
have 3: $\vdash f \text{ oproj } (\circ g) = (f \wedge \text{more} \wedge \text{finite}); (f \text{ oproj } g)$
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 OmegaWeakCoinduct int-eq*)

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*:

$(\forall n. \text{enat } n \leq \text{nlength } wa \longrightarrow w \text{ (NNil (nfirst (ndropn } n \text{ } wa)))) \wedge$
 $(\exists ls. (\forall i. \text{enat } (\text{Suc } i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls \text{ } i < \text{nnth } ls \text{ } (\text{Suc } i)) \wedge$
 $\text{nnth } ls \text{ } 0 = 0 \wedge$
 $\text{nfinite } ls \wedge$
 $\text{nfinite } wa \wedge$

$nlast\ ls = the-enat\ (nlength\ wa) \wedge$
 $(\forall i. enat\ i < nlength\ ls \longrightarrow f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \wedge g\ (pfilt\ wa\ ls) \longrightarrow$
 $(\exists ls. (\forall i. enat\ (Suc\ i) \leq nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i)) \wedge$
 $nnth\ ls\ 0 = 0 \wedge$
 $nfinite\ ls \wedge$
 $nfinite\ wa \wedge$
 $nlast\ ls = the-enat\ (nlength\ wa) \wedge$
 $(\forall i. enat\ i < nlength\ ls \longrightarrow$
 $f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \wedge$
 $(\forall n. enat\ n \leq nlength\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \longrightarrow$
 $w\ (NNil\ (nfirst\ (ndropn\ n\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))))) \wedge$
 $g\ (pfilt\ wa\ ls))$

proof

assume 0: $(\forall n. enat\ n \leq nlength\ wa \longrightarrow w\ (NNil\ (nfirst\ (ndropn\ n\ wa)))) \wedge$
 $(\exists ls. (\forall i. enat\ (Suc\ i) \leq nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i)) \wedge$
 $nnth\ ls\ 0 = 0 \wedge$
 $nfinite\ ls \wedge$
 $nfinite\ wa \wedge$
 $nlast\ ls = the-enat\ (nlength\ wa) \wedge$
 $(\forall i. enat\ i < nlength\ ls \longrightarrow f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))) \wedge g\ (pfilt\ wa\ ls))$
show $(\exists ls. (\forall i. enat\ (Suc\ i) \leq nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i)) \wedge$
 $nnth\ ls\ 0 = 0 \wedge$
 $nfinite\ ls \wedge$
 $nfinite\ wa \wedge$
 $nlast\ ls = the-enat\ (nlength\ wa) \wedge$
 $(\forall i. enat\ i < nlength\ ls \longrightarrow$
 $f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \wedge$
 $(\forall n. enat\ n \leq nlength\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \longrightarrow$
 $w\ (NNil\ (nfirst\ (ndropn\ n\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i)))))) \wedge$
 $g\ (pfilt\ wa\ ls))$

proof –

have 1: $(\forall n. enat\ n \leq nlength\ wa \longrightarrow w\ (NNil\ (nfirst\ (ndropn\ n\ wa))))$
using 0 **by** auto
have 2: $(\exists ls. (\forall i. enat\ (Suc\ i) \leq nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i)) \wedge$
 $nnth\ ls\ 0 = 0 \wedge$
 $nfinite\ ls \wedge$
 $nfinite\ wa \wedge$
 $nlast\ ls = the-enat\ (nlength\ wa) \wedge$
 $(\forall i. enat\ i < nlength\ ls \longrightarrow$
 $f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \wedge g\ (pfilt\ wa\ ls))$
using 0 **by** auto
obtain ls **where** 3: $(\forall i. enat\ (Suc\ i) \leq nlength\ ls \longrightarrow nnth\ ls\ i < nnth\ ls\ (Suc\ i)) \wedge$
 $nnth\ ls\ 0 = 0 \wedge$
 $nfinite\ ls \wedge$
 $nfinite\ wa \wedge$
 $nlast\ ls = the-enat\ (nlength\ wa) \wedge$
 $(\forall i. enat\ i < nlength\ ls \longrightarrow$
 $f\ (nsubn\ wa\ (nnth\ ls\ i)\ (nnth\ ls\ (Suc\ i))) \wedge g\ (pfilt\ wa\ ls))$
using 2 **by** auto
have 4: $nnth\ ls\ 0 = 0$

```

using 3 by auto
have 5: (∀ i. enat (Suc i) ≤ nlength ls → nnth ls i < nnth ls (Suc i))
using 3 by auto
have 6: nlast ls = the-enat(nlength wa)
using 3 by auto
have 7: (∀ 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: (∀ i < nlength ls.
  f (nsubn wa ( nnth ls i ) ( nnth ls (Suc i))) ∧
  (∀ n ≤ (nnth ls (Suc i)) − (nnth ls i).
    w (NNil (nnth wa ((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: (∀ i < nlength ls.
  nlength (nsubn wa ( nnth ls i ) ( nnth ls (Suc i)) ) =
  (nnth ls (Suc i)) − (nnth ls i) )
by (simp add: 3 PJ6help1 Suc-ile-eq nidx-expand)
have 11: (∀ i < nlength ls.
  (∀ n ≤ (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: (∀ i < nlength ls.
  f (nsubn wa ( nnth ls i ) ( nnth ls (Suc i)) ) ∧
  (∀ n ≤ 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: (∀ i. enat i < nlength ls →
  f (nsubn wa (nnth ls i) (nnth ls (Suc i))) ∧
  (∀ n. enat n ≤ nlength (nsubn wa (nnth ls i) (nnth ls (Suc i))) →
    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 *LeftBoxStateImportOProjhelp:*

```

(∀ n. enat n ≤ nlength wa → w (NNil (nfirst (ndropn n wa)))) ∧
(∃ ls. (∀ i. enat (Suc i) ≤ nlength ls → nnth ls i < nnth ls (Suc i)) ∧
  nnth ls 0 = 0 ∧
  ¬ nfinite ls ∧ ¬ nfinite wa ∧
  (∀ i. enat i < nlength ls → f (nsubn wa (nnth ls i) (nnth ls (Suc i)))) ∧ g (pfilt wa ls)) →
(∃ ls. (∀ i. enat (Suc i) ≤ nlength ls → nnth ls i < nnth ls (Suc i)) ∧
  nnth ls 0 = 0 ∧
  ¬ nfinite ls ∧
  ¬ nfinite wa ∧

```

$(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow$
 $f (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) \wedge$
 $(\forall n. \text{enat } n \leq \text{nlength } (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) \longrightarrow$
 $w (\text{NNil } (\text{nfirst } (\text{ndropn } n (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i)))))) \wedge$
 $g (\text{pfilt } wa \ ls))$

proof

assume 0: $(\forall n. \text{enat } n \leq \text{nlength } wa \longrightarrow w (\text{NNil } (\text{nfirst } (\text{ndropn } n \ wa)))) \wedge$
 $(\exists ls. (\forall i. \text{enat } (\text{Suc } i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls \ i < \text{nnth } ls (\text{Suc } i)) \wedge$
 $\text{nnth } ls \ 0 = 0 \wedge$
 $\neg \text{nfinite } ls \wedge \neg \text{nfinite } wa \wedge$
 $(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow f (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i)))) \wedge g (\text{pfilt } wa \ ls))$
show $(\exists ls. (\forall i. \text{enat } (\text{Suc } i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls \ i < \text{nnth } ls (\text{Suc } i)) \wedge$
 $\text{nnth } ls \ 0 = 0 \wedge$
 $\neg \text{nfinite } ls \wedge$
 $\neg \text{nfinite } wa \wedge$
 $(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow$
 $f (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) \wedge$
 $(\forall n. \text{enat } n \leq \text{nlength } (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i))) \longrightarrow$
 $w (\text{NNil } (\text{nfirst } (\text{ndropn } n (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i)))))) \wedge$
 $g (\text{pfilt } wa \ ls))$

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: $(\exists ls. (\forall i. \text{enat } (\text{Suc } i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls \ i < \text{nnth } ls (\text{Suc } i)) \wedge$
 $\text{nnth } ls \ 0 = 0 \wedge$
 $\neg \text{nfinite } ls \wedge \neg \text{nfinite } wa \wedge$
 $(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow f (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i)))) \wedge g (\text{pfilt } wa \ ls))$
using 0 **by** *auto*
obtain *ls* **where** 3: $(\forall i. \text{enat } (\text{Suc } i) \leq \text{nlength } ls \longrightarrow \text{nnth } ls \ i < \text{nnth } ls (\text{Suc } i)) \wedge$
 $\text{nnth } ls \ 0 = 0 \wedge$
 $\neg \text{nfinite } ls \wedge \neg \text{nfinite } wa \wedge$
 $(\forall i. \text{enat } i < \text{nlength } ls \longrightarrow f (\text{nsbn } wa (\text{nnth } ls \ i) (\text{nnth } ls (\text{Suc } i)))) \wedge g (\text{pfilt } wa \ ls)$
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 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 *linorder-le-cases ndropn-eq-NNil ndropn-nfirst nfinite-NNil nfinite-ndropn-b*)
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 OPJ6help1 nidx-expand)
  have 11: (∀ i < nlength ls.
    (∀ n ≤ (nnth ls (Suc i)) ¬ (nnth ls i).
      nnth (nsbn 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: (∀ i < nlength ls.
    f (nsbn wa ( nnth ls i ) ( nnth ls (Suc i)) ) ∧
    (∀ n ≤ nlength (nsbn wa ( nnth ls i ) ( nnth ls (Suc i)) ).
      w (NNil (nnth (nsbn wa ( nnth ls i ) ( nnth ls (Suc i)) ) n))))
    using 9 10 11 by simp
  have 13: (∀ i. enat i < nlength ls ⟶
    f (nsbn wa (nnth ls i) (nnth ls (Suc i))) ∧
    (∀ n. enat n ≤ nlength (nsbn wa (nnth ls i) (nnth ls (Suc i))) ⟶
      w (NNil (nfirst (ndropn n (nsbn 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*:

```

  ⊢ □(init w) ∧ f fproj g ⟶ (f ∧ □ (init w)) 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*:

```

  ⊢ □(init w) ∧ f oproj g ⟶ (f ∧ □ (init w)) oproj g
  using LeftBoxStateImportOProjhelp[of - w f g]
  by (simp add: Valid-def always-defs init-defs oprojection-d-def powerinterval-def nidx-expand)

```

10.5.2 fdp, fbp odp and obp

lemma *NotFDpEqvFBpNot*:

```

  ⊢ (¬(fdp f)) = fbp (¬ f)
  by (simp add: fbp-d-def fdp-d-def ufprojection-d-def)

```

lemma *NotODpEqvOBpNot*:

```

  ⊢ (¬(odp f)) = obp (¬ f)
  by (simp add: obp-d-def odp-d-def uoprojection-d-def)

```

lemma *NotFDBpEqvFDpNot*:

```

  ⊢ (¬(fbp f)) = fdp(¬ f)
  by (simp add: fbp-d-def fdp-d-def ufprojection-d-def)

```

lemma *NotODBpEqvODpNot*:

```

  ⊢ (¬(obp f)) = odp(¬ 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: $\vdash \text{ba}(\text{skip} \longrightarrow \# \text{True})$
using 1 **by** (*simp add: BaGen*)
have 3: $\vdash \text{ba}(\text{skip} \longrightarrow \# \text{True}) \longrightarrow (\text{skip } \text{oproj } f \longrightarrow \# \text{True } \text{oproj } f)$
using *OPJ8* **by** *blast*
have 4: $\vdash (\text{skip } \text{oproj } f \longrightarrow \# \text{True } \text{oproj } f)$
using 2 3 *MP* **by** *blast*
show *?thesis* **using** 4 *OPJ6* **by** (*metis* *int-eq* *odp-d-def*)
qed

lemma *FBpElim*:
 $\vdash \text{fbp } f \wedge \text{finite} \longrightarrow f$
proof –
have 1: $\vdash \neg f \wedge \text{finite} \longrightarrow \text{fdp } (\neg f)$
by (*simp add: NowImpFDp*)
hence 2: $\vdash \neg(\text{fdp } (\neg f)) \longrightarrow f \vee \text{inf}$
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*:
 $\vdash \text{obp } f \wedge \text{inf} \longrightarrow f$
proof –
have 1: $\vdash \neg f \wedge \text{inf} \longrightarrow \text{odp } (\neg f)$
by (*simp add: NowImpODp*)
hence 2: $\vdash \neg(\text{odp } (\neg f)) \longrightarrow f \vee \text{finite}$
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*:

$\vdash \text{fbp } (f \longrightarrow g) \longrightarrow \text{fdp } f \longrightarrow \text{fdp } g$

proof –

have 1: $\vdash \text{fbp } (f \longrightarrow g) \longrightarrow (\# \text{True } \text{fproj } f) \longrightarrow (\# \text{True } \text{fproj } g)$

by (*simp add: PJ9 fbp-d-def*)

from 1 **show** *?thesis* **by** (*simp add: fdp-d-def*)

qed

lemma *OBpImpODpImpODp*:

$\vdash \text{obp } (f \longrightarrow g) \longrightarrow \text{odp } f \longrightarrow \text{odp } g$

proof –

have 1: $\vdash \text{obp } (f \longrightarrow g) \longrightarrow (\# \text{True } \text{oproj } f) \longrightarrow (\# \text{True } \text{oproj } g)$

by (*simp add: OPJ9 obp-d-def*)

from 1 **show** *?thesis* **by** (*simp add: odp-d-def*)

qed

lemma *FBpContraPosImpDist*:

$\vdash \text{fbp } (\neg g \longrightarrow \neg f) \longrightarrow (\text{fbp } f) \longrightarrow (\text{fbp } g)$

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: $\vdash \# \text{True } \text{fproj } f \longrightarrow \# \text{True } \text{fproj } g$
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 $\vdash \text{odp } f \longrightarrow \text{odp } g$

proof –

have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*

hence 2: $\vdash \# \text{True } \text{oproj } f \longrightarrow \# \text{True } \text{oproj } g$
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 $\vdash \text{fbp } f \longrightarrow \text{fbp } g$

proof –

have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*

hence 2: $\vdash \neg g \longrightarrow \neg f$ **by** *auto*

hence 3: $\vdash \text{fdp } (\neg g) \longrightarrow \text{fdp } (\neg f)$ **by** (*rule FDpImpDpRule*)

hence 4: $\vdash \neg (\text{fdp } (\neg f)) \longrightarrow \neg (\text{fdp } (\neg g))$ **by** *auto*

from 4 **show** *?thesis*

by (*meson FBpGen FBpImpDist MP assms*)

qed

lemma *OBpImpOBpRule*:

assumes $\vdash f \longrightarrow g$

shows $\vdash \text{obp } f \longrightarrow \text{obp } g$

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 $\vdash fdp f = fdp g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash \#True fproj f = \#True fproj g$
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*:

$\vdash \text{fdp } (\text{init } w) = ((\text{init } w) \wedge \text{finite})$

proof –

have 1: $\vdash \text{init } w \wedge \text{finite} \longrightarrow \text{fdp } (\text{init } w)$

using *NowImpFDp*[of *LIFT* (*init w*)] **by** *blast*

have 2: $\vdash \text{fdp } (\text{init } w) \longrightarrow \text{init } w$

unfolding *fdp-d-def* **by** (*simp add: PJ5*)

have 3: $\vdash \text{fdp } (\text{init } w) \longrightarrow \text{finite}$

unfolding *fdp-d-def* **by** (*simp add: PJ01*)

show *?thesis*

by (*simp add: 1 2 3 Prop12 int-iffI*)

qed

lemma *ODpState*:

$\vdash \text{odp } (\text{init } w) = ((\text{init } w) \wedge \text{inf})$

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*:

$\vdash fdp (fdp f) = fdp f$
proof –
have 2: $\vdash \#True \text{ fproj } (\#True \text{ fproj } f) = (\#True \text{ fproj } \#True) \text{ fproj } f$
by (*simp add: PJ7*)
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: $\vdash \text{finite fproj } f = \#True \text{ fproj } f$
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*:

$\vdash odp (odp f) = odp f$
proof –
have 2: $\vdash \#True \text{ oproj } (\#True \text{ oproj } f) = (\#True \text{ fproj } \#True) \text{ oproj } f$
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: $\vdash \text{finite oproj } f = \#True \text{ oproj } f$
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*:

$\vdash fbp (fbp f) = fbp f$
proof –
have 1: $\vdash fdp (fdp (\neg f)) = fdp (\neg f)$
using *FDpFDpEqvFDp* **by** *blast*
have 2: $\vdash (\neg (fdp (fdp (\neg f)))) = (\neg (fdp (\neg f)))$
using 1 **by** *auto*
have 3: $\vdash (\neg (fdp (\neg f))) = fbp f$
by (*simp add: fbp-d-def fdp-d-def ufprojection-d-def*)
have 4: $\vdash (\neg (fdp (fdp (\neg f)))) = fbp (fbp f)$
by (*simp add: fbp-d-def fdp-d-def ufprojection-d-def*)
from 2 3 4 **show** ?thesis
by *fastforce*
qed

lemma *OBpOBpEqvOBp*:

$\vdash obp (obp f) = obp f$
proof –
have 1: $\vdash odp (odp (\neg f)) = odp (\neg f)$
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: $\vdash \# \text{True } \text{fproj } (f \vee g) = (\# \text{True } \text{fproj } f \vee \# \text{True } \text{fproj } g)$

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: $\vdash \# \text{True } \text{oproj } (f \wedge g) \longrightarrow \# \text{True } \text{oproj } g$
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: $\vdash \text{fdp } \text{skip} = \# \text{True } \text{fproj } \text{skip}$
by (*simp add: fdp-d-def*)
have 2: $\vdash \# \text{True } \text{fproj } \text{skip} = (\# \text{True} \wedge \text{more} \wedge \text{finite})$
using *PJ3* **by** *blast*
have 3: $\vdash (\# \text{True} \wedge \text{more}) = \text{more}$
by *auto*
from 1 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *FDpMoreEqvMore*:

$\vdash \text{fdp more} = (\text{more} \wedge \text{finite})$

using *FDpFDpEqvFDp FDpSkipEqvMore*

by (*metis PJ03 fdp-d-def inteq-reflection*)

lemma *ODpMoreEqvInf*:

$\vdash \text{odp more} = (\text{inf})$

by (*metis MoreAndInfEqvInf NowImpODp OPJ01 int-iffI inteq-reflection odp-d-def*)

lemma *FBpEmptyEqvEmpty*:

$\vdash \text{fbp empty} = (\text{empty} \vee \text{inf})$

proof –

have 1: $\vdash \text{fbp empty} = (\neg (\text{fdp more}))$

by (*metis NotFDpEqvFBpNot Prop11 empty-d-def*)

have 2: $\vdash (\neg (\text{fdp more})) = (\neg (\text{more} \wedge \text{finite}))$

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*:
 $\vdash \bigcirc (\text{fdp } f) \longrightarrow \text{fdp } (\bigcirc f)$
proof –
have 1: $\vdash \text{fdp}(\bigcirc f) = \# \text{True fproj (skip;f)}$
by (*simp add: fdp-d-def next-d-def*)
have 2: $\vdash \# \text{True fproj (skip;f)} = (\# \text{True fproj skip});(\# \text{True fproj } f)$
by (*simp add: PJ4*)
have 3: $\vdash (\# \text{True fproj skip}) = (\# \text{True} \wedge \text{more} \wedge \text{finite})$
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: $\vdash \text{skip};(\# \text{True fproj } f) \longrightarrow (\text{more} \wedge \text{finite});(\# \text{True fproj } f)$
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*:
 $\vdash \bigcirc (\text{odp } f) \longrightarrow \text{odp } (\bigcirc f)$
proof –
have 1: $\vdash \text{odp}(\bigcirc f) = \# \text{True oproj (skip;f)}$
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: $\vdash \# \text{True oproj (skip;f)} = (\# \text{True fproj skip});(\# \text{True oproj } f)$
using *OPJ4* **by** (*metis 10 inteq-reflection*)
have 3: $\vdash (\# \text{True fproj skip}) = (\# \text{True} \wedge \text{more} \wedge \text{finite})$
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: $\vdash (\text{fdp } \text{finite}) = \text{finite}$

by (*metis EmptyOrMoreSplit FDPEmptyEqvEmpty FiniteAndEmptyEqvEmpty FiniteChopSkipEqvFinite-AndMore*

FiniteChopSkipImpFinite NowImpFDP PJ01 Prop02 Prop10 fdp-d-def int-iffI inteq-reflection)

have 5: $\vdash (\text{fdp } (\neg(\text{init } w))) = ((\neg(\text{init } w)) \wedge \text{finite})$

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: $\vdash \text{odp } (\text{finite};(\neg(\text{init } w))) = (\text{fdp } \text{finite}); (\text{odp } (\neg(\text{init } w)))$

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: $\vdash (\text{fdp } \text{finite}) = \text{finite}$

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:  $\vdash \Box (\text{init } w) \wedge \text{finite} \longrightarrow \text{fdp} (\Box (\text{init } w))$ 
    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 inteq-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

```

11 Infinite and Finite 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

```

11.1 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 = {  $\sigma \cdot (\sigma \models f)$  }

```

definition *fusion* :: 'a iintervals \Rightarrow 'a iintervals \Rightarrow 'a iintervals (**infixl** · 70)

where $X \cdot Y = \{ \sigma .$

$(\exists \sigma 1 \sigma 2. \sigma = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge$
 $(\sigma 1 \in X) \wedge (\sigma 2 \in Y) \wedge (\text{nlast } \sigma 1 = \text{nfirst } \sigma 2))$
 $\vee (\neg \text{nfinite } \sigma \wedge \sigma \in X) \}$

definition *empty* :: 'a iintervals (*SEmpty*)

where

$SEmpty \equiv \{ \sigma . \text{nlength } \sigma = 0 \}$

definition *smore* :: 'a iintervals (*SMore*)

where

$SMore \equiv -\text{SEmpty}$

definition *sskip* :: 'a iintervals (*SSkip*)

where

$SSkip \equiv -(\text{SEmpty} \cup (\text{SMore} \cdot \text{SMore}))$

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

$SInit\ X \equiv (X \cap \text{SEmpty}) \cdot \text{STrue}$

definition *sinf* :: 'a iintervals (*SInf*)

where $SInf \equiv \text{STrue} \cdot \text{SFalse}$

definition *sfinite* :: 'a iintervals (*SFinite*)

where $SFinite \equiv -SInf$

definition *sfmore* :: 'a iintervals (*SFMore*)

where $SFMore \equiv SFinite \cap \text{SMore}$

definition *sfin* :: 'a iintervals \Rightarrow 'a iintervals ((*SFin* -) [85] 85)

where

$SFin\ X \equiv SFinite \cdot (X \cap \text{SEmpty})$

definition *ssometime* :: 'a iintervals \Rightarrow 'a iintervals ((*SSometime* -) [85] 85)

where

$SSometime\ X \equiv SFinite \cdot X$

definition *salways* :: 'a iintervals \Rightarrow 'a iintervals ((*SAlways* -) [85] 85)

where

$SAlways\ X \equiv \neg(SSometime\ (\neg X))$

definition $sdi :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SDi\ -)\ [85]\ 85)$

where

$SDi\ X \equiv X \cdot STrue$

definition $sbi :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SBi\ -)\ [85]\ 85)$

where

$SBi\ X \equiv \neg(SDi\ (\neg X))$

definition $sda :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SDa\ -)\ [85]\ 85)$

where

$SDa\ X \equiv SFinite \cdot X \cdot STrue$

definition $sba :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SBa\ -)\ [85]\ 85)$

where

$SBa\ X \equiv \neg(SDa\ (\neg X))$

definition $snext :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SNext\ -)\ [85]\ 85)$

where

$SNext\ X \equiv SSkip \cdot X$

definition $swnext :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SWnext\ -)\ [85]\ 85)$

where

$SWnext\ X \equiv (\neg(SSkip \cdot (\neg X)))$

definition $sprev :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SPrev\ -)\ [85]\ 85)$

where

$SPrev\ X \equiv X \cdot SSkip$

definition $swprev :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SWprev\ -)\ [85]\ 85)$

where

$SWprev\ X \equiv (\neg((\neg X) \cdot SSkip))$

primrec $spower :: 'a\ iintervals \Rightarrow nat \Rightarrow 'a\ iintervals\ ((SPower\ -\ -)\ [88,88]\ 87)$

where

$pwr-0 : SPower\ X\ 0 = SEmpty$

$| pwr-Suc: SPower\ X\ (Suc\ n) = ((X \cap SFinite) \cdot (SPower\ X\ n))$

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

$SStar\ X \equiv SPowerstar(X \cap SMore)$

11.2 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*:

$$\sigma \in X \cdot Y \longleftrightarrow \\ (\\ (\exists\ \sigma 1\ \sigma 2. \sigma = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge \\ (\sigma 1 \in X) \wedge (\sigma 2 \in Y) \wedge (nlast\ \sigma 1 = nfirst\ \sigma 2)) \\ \vee (\neg nfinite\ \sigma \wedge \sigma \in X))$$

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* σ]

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*
empty-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* :

$\sigma \in SPower\ X\ 0 \longleftrightarrow \sigma \in SEmpty$

by *simp*

lemma *spower-elim-suc* :

$\sigma \in SPower\ X\ (Suc\ n) \longleftrightarrow \sigma \in (X \cap SFinite) \cdot (SPower\ X\ n)$

by *simp*

lemma *spower-elim-suc-1* :

$\sigma \in (X \cap SFinite) \cdot (SPower\ X\ n) \longleftrightarrow$
 $(\exists\ \sigma 1\ \sigma 2. \sigma = n\text{fuse}\ \sigma 1\ \sigma 2 \wedge n\text{finite}\ \sigma 1 \wedge$
 $\sigma 1 \in X \wedge \sigma 2 \in (SPower\ X\ n) \wedge$
 $n\text{last}\ \sigma 1 = n\text{first}\ \sigma 2)$
 $)$

by (*simp add: ffusion-iff*)

lemma *ffusionimp*:

assumes $Y0 \subseteq Y1$

shows $\sigma \in (X \cap SFinite) \cdot Y0 \longrightarrow \sigma \in (X \cap SFinite) \cdot Y1$

using *assms unfolding ffusion-iff-1 by (auto)*

lemma *spower-finite*:

$(SPower\ X\ n) \subseteq SFinite$

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: $(SPower\ X\ (Suc\ n)) = (X \cap SFinite) \cdot (SPower\ X\ n)$

by *simp*

have 2: $SPower\ X\ n \subseteq SFinite$

using *Suc.hyps by blast*

have 3: $(X \cap SFinite) \cdot SFinite \subseteq SFinite$

using *ffusion-iff-1[of - X SFinite]*

by (*simp add: sfinite-elim*)

(*metis ComplI sfinite-def sinfinite-elim subsetI*)

have 4: $(X \cap SFinite) \cdot (SPower\ X\ n) \subseteq (X \cap SFinite) \cdot SFinite$

using *Suc.hyps ffusionimp by blast*

show ?*thesis*

using 3 4 **by** *auto*

qed

qed

lemma *sfpowerstar-elim*:

$\sigma \in SFPowerstar\ X \longleftrightarrow (\exists\ n. \sigma \in SPower\ X\ n)$

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*)

11.3 Algebraic Laws

11.3.1 Commutative Additive Monoid

lemma *UnionCommute*:

$$(X::'a \text{ iintervals}) \cup Y = Y \cup X$$

by (*simp add: Un-commute*)

lemma *UnionSFalse*:

$$X \cup SFalse = X$$

by (*simp add: sfalse-def*)

lemma *UnionAssoc*:

$$(X::'a \text{ iintervals}) \cup (Y \cup Z) = (X \cup Y) \cup Z$$

by (*simp add: sup-assoc*)

11.3.2 Boolean algebra

lemma *Huntington*:

$$(X::'a \text{ iintervals}) = -(-X \cup -Y) \cup -(-X \cup Y)$$

by *auto*

lemma *Morgan*:

$$(X::'a \text{ iintervals}) \cap Y = -(-X \cup -Y)$$

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)

11.3.3 multiplicative monoid

lemma *FusionSEmptyLsem*:

$\sigma \in SEmpty \cdot X \longleftrightarrow \sigma \in X$

using fusion-iff-1[of $\sigma SEmpty X$]

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 $SEmpty \cdot X X$] *FusionSEmptyLsem*

by auto

lemma *FusionSEmptyRsem* :

$\sigma \in X \cdot SEmpty \longleftrightarrow \sigma \in X$

using fusion-iff-1[of $\sigma X SEmpty$]

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 $X \cdot SEmpty X$] *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 $x X Y \cdot Z$] 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*

11.3.4 Subsumption order

lemma *Subsumption*:

$((X :: 'a \text{ intervals}) \subseteq Y) = (X \cup Y = Y)$

by *auto*

11.3.5 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:  $X \cap SFinite = ((X \cap SMore) \cup ((X \cap SEmpty) \cap SFinite))$ 
  by (simp add: sfmore-def smore-def) blast
have 4:  $(X \cap SFinite) \cdot Z = ((X \cap SMore) \cdot Z \cup ((X \cap SEmpty) \cap SFinite) \cdot Z)$ 
  by (simp add: 31 FusionUnionDistL)
have 5:  $(X \cap SMore) \cdot Z \subseteq Z$ 
  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 –

$$\text{have } 1: Y \cdot (SStar\ X) = Y \cdot ((\bigcup n. (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup X \cap SMore \cap SInf))$$

by (*simp add: sstar-def spowerstar-def sfpowerstar-def*)

$$\text{have } 2: Y \cdot ((\bigcup n. (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup X \cap SMore \cap SInf)) =$$

$$(Y \cdot (\bigcup n. (SPower\ (X \cap SMore)\ n))) \cdot (SEmpty \cup X \cap SMore \cap SInf)$$

using *FusionAssoc* **by** *blast*

$$\text{have } 3: (Y \cdot (\bigcup n. (SPower\ (X \cap SMore)\ n))) = (\bigcup n. Y \cdot (SPower\ (X \cap SMore)\ n))$$

using *FusionUnionSPower1* **by** *blast*

$$\text{have } 4: (Y \cdot (\bigcup n. (SPower\ (X \cap SMore)\ n))) \cdot (SEmpty \cup X \cap SMore \cap SInf) =$$

$$(\bigcup n. Y \cdot (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup X \cap SMore \cap SInf)$$

by (*simp add: 3*)

$$\text{have } 5: (\bigcup n. Y \cdot (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup X \cap SMore \cap SInf) =$$

$$(\bigcup n. (Y \cdot (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup X \cap SMore \cap SInf))$$

using *UnionFusionSPower[of Y X 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: $(SFPowerstar\ X) \cdot Y = (\bigcup n. (SPower\ X\ n)) \cdot Y$
 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: $\bigwedge n. (SPower\ X\ n) \cdot Y \subseteq Z$
 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* $(\bigcup n. ((SPower\ (X \cap SMore)\ n)) (SEmpty \cup (X \cap SMore) \cap SInf)\ Y)$]
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 $Y (X \cap SMore) (SEmpty \cup ((X \cap SMore) \cap SInf))$]

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-eqv-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: sempty-elim zero-enat-def)
next
case (Suc n)
then show ?case
proof –
 have 1: $(\sigma \in SPower SSkip (Suc n)) \longleftrightarrow \sigma \in (SFinite \cap SSkip) \cdot (SPower SSkip n)$
 by (simp add: inf-commute)
 have 2: $\sigma \in (SFinite \cap SSkip) \longleftrightarrow \sigma \in SSkip$
 using SSkipSFinite by auto
 have 3: $\sigma \in (SFinite \cap SSkip) \cdot (SPower SSkip n) \longleftrightarrow$
 $(\exists \sigma 1 \sigma 2. \sigma = nfuse \sigma 1 \sigma 2 \wedge nfinite \sigma 1 \wedge \sigma 1 \in SSkip \wedge \sigma 2 \in SPower SSkip n \wedge nlast \sigma 1 = nfirst$

$\sigma 2)$
using *ffusion-iff*[of σ *SSkip* *SPower* *SSkip* n] **by** (*simp add: inf-commute*)
have 4: $(\exists \sigma 1 \sigma 2. \sigma = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge \sigma 1 \in \text{SSkip} \wedge \sigma 2 \in \text{SPower } \text{SSkip } n \wedge \text{nlast } \sigma 1 = \text{nfir} \text{st } \sigma 2) \longleftrightarrow$
 $(\text{nlength } \sigma = \text{enat } (\text{Suc } n))$
by (*auto simp add: Suc.hyps nfuse-nlength one-enat-def sskip-elim*)
 $(\text{metis One-nat-def add.commute eSuc-enat enat.simps(3) enat-add-sub-same enat-le-plus-same(2)}$
 $\text{min.orderE ndropn-nfir} \text{st ndropn-nlength nfinite-ntaken nfuse-ntaken-ndropn ntaken-nlast}$
 $\text{ntaken-nlength one-enat-def plus-1-eSuc(2)})$
show ?thesis
using 1 3 4 **by** *presburger*
qed
qed

11.3.6 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 $(\text{SStar } X) \cdot Y \subseteq Z$
proof —
have 1: $(\text{SStar } X) \cdot Y = ((\bigcup n. (\text{SPower } (X \cap \text{SMore}) n)) \cdot (\text{SEmpty} \cup (X \cap \text{SMore}) \cap \text{SInf})) \cdot Y$

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

11.3.7 Omega Algebra

lemma *SFMoresem* [*mono*]:
 $x \in (((F \cap SMore) \cap SFinite) \cdot G) \iff$
 $((\exists \sigma 1 \sigma 2. x = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge \sigma 1 \in F \wedge 0 < (\text{nlength } \sigma 1) \wedge \sigma 2 \in G \wedge \text{nlast } \sigma 1 = \text{nfirst } \sigma 2))$

by (simp add: fusion-iff sinfinite-elim smore-elim sfinite-elim) blast

lemma *monoomega* [*mono*]:
 $\text{mono } (\lambda p x. x \in ((F \cap SMore) \cap SFinite \cdot p))$
 by (auto simp add: mono-def sinfinite-elim fusion-iff sfmore-elim)

coinductive-set *somega* :: 'a iintervals \Rightarrow 'a iintervals
for *F* **where**
 $((\exists \sigma 1 \sigma 2. x = \text{nfuse } \sigma 1 \sigma 2 \wedge \text{nfinite } \sigma 1 \wedge \sigma 1 \in F \wedge 0 < (\text{nlength } \sigma 1) \wedge$
 $\sigma 2 \in (\text{somega } F) \wedge \text{nlast } \sigma 1 = \text{nfirst } \sigma 2))$
 $\implies x \in (\text{somega } F)$

lemma *SOmegaIntros*:
 $((F \cap SMore) \cap SFinite) \cdot (\text{somega } F) \subseteq (\text{somega } F)$
using *somega.intros*[of - *F*] *SFMoresem*[of - *F* (*somega F*)]
by (*meson subsetI*)

lemma *SOmegaCases*:
assumes $x \in \text{somega } F$
 $x \in (((F \cap SMore) \cap SFinite) \cdot (\text{somega } F)) \implies P$
shows *P*
using *assms somega.cases*[of *x F P*] *SFMoresem* **by** blast

lemma *SOmegaUnrollsem*:
 $x \in (\text{somega } F) \iff x \in ((F \cap SMore) \cap SFinite) \cdot (\text{somega } F)$
proof *auto*
show $x \in \text{somega } F \implies x \in (F \cap SMore) \cap SFinite \cdot \text{somega } F$
using *SOmegaCases* **by** blast
show $x \in (F \cap SMore) \cap SFinite \cdot \text{somega } F \implies x \in \text{somega } F$
by (*metis* (*no-types*, *lifting*) *SOmegaIntros inf-commute subset-eq*)
 qed

lemma *SOmegaUnroll*:
 $(\text{somega } F) = ((F \cap SMore) \cap SFinite) \cdot (\text{somega } F)$
using *SOmegaUnrollsem* **by** blast

lemma *SOmegaCoinductsem*:
assumes $\bigwedge x. x \in X \implies x \in (((F \cap SMore) \cap SFinite) \cdot (X \cup (\text{somega } F)))$
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*

11.3.8 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*:

$\text{SSkip} \cdot \text{STrue} \subseteq \text{SMore}$

using *subsetI*[of $\text{SSkip} \cdot \text{STrue} \ \text{SMore}$]

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)

11.4 Derived Laws

11.4.1 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:

$$\text{assumes } X \cdot (SStar Y) \cup SEmpty \subseteq (SStar Y)$$

$$\text{shows } (SStar X) \subseteq (SStar Y)$$

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:

$$\text{assumes } ((SStar X) \cdot (SStar X)) \cup SEmpty \subseteq (SStar X)$$

$$\text{shows } (SStar (SStar X)) \subseteq (SStar X)$$

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: <SEmpty \cup SSkip \cdot SStar SSkip = SStar SSkip>*)

have *f2*: $SSkip \cdot -X \subseteq -X \cup SStar\ SSkip \cdot (X \cap (SSkip \cdot -X))$

by (*metis B20 Morgan ST47 UnionIdem <SEmpty \cup SSkip \cdot SStar SSkip = SStar SSkip>*
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: < $\bigwedge Z\ Y. SSkip \cdot (Y \cup Z) = SSkip \cdot Y \cup SSkip \cdot Z$ >*)

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*: $-(-((-X) \cup X1) \cdot (Y \cup (-Y))) \cap (X \cdot Y) \subseteq$
 $-(-((-X) \cup X1) \cdot (Y)) \cap (X \cdot Y)$

using *FusionUnionDistR* **by** *fastforce*

have *3*: $-(-((-X) \cup X1) \cdot (Y)) \cap (X \cdot Y) \subseteq (((-X) \cup X1) \cap X) \cdot Y$

by (*metis (no-types, opaque-lifting) B20 FusionRuleL FusionUnionDistL Morgan UnionCommute*
double-compl order-refl)

have *4*: $(((-X) \cup X1) \cap X) \cdot Y \subseteq X1 \cdot Y$

by (*metis B20 double-compl FusionRuleL inf.right-idem inf-le1*)

from 1 2 3 4 show ?thesis by blast
qed

lemma BD46:

$$(SAlways ((-Y) \cup Y1)) \cap (X1 \cdot Y) \subseteq (X1 \cdot Y1)$$

proof –

$$\text{have 1: } (SAlways ((-Y) \cup Y1)) = -((SFinite \cap (X1 \cup -X1)) \cdot (-((-Y) \cup Y1)))$$

by (simp add: salways-def ssometime-def)

$$\text{have 2: } -((SFinite \cap (X1 \cup -X1)) \cdot (-((-Y) \cup Y1))) = \\ -((SFinite \cap X1) \cup (SFinite \cap -X1)) \cdot (-((-Y) \cup Y1))$$

by (simp add: B28)

$$\text{have 3: } -((SFinite \cap X1) \cup (SFinite \cap -X1)) \cdot (-((-Y) \cup Y1)) = \\ -(SFinite \cap X1) \cdot (-((-Y) \cup Y1)) \cup (SFinite \cap -X1) \cdot (-((-Y) \cup Y1))$$

by (simp add: FusionUnionDistL)

$$\text{have 4: } -(SFinite \cap X1) \cdot (-((-Y) \cup Y1)) \cup (SFinite \cap -X1) \cdot (-((-Y) \cup Y1)) = \\ -((SFinite \cap X1) \cdot (-((-Y) \cup Y1))) \cap -((SFinite \cap -X1) \cdot (-((-Y) \cup Y1)))$$

by blast

$$\text{have 5: } (X1 \cdot Y) = ((SFinite \cap X1) \cdot Y) \cup (SInf \cap X1)$$

by (metis Powerstarhelp2 Powerstarhelp3 UnionCommute inf-commute)

$$\text{have 6: } -((SFinite \cap X1) \cdot (-((-Y) \cup Y1))) = \\ -((SFinite \cap X1) \cdot (Y \cap -Y1))$$

by simp

$$\text{have 7: } -((SFinite \cap X1) \cdot (-((-Y) \cup Y1))) \cap ((SFinite \cap X1) \cdot Y) \subseteq \\ (SFinite \cap X1) \cdot ((-Y) \cup Y1) \cap Y$$

by (metis (no-types) B20 FusionRuleR FusionUnionDistR double-compl inf-sup-aci(1) order-refl)

$$\text{have 8: } (SFinite \cap X1) \cdot ((-Y) \cup Y1) \cap Y \subseteq (SFinite \cap X1) \cdot Y1$$

by (metis B20 FusionRuleR double-compl inf.cobounded1 inf.right-idem)

$$\text{have 9: } (SFinite \cap X1) \cdot Y1 \subseteq X1 \cdot Y1$$

by (simp add: FusionRuleL)

$$\text{have 10: } -((SFinite \cap X1) \cdot (-((-Y) \cup Y1))) \cap (SInf \cap X1) \subseteq ((SFinite \cap X1) \cdot Y1) \cup (SInf \cap X1)$$

by blast

$$\text{have 11: } ((SFinite \cap X1) \cdot Y1) \cup (SInf \cap X1) \subseteq X1 \cdot Y1$$

by (metis B04 Powerstarhelp2 Powerstarhelp3 UnionCommute inf-commute)

$$\text{have 12: } -((SFinite \cap (X1 \cup -X1)) \cdot (-((-Y) \cup Y1))) \cap (X1 \cdot Y) = \\ -((SFinite \cap (X1 \cup -X1)) \cdot (-((-Y) \cup Y1))) \cap ((SFinite \cap X1) \cdot Y) \cup \\ -((SFinite \cap (X1 \cup -X1)) \cdot (-((-Y) \cup Y1))) \cap (SInf \cap X1)$$

by (simp add: 5 B28)

$$\text{have 13: } -((SFinite \cap (X1 \cup -X1)) \cdot (-((-Y) \cup Y1))) \cap ((SFinite \cap X1) \cdot Y) \subseteq X1 \cdot Y1$$

using 7 8 9 2 3 by blast

$$\text{have 14: } -((SFinite \cap (X1 \cup -X1)) \cdot (-((-Y) \cup Y1))) \cap (SInf \cap X1) \subseteq X1 \cdot Y1$$

using 10 11 by blast

$$\text{have 15: } -((SFinite \cap (X1 \cup -X1)) \cdot (-((-Y) \cup Y1))) \cap ((SFinite \cap X1) \cdot Y) \cup \\ -((SFinite \cap (X1 \cup -X1)) \cdot (-((-Y) \cup Y1))) \cap (SInf \cap X1) \subseteq X1 \cdot Y1$$

using 13 14 by blast

show ?thesis

using 1 12 15 by blast

qed

11.4.2 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: $((SNext X) \cap (SNext (\neg X))) = SSkip \cdot (X \cap (\neg X))$

by (metis CH07 SStar16 inf.orderE snext-def)
 have 3: (SSkip)·(X ∩ (−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 ((−X) ∪ X1)) ∩ (SAlways ((−Y) ∪ Y1)) ∩ (X·Y) ⊆ (X1·Y1)
 using BD45 BD46 by blast

lemma SBoxInduct:
 (SAlways (−X ∪ (SWnext X))) ∩ X ⊆ (SAlways X)
proof −
 have 1: ((SAlways (−X ∪ (SWnext X))) ∩ X ⊆ (SAlways X)) =
 ((SSometime (−X)) ⊆ ((−X) ∪ (SSometime (X ∩ (SNext (−X))))))
 by (simp add: salways-def snext-def swnext-def)
 blast
 have 2: ((SSometime (−X)) ⊆ ((−X) ∪ (SSometime (X ∩ (SNext (−X)))))) =
 (((SStar SSkip)·(−X)) ⊆ ((−X) ∪ ((SStar SSkip)·(X ∩ (SSkip·(−X))))))
 by (simp add: SStarSkip snext-def ssometime-def)
 have 3: (((SStar SSkip)·(−X)) ⊆ ((−X) ∪ ((SStar SSkip)·(X ∩ (SSkip·(−X))))))
 using SStar51 SStar50 by blast
 from 1 2 3 show ?thesis by auto
 qed

lemma SChopstarEqv:
 (SStar X) = SEmpty ∪ (X ∩ SMore)·(SStar X)
 using SStar52 SStar53 by blast

11.5 Extra Laws

11.5.1 Boolean Laws

lemma B02:
 assumes −Y ⊆ −X
 shows (X:: 'a iintervals) ⊆ Y
 using assms by auto

lemma B03:
 ((X:: 'a iintervals) = Y) ⟷ (−X = −Y)
 by auto

lemma B05:
 assumes (X:: 'a iintervals) ∪ Y ⊆ Z
 shows X ⊆ Z ∧ Y ⊆ Z
 using assms by auto

lemma B06:
 assumes X ⊆ Z ∧ Y ⊆ 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*

11.5.2 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

11.5.3 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 *sprev-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 swprev-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 spreved swprev-def)
 have 2: $((SPrev X) \cap (SPrev (-X))) = (X \cap (-X)) \cdot SSkip$
 by (metis CH08 SStar16 inf.orderE spreved)
 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 UnionCommutate compl-le-swap1 smore-def sup-ge2 swprev-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 –
have 1: $SMore \subseteq SSkip \cdot STrue$
by (simp add: SMoreImpSSkipFusion)
have 2: $SSkip \cdot STrue \subseteq STrue \cdot SSkip$
proof –
have f3: $SSkip \cdot SFinite \subseteq STrue \cdot SSkip$
by (metis B19 FusionRuleL SStar19 SStarSkip STrueTop inf-sup-ord(2))
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 –
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 –
have 1: $((SEmpty) \subseteq (SWnext X)) = (-(SWnext X) \subseteq SMore)$
by (simp add: smore-def)
have 2: $-(SWnext X) \subseteq SMore = (SNext (-X)) \subseteq SMore$
by (simp add: snext-def swnext-def)
have 3: $(SNext (-X)) \subseteq SSkip \cdot STrue$
by (metis FusionUnionDistR STrueTop snext-def sup.orderI sup.right-idem)
hence 4: $(SNext (-X)) \subseteq SMore$ **using** SSkipFusionImpSMore **by** auto
from 1 2 4 **show** ?thesis **by** auto
qed

lemma N21:

$$(SEmpty \cup (SNext X)) \subseteq (SWnext X)$$

using N20

by (metis B06 SNextImpNotNextNot snext-def sunext-def)

lemma N22:

$$(SWnext X) = (SEmpty \cup (SNext X))$$

using N21 N19 **by** blast

lemma N16:

$$(SNext X) = SMore \cap (SWnext X)$$

using N12 N22 smore-def **by** blast

lemma N17:

$$(SWnext (X \cap Y)) = (SWnext X) \cap (SWnext Y)$$

by (simp add: N05 N22 Un-Int-distrib)

lemma N18:

$$(SWnext (X \cup Y)) = (SWnext X) \cup (SWnext Y)$$

by (simp add: sunext-def)

(metis CH07 SSkipSFinite compl-inf)

lemma N27:

$$(SNext ((-X) \cup Y)) \subseteq (-(SNext X) \cup (SNext Y))$$

using N04 SNextImpNotNextNot **by** blast

lemma N28:

$$(SPrev ((-X) \cup Y)) \subseteq (-(SPrev X) \cup (SPrev Y))$$

unfolding sprev-def

proof –

have $\bigwedge I. (SEmpty) \cup I \cdot SSkip = - (- I \cdot SSkip)$

using N26 sprev-def swprev-def **by** blast

then have $(Y \cup - X) \cdot SSkip \subseteq Y \cdot SSkip \cup - (X \cdot SSkip)$

using FusionUnionDistL **by** blast

then show $(- X \cup Y) \cdot SSkip \subseteq - (X \cdot SSkip) \cup Y \cdot SSkip$

by (simp add: UnionCommute)

qed

lemma N29:

$$(SPrev X) = -(SWprev (-X))$$

by (simp add: sprev-def swprev-def)

11.5.4 SInit

lemma ST01:

$$(X \cap SEmpty) \cdot Y \subseteq Y$$

by (metis Int-commute FusionUnionDistL FusionSEmptyL le-iff-sup sup-inf-absorb UnionCommute)

lemma ST02:

$(X \cap SEmpty) \cdot Y \subseteq (X \cap SEmpty) \cdot STrue$
by (*simp add: FusionRuleR strue-def*)

lemma ST03:
 $(X \cap SEmpty) \cdot (X \cap SEmpty) \subseteq (X \cap SEmpty)$
using ST01 **by** *auto*

lemma ST04:
 $(X \cap SEmpty) \subseteq (X \cap SEmpty) \cdot (X \cap SEmpty)$
proof –
have $\forall S Sa Sb Sc. (Sc) \cdot (Sb \cap SEmpty) \cap (Sa \cdot (S \cap SEmpty)) = Sc \cap Sa \cdot (Sb \cap (S \cap SEmpty))$
by (*simp add: CH10 inf-assoc*)
then have $\forall S Sa. (Sa) \cdot (S \cap SEmpty) \cdot (S \cap SEmpty) = Sa \cdot (S \cap SEmpty)$
by (*metis FusionSEmptyR inf.idem inf-commute*)
then show *?thesis*
by (*metis FusionSEmptyL subset-refl*)
qed

lemma ST05:
 $(X \cap SEmpty) \subseteq -((-X) \cap SEmpty)$
by *blast*

lemma ST06:
 $((-X) \cap SEmpty) \subseteq -(X \cap SEmpty)$
by *auto*

lemma ST07:
 $(X \cap SEmpty) \cap (Y \cap SEmpty) \subseteq (X \cap SEmpty) \cdot STrue$
using ST02 *FusionSEmptyR* **by** *blast*

lemma ST08:
 $(X \cap SEmpty) \cap (Y \cap SEmpty) \subseteq (STrue \cap SEmpty) \cdot (Y \cap SEmpty)$
by (*metis FusionSEmptyL FusionSEmptyR ST33 inf.cobounded2*)

lemma ST09:
 $((X \cap SEmpty) \cdot STrue) \cap (STrue \cap SEmpty) \cdot (Y \cap SEmpty) \subseteq (X \cap SEmpty) \cdot (Y \cap SEmpty)$
by (*metis compl-bot-eq eq-refl FusionSEmptyR inf.commute inf-top.left-neutral CH09 strue-def*)

lemma ST10:
 $(X \cap SEmpty) \cdot (Y \cap SEmpty) \subseteq (X \cap SEmpty)$
by (*metis FusionRuleR FusionSEmptyR inf-le2*)

lemma ST11:
 $(X \cap SEmpty) \cdot (Y \cap SEmpty) \subseteq (Y \cap SEmpty)$
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:

$$-((X \cap SEmpty) \cup (Y \cap SEmpty)) = -(X \cap SEmpty) \cap -(Y \cap SEmpty)$$

by *auto*

lemma ST19:

$$(X \cap SEmpty) \cdot ((-X) \cap SEmpty) \subseteq (X \cap SEmpty)$$

using ST10 **by** *blast*

lemma ST20:

$$(X \cap SEmpty) \cdot ((-X) \cap SEmpty) \subseteq ((-X) \cap SEmpty)$$

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 pur-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*:
 $\neg(STrue \cdot (X \cap SEmpty)) \subseteq STrue \cdot ((-X) \cap SEmpty)$
by (*metis B09 FusionSEmptyR FusionUnionDistR ST21 double-compl*)

11.5.5 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*

11.5.6 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*:

$(S\text{Bi } S\text{True}) = S\text{True}$

by (*simp add: SBiGen*)

lemma *BD16*:

$(S\text{Di } (X \cup Y)) = (S\text{Di } X) \cup (S\text{Di } Y)$

by (*simp add: FusionUnionDistL sdi-def*)

lemma *BD17*:

assumes $X \subseteq Y$

shows $(S\text{Di } X) \subseteq (S\text{Di } Y)$

using *assms*

by (*metis FusionUnionDistL Subsumption sdi-def*)

lemma *BD18*:

$(S\text{Di } (S\text{Di } X)) = (S\text{Di } X)$

by (*metis CH01 FusionAssoc sdi-def*)

lemma *BD19*:

$(S\text{Da } S\text{Empty}) = S\text{True}$

by (*metis BD01 BD06 sda-def ssometime-def*)

lemma *BD20*:

$(S\text{Da } S\text{True}) = S\text{True}$

by (*metis BD06 CH01 sda-def ssometime-def*)

lemma *BD21*:

$(S\text{Ba } S\text{True}) = S\text{True}$

by (*metis BD15 BD08 BD09 sba-def sbi-def sda-def sdi-def ssometime-def*)

lemma *BD22*:

$(S\text{Da } (X \cup Y)) = (S\text{Da } X) \cup (S\text{Da } Y)$

by (*simp add: FusionUnionDistL FusionUnionDistR sda-def*)

lemma *BD23*:

assumes $X \subseteq Y$

shows $(S\text{Da } X) \subseteq (S\text{Da } Y)$

using *assms*

by (*metis BD22 Subsumption*)

lemma *BD24*:

assumes $X \subseteq Y$

shows $(S\text{Da } (-Y)) \subseteq (S\text{Da } (-X))$

using *assms*

by (*simp add: BD23*)

lemma *BD25*:

$(S\text{Di } X) \subseteq (S\text{Da } X)$

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:

$(SAways ((-X) \cup Y)) \subseteq ((- (SAways X)) \cup (SAways Y))$

by (metis B20 BD12 BD29 BD38 BD42 double-compl)

lemma BD48:

$(SAways X) \subseteq X \cap (SWnext (SAways X))$

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:

$(SBi X) \subseteq X \cap (SWprev (SBi X))$

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:

$(SAways (SMore \cup X)) = STrue \cdot (X \cap SEmpty)$

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: $(SAways (SMore \cup X)) \cap SFinite \subseteq SFinite \cdot (X \cap SEmpty)$

using 11

by (simp add: B09 BD09 FusionUnionDistR UnionCommute inf-commute inf-sup-aci(7))

$\text{smore-def ssometime-def sfinite-def}$
have 3: $(SAlways (SMore \cup X)) \subseteq ((SFinite.(X \cap SEmpty)) \cup SInf)$
using 2 sfinite-def **by** fastforce
have 4: $(SAlways (SMore \cup X)) \subseteq STrue.(X \cap SEmpty)$
using 1 3 **by** blast
have 5: $SFinite.(X \cap SEmpty) \subseteq (SAlways (SMore \cup X))$
unfolding $\text{salways-def ssometime-def smore-def}$
using $\text{CH10[of SFinite X SFinite] FusionSFalse}$
by ($\text{metis (no-types, opaque-lifting) SFalseBottom compl-sup disjoint-eq-subset-Compl double-compl inf-commute inf-le2 sfinite-def}$)
have 6: $SInf \subseteq (SAlways (SMore \cup X))$
by ($\text{metis B05 BD30 FusionSEmptyR double-compl salways-def sfinite-def smore-def ssometime-def}$)
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

11.5.7 Finite and Infinite

lemma FI01 :
 $SFinite \cdot SFinite = SFinite$
by ($\text{metis BD01 BD11 ssometime-def}$)

lemma FI02 :
 $(X \cap SFinite) \cdot (Y \cap SFinite) \subseteq (X \cdot Y) \cap SFinite$
by ($\text{metis B16 CH06 FI01 inf.cobounded1 inf-le2}$)

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: $SFinite \cap SMore \subseteq SFinite \cdot SSkip$
by (*metis B20 FI01 FI02 SStar19 SStarSkip UnfoldL inf.idem smore-def*)
show *?thesis* **using** 1 2 **by** *blast*
qed

lemma *FI10*:
 $SFinite \cdot SSkip = SSkip \cdot SFinite$

by (*metis SStar19 SStarSkip*)

lemma FI11:

$SFinite \cap SEmpty = SEmpty$

by (*simp add: FI06 Int-absorb2 inf-sup-aci(1)*)

lemma FI12:

$SInf \cdot SInf = SInf$

by (*metis FusionSFalse Powerstarhelp2 sinf-def*)

lemma FI13:

$SFinite \cdot SInf = SInf$

by (*metis BD06 FusionAssoc1 sinf-def ssometime-def*)

lemma FI14:

$SInf \cdot SFinite = SInf$

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:

$((F \cdot STrue) \cap SFinite) = (F \cap SFinite) \cdot SFinite$

by (*metis FI04 boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def*)

lemma FI18:

$((STrue \cdot F) \cap SFinite) = SFinite \cdot (F \cap SFinite)$

by (*metis BD19 FI01 FI04 FI17 FusionSEmptyR inf.idem sda-def*)

lemma FI19:

$SFinite \cdot SMore = SMore$

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:

$$((F \cap SMore) \cap SFinite) = (F \cap SMore)$$

by (*metis inf-assoc inf-commute sfmore-def*)

lemma FI30:

$$(SEmpty \cap SMore) = SFalse$$

by (*simp add: sfalse-def sfmore-def smore-def*)

lemma FI31:

$$((F \cap SInf) \cap SMore) = SFalse$$

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:

$$\text{assumes } F \subseteq SNext F$$

```

shows     $SFinite \subseteq \neg F$ 
using assms
by (metis B01 FusionSEmptyR N12 N22 SStarInductL SStarSkip double-compl snext-def)

lemma FI36:
assumes  $F \cap \neg G \subseteq SNext\ F$ 
shows     $F \cap SFinite \subseteq SSometime\ G$ 
using assms
proof -
have 1:  $F \cap \neg G \subseteq SNext\ F$ 
using assms by auto
have 2:  $F \cap \neg G \cap SAlways\ (\neg G) \subseteq SNext\ F \cap SAlways\ (\neg G)$ 
using assms by blast
have 3:  $SAlways\ (\neg G) \subseteq \neg G$ 
by (simp add: BD12)
have 4:  $SAlways\ (\neg G) = SAlways\ (\neg G) \cap \neg G$ 
using 3 by blast
have 5:  $F \cap SAlways\ (\neg G) \subseteq SNext\ F \cap SAlways\ (\neg G)$ 
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:  $SAlways\ (\neg G) = \neg G \cap SWnext\ (SAlways\ (\neg G))$ 
using 51 by (simp add: salways-def ssometime-def swnext-def) blast
have 7:  $SNext\ F \cap SAlways\ (\neg G) \subseteq SNext\ F \cap SWnext\ (SAlways\ (\neg G))$ 
using 6 by blast
have 8:  $F \cap SAlways\ (\neg G) \subseteq SNext\ F \cap SWnext\ (SAlways\ (\neg G))$ 
using 5 7 by blast
have 9:  $F \cap SAlways\ (\neg G) \subseteq SMore \cap SWnext\ F \cap SWnext\ (SAlways\ (\neg G))$ 
using 8 N16 by blast
have 10:  $F \cap SAlways\ (\neg G) \subseteq SWnext\ F \cap SWnext\ (SAlways\ (\neg G))$ 
using 8 N16 by fastforce
have 11:  $F \cap SAlways\ (\neg G) \subseteq SWnext\ (F \cap SAlways\ (\neg G))$ 
using 10 N17 by blast
have 12:  $SAlways\ (F \cap SAlways\ (\neg G)) \subseteq SWnext\ (F \cap SAlways\ (\neg G))$ 
by (metis 10 B15 BD12 N17 inf.absorb-iff2)
have 13:  $SAlways\ (F \cap SAlways\ (\neg G)) \subseteq$ 
 $SWnext\ (F \cap SAlways\ (\neg G)) \cap F \cap (\neg(SAlways\ (\neg G)) \cup SAlways\ (F \cap SAlways\ (\neg G)))$ 
using 12 BD12 by auto
have 14:  $(F \cap SAlways\ (\neg G)) \subseteq SAlways\ (F \cap SAlways\ (\neg G))$ 
using 12 13
by (metis 10 B08 BD08 N17 SBoxInduct boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def)
have 15:  $SAlways\ (F \cap SAlways\ (\neg G)) \subseteq (F \cap SAlways\ (\neg G))$ 
using BD12 by blast
have 16:  $SAlways\ (F \cap SAlways\ (\neg G)) = (F \cap SAlways\ (\neg G))$ 
using 14 15 by blast
have 17:  $(F \cap SAlways\ (\neg G)) \subseteq SMore$ 
using 9 by blast
have 18:  $SAlways\ (F \cap SAlways\ (\neg G)) \subseteq SAlways\ SMore$ 
by (simp add: 17 BD38)
have 19:  $SFinite = \neg(SAlways\ SMore)$ 

```

by (simp add: BD01 salways-def smore-def)
 have 20: $SFinite \subseteq \neg(F \cap SAlways \neg G)$
 using 16 18 19 by blast
 have 21: $SFinite \subseteq \neg F \cup \neg(SAlways \neg G)$
 using 20 by blast
 have 22: $\neg(SAlways \neg G) = SSometime G$
 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 $F \cap \neg G$]
 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: $\neg F \subseteq SNext \neg F$
 using N12 N19 assms(1) assms(2) by blast
 from 1 show ?thesis using FI35 by blast
 qed

lemma FI41:

assumes $S\text{Empty} \cap F \subseteq G$
 $S\text{Next} (-F \cup G) \cap F \subseteq G$
shows $F \cap S\text{Finite} \subseteq G$
proof –
have 1: $(F \cap -G) \subseteq S\text{Next} (F \cap -G)$
using $N19[\text{of } (-F \cup G)]$
using $\text{assms}(1) \text{ assms}(2) \text{ snext-def swnext-def}$ **by** fastforce
have 2: $S\text{Finite} \subseteq - (F \cap -G)$
using 1 $FI35$ **by** auto
from 2 **show** $?thesis$ **by** blast
qed

lemma $FI42$:
assumes $F \subseteq S\text{More} \cdot F$
shows $S\text{Finite} \subseteq -F$
proof –
have 1: $SS\text{ sometime } F \subseteq S\text{Next} (SS\text{ sometime } F)$
by $(\text{simp add: ssometime-def snext-def})$
 $(\text{metis } FI21 \text{ SChopAssoc } S\text{StarSkip } ST47 \text{ UnfoldL assms order-refl subset-Un-eq})$
have 2: $S\text{Finite} \subseteq -(SS\text{ sometime } F)$
by $(\text{simp add: } 1 \text{ } FI35)$
show $?thesis$
by $(\text{metis } 2 \text{ } B01 \text{ } FI21 \text{ } S\text{StarSkip } ST47 \text{ UnfoldL assms le-iff-sup ssometime-def subset-trans})$
qed

lemma $FI43$:
assumes $F \cap -G \subseteq S\text{More} \cdot (F \cap -G)$
shows $F \cap S\text{Finite} \subseteq G$
proof –
have 1: $S\text{Finite} \subseteq -(F \cap -G)$
using $FI42 \text{ assms}$ **by** blast
from 1 **show** $?thesis$ **by** blast
qed

lemma $FI44$:
assumes $F \cap S\text{Finite} \subseteq S\text{More} \cdot F$
shows $S\text{Finite} \subseteq -F$
proof –
have 1: $F \cap S\text{Finite} \subseteq S\text{More} \cdot (F \cap S\text{Finite})$
by $(\text{metis } FI04 \text{ } FI21 \text{ } FI22 \text{ } S\text{SkipSFinite assms inf.idem})$
show $?thesis$
by $(\text{metis } 1 \text{ } B01 \text{ } B11 \text{ } B12 \text{ } FI43 \text{ inf.right-idem sfinite-def})$
qed

lemma $FI45$:
assumes $F \cap S\text{Finite} \subseteq S\text{More} \cdot F$
shows $S\text{Finite} \subseteq -F$
proof –
have 1: $F \cap S\text{Finite} \subseteq S\text{More} \cdot (F \cap S\text{Finite})$
by $(\text{metis } FI04 \text{ } FI22 \text{ assms inf-commute sfmore-def})$

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: $F \cap SFinite \subseteq SFMore \cdot (F \cap SFinite)$
 using B11 FI45 assms **by** auto
show ?thesis
 by (metis 1 B01 B11 B12 FI43 inf.right-idem sfinite-def)
qed

lemma FI47:
assumes $(F \cap -G) \cap SFinite \subseteq SFMore \cdot (F \cap -G)$
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 $F \subseteq G \cdot F$
 $G \subseteq SFMore$
shows $SFinite \subseteq -F$
proof –
have 1: $G \cdot F \subseteq SFMore \cdot F$
 by (simp add: FusionRuleL assms(2))
have 2: $F \subseteq SFMore \cdot F$
 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*)

11.5.8 Omega

lemma OA01:
 $(somega SSkip) = SSkip \cdot (somega SSkip)$
by (*metis SOmegaUnroll SSkipSFinite SSkipSMore*)

lemma OA02:
 $(somega SEmpty) = SFalse$
by (*metis FI30 SFalseFusion SOmegaUnroll inf-assoc inf-commute sfmore-def*)

lemma OA03:
 $(somega SFalse) = SFalse$

by (*metis Compl-disjoint2 Powerstarhelp2 SOmegaUnroll inf-assoc inf-compl-bot-right sfalse-def*)

lemma *OA04*:

$(\text{somega } SInf) = SFalse$

by (*metis FI25 SFalseBottom SFalseFusion SOmegaUnroll inf-commute sfinite-def*)

lemma *BD59*:

$$SInf \cap G \cap SAlways (-G \cup (((F \cap SMore) \cap SFinite) \cdot G)) \subseteq$$

$$((F \cap SMore) \cap SFinite) \cdot (SInf \cap G \cap SAlways (-G \cup (((F \cap SMore) \cap SFinite) \cdot G)))$$

proof –

have 1: $SInf \cap SAlways (-G \cup (((F \cap SMore) \cap SFinite) \cdot G)) \subseteq (-G \cup (((F \cap SMore) \cap SFinite) \cdot G))$
using *BD12* **by** *blast*

have 2: $SInf \cap G \cap SAlways (-G \cup (((F \cap SMore) \cap SFinite) \cdot G)) \subseteq$
 $SInf \cap (((F \cap SMore) \cap SFinite) \cdot G) \cap SAlways (-G \cup (((F \cap SMore) \cap SFinite) \cdot G))$
using 1 **by** *blast*

have 3: $SInf \cap (((F \cap SMore) \cap SFinite) \cdot G) \cap SAlways (-G \cup (((F \cap SMore) \cap SFinite) \cdot G)) \subseteq$
 $((F \cap SMore) \cap SFinite) \cdot (SInf \cap G \cap SAlways (-G \cup (((F \cap SMore) \cap SFinite) \cdot G)))$

by (*metis BD58 FI23 inf-commute*)

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

qed

lemma *OA06*:

$SInf \cap G \cap SAlways (-G \cup (((F \cap SMore) \cap SFinite) \cdot G)) \subseteq (\text{somega } F)$

by (*metis (no-types, lifting) BD59 SOmegaWeakCoinduct*)

lemma *FI54*:

$SAlways SFMore \subseteq SInf$

by (*metis BD38 BD56 Compl-disjoint2 SMoreImpSSkipFusion SSkipFusionImpSMore*
inf-le2 sfalse-def sfmore-def sinf-def smore-def subset-antisym sup.orderE)

lemma *FI55*:

$SAlways SFinite = SFinite$

using *FI13 FI15 salways-def ssometime-def* **by** *fastforce*

lemma *FI56*:

$(SAlways SFMore) = SFalse$

using *BD12 FI31 FI54* **by** *blast*

lemma *OA07*:

$(\text{somega } ((SPower SSkip (Suc n)))) \subseteq SInf$

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: $SInf \cap STrue \cap SAlways (-STrue \cup (((SPower SSkip (Suc n)) \cap SMore) \cap SFinite) \cdot STrue)) \subseteq$
 $(\text{somega } (SPower SSkip (Suc n)))$

by (*simp add: OA06*)

have 2: $- STrue \cup (SPower SSkip (Suc n) \cap SMore) \cap SFinite \cdot STrue =$

```

    (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)  $\cdot$  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:  $SPower (SSkip) (Suc (Suc n)) \cap SMore \cap SFinite = SSkip \cap SFinite \cdot SPower SSkip$ 
     $(Suc n)$ 
    proof –
    have  $(SSkip) \cap SFinite \cdot SPower SSkip (Suc n) \subseteq SFinite$ 
      using pwr-Suc spower-finite by blast
    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
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*:

$SInf \subseteq (\text{somega } STrue)$

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 SMoreImpSSkipFusion 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: $SInf \cap STrue \cap (SAlways (SFalse \cup ((STrue \cap SMore) \cap SFinite) \cdot STrue)) \subseteq (\text{somega } STrue)$

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*:

$(\text{somega } STrue) = SInf$

by (*simp add: B04 OA11 OA12*)

lemma *OA14*:

$(\text{somega } SMore) \subseteq SInf$

by (*metis Int-absorb1 Int-lower2 OA13 SOmegaUnroll SOmegaWeakCoinduct compl-bot-eq inf-top-left strue-def*)

lemma *OA15*:

$SInf \subseteq (\text{somega } SMore)$

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 SSkipFusionImpSMore*

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: $SInf \cap STrue \cap (SAlways (SFalse \cup ((SMore \cap SMore) \cap SFinite) \cdot STrue)) \subseteq (\text{somega } STrue)$

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*:

$(\text{somega } SMore) = SInf$
using OA14 OA15 **by** blast

lemma OA17:

$(\text{somega } SFinite) \subseteq SInf$
by (metis FI07 FI15 FI21 FI42 SOmegaUnroll compl-le-swap1 dual-order.refl inf.orderE sfmore-def)

lemma OA18:

$SInf \subseteq (\text{somega } SFinite)$
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: $SInf \cap STrue \cap (SAlways (SFalse \cup ((SFinite \cap SMore) \cap SFinite) \cdot STrue)) \subseteq (\text{somega } STrue)$
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:

$(\text{somega } SFinite) = SInf$
by (simp add: B04 OA17 OA18)

lemma OA20:

assumes $H \subseteq ((F \cap SMore) \cap SFinite) \cdot H$
shows $H \cap SInf \subseteq (\text{somega } F)$
using assms
using SOmegaWeakCoinduct **by** blast

lemma OA21:

assumes $F \subseteq G$
shows $(\text{somega } F) \cap SInf \subseteq (\text{somega } G)$
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 –

have 1: $(SBA F) = (SBA (SBA F))$

using *BD44* **by** *blast*

have 2: $(SBa (SBa F)) \cap (G \cdot G1) \subseteq G \cdot (SBa F \cap G1)$
using *BD28 BD61 by blast*
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 –
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)*
have 2: $(-F \cup G) \cap ((F \cap SMore) \cap SFinite) \subseteq ((G \cap SMore) \cap SFinite)$
by *auto*
have 3: $(-F \cup G) \cap (somega F) \subseteq (somega F)$
by *auto*
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*
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*
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*
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*
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 –
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)*
have 2: $SBa (-F \cup G) \cap SInf \subseteq (-(somega F) \cup (somega G))$
by *(simp add: B19 OA25)*
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)

11.6 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*:

$$-(\text{lan } f) = (\text{lan } (\text{LIFT}(\neg f)))$$

using *interval-lan lan-def* by *fastforce*

lemma *inter-and*:

$$(\sigma \in ((\text{lan } f) \cap (\text{lan } g))) \longleftrightarrow (\sigma \models f \wedge g)$$

by (*simp add: lan-def*)

lemma *inter-and-1*:

$$((\text{lan } f) \cap (\text{lan } g)) = (\text{lan } (\text{LIFT}(f \wedge g)))$$

using *inter-and lan-def* by *fastforce*

lemma *union-or*:

$$(\sigma \in ((\text{lan } f) \cup (\text{lan } g))) \longleftrightarrow (\sigma \models f \vee g)$$

by (*simp add: lan-def*)

lemma *union-or-1*:

$$((\text{lan } f) \cup (\text{lan } g)) = (\text{lan } (\text{LIFT}(f \vee g)))$$

using *union-or lan-def* by *fastforce*

lemma *subset-impl*:

$$(\sigma \in (-(\text{lan } f) \cup (\text{lan } g))) \longleftrightarrow (\sigma \models f \longrightarrow g)$$

by *simp*

lemma *subset-impl-1*:

$$(-(\text{lan } f) \cup (\text{lan } g)) = (\text{lan } (\text{LIFT}(f \longrightarrow g)))$$

using *subset-impl lan-def* by *fastforce*

lemma *fusion-chop*:

$$(\sigma \in ((\text{lan } f) \cdot (\text{lan } g))) \longleftrightarrow (\sigma \models f;g)$$

by (*auto simp add: fusion-iff chop-nfuse*)

lemma *fusion-chop-1*:

$$((\text{lan } f) \cdot (\text{lan } g)) = (\text{lan } (\text{LIFT}(f;g)))$$

using *fusion-chop lan-def* by *blast*

lemma *empty-empty*:

$$\sigma \in S\text{Empty} \longleftrightarrow (\sigma \models \text{empty})$$

by (*simp add: itl-defs empty-elim zero-enat-def*)

lemma *empty-empty-1*:

$$S\text{Empty} = (\text{lan } (\text{LIFT } \text{empty}))$$

using *empty-empty lan-def* by *fastforce*

lemma *smore-more*:

$$\sigma \in S\text{More} \longleftrightarrow (\sigma \models \text{more})$$

using *zero-enat-def* by (*auto simp add: itl-defs empty-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*:

$\sigma \in SFinite \longleftrightarrow (\sigma \models finite)$

by (*simp add: sfinite-def sinf-def finite-d-def infinite-d-def fusion-chop sfalse-false-1 strue-true-1*)

lemma *sfinite-1*:

$SFinite = \text{lan}(LIFT(finite))$

using *sfinite lan-def* **by** *fastforce*

lemma *and-inter-finite*:

$\sigma \in (((\text{lan } f) \cap SFinite)) \longleftrightarrow (\sigma \models (f \wedge finite))$

using *sfinite inter-and* **by** *auto*

lemma *and-inter-finite-1*:

$((\text{lan } f) \cap SFinite) = \text{lan}(LIFT(f \wedge finite))$

by (*simp add: inter-and-1 sfinite-1*)

lemma *and-inter-more*:

$\sigma \in (((\text{lan } f) \cap SMore)) \longleftrightarrow (\sigma \models (f \wedge more))$

using *smore-more inter-and* **by** *auto*

lemma *and-inter-more-1*:

$\sigma \in (((\text{lan } f) \cap SMore)) \longleftrightarrow (\sigma \in (\text{lan}(LIFT(f \wedge more))))$

using *and-inter-more lan-def* **by** (*simp add: smore-more-1*)

lemma *and-inter-more-2*:

$((\text{lan } f) \cap SMore) = (\text{lan}(LIFT(f \wedge more)))$

using *and-inter-more-1* **by** *blast*

lemma *and-chop*:

$\sigma \in (((\text{lan } f) \cap SMore) \cdot (\text{lan } g)) \longleftrightarrow (\sigma \models (f \wedge more);g)$

by (*metis fusion-chop inter-and-1 smore-more-1*)

lemma *and-chop-1*:

$((\text{lan } f) \cap SMore) \cdot (\text{lan } g) = (\text{lan}(LIFT((f \wedge more);g)))$

using *and-chop lan-def* **by** *blast*

lemma *spower-chop-power*:

$(SPower(\text{lan } f) \ n) = (\text{lan}(LIFT(fpower \ f \ n)))$

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 *spowerstar*:

$\sigma \in SPowerstar\ (lan\ f) \longleftrightarrow \sigma \in SFPowerstar\ (lan\ (f)) \cdot (SEmpty \cup ((lan\ f) \cap SInf))$
by (*simp add: spowerstar-def*)

lemma *sstar-spowerstar*:

$\sigma \in SStar\ (lan\ f) \longleftrightarrow \sigma \in SPowerstar\ ((lan\ f) \cap SMore)$
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: spower-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-spowerstar* **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: spowerstar-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 *spower-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 sempty-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*:

$(\text{lan } (LIFT(f^*)) = (\text{lan } (LIFT(\text{empty} \vee (f \wedge \text{more}); f^*)))$

using *chopstar-seqv lan-def* **by** *blast*

lemma *sinf*:

$\sigma \in SInf \longleftrightarrow (\sigma \models \text{inf})$

by (*simp add: fusion-chop infinite-d-def sfalse-false-1 sinf-def strue-true-1*)

lemma *sinf-1*:

$SInf = \text{lan}(LIFT(\text{inf}))$

using *sinf* **by** *fastforce*

lemma *fmore*:

$\sigma \in SFMore \longleftrightarrow (\sigma \models \text{fmore})$

by (*metis fmore-d-def inf-commute inter-and-1 interval-lan sfinite-1 sfmore-def smore-more-1*)

lemma *fmore-1*:

$SFMore = \text{lan}(LIFT(\text{fmore}))$

using *fmore* **by** *fastforce*

lemma *omega-induct-sem*:

$x \in -(\text{lan } g) \cup (((\text{lan } f) \cap SFMore) \cap SFFinite) \cdot (\text{lan } g) \longleftrightarrow$

$(x \models g \longrightarrow ((f \wedge \text{more}) \wedge \text{finite}); g)$

by (*simp add: fusion-chop-1 inter-and-1 sfinite-1 smore-more-1*)

lemma *omega-induct*:

$-(\text{lan } g) \cup (((\text{lan } f) \cap SFMore) \cap SFFinite) \cdot (\text{lan } g) =$

$\text{lan}(LIFT(g \longrightarrow ((f \wedge \text{more}) \wedge \text{finite}); g))$

using *omega-induct-sem[of - g f]* **by** *fastforce*

lemma *somega-omega-sem-1*:

assumes $x \models f^\omega$

shows $x \in \text{somega } (\text{lan } f)$

proof –

have 1: $x \models f^\omega \longrightarrow ((f \wedge \text{more}) \wedge \text{finite}); f^\omega$

by (*metis OmegaUnroll intD int-iffD1 inteq-reflection*)

have 2: $x \in -(\text{lan } (LIFT(f^\omega))) \cup (((\text{lan } f) \cap SFMore) \cap SFFinite) \cdot (\text{lan } (LIFT(f^\omega)))$

using 1 *omega-induct-sem* **by** *blast*

have 3: $\bigwedge x . x \in (\text{lan } (LIFT(f^\omega))) \implies x \in ((((\text{lan } f) \cap SFMore) \cap SFFinite) \cdot (\text{lan } (LIFT(f^\omega))))$

by (*metis OmegaUnroll fusion-chop-1 inteq-reflection inter-and-1 sfinite-1 smore-more-1*)

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 (λ x. x ∈ somega (lan f)) f]
  by (metis Int-iff Prop10 intI integ-reflection inter-and-1 interval-lan)
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

12 Until operator for Finite and Infinite Intervals

```

theory Until
imports Semantics SChopTheorems
begin

```

This theory introduces the weak and strong versions of the until operator. The theorems from [11] are proven in a mostly deductive style.

12.1 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      ((- U -) [84,84] 83)

```

syntax (*ASCII*)

$-until-d \quad :: [lift, lift] \Rightarrow lift \quad ((- \text{ until } -) [84, 84] \ 83)$

translations

$-until-d \quad \Rightarrow CONST \text{ until-}d$

definition $suntil-d :: ('a :: world) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $suntil-d \ F \ G \equiv LIFT(\bigcirc(F \ \mathcal{U} \ G))$

definition $wait-d :: ('a :: world) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $wait-d \ F \ G \equiv LIFT(\Box \ F \vee F \ \mathcal{U} \ G)$

definition $release-d :: ('a :: world) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $release-d \ F \ G \equiv LIFT(\neg((\neg F) \ \mathcal{U} \ (\neg G)))$

syntax

$-wait-d \quad :: [lift, lift] \Rightarrow lift \quad ((- \ \mathcal{W} \ -) [84, 84] \ 83)$
 $-release-d \quad :: [lift, lift] \Rightarrow lift \quad ((- \ \mathcal{R} \ -) [84, 84] \ 83)$
 $-suntil-d \quad :: [lift, lift] \Rightarrow lift \quad ((- \ \mathcal{U}^s \ -) [84, 84] \ 83)$

syntax (*ASCII*)

$-wait-d \quad :: [lift, lift] \Rightarrow lift \quad ((- \text{ wait } -) [84, 84] \ 83)$
 $-release-d \quad :: [lift, lift] \Rightarrow lift \quad ((- \text{ release } -) [84, 84] \ 83)$
 $-suntil-d \quad :: [lift, lift] \Rightarrow lift \quad ((- \text{ suntil } -) [84, 84] \ 83)$

translations

$-wait-d \quad \Rightarrow CONST \text{ wait-}d$
 $-release-d \quad \Rightarrow CONST \text{ release-}d$
 $-suntil-d \quad \Rightarrow CONST \text{ suntil-}d$

definition $srelease-d :: ('a :: world) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $srelease-d \ F \ G \equiv LIFT(\neg((\neg F) \ \mathcal{W} \ (\neg G)))$

syntax

$-srelease-d \quad :: [lift, lift] \Rightarrow lift \quad ((- \ \mathcal{M} \ -) [84, 84] \ 83)$

syntax (*ASCII*)

$-srelease-d \quad :: [lift, lift] \Rightarrow lift \quad ((- \text{ srelease } -) [84, 84] \ 83)$

translations

$-srelease-d \quad \Rightarrow CONST \text{ srelease-}d$

12.2 Axioms

12.2.1 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 iless-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 < nlength σ
using 2 by auto
have 4: k ≤ nlength (ndropn (Suc 0) σ)
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 ( (ndropn (Suc k) σ))
using 2 by auto
have 6: (∀ j < k. j < nlength σ ∧ f ( (ndropn (Suc j) σ)))
using 2 by blast
have 7: 0 < nlength σ ∧
(∃ k. k ≤ nlength (ndropn (Suc 0) σ) ∧ g ( (ndropn (Suc k) σ)) ∧
(∀ j < k. f ( (ndropn (Suc j) σ)) ) )
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*

12.2.2 UntilNextUntil

lemma *UntilNextUntilsema*:

assumes $0 < nlength \sigma \wedge$

$(\exists k. 0 < k \wedge k \leq nlength \sigma \wedge g ((ndropn k \sigma)) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f ((ndropn j \sigma))))$

shows $(\sigma) \models \circ (f \mathcal{U} g)$

proof –

have 1: $0 < nlength \sigma \wedge$

$(\exists k. 0 < k \wedge k \leq nlength \sigma \wedge g ((ndropn k \sigma)) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f ((ndropn j \sigma))))$

using *assms* **by** *auto*

have 3: $(\exists k. 0 < k \wedge k \leq nlength \sigma \wedge g ((ndropn k \sigma)) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f ((ndropn j \sigma))))$

using 1 **by** *auto*

obtain k **where** 4: $0 < (Suc k) \wedge (Suc k) \leq nlength \sigma \wedge g ((ndropn (Suc k) \sigma)) \wedge$

$(\forall j. 0 < j \wedge j < (Suc k) \longrightarrow f ((ndropn j \sigma)))$

using 3 **by** (*metis Suc-pred*)

have 5: $k \leq nlength (ndropn (Suc 0) \sigma)$

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: $g ((ndropn (Suc k) \sigma))$

using 4 **by** *auto*

have 7: $(\forall j < k. f ((ndropn (Suc j) \sigma)))$

using 4 **by** *blast*

have 8: $(\exists k. k \leq nlength (ndropn (Suc 0) \sigma) \wedge g ((ndropn (Suc k) \sigma)) \wedge (\forall j < k. f ((ndropn (Suc j) \sigma))))$

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 $\sigma \models \bigcirc (f \mathcal{U} g)$

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: $0 < (\text{Suc } k)$

by *simp*

have 5: $g ((\text{ndropn } (\text{Suc } k) \sigma))$

using 3 **by** auto

have 6: $(\text{Suc } k) \leq \text{nlength } \sigma$

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: $(\forall j. 0 < j \wedge j < (\text{Suc } k) \longrightarrow f ((\text{ndropn } j \sigma)))$

using 3 *less-Suc-eq-0-disj* **by** auto

have 8: $(\exists k. 0 < k \wedge k \leq \text{nlength } \sigma \wedge g ((\text{ndropn } k \sigma)) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f ((\text{ndropn } j \sigma))))$

using 3 6 7 **by** blast

show ?thesis **using** 1 8 **by** blast

qed

lemma *UntilNextUntilsem:*

$(\sigma \models \bigcirc (f \mathcal{U} g)) =$

$(0 < \text{nlength } \sigma \wedge$

$(\exists k. 0 < k \wedge k \leq \text{nlength } \sigma \wedge g ((\text{ndropn } k \sigma)) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f ((\text{ndropn } j \sigma))))$

using *UntilNextUntilsema*[of σ g f] *UntilNextUntilsemb*[of f g σ] **by** *meson*

lemma *UntilNextUntilsem1:*

$(\sigma \models f \mathcal{U} g) = (\sigma \models (g \vee (f \wedge \bigcirc(f \mathcal{U} g))))$

unfolding *UntilNextUntilsem*

proof

assume *a*: $(\sigma \models f \mathcal{U} g)$

show $(\sigma \models g \vee$

$f \wedge$

$(\lambda \sigma. 0 < \text{nlength } \sigma \wedge$

$(\exists k. 0 < k \wedge k \leq \text{nlength } \sigma \wedge g((\text{ndropn } k \sigma)) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f((\text{ndropn } j \sigma))))$

```

    ))
  using a by (simp add: until-d-def) (metis enat-0-iff(2) i0-less ndropn-0 neq0-conv not-le)
next
next
assume b: ( $\sigma \models g \vee$ 
   $f \wedge$ 
   $(\lambda \sigma. 0 < \text{nlength } \sigma \wedge$ 
     $(\exists k. 0 < k \wedge k \leq \text{nlength } \sigma \wedge g((\text{ndropn } k \ \sigma)) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f((\text{ndropn } j \ \sigma))))))$ 
show ( $\sigma \models f \mathcal{U} g$ )
  using b by (simp add: until-d-def) (metis i0-lb linorder-cases ndropn-0 not-less-zero zero-enat-def)
qed

```

lemma *UntilNextUntil*:

```

 $\vdash f \mathcal{U} g = (g \vee (f \wedge \bigcirc(f \mathcal{U} g)))$ 
by (simp add: UntilNextUntilsem1 Valid-def)

```

12.2.3 NotUntilFalse

lemma *NotUntilFalse*:

```

 $\vdash \neg (f \mathcal{U} \#False)$ 
by (simp add: intI until-d-def)

```

12.2.4 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

```

12.2.5 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)

```

12.2.6 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)

```

12.2.7 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*

12.2.8 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*)

12.2.9 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*
 (*metis add.commute antisym-conv2 enat.simps(3) enat-add-sub-same enat-min le-iff-add less-imp-le order-refl plus-enat-simps(1)*)
have 6: $g (\text{ndropn } (ka+k) \sigma)$
by (*metis 4 add.commute ndropn-ndropn*)
have 7: $(\forall j < (ka+k). f (\text{ndropn } j \sigma))$

```

    by (metis 2 4 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 (simp add: 8)
qed
qed
qed

```

```

lemma UntilUntil:
 $\vdash f \ \mathcal{U} \ g = f \ \mathcal{U} \ (f \ \mathcal{U} \ g)$ 
using UntilUntilsem by fastforce

```

12.2.10 UntilRightor

```

lemma UntilRightOr:
 $\vdash f \ \mathcal{U} \ (g \ \mathcal{U} \ h) \longrightarrow (f \vee g) \ \mathcal{U} \ h$ 
proof (auto simp add: Valid-def until-d-def ndropn-ndropn)
  fix w :: 'a nelist
  fix k
  fix ka
  assume a0:  $\text{enat } k \leq \text{nlength } w$ 
  assume a1:  $\forall j < k. f (\text{ndropn } j \ w)$ 
  assume a2:  $\text{enat } ka \leq \text{nlength } w - \text{enat } k$ 
  assume a3:  $h (\text{ndropn } (k + ka) \ w)$ 
  assume a4:  $\forall j < ka. g (\text{ndropn } (k + j) \ w)$ 
  show  $\exists k. \text{enat } k \leq \text{nlength } w \wedge h (\text{ndropn } k \ w) \wedge (\forall j < k. f (\text{ndropn } j \ w) \vee g (\text{ndropn } j \ w))$ 
  proof -
    have 1:  $ka + k \leq \text{nlength } w$ 
      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:  $h (\text{ndropn } (ka + k) \ w)$ 
      using a3 by (simp add: add.commute)
    have 3:  $(\forall j < (ka + k). f (\text{ndropn } j \ w) \vee g (\text{ndropn } j \ w))$ 
      by (metis a1 a4 add-diff-inverse-nat less-diff-conv2 not-less)
    show ?thesis using 1 2 3 by blast
  qed
qed

```

12.2.11 UntilRightAnd

```

lemma UntilRightAndsem:
assumes  $(\sigma \models f \ \mathcal{U} \ (g \wedge h))$ 
shows  $(\sigma \models (f \ \mathcal{U} \ g) \ \mathcal{U} \ h)$ 
proof -
  have 1:  $\exists k. k \leq \text{nlength } \sigma \wedge g ((\text{ndropn } k \ \sigma)) \wedge h ((\text{ndropn } k \ \sigma)) \wedge (\forall j < k. f ((\text{ndropn } j \ \sigma)))$ 
    using assms by (simp add: until-d-def)
  obtain k where 2:  $k \leq \text{nlength } \sigma \wedge g ((\text{ndropn } k \ \sigma)) \wedge h ((\text{ndropn } k \ \sigma)) \wedge (\forall j < k. f ((\text{ndropn } j \ \sigma)))$ 
    using 1 by auto
  have 3:  $h ((\text{ndropn } k \ \sigma))$ 
    using 2 by auto
  have 4:  $k \leq \text{nlength } \sigma$ 

```



```

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
have 6: ∃ k. k ≤ nlength σ ∧
  h ( (ndropn k σ)) ∧
  (∀ j < k. ∃ k' ≤ nlength (ndropn j σ). g ((ndropn (k + j) σ)) ∧ (∀ ja < k'. f ((ndropn (ja + j) σ))))
using 2 5 by blast
from 6 show ?thesis by (simp add: until-d-def ndropn-ndropn add commute)
qed

```

```

lemma UntilRightAnd:
  ⊢ f U (g ∧ h) ⟶ (f U g) U h
using UntilRightAndsem Valid-def by auto

```

12.2.12 DiamondEqvTrueUntil

```

lemma DiamondEqvTrueUntil:
  ⊢ ◇ f = # True U f
by (simp add: Valid-def sometimes-defs until-d-def)

```

12.2.13 TrueUntillImpNotUntil

```

lemma nellist-ndropn-first-upto:
  assumes (∃ i ≤ k. f ( (ndropn i xs)))
  shows (∃ i ≤ k. f ( (ndropn i xs)) ∧ (∀ j < i . ¬ (f ( (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  $(\exists n \leq \text{nlength } xs. \neg f \ ( \text{ndropn } n \ xs))$ 
shows  $(\exists n \leq \text{nlength } xs. \neg f \ ( \text{ndropn } n \ xs)) \wedge (\forall k. k < n \longrightarrow f \ ( \text{ndropn } k \ xs))$ 
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)

```

12.2.14 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:  $(\forall k. g \ ( \text{ndropn } k \ \sigma) \longrightarrow k \leq \text{nlength } \sigma \longrightarrow (\exists j < k. \neg f \ ( \text{ndropn } j \ \sigma))) \wedge$ 
 $(\exists n \leq \text{nlength } \sigma. \neg f \ ( \text{ndropn } n \ \sigma))$ 
using assms by (simp add: wait-d-def until-d-def always-defs)
have 2:  $(\forall k. k \leq \text{nlength } \sigma \longrightarrow \neg g \ ( \text{ndropn } k \ \sigma) \vee (\exists j < k. \neg f \ ( \text{ndropn } j \ \sigma)))$ 
using 1 by auto
have 3:  $(\exists n \leq \text{nlength } \sigma. \neg f \ ( \text{ndropn } n \ \sigma))$ 
using 1 by auto

```

```

obtain  $n$  where  $4: n \leq \text{nlength } \sigma \wedge \neg f ( (\text{ndropn } n \ \sigma) ) \wedge$ 
 $(\forall k < n . f ( (\text{ndropn } k \ \sigma) ))$ 
using 3 using NotSuffixFirstfinite by blast
have 16:  $n \leq \text{nlength } \sigma$ 
by (simp add: 4)
have 17:  $\neg g ( (\text{ndropn } n \ \sigma) )$ 
using 1 4 by blast
have 18:  $(\forall j < n . \neg g ( (\text{ndropn } j \ \sigma) ))$ 
by (metis 2 4 dual-order.strict-iff-order dual-order.strict-trans1 enat-ord-simps(2))
have 19:  $\exists k \leq \text{nlength } \sigma . \neg f ( (\text{ndropn } k \ \sigma) ) \wedge \neg g ( (\text{ndropn } k \ \sigma) ) \wedge (\forall j < k . \neg g ( (\text{ndropn } j \ \sigma) ))$ 
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 )

```

12.2.15 UntilInduction

```

lemma LFPUntilsem1:
assumes  $\forall n \leq \text{nlength } \sigma .$ 
 $(g ( (\text{ndropn } n \ \sigma) ) \longrightarrow h ( (\text{ndropn } n \ \sigma) )) \wedge$ 
 $(f ( (\text{ndropn } n \ \sigma) ) \wedge n < \text{nlength } \sigma \wedge h ( (\text{ndropn } (\text{Suc } n) \ \sigma) ) \longrightarrow$ 
 $h ( (\text{ndropn } n \ \sigma) ))$ 
 $k \leq \text{nlength } \sigma$ 
 $g ( (\text{ndropn } k \ \sigma) )$ 
 $\forall j < k . f ( (\text{ndropn } j \ \sigma) )$ 
shows  $h ( \sigma )$ 
using assms
proof (induct k arbitrary: \sigma )
case 0
then show ?case by auto
next
case (Suc k)
then show ?case
proof –
have 1:  $g ( \text{ndropn } (\text{Suc } k) \ \sigma ) \longrightarrow h ( \text{ndropn } (\text{Suc } k) \ \sigma )$ 

```

```

    using Suc.premis(1) Suc.premis(2) by blast
  have 2: (f ( (ndropn k σ)) ∧ k < nlength σ ∧ h ( (ndropn (Suc k) σ)) →
    h ( (ndropn k σ)))
    by (simp add: Suc.premis(1) less-le-not-le)
  have 3: k < nlength σ
    using Suc.premis(2) Suc-ile-eq by auto
  have 4: f ( (ndropn k σ))
    by (simp add: Suc.premis(4))
  have 5: h ( (ndropn k σ))
    using 1 2 3 4 Suc.premis(3) by blast
  have 6: h ( (ndropn 0 σ))
    using zero-induct[of λk . h (ndropn k σ) k]
    3 5 Suc.premis nellist-ndropn-first[of σ h]
    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*:

```

  σ ⊨ □((g ∨ (f ∧ ○h)) → h) → (f U g → 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*:

```

  ⊢ □((g ∨ (f ∧ ○h)) → h) → (f U g → h)
using LFPUntilsem Valid-def
by (metis )

```

lemma *UntilInduction-a*:

```

  ⊢ □(f → ((○ f) ∧ g) ∨ h) → (f → □ g ∨ g U h)
proof -
  have 1: ⊢ (□ g ∨ g U h) = g W h
    by (auto simp add: wait-d-def)
  have 2: ⊢ (f → □ g ∨ g U h) = ( (¬ h) U (¬ g ∧ ¬ h) → ¬ f)
    using 1 WaitNotDistUntil by fastforce
  have 3: ⊢ □( (¬ g ∧ ¬ h) ∨ (¬ h ∧ ○(¬ f))) → ¬ f → ( (¬ h) U (¬ g ∧ ¬ h) → ¬ f)
    using LFPUntil by blast
  have 4: ⊢ (f → ((○ f) ∧ g) ∨ h) → ( (¬ g ∧ ¬ h) ∨ (¬ h ∧ ○(¬ f))) → ¬ f
    using NextImpNotNextNot[of f] by auto
  have 5: ⊢ □(f → ((○ f) ∧ g) ∨ h) → □( (¬ g ∧ ¬ h) ∨ (¬ h ∧ ○(¬ f))) → ¬ f
    using 4 by (rule ImpBoxRule)
  show ?thesis
    using 2 3 5 by fastforce
qed

```

lemma *UntilInduction-b*:

```

  ⊢ □(f → (○f) ∨ g) → (f → □ f ∨ f 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

```

12.3 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 suntil-d-def using 1 inteq-reflection by force

qed

lemma *WNextUntil*:

$\vdash wnext(f \mathcal{U} g) = (empty \vee (\bigcirc f) \mathcal{U} (\bigcirc g))$

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 integ-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: $\vdash \Diamond g = \# \text{True } \mathcal{U} g$

by (simp add: DiamondEqvTrueUntil)

have 3: $\vdash \# \text{True } \mathcal{U} g \longrightarrow (\neg g) \mathcal{U} g$

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: $\vdash ((\Box f \wedge \Box g) \rightarrow ((\Box f \vee f \mathcal{U} h) \wedge (\Box g \vee g \mathcal{U} h)))$

by (simp add: intI)

have 4: $\vdash (f \mathcal{U} h \wedge g \mathcal{U} h) \rightarrow ((\Box f \vee f \mathcal{U} h) \wedge (\Box g \vee g \mathcal{U} h))$

by auto

have 5: $\vdash ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h)) \rightarrow ((\Box f \vee f \mathcal{U} h) \wedge (\Box g \vee g \mathcal{U} h))$

using 3 4 by fastforce

have 6: $\vdash \Box f \wedge \Box g \rightarrow ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$

by auto

have 7: $\vdash \Box f \wedge g \mathcal{U} h \rightarrow ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$

by (metis 2 Prop05 Prop12 UntilAlwaysAndDist UntilLeftDistAnd inteq-reflection lift-imp-trans)

have 8: $\vdash f \mathcal{U} h \wedge \Box g \rightarrow ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$

by (metis Prop05 Prop08 Prop12 UntilAlwaysAndDist UntilAndDist UntilLeftDistAnd inteq-reflection lift-and-com)

have 9: $\vdash f \mathcal{U} h \wedge g \mathcal{U} h \rightarrow ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$

by auto

have 10: $\vdash ((\Box f \vee f \mathcal{U} h) \wedge (\Box g \vee g \mathcal{U} h)) \rightarrow ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$

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*:

$\vdash \text{inf} \longrightarrow \bigcirc (f \mathcal{W} g) = (\bigcirc f) \mathcal{W} (\bigcirc g)$

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*) *BoxEqvFiniteYieldsFiniteChopSkipEqvSkipChopFinite*
SkipYieldsEqvWeakNextYieldsYieldsEqvChopYieldsintIinteq-reflectionunl-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 ChopOrEqvProp11 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) int-simps(32) int-simps(33) 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) int-simps(32) int-simps(33) 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*:

$\vdash \Box f \longrightarrow (\Box g \longrightarrow \Box (f \mathcal{W} g))$

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*:

$\vdash \Box (f \longrightarrow (\Box f \wedge g) \vee h) \longrightarrow (f \longrightarrow g \mathcal{W} h)$

by (*simp add: UntilInduction-a wait-d-def*)

lemma *WaitInductionb*:

$\vdash \Box (f \longrightarrow \Box f \vee g) \longrightarrow (f \longrightarrow f \mathcal{W} g)$

by (*simp add: UntilInduction-b wait-d-def*)

lemma *WaitInductionc*:

$\vdash \Box(f \longrightarrow \Box f) \longrightarrow (f \longrightarrow f \mathcal{W} g)$

proof –

have 1: $\vdash (f \longrightarrow \Box f) \longrightarrow (f \longrightarrow \text{wnext } f)$

unfolding *wnext-d-def* **using** *NextImpNotNextNot[of f]* **by** *auto*

have 2: $\vdash \Box(f \longrightarrow \Box f) \longrightarrow \Box (f \longrightarrow \text{wnext } f)$

using 1 *BoxImpBoxRule* **by** *blast*

show ?thesis **by** (*meson 2 BoxImpWait BoxInduct Prop09 lift-imp-trans*)

qed

lemma *WaitInductiond*:

$\vdash \Box (f \longrightarrow g \wedge \Box f) \longrightarrow (f \longrightarrow f \mathcal{W} g)$

proof –

have 1: $\vdash (f \longrightarrow g \wedge \Box f) \longrightarrow (f \longrightarrow \text{wnext } f)$

unfolding *wnext-d-def* **using** *NextImpNotNextNot[of f]* **by** *auto*

have 2: $\vdash \Box(f \longrightarrow g \wedge \Box f) \longrightarrow \Box (f \longrightarrow \text{wnext } f)$

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 *WaitAbsorbcb*:
 $\vdash (f \mathcal{W} g \wedge g) = g$
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 *WaitAbsorbe*:
 $\vdash (f \mathcal{W} g \vee (f \wedge g)) = f \mathcal{W} g$
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 integ-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: ⊢ f W g = (□f ∨ f U g)
  by (simp add: wait-d-def)
have 2: ⊢ f W h = (□f ∨ f U h)
  by (simp add: wait-d-def)
have 3: ⊢ (¬ g) W h = (□ (¬ g) ∨ (¬ g) U h)
  by (simp add: wait-d-def)
have 4: ⊢ □ f ∧ (□ (¬ g) ∨ (¬ g) U h) → (□ f ∨ f U h)
  by auto
have 5: ⊢ (□f ∨ f U g) ∧ □ (¬ g) → (□ f ∨ f U h)
  using 1 WaitAndBoxImpBox by fastforce
have 6: ⊢ f U g ∧ (¬ g) U h → (□ f ∨ f U h)
  by (simp add: Prop05 UntilNotImp)
have 7: ⊢ □ f ∧ (¬ g) U h → (□ f ∨ f U h)
  by auto
have 8: ⊢ (□f ∨ f U g) ∧ (¬ g) U h → (□ f ∨ f U h)
  using 6 7 by fastforce
have 9: ⊢ (□f ∨ f U g) ∧ (□ (¬ g) ∨ (¬ g) U h) → (□ f ∨ f U h)
  using 5 8 by fastforce
show ?thesis by (simp add: 9 wait-d-def)
qed

```

end

13 Pi operator for infinite and finite ITL

```

theory Pi
imports Until Omega
begin

```

This theory introduces the Pi operator [10, 7]. 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 [7].

13.1 Definitions

```

definition sxfilt :: 'a nelist ⇒ ('a:: world) formula ⇒ 'a nelist nelist
where sxfilt s f = (nfilter (λ σ. σ ⊨ f) (ndropns s))

```

```

definition pifilt :: 'a nelist ⇒ ('a:: world) formula ⇒ 'a nelist
where pifilt s f = (nmap (λ s. nnth s 0) (sxfilt s f))

```

```

definition pi-d :: ('a:: world) formula ⇒ 'a formula ⇒ 'a formula
where pi-d F G ≡ λ s. ( (∃ i ≤ nlength s. (ndropn i s) ⊨ F) ∧ ( (pifilt s F) ⊨ G ) )

```

syntax

```

-pi-d    :: [lift, lift] ⇒ lift      ((- Π -) [84, 84] 83)

```

syntax (*ASCII*)
 $-pi-d \quad :: [lift, lift] \Rightarrow lift \quad ((- PI -) [84, 84] 83)$

translations
 $-pi-d \quad \Rightarrow CONST pi-d$

definition $upi-d :: ('a :: world) formula \Rightarrow 'a formula \Rightarrow 'a formula$
where $upi-d F G \equiv LIFT(\neg(F \Pi (\neg G)))$

syntax
 $-upi-d \quad :: [lift, lift] \Rightarrow lift \quad ((- \Pi^u -) [84, 84] 83)$

syntax (*ASCII*)
 $-upi-d \quad :: [lift, lift] \Rightarrow lift \quad ((- UPI -) [84, 84] 83)$

translations
 $-upi-d \quad \Rightarrow CONST upi-d$

13.2 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-pifilt*:

shows $(\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{sfxfilt } \sigma \ f)) = \text{pifilt } \sigma \ f$
by (*simp add: pifilt-def*)

lemma *sfxfilt-nlength-bound*:

assumes $(\exists i \leq \text{nlength } xs. f (\text{ndropn } i \ xs))$
shows $\text{nlength } (\text{sfxfilt } xs \ f) \leq \text{nlength } xs$
using *assms*
by (*metis length-nfilter-le ndropns-nlength sfxfilt-def*)

lemma *pifilt-nlength-bound*:

assumes $(\exists i \leq \text{nlength } \sigma. f (\text{ndropn } i \ \sigma))$
shows $\text{nlength } (\text{pifilt } \sigma \ f) \leq \text{nlength } \sigma$
using *assms* **by** (*simp add: pifilt-def sfxfilt-nlength-bound*)

lemma *sfxfilt-nlength-nnth-bound*:

assumes $(\exists i \leq \text{nlength } \sigma. f (\text{ndropn } i \ \sigma))$
 $j \leq \text{nlength } (\text{sfxfilt } \sigma \ f)$
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 sxfilter-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) sxfilter-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 σ $\lambda s. s \models f \ j \ 0$]* *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 (ndropns σ) $f \ j \ 0$]* *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 (nnth (nkfilter $f \ 0$ (ndropns σ)) 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 σ f g]* **by** *auto*
show *?thesis* **using** 1 2 4 **by** *auto*
qed

lemma *pfilt-nlength-imp*:
assumes $(\exists i \leq \text{nlength } \sigma. f (\text{ndropn } i \ \sigma) \wedge g (\text{ndropn } i \ \sigma))$
shows $\text{nlength } (\text{pfilt } \sigma (\text{LIFT}(f \wedge g))) \leq \text{nlength } (\text{pfilt } \sigma f)$
using *assms*
by (*simp add: sfxfilt-nlength-imp pfilt-def*)

lemma *nellist-nset-sfxfilt [simp]*:
assumes $(\exists i \leq \text{nlength } xs. f (\text{ndropn } i \ xs))$
shows $(\text{nset } (\text{sfxfilt } xs \ f)) = \{ys. ys \in \text{nset } (\text{ndropns } xs) \wedge f \ (ys)\}$
using *assms*
proof –
have 1: $\text{nset } (\text{sfxfilt } xs \ f) = \text{nset } (\text{nfilter } f (\text{ndropns } xs))$
by (*simp add: sfxfilt-def*)
have 2: $\exists x \in \text{nset } (\text{ndropns } xs). f \ x$
using *assms using in-nset-ndropns* **by** *blast*
have 3: $\text{nset } (\text{nfilter } f (\text{ndropns } xs)) = \{ys. ys \in \text{nset } (\text{ndropns } xs) \wedge f \ ys\}$
using 2 *nset-nfilter[of ndropns xs f]* **by** *auto*
show *?thesis* **by** (*simp add: 1 3*)
qed

lemma *subset-nfilter*:
assumes $\exists x \in \text{nset } xs. P \ x$
shows $\text{nset}(\text{nfilter } P \ xs) \leq \text{nset}(\text{nfilter } (\lambda x. P \ x \vee Q \ x) \ xs)$
proof –
have 1: $\exists x \in \text{nset } xs. P \ x \vee Q \ x$
using *assms* **by** *blast*
have 2: $\text{nset}(\text{nfilter } (\lambda x. P \ x \vee Q \ x) \ xs) = \{x \in \text{nset } xs. P \ x \vee Q \ x\}$
using *assms nset-nfilter[of xs $(\lambda x. P \ x \vee Q \ x)$]* **by** *blast*
have 3: $\text{nset}(\text{nfilter } P \ xs) = \{x \in \text{nset } xs. P \ x\}$
using *assms nset-nfilter[of xs P]* **by** (*simp add: Collect-conj-eq*)
have 4: $\{x \in \text{nset } xs. P \ x\} \leq \{x \in \text{nset } xs. P \ x \vee Q \ x\}$
by *auto*

show *?thesis* **by** (*simp add: 2 3 4*)
qed

lemma *nellist-subset-sxfilt* [*simp*]:

assumes $(\exists i \leq \text{nlength } xs. f (\text{ndropn } i \text{ } xs))$

shows $(\text{nset } (\text{sxfilt } xs \text{ } f)) \leq (\text{nset } (\text{sxfilt } xs \text{ } (\text{LIFT}(f \vee g))))$

proof –

have 1: $\exists x \in \text{nset } (\text{ndropns } xs). f \ x$

using *assms* **using** *in-nset-ndropns* **by** *blast*

have 2: $\text{nset } (\text{sxfilt } xs \text{ } f) = \text{nset } (\text{nfilter } f \text{ } (\text{ndropns } xs))$

by (*simp add: sxfilt-def*)

have 3: $\text{nset } (\text{sxfilt } xs \text{ } (\text{LIFT}(f \vee g))) = \text{nset } (\text{nfilter } (\lambda x. f \ x \vee g \ x) \text{ } (\text{ndropns } xs))$

by (*simp add: sxfilt-def*)

have 4: $\text{nset } (\text{nfilter } f \text{ } (\text{ndropns } xs)) \leq \text{nset } (\text{nfilter } (\lambda x. f \ x \vee g \ x) \text{ } (\text{ndropns } xs))$

using 1 *subset-nfilter*[*of ndropns xs f g*] **by** *auto*

show *?thesis* **using** 2 3 4 **by** *blast*

qed

lemma *nellist-nset-nmap*:

$(\text{nset } (\text{nmap } (\lambda s. \text{nnth } s \ 0) \text{ } xs)) = \{(\text{nnth } ys \ 0) \mid ys. ys \in \text{nset } xs\}$

by (*auto simp add: in-nset-conv-nnth nset-conv-nnth*)

(*metis nnth-nmap*)

lemma *nellist-nset-pifilt* [*simp*]:

assumes $(\exists i \leq \text{nlength } xs. f (\text{ndropn } i \text{ } xs))$

shows $(\text{nset } (\text{pifilt } xs \text{ } f)) = \{(\text{nnth } ys \ 0) \mid ys. ys \in \text{nset } (\text{ndropns } xs) \wedge f \text{ } (ys)\}$

proof –

have 1: $(\text{nset } (\text{pifilt } xs \text{ } f)) = (\text{nset } (\text{nmap } (\lambda s. \text{nnth } s \ 0) \text{ } (\text{sxfilt } xs \text{ } f)))$

by (*simp add: pifilt-def*)

have 2: $(\text{nset } (\text{nmap } (\lambda s. \text{nnth } s \ 0) \text{ } (\text{sxfilt } xs \text{ } f))) =$

$\{(\text{nnth } ys \ 0) \mid ys. ys \in \text{nset } (\text{sxfilt } xs \text{ } f)\}$

using *nellist-nset-nmap*[*of sxfilt xs f*] **by** *auto*

have 3: $\{(\text{nnth } ys \ 0) \mid ys. ys \in \text{nset } (\text{sxfilt } xs \text{ } f)\} =$

$\{(\text{nnth } ys \ 0) \mid ys. ys \in \text{nset } (\text{ndropns } xs) \wedge f \text{ } (ys)\}$

using *assms* **by** *auto*

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

qed

lemma *nellist-nnth-sxfilt-in-nset*:

$x \in \text{nset } (\text{sxfilt } \sigma \text{ } (\text{LIFT}(f \vee g))) =$

$(\exists i \leq \text{nlength } (\text{sxfilt } \sigma \text{ } (\text{LIFT}(f \vee g))). x = (\text{nnth } (\text{sxfilt } \sigma \text{ } (\text{LIFT}(f \vee g))) \ i))$

by (*metis in-nset-conv-nnth*)

lemma *nfilter-nnth-or*:

assumes $\exists x \in \text{nset } xs. P \ x$

shows $\exists x \in \text{nset } (\text{nfilter } (\lambda x. P \ x \vee Q \ x) \text{ } xs). P \ x$

proof –

have 0: $\exists x \in \text{nset } xs. P \ x \vee Q \ x$

using *assms* **by** *auto*

have 1: $\exists i \leq \text{nlength } (\text{nfilter } (\lambda x. P \ x \vee Q \ x) \text{ } xs). P \ (\text{nnth } (\text{nfilter } P \text{ } xs) \ i)$

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: $i \leq \text{nlength}(\text{nfilter } (\lambda x. P\ x \vee Q\ x)\ xs) \wedge P\ (\text{nnth } (\text{nfilter } P\ xs)\ i) \wedge$
 $(\text{nnth } (\text{nfilter } P\ xs)\ i) \in \text{nset } (\text{nfilter } P\ xs)$
by (*metis* 1 *in-nset-conv-nnth* *le-cases* *nfinite-ntaken* *nset-nlast* *ntaken-all* *ntaken-nlast*)
have 3: $\text{nset } (\text{nfilter } P\ xs) \leq \text{nset}(\text{nfilter } (\lambda x. P\ x \vee Q\ x)\ xs)$
using *assms subset-nfilter*[of *xs* *P* *Q*] **by** *auto*
have 4: $(\text{nnth } (\text{nfilter } P\ xs)\ i) \in \text{nset } (\text{nfilter } P\ xs)$
using 2 **by** *auto*
have 5: $(\text{nnth } (\text{nfilter } P\ xs)\ i) \in \text{nset}(\text{nfilter } (\lambda x. P\ x \vee Q\ x)\ xs)$
using 3 4 **by** *blast*
show ?thesis **using** 2 5 **by** *blast*
qed

lemma *sfxfilt-nnth-or*:

assumes $(\exists\ i \leq \text{nlength } \sigma. (\text{ndropn } i\ \sigma) \models f)$

shows $(\exists\ i \leq \text{nlength } (\text{sfxfilt } \sigma\ (\text{LIFT}(f \vee g))). (\text{nnth } (\text{sfxfilt } \sigma\ (\text{LIFT}(f \vee g)))\ i) \models f)$

proof –

have 1: $\exists\ x \in \text{nset } (\text{ndropns } \sigma). f\ x$

using *assms* **by** (*simp* *add*: *sfxfilter-help*)

have 2: $\text{sfxfilt } \sigma\ (\text{LIFT}(f \vee g)) = \text{nfilter } (\lambda\ x. f\ x \vee g\ x)\ (\text{ndropns } \sigma)$

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*:

$\sigma \models \neg ((\#False) \Pi f)$

by (*simp* *add*: *pi-d-def*)

lemma *pfilt-true*:

$\text{pfilt } \sigma\ (\text{LIFT}(\#True)) = \sigma$

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 $(\text{pfilt } xs\ (\lambda s. s \models \text{init } w)) = \text{nfilter } (\lambda y. w\ ((\text{NNil } y)))\ xs$

proof (*simp* *add*: *init-defs* *pfilt-def* *sfxfilt-def*)

show $\text{nmap } (\lambda s. \text{nnth } s\ 0)\ (\text{nfilter } (\lambda s. w\ (\text{NNil } (\text{nfirst } s))))\ (\text{ndropns } xs) =$

$\text{nfilter } (\lambda y. w\ (\text{NNil } y))\ xs$

proof –

have 1: $\bigwedge x. ((\lambda y. w ((NNil\ y))) \circ (\lambda s. nnth\ s\ 0))\ x = (\lambda s. w (NNil\ (nfirst\ s)))\ x$

by *simp* (*metis* *ndropn-0* *ndropn-nfirst*)

have 2: $((\lambda y. w ((NNil\ y))) \circ (\lambda s. nnth\ s\ 0)) = (\lambda s. w (NNil\ (nfirst\ s)))$

using 1 **by** *blast*

have 4: $nmap\ (\lambda s. nnth\ s\ 0)\ (nfilter\ (\lambda s. w (NNil\ (nfirst\ s)))\ (ndropns\ xs)) =$

$nfilter\ (\lambda y. w ((NNil\ y)))\ (nmap\ (\lambda s. nnth\ s\ 0)\ (ndropns\ xs))$

by (*metis* (*mono-tags*, *lifting*) 1 *nfilter-cong* *nfilter-nmap*)

have 5: $(nmap\ (\lambda s. nnth\ s\ 0)\ (ndropns\ xs)) = xs$

by (*simp* *add*: *nmap-first-ndropns*)

show *?thesis*

by (*simp* *add*: 4 5)

qed

qed

lemma *pfilt-init-a*:

shows $(pfilt\ xs\ (\lambda s. w (NNil\ (nnth\ s\ 0)))) = nfilter\ (\lambda y. w (NNil\ y))\ xs$

proof –

have 2: $(pfilt\ xs\ (\lambda s. s \models init\ w)) = (pfilt\ xs\ (\lambda s. w (NNil\ (nnth\ s\ 0))))$

by (*metis* (*no-types*, *lifting*) *in-nset-ndropns* *init-defs* *ndropn-nfirst* *nfilter-cong* *nnth-zero-ndropn* *ntaken-0* *pfilt-def* *sxfilt-def*)

show *?thesis* **using** *pfilt-init* 2 **by** (*metis*)

qed

lemma *pfilt-pfilt* :

assumes $(\exists i \leq nlength\ xs. f\ (ndropn\ i\ xs))$

$(\exists i \leq nlength\ (pfilt\ xs\ f). w\ (NNil\ (nnth\ (ndropn\ i\ (pfilt\ xs\ f))\ 0)))$

shows $(pfilt\ (pfilt\ xs\ f)\ (LIFT(init\ w))) = pfilt\ xs\ (LIFT(f \wedge init\ w))$

proof –

have 1: $\exists i \leq nlength\ (pfilt\ xs\ f). (LIFT(init\ w))\ (ndropn\ i\ (pfilt\ xs\ f))$

using *assms* **by** (*simp* *add*: *init-defs* *ndropn-nfirst*)

have 2: $(pfilt\ (pfilt\ xs\ f)\ (LIFT(init\ w))) = nfilter\ (\lambda y. w ((NNil\ y)))\ (pfilt\ xs\ f)$

using 1 *pfilt-init*[*of* (*pfilt* *xs* *f*) *w*] **by** *auto*

have 3: $(pfilt\ xs\ f) = nmap\ (\lambda s. nnth\ s\ 0)\ (sxfilt\ xs\ f)$

by (*simp* *add*: *assms* *sxfilt-pfilt*)

have 4: $(sxfilt\ xs\ f) = nfilter\ (\lambda ys. f\ ys)\ (ndropns\ xs)$

using *sxfilt-def* **by** *blast*

have 5: $(pfilt\ xs\ f) = nmap\ (\lambda s. nnth\ s\ 0)\ (nfilter\ (\lambda ys. f\ ys)\ (ndropns\ xs))$

by (*simp* *add*: 3 4)

have 6: $nfilter\ (\lambda y. w (NNil\ y))\ (pfilt\ xs\ f) =$

$nfilter\ (\lambda y. w (NNil\ y))\ (nmap\ (\lambda s. nnth\ s\ 0)\ (nfilter\ (\lambda ys. f\ ys)\ (ndropns\ xs)))$

using 5 **by** *simp*

have 7: $nfilter\ (\lambda y. w (NNil\ y))\ (nmap\ (\lambda s. nnth\ s\ 0)\ (nfilter\ (\lambda ys. f\ ys)\ (ndropns\ xs))) =$

$nmap\ (\lambda s. nnth\ s\ 0)\ (nfilter\ ((\lambda y. w (NNil\ y)) \circ (\lambda s. nnth\ s\ 0))\ (nfilter\ (\lambda ys. f\ ys)\ (ndropns\ xs)))$

using *assms* 3 4 **by** (*simp* *add*: *nfilter-nmap* *pfiltinit-help*)

have 8: $\exists x \in nset\ (nfilter\ (\lambda ys. f\ ys)\ (ndropns\ xs)). ((\lambda y. w ((NNil\ y))) \circ (\lambda s. nnth\ s\ 0))\ x$

using *assms* 3 4 *in-nset-conv-nnth*[*of* - (*nfilter* *f* (*ndropns* *xs*))]

by *simp*

(*metis* *in-nset-conv-nnth* *ndropn-0* *ndropn-nfirst* *nlength-nmap* *nnth-nmap*)

have 9: $\exists x \in nset\ (ndropns\ xs). (\lambda ys. f\ ys)\ x$

using *assms* **by** (*simp add: sfxfilter-help*)
have 10: $\exists x \in \text{nset } (\text{ndropns } xs). ((\lambda y. w (\text{NNil } y)) \circ (\lambda s. \text{nnth } s \ 0)) \ x \wedge (\lambda ys. f \ ys) \ x$
using 8 9
using *exist-conj* [*of* $f (\text{ndropns } xs) ((\lambda y. w (\text{NNil } y)) \circ (\lambda s. \text{nnth } s \ 0))$] **by** *fastforce*
have 11: $\text{nfilter } ((\lambda y. w (\text{NNil } y)) \circ (\lambda s. \text{nnth } s \ 0)) (\text{nfilter } (\lambda ys. f \ ys) (\text{ndropns } xs)) =$
 $\text{nfilter } (\lambda zs. ((\lambda y. w (\text{NNil } y)) \circ (\lambda s. \text{nnth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) (\text{ndropns } xs)$
using *nfilter-nfilter* [*of* $(\lambda ys. f \ ys) (\text{ndropns } xs) ((\lambda y. w (\text{NNil } y)) \circ (\lambda s. \text{nnth } s \ 0))$]]
8 10 **by** *blast*
have 12: $\exists i \leq \text{nlength } xs. (\lambda zs. ((\lambda y. w (\text{NNil } y)) \circ (\lambda s. \text{nnth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) (\text{ndropn } i \ xs)$
by (*metis* 10 *in-nset-ndropns*)
have 13: $(\text{nfilter } (\lambda zs. ((\lambda y. w (\text{NNil } y)) \circ (\lambda s. \text{nnth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) (\text{ndropns } xs))$
 $= (\text{sfxfilt } xs (\lambda zs. ((\lambda y. w (\text{NNil } y)) \circ (\lambda s. \text{nnth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs))$
by (*simp add: sfxfilt-def*)
have 14: $\exists i \leq \text{nlength } xs. (\lambda zs. ((\lambda y. w (\text{NNil } y)) \circ (\lambda s. \text{nnth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) (\text{ndropn } i \ xs)$
using 12 **by** *blast*
have 15: $\text{nmap } (\lambda s. \text{nnth } s \ 0)$
 $((\text{sfxfilt } xs (\lambda zs. ((\lambda y. w (\text{NNil } y)) \circ (\lambda s. \text{nnth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs))) =$
 $\text{pifilt } xs (\lambda zs. ((\lambda y. w (\text{NNil } y)) \circ (\lambda s. \text{nnth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs)$
using 14 **by** (*simp add: sfxfilt-pifilt*)
have 16: $\bigwedge xs. (\lambda zs. ((\lambda y. w (\text{NNil } y)) \circ (\lambda s. \text{nnth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) \ xs =$
 $(\text{LIFT}(f \wedge \text{init } w)) \ xs$
by (*simp add: init-defs*) (*metis ndropn-0 ndropn-nfirst*)
have 17: $\text{pifilt } xs (\lambda zs. ((\lambda y. w (\text{NNil } y)) \circ (\lambda s. \text{nnth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) =$
 $\text{pifilt } xs (\text{LIFT}(f \wedge \text{init } w))$
using 16 **by** *presburger*
show ?thesis
using 11 13 15 17 2 3 4 7 **by** *auto*
qed

lemma *PiStatesem*:

$(\sigma \models (\text{init } w) \ \Pi \ f) =$
 $((\exists i \leq \text{nlength } \sigma. w (\text{NNil } (\text{nnth } \sigma \ i))) \wedge f (\text{nfilter } (\lambda y. w ((\text{NNil } y))) \ \sigma))$

by (*simp add: pi-d-def init-defs ndropn-nfirst pifilt-init*)

13.3 Soundness of Axioms

13.3.1 PiK

lemma *PiKsem*:

$\sigma \models f1 \ \Pi^u \ (f \longrightarrow g) \longrightarrow (f1 \ \Pi^u \ f \longrightarrow f1 \ \Pi^u \ g)$

by (*simp add: upi-d-def init-defs pi-d-def*) *auto*

lemma *PiK*:

$\vdash f1 \ \Pi^u \ (f \longrightarrow g) \longrightarrow (f1 \ \Pi^u \ f \longrightarrow f1 \ \Pi^u \ g)$

using *PiKsem Valid-def* **by** *blast*

13.3.2 PiDc

lemma *PiDcsem*:

$\sigma \models f \ \Pi \ g \longrightarrow f \ \Pi^u \ g$

by (simp add: upi-d-def init-defs pi-d-def)

lemma *PiDc*:

$\vdash f \Pi g \longrightarrow f \Pi^u g$

using *PiDcsem Valid-def* by blast

13.3.3 PiN

lemma *PiN*:

assumes $\vdash g$

shows $\vdash f \Pi^u g$

using *assms* by (simp add: Valid-def pi-d-def upi-d-def)

13.3.4 PiTrueEqvDiamond

lemma *PiTrueEqvDiamond*:

$\vdash f \Pi \#True = \Diamond f$

by (simp add: Valid-def pi-d-def itl-defs)

13.3.5 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

13.3.6 UPiFalseEqvBoxNot:

lemma *UPiFalseEqvBoxNot*:

$\vdash f \Pi^u \#False = \Box (\neg f)$

by (simp add: Valid-def upi-d-def pi-d-def itl-defs)

13.3.7 BoxEqvImpPiEqv

lemma *BoxEqvImpPiEqvsem*:

assumes $(\sigma \models \Box (f1 = f2))$

shows $(\sigma \models (f1 \Pi g = f2 \Pi g))$

proof –

show $(\sigma \models (f1 \Pi g = f2 \Pi g))$

proof –

have 1: $\forall n \leq nlength \sigma. f1 (ndropn n \sigma) = f2 (ndropn n \sigma)$

using *assms* by (simp add: itl-defs)

have 2: $(\sigma \models (f1 \Pi g)) = ((\exists i \leq nlength \sigma. f1 (ndropn i \sigma)) \wedge g (pifilt \sigma f1))$

by (simp add: pi-d-def)

have 3: $(\exists i \leq nlength \sigma. f1 (ndropn i \sigma)) = (\exists i \leq nlength \sigma. f2 (ndropn i \sigma))$

using 1 by blast

have 6: $(\exists i \leq nlength \sigma. f1 (ndropn i \sigma)) \implies$

$\{ (ndropn n \sigma) \mid n. n \leq nlength \sigma \wedge f1 (ndropn n \sigma) \} =$

$\{ (ndropn n \sigma) \mid n. n \leq nlength \sigma \wedge f2 (ndropn n \sigma) \}$

using 1 by blast

have 7: $(\exists i \leq nlength \sigma. f1 (ndropn i \sigma)) \implies (sfxfilt \sigma f1) = (sfxfilt \sigma f2)$

by (metis 1 in-nset-ndropns nfilter-cong sfxfilt-def)

have 8: $((\exists i \leq nlength \sigma. f1 (ndropn i \sigma)) \wedge g (pifilt \sigma f1)) =$

```

      (( $\exists i \leq \text{nlength } \sigma. f2 (\text{ndropn } i \ \sigma) \wedge g (\text{pifilt } \sigma \ f2)$ )
      by (metis 3 7 pifilt-def)
show ?thesis
      by (simp add: 8 pi-d-def)
qed
qed

```

```

lemma BoxEqvImpPiEqv:
   $\vdash \Box (f1 = f2) \longrightarrow (f1 \Pi g = f2 \Pi g)$ 
using BoxEqvImpPiEqvsem by (simp add: Valid-def, auto)

```

13.3.8 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:
   $\vdash f \Pi (\Diamond (\text{init } w)) \longrightarrow \Diamond (\text{init } w)$ 
using PiDiamondImpDiamondsem Valid-def by blast

```

13.3.9 PiAssoc

```

lemma PiAssocsem1:
assumes  $i \leq \text{nlength } \sigma$ 
           $f (\text{ndropn } i \ \sigma)$ 
           $ia \leq \text{nlength } (\text{pifilt } \sigma \ f)$ 
           $w (\text{NNil } (\text{nnth } (\text{pifilt } \sigma \ f) \ ia))$ 
shows  $\exists i \leq \text{nlength } \sigma. f (\text{ndropn } i \ \sigma) \wedge w (\text{NNil } (\text{nnth } (\text{ndropn } i \ \sigma) \ 0))$ 
proof –
  have 1:  $(\text{nnth } (\text{pifilt } \sigma \ f) \ ia) = (\text{nnth } (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{sxfilt } \sigma \ f)) \ ia)$ 
    using assms(1) assms(2) assms(3) sxfilt-pifilt-nnth by blast
  have 2:  $(\text{nnth } (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{sxfilt } \sigma \ f)) \ ia) =$ 
     $(\lambda s. \text{nnth } s \ 0) (\text{nnth } (\text{sxfilt } \sigma \ f) \ ia)$ 
    by (metis assms(3) nlength-nmap nnth-nmap pifilt-def)
  have 3:  $f (\text{nnth } (\text{sxfilt } \sigma \ f) \ ia)$ 
    using sxfilt-nnth
    by (metis assms(1) assms(2) assms(3) nlength-nmap pifilt-def)
  have 4:  $\text{nnth } (\text{sxfilt } \sigma \ f) \ ia = \text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ ia) \ \sigma$ 
    using sxfilt-pifilt-nnth-ndropn assms(1) assms(2) assms(3) by blast
  have 5:  $w (\text{NNil } (\text{nnth } (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ ia) \ \sigma) \ 0))$ 
    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  $i \leq \text{nlength } \sigma$ 
           $f (\text{ndropn } i \ \sigma)$ 
           $w (\text{NNil } (\text{nnth } \sigma \ i))$ 
shows  $\exists j \leq \text{nlength } (\text{pifilt } \sigma \ f). w (\text{NNil } (\text{nnth } (\text{pifilt } \sigma \ f) \ j))$ 

```


proof –

have 1: $\exists j \leq \text{nlength } (\text{sfxfilt } \sigma f). f (\text{nnth } (\text{sfxfilt } \sigma f) j)$
using *assms pifilt-exists* **by** *blast*
have 2: $(\text{LIFT } (\text{init } w)) (\text{ndropn } i \sigma)$
by (*simp add: assms init-defs ndropn-nfirst*)
have 3: $\exists j \leq \text{nlength } (\text{sfxfilt } \sigma (\text{LIFT}(\text{init } w))). (\text{LIFT}(\text{init } w)) (\text{nnth } (\text{sfxfilt } \sigma (\text{LIFT}(\text{init } w))) j)$
using *pifilt-exists 2 assms* **by** *blast*
have 4: $(\text{LIFT } (f \wedge \text{init } w)) (\text{ndropn } i \sigma)$
by (*simp add: 2 assms*)
have 5: $\exists j \leq \text{nlength } (\text{sfxfilt } \sigma (\text{LIFT}(f \wedge \text{init } w))).$
 $(\text{LIFT}(f \wedge \text{init } w)) (\text{nnth } (\text{sfxfilt } \sigma (\text{LIFT}(f \wedge \text{init } w))) j)$
using *pifilt-exists 4 assms* **by** *blast*
have 6: $\exists i \leq \text{nlength } \sigma. \text{ndropn } i \sigma \models f \wedge \text{init } w$
using *4 assms* **by** *blast*
have 7: $\exists j \leq \text{nlength } (\text{sfxfilt } \sigma (\text{LIFT}((f \wedge \text{init } w) \vee (f \wedge \neg (\text{init } w))))).$
 $(\text{LIFT}(f \wedge \text{init } w)) (\text{nnth } (\text{sfxfilt } \sigma (\text{LIFT}((f \wedge \text{init } w) \vee (f \wedge \neg (\text{init } w)))) j)$
using *6 sfxfilt-nnth-or[of σ LIFT($f \wedge \text{init } w$) LIFT($f \wedge \neg(\text{init } w)$)]*
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*:

```

 $\sigma \models f \amalg ((init\ w) \amalg g) = (f \wedge (init\ w)) \amalg g$ 
proof (auto simp add: pi-d-def init-defs)
  fix i
  fix ia
  assume a0:  $g\ (pifilt\ (pifilt\ \sigma\ f)\ (LIFT(init\ w)))$ 
  assume a1:  $(enat\ i) \leq nlength\ \sigma$ 
  assume a2:  $f\ (ndropn\ i\ \sigma)$ 
  assume a3:  $(enat\ ia) \leq nlength\ (pifilt\ \sigma\ f)$ 
  assume a4:  $w\ (NNil\ (nfirst\ (ndropn\ ia\ (pifilt\ \sigma\ f))))$ 
  show  $\exists i. enat\ i \leq nlength\ \sigma \wedge f\ (ndropn\ i\ \sigma) \wedge w\ (NNil\ (nfirst\ (ndropn\ i\ \sigma)))$ 
    using a0 a1 a2 a3 a4 PiAssocsem1
    by (metis ndropn-nfirst nnth-zero-ndropn)
next
  fix i
  fix ia
  assume a0:  $g\ (pifilt\ (pifilt\ \sigma\ f)\ (LIFT(init\ w)))$ 
  assume a1:  $(enat\ i) \leq nlength\ \sigma$ 
  assume a2:  $f\ (ndropn\ i\ \sigma)$ 
  assume a3:  $(enat\ ia) \leq nlength\ (pifilt\ \sigma\ f)$ 
  assume a4:  $w\ (NNil\ (nfirst\ (ndropn\ ia\ (pifilt\ \sigma\ f))))$ 
  show  $g\ (pifilt\ \sigma\ (LIFT(f \wedge init\ w)))$ 
    using a0 a1 a2 a3 a4 by (metis ndropn-nfirst nnth-zero-ndropn pifilt-pifilt)
next
  fix i
  assume a0:  $g\ (pifilt\ \sigma\ (LIFT(f \wedge init\ w)))$ 
  assume a1:  $(enat\ i) \leq nlength\ \sigma$ 
  assume a2:  $f\ (ndropn\ i\ \sigma)$ 
  assume a3:  $w\ (NNil\ (nfirst\ (ndropn\ i\ \sigma)))$ 
  show  $\exists i. enat\ i \leq nlength\ (pifilt\ \sigma\ f) \wedge w\ (NNil\ (nfirst\ (ndropn\ i\ (pifilt\ \sigma\ f))))$ 
    using a0 a1 a2 a3 by (metis PiAssocsem2 ndropn-nfirst)
next
  fix i
  assume a0:  $g\ (pifilt\ \sigma\ (LIFT(f \wedge init\ w)))$ 
  assume a1:  $(enat\ i) \leq nlength\ \sigma$ 
  assume a2:  $f\ (ndropn\ i\ \sigma)$ 
  assume a3:  $w\ (NNil\ (nfirst\ (ndropn\ i\ \sigma)))$ 
  show  $g\ (pifilt\ (pifilt\ \sigma\ f)\ (LIFT(init\ w)))$ 
    using a0 a1 a2 a3 by (metis PiAssocsem2 ndropn-nfirst nnth-zero-ndropn pifilt-pifilt)
qed

```

lemma *PiAssoc*:

```

 $\vdash f \amalg ((init\ w) \amalg g) = (f \wedge (init\ w)) \amalg g$ 
using PiAssocsem Valid-def by blast

```

13.3.10 PiNotEqvDiamondAndNotPi

lemma *PiNotEqvDiamondAndNotPisem*:

```

 $\sigma \models f \amalg (\neg g) = (\Diamond f \wedge \neg(f \amalg g))$ 
by (simp add: pi-d-def itl-defs) blast

```

lemma *PiNotEqvDiamondAndNotPi*:
 $\vdash f \Pi (\neg g) = (\Diamond f \wedge \neg(f \Pi g))$
using *PiNotEqvDiamondAndNotPisem Valid-def* **by** *blast*

13.3.11 PiSchopDist

lemma *PiSchopDistsema*:
assumes $(\sigma \models (\text{init } w) \Pi (g \frown h))$
shows $(\sigma \models ((\text{init } w) \Pi g) \frown ((\text{init } w) \wedge ((\text{init } w) \Pi h)))$
proof –
have 1: $(\sigma \models (\text{init } w) \Pi (g \frown h))$
using *assms* **by** *auto*
have 2: $((\exists i \leq \text{nlength } \sigma. (\text{LIFT}(\text{init } w)) (\text{ndropn } i \sigma)) \wedge ((\text{pifilt } \sigma (\text{LIFT}(\text{init } w))) \models g \frown h))$
using 1 **by** (*simp add: pi-d-def*)
have 3: $(\exists i \leq \text{nlength } \sigma. (\text{LIFT}(\text{init } w)) (\text{ndropn } i \sigma))$
using 2 **by** *auto*
have 4: $((\text{pifilt } \sigma (\text{LIFT}(\text{init } w))) \models g \frown h)$
using 2 **by** *auto*
have 5: $\text{nfilter } (\lambda y. w ((\text{NNil } y))) \sigma \models g \frown h$
using *pifilt-init* **by** (*metis* 4)
have 6: $(\exists n. (\text{enat } n) \leq \text{nlength } (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma) \wedge g (\text{ntaken } n (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma)) \wedge h (\text{ndropn } n (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma)))$
using 5 **by** (*simp add: itl-defs*)
obtain *n* **where** 7: $(\text{enat } n) \leq \text{nlength } (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma) \wedge g (\text{ntaken } n (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma)) \wedge h (\text{ndropn } n (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma))$
using 6 **by** *auto*
have 8: $\exists i \leq \text{nlength } \sigma. w (\text{NNil } (\text{nnth } \sigma \ i))$
using 3 **using** *init-defs PiStatesem assms* **by** *blast*
have 9: $\exists x \in \text{nset } \sigma. w (\text{NNil } x)$
using 8 **by** (*simp add: pifiltinit-help*)
have 10: $(\text{ntaken } n (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma)) = (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{ntaken } ((\text{nnth } (\text{nkfilter } (\lambda y. w (\text{NNil } y)) 0 \sigma) n) \sigma) \sigma))$
by (*simp add: 7 9 nfilter-nkfilter-ntaken-1 nkfilter-nlength*)
have 11: $(\text{ndropn } n (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma)) = (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{ndropn } ((\text{nnth } (\text{nkfilter } (\lambda y. w (\text{NNil } y)) 0 \sigma) n) \sigma) \sigma))$
by (*simp add: 7 9 nfilter-nkfilter-ndropn-1 nkfilter-nlength*)
have 12: $g (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{ntaken } ((\text{nnth } (\text{nkfilter } (\lambda y. w (\text{NNil } y)) 0 \sigma) n) \sigma) \sigma))$
using 10 7 **by** *auto*
have 13: $h (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{ndropn } ((\text{nnth } (\text{nkfilter } (\lambda y. w (\text{NNil } y)) 0 \sigma) n) \sigma) \sigma))$
using 11 7 **by** *auto*
have 14: $((\text{nnth } (\text{nkfilter } (\lambda y. w (\text{NNil } y)) 0 \sigma) n) \leq \text{nlength } \sigma$
using 7 9 *nkfilter-upperbound*[*of* $\sigma (\lambda y. w (\text{NNil } y)) - 0]$
by (*metis gen-nlength-def nkfilter-nlength nlength-code*)
have 15: $w (\text{NNil } (\text{nnth } (\text{ndropn } ((\text{nnth } (\text{nkfilter } (\lambda y. w (\text{NNil } y)) 0 \sigma) n) \sigma) 0))$
using 7 9 *nkfilter-nmap-nfilter*[*of* $\sigma \lambda x. w (\text{NNil } x)]$ *nkfilter-nnth-aa*[*of* $\sigma \lambda x. w (\text{NNil } x)]$
by *simp*

(metis (mono-tags, lifting) 7 nkfilter-nlength nnth-nmap)
have 16: $(\exists i. (enat\ i) \leq nlength\ (ntaken\ ((nnth\ (nkfilter\ (\lambda y. w\ (NNil\ y))\ 0\ \sigma)\ n)\ \sigma) \wedge$
 $w\ (NNil\ (nnth\ (ndropn\ i\ (ntaken\ ((nnth\ (nkfilter\ (\lambda y. w\ (NNil\ y))\ 0\ \sigma)\ n)\ \sigma))\ 0)))$
using 14 15
by (metis le-cases min.orderE ndropn-0 ndropn-nfirst ntaken-nlength ntaken-nnth)
have 17: $(\exists i. (enat\ i) \leq nlength\ (ndropn\ ((nnth\ (nkfilter\ (\lambda y. w\ (NNil\ y))\ 0\ \sigma)\ n)\ \sigma) \wedge$
 $w\ (NNil\ (nnth\ (ndropn\ (i + ((nnth\ (nkfilter\ (\lambda y. w\ (NNil\ y))\ 0\ \sigma)\ n)\ \sigma)\ 0)))$
using 15 by (metis add.left-neutral zero-enat-def zero-le)
have 18: $(\exists n. (enat\ n) \leq nlength\ \sigma \wedge$
 $(\exists i. (enat\ i) \leq nlength\ (ntaken\ n\ \sigma) \wedge w\ (NNil\ (nnth\ (ndropn\ i\ (ntaken\ n\ \sigma))\ 0)) \wedge$
 $g\ (nfilt\ (\lambda y. w\ ((NNil\ y)))\ (ntaken\ n\ \sigma)) \wedge$
 $w\ (NNil\ (nnth\ (ndropn\ n\ \sigma)\ 0)) \wedge$
 $(\exists i. (enat\ i) \leq nlength\ (ndropn\ n\ \sigma) \wedge w\ (NNil\ (nnth\ (ndropn\ (i + n)\ \sigma)\ 0))) \wedge$
 $h\ (nfilt\ (\lambda y. w\ ((NNil\ y)))\ (ndropn\ n\ \sigma)))$
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: $(\sigma \models ((init\ w) \amalg g) \frown ((init\ w) \wedge ((init\ w) \amalg h)))$
using 20 by (simp add: add.commute itl-defs ndropn-ndropn pi-d-def)
show ?thesis by (simp add: 21)
qed

lemma PiSChopDistsemb:

assumes $(\sigma \models ((init\ w) \amalg g) \frown ((init\ w) \wedge ((init\ w) \amalg h)))$

shows $(\sigma \models (init\ w) \amalg (g \frown h))$

proof –

have 1: $(\sigma \models ((init\ w) \amalg g) \frown ((init\ w) \wedge ((init\ w) \amalg h)))$

using *assms* **by** *auto*

have 2: $(\exists n. (enat\ n) \leq nlength\ \sigma \wedge$
 $((ntaken\ n\ \sigma) \models ((init\ w) \amalg g)) \wedge$
 $((ndropn\ n\ \sigma) \models ((init\ w) \wedge ((init\ w) \amalg h))))$

```

using assms schop-defs by blast
obtain n where 3:  $(\text{enat } n) \leq \text{nlength } \sigma \wedge ((\text{ntaken } n \ \sigma) \models ((\text{init } w) \amalg g)) \wedge$ 
 $((\text{ndropn } n \ \sigma) \models ((\text{init } w) \wedge ((\text{init } w) \amalg h)))$ 
using 2 by auto
have 4:  $((\exists i. (\text{enat } i) \leq \text{nlength } (\text{ntaken } n \ \sigma) \wedge (\text{LIFT}(\text{init } w)) (\text{ndropn } i (\text{ntaken } n \ \sigma))) \wedge$ 
 $((\text{pifilt } (\text{ntaken } n \ \sigma) (\text{LIFT}(\text{init } w))) \models g)$ 
)
by (meson 3 pi-d-def)
have 5:  $(\exists i. (\text{enat } i) \leq \text{nlength } (\text{ntaken } n \ \sigma) \wedge (\text{LIFT}(\text{init } w)) (\text{ndropn } i (\text{ntaken } n \ \sigma)))$ 
using 4 by auto
have 6:  $g (\text{pifilt } (\text{ntaken } n \ \sigma) (\text{LIFT}(\text{init } w)))$ 
using 4 by auto
have 7:  $g (\text{nfilter } (\lambda y. w ((\text{NNil } y))) (\text{ntaken } n \ \sigma))$ 
using 5 6 pifilt-init by (metis)
have 8:  $((\exists i. (\text{enat } i) \leq \text{nlength } (\text{ndropn } n \ \sigma) \wedge (\text{LIFT}(\text{init } w)) (\text{ndropn } i (\text{ndropn } n \ \sigma))) \wedge$ 
 $((\text{pifilt } (\text{ndropn } n \ \sigma) (\text{LIFT}(\text{init } w))) \models h)$ 
)
by (metis 3 intensional-rews(3) pi-d-def)
have 9:  $(\exists i. (\text{enat } i) \leq \text{nlength } (\text{ndropn } n \ \sigma) \wedge (\text{LIFT}(\text{init } w)) (\text{ndropn } i (\text{ndropn } n \ \sigma)))$ 
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 pifilt-init by metis
have 12:  $(\lambda y. w ((\text{NNil } y))) (\text{nlast } (\text{ntaken } n \ \sigma))$ 
by (metis 3 init-defs intensional-rews(3) ndropn-nfirst ntaken-0 ntaken-nlast)
have 13:  $\exists x \in \text{nset } \sigma. (\lambda y. w ((\text{NNil } y))) x$ 
using 12 3 by (metis in-nset-conv-nnth ntaken-nlast)
have 14:  $\exists x \in \text{nset } (\text{ntaken } n \ \sigma) . (\lambda y. w ((\text{NNil } y))) x$ 
using 12 using nfinite-ntaken 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 dual-order.order-iff-strict ndropn-Suc-conv-ndropn ndropn-nlast
nellist.set-intros(1) nellist.set-intros(2) nfinite-ntaken ntaken-all ntaken-nlast 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 nfilter-chop1-ntaken[of n σ (λy. w (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 nfilter-chop1-ndropn[of n σ (λy. w (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 enat-the-enat infinity-ileE le-cases ntaken-all)
have 19:  $\text{nfilter } (\lambda y. w ((\text{NNil } y))) \sigma \models g \smallfrown h$ 
by (simp add: 18 schop-defs)
have 20:  $((\text{pifilt } \sigma (\text{LIFT}(\text{init } w))) \models g \smallfrown h)$ 
by (metis 19 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 (simp add: 20 21 pi-d-def)
 qed

lemma *PiSchopDistsem*:

$\sigma \models (init\ w) \amalg (g \smallfrown h) = ((init\ w) \amalg g) \smallfrown ((init\ w) \wedge ((init\ w) \amalg h))$
 using *PiSchopDistsema PiSchopDistsemb unl-lift2* by blast

lemma *PiSchopDist*:

$\vdash (init\ w) \amalg (g \smallfrown h) = ((init\ w) \amalg g) \smallfrown ((init\ w) \wedge ((init\ w) \amalg h))$
 using *PiSchopDistsem Valid-def* by blast

13.3.12 PiProp

lemma *Pistate*:

$(\sigma \models (init\ w) \amalg f) =$
 $((\exists x \in nset\ \sigma. w\ (NNil\ x)) \wedge ((nfilter\ (\lambda y. w\ (NNil\ y))\ \sigma) \models f))$

proof –

have 1: $(\sigma \models (init\ w) \amalg f) =$
 $((\exists i \leq nlength\ \sigma. w\ (NNil\ (nnth\ \sigma\ i))) \wedge ((pifilt\ \sigma\ (LIFT\ (init\ w))) \models f))$
by (auto simp add: pi-d-def init-defs ndropn-nfirst)
have 2: $(\exists i \leq nlength\ \sigma. (LIFT\ (init\ w))\ (ndropn\ i\ \sigma)) =$
 $(\exists i \leq nlength\ \sigma. w\ (NNil\ (nnth\ \sigma\ i)))$
by (auto simp add: init-defs ndropn-nfirst)
have 3: $(\exists i \leq nlength\ \sigma. w\ (NNil\ (nnth\ \sigma\ i))) \longrightarrow$
 $(pifilt\ \sigma\ (LIFT\ (init\ w))) = (nfilter\ (\lambda y. w\ (NNil\ y))\ \sigma)$
using pifilt-init **using** 2 **by** blast
have 4: $(\exists i. (enat\ i) \leq nlength\ \sigma \wedge w\ (NNil\ (nnth\ \sigma\ i))) = (\exists x \in nset\ \sigma. w\ (NNil\ x))$
using in-nset-conv-nnth **by** force
show ?thesis
using 1 3 4 **by** auto
 qed

lemma *PiPropsem1a*:

$(\sigma \models f \amalg \$p) =$
 $((\exists i. (enat\ i) \leq nlength\ \sigma \wedge f\ (ndropn\ i\ \sigma)) \wedge p\ (nnth\ \sigma\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)))$
using sfxfilt-pifilt-nnth-ndropn[of $\sigma\ f$]
by (simp add: pi-d-def current-val-d-def pifilt-def)
 $(metis\ ndropn-0\ ndropn-nfirst\ nnth-nmap\ zero-enat-def\ zero-order(1))$

lemma *PiPropsem2a*:

$(\sigma \models (\neg f) \mathcal{U} (f \wedge \$p)) =$
 $(\exists k \leq nlength\ \sigma. f\ (ndropn\ k\ \sigma) \wedge p\ (nnth\ \sigma\ k) \wedge (\forall j < k. \neg f\ (ndropn\ j\ \sigma)))$
by (simp add: until-d-def current-val-d-def ndropn-nfirst)

lemma *PiPropsem3a*:

assumes $(\sigma \models f \amalg \$p)$
shows $(\sigma \models (\neg f) \mathcal{U} (f \wedge \$p))$
proof –
have 1: $((\exists i. (enat\ i) \leq nlength\ \sigma \wedge f\ (ndropn\ i\ \sigma)) \wedge$

$p \ (nnth \ \sigma \ (nnth \ (nkfilter \ f \ 0 \ (ndropns \ \sigma)) \ 0) \) \)$
using *assms PiPropsem1a* **by** *auto*
have 2: $\exists x \in nset(ndropns \ \sigma). f \ x$
using 1 **by** (*simp add: sfxfilter-help*)
have 3: $\forall x \in nset(nkfilter \ f \ 0 \ (ndropns \ \sigma)). f \ (nnth \ (ndropns \ \sigma) \ x)$
using 2 *nkfilter-holds* **by** *fastforce*
have 31: $(nnth \ (nkfilter \ f \ 0 \ (ndropns \ \sigma)) \ 0) \leq nlength \ \sigma$
by (*metis 1 2 dual-order.strict-trans2 enat-ord-simps(2) leI ndropns-nnth nkfilter-not-before*)
have 4: $f \ (ndropn \ (nnth \ (nkfilter \ f \ 0 \ (ndropns \ \sigma)) \ 0) \ \sigma)$
by (*metis 3 31 in-nset-ndropns in-nset-ndropns-nhd ndropn-0 ndropns-nnth zero-enat-def zero-le*)
have 5: $(\forall j < (nnth \ (nkfilter \ f \ 0 \ (ndropns \ \sigma)) \ 0). \neg f \ (ndropn \ j \ \sigma))$
using *nkfilter-not-before[of ndropns σ f] 2*
by (*simp add: 31 less-imp-le ndropns-nnth order-less-le-subst2*)
have 6: $(\exists k \leq nlength \ \sigma. f \ (ndropn \ k \ \sigma) \wedge p \ (nnth \ \sigma \ k) \wedge (\forall j < k. \neg f \ (ndropn \ j \ \sigma)))$
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: $(\exists k \leq nlength \ \sigma. f \ (ndropn \ k \ \sigma) \wedge p \ (nnth \ \sigma \ k) \wedge (\forall j < k. \neg f \ (ndropn \ j \ \sigma)))$

using *assms PiPropsem2a* **by** *auto*

obtain *k* **where** 2: $k \leq nlength \ \sigma \wedge f \ (ndropn \ k \ \sigma) \wedge p \ (nnth \ \sigma \ k) \wedge (\forall j < k. \neg f \ (ndropn \ j \ \sigma))$

using 1 **by** *auto*

have 3: $(\exists i. (enat \ i) \leq nlength \ \sigma \wedge f \ (ndropn \ i \ \sigma))$

using 2 **by** *blast*

have 31: $\exists x \in nset(ndropns \ \sigma). f \ x$

using 2 *in-nset-ndropns* **by** *auto*

have 331: $(nnth \ (nkfilter \ f \ 0 \ (ndropns \ \sigma)) \ 0) \leq nlength \ \sigma$

by (*metis 31 gen-nlength-def ndropns-nlength nkfilter-upperbound nlength-code zero-enat-def zero-le*)

have 32: $f \ (ndropn \ (nnth \ (nkfilter \ f \ 0 \ (ndropns \ \sigma)) \ 0) \ \sigma)$

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 \ (nnth \ \sigma \ (nnth \ (nkfilter \ f \ 0 \ (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*

13.3.13 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: 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 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** $(\text{metis } \text{enat-minus-mono1 } \text{idiff-enat-enat})$
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 $(\text{metis } \text{One-nat-def } \text{ileI1 } \text{nfilter-ndropns-nmap } \text{nkfilter-nlength } \text{one-eSuc } \text{one-enat-def})$
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 $(\text{ndropns } \sigma) \ f]$
by $(\text{metis } \text{enat-ord-simps}(1) \ \text{less-imp-le } \text{ndropns-nnth } \text{order.trans})$
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** $(\text{simp add: } \text{PiNextsem2})$
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** $(\text{simp add: } \text{PiNextsem2})$

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*
(metis (no-types, lifting) add commute eSuc-enat enat.simps(3) enat-add-sub-same
enat-less-enat-plusI2 ileI1 iless-Suc-eq less-add-Suc2 less-eqE ndropn-Suc-conv-ndropn
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: $g \ (\text{ndropn } (\text{Suc } 0) \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfilter } f \ (\text{ndropns } \sigma))))$
using 2 3 5 6 *nkfilter-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: $((\exists i. (\text{enat } i) \leq \text{nlength } \sigma \wedge f \ (\text{ndropn } i \ \sigma)) \wedge$
 $0 < \text{nlength } (\text{nfilter } f \ (\text{ndropns } \sigma)) \wedge$
 $g \ (\text{ndropn } (\text{Suc } 0) \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfilter } f \ (\text{ndropns } \sigma))))$
using 6 8 9 **by** *blast*
show ?thesis **using** 10
by (*simp* *add: PiNextsem1*)
qed

lemma *PiNextsem*:

$(\sigma \models f \ \Pi \ (\bigcirc g) = (\neg f) \ \mathcal{U} \ (f \wedge \bigcirc(f \ \Pi \ g)))$
using *PiNextsem3* *PiNextsem4* *unl-lift2* **by** *blast*

lemma *PiNext*:

$\vdash f \ \Pi \ (\bigcirc g) = (\neg f) \ \mathcal{U} \ (f \wedge \bigcirc(f \ \Pi \ g))$
using *PiNextsem* *Valid-def* **by** *blast*

13.3.14 PiUntil

lemma *PiUntilDistsem1*:

$(\sigma \models f \ \Pi \ (g \ \mathcal{U} \ h)) =$
 $((\exists i \leq \text{nlength } \sigma. f \ (\text{ndropn } i \ \sigma)) \wedge$
 $(\exists k \leq \text{nlength } (\text{nfilter } f \ (\text{ndropns } \sigma)).$
 $h \ (\text{ndropn } k \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfilter } f \ (\text{ndropns } \sigma)))) \wedge$
 $(\forall j < k. g \ (\text{ndropn } j \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfilter } f \ (\text{ndropns } \sigma))))))$
by (*simp* *add: pi-d-def* *pifilt-def* *sxfilt-def* *until-d-def*)

lemma *PiUntilDistsem2*:

$(\sigma \models (f \Pi g) \mathcal{U} (f \Pi h)) =$
 $(\exists k \leq \text{nlength } \sigma.$
 $(\exists i \leq \text{nlength } \sigma - k. f (\text{ndropn } (i + k) \sigma)) \wedge$
 $h (\text{nmap } (\lambda s. \text{nnth } s 0) (\text{nfilter } f (\text{ndropns } (\text{ndropn } k \sigma)))) \wedge$
 $(\forall j < k. (\exists i \leq \text{nlength } \sigma - j. f (\text{ndropn } (i + j) \sigma)) \wedge$
 $g (\text{nmap } (\lambda s. \text{nnth } s 0) (\text{nfilter } f (\text{ndropns } (\text{ndropn } j \sigma))))))$
by (*simp add: until-d-def pi-d-def pifilt-def sfxfilt-def ndropn-ndropn add commute*)

lemma *cover*:

assumes $(\exists i < k. \text{ff}(i) < (j::\text{nat}) \wedge j \leq \text{ff}(\text{Suc } i))$
 $(\forall i < k. \text{ff}(i) < \text{ff}(\text{Suc } i))$
shows $\text{ff}(0) < j \wedge j \leq \text{ff}(k)$
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.premis(1) less-SucE* **by** *blast*
have 2: $(\forall i < k. \text{ff}(i) < \text{ff}(\text{Suc } i))$
using *Suc.premis(2)* **by** *auto*
have 3: $\text{ff } k < \text{ff}(\text{Suc } k)$
by (*simp add: Suc.premis(2)*)
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 $(\forall j.$
 $(j \leq \text{ff } 0) \vee (\exists i < k. \text{ff}(i) < (j::\text{nat}) \wedge j \leq \text{ff}(\text{Suc } i))$
 $\longrightarrow \text{gg } j)$
 $(\forall i < k. \text{ff}(i) < \text{ff}(\text{Suc } i))$
shows $(\forall j < \text{ff } k. \text{gg } j)$
proof –
have 1: $(\forall j < \text{ff } 0. \text{gg } j)$
by (*simp add: assms(1)*)

```

have 2: (∀ j. ff 0 < j ∧ j ≤ ff k → gg j)
proof
  fix j
  show ff 0 < j ∧ j ≤ ff k → 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: (∀ j. (j ≤ ff 0) ∨ (∃ i < k. ff(i) < (j::nat) ∧ j ≤ ff(Suc i))
        ∨ (ff k < j ∧ j ≤ ff (Suc k) ) ) → gg j)
        using Suc.prem1(1) less-SucI by blast
      have 22: (∀ i < k. ff(i) < ff(Suc i))
        using Suc.prem1(2) by auto
      have 23: ff k < ff (Suc k)
        by (simp add: Suc.prem1(2))
      have 24: (∀ 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: sfilter-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 pifilt-def sxfilt-def sxfilt-pifilt-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 → f (ndropn (nnth (nkfilter f 0 (ndropns σ)) (k - 1)) σ)
  using 3 2 71 72 nkfilter-upperbound[of (ndropns σ) f k - 1 0]
  by (metis add-0 ndropns-nlength ndropns-nnth zero-enat-def)
have 9: k > 0 → (nnth (nkfilter f 0 (ndropns σ)) (k - 1)) < (nnth (nkfilter f 0 (ndropns σ)) k)
  by (metis 2 5 One-nat-def Suc-pred nkfilter-mono nkfilter-nlength)
have 10: k > 0 → (nnth (nkfilter f 0 (ndropns σ)) (k - 1)) ≤ nlength σ
  by (metis 2 71 gen-nlength-def in-nset-conv-nnth ndropns-nlength nkfilter-upperbound
    nlength-code)
have 11: k > 0 → (nnth (nkfilter f 0 (ndropns σ)) k) ≤ nlength σ
  using nkfilter-upperbound[of ndropns σ f k 0]
  using 51 by blast
have 12: k > 0 →
  h (nmap (λs. nnth s 0)
    (nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) k) σ))))
  using nfilter-nkfilter-ndropn[of f (ndropns σ) k]
    ndropns-nfilter-ndropn-a[of k f σ]
  by (metis 2 5 ndropn-nmap)
have 121: k > 0 →
  h (nmap (λs. nnth s 0)
    (nfilter f (ndropns (ndropn (Suc (nnth (nkfilter f 0 (ndropns σ)) (k - 1))) σ))))
  by (metis 2 5 One-nat-def Suc-pred nfilter-ndropns-nmap)
have 13: k > 0 → (∃ i ≤ nlength σ - (nnth (nkfilter f 0 (ndropns σ)) k).
  f (ndropn (i + (nnth (nkfilter f 0 (ndropns σ)) k)) σ))
  using 6 by (metis add.left-neutral zero-enat-def zero-le)
have 130: k > 0 → (∃ i. i + (Suc (nnth (nkfilter f 0 (ndropns σ)) (k - 1))) ≤ nlength σ ∧
  (ndropn (i + (Suc (nnth (nkfilter f 0 (ndropns σ)) (k - 1))) σ) =
    (ndropn (nnth (nkfilter f 0 (ndropns σ)) k) σ))
  using 9 by (metis 11 Suc-leI diff-add)
have 131: k > 0 → (∃ i. i + (Suc (nnth (nkfilter f 0 (ndropns σ)) (k - 1))) ≤ nlength σ ∧

```

```

      f (ndropn (i + (Suc (nnth (nkfilter f 0 (ndropns σ)) (k-1)))) σ))
using 10 6 9 11 130 by auto
have 132: k>0 ⟶ (∃ i≤nlength σ - (Suc (nnth (nkfilter f 0 (ndropns σ)) (k-1))).
      f (ndropn (i + (Suc (nnth (nkfilter f 0 (ndropns σ)) (k-1)))) σ))
using 131
by (metis add-diff-cancel-right' enat-minus-mono1 idiff-enat-enat)
have 14: k>0 ⟶ (∀ j< (nnth (nkfilter f 0 (ndropns σ)) k).
      (∃ i. i+j≤ nlength σ ∧ f (ndropn (i + j) σ)))
using 131 by (metis 11 6 diff-add less-imp-le plus-enat-simps(1))
have 141: k>0 ⟶ (∀ j< (Suc (nnth (nkfilter f 0 (ndropns σ)) (k-1))).
      (∃ i. i+j≤ nlength σ ∧ f (ndropn (i + j) σ)))
using 14 9 by auto
have 142: k>0 ⟶ (∀ j< (Suc (nnth (nkfilter f 0 (ndropns σ)) (k-1))).
      (∃ i≤nlength σ -j. f (ndropn (i + j) σ)))
using 141 by (metis add.commute enat.simps(3) enat-add-sub-same enat-minus-mono1)
have 15: (∀ j<k. g (ndropn j (nmap (λs. nnth s 0) (nfilter f (ndropns σ)))))
using 5 by blast
have 151: (∀ j<k. g (nmap (λs. nnth s 0) (ndropn j (nfilter f (ndropns σ)))))
by (metis 15 ndropn-nmap)
have 152: (∀ j<k. (ndropn j (nfilter f (ndropns σ))) =
      (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:  $k > 0 \longrightarrow (\forall j \leq (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0).
  ( (\text{ndropns } (\text{ndropn } j \ \sigma))) =
  ( (\text{ndropn } j \ (\text{ndropns } \sigma))))$ 

using 200 ndropn-ndropns by (simp add: ndropn-ndropns order-subst2)
have 22:  $k > 0 \longrightarrow (\forall j \leq (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0).
  (\text{nfilter } f \ (\text{ndropns } (\text{ndropn } j \ \sigma))) =
  (\text{nfilter } f \ (\text{ndropn } j \ (\text{ndropns } \sigma))))$ 

using 21 by auto
have 23:  $k > 0 \longrightarrow
  (\forall j \leq (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0).
    (\text{nmap } (\lambda s. \text{nnth } s \ 0)
      (\text{nfilter } f \ (\text{ndropns } (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0) \ \sigma)))) =
      (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f \ (\text{ndropns } (\text{ndropn } j \ \sigma))))))
  )$ 
by (simp add: 2 22 nfilter-ndropns-nmap-help-0)
have 24:  $k > 0 \longrightarrow
  (\forall j \leq (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0).
    g \ (\text{nmap } (\lambda s. \text{nnth } s \ 0)
      (\text{nfilter } f \ (\text{ndropns } (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0) \ \sigma)))) =
      g \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f \ (\text{ndropns } (\text{ndropn } j \ \sigma))))))
  )$ 
using 23 by auto
have 241:  $k > 0 \longrightarrow (\forall j \leq (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0).
  g \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f \ (\text{ndropns } (\text{ndropn } j \ \sigma)))) )$ 

using 19 24 by blast
have 25:  $k > 0 \longrightarrow (\forall i < k - 1.
  (\forall l. l \leq (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (\text{Suc } i)) \wedge
    (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ i) < l \longrightarrow
    (\text{nfilter } f \ (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (\text{Suc } i)) \ (\text{ndropns } \sigma)) =
    (\text{nfilter } f \ (\text{ndropn } l \ (\text{ndropns } \sigma)))) )$ 

proof auto
  fix i
  fix l
  assume a0:  $0 < k$ 
  assume a1:  $i < k - \text{Suc } 0$ 
  assume a2:  $l \leq \text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (\text{Suc } i)$ 
  assume a3:  $\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ i < l$ 
  show  $\text{nfilter } f \ (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (\text{Suc } i)) \ (\text{ndropns } \sigma)) =
    \text{nfilter } f \ (\text{ndropn } l \ (\text{ndropns } \sigma))$ 
  proof -
    have 251:  $k = 1 \implies
      \text{nfilter } f \ (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (\text{Suc } i)) \ (\text{ndropns } \sigma)) =
      \text{nfilter } f \ (\text{ndropn } l \ (\text{ndropns } \sigma))$ 
    using a1 by force
    have 2511:  $1 < k \implies \text{enat } (\text{Suc } i) \leq \text{nlength } (\text{nfilter } f \ (\text{ndropns } \sigma))$ 
    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:  $1 < k \implies
      \text{nfilter } f \ (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (\text{Suc } i)) \ (\text{ndropns } \sigma)) =$ 

```

```

      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:  $k > 0 \longrightarrow (\forall i < k - 1.$ 
     $(\forall l. l \leq (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (\text{Suc } i)) \wedge$ 
     $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ i) < l \longrightarrow$ 
     $l \leq \text{nlength } (\text{ndropns } \sigma))$ 
    using 1612 by (meson diff-less enat-ord-simps(1) less-one less-trans-Suc order-subst2)
  have 262:  $k > 0 \longrightarrow (\forall i < k - 1.$ 
     $(\forall l. l \leq (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (\text{Suc } i)) \wedge$ 
     $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ i) < l \longrightarrow$ 
     $(\text{ndropn } l \ (\text{ndropns } \sigma)) = (\text{ndropns } (\text{ndropn } l \ \sigma)))$ 
    using 261 by (simp add: ndropn-ndropns)
  have 26:  $k > 0 \longrightarrow (\forall i < k - 1.$ 
     $(\forall l. l \leq (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (\text{Suc } i)) \wedge$ 
     $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ i) < l \longrightarrow$ 
     $(\text{nmap } (\lambda s. \text{nnth } s \ 0)$ 
     $(\text{nfilter } f \ (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (\text{Suc } i)) \ (\text{ndropns } \sigma)))) =$ 
     $(\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f \ (\text{ndropns } (\text{ndropn } l \ \sigma)))$ 
    using 25 262 by auto
  have 27:  $k > 0 \longrightarrow (\forall i < k - 1.$ 
     $(\forall l. l \leq (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (\text{Suc } i)) \wedge$ 
     $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ i) < l \longrightarrow$ 
     $g \ (\text{nmap } (\lambda s. \text{nnth } s \ 0)$ 
     $(\text{nfilter } f$ 
     $(\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (\text{Suc } i)) \ (\text{ndropns } \sigma))))))$ 
    by (simp add: 16)
  have 28:  $k > 0 \longrightarrow (\forall i < k - 1.$ 
     $(\forall l. l \leq (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (\text{Suc } i)) \wedge$ 
     $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ i) < l \longrightarrow$ 
     $g \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f \ (\text{ndropns } (\text{ndropn } l \ \sigma)))$ 
    using 25 262 27 by auto
  have 281:  $k > 0 \longrightarrow$ 
     $(\forall j.$ 
     $(j \leq (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0) \vee$ 
     $(\exists i. i < k - 1 \wedge$ 
     $j \leq (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (\text{Suc } i)) \wedge$ 
     $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ i) < j) \longrightarrow$ 
     $g \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f \ (\text{ndropns } (\text{ndropn } j \ \sigma)))$ 
    using 241 28 by blast
  have 282:  $k > 0 \longrightarrow (\forall i < k - 1.$ 
     $\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ i < \text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (\text{Suc } i))$ 
    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:  $k > 0 \longrightarrow (\forall j < (\text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (k - 1)))).$ 
     $g \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f \ (\text{ndropns } (\text{ndropn } j \ \sigma))))$ 
    using 281 282 cover-a[of λ x. nnth (nkfilter f 0 (ndropns σ)) x k-1]

```


$(\lambda j. g (nmap (\lambda s. nnth s 0) (nfilter f (ndropns (ndropn j \sigma))))]$
using 18 less-antisym **by** blast
have 30: $k > 0 \longrightarrow (\exists k \leq nlength \sigma.$
 $(\exists i \leq nlength \sigma - k. f (ndropn (i + k) \sigma)) \wedge$
 $h (nmap (\lambda s. nnth s 0) (nfilter f (ndropns (ndropn k \sigma)))) \wedge$
 $(\forall j < k. (\exists i \leq nlength \sigma - j. f (ndropn (i + j) \sigma)) \wedge$
 $g (nmap (\lambda s. nnth s 0) (nfilter f (ndropns (ndropn j \sigma)))))$
using 10 11 121 132 142 29 9 **by** simp
 $(metis add-Suc-right antisym-conv2 eSuc-enat enat-ord-simps(1) ileI1 leD)$
have 31: $(\exists k \leq nlength \sigma.$
 $(\exists i \leq nlength \sigma - k. f (ndropn (i + k) \sigma)) \wedge$
 $h (nmap (\lambda s. nnth s 0) (nfilter f (ndropns (ndropn k \sigma)))) \wedge$
 $(\forall j < k. (\exists i \leq nlength \sigma - j. f (ndropn (i + j) \sigma)) \wedge$
 $g (nmap (\lambda s. nnth s 0) (nfilter f (ndropns (ndropn j \sigma)))))$
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: $(\exists k \leq nlength \sigma.$
 $(\exists i \leq nlength \sigma - k. f (ndropn (i + k) \sigma)) \wedge$
 $h (nmap (\lambda s. nnth s 0) (nfilter f (ndropns (ndropn k \sigma)))) \wedge$
 $(\forall j < k.$
 $(\exists i \leq nlength \sigma - j. f (ndropn (i + j) \sigma)) \wedge$
 $g (nmap (\lambda s. nnth s 0) (nfilter f (ndropns (ndropn j \sigma)))))$
using *assms* **by** $(simp add: PiUntilDistsem2)$
obtain k **where** 2: $k \leq nlength \sigma \wedge$
 $(\exists i \leq nlength \sigma - k. f (ndropn (i + k) \sigma)) \wedge$
 $h (nmap (\lambda s. nnth s 0) (nfilter f (ndropns (ndropn k \sigma)))) \wedge$
 $(\forall j < k.$
 $(\exists i \leq nlength \sigma - j. f (ndropn (i + j) \sigma)) \wedge$
 $g (nmap (\lambda s. nnth s 0) (nfilter f (ndropns (ndropn j \sigma)))))$
using 1 **by** auto
have 3: $(\exists i \leq nlength \sigma - k. f (ndropn (i + k) \sigma))$
using 2 **by** auto
have 31: $(\exists i \leq nlength (ndropns (ndropn k \sigma)). f (nnth (ndropns (ndropn k \sigma)) i))$
by $(metis 3 add.commute ndropn-ndropn ndropn-nlength ndropns-nlength ndropns-nnth)$
have 4: $k \leq nlength \sigma$
using 2 **by** auto
obtain i **where** 5: $(enat i) \leq nlength (ndropns (ndropn k \sigma)) \wedge f (nnth (ndropns (ndropn k \sigma)) i) \wedge$
 $i = (LEAST na. enat na \leq nlength (ndropns (ndropn k \sigma)) \wedge$
 $f (nnth (ndropns (ndropn k \sigma)) na))$
by $(metis (no-types, lifting) 31 LeastI-ex)$
have 51: $i = (LEAST na. enat na \leq nlength (ndropns (ndropn k \sigma)) \wedge f (nnth (ndropns (ndropn k \sigma))$
 $na))$
using 5 **by** auto
have 60: $(enat i) \leq nlength \sigma - k$

```

  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
      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  $(\text{ndropns } \sigma) \ f$ ] sfxfilter-help[of  $\sigma$ ]
  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:  $f \ (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma)))) \ 0) \ (\text{ndropn } k \ \sigma)$ 
  using nkfilter-holds[of  $\text{ndropns } (\text{ndropn } k \ \sigma) \ f \ 0$ ]
  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:  $f \ (\text{ndropn } ((\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma)))) \ 0) + k) \ \sigma$ 
  using 73 by (simp add: add commute ndropn-ndropn)
have 75:  $f \ (\text{ndropn } k \ (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma)))) \ 0) \ \sigma)$ 
  by (simp add: 74 ndropn-ndropn)
have 76:  $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma)))) \ 0 = \text{nleast } f \ (\text{ndropns } (\text{ndropn } k \ \sigma))$ 
  by (simp add: 61 nkfilter-nleast)
have 77:  $\text{nleast } f \ (\text{ndropns } (\text{ndropn } k \ \sigma)) = (\text{LEAST } na. \text{ enat } na \leq \text{nlength } (\text{ndropns } (\text{ndropn } k \ \sigma)) \wedge$ 
 $f \ (\text{nnth } (\text{ndropns } (\text{ndropn } k \ \sigma)) \ na))$ 
  using 61 nleast-conv[of  $\text{ndropns } (\text{ndropn } k \ \sigma) \ f$ ] by auto
have 78:  $\text{nleast } f \ (\text{ndropns } (\text{ndropn } k \ \sigma)) = i$ 
  using 5 77 by blast
have 8:  $h \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfilter } f \ (\text{ndropns } (\text{ndropn } k \ \sigma))))$ 
  using 2 by auto
have 10:  $\text{nfirst } (\text{nkfilter } f \ k \ (\text{ndropn } k \ (\text{ndropns } \sigma))) =$ 
 $((\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma)))) \ 0) + k$ 
  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:  $\text{nfirst } (\text{nkfilter } f \ k \ (\text{ndropn } k \ (\text{ndropns } \sigma))) = k + i$ 
  by (simp add: 10 76 78)
have 90:  $f \ (\text{nlast } (\text{ntaken } ((\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma)))) \ 0) + k) \ (\text{ndropns } \sigma))$ 
  by (metis 6 74 76 78 ndropns-nnth ntaken-nlast)
have 91:  $((\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma)))) \ 0) + k \leq \text{nlength } (\text{ndropns } \sigma)$ 
  using nkfilter-upperbound[of  $(\text{ndropns } (\text{ndropn } k \ \sigma)) \ f \ 0 \ 0$ ]
      ndropn-nlength[of  $k \ \sigma$ ] ndropns-nlength[of  $\text{ndropn } k \ \sigma$ ]
  by (simp add: 6 76 78 ndropns-nlength)

```

have 92: $((\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma))) \ 0) + k) = ((\text{nnth } (\text{nkfilter } f \ k \ (\text{ndropns } (\text{ndropn } k \ \sigma))) \ 0))$
by (*simp add: 61 nkfilter-nleast*)
let ?kf = (*the-enat* $((\text{nlength}(\text{nfilter } f \ (\text{ntaken } ((\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma))) \ 0) + k) \ (\text{ndropns } \sigma))))$)
let ?nkf = (*the-enat* $((\text{nlength}(\text{nkfilter } f \ 0 \ (\text{ntaken } ((\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma))) \ 0) + k) \ (\text{ndropns } \sigma))))$)
let ?pkf = (*the-enat* $((\text{nlength } (\text{nkfilter } f \ 0 \ (\text{ntaken } k \ (\text{ndropns } \sigma))))$)
have 93: ?kf = ?nkf
by (*metis 90 nfinite-ntaken nkfilter-nlength nset-nlast*)
have 94: $((\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma))) \ 0) + k) \leq \text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ ?kf$
by (*metis 74 90 91 add.left-neutral eq-iff ndropn-nfirst ndropns-nlength ndropns-nnth nkfilter-chop1-ndropn nkfilter-nfirst*)
have 95: $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma))) \ 0) = 0 \implies k = \text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ ?nkf$
by (*metis 10 90 91 add-cancel-left-left ndropn-nfirst nkfilter-chop1-ndropn*)
have 98: $0 < ?nkf \wedge 0 < (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma))) \ 0) \implies \text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (?nkf - 1) < k$
proof –
assume b: $0 < ?nkf \wedge 0 < (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma))) \ 0)$
have 980: $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (?nkf - 1)) \in \text{nset } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma))$
by (*metis in-nset-conv-nnth le-cases nfinite-ntaken nset-nlast ntaken-all ntaken-nlast*)
have 981: $f \ (\text{nnth } (\text{ndropns } \sigma) \ (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (?nkf - 1)))$
using *nkfilter-holds*[of $(\text{ndropns } \sigma) \ f \ (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (?nkf - 1)) \ 0]$
using 71 980 **by** *auto*
have 982: $\neg f \ (\text{ndropn } k \ \sigma)$
by (*metis 10 4 add-cancel-left-left b gr-implies-not0 ndropn-nfirst ndropns-nnth nkfilter-nfirst*)
have 984: $f \ (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (?nkf - 1)) \ \sigma)$
using 71 980 981 *ndropns-nnth*[of $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (?nkf - 1)) \ \sigma]$
by (*metis gen-nlength-def in-nset-conv-nnth ndropns-nlength nkfilter-upperbound nlength-code*)
have 985: $\bigwedge j. k \leq j \implies j < k + i \implies \neg f \ (\text{ndropn } j \ \sigma)$
proof –
fix j
assume a0: $k \leq j$
assume a1: $j < k + i$
show $\neg f \ (\text{ndropn } j \ \sigma)$
proof –
have 9851: $\exists x \in \text{nset } (\text{ndropn } k \ (\text{ndropns } \sigma)). f \ x$
by (*simp add: 4 61 ndropn-ndropns ndropns-nlength*)
have 9852: $\text{enat } k \leq \text{nlength } (\text{ndropns } \sigma)$
by (*simp add: 4 ndropns-nlength*)
have 9853: $k \leq j$
by (*simp add: a0*)
have 9854: $j < \text{nnth } (\text{nkfilter } f \ k \ (\text{ndropn } k \ (\text{ndropns } \sigma))) \ 0$
using 10 101 92 9852 a1 *ndropn-ndropns* **by** *fastforce*
have 9855: $\neg f \ (\text{nnth } (\text{ndropns } \sigma) \ j)$
using 9851 9852 9853 9854 *nkfilter-n-not-before*[of $k \ \text{ndropns } \sigma \ f \ j]$ **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:  $\text{ntaken } ?\text{nkf } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) =$ 
     $\text{nkfilter } f \ 0 \ (\text{ntaken } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma))) \ 0 + k) \ (\text{ndropns } \sigma))$ 
    using nkfilter-chop1-ntaken[of  $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma))) \ 0 + k) \ (\text{ndropns } \sigma) \ f \ 0]$ 
    using 90 91 by blast
  have 987:  $\neg \text{enat } (? \text{nkf}) \leq \text{nlength } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \vee$ 
     $\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (? \text{nkf} - 1) < \text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (? \text{nkf})$ 
    using 71 by (meson b diff-less less-one nidx-nkfilter-gr)
  have 988:  $\text{nlast } (\text{nkfilter } f \ 0 \ (\text{ntaken } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma))) \ 0 + k) \ (\text{ndropns } \sigma)))$ 
    =
     $0 + (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma))) \ 0 + k)$ 
    by (simp add: 90 91 nkfilter-nlast)
  have 989:  $\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (? \text{nkf}) = \text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma))) \ 0 + k$ 
    using 986
    by (metis 988 add.left-neutral ntaken-nlast)
  have 990:  $\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (? \text{nkf}) = \text{nfirst } (\text{nkfilter } f \ k \ (\text{ndropn } k \ (\text{ndropns } \sigma)))$ 
    using 10 989 by fastforce
  have 991:  $\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ (? \text{nkf}) = k + i$ 
    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:  $h \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{ndropn } ?kf \ (\text{nfilter } f \ (\text{ndropns } \sigma))))$ 
    using ndropns-nfilter-ndropn-a[of  $- \ f \ \text{ndropn } k \ \sigma$ ]
     $\text{nfilter-chop1-ndropn}$ [of  $((\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma))) \ 0) + k) \ (\text{ndropns } \sigma) \ f \ ]$ 
    101 61 76 78 8 90 91
    by simp
    (metis 10 ndropn-0 ndropn-ndropn ndropn-ndropns zero-enat-def zero-le)
  have 11:  $h \ (\text{ndropn } ?kf \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfilter } f \ (\text{ndropns } \sigma))))$ 
    by (simp add: 100 ndropn-nmap)
  have 111:  $?kf \leq \text{nlength } (\text{nfilter } f \ (\text{ndropns } \sigma))$ 
    by (metis 90 enat-ile le-cases nfilter-chop1-ntaken ntaken-all the-enat.simps)
  have 112:  $\text{nfinite } (\text{nfilter } f \ (\text{ntaken } ((\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } (\text{ndropn } k \ \sigma))) \ 0) + k) \ (\text{ndropns } \sigma)))$ 
    by (metis 90 91 nfilter-chop1-ntaken nfinite-ntaken)
  have 13:  $(\forall j < k.$ 
     $(\exists x \in \text{nset}(\text{ndropns } (\text{ndropn } j \ \sigma)). \ f \ x) \wedge$ 
     $g \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfilter } f \ (\text{ndropns } (\text{ndropn } j \ \sigma))))$ 
    by (metis 2 add commute in-nset-ndropns ndropn-ndropn ndropn-nlength)
  have 151:  $(\forall jj < ?kf.$ 
     $(\text{ndropn } jj \ (\text{nfilter } f \ (\text{ndropns } \sigma))) =$ 
     $\text{nfilter } f \ (\text{ndropns } (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ jj) \ \sigma))$ 
    ) using nfilter-nkfilter-ndropn
    by (simp add: 111 71 less-imp-le ndropns-nfilter-ndropn-a order-less-le-subst2)
  have 152:  $(\bigwedge jj. (\text{enat } jj) < ?kf \implies (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ jj) < k)$ 
    proof –
      fix jj
      assume a:  $(\text{enat } jj) < ?kf$ 
      show  $\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ jj < k$ 
    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:
  σ ⊨ f Π (g U h) = ( f Π g ) U ( f Π h )
using PiUntilDistsem3 PiUntilDistsem4 using unl-lift2 by blast

lemma PiUntilDist:
  ⊢ f Π (g U h) = ( f Π g ) U ( f Π h )
using PiUntilDistsem Valid-def by blast

```

13.3.15 PiChopstar

lemma *wnextboznnotstatesem:*

assumes $k \leq \text{nlength } \sigma$

shows $(\forall j \leq \text{nlength } \sigma. k < j \longrightarrow \neg (\lambda y. w (\text{NNil } y)) (\text{nnth } \sigma j)) =$
 $(\text{LIFT}(w\text{next } (\Box (\text{init } (\neg w)))) (\text{ndropn } k \sigma))$

using *assms*

proof (*auto simp add: itl-defs ndropn-nfirst*)

show $\bigwedge n. \text{enat } k \leq \text{nlength } \sigma \Longrightarrow$

$\forall j. \text{enat } j \leq \text{nlength } \sigma \longrightarrow k < j \longrightarrow \neg w (\text{NNil } (\text{nnth } \sigma j)) \Longrightarrow$

$\text{nlength } \sigma - \text{enat } k \neq \text{enat } 0 \Longrightarrow$

$\text{enat } n \leq \text{nlength } \sigma - \text{enat } k - \text{enat } (\text{Suc } 0) \Longrightarrow w (\text{NNil } (\text{nnth } \sigma (\text{Suc } (k + n)))) \Longrightarrow \text{False}$

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 $\bigwedge j. \text{enat } k \leq \text{nlength } \sigma \Longrightarrow \text{enat } j \leq \text{nlength } \sigma \Longrightarrow k < j \Longrightarrow w (\text{NNil } (\text{nnth } \sigma j)) \Longrightarrow$
 $\text{nlength } \sigma - \text{enat } k = \text{enat } 0 \Longrightarrow \text{False}$

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 $\bigwedge j. \text{enat } k \leq \text{nlength } \sigma \Longrightarrow$

$\text{enat } j \leq \text{nlength } \sigma \Longrightarrow$

$k < j \Longrightarrow w (\text{NNil } (\text{nnth } \sigma j)) \Longrightarrow$

$\forall n. \text{enat } n \leq \text{nlength } \sigma - \text{enat } k - \text{enat } (\text{Suc } 0) \longrightarrow \neg w (\text{NNil } (\text{nnth } \sigma (\text{Suc } (k + n)))) \Longrightarrow \text{False}$

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*: $w (\text{NNil } (\text{nnth } \sigma j))$

assume *a4*: $\forall n. \text{enat } n \leq \text{nlength } \sigma - \text{enat } k - \text{enat } (\text{Suc } 0) \longrightarrow \neg w (\text{NNil } (\text{nnth } \sigma (\text{Suc } (k + n))))$

show *False*

proof –

have 1: $j = (\text{Suc } (k + (j - (\text{Suc } k))))$

using *a2* **by** *auto*

have 2: $\forall n. \text{enat } n + 1 + k \leq \text{nlength } \sigma \longrightarrow \neg w (\text{NNil } (\text{nnth } \sigma (\text{Suc } (k + n))))$

by *auto*

(*metis One-nat-def a4 add commute enat.simps(3) enat-add-sub-same enat-minus-mono1*
one-enat-def)

have 3: $(j - (\text{Suc } k)) + 1 + k \leq \text{nlength } \sigma$

using 1 *a1* **by** *simp*

have 4: $\neg w (\text{NNil } (\text{nnth } \sigma (\text{Suc } (k + (j - (\text{Suc } k))))))$

using 2 3 *eSuc-enat plus-1-eSuc(2)* **by** *auto*

from 1 4 *a3* **show** *?thesis* **by** *auto*

qed

qed

qed

lemma *NotStateUntilStateAndsem:*

$(\sigma \models (\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge f)) =$

$(\exists k. (enat\ k) \leq nlength\ \sigma \wedge w\ (NNil\ (nnth\ \sigma\ k)) \wedge f\ (ndropn\ k\ \sigma) \wedge (\forall j < k. \neg w\ (NNil\ (nnth\ \sigma\ j))))$

by (auto simp add: until-d-def init-defs ndropn-nfirst)

lemma *StateUntilEqvWPrevChopsem:*

$\sigma \models (init\ w)\ \mathcal{U}\ f = (wprev\ (\Box\ (init\ w))) \frown f$

proof (auto simp add: until-d-def itl-defs ntaken-nnth ndropn-nfirst min-absorb1)

show $\bigwedge k. enat\ k \leq nlength\ \sigma \implies$

$f\ (ndropn\ k\ \sigma) \implies$

$\forall j < k. w\ (NNil\ (nnth\ \sigma\ j)) \implies$

$\exists n. enat\ n \leq nlength\ \sigma \wedge$

$(min\ (enat\ n)\ (nlength\ \sigma) = enat\ 0 \vee$

$(\forall na. na \leq the-enat\ (min\ (enat\ (n - Suc\ 0))\ (epred\ (nlength\ \sigma)))) \wedge na \leq n \wedge enat\ na \leq$

$nlength\ \sigma \longrightarrow$

$w\ (NNil\ (nnth\ \sigma\ na)))) \wedge$

$f\ (ndropn\ n\ \sigma)$

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 $a0: enat\ n \leq nlength\ \sigma$

assume $a1: f\ (ndropn\ n\ \sigma)$

assume $a2: \forall na. na \leq the-enat\ (min\ (enat\ (n - Suc\ 0))\ (epred\ (nlength\ \sigma))) \wedge na \leq n \wedge$
 $enat\ na \leq nlength\ \sigma \longrightarrow w\ (NNil\ (nnth\ \sigma\ na))$

show $\exists k. enat\ k \leq nlength\ \sigma \wedge f\ (ndropn\ k\ \sigma) \wedge (\forall j < k. w\ (NNil\ (nnth\ \sigma\ j)))$

proof –

have 1: $enat\ n \leq nlength\ \sigma \wedge f\ (ndropn\ n\ \sigma)$

using $a0\ a1$ **by** auto

have 2: $(\forall na. na \leq the-enat\ (min\ (enat\ (n - Suc\ 0))\ (epred\ (nlength\ \sigma))) \wedge na \leq n \wedge$
 $enat\ na \leq nlength\ \sigma \longrightarrow w\ (NNil\ (nnth\ \sigma\ na))) \longrightarrow$

$(\forall j < n. w\ (NNil\ (nnth\ \sigma\ j)))$

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: $(\forall j < n. w\ (NNil\ (nnth\ \sigma\ j)))$

using 2 a2 **by** blast

show ?thesis

using 1 3 **by** blast

qed

qed

lemma *StateUntilEqvWPrevChop:*

$\vdash (init\ w)\ \mathcal{U}\ f = (wprev\ (\Box\ (init\ w))) \frown f$

using StateUntilEqvWPrevChopsem Valid-def **by** blast

lemma *UntilChopDist:*

$\vdash (init\ w)\ \mathcal{U}\ (g \frown h) = ((init\ w)\ \mathcal{U}\ g) \frown h$

using StateUntilEqvWPrevChop[of w]

by (metis SChopAssoc inteq-reflection)

lemma *PiEmptysem*:

$\sigma \models (\text{init } w) \Pi \text{ empty} = (\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge \text{wnext } (\Box (\text{init } (\neg w))))$

proof –

have 1: $(\sigma \models (\text{init } w) \Pi \text{ empty}) =$

$((\exists x \in \text{nset } \sigma. w (\text{NNil } x)) \wedge \text{nlength } (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma) = 0)$

by (*simp add: Pistate itl-defs zero-enat-def*)

have 2: $((\exists x \in \text{nset } \sigma. w (\text{NNil } x)) \wedge \text{nlength } (\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma) = 0) =$

$(\exists k \leq \text{nlength } \sigma. (\lambda y. w (\text{NNil } y)) (\text{nnth } \sigma k) \wedge$
 $(\forall j. j < k \longrightarrow \neg (\lambda y. w (\text{NNil } y)) (\text{nnth } \sigma j)) \wedge$
 $(\forall j \leq \text{nlength } \sigma. k < j \longrightarrow \neg (\lambda y. w (\text{NNil } y)) (\text{nnth } \sigma j)))$

by (*simp add: nfilter-nlength-zero-conv-2*)

have 3: $(\exists k \leq \text{nlength } \sigma. (\lambda y. w (\text{NNil } y)) (\text{nnth } \sigma k) \wedge$

$(\forall j. j < k \longrightarrow \neg (\lambda y. w (\text{NNil } y)) (\text{nnth } \sigma j)) \wedge$

$(\forall j \leq \text{nlength } \sigma. k < j \longrightarrow \neg (\lambda y. w (\text{NNil } y)) (\text{nnth } \sigma j))) =$

$(\exists k \leq \text{nlength } \sigma.$

$w (\text{NNil } (\text{nnth } \sigma k)) \wedge$

$(\text{LIFT}(\text{wnext } (\Box (\text{init } (\neg w)))) (\text{ndropn } k \sigma) \wedge$

$(\forall j < k. \neg w (\text{NNil } (\text{nnth } \sigma j))))$ (**is** ?lhs = ?rhs)

proof

assume a: ?lhs

show ?rhs **using** a *wnextboxnnotstatesem* **by** *auto*

next

assume b: ?rhs

show ?lhs

proof –

obtain k **where** 1:

$\text{enat } k \leq \text{nlength } \sigma \wedge w (\text{NNil } (\text{nnth } \sigma k)) \wedge$

$(\text{LIFT}(\text{wnext } (\Box (\text{init } (\neg w)))) (\text{ndropn } k \sigma) \wedge (\forall j < k. \neg w (\text{NNil } (\text{nnth } \sigma j))))$

using b **by** *auto*

have 2: $\text{enat } k \leq \text{nlength } \sigma \wedge w (\text{NNil } (\text{nnth } \sigma k)) \wedge (\forall j < k. \neg w (\text{NNil } (\text{nnth } \sigma j))) \wedge$

$(\forall j. \text{enat } j \leq \text{nlength } \sigma \longrightarrow k < j \longrightarrow \neg w (\text{NNil } (\text{nnth } \sigma j)))$

using 1 *wnextboxnnotstatesem*[of k σ w]

proof –

have $\forall x0. (x0 < k \longrightarrow \neg w (\text{NNil } (\text{nnth } \sigma x0))) = (\neg x0 < k \vee \neg w (\text{NNil } (\text{nnth } \sigma x0)))$

by *meson*

then have f1: $\text{enat } k \leq \text{nlength } \sigma \wedge w (\text{NNil } (\text{nnth } \sigma k)) \wedge$

$(\text{LIFT}(\text{wnext } (\Box (\text{init } (\neg w)))) (\text{ndropn } k \sigma) \wedge (\forall n. \neg n < k \vee \neg w (\text{NNil } (\text{nnth } \sigma n))))$

by (*metis* $\langle \text{enat } k \leq \text{nlength } \sigma \wedge w (\text{NNil } (\text{nnth } \sigma k)) \wedge$

$(\text{LIFT}(\text{wnext } (\Box (\text{init } (\neg w)))) (\text{ndropn } k \sigma) \wedge (\forall j < k. \neg w (\text{NNil } (\text{nnth } \sigma j))) \rangle$

have $(\text{enat } k \leq \text{nlength } \sigma \longrightarrow (\forall n. \text{enat } n \leq \text{nlength } \sigma \wedge k < n \longrightarrow \neg w (\text{NNil } (\text{nnth } \sigma n)))) =$

$(\text{LIFT}(\text{wnext } (\Box (\text{init } (\neg w)))) (\text{ndropn } k \sigma)) =$

$(\neg \text{enat } k \leq \text{nlength } \sigma \vee (\forall n. (\neg \text{enat } n \leq \text{nlength } \sigma \vee \neg k < n) \vee \neg w (\text{NNil } (\text{nnth } \sigma n)))) =$

$(\text{LIFT}(\text{wnext } (\Box (\text{init } (\neg w)))) (\text{ndropn } k \sigma))$

by *blast*

then have $\neg \text{enat } k \leq \text{nlength } \sigma \vee (\forall n. (\neg \text{enat } n \leq \text{nlength } \sigma \vee \neg k < n) \vee \neg w (\text{NNil } (\text{nnth } \sigma$

$n))) =$

$(\text{LIFT}(\text{wnext } (\Box (\text{init } (\neg w)))) (\text{ndropn } k \sigma))$

by (*metis* $\langle \text{enat } k \leq \text{nlength } \sigma \implies (\forall j. \text{enat } j \leq \text{nlength } \sigma \longrightarrow k < j \longrightarrow \neg w (\text{NNil } (\text{nnth } \sigma j)))$

$=$

$(\text{LIFT}(\text{wnext } (\Box (\text{init } (\neg w)))) (\text{ndropn } k \sigma)) \rangle$

then show ?thesis using f1 by presburger
 qed
 show ?thesis using 2 by blast
 qed
 qed
 have 4: $(\exists k \leq \text{nlength } \sigma. w (\text{NNil } (\text{nnth } \sigma k)) \wedge$
 $(\text{LIFT}(\text{wnext } (\Box (\text{init } (\neg w)))) (\text{ndropn } k \sigma) \wedge$
 $(\forall j < k. \neg w (\text{NNil } (\text{nnth } \sigma j)))) =$
 $(\sigma \models (\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge \text{wnext } (\Box (\text{init } (\neg w))))$
 by (simp add: NotStateUntilStateAndsem)
 from 1 2 3 4 show ?thesis by auto
 qed

lemma PiEmpty:

$\vdash (\text{init } w) \Pi \text{ empty} = (\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge \text{wnext } (\Box (\text{init } (\neg w))))$
 using PiEmptysem Valid-def by blast

lemma StatePiBoxStatesem:

$\sigma \models (\text{init } w) \Pi f = (\text{init } w) \Pi (f \wedge \Box (\text{init } w))$
 proof –
 have 1: $(\sigma \models (\text{init } w) \Pi f) =$
 $((\exists x \in \text{nset } \sigma. w (\text{NNil } x)) \wedge ((\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma) \models f))$
 by (metis Pistate)
 have 2: $(\sigma \models (\text{init } w) \Pi (f \wedge \Box (\text{init } w))) =$
 $((\exists x \in \text{nset } \sigma. w (\text{NNil } x)) \wedge ((\text{nfilter } (\lambda y. w (\text{NNil } y)) \sigma) \models f \wedge \Box (\text{init } w)))$
 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:

$\vdash (\text{init } w) \Pi f = (\text{init } w) \Pi (f \wedge \Box (\text{init } w))$
 using StatePiBoxStatesem Valid-def by blast

lemma StatePiUntil1:

$\vdash ((\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge (\text{init } w) \Pi f)) =$
 $(\text{wprev } (\Box (\text{init } (\neg w)))) \frown ((\text{init } w) \wedge (\text{init } w) \Pi f)$
 using StateUntilEqvWPrevChop by blast

lemma StatePiUntilsem2:

$(\sigma \models (\text{wprev } (\Box (\text{init } (\neg w)))) \frown ((\text{init } w) \wedge (\text{init } w) \Pi f)) =$
 $(\sigma \models ((\text{wprev } (\Box (\text{init } (\neg w)))) \frown ((\text{init } w) \wedge \text{empty})) \frown ((\text{init } w) \wedge (\text{init } w) \Pi f))$

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:

$\vdash (wprev (\Box (init (\neg w)))) \frown ((init w) \wedge (init w) \amalg f) =$
 $((wprev (\Box (init (\neg w)))) \frown ((init w) \wedge empty)) \frown ((init w) \wedge (init w) \amalg f)$

by (simp add: StatePiUntilsem2 Valid-def)

lemma StatePiUntil3:

$\vdash ((wprev (\Box (init (\neg w)))) \frown ((init w) \wedge empty)) =$
 $((init (\neg w)) \mathcal{U} ((init w) \wedge wnext (\Box (init (\neg w))))) \frown ((init w) \wedge empty)$

proof –

have 1: $\vdash (wprev (\Box (init (\neg w)))) \frown ((init w) \wedge empty) =$
 $(init (\neg w)) \mathcal{U} ((init w) \wedge empty)$

by (meson Prop11 StateUntilEqvWPrevChop)

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:

$(\sigma \models ((init (\neg w)) \mathcal{U} ((init w) \wedge wnext (\Box (init (\neg w))))) \frown ((init w) \wedge empty)) =$
 $(\sigma \models ((init w) \amalg empty) \frown ((init w) \wedge empty))$

by (metis PiEmpty inteq-reflection)

lemma StatePiUntil4:

$\vdash ((init (\neg w)) \mathcal{U} ((init w) \wedge wnext (\Box (init (\neg w))))) \frown ((init w) \wedge empty) =$
 $((init w) \amalg empty) \frown ((init w) \wedge empty)$

by (simp add: StatePiUntilsem4 Valid-def)

lemma StatePiUntilsem:

$\sigma \models (init w) \amalg f = (init (\neg w)) \mathcal{U} ((init w) \wedge (init w) \amalg f)$

proof –

have 2: $(\sigma \models (init (\neg w)) \mathcal{U} ((init w) \wedge (init w) \amalg f)) =$

$(\sigma \models (wprev (\Box (init (\neg w)))) \frown ((init w) \wedge (init w) \amalg f))$

using StateUntilEqvWPrevChopsem[of LIFT($\neg w$) LIFT($((init w) \wedge (init w) \amalg f)$) σ]

by simp

have 7: $(\sigma \models (wprev (\Box (init (\neg w)))) \frown ((init w) \wedge (init w) \amalg f)) =$

$(\sigma \models (((init w) \amalg empty) \frown ((init w) \wedge empty)) \frown ((init w) \wedge (init w) \amalg f))$

by (metis PiEmpty StatePiUntil2 StatePiUntil3 inteq-reflection)

have 8: $(\sigma \models (((init w) \amalg empty) \frown ((init w) \wedge empty)) \frown ((init w) \wedge (init w) \amalg f)) =$

$(\sigma \models (((init w) \amalg empty) \frown ((init w) \wedge (init w) \amalg f)))$

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: $(\sigma \models ((init\ w) \amalg empty)) \frown ((init\ w) \wedge (init\ w) \amalg f)) =$
 $(\sigma \models (init\ w) \amalg (empty \frown f))$
using *PiSchopDistsema PiSchopDistsemb* **by** *blast*
have 10: $(\sigma \models (init\ w) \amalg (empty \frown f)) = (\sigma \models (init\ w) \amalg f)$
by $(metis\ EmptySCHop\ inteq-reflection)$
show *?thesis*
by $(simp\ add:\ 10\ 2\ 7\ 8\ 9)$
qed

lemma *StatePiUntil*:

$\vdash (init\ w) \amalg f = (init\ (\neg w)) \mathcal{U} ((init\ w) \wedge (init\ w) \amalg f)$
using *StatePiUntilsem* **by** *blast*

lemma *StateAndPiEmpty*:

$\vdash ((init\ w) \wedge (init\ w) \amalg empty) = (w \wedge empty) \frown (wnext\ (\Box (init\ (\neg w))))$
proof –
have 1: $\vdash ((init\ w) \wedge (init\ w) \amalg empty) =$
 $((init\ w) \wedge (init\ (\neg w)) \mathcal{U} ((init\ w) \wedge (wnext\ (\Box (init\ (\neg w))))))$
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*:

$(\sigma \models (\exists k.\ (init\ w) \amalg (fpower\ f\ k))) =$
 $(\sigma \models (init\ w) \amalg empty \vee (\exists k.\ (init\ w) \amalg (f \frown (fpower\ f\ (k)))))$
proof –
have 1: $(\sigma \models (\exists k.\ (init\ w) \amalg (fpower\ f\ k))) =$
 $(\exists k.\ (\sigma \models (init\ w) \amalg (fpower\ f\ k)))$
by *simp*
have 2: $(\exists k.\ (\sigma \models (init\ w) \amalg (fpower\ f\ k))) =$
 $((\sigma \models (init\ w) \amalg (fpower\ f\ 0)) \vee (\exists k.\ 1 \leq k \wedge (\sigma \models (init\ w) \amalg (fpower\ f\ (k)))))$
by $(metis\ One-nat-def\ diff-Suc-1\ le-SucE\ le-add1\ plus-1-eq-Suc)$
have 3: $(\sigma \models (init\ w) \amalg (fpower\ f\ 0)) = (\sigma \models (init\ w) \amalg empty)$
by $(simp\ add:\ itl-def\ fpower-d-def)$
have 4: $(\exists k.\ 1 \leq k \wedge (\sigma \models (init\ w) \amalg (fpower\ f\ (k)))) =$

$(\exists k. (\sigma \models (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k))))$
by (*metis le-add1 ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc*)
have 5: $(\exists k. (\sigma \models (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k)))) =$
 $(\sigma \models (\exists k. (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k))))$
by *simp*
have 6: $((\sigma \models (\text{init } w) \Pi \text{empty}) \vee (\sigma \models (\exists k. (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k)))) =$
 $(\sigma \models (\text{init } w) \Pi \text{empty} \vee (\exists k. (\text{init } w) \Pi (f \frown \text{fpower } f (k))))$
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*:

$\forall \sigma. \sigma \models (\exists k. (\text{init } w) \Pi (\text{fpower } f k)) =$
 $((\text{init } w) \Pi \text{empty} \vee (\exists k. (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k))))$
proof
fix σ
show $\sigma \models (\exists k. (\text{init } w) \Pi (\text{fpower } f k)) =$
 $((\text{init } w) \Pi \text{empty} \vee (\exists k. (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k))))$
proof –
have 1: $(\sigma \models (\exists k. (\text{init } w) \Pi (\text{fpower } f k)) =$
 $((\text{init } w) \Pi \text{empty} \vee (\exists k. (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k))))$
 $= (\sigma \models (\exists k. (\text{init } w) \Pi (\text{fpower } f k))) =$
 $(\sigma \models (\text{init } w) \Pi \text{empty} \vee (\exists k. (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k))))$
by *auto*
have 2: $(\sigma \models (\exists k. (\text{init } w) \Pi (\text{fpower } f k)) =$
 $(\sigma \models (\text{init } w) \Pi \text{empty} \vee (\exists k. (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k))))$
using *PiFPowerExpandsem[of w f σ]*
unfolding *fpower-d-def* **by** (*auto simp add: schop-d-def*)
show ?thesis **using** 1 2 **by** *blast*
qed
qed

lemma *PiFPowerExpand*:

$\vdash (\exists k. (\text{init } w) \Pi (\text{fpower } f k)) =$
 $((\text{init } w) \Pi \text{empty} \vee (\exists k. (\text{init } w) \Pi (\text{fpower } f (\text{Suc } k))))$
using *PiFPowerExpandsem1[of w f]* **by** (*auto simp add: Valid-def PiFPowerExpandsem1*)

lemma *exists-expand-sem*:

$(\sigma \models (\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{fin } w) k))) =$
 $((\sigma \models (\text{fpower } ((\text{init } w) \Pi f \wedge \text{fin } w) 0)) \vee$
 $(\sigma \models (\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{fin } w) (\text{Suc } k))))$
by (*metis (no-types, lifting) not0-implies-Suc unl-Rex*)

lemma *exists-expand*:

$\vdash (\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{fin } w) k)) =$
 $((\text{fpower } ((\text{init } w) \Pi f \wedge \text{fin } w) 0) \vee (\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{fin } w) (\text{Suc } k))))$
using *exists-expand-sem Valid-def* **by** *fastforce*

13.3.16 TruePiEqv

lemma *TruePiEqvsem*:

$\sigma \models \#True \Pi f = f$

by (*simp add: pi-d-def pifilt-true*) (*metis zero-enat-def zero-le*)

lemma *TruePiEqv*:

$\vdash (\#True) \Pi f = f$

using *TruePiEqvsem* **by** (*auto simp add: Valid-def*)

13.3.17 BoxImpEqvPi

lemma *BoxImpEqvPi*:

$\vdash \Box f \longrightarrow g = f \Pi g$

proof (*simp add: Valid-def itl-defs pi-d-def pifilt-def sfxfilt-def*)

show $\forall w. (\forall n. \text{enat } n \leq \text{nlength } w \longrightarrow f (\text{ndropn } n \ w)) \longrightarrow$

$g \ w = ((\exists i. \text{enat } i \leq \text{nlength } w \wedge f (\text{ndropn } i \ w)) \wedge$

$g (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f (\text{ndropns } w))))$

proof

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 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

13.3.18 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*)

13.3.19 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*)

13.3.20 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*)

13.4 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*:

$\vdash f \Pi\ inf \longrightarrow inf$

unfolding *Valid-def pi-d-def init-defs infinite-defs*

by *auto*

(*metis enat-ile nfinite-conv-nlength-enat pifilt-nlength-bound*)

lemma *DiamondWPrevBoxSChop*:

$\vdash \Diamond\ (init\ w) = (wprev\ (\Box\ (init\ (\neg\ w)))) \frown (init\ w)$

by (*metis Initprop(2) StateUntilEqvWPrevChop UntilRule inteq-reflection*)

lemma *PiChopDist1*:

$\vdash ((init\ w) \Pi\ (f;g) \vee ((init\ w) \Pi\ (f \wedge finite) \wedge inf)) =$
 $((init\ w) \Pi\ f);((init\ w) \wedge (init\ w) \Pi\ g)$

proof –

have 1: $\vdash f;g = (f \frown g \vee (f \wedge inf))$
by (*simp add: ChopSChopdef*)

have 2: $\vdash (init\ w) \Pi\ (f;g) = ((init\ w) \Pi\ (f \frown g) \vee (init\ w) \Pi\ (f \wedge inf))$
by (*metis 1 PiOr int-eq*)

have 3: $\vdash (init\ w) \Pi\ (f \frown g) = ((init\ w) \Pi\ f) \frown ((init\ w) \wedge (init\ w) \Pi\ g)$
by (*simp add: PiSChopDist*)

have 4: $\vdash ((init\ w) \Pi\ f);((init\ w) \wedge (init\ w) \Pi\ g) =$
 $((init\ w) \Pi\ f) \frown ((init\ w) \wedge (init\ w) \Pi\ g) \vee ((init\ w) \Pi\ f \wedge inf)$

by (*simp add: ChopSChopdef*)

have 41: $\vdash f = (f \wedge (finite \vee inf))$
unfolding *finite-d-def* by fastforce

have 5: $\vdash (init\ w) \Pi\ f = (init\ w) \Pi\ (f \wedge (finite \vee inf))$
by (*simp add: 41 PiEqvRule*)
have 6: $\vdash ((init\ w) \Pi\ f \wedge inf) =$
 $((init\ w) \Pi\ (f \wedge finite) \wedge inf) \vee ((init\ w) \Pi\ (f \wedge inf) \wedge inf)$
by (*metis AndInfEqvChopFalse OrChopEqvRule OrFiniteInf PiOr inteq-reflection*)
have 7: $\vdash ((init\ w) \Pi\ (f \wedge inf) \wedge inf) = ((init\ w) \Pi\ (f \wedge inf))$
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*:

$\vdash ((init\ w) \Pi\ (f;g)) =$
 $((init\ w) \Pi\ (f \wedge finite) \wedge (init\ w) \wedge (init\ w) \Pi\ g) \vee ((init\ w) \Pi\ (f \wedge inf) \wedge inf)$

proof –

have 1: $\vdash (init\ w) \Pi\ (f;g) = ((init\ w) \Pi\ (f \frown g) \vee (init\ w) \Pi\ (f \wedge inf))$
by (*metis ChopSChopdef PiOr inteq-reflection*)
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: $\vdash ((init\ w) \Pi\ f) \frown ((init\ w) \wedge (init\ w) \Pi\ g) =$
 $((init\ w) \Pi\ f \wedge finite);((init\ w) \wedge (init\ w) \Pi\ g)$
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*:

$\vdash (g \amalg (f \wedge \text{finite}) \wedge \text{finite}) = (g \amalg f \wedge \text{finite})$

proof –

have 1: $\vdash (g \amalg (f \wedge \text{finite}) \wedge \text{finite}) \longrightarrow (g \amalg f \wedge \text{finite})$

by (*metis FiniteImp PiAnd PiAndPiImpPiAnd Prop01 Prop05 Prop12 inteq-reflection lift-and-com*)

have 31: $\vdash f = ((f \wedge \text{finite}) \vee (f \wedge \text{inf}))$

unfolding *finite-d-def* **by** *fastforce*

have 32: $\vdash g \amalg f = g \amalg ((f \wedge \text{finite}) \vee (f \wedge \text{inf}))$

using 31 *PiEqvRule* **by** *blast*

have 33: $\vdash g \amalg ((f \wedge \text{finite}) \vee (f \wedge \text{inf})) = (g \amalg (f \wedge \text{finite}) \vee g \amalg (f \wedge \text{inf}))$

by (*simp add: PiOr*)

have 34: $\vdash \neg(g \amalg (f \wedge \text{inf}) \wedge \text{finite})$

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 sxfilt-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: $\vdash (g \amalg f \wedge \text{empty}) \longrightarrow (g \amalg (f \wedge \text{empty}) \wedge \text{empty})$
using *EmptyImpPiEmpty PiAnd* **by** *fastforce*
show ?thesis **using** 1 2 **by** *fastforce*
qed

lemma *SlideInitSFinVar1*:
 $\vdash (\text{init } w) \amalg (((F \wedge \text{more}) \wedge \text{finite}); X) =$
 $((\text{init } w) \amalg (F \wedge \text{more}) \wedge \text{fin } w) \wedge \text{finite}; ((\text{init } w) \amalg X)$
proof –
have 1: $\vdash (\text{init } w) \amalg (((F \wedge \text{more}) \wedge \text{finite}); X) =$
 $((\text{init } w) \amalg (F \wedge \text{more}) \wedge \text{finite}); ((\text{init } w) \wedge (\text{init } w) \amalg X)$
by (*metis PiSChopDist schop-d-def*)
have 2: $\vdash ((\text{init } w) \amalg (F \wedge \text{more}) \wedge \text{finite}); ((\text{init } w) \wedge (\text{init } w) \amalg X) =$
 $((\text{init } w) \amalg (F \wedge \text{more}) \wedge \text{fin } w) \wedge \text{finite}; ((\text{init } w) \amalg X)$
using *SlideInitSFin[of LIFT (init w) $\amalg (F \wedge \text{more})$ w LIFT (init w) $\amalg X$]* **unfolding** *schoop-d-def*
by *blast*
show ?thesis **using** 1 2 **by** (*metis int-eq*)
qed

lemma *UPiAbsorp*:
 $\vdash (((\text{init } w) \amalg (F \wedge \text{more}) \wedge \text{fin } w) \wedge \text{finite}); ((\text{init } w) \amalg^u X) =$
 $((\text{init } w) \amalg (F \wedge \text{more}) \wedge \text{fin } w) \wedge \text{finite}; ((\text{init } w) \amalg X)$
proof –
have 0: $\vdash ((\text{init } w) \amalg^u X) = (\Box (\text{init } (\neg w)) \vee (\text{init } w) \amalg X)$
by (*metis Initprop(2) UPiEqvBoxOrPi inteq-reflection*)
have 1: $\vdash (((\text{init } w) \amalg (F \wedge \text{more}) \wedge \text{fin } w) \wedge \text{finite}); ((\text{init } w) \amalg^u X) =$
 $((\text{init } w) \amalg (F \wedge \text{more}) \wedge \text{fin } w) \wedge \text{finite}; (\Box (\text{init } (\neg w)) \vee (\text{init } w) \amalg X)$

```

    using 0 RightChopEqvChop by blast
have 2: ⊢ (((init w) Π (F ∧ more) ∧ sfin w) ∧ finite);(□ (init (¬w)) ∨ (init w) Π X) =
    (((init w) Π (F ∧ more)) ∧ finite);((init w) ∧ (□ (init (¬w)) ∨ (init w) Π X))
    using SlideInitSFin[of LIFT (init w) Π (F ∧ more) w LIFT (□ (init (¬w)) ∨ (init w) Π X) ]
    unfolding schop-d-def
    by (metis inteq-reflection)
have 3: ⊢ ((init w) ∧ (□ (init (¬w)) ∨ (init w) Π X)) =
    ((init w) ∧ ( (init w) Π X))
    using BoxElim Initprop(2) by fastforce
have 4: ⊢ (((init w) Π (F ∧ more)) ∧ finite);((init w) ∧ (□ (init (¬w)) ∨ (init w) Π X)) =
    (((init w) Π (F ∧ more)) ∧ finite);((init w) ∧ (init w) Π X)
    using 3 RightChopEqvChop by blast
have 5: ⊢ (((init w) Π (F ∧ more)) ∧ finite);((init w) ∧ (init w) Π X) =
    (((init w) Π (F ∧ more) ∧ sfin w) ∧ finite);((init w) Π X)
    using SlideInitSFin[of LIFT ((init w) Π (F ∧ more)) w LIFT (init w) Π X]
    unfolding schop-d-def
    by auto
show ?thesis
by (metis 1 2 4 5 inteq-reflection)
qed

```

lemma *PiStateFinite*:

⊢ (init w) Π finite = ◇ ((init w) ∧ (wnext (□ (init (¬w)))))

proof –

```

have 1: ⊢ (init w) Π finite = (init w) Π (#True ∧ empty)
    by (metis DiamondEmptyEqvFinite DiamondSChopdef PiEqvRule int-eq)
have 2: ⊢ (init w) Π (#True ∧ empty) = ((init w) Π #True) ∧ ((init w) ∧ (init w) Π empty)
    by (simp add: PiSChopDist)
have 3: ⊢ ((init w) ∧ (init w) Π empty) = ((init w) ∧ wnext (□ (init (¬w))))
    by (metis InitAndEmptyEqvAndEmpty StateAndEmptySChop StateAndPiEmpty inteq-reflection)
have 4: ⊢ (init w) Π #True = ◇ (init w)
    by (simp add: PiTrueEqvDiamond)
have 5: ⊢ ((init w) Π #True) ∧ ((init w) ∧ (init w) Π empty) =
    (◇ (init w)) ∧ ((init w) ∧ wnext (□ (init (¬w))))
    by (simp add: 3 4 SChopEqvSChop)
have 6: ⊢ (◇ (init w)) ∧ ((init w) ∧ wnext (□ (init (¬w)))) =
    (◇ (init w) ∧ sfin w) ∧ (wnext (□ (init (¬w))))
    by (simp add: SlideInitSFin)
have 7: ⊢ (◇ (init w) ∧ sfin w) = sfin w
    by (metis DiamondSChopdef InitAndEmptyEqvAndEmpty Prop10 SChopAndA SFinEqvTrueSChopAndEmpty
    inteq-reflection lift-and-com)
have 8: ⊢ (◇ (init w) ∧ sfin w) ∧ (wnext (□ (init (¬w)))) =
    (sfin w) ∧ (wnext (□ (init (¬w))))
    using 7 LeftSChopEqvSChop by blast
have 9: ⊢ (sfin w) ∧ (wnext (□ (init (¬w)))) = #True ∧ ((init w) ∧ (wnext (□ (init (¬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*:

$\vdash (init\ w) \Pi ((init\ w) \Pi^u f) = (init\ w) \Pi\ f$

proof –

have 1: $\vdash (init\ w) \Pi ((init\ w) \Pi^u f) = (\Diamond(init\ w) \wedge (init\ w) \Pi^u ((init\ w) \Pi^u f))$

by (*simp add: PiEqvDiamondUPi*)

have 2: $\vdash (init\ w) \Pi^u ((init\ w) \Pi^u f) = ((init\ w) \Pi^u f)$

by (*metis (no-types, lifting) NotUPiEqvNotPi UPiPiState inteq-reflection upi-d-def*)

have 3: $\vdash (\Diamond(init\ w) \wedge (init\ w) \Pi^u ((init\ w) \Pi^u f)) =$
 $(\Diamond(init\ w) \wedge ((init\ w) \Pi^u f))$

using 2 **by** *auto*

have 4: $\vdash (\Diamond(init\ w) \wedge ((init\ w) \Pi^u f)) = (init\ w) \Pi\ f$

by (*meson PiEqvDiamondUPi Prop11*)

show *?thesis*

by (*metis 1 3 4 int-eq*)

qed

lemma *PiStateFiniteAndFinite*:

$\vdash ((init\ w) \Pi\ finite \wedge finite) = (\Diamond(init\ w) \wedge finite)$

proof –

have 1: $\vdash ((init\ w) \Pi\ finite \wedge finite) \longrightarrow (\Diamond(init\ w) \wedge finite)$

using *PiImpDiamond* **by** *fastforce*

have 2: $\vdash (\Diamond(init\ w) \wedge finite) \longrightarrow ((init\ w) \Pi\ finite \wedge finite)$

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*:

$\vdash (init\ w) \Pi (init\ w) = (init\ w) \Pi\ \#True$

by (*metis BoxElim PiImpDiamond PiMPBC PiTrueEqvDiamond Prop11 StatePiBoxState int-simps(17) inteq-reflection*)

lemma *StateAndPiState*:

$\vdash ((init\ w) \wedge (init\ w) \Pi (empty \wedge w)) = ((init\ w) \wedge wnext\ (\Box (init\ (\neg w))))$

proof –

have 2: $\vdash ((init\ w) \wedge init\ w \Pi empty) = (init\ w \wedge wnext\ (\Box (init\ (\neg w))))$

by (*metis InitAndEmptyEqvAndEmpty StateAndEmptySChop StateAndPiEmpty inteq-reflection*)

have 5: $\vdash (init\ w) \Pi \#True = \Diamond (init\ w)$

by (*simp add: PiTrueEqvDiamond*)

show *?thesis*

by (*metis 2 5 InitAndEmptyEqvAndEmpty PiAnd PiImpDiamond PiStateState Prop10 inteq-reflection lift-and-com*)

qed

lemma *PiStateFiniteSfin*:

$\vdash ((init\ w) \Pi finite \wedge sfin\ w) = sfin\ w$

proof –

have 1: $\vdash ((init\ w) \Pi finite \wedge sfin\ w) =$

$((((init\ w) \Pi finite \wedge finite) \wedge sfin\ w))$

by (*metis AndSFinEqvSChopAndEmpty SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty*

inteq-reflection lift-and-com)

have 2: $\vdash (((init\ w) \Pi finite \wedge finite) \wedge sfin\ w) = ((\Diamond (init\ w) \wedge finite) \wedge sfin\ w)$

using *PiStateFiniteAndFinite* **by** *fastforce*

have 3: $\vdash ((\Diamond (init\ w) \wedge finite) \wedge sfin\ w) = ((\Diamond (init\ w)) \wedge sfin\ w)$

by (*metis AndSFinEqvSChopAndEmpty SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty*

inteq-reflection lift-and-com)

have 4: $\vdash ((\Diamond (init\ w)) \wedge sfin\ w) = sfin\ w$

by (*metis 3 InitAndEmptyEqvAndEmpty Prop11 Prop12 SChopAndA SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection*)

show *?thesis*

by (*metis 1 2 3 4 inteq-reflection*)

qed

lemma *PiStateSFin*:

$\vdash (init\ w) \Pi (sfin\ w) = (sfin\ w);(wnext\ (\Box (init\ (\neg w))))$

proof –

have 1: $\vdash (sfin\ w) = finite;(empty \wedge w)$

by (*metis ChopAndCommute SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection sometimes-d-def*)

have 2: $\vdash (init\ w) \Pi (finite;(empty \wedge w)) =$

$((init\ w) \Pi finite \wedge finite);((init\ w) \wedge (init\ w) \Pi (empty \wedge w))$

using *PiSChopDist*[of *w LIFT finite LIFT (empty \wedge w)*] **unfolding** *schop-d-def*

by *auto*

have 3: $\vdash ((init\ w) \Pi finite \wedge finite);((init\ w) \wedge (init\ w) \Pi (empty \wedge w)) =$

$((init\ w) \Pi finite \wedge finite);((init\ w) \wedge wnext\ (\Box (init\ (\neg w))))$

using *RightChopEqvChop StateAndPiState* **by** *blast*

```

have 4:  $\vdash (sfin\ w \wedge finite) = sfin\ w$ 
  by (metis 1 ChopAndA DiamondEmptyEqvFinite Prop10 inteq-reflection sometimes-d-def)
have 5:  $\vdash ((init\ w) \Pi\ finite \wedge finite); ((init\ w) \wedge wnext\ (\Box\ (init\ (\neg w)))) =$ 
   $((init\ w) \Pi\ finite \wedge sfin\ w); (wnext\ (\Box\ (init\ (\neg w))))$ 
  using SlideInitSFin[of LIFT (init w)  $\Pi$  finite w LIFT wnext ( $\Box$  (init ( $\neg w$ )))] 4
  unfolding schop-d-def
  by (metis Prop10 Prop12 int-eq int-iffD2 lift-and-com)
have 6:  $\vdash ((init\ w) \Pi\ finite \wedge sfin\ w); (wnext\ (\Box\ (init\ (\neg w)))) = (sfin\ w); (wnext\ (\Box\ (init\ (\neg w))))$ 
  using LeftChopEqvChop PiStateFiniteSfin by blast
show ?thesis
by (metis 1 2 3 5 6 int-eq)
qed

```

13.5 SChopstar and Pi

lemma *PiChopstarhelp2a*:

```

 $\vdash (w \wedge empty) \frown (fpower\ (f \frown (w \wedge empty)) \wedge more)\ k =$ 
   $(fpower\ (((w \wedge empty) \frown f) \wedge more)\ k) \frown (w \wedge empty)$ 
proof (induction k)
case 0
then show ?case
proof –
  have 1:  $\vdash (w \wedge empty) \frown empty = empty \frown (w \wedge empty)$ 
    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:  $\vdash (w \wedge empty) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ (Suc\ k) =$ 
     $(w \wedge empty) \frown ((f \frown (w \wedge empty) \wedge more) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k)$ 
    unfolding fpower-d-def
    by (simp add: chop-d-def)
  have 11:  $\vdash (f \wedge more) \frown (w \wedge empty) = (sfin\ w \wedge (f \wedge more))$ 
    by (metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty int-eq)
  have 12:  $\vdash (f \frown (w \wedge empty)) = (sfin\ w \wedge f)$ 
    by (metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection)
  have 13:  $\vdash (f \frown (w \wedge empty) \wedge more) = ((sfin\ w \wedge f) \wedge more)$ 
    using 12 by auto
  have 14:  $\vdash ((sfin\ w \wedge f) \wedge more) = (sfin\ w \wedge (f \wedge more))$ 
    by fastforce
  have 2:  $\vdash (f \frown (w \wedge empty) \wedge more) = (f \wedge more) \frown (w \wedge empty)$ 
    using 11 13 by fastforce
  have 20:  $\vdash ((f \frown (w \wedge empty) \wedge more) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k) =$ 
     $((f \wedge more) \frown (w \wedge empty)) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k$ 
    by (simp add: 2 LeftSChopEqvSChop)
  have 21:  $\vdash ((f \wedge more) \frown (w \wedge empty)) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k =$ 
     $(f \wedge more) \frown ((w \wedge empty) \frown fpower\ (f \frown (w \wedge empty) \wedge more)\ k)$ 

```

by (meson Prop11 SChopAssoc)
 have 22: $\vdash (f \wedge \text{more}) \neg ((w \wedge \text{empty}) \neg \text{fpower } (f \neg (w \wedge \text{empty}) \wedge \text{more}) k) =$
 $((f \wedge \text{more}) \neg (\text{fpower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k \neg (w \wedge \text{empty})))$
 by (simp add: RightSChopEqvSChop Suc.IH)
 have 23: $\vdash (w \wedge \text{empty}) \neg ((f \neg (w \wedge \text{empty}) \wedge \text{more}) \neg \text{fpower } (f \neg (w \wedge \text{empty}) \wedge \text{more}) k) =$
 $(w \wedge \text{empty}) \neg (((f \wedge \text{more}) \neg (w \wedge \text{empty})) \neg \text{fpower } (f \neg (w \wedge \text{empty}) \wedge \text{more}) k)$
 using 20 RightSChopEqvSChop by blast
 have 24: $\vdash (w \wedge \text{empty}) \neg (((f \wedge \text{more}) \neg (w \wedge \text{empty})) \neg \text{fpower } (f \neg (w \wedge \text{empty}) \wedge \text{more}) k) =$
 $(w \wedge \text{empty}) \neg ((f \wedge \text{more}) \neg ((w \wedge \text{empty}) \neg \text{fpower } (f \neg (w \wedge \text{empty}) \wedge \text{more}) k))$
 using 21 RightSChopEqvSChop by blast
 have 25: $\vdash (w \wedge \text{empty}) \neg ((f \wedge \text{more}) \neg ((w \wedge \text{empty}) \neg \text{fpower } (f \neg (w \wedge \text{empty}) \wedge \text{more}) k)) =$
 $(w \wedge \text{empty}) \neg ((f \wedge \text{more}) \neg (\text{fpower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k \neg (w \wedge \text{empty})))$
 using 23 22 RightSChopEqvSChop by blast
 have 3: $\vdash (w \wedge \text{empty}) \neg ((f \neg (w \wedge \text{empty}) \wedge \text{more}) \neg \text{fpower } (f \neg (w \wedge \text{empty}) \wedge \text{more}) k) =$
 $(w \wedge \text{empty}) \neg ((f \wedge \text{more}) \neg (\text{fpower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k \neg (w \wedge \text{empty})))$
 using 23 24 25 by fastforce
 have 4: $\vdash (w \wedge \text{empty}) \neg ((f \wedge \text{more}) \neg (\text{fpower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k \neg (w \wedge \text{empty}))) =$
 $((w \wedge \text{empty}) \neg (f \wedge \text{more})) \neg ((\text{fpower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k \neg (w \wedge \text{empty})))$
 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*:

$\vdash (init\ w) \Pi (fpower\ f\ (Suc\ k)) = ((init\ w) \Pi f) \frown ((init\ w) \wedge (init\ w) \Pi (fpower\ f\ k))$

unfolding *fpower-d-def* **by** (*metis PiSChopDist schop-d-def wpow-Suc*)

lemma *PiFPowerSuch*:

$\vdash (init\ w) \Pi (fpower\ f\ (Suc\ k)) = ((init\ w) \Pi f \wedge sfin\ w) \frown ((init\ w) \wedge (init\ w) \Pi (fpower\ f\ k))$

proof –

have 0: $\vdash (init\ w) \Pi (fpower\ f\ (Suc\ k)) = ((init\ w) \Pi f) \frown ((init\ w) \wedge (init\ w) \Pi (fpower\ f\ k))$

by (*meson PiFPowerSuca*)

have 1: $\vdash ((init\ w) \wedge (init\ w) \Pi (fpower\ f\ k)) = (w \wedge empty) \frown ((init\ w) \wedge (init\ w) \Pi (fpower\ f\ k))$

by (*metis InitAndEmptyEqvAndEmpty Prop10 Prop12 StateAndEmptySChop int-iffD2 inteq-reflection lift-and-com*)

have 2: $\vdash ((init\ w) \Pi f) \frown ((init\ w) \wedge (init\ w) \Pi (fpower\ f\ k)) =$
 $((init\ w) \Pi f) \frown ((w \wedge empty) \frown ((init\ w) \wedge (init\ w) \Pi (fpower\ f\ k)))$

using 1 *RightSChopEqvSChop* **by** *blast*

have 3: $\vdash ((init\ w) \Pi f) \frown ((w \wedge empty) \frown ((init\ w) \wedge (init\ w) \Pi (fpower\ f\ k))) =$
 $((init\ w) \Pi f) \frown ((w \wedge empty) \frown ((init\ w) \wedge (init\ w) \Pi (fpower\ f\ k)))$

by (*simp add: SChopAssoc*)

have 4: $\vdash (((init\ w) \Pi f) \frown (w \wedge empty)) = ((init\ w) \Pi f \wedge sfin\ w)$

by (*metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection lift-and-com*)

have 5: $\vdash (((init\ w) \Pi f) \frown (w \wedge empty)) \frown ((init\ w) \wedge (init\ w) \Pi (fpower\ f\ k)) =$
 $((init\ w) \Pi f \wedge sfin\ w) \frown ((init\ w) \wedge (init\ w) \Pi (fpower\ f\ k))$

by (*simp add: 4 LeftSChopEqvSChop*)

show *?thesis*

using 0 2 3 5 **by** *fastforce*

qed

lemma *PiFPowerSucc*:

$\vdash (init\ w) \Pi (fpower\ f\ (Suc\ k)) =$
 $(fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w) \wedge (init\ w) \Pi empty)$

proof (*induction k*)

case 0

then show *?case*

using *PiFPowerSuch*[*of w f 0*]

unfolding *schop-d-def fpower-d-def wpow-0 wpow-Suc*

proof –

assume *a1*: $\vdash init\ w \Pi ((f \wedge finite); empty) =$
 $((init\ w \Pi f \wedge sfin\ w) \wedge finite); (init\ w \wedge init\ w \Pi empty)$

have $\vdash (((init\ w \Pi f \wedge sfin\ w) \wedge finite); empty \wedge finite) = ((init\ w \Pi f \wedge sfin\ w) \wedge finite)$

by (*metis AndChopB ChopEmpty Prop10 inteq-reflection*)

then show $\vdash init\ w \Pi ((f \wedge finite); empty) =$

$((init\ w \Pi f \wedge sfin\ w) \wedge finite); (init\ w \wedge init\ w \Pi empty)$

by (*metis a1 int-eq*)

qed

next

case (*Suc k*)

then show *?case*

proof –

have 3: $\vdash (init\ w \ \Pi\ f) \frown$

$(init\ w \wedge fpower\ (init\ w \ \Pi\ f \wedge sfin\ w)\ (Suc\ k) \frown (init\ w \wedge init\ w \ \Pi\ empty)) =$
 $((init\ w \ \Pi\ f \wedge sfin\ w) \frown fpower\ (init\ w \ \Pi\ f \wedge sfin\ w)\ (Suc\ k)) \frown$
 $(init\ w \wedge init\ w \ \Pi\ empty)$

by (*metis* (*no-types*, *lifting*) *AndSFinSChopEqvStateAndSChop InitAndEmptyEqvAndEmpty SChopAssoc*
SFinEqvTrueSChopAndEmpty inteq-reflection)

have 4: $\vdash (init\ w \ \Pi\ f) \frown (init\ w \wedge init\ w \ \Pi\ (f \frown fpower\ f\ k)) =$

$((init\ w \ \Pi\ f \wedge sfin\ w) \frown$
 $((init\ w \ \Pi\ f \wedge sfin\ w) \frown fpower\ (init\ w \ \Pi\ f \wedge sfin\ w)\ k)) \frown$
 $(init\ w \wedge init\ w \ \Pi\ empty)$

using 3 *Suc.IH* **unfolding** *fpower-d-def wpow-Suc schop-d-def* **by** (*metis int-eq*)

show ?thesis

by (*metis* 4 *PiFPowerSuca fpower-d-def inteq-reflection schop-d-def wpow-Suc*)

qed

qed

lemma *PiFPowerSucc*:

$\vdash (init\ w) \ \Pi\ (fpower\ f\ (Suc\ k)) = (fpower\ ((init\ w) \ \Pi\ f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w) \ \Pi\ empty)$

proof (*induction* *k*)

case 0

then show ?case **unfolding** *fpower-d-def wpow-0 wpow-Suc*

by (*metis* (*no-types*, *opaque-lifting*) *InitAndEmptyEqvAndEmpty PiSChopDist SChopAndEmptyEqvSChopAn-*
dEmpty

SChopAssoc SFinEqvTrueSChopAndEmpty StateAndEmptySChop inteq-reflection lift-and-com schop-d-def)

next

case (*Suc* *k*)

then show ?case

proof –

have 1: $\vdash init\ w \ \Pi\ fpower\ f\ (Suc\ (Suc\ k)) = (init\ w) \ \Pi\ (f \frown (fpower\ f\ (Suc\ k)))$

by (*metis* *PiFPowerSuca PiSChopDist inteq-reflection*)

have 2: $\vdash (init\ w) \ \Pi\ (f \frown (fpower\ f\ (Suc\ k))) = ((init\ w) \ \Pi\ f) \frown ((init\ w) \wedge (init\ w) \ \Pi\ (fpower\ f\ (Suc\ k)))$

by (*simp* *add: PiSChopDist*)

have 3: $\vdash ((init\ w) \ \Pi\ f) \frown ((init\ w) \wedge (init\ w) \ \Pi\ (fpower\ f\ (Suc\ k))) =$

$((init\ w) \ \Pi\ f) \frown ((init\ w) \wedge (fpower\ ((init\ w) \ \Pi\ f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w) \wedge (init\ w) \ \Pi\$
 $empty))$

by (*metis* 2 *PiFPowerSucc inteq-reflection*)

have 4: $\vdash ((init\ w) \ \Pi\ f) \frown ((init\ w) \wedge (fpower\ ((init\ w) \ \Pi\ f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w) \wedge (init\ w) \ \Pi\$
 $empty)) =$

$((init\ w) \ \Pi\ f \wedge sfin\ w) \frown (fpower\ ((init\ w) \ \Pi\ f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w) \wedge (init\ w) \ \Pi\$
 $empty))$

by (*metis* *AndSFinSChopEqvStateAndSChop InitAndEmptyEqvAndEmpty SFinEqvTrueSChopAn-*
dEmpty inteq-reflection)

have 5: $\vdash ((init\ w) \ \Pi\ f \wedge sfin\ w) \frown (fpower\ ((init\ w) \ \Pi\ f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w) \wedge (init\ w) \ \Pi\$
 $empty)) =$

$((fpower\ ((init\ w) \ \Pi\ f \wedge sfin\ w)\ (Suc\ (Suc\ k))) \frown ((init\ w) \wedge (init\ w) \ \Pi\ empty))$

by (*metis* 1 2 4 *PiFPowerSucc inteq-reflection*)

have 6: $\vdash ((fpower\ ((init\ w) \ \Pi\ f \wedge sfin\ w)\ (Suc\ (Suc\ k))) \frown ((init\ w) \wedge (init\ w) \ \Pi\ empty)) =$

$((fpower\ ((init\ w) \ \Pi\ f \wedge sfin\ w)\ (Suc\ (Suc\ k))) \frown (init\ w \ \Pi\ empty))$

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:

$\vdash (f \wedge \text{sf}in\ w) \frown g = (f \wedge \text{f}in\ w) \frown g$
proof –
 have 1: $\vdash (f \wedge \text{f}in\ w) \frown g = (f \wedge (\text{f}in\ w \wedge \text{f}inite)) \frown g$
 by (metis (no-types, lifting) LeftChopEqvChop Prop11 Prop12 lift-and-com schop-d-def)
 have 2: $\vdash (\text{f}in\ w \wedge \text{f}inite) = (\text{sf}in\ w \wedge \text{f}inite)$
 by (metis (no-types, lifting) EmptyImpFinite FinAndEmpty FinEqvTrueChopAndEmpty Finprop(5)
 Prop10 SChopAndB SChopImpFinite SFinAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection
 lift-imp-trans sf-in-d-def)
 have 3: $\vdash (f \wedge \text{sf}in\ w) \frown g = (f \wedge (\text{sf}in\ w \wedge \text{f}inite)) \frown g$
 by (metis AndSChopCommute DiamondEmptyEqvFinite Prop10 SChopAndB SFinEqvTrueSChopAn-
 dEmpty
 TrueSChopEqvDiamond int-eq)
 show ?thesis
 using 1 2 3 by (metis inteq-reflection)
 qed

lemma PiChopstar:

$\vdash (\text{init}\ w) \amalg (\text{schopstar}\ f) =$
 $(\text{init}\ (\neg w)) \mathcal{U} ((\text{init}\ w) \wedge (\text{schopstar}(((\text{init}\ w) \amalg f) \wedge \text{sf}in\ w)) \frown \text{wnext}(\Box (\text{init}\ (\neg w))))$
proof –
 have 1: $\vdash (\text{init}\ w) \amalg (\text{schopstar}\ f) = (\text{init}\ w) \amalg (\exists k. \text{f}power\ f\ k)$
 by (metis FPowerstardef PiEqvRule SChopstar-FPowerstar inteq-reflection)
 have 2: $\vdash (\text{init}\ w) \amalg (\exists k. \text{f}power\ f\ k) = (\exists k. (\text{init}\ w) \amalg (\text{f}power\ f\ k))$
 by (simp add: Valid-def pi-d-def)
 have 3: $\vdash (\exists k. (\text{init}\ w) \amalg (\text{f}power\ f\ k)) =$
 $((\text{init}\ w) \amalg \text{empty} \vee (\exists k. (\text{init}\ w) \amalg (\text{f}power\ f\ (\text{Suc}\ k))))$
 using PiFPowerExpand by auto
 have 4: $\vdash (\exists k. (\text{init}\ w) \amalg (\text{f}power\ f\ (\text{Suc}\ k))) =$
 $(\exists k. (\text{f}power\ ((\text{init}\ w) \amalg f \wedge \text{sf}in\ w)\ (\text{Suc}\ k)) \frown ((\text{init}\ w) \amalg \text{empty}))$
 by (meson ExEqvRule PiFPowerSuccd)
 have 5: $\vdash (\text{init}\ w) \amalg \text{empty} = \text{empty} \frown ((\text{init}\ w) \amalg \text{empty})$
 by (simp add: EmptySChop int-iffD1 int-iffD2 int-iffI)
 have 6: $\vdash (\exists k. (\text{f}power\ ((\text{init}\ w) \amalg f \wedge \text{sf}in\ w)\ (\text{Suc}\ k)) \frown ((\text{init}\ w) \amalg \text{empty})) =$
 $(\exists k. (\text{f}power\ ((\text{init}\ w) \amalg f \wedge \text{sf}in\ w)\ (\text{Suc}\ k)) \frown ((\text{init}\ w) \amalg \text{empty}))$
 by (simp add: ExistSChop)
 have 7: $\vdash ((\text{init}\ w) \amalg \text{empty} \vee (\exists k. (\text{f}power\ ((\text{init}\ w) \amalg f \wedge \text{sf}in\ w)\ (\text{Suc}\ k)) \frown ((\text{init}\ w) \amalg \text{empty}))) =$
 $(\text{empty} \frown ((\text{init}\ w) \amalg \text{empty}) \vee$
 $(\exists k. (\text{f}power\ ((\text{init}\ w) \amalg f \wedge \text{sf}in\ w)\ (\text{Suc}\ k)) \frown ((\text{init}\ w) \amalg \text{empty})))$
 using 5 6 by fastforce
 have 8: $\vdash (\text{empty} \frown ((\text{init}\ w) \amalg \text{empty}) \vee$
 $(\exists k. (\text{f}power\ ((\text{init}\ w) \amalg f \wedge \text{sf}in\ w)\ (\text{Suc}\ k)) \frown ((\text{init}\ w) \amalg \text{empty})) =$
 $(\text{empty} \vee (\exists k. (\text{f}power\ ((\text{init}\ w) \amalg f \wedge \text{sf}in\ w)\ (\text{Suc}\ k)))) \frown ((\text{init}\ w) \amalg \text{empty})$

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: $\vdash (\exists k. (\text{fpower } ((\text{init } w) \Pi f \wedge \text{sfin } w) k)) \neg ((\text{init } w) \Pi \text{empty}) =$
 $(\text{s chopstar } ((\text{init } w) \Pi f \wedge \text{sfin } w)) \neg ((\text{init } w) \Pi \text{empty})$
 by (metis FPowerstardef LeftSChopEqvSChop SChopstar-FPowerstar inteq-reflection)
 have 14: $\vdash (\text{s chopstar } ((\text{init } w) \Pi f \wedge \text{sfin } w)) \neg ((\text{init } w) \Pi \text{empty}) =$
 $((\text{init } w) \Pi \text{empty}) \vee$
 $((\text{init } w) \Pi f \wedge \text{sfin } w) \neg (\text{s chopstar } ((\text{init } w) \Pi f \wedge \text{sfin } w)) \neg ((\text{init } w) \Pi \text{empty})$
 by (metis 5 OrSChopEqvRule SChopstar-unfoldl-eq inteq-reflection)
 have 15: $\vdash ((\text{init } w) \Pi \text{empty}) = (\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge \text{wnext } (\Box (\text{init } (\neg w))))$
 by (simp add: PiEmpty)
 have 16: $\vdash (((\text{init } w) \Pi f \wedge \text{sfin } w) \neg (\text{s chopstar } ((\text{init } w) \Pi f \wedge \text{sfin } w))) \neg ((\text{init } w) \Pi \text{empty}) =$
 $(((\text{init } w) \Pi f \wedge \text{sfin } w) \neg (\text{s chopstar } ((\text{init } w) \Pi f \wedge \text{sfin } w))) \neg \text{wnext } (\Box (\text{init } (\neg w)))$
 proof -
 have 161: $\vdash (((\text{init } w) \Pi f \wedge \text{sfin } w) \neg (\text{s chopstar } ((\text{init } w) \Pi f \wedge \text{sfin } w))) \neg ((\text{init } w) \Pi \text{empty}) =$
 $(((\text{init } w) \Pi f \wedge \text{sfin } w) \neg ((\text{s chopstar } ((\text{init } w) \Pi f \wedge \text{sfin } w)) \neg ((\text{init } w) \Pi \text{empty})))$
 by (meson Prop11 SChopAssoc)
 have 162: $\vdash ((\text{init } w) \Pi f \wedge \text{sfin } w) = ((\text{init } w) \Pi f) \neg (w \wedge \text{empty})$
 by (metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection lift-and-com)
 have 163: $\vdash (\text{s chopstar } ((\text{init } w) \Pi f \wedge \text{sfin } w)) = (\text{s chopstar } (((\text{init } w) \Pi f) \neg (w \wedge \text{empty})))$
 by (metis 162 SChopstardef inteq-reflection)
 have 164: $\vdash ((\text{init } w) \Pi f \wedge \text{sfin } w) \neg (\text{s chopstar } ((\text{init } w) \Pi f \wedge \text{sfin } w)) =$
 $((\text{init } w) \Pi f) \neg ((w \wedge \text{empty}) \neg (\text{s chopstar } (((\text{init } w) \Pi f) \neg (w \wedge \text{empty}))))$
 using 162 SChopAssoc int-eq by metis
 have 165: $\vdash ((\text{init } w) \Pi f) \neg ((w \wedge \text{empty}) \neg (\text{s chopstar } (((\text{init } w) \Pi f) \neg (w \wedge \text{empty})))) =$
 $((\text{init } w) \Pi f) \neg ((\text{s chopstar } ((w \wedge \text{empty}) \neg ((\text{init } w) \Pi f))) \neg (w \wedge \text{empty}))$
 by (simp add: PiChopstarhelp2 RightSChopEqvSChop)
 have 166: $\vdash (((\text{init } w) \Pi f) \neg ((\text{s chopstar } ((w \wedge \text{empty}) \neg ((\text{init } w) \Pi f))) \neg (w \wedge \text{empty}))) \neg ((\text{init } w) \Pi \text{empty}) =$
 $((((\text{init } w) \Pi f) \neg (\text{s chopstar } ((w \wedge \text{empty}) \neg ((\text{init } w) \Pi f)))) \neg ((\text{init } w) \wedge ((\text{init } w) \Pi \text{empty})))$
 by (metis (no-types, lifting) InitAndEmptyEqvAndEmpty SChopAssoc StateAndEmptySChop int-eq)
 have 167: $\vdash ((((\text{init } w) \Pi f) \neg (\text{s chopstar } ((w \wedge \text{empty}) \neg ((\text{init } w) \Pi f)))) \neg ((\text{init } w) \wedge ((\text{init } w) \Pi \text{empty}))) =$
 $((((\text{init } w) \Pi f) \neg (\text{s chopstar } ((w \wedge \text{empty}) \neg ((\text{init } w) \Pi f)))) \neg ((w \wedge \text{empty}) \neg \text{wnext } (\Box (\text{init } (\neg w)))))$
 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: $\vdash wprev(\Box (init\ (\neg\ w))) \frown (((init\ w) \wedge empty) \frown ((init\ w) \Pi\ f)) =$
 $(init\ (\neg\ w)) \mathcal{U} ((init\ w) \wedge ((init\ w) \Pi\ f))$
using 18 19 StatePiUntil by fastforce
have 21: $\vdash (((init\ w) \Pi\ f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi\ f \wedge sfin\ w))) \frown wnext\ (\Box (init\ (\neg\ w))) =$
 $(init\ (\neg\ w)) \mathcal{U} ((init\ w) \wedge (((init\ w) \Pi\ f) \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi\ f \wedge sfin\ w))) \frown wnext\ (\Box (init\ (\neg\ w)))$
proof –
have $\vdash (init\ w \Pi\ f) \frown (init\ w \wedge empty) = (init\ w \Pi\ f \wedge sfin\ w)$
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* Π *f* \wedge *sfin w*))) \frown *wnext* (\Box (*init* (\neg *w*)))
by (*metis (no-types, lifting) AndSFinSChopEqvStateAndSChop SChopAssoc StateUntilEqvWPrevChop int-eq*)
qed
have 22: $\vdash ((init\ (\neg\ w)) \mathcal{U} ((init\ w) \wedge wnext\ (\Box (init\ (\neg\ w))))) \vee$
 $(init\ (\neg\ w)) \mathcal{U} ((init\ w) \wedge (((init\ w) \Pi\ f) \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi\ f \wedge sfin\ w))) \frown wnext\ (\Box (init\ (\neg\ w)))) =$
 $(init\ (\neg\ w)) \mathcal{U} ((init\ w) \wedge wnext\ (\Box (init\ (\neg\ w)))) \vee$
 $((init\ w) \wedge (((init\ w) \Pi\ f) \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi\ f \wedge sfin\ w))) \frown wnext\ (\Box (init\ (\neg\ w))))$
using UntilOrDist by fastforce
have 23: $\vdash ((init\ w) \wedge wnext\ (\Box (init\ (\neg\ w)))) \vee$
 $((init\ w) \wedge (((init\ w) \Pi\ f) \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi\ f \wedge sfin\ w))) \frown wnext\ (\Box (init\ (\neg\ w)))) =$

$((init\ w) \wedge (schopstar\ (((init\ w) \Pi f) \wedge sfin\ w)) \neg wnext(\Box (init\ (\neg w))))$
by (*metis* (*mono-tags*, *lifting*) *EmptyOrSCHopEqv Prop11 SCHopOrEqvRule SCHopstar-unfoldl-eq State-AndEmptySCHop int-eq*)
have 24: $\vdash (init\ w) \Pi (schopstar\ f) = ((init\ w) \Pi empty \vee (\exists k. (init\ w) \Pi (fpower\ f\ (Suc\ k))))$
using 1 2 3 **by** *fastforce*
have 25: $\vdash ((init\ w) \Pi empty \vee (\exists k. (init\ w) \Pi (fpower\ f\ (Suc\ k)))) =$
 $((init\ w) \Pi empty \vee (\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \neg ((init\ w) \Pi empty)))$
using 4 **by** *fastforce*
have 26: $\vdash ((init\ w) \Pi empty \vee$
 $(\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \neg ((init\ w) \Pi empty))) =$
 $(schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \neg ((init\ w) \Pi empty)$
using 12 13 **by** *fastforce*
have 27: $\vdash (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \neg ((init\ w) \Pi empty) =$
 $((init\ (\neg w)) \mathcal{U} ((init\ w) \wedge wnext(\Box (init\ (\neg w)))) \vee$
 $((init\ w) \Pi f \wedge sfin\ w) \neg (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \neg wnext(\Box (init\ (\neg w))))$
using 14 15 16 **by** *fastforce*
have 28: $\vdash (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \neg ((init\ w) \Pi empty) =$
 $(init\ (\neg w)) \mathcal{U} ((init\ w) \wedge (schopstar\ (((init\ w) \Pi f) \wedge sfin\ w)) \neg wnext(\Box (init\ (\neg w))))$
using 27 21 22 23
by (*metis* *inteq-reflection*)
show ?thesis
by (*metis* 24 25 26 28 *inteq-reflection*)
qed

lemma *PiChopstar-var1*:

$\vdash (init\ w) \Pi (schopstar\ f) =$
 $(schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \neg ((init\ w) \Pi empty)$
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: $\vdash (\exists k. (init\ w) \Pi (fpower\ f\ (Suc\ k))) =$
 $(\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \neg ((init\ w) \Pi empty))$
by (*meson* *ExEqvRule PiFPowerSuccd*)
have 5: $\vdash (init\ w) \Pi empty = empty \neg ((init\ w) \Pi empty)$
by (*simp* *add: EmptySCHop int-iffD1 int-iffD2 int-iffI*)
have 6: $\vdash (\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \neg ((init\ w) \Pi empty)) =$
 $(\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \neg ((init\ w) \Pi empty))$
by (*simp* *add: ExistSCHop*)
have 7: $\vdash ((init\ w) \Pi empty \vee (\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \neg ((init\ w) \Pi empty))) =$
 $(empty \neg ((init\ w) \Pi empty) \vee$
 $(\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \neg ((init\ w) \Pi empty)))$
using 5 6 **by** *fastforce*
have 8: $\vdash (empty \neg ((init\ w) \Pi empty) \vee$
 $(\exists k. (fpower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \neg ((init\ w) \Pi empty))) =$

(empty \vee ($\exists k.$ (fpower ((init w) Π f \wedge sfin w) (Suc k))) \frown ((init w) Π empty)
 by (meson OrSChopEqv Prop11)
 have 9: \vdash empty = (fpower ((init w) Π f \wedge sfin w) 0)
 unfolding fpower-d-def by simp
 have 10: \vdash (empty \vee ($\exists k.$ (fpower ((init w) Π f \wedge sfin w) (Suc k)))) =
 ((fpower ((init w) Π f \wedge sfin w) 0) \vee ($\exists k.$ (fpower ((init w) Π f \wedge sfin w) (Suc k))))
 unfolding fpower-d-def by simp
 have 11: \vdash ((fpower ((init w) Π f \wedge sfin w) 0) \vee ($\exists k.$ (fpower ((init w) Π f \wedge sfin w) (Suc k)))) =
 ($\exists k.$ (fpower ((init w) Π 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) Π empty \vee
 ($\exists k.$ (fpower ((init w) Π f \wedge sfin w) (Suc k))) \frown ((init w) Π empty)) =
 ($\exists k.$ (fpower ((init w) Π f \wedge sfin w) (k))) \frown ((init w) Π empty)
 by (metis 11 7 8 9 inteq-reflection)
 have 13: \vdash ($\exists k.$ (fpower ((init w) Π f \wedge sfin w) (k))) \frown ((init w) Π empty) =
 (schopstar ((init w) Π f \wedge sfin w)) \frown ((init w) Π 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:

\vdash (init w) Π (schopstar f) =
 ((init w) Π empty \vee
 (schopstar ((init w) Π f \wedge sfin w))) \frown (((init w) Π f \wedge sfin w) \frown (wnext (\Box (init (\neg w)))))

proof –

have 1: \vdash (schopstar ((init w) Π f \wedge sfin w)) =
 (empty \vee (schopstar ((init w) Π f \wedge sfin w))) \frown ((init w) Π f \wedge sfin w)
 by (metis (no-types, lifting) DiamondEmptyEqvFinite Prop10 Prop12 SCSEqvOrChopSCSB SChopAndB
 SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond int-eq int-iffD2 lift-and-com)
 have 2: \vdash (schopstar ((init w) Π f \wedge sfin w)) \frown ((init w) Π empty) =
 (((init w) Π empty) \vee ((schopstar ((init w) Π f \wedge sfin w)) \frown ((init w) Π f \wedge sfin w))) \frown ((init w) Π empty))
 using 1 EmptyOrSChopEqvRule by blast
 have 3: \vdash ((schopstar ((init w) Π f \wedge sfin w)) \frown ((init w) Π f \wedge sfin w)) \frown ((init w) Π empty) =
 (schopstar ((init w) Π f \wedge sfin w)) \frown (((init w) Π f \wedge sfin w) \frown ((init w) Π empty))
 by (meson Prop11 SChopAssoc)
 have 4: \vdash (((init w) Π f \wedge sfin w) \frown ((init w) Π empty)) =
 (((init w) Π f \wedge sfin w) \frown (wnext (\Box (init (\neg w)))))
 by (metis (no-types, lifting) InitAndEmptyEqvAndEmpty SlideInitSFin StateAndEmptySChop StateAnd-PiEmpty inteq-reflection)
 have 5: \vdash (schopstar ((init w) Π f \wedge sfin w)) \frown (((init w) Π f \wedge sfin w) \frown ((init w) Π empty)) =
 (schopstar ((init w) Π f \wedge sfin w)) \frown (((init w) Π f \wedge sfin w) \frown (wnext (\Box (init (\neg w)))))
 by (metis 4 RightChopEqvChop schop-d-def)
 show ?thesis
 by (metis 2 3 5 PiChopstar-var1 int-eq)
 qed

lemma *PiChopstar-var3*:

$\vdash (init\ w) \Pi (f \frown (schopstar\ f)) =$
 $(schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown (((init\ w) \Pi f \wedge sfin\ w) \frown (wnext\ (\Box (init\ (\neg w)))))$

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: $\vdash ((init\ w) \Pi f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi empty) =$
 $((init\ w) \Pi f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \frown ((init\ w) \Pi empty)$
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*:

$\vdash (init\ w) \Pi (f \neg (schopstar\ f)) =$
 $(schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \neg ((init\ w) \Pi (f \wedge finite))$

proof –

have 1: $\vdash (init\ w) \Pi (f \neg (schopstar\ f)) =$
 $((init\ w) \Pi f \wedge sfin\ w) \neg ((init\ w) \Pi (schopstar\ f))$
by (*metis PiSChopDist SlideInitSFin inteq-reflection*)

have 2: $\vdash ((init\ w) \Pi (schopstar\ f)) =$
 $(schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \neg ((init\ w) \Pi empty)$
using *PiChopstar-var1* **by** *auto*

have 3: $\vdash ((init\ w) \Pi f \wedge sfin\ w) \neg ((init\ w) \Pi (schopstar\ f)) =$
 $((init\ w) \Pi f \wedge sfin\ w) \neg (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \neg ((init\ w) \Pi empty)$
using 2 *RightSChopEqvSChop* **by** *blast*

have 4: $\vdash ((init\ w) \Pi f \wedge sfin\ w) \neg (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \neg ((init\ w) \Pi empty) =$
 $((init\ w) \Pi f \wedge sfin\ w) \neg (schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \neg ((init\ w) \Pi empty)$
using *SChopAssoc* **by** *blast*

have 5: $\vdash (((init\ w) \Pi f \wedge sfin\ w) \neg (schopstar\ ((init\ w) \Pi f \wedge sfin\ w))) =$
 $(schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \neg (((init\ w) \Pi f \wedge sfin\ w) \wedge finite)$
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: $\vdash (((init\ w) \Pi f \wedge sfin\ w)) \neg ((init\ w) \Pi empty) =$
 $(init\ w) \Pi (f \neg empty)$
by (*metis PiSChopDist SlideInitSFin inteq-reflection*)

have 8: $\vdash (init\ w) \Pi (f \neg empty) = (init\ w) \Pi (f \wedge finite)$
by (*simp add: ChopEmpty PiEqvRule chop-d-def*)

have 9: $\vdash (((init\ w) \Pi f \wedge sfin\ w) \neg (schopstar\ ((init\ w) \Pi f \wedge sfin\ w))) \neg ((init\ w) \Pi empty) =$
 $((schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \neg (((init\ w) \Pi f \wedge sfin\ w))) \neg ((init\ w) \Pi empty)$
by (*metis 4 5 6 inteq-reflection*)

have 10: $\vdash ((schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \neg (((init\ w) \Pi f \wedge sfin\ w))) \neg ((init\ w) \Pi empty) =$
 $(schopstar\ ((init\ w) \Pi f \wedge sfin\ w)) \neg (((init\ w) \Pi f \wedge sfin\ w) \neg ((init\ w) \Pi empty))$
by (*meson Prop11 SChopAssoc*)

show *?thesis*

by (*metis 1 10 3 4 7 8 9 int-eq*)

qed

lemma *PiChopstar-var6*:

$\vdash (init\ w) \Pi (schopstar\ f) =$
 $(((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \neg (schopstar\ ((init\ w) \Pi (f \vee empty) \wedge sfin\ w)))$
 $\neg (wnext(\Box(init\ (\neg w))))$

proof –

have 1: $\vdash (schopstar\ f) = (schopstar\ (f \vee empty))$
by (*metis SChopstar-star2 SChopstar-swap inteq-reflection*)

have 1: $\vdash (init\ w) \Pi (schopstar\ (f \vee empty)) =$
 $(schopstar\ ((init\ w) \Pi (f \vee empty) \wedge sfin\ w)) \neg ((init\ w) \Pi empty)$
by (*simp add: PiChopstar-var1*)

have 2: $\vdash (schopstar\ ((init\ w) \Pi (f \vee empty) \wedge sfin\ w)) =$

$(empty \vee ((init\ w) \Pi (f \vee empty) \wedge sfin\ w)) \neg (schopstar ((init\ w) \Pi (f \vee empty) \wedge sfin\ w)))$
by (*meson Prop06 SCSEqvOrChopSCSB SChopstar-slide-var*)
have 3: $\vdash (empty \vee ((init\ w) \Pi (f \vee empty) \wedge sfin\ w)) \neg (schopstar ((init\ w) \Pi (f \vee empty) \wedge sfin\ w))) \neg$
 $((init\ w) \Pi empty)$
 $= (((init\ w) \Pi empty) \vee (((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \neg (schopstar ((init\ w) \Pi (f \vee empty)$
 $\wedge sfin\ w)))) \neg ((init\ w) \Pi empty)$)

using *EmptyOrSChopEqv* **by** *blast*
have 4: $((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \neg (schopstar ((init\ w) \Pi (f \vee empty) \wedge sfin\ w)) =$
 $(schopstar ((init\ w) \Pi (f \vee empty) \wedge sfin\ w)) \neg (((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \wedge finite)$
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: $\vdash (((init\ w) \Pi (f \vee empty) \wedge sfin\ w)) \neg ((init\ w) \Pi empty) =$
 $((init\ w) \Pi (f \vee empty) \wedge sfin\ w)) \neg (wnext(\Box(init\ \neg w)))$
by (*metis (no-types, lifting) InitAndEmptyEqvAndEmpty SlideInitSFin StateAndEmptySChop State-*
AndPiEmpty int-eq)
have 7: $\vdash (((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \neg (schopstar ((init\ w) \Pi (f \vee empty) \wedge sfin\ w))) \neg ((init\ w)$
 $\Pi empty) =$
 $((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \neg (schopstar ((init\ w) \Pi (f \vee empty) \wedge sfin\ w))) \neg (wnext(\Box(init$
 $\neg w)))$
by (*metis (no-types, lifting) 4 5 6 SChopAssoc inteq-reflection*)
have 8: $\vdash (((init\ w) \Pi empty) \vee (((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \neg (schopstar ((init\ w) \Pi (f \vee empty)$
 $\wedge sfin\ w)))) \neg ((init\ w) \Pi empty) =$
 $((init\ w) \Pi (f \vee empty) \wedge sfin\ w) \neg (schopstar ((init\ w) \Pi (f \vee empty) \wedge sfin\ w))) \neg ((init\ w)$
 $\Pi empty)$)
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

13.6 Omega and Pi

lemma *OmegaImpDiamond*:

$\vdash (omega ((init\ w) \Pi (f \wedge more) \wedge sfin\ w)) \longrightarrow \Diamond (init\ w)$

proof –

have 1: $\vdash (omega ((init\ w) \Pi (f \wedge more) \wedge sfin\ w)) =$
 $(((((init\ w) \Pi (f \wedge more) \wedge sfin\ w) \wedge more) \wedge finite); (omega ((init\ w) \Pi (f \wedge more) \wedge sfin\ w)))$
using *OmegaUnroll[of LIFT ((init\ w) \Pi (f \wedge more) \wedge sfin\ w)]* **by** *blast*
have 2: $\vdash (((init\ w) \Pi (f \wedge more) \wedge sfin\ w) \wedge more) \wedge finite \longrightarrow \Diamond (init\ w)$
using *PiImpDiamond* **by** *fastforce*
have 3: $\vdash (((init\ w) \Pi (f \wedge more) \wedge sfin\ w) \wedge more) \wedge finite =$
 $(((((init\ w) \Pi (f \wedge more) \wedge more) \wedge sfin\ w) \wedge finite)$
by *auto*
have 4: $\vdash (((init\ w) \Pi (f \wedge more) \wedge sfin\ w) \wedge more) \wedge finite; (omega ((init\ w) \Pi (f \wedge more) \wedge sfin\ w)))$
 $=$
 $(((((init\ w) \Pi (f \wedge more) \wedge more) \wedge sfin\ w) \wedge finite); (omega ((init\ w) \Pi (f \wedge more) \wedge sfin\ w)))$

using 3 *LeftChopEqvChop* **by** *blast*
have 5: $\vdash (((init\ w) \Pi (f \wedge more) \wedge more) \wedge sfin\ w) \wedge finite); (\omega ((init\ w) \Pi (f \wedge more) \wedge sfin\ w))$
 $= (((init\ w) \Pi (f \wedge more) \wedge more) \wedge finite); ((init\ w) \wedge \omega ((init\ w) \Pi (f \wedge more) \wedge sfin\ w))$
using *SlideInitSFin*[of *LIFT* $((init\ w) \Pi (f \wedge more) \wedge more)\ w$
 $LIFT\ (\omega ((init\ w) \Pi (f \wedge more) \wedge sfin\ w))]$ **unfolding** *schop-d-def*
by (*metis* *inteq-reflection*)
have 6: $\vdash (((init\ w) \Pi (f \wedge more) \wedge more) \wedge finite); ((init\ w) \wedge \omega ((init\ w) \Pi (f \wedge more) \wedge sfin\ w))$
 $\longrightarrow \Diamond (init\ w)$
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*:

$\vdash (init\ w) \Pi (\omega f) = ((init\ w) \Pi ((f \wedge more) \wedge finite) \wedge more) \wedge sfin\ w; ((init\ w) \Pi (\omega f))$
proof –
have 1: $\vdash (init\ w) \Pi (\omega f) = (init\ w) \Pi (((f \wedge more) \wedge finite); (\omega f))$
by (*simp* *add: OmegaUnroll* *PiEqvRule*)
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** *schop-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 *schop-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 $((\text{nsbn } \sigma \text{ (nnth } l \text{ } i) \text{ (nnth } l \text{ (Suc } i))) \models \text{sfin } w) =$
 $w \text{ (NNil (nnth } \sigma \text{ (nnth } l \text{ (Suc } i))))$
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 $\neg \text{nfinite } l$
 $\text{nnth } l \text{ } 0 = 0$
 $\text{nidx } l$
 $(\forall i. \text{enat (nnth } l \text{ } i) \leq \text{nlength } \sigma)$
 $w \text{ (NNil (nnth } \sigma \text{ (nnth } l \text{ (Suc } i))))$
shows $(\exists x \in \text{nset (nsbn } \sigma \text{ (nnth } l \text{ } i) \text{ (nnth } l \text{ (Suc } i)))} . w \text{ (NNil } x))$
using *assms nidx-expand*[of *l*] *in-nset-conv-nnth*[of - (*nsbn* σ (*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:*

$(\sigma \models (\text{aomega ((init } w) \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfin } w))) =$
 $(\exists l. \neg \text{nfinite } l \wedge$
 $\text{nnth } l \text{ } 0 = 0 \wedge$
 $\text{nidx } l \wedge (\forall i. \text{enat (nnth } l \text{ } i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. ($
 $((\text{nfilter } (\lambda y. w \text{ (NNil } y)) \text{ (nsbn } \sigma \text{ (nnth } l \text{ } i) \text{ (nnth } l \text{ (Suc } i)))} \models f) \wedge$
 $0 < \text{nlength (nfilter } (\lambda y. w \text{ (NNil } y)) \text{ (nsbn } \sigma \text{ (nnth } l \text{ } i) \text{ (nnth } l \text{ (Suc } i)))} \wedge$
 $\text{nfinite (nfilter } (\lambda y. w \text{ (NNil } y)) \text{ (nsbn } \sigma \text{ (nnth } l \text{ } i) \text{ (nnth } l \text{ (Suc } i)))} \wedge$
 $w \text{ (NNil (nnth } \sigma \text{ (nnth } l \text{ (Suc } i))))$
 $)$
 $)$

proof –

have 1: $(\sigma \models (\text{aomega ((init } w) \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfin } w))) =$
 $(\exists l. \neg \text{nfinite } l \wedge$
 $\text{nnth } l \text{ } 0 = 0 \wedge$
 $\text{nidx } l \wedge (\forall i. \text{enat (nnth } l \text{ } i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. \text{nsbn } \sigma \text{ (nnth } l \text{ } i) \text{ (nnth } l \text{ (Suc } i))} \models \text{init } w \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfin } w))$

unfolding *aomega-d-def* **by** *blast*

have 2: $(\exists l. \neg \text{nfinite } l \wedge$
 $\text{nnth } l \text{ } 0 = 0 \wedge$
 $\text{nidx } l \wedge (\forall i. \text{enat (nnth } l \text{ } i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. \text{nsbn } \sigma \text{ (nnth } l \text{ } i) \text{ (nnth } l \text{ (Suc } i))} \models \text{init } w \Pi ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfin } w)) =$
 $(\exists l. \neg \text{nfinite } l \wedge$
 $\text{nnth } l \text{ } 0 = 0 \wedge$
 $\text{nidx } l \wedge (\forall i. \text{enat (nnth } l \text{ } i) \leq \text{nlength } \sigma) \wedge$
 $(\forall i. ($
 $(\exists x \in \text{nset (nsbn } \sigma \text{ (nnth } l \text{ } i) \text{ (nnth } l \text{ (Suc } i)))} . w \text{ (NNil } x)) \wedge$
 $(\text{nfilter } (\lambda y. w \text{ (NNil } y)) \text{ (nsbn } \sigma \text{ (nnth } l \text{ } i) \text{ (nnth } l \text{ (Suc } i)))} \models (f \wedge \text{more}) \wedge \text{finite})$
 $\wedge ((\text{nsbn } \sigma \text{ (nnth } l \text{ } i) \text{ (nnth } l \text{ (Suc } i))) \models \text{sfin } w)$
 $)$
 $)$

```

) )
using Pistate[of w LIFT ((f  $\wedge$  more)  $\wedge$  finite)] by auto
have 3: ( $\exists l. \neg \text{nfinite } l \wedge$ 
   $\text{nnth } l \ 0 = 0 \wedge$ 
   $\text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$ 
   $(\forall i. ( (\exists x \in \text{nset } (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))) . w (\text{NNil } x)) \wedge$ 
     $(\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)))) \models (f \wedge \text{more}) \wedge \text{finite})$ 
     $\wedge ((\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))) \models \text{sfin } w)$ 
  )
) ) =
( $\exists l. \neg \text{nfinite } l \wedge$ 
   $\text{nnth } l \ 0 = 0 \wedge$ 
   $\text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$ 
   $(\forall i. ( (\exists x \in \text{nset } (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))) . w (\text{NNil } x)) \wedge$ 
     $((\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)))) \models f) \wedge$ 
     $0 < \text{nlength } (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)))) \wedge$ 
     $\text{nfinite } (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))))$ 
     $\wedge ((\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))) \models \text{sfin } w)$ 
  )
) )
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 -  $\sigma \ w$ ] by blast
have 5: ( $\exists l. \neg \text{nfinite } l \wedge$ 
   $\text{nnth } l \ 0 = 0 \wedge$ 
   $\text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$ 
   $(\forall i. ( (\exists x \in \text{nset } (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))) . w (\text{NNil } x)) \wedge$ 
     $((\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)))) \models f) \wedge$ 
     $0 < \text{nlength } (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i)))) \wedge$ 
     $\text{nfinite } (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{nsbn } \sigma (\text{nnth } l \ i) (\text{nnth } l \ (\text{Suc } i))))$ 
     $\wedge w (\text{NNil } (\text{nnth } \sigma (\text{nnth } l \ (\text{Suc } i))))$ 
  )
) )

```

```

    )
  ) ) =
  (∃ 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 PiAOmega-sfin-exist[of - σ w] by blast
show ?thesis
using 1 2 3 4 5 by presburger
qed

```

```

primcorec oml :: ('a ⇒ bool) ⇒ 'a nelist ⇒ nat nelist ⇒ nat ⇒ nat nelist
where
  oml P xs l x =
    NCons (case x of 0 ⇒ 0 |
              Suc j ⇒ (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 ¬ nfinite l
          ¬ 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 ¬ nfinite l
          ¬ 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  $(\sigma \models (\text{aomega } ((\text{init } w) \amalg ((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfin } w)))$ 
  shows  $(\sigma \models (\text{init } w) \amalg (\text{aomega } f))$ 
proof -
  have 1:  $(\exists l. \neg \text{nfinite } l \wedge$ 
             $\text{nnth } l \ 0 = 0 \wedge$ 
             $\text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$ 
             $(\forall i. ($ 
               $((\text{nfilter } (\lambda y. w \ (\text{NNil } y)) \ (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)))) \models f) \wedge$ 
               $0 < \text{nlength } (\text{nfilter } (\lambda y. w \ (\text{NNil } y)) \ (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)))) \wedge$ 
               $\text{nfinite } (\text{nfilter } (\lambda y. w \ (\text{NNil } y)) \ (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)))) \wedge$ 
               $w \ (\text{NNil } (\text{nnth } \sigma \ (\text{nnth } l \ (\text{Suc } i))))$ 
             $)$ 
             $)$ 
  )
  using assms PiAOmega-help2[of w f  $\sigma$ ] by blast
  obtain l where 2:  $\neg \text{nfinite } l \wedge$ 
                     $\text{nnth } l \ 0 = 0 \wedge$ 
                     $\text{nidx } l \wedge (\forall i. \text{enat } (\text{nnth } l \ i) \leq \text{nlength } \sigma) \wedge$ 
                     $(\forall i. ($ 
                       $((\text{nfilter } (\lambda y. w \ (\text{NNil } y)) \ (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)))) \models f) \wedge$ 
                       $0 < \text{nlength } (\text{nfilter } (\lambda y. w \ (\text{NNil } y)) \ (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)))) \wedge$ 
                       $\text{nfinite } (\text{nfilter } (\lambda y. w \ (\text{NNil } y)) \ (\text{nsbn } \sigma \ (\text{nnth } l \ i) \ (\text{nnth } l \ (\text{Suc } i)))) \wedge$ 
                       $w \ (\text{NNil } (\text{nnth } \sigma \ (\text{nnth } l \ (\text{Suc } i))))$ 
                     $)$ 
                     $)$ 

```

```

using 1 by auto
have 3:  $\bigwedge j. 0 < j \longrightarrow w \ (NNil \ (nnth \ \sigma \ (nnth \ l \ j)))$ 
  using 2 by (metis Suc-diff-1)
have 31:  $\bigwedge j. 0 < j \longrightarrow nfinite \ (nfilter \ (\lambda y. w \ (NNil \ y)) \ (ntaken \ (nnth \ l \ j) \ \sigma))$ 
  by (metis 2 3 nfilter-chop1-ntaken nfinite-ntaken ntaken-nlast)
have 32:  $\bigwedge j. (nnth \ l \ j) < (nnth \ l \ (Suc \ j))$ 
  by (metis 2 linorder-le-cases nfinite-ntaken nidx-expand ntaken-all)
have 4:  $\bigwedge j. 0 < j \longrightarrow$ 
  ( $nfilter \ (\lambda y. w \ (NNil \ y)) \ (nsubn \ \sigma \ (nnth \ l \ j) \ (nnth \ l \ (Suc \ j))) =$ 
    ( $nsubn \ (nfilter \ (\lambda y. w \ (NNil \ y)) \ \sigma)$ 
      ( $the-enat \ (nlength(nfilter \ (\lambda y. w \ (NNil \ y)) \ (ntaken \ (nnth \ l \ j) \ \sigma)))$ 
        ( $the-enat \ (nlength(nfilter \ (\lambda y. w \ (NNil \ y)) \ (ntaken \ (nnth \ l \ (Suc \ j)) \ \sigma)))$ 
          )
    )

  using 2 3 32 nfilter-nsubn[of -  $\sigma \ (\lambda y. w \ (NNil \ y))$ ]
  by (metis 31 less-imp-le-nat nfinite-nlength-enat the-enat.simps zero-less-Suc)
have 5: ( $nfilter \ (\lambda y. w \ (NNil \ y)) \ (nsubn \ \sigma \ 0 \ (nnth \ l \ (Suc \ 0))) =$ 
  ( $nsubn \ (nfilter \ (\lambda y. w \ (NNil \ y)) \ \sigma)$ 
    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:  $nnth \ (oml \ (\lambda y. w \ (NNil \ y)) \ \sigma \ l \ 0) \ 0 = 0$ 
  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 ∧
  nidx l ∧
  (∀ i. enat (nnth l i) ≤ nlength (nfilter (λy. w (NNil y)) σ)) ∧
  (∀ i. f (nsubn (nfilter (λy. w (NNil y)) σ) (nnth l i) (nnth l (Suc i))))))
using 12 13 14 15 8 oml-nnth-zero by blast
have 17: (σ ⊨ (init w) Π (aomega f))
using 16
using Pistate[of w LIFT aomega f σ] unfolding aomega-d-def by blast
show ?thesis using 17 by auto
qed

```

```

lemma PiAOmegaImp:
  ⊢ (aomega ((init w) Π ((f ∧ more) ∧ finite) ∧ sfin w)) ⟶ (init w) Π (aomega f)
unfolding Valid-def using PiAOmegaImpSem
using unl-lift2 by blast

```

```

lemma PiOmegaImpSem:
  assumes (σ ⊨ (omega ((init w) Π ((f ∧ more) ∧ finite) ∧ sfin w)))
  shows (σ ⊨ (init w) Π (omega f))
by (metis OmegaEqvAOmega PiAOmegaImpSem assms inteq-reflection)

```

```

lemma PiOmegaImp:
  ⊢ (omega ((init w) Π ((f ∧ more) ∧ finite) ∧ sfin w)) ⟶ (init w) Π (omega f)
unfolding Valid-def using PiOmegaImpSem using unl-lift2 by blast

```

```

lemma PiOmega:
  ⊢ (init w) Π (omega f) = (omega ((init w) Π ((f ∧ more) ∧ finite) ∧ sfin w))
proof –
  have 0: ⊢ (((init w) Π ((f ∧ more) ∧ finite) ∧ more) ∧ sfin w) =
    (((init w) Π ((f ∧ more) ∧ finite) ∧ sfin w) ∧ more) ∧ finite)
    using SFinEQvFinAndFinite by fastforce
  have 1: ⊢ (init w) Π (omega f) ⟶ inf
    by (metis AndChopB AndMoreAndFiniteEqvAndFmore OmegaMoreEqvInf OmegaWeakCoinduct 0
      PiStateOmegaUnroll fmore-d-def int-eq int-simps(20))
  have 2: ⊢ (init w) Π (omega f) ⟶
    inf ∧ (((init w) Π ((f ∧ more) ∧ finite) ∧ sfin w) ∧ fmore);((init w) Π (omega f))
    by (metis 1 AndMoreSChopEqvAndFmoreChop 0 PiStateOmegaUnroll Prop12 int-eq int-iffD1 schop-d-def)
  have 3: ⊢ (init w) Π (omega f) ⟶ (omega ((init w) Π ((f ∧ more) ∧ finite) ∧ sfin w))
    using OmegaIntro by (metis 2 Prop12)
  have 4: ⊢ (omega ((init w) Π ((f ∧ more) ∧ finite) ∧ sfin w)) ⟶ (init w) Π (omega f)
    by (simp add: PiOmegaImp)
  show ?thesis
  by (simp add: 3 4 int-iffI)
qed

```

end

14 First Order Finite and Infinite ITL theorems

theory *FOTheorems*

imports

SChopTheorems Chopstar

begin

We give the proofs of a list of first order (in)finite ITL theorems.

lemma *EEExI-unl*:

$w \models f x \implies w \models (\exists \exists x. f x)$

by (*meson exist-state-d-def*)

lemma *EEExNoDep*:

$\vdash (\exists \exists x. g) = g$

proof –

have 1: $\vdash g \longrightarrow (\exists \exists x. g)$ **by** (*meson EEExI*)

have 2: $\bigwedge x. \vdash g \longrightarrow g$ **by** *simp*

have 3: $\vdash (\exists \exists x. g) \longrightarrow g$ **using** 2 **by** (*meson EEExE*)

from 1 3 **show** *?thesis* **using** *int-iffI* **by** *blast*

qed

lemma *AAxNoDep*:

$\vdash (\forall \forall x. g) = g$

using *EEExNoDep*[of *LIFT*($\neg g$)] *AAxDef EEExE EEExI*

by (*simp add: exist-state-d-def forall-state-d-def intI*)

lemma *EEExEqvRule*:

assumes $\bigwedge x. \vdash f x = g x$

shows $\vdash (\exists \exists x. f x) = (\exists \exists x. g x)$

by (*metis EEExE EEExI assms int-iffD1 int-iffD2 int-iffI lift-imp-trans*)

lemma *AAxImpEEEx*:

$\vdash (\forall \forall x. f x) \longrightarrow (\exists \exists x. f x)$

by (*simp add: exist-state-d-def forall-state-d-def intI*)

lemma *EEExImpRule*:

assumes $\vdash f x \longrightarrow g x$

shows $\vdash (\exists \exists x. f x \longrightarrow g x)$

using *assms* **by** (*meson MP EEExI*)

lemma *EEExImpRuleDist*:

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** *EEExI 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 EEExI*)

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 *EExI[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 *EEAndNoDepDist*:
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* *EEI* *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* *EEI*)
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 $\lambda x. LIFT(h;(f x)) x \lambda x. LIFT(h;(g x))$] 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 *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

15 The First Occurrence Operator in finite and infinite ITL

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.

15.1 Definitions

15.1.1 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 :: *lift* \Rightarrow *lift* ((*fprev* -) [88] 87)

syntax (*ASCH*)
 -fprev-d :: *lift* \Rightarrow *lift* ((*fprev* -) [88] 87)

translations
 -fprev-d \equiv *CONST fprev-d*

definition *wfprev-d* :: (*'a::world*) *formula* \Rightarrow *'a formula*
where *wfprev-d* *F* \equiv *LIFT*(\neg (*fprev*(\neg *F*)))

syntax

$-wfp_{prev-d} :: lift \Rightarrow lift ((wfp_{prev} -) [88] 87)$

syntax (*ASCII*)

$-wfp_{prev-d} :: lift \Rightarrow lift ((wfp_{prev} -) [88] 87)$

translations

$-wfp_{prev-d} \Rightarrow CONST wfp_{prev-d}$

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

$-ds-d \Rightarrow CONST ds-d$

15.1.2 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*

15.2 First and Time Reversal

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 *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

15.3 Semantic Theorems

15.4 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*:

$(\forall n. enat\ n \leq nlength\ w \longrightarrow f\ (ntaken\ n\ w) \longrightarrow n = 0 \vee f\ (ntaken\ (n - Suc\ 0)\ w)) \Longrightarrow$

$f\ w \Longrightarrow$

$nfinite\ w \Longrightarrow$

$enat\ j \leq nlength\ w \Longrightarrow$

$enat\ k = nlength\ w \Longrightarrow$

$f\ (ntaken\ j\ w)$

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  $\implies$  ?thesis
      by (simp add: Suc.premis(2) ntaken-all)
    have 6: j < 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*:

$\vdash \text{bi}(f \longrightarrow \text{wprev } f) \wedge f \wedge \text{finite} \longrightarrow \text{bi } f$

proof –

have 1: $\bigwedge j \ w.$

$(\forall n. \text{enat } n \leq \text{nlength } w \longrightarrow f \ (\text{ntaken } n \ w) \longrightarrow n = 0 \vee f \ (\text{ntaken } (n - \text{Suc } 0) \ w)) \implies$

$f \ w \implies$

$n\text{finite } w \implies$

$\text{enat } j \leq \text{nlength } w \implies$

$f \ (\text{ntaken } j \ w)$

by (metis biinduct-help nfinite-conv-nlength-enat)

have 2: $(\vdash \text{bi}(f \longrightarrow \text{wprev } f) \wedge f \wedge \text{finite} \longrightarrow \text{bi } f) =$

$(\forall w \ n. (\forall n. \text{enat } n \leq \text{nlength } w \longrightarrow f \ (\text{ntaken } n \ w) \longrightarrow n = 0 \vee f \ (\text{ntaken } (n - \text{Suc } 0) \ w)) \longrightarrow$

$f \ w \longrightarrow n\text{finite } w \longrightarrow \text{enat } n \leq \text{nlength } w \longrightarrow f \ (\text{ntaken } n \ w))$

by (auto simp add: Valid-def itl-defs)

show ?thesis **using** 1 2 **by** blast

qed

15.4.1 Semantics First Operator

lemma *FstAndBfsem*:

$(\text{nlength } \sigma > 0 \wedge (\sigma \models f) \wedge (\sigma \models \text{bi } (\neg f) \frown \text{skip})) =$

$(\text{nlength } \sigma > 0 \wedge n\text{finite } \sigma \wedge (\sigma \models f) \wedge (\forall ia < \text{nlength } (\sigma). (\text{ntaken } ia \ \sigma \models \neg f)))$

proof –

have $(nlength\ \sigma > 0 \wedge (\sigma \models f) \wedge (\sigma \models bi\ (\neg f) \frown skip)) =$
 $(0 < nlength\ \sigma \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge$
 $(\exists i. (i \leq nlength\ \sigma \longrightarrow (\forall ia \leq i. \neg (ntaken\ ia\ (ntaken\ i\ \sigma) \models f))) \wedge$
 $nlength\ \sigma - i = Suc\ 0) \wedge i \leq nlength\ \sigma)$
 $)$
unfolding *itl-defs* **by** *simp*
 $(metis\ enat-ord-simps(1)\ ndropn-nlength\ nfinite-ndropn-b\ nlength-eq-enat-nfiniteD\ order-subst2)$
also have $\dots =$
 $(0 < nlength\ \sigma \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge$
 $(\exists i. (i \leq nlength\ \sigma \longrightarrow (\forall ia \leq i. \neg (ntaken\ ia\ (ntaken\ i\ \sigma) \models f))) \wedge$
 $i = nlength\ \sigma - Suc\ 0) \wedge i \leq nlength\ \sigma)$
 $)$
by $(auto\ simp\ add: nfinite-conv-nlength-enat)$
 $(metis\ Suc-diff-Suc\ cancel-comm-monoid-add-class.diff-cancel\ diff-zero\ lessI\ ntaken-all$
 $order.order-iff-strict\ zero-less-iff-neq-zero)$
also have $\dots =$
 $(0 < nlength\ \sigma \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge$
 $(\forall ia. enat\ ia \leq (nlength\ \sigma - Suc\ 0) \longrightarrow \neg (ntaken\ ia\ (ntaken\ (the-enat\ (nlength\ \sigma - Suc\ 0))\ \sigma) \models$
 $f))$
 $)$
using *diff-le-self* **by** $(auto\ simp\ add: min-def\ nfinite-conv-nlength-enat, blast)$
also have $\dots =$
 $(nlength\ \sigma > 0 \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge$
 $(\forall ia. ia < nlength\ (\sigma) \longrightarrow \neg (ntaken\ ia\ (ntaken\ (the-enat\ (nlength\ \sigma - Suc\ 0))\ \sigma) \models f)))$
 $)$
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 $\dots =$
 $(nlength\ \sigma > 0 \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge$
 $(\forall ia. ia < nlength\ (\sigma) \longrightarrow (ntaken\ ia\ (ntaken\ (the-enat\ (nlength\ \sigma - Suc\ 0))\ \sigma) \models \neg f)))$
 $)$
by *auto*
also have $\dots =$
 $(nlength\ \sigma > 0 \wedge nfinite\ \sigma \wedge (\sigma \models f) \wedge (\forall ia < nlength\ (\sigma). (ntaken\ ia\ \sigma \models \neg f)))$
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*:

$$(\sigma \models \triangleright f) =$$

$$(\quad$$

$$(\quad (\sigma \models f) \wedge \text{nlength } \sigma = 0) \vee$$

$$(\quad \text{nlength } \sigma > 0 \wedge \text{nfinite } \sigma \wedge (\sigma \models f) \wedge (\forall ia < \text{nlength } (\sigma). (\text{ntaken } ia \ \sigma \models \neg f)))$$

$$\quad)$$

using *Fstsem-0 Emptysem FstAndBfsem* **by** *metis*

15.4.2 Various Semantic Lemmas

lemma *DiLensem*:

$$(\sigma \models di (f \wedge \text{len}(i))) =$$

$$(\quad (\text{ntaken } i \ \sigma \models f) \wedge i \leq \text{nlength } \sigma)$$

using *nlength-eq-enat-nfiniteD* **by** *(auto simp add: itl-defs len-defs)*

lemma *DfLensem*:

$$(\sigma \models df (f \wedge \text{len}(i))) =$$

$$(\quad (\text{ntaken } i \ \sigma \models f) \wedge i \leq \text{nlength } \sigma)$$

by *(auto simp add: itl-defs len-defs)*

lemma *PrefixFstsem*:

$$(\quad (\text{ntaken } i \ \sigma \models \triangleright f) \wedge i \leq \text{nlength } \sigma) =$$

$$(\quad i \leq \text{nlength } \sigma \wedge$$

$$(\quad$$

$$(\quad (\text{ntaken } i \ \sigma \models f) \wedge i = 0) \vee$$

$$(\quad i > 0 \wedge (\text{ntaken } i \ \sigma \models f) \wedge (\forall ia < i. (\text{ntaken } ia \ \sigma \models \neg f)))$$

$$\quad)$$

$$\quad)$$

proof –

have 1: $((\text{ntaken } i \ \sigma) \models \triangleright f) =$

$$(\quad$$

$$(\quad ((\text{ntaken } i \ \sigma) \models f) \wedge \text{nlength } (\text{ntaken } i \ \sigma) = 0) \vee$$

$$(\quad \text{nlength } (\text{ntaken } i \ \sigma) > 0 \wedge ((\text{ntaken } i \ \sigma) \models f) \wedge$$

$$(\quad \forall ia < \text{nlength } (\text{ntaken } i \ \sigma). (\text{ntaken } ia \ (\text{ntaken } i \ \sigma) \models \neg f)) \quad)$$

$$\quad)$$

using *Fstsem* **by** *(metis nfinite-ntaken)*

hence 2: $((\text{ntaken } i \ \sigma) \models \triangleright f) \wedge i \leq \text{nlength } \sigma =$

$$(\quad i \leq \text{nlength } \sigma \wedge$$

$$(\quad ((\text{ntaken } i \ \sigma) \models f) \wedge \text{nlength } (\text{ntaken } i \ \sigma) = 0) \vee$$

$$(\quad \text{nlength } (\text{ntaken } i \ \sigma) > 0 \wedge ((\text{ntaken } i \ \sigma) \models f) \wedge$$

$$(\quad \forall ia < \text{nlength } (\text{ntaken } i \ \sigma). (\text{ntaken } ia \ (\text{ntaken } i \ \sigma) \models \neg f)) \quad)$$

$$\quad)$$

by *auto*

hence 3: $((\text{ntaken } i \ \sigma) \models \triangleright f) \wedge i \leq \text{nlength } \sigma =$

$$(\quad i \leq \text{nlength } \sigma \wedge$$

$$(\quad ((\text{ntaken } i \ \sigma) \models f) \wedge i = 0) \vee$$

$$(\quad i > 0 \wedge ((\text{ntaken } i \ \sigma) \models f) \wedge (\forall ia < i. (\text{ntaken } ia \ (\text{ntaken } i \ \sigma) \models \neg f)))$$

$$\quad)$$

```

)
  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 σ) ⊨ ▷f) ∧ i ≤ nlength σ ) =
  ( i ≤ nlength σ ∧ (
    ( (ntaken i σ) ⊨ f ) ∧ i = 0 ) ∨
    ( i > 0 ∧ ((ntaken i σ) ⊨ f) ∧ (∀ ia < i. (ntaken ia σ ⊨ ¬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:
( $\sigma \models df ((\triangleright f \wedge g) \wedge len(i))$ ) =
  (  $i \leq nlength \ \sigma \wedge$ 
    (
      (  $(ntaken \ i \ \sigma \models f \wedge g) \wedge i = 0$ )  $\vee$ 
      (  $i > 0 \wedge (ntaken \ i \ \sigma \models f \wedge g) \wedge (\forall ia < i. (ntaken \ ia \ \sigma \models \neg f))$ )
    )
  )
using DfLensem PrefixFstAndsem by metis

```

```

lemma FstLenSamesem:
( (  $i \leq nlength \ \sigma \wedge$ 
  (
    (  $(ntaken \ i \ \sigma \models f) \wedge i = 0$ )  $\vee$ 
    (  $i > 0 \wedge (ntaken \ i \ \sigma \models f) \wedge (\forall ia < i. (ntaken \ ia \ \sigma \models \neg f))$ )
  )
)  $\wedge$ 
(  $j \leq nlength \ \sigma \wedge$ 
  (
    (  $(ntaken \ j \ \sigma \models f) \wedge j = 0$ )  $\vee$ 
    (  $j > 0 \wedge (ntaken \ j \ \sigma \models f) \wedge (\forall ia < j. (ntaken \ ia \ \sigma \models \neg f))$ )
  )
)
)  $\longrightarrow (i=j)$ 

```

by (*metis not-less-iff-gr-or-eq unl-lift*)

15.5 Theorems

15.5.1 Fixed length intervals

lemma *LenZeroEqvEmpty*:

```

 $\vdash len(0) = empty$ 
by (simp add: len-d-def)

```

lemma *LenOneEqvSkip*:

```

 $\vdash len(1) = skip$ 
by (simp add: len-d-def ChopEmpty)

```

lemma *LenNPlusOneA*:

```

 $\vdash len(n+1) = skip;len(n)$ 
by (simp add: len-d-def)

```

lemma *SkipFiniteEqvFiniteSkip*:

```

 $\vdash skip;finite = finite;skip$ 
using FiniteChopSkipEqvSkipChopFinite by auto

```

lemma *TrueEqvFiniteOrInfinite*:

$\vdash \#True = finite \vee inf$

by (*simp add: FiniteOrInfinite*)

lemma *SkipInfEqvInfSkip*:

$\vdash skip;inf = inf;skip$

by (*metis MoreAndInfEqvInf PowerstarEqvSemhelp2 WOmegaUnroll int-eq womega-skip-inf-eq*)

lemma *SkipTrueEqvTrueSkip*:

$\vdash skip;\#True = \#True;skip$

proof –

have 1: $\vdash skip;\#True = (skip;finite \vee skip;inf)$

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 inteq-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*:

$\vdash ((f0 \wedge \text{len}(k)) \frown g0 \wedge (f1 \wedge \text{len}(k)) \frown g1) = ((f0 \wedge f1) \wedge \text{len}(k)) \frown (g0 \wedge g1)$

by (*auto simp add: Valid-def len-defs schop-defs*)

lemma *RFixedAndDistr*:

$\vdash (f0; (g0 \wedge \text{len}(k)) \wedge f1; (g1 \wedge \text{len}(k))) = (f0 \wedge f1); ((g0 \wedge g1) \wedge \text{len}(k))$

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:  $\vdash ((\#True \wedge len(k)); f \wedge (\#True \wedge len(k)); g) = (\#True \wedge len(k)); (f \wedge g)$ 
    using LFixedAndDistrB by blast
  have 5:  $\vdash (\#True \wedge len(k)); (f \wedge g) = (len(k)); (f \wedge g)$ 
    by auto
  from 1 2 3 4 5 show ?thesis by auto
qed

```

```

lemma RFixedAndDistrA:
 $\vdash (f0;(g0 \wedge len(k)) \wedge f0;(g1 \wedge len(k))) = f0;((g0 \wedge g1) \wedge len(k))$ 
proof -
  have 1:  $\vdash (f0;(g0 \wedge len(k)) \wedge f0;(g1 \wedge len(k))) = (f0 \wedge f0);((g0 \wedge g1) \wedge len(k))$ 
    by (rule RFixedAndDistr)
  have 2:  $\vdash (f0 \wedge f0);((g0 \wedge g1) \wedge len(k)) = f0;((g0 \wedge g1) \wedge len(k))$ 
    by auto
  from 1 2 show ?thesis by fastforce
qed

```

```

lemma RFixedAndDistrAS:
 $\vdash (f0 \neg (g0 \wedge len(k)) \wedge f0 \neg (g1 \wedge len(k))) = f0 \neg ((g0 \wedge g1) \wedge len(k))$ 
proof -
  have 1:  $\vdash (f0 \neg (g0 \wedge len(k)) \wedge f0 \neg (g1 \wedge len(k))) = (f0 \wedge f0) \neg ((g0 \wedge g1) \wedge len(k))$ 
    by (rule RFixedAndDistrS)
  have 2:  $\vdash (f0 \wedge f0) \neg ((g0 \wedge g1) \wedge len(k)) = f0 \neg ((g0 \wedge g1) \wedge len(k))$ 
    by auto
  from 1 2 show ?thesis by fastforce
qed

```

```

lemma RFixedAndDistrB:
 $\vdash (f0;(g0 \wedge len(k)) \wedge f1;(g0 \wedge len(k))) = (f0 \wedge f1);(g0 \wedge len(k))$ 
proof -
  have 1:  $\vdash (f0;(g0 \wedge len(k)) \wedge f1;(g0 \wedge len(k))) = (f0 \wedge f1);((g0 \wedge g0) \wedge len(k))$ 
    by (rule RFixedAndDistr)
  have 2:  $\vdash (f0 \wedge f1);((g0 \wedge g0) \wedge len(k)) = (f0 \wedge f1);(g0 \wedge len(k))$ 
    by auto
  from 1 2 show ?thesis by fastforce
qed

```

```

lemma RFixedAndDistrBS:
 $\vdash (f0 \neg (g0 \wedge len(k)) \wedge f1 \neg (g0 \wedge len(k))) = (f0 \wedge f1) \neg (g0 \wedge len(k))$ 
proof -
  have 1:  $\vdash (f0 \neg (g0 \wedge len(k)) \wedge f1 \neg (g0 \wedge len(k))) = (f0 \wedge f1) \neg ((g0 \wedge g0) \wedge len(k))$ 
    by (rule RFixedAndDistrS)
  have 2:  $\vdash (f0 \wedge f1) \neg ((g0 \wedge g0) \wedge len(k)) = (f0 \wedge f1) \neg (g0 \wedge len(k))$ 
    by auto
  from 1 2 show ?thesis by fastforce
qed

```

```

lemma ChopSkipAndChopSkip:
 $\vdash (f0;skip \wedge f1;skip) = (f0 \wedge f1);skip$ 

```

proof –

have 1: $\vdash (f0;(\#True \wedge len(1)) \wedge f1;(\#True \wedge len(1))) = (f0 \wedge f1);(\#True \wedge len(1))$
by (rule *RFixedAndDistrB*)
have 2: $\vdash (\#True \wedge len(1)) = skip$
using *LenOneEqvSkip* **by** *fastforce*
hence 3: $\vdash f0;(\#True \wedge len(1)) = f0;skip$
using *RightChopEqvChop* **by** *blast*
have 4: $\vdash f1;(\#True \wedge len(1)) = f1;skip$
using 2 *RightChopEqvChop* **by** *blast*
have 5: $\vdash (f0;(\#True \wedge len(1)) \wedge f1;(\#True \wedge len(1))) = (f0;skip \wedge f1;skip)$
using 3 4 **by** *fastforce*
have 6: $\vdash (f0 \wedge f1);(\#True \wedge len(1)) = (f0 \wedge f1);skip$
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*:

$\vdash \text{more} = \text{more};\# \text{True}$

proof –

have 1: $\vdash \text{more} = \text{skip};\# \text{True}$

using *MoreEqvSkipChopTrue* **by** *blast*

have 2: $\vdash \# \text{True} = \# \text{True};\# \text{True}$

by (*auto simp add: Valid-def chop-defs*)
(*metis zero-enat-def zero-le*)

hence 3: $\vdash \text{skip};\# \text{True} = \text{skip};(\# \text{True};\# \text{True})$

using *RightChopEqvChop* **by** *blast*

have 4: $\vdash \text{skip};(\# \text{True};\# \text{True}) = (\text{skip};\# \text{True});\# \text{True}$

using *ChopAssoc* **by** *blast*

have 5: $\vdash (\text{skip};\# \text{True});\# \text{True} = \text{more};\# \text{True}$

using *MoreEqvSkipChopTrue* **by** (*simp add: more-d-def next-d-def*)

from 1 3 4 5 **show** *?thesis* **by** *fastforce*

qed

lemma *MoreEqvMoreSChopTrue*:

$\vdash \text{more} = \text{more} \frown \# \text{True}$

by (*metis MoreEqvSkipSChopTrue SChopAssoc TrueEqvTrueSChopTrue inteq-reflection*)

lemma *NotNotChopSkip*:

$\vdash (\neg((\neg f) ; \text{skip})) = (\text{empty} \vee (f; \text{skip}))$

by (*metis WprevEqvEmptyOrPrev prev-d-def wprev-d-def*)

lemma *NotSChopFixed*:

$\vdash (\neg(f \frown (g \wedge \text{len}(k)))) = (\neg(\Diamond(g \wedge \text{len}(k)))) \vee ((\neg f) \frown (g \wedge \text{len}(k)))$

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*:

$\vdash (\neg((\neg f) \frown \text{skip})) = (\text{empty} \vee \text{inf} \vee (f \frown \text{skip}))$

proof –

have 1: $\vdash ((\neg f) \frown \text{skip} \vee f \frown \text{skip}) = (\neg f \vee f) \frown \text{skip}$

by (*meson OrSChopEqv Prop11*)

have 2: $\vdash (\neg f \vee f) = \# \text{True}$

by *simp*

have 3: $\vdash \text{more} \wedge \text{finite} \longrightarrow \# \text{True} \frown \text{skip}$

by (*metis DiamondSChopdef FmoreEqvSkipChopFinite SkipFiniteEqvFiniteSkip fmore-d-def*
int-simps(13) int-simps(9) inteq-reflection sometimes-d-def)

have 4: $\vdash \text{more} \wedge \text{finite} \longrightarrow ((\neg f) \frown \text{skip}) \vee (f \frown \text{skip})$

using 1 3 **by** *fastforce*

have 5: $\vdash \text{more} \wedge \text{finite} \wedge \neg((f \frown \text{skip})) \longrightarrow ((\neg f) \frown \text{skip})$

using 4 **by** *fastforce*

have 6: $\vdash (\neg((\neg f) \frown \text{skip})) \longrightarrow \text{empty} \vee \text{inf} \vee (f \frown \text{skip})$

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*:

$\vdash (\neg((g \wedge len(k)) \frown f)) = (\neg(df(g \wedge len(k))) \vee ((g \wedge len(k)) \frown (\neg f)))$

proof –

```

have 1:  $\vdash (g \wedge len(k)) \frown f = (g \wedge len(k)); f$ 
  by (meson LFixedChopEqvFixedSChop Prop11)
have 2:  $\vdash (g \wedge len(k)) \frown (\neg f) = ((g \wedge len(k)); (\neg f))$ 
  by (meson LFixedChopEqvFixedSChop Prop11)
have 3:  $\vdash df(g \wedge len(k)) = di(g \wedge len(k))$ 
  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*:

$\vdash (\neg(f; skip)) = (empty \vee ((\neg f); skip))$

proof –

```

have 1:  $\vdash (\neg((\neg(\neg f)); skip)) = (empty \vee ((\neg f); skip))$  using NotNotChopSkip by blast
have 2:  $\vdash (\neg((\neg(\neg f)); skip)) = (\neg(f; skip))$  by auto
from 1 2 show ?thesis by auto

```

qed

lemma *NotSChopNotSkip*:

$\vdash (\neg(f \frown skip)) = (empty \vee inf \vee ((\neg f) \frown skip))$

by (*metis NotNotSChopSkip int-simps(4) inteq-reflection*)

lemma *NotNotSChopInf*:

$\vdash (\neg(\neg f \frown inf)) = (finite \vee (bf f))$

by (*auto simp add: Valid-def itl-defs*)

15.5.2 Additional ITL theorems

lemma *DiAndFiniteEqvDfAndFinite*:

$\vdash (di f \wedge finite) = (df f \wedge finite)$

by (*auto simp add: Valid-def di-defs df-defs finite-defs*)

lemma *DiEqvDfOrInf*:
 $\vdash di\ f = (df\ f \vee (f \wedge inf))$
by (*simp add: ChopSChopdef df-d-def di-d-def*)

lemma *BiEqvBfAndfinite*:
 $\vdash bi\ f = (bf\ f \wedge (inf \longrightarrow f))$
proof –
have 1: $\vdash di\ (\neg f) = (df\ (\neg f) \vee \neg f \wedge inf)$
using *DiEqvDfOrInf[of LIFT $\neg f$]* **by** *blast*
have 2: $\vdash (\neg di\ (\neg f)) = (\neg df\ (\neg f) \wedge (inf \longrightarrow f))$
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*:

$\vdash \text{more} \wedge \text{bf } f \longrightarrow \text{inf} \vee (\text{bf } f) \frown \text{skip}$

proof –

have 1: $\vdash (\text{bf } f) \frown \text{skip} = ((\neg(\text{df } (\neg f))) \frown \text{skip})$ **by** (*simp add: bf-d-def*)

have 2: $\vdash (\neg(\neg(\neg(\text{df } (\neg f))) \frown \text{skip})) = (\text{empty} \vee \text{inf} \vee (\text{df } (\neg f)) \frown \text{skip})$ **by** (*rule NotNotSChopSkip*)

have 3: $\vdash \text{empty} \longrightarrow \text{empty} \vee \text{inf} \vee \text{df } (\neg f)$ **by** *auto*

have 4: $\vdash (\text{df } (\neg f)) \frown \text{skip} \longrightarrow \text{df } (\neg f)$

by (*metis DfEqvDfDf SChopImpDf inteq-reflection*)

hence 5: $\vdash (\text{df } (\neg f)) \frown \text{skip} \longrightarrow \text{empty} \vee \text{inf} \vee \text{df } (\neg f)$

by (*simp add: Prop05*)

have 6: $\vdash \neg(\neg(\text{df } (\neg f))) \frown \text{skip} \longrightarrow \text{empty} \vee \text{inf} \vee \text{df } (\neg f)$ **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*:

$\vdash bi (bi f \vee bi g) \longrightarrow bi f \vee bi g$

using *BiElim* **by** *auto*

lemma *BfBfOrImpBf*:

$\vdash bf (bf f \vee bf g) \longrightarrow bf f \vee bf g$

proof –

have 1: $\vdash bf (bf f \vee bf g) \wedge finite \longrightarrow bf f \vee bf g$

using *BfElim* **by** *fastforce*

have 2: $\vdash bf (bf f \vee bf g) \wedge inf \longrightarrow bf f \vee bf g$

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: $\vdash (empty \vee (\neg(f \wedge finite));skip) = (empty \vee (\neg f \vee inf);skip)$
by (metis 1 2 inteq-reflection)
show ?thesis
by (metis 1 3 inteq-reflection)
qed

lemma WPrevAndFiniteEqv:
 $\vdash wprev\ (f \wedge finite) = (empty \vee f \frown skip)$
unfolding wprev-d-def prev-d-def schop-d-def
by (simp add: NotNotChopSkip)

lemma BiInductB:
 $\vdash bi(f \longrightarrow wprev\ (f \wedge finite)) \wedge f \wedge finite \longrightarrow bi\ f$
unfolding Valid-def itl-defs
by simp
(metis biinduct-help nfinite-conv-nlength-enat)

lemma BfInduct:
 $\vdash bf(f \longrightarrow wprev\ f) \wedge f \wedge finite \longrightarrow bf\ f$
by (auto simp add: Valid-def itl-defs)
(metis biinduct-help nfinite-conv-nlength-enat)

lemma PrevLoop:
assumes $\vdash f \longrightarrow prev\ f$
shows $\vdash finite \longrightarrow \neg f$
proof –
have 1: $\vdash f \longrightarrow prev\ f$ **using** assms **by** auto
hence 2: $\vdash f \longrightarrow (more \wedge wprev\ f)$
by (metis ChopSkipImpMore Prop05 Prop12 WprevEqvEmptyOrPrev inteq-reflection lift-imp-trans prev-d-def)
hence 3: $\vdash f \longrightarrow wprev\ f$ **by** auto
hence 4: $\vdash bi(f \longrightarrow wprev\ f)$ **by** (rule BiGen)
have 5: $\vdash bi(f \longrightarrow wprev\ f) \wedge f \wedge finite \longrightarrow bi\ f$ **by** (rule BiInduct)
hence 6: $\vdash bi(f \longrightarrow wprev\ f) \longrightarrow (f \wedge finite \longrightarrow bi\ f)$ **by** fastforce
have 7: $\vdash (f \wedge finite \longrightarrow bi\ f)$ **using** 4 6 MP **by** blast
have 8: $\vdash bi\ f \longrightarrow f$ **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: $\vdash bi(f \wedge (bi(\neg g)) \longrightarrow wprev(f \wedge bi(\neg g)))$
 by (rule BiGen)
 have 13: $\vdash bi(f \wedge (bi(\neg g)) \longrightarrow wprev(f \wedge bi(\neg g))) \wedge$
 $(f \wedge (bi(\neg g))) \wedge finite \longrightarrow bi(f \wedge (bi(\neg g)))$
 using BiInduct by auto
 hence 14: $\vdash bi(f \wedge (bi(\neg g)) \longrightarrow wprev(f \wedge bi(\neg g))) \longrightarrow$
 $((f \wedge (bi(\neg g))) \wedge finite \longrightarrow bi(f \wedge (bi(\neg g))))$
 by fastforce
 have 15: $\vdash ((f \wedge (bi(\neg g))) \wedge finite \longrightarrow bi(f \wedge (bi(\neg g))))$
 using 12 14 MP by blast
 have 16: $\vdash bi(f \wedge (bi(\neg g))) \longrightarrow (f \wedge (bi(\neg g)))$
 by (rule BiElim)
 have 17: $\vdash finite \longrightarrow bi(f \wedge (bi(\neg g))) = (f \wedge (bi(\neg g)))$
 using 16 15 by fastforce
 have 18: $\vdash (f \wedge (bi(\neg g))) \longrightarrow more$
 using 9 by auto
 hence 19: $\vdash bi(f \wedge (bi(\neg g))) \longrightarrow bi\ more$
 by (simp add: BiImpBiRule)
 have 191: $\vdash (\neg more) = empty$
 by (simp add: empty-d-def)
 have 20: $\vdash (\neg(bi\ more))$
 unfolding bi-d-def di-d-def
 by (metis EmptyChop TrueW empty-d-def int-simps(2) int-simps(3) inteq-reflection)
 have 21: $\vdash finite \longrightarrow \neg(f \wedge (bi(\neg g)))$
 using 17 19 20 by fastforce
 hence 22: $\vdash finite \longrightarrow \neg f \vee \neg(bi(\neg g))$
 by auto
 have 23: $\vdash (\neg(bi(\neg g))) = di\ g$
 by (auto simp: bi-d-def)
 show ?thesis using 22 23 by fastforce
 qed

lemma DiIntroC:

assumes $\vdash (f \wedge \neg g) \longrightarrow prev(f \wedge finite)$
 shows $\vdash f \wedge finite \longrightarrow di\ g$
 using assms
 by (metis AndChopA DiIntroB lift-imp-trans prev-d-def)

lemma DfIntroB:

assumes $\vdash (f \wedge \neg g) \longrightarrow prev\ f$
 shows $\vdash f \wedge finite \longrightarrow df\ g$
 proof –
 have 1: $\vdash f \wedge finite \longrightarrow di\ g$
 by (simp add: DiIntroB assms)
 have 2: $\vdash di\ g = (df\ g \vee (g \wedge inf))$
 by (simp add: DiEqvDfOrInf)
 have 3: $\vdash f \wedge finite \longrightarrow finite \wedge (df\ g \vee (g \wedge inf))$
 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:
   $\vdash \text{di } f = (f \vee f;\text{more})$ 
proof –
  have 1:  $\vdash \text{di } f = f;\#\text{True}$  by (simp add: di-d-def)
  hence 2:  $\vdash \text{di } f = f; (\text{empty} \vee \text{more})$  by (simp add: empty-d-def)
  have 3:  $\vdash f; (\text{empty} \vee \text{more}) = (f;\text{empty} \vee f;\text{more})$  by (simp add: ChopOrEqv)
  have 4:  $\vdash f;\text{empty} = f$  by (rule ChopEmpty)
  from 2 3 4 show ?thesis by fastforce
qed

```

```

lemma DfEqvOrSChopMore:
   $\vdash \text{df } f = ((f \wedge \text{finite}) \vee f \frown \text{more})$ 
proof –
  have 1:  $\vdash \text{df } f = f \frown \#\text{True}$  by (simp add: df-d-def)
  hence 2:  $\vdash \text{df } f = f \frown (\text{empty} \vee \text{more})$  by (simp add: empty-d-def)
  have 3:  $\vdash f \frown (\text{empty} \vee \text{more}) = (f \frown \text{empty} \vee f \frown \text{more})$  by (simp add: SChopOrEqv)
  have 4:  $\vdash f \frown \text{empty} = (f \wedge \text{finite})$  by (simp add: ChopEmpty schop-d-def)
  from 2 3 4 show ?thesis by fastforce
qed

```

```

lemma DiAndDiEqvDiAndDiOrDiAndDi:
   $\vdash (\text{di } f \wedge \text{di } g) = (\text{di}(f \wedge \text{di } g) \vee \text{di}(g \wedge \text{di } f))$ 
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: $\vdash \Box (init\ w) = (\neg (di\ (finite \wedge fin\ (init\ (\neg w)))))$

using 5 *always-d-def Initprop(2)* **by** (*metis int-eq*)

have 7: $\vdash \Box (init\ w) = bi\ (finite \longrightarrow \neg (fin\ (init\ (\neg w))))$

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: $\vdash \Box (init\ w) = bi\ (finite \longrightarrow fin\ (init\ w))$

using 7 12 **by** *fastforce*

from 13 **show** *?thesis* **by** *simp*

qed

lemma *BoxStateEqvBfFinState*:

$\vdash \Box (init\ w) = bf\ (fin\ (init\ w))$

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: $\vdash \square (init (\neg w)) = bi (finite \longrightarrow_{fin} (init (\neg w)))$
using *BoxStateEqvBiFinState* **by** *blast*
have 2: $\vdash (\neg (\square (init (\neg w)))) = (\neg (bi (finite \longrightarrow_{fin} (init (\neg w)))))$
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*:

$\vdash ((f \wedge \text{finite}) \wedge \text{df } f) = (f \wedge \text{finite})$

by (*meson AndFiniteImpDf Prop10 Prop11*)

lemma *BiEmptyEqvEmpty*:

$\vdash \text{bi empty} = \text{empty}$

proof –

have 1: $\vdash \text{bi empty} = (\neg (\text{di } (\neg \text{empty})))$ **by** (*simp add: bi-d-def*)

have 2: $\vdash (\neg (\text{di } (\neg \text{empty}))) = (\neg ((\neg \text{empty}); \# \text{True}))$ **by** (*simp add: di-d-def*)

have 3: $\vdash (\neg ((\neg \text{empty}); \# \text{True})) = (\neg (\text{more}; \# \text{True}))$ **by** (*simp add: empty-d-def*)

have 4: $\vdash \text{more}; \# \text{True} = \text{more}$ **using** *MoreEqvMoreChopTrue* **by** *auto*

hence 5: $\vdash (\neg (\text{more}; \# \text{True})) = (\neg \text{more})$ **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: $\vdash ((\neg f);skip) = (\neg(empty \vee (f;skip)))$
by *fastforce*
have 6: $\vdash \neg(empty \vee (f;skip)) = (more \wedge \neg(f;skip))$
using 5 *NotChopSkipEqvMoreAndNotChopSkip* **by** *fastforce*
have 7: $\vdash ((\neg f);skip \vee g;skip) = ((more \wedge \neg(f;skip)) \vee g;skip)$
using 5 6 **by** *fastforce*
hence 8: $\vdash (more \wedge (\neg f;skip \vee g;skip)) = (more \wedge ((more \wedge \neg(f;skip)) \vee g;skip))$
by *auto*
have 9: $\vdash (more \wedge ((more \wedge \neg(f;skip)) \vee g;skip)) = (more \wedge (\neg(f;skip) \vee g;skip))$
by *auto*
have 10: $\vdash (more \wedge (\neg(f;skip) \vee g;skip)) = (more \wedge ((f;skip) \longrightarrow (g;skip)))$
by *auto*
have 11: $\vdash (more \wedge (f \longrightarrow g);skip) = (more \wedge ((f;skip) \longrightarrow (g;skip)))$
using 1 2 3 8 9 10 7 **by** *fastforce*
from 11 show ?thesis using MP by fastforce
qed

lemma *MoreImpImpSChopSkipEqv:*

$\vdash more \wedge finite \longrightarrow ((f \longrightarrow g) \frown skip = ((f \frown skip) \longrightarrow (g \frown skip)))$

proof –

have 01: $\vdash (f \longrightarrow g) = (\neg f \vee g)$ **by** *auto*
hence 02: $\vdash (f \longrightarrow g) \frown skip = (\neg f \vee g) \frown skip$ **by** (*simp add: LeftSChopEqvSChop*)
have 03: $\vdash (f \longrightarrow g) \frown skip \longrightarrow finite$
by (*metis LenOneEqvSkip SChopImpFinite inteq-reflection len-k-finite*)
have 04: $\vdash (more \wedge (f \longrightarrow g) \frown skip) = (more \wedge finite \wedge (f \longrightarrow g) \frown skip)$
using 03 **by** *auto*
have 1: $\vdash (more \wedge finite \wedge (f \longrightarrow g) \frown skip) = (more \wedge finite \wedge (\neg f \vee g) \frown skip)$
by (*metis 01 04 inteq-reflection*)
have 2: $\vdash (\neg f \vee g) \frown skip = ((\neg f) \frown skip \vee g \frown skip)$
using *OrSChopEqv* **by** *auto*
hence 3: $\vdash (more \wedge finite \wedge (\neg f \vee g) \frown skip) = (more \wedge finite \wedge ((\neg f) \frown skip \vee g \frown skip))$
by *auto*
have 4: $\vdash (\neg((\neg f) \frown skip)) = (empty \vee inf \vee (f \frown skip))$
using *NotNotSChopSkip* **by** *blast*
hence 5: $\vdash ((\neg f) \frown skip) = (\neg(empty \vee inf \vee (f \frown skip)))$
by *fastforce*
have 6: $\vdash \neg(empty \vee inf \vee (f \frown skip)) = (more \wedge finite \wedge \neg(f \frown skip))$
using 5 **unfolding** *finite-d-def empty-d-def* **by** *fastforce*
have 7: $\vdash ((\neg f) \frown skip \vee g \frown skip) = ((more \wedge finite \wedge \neg(f \frown skip)) \vee g \frown skip)$
using 5 6 **by** *fastforce*
hence 8: $\vdash (more \wedge finite \wedge (\neg(f \frown skip) \vee g \frown skip)) =$
 $(more \wedge finite \wedge ((more \wedge finite \wedge \neg(f \frown skip)) \vee g \frown skip))$
by *fastforce*
have 9: $\vdash (more \wedge finite \wedge ((more \wedge finite \wedge \neg(f \frown skip)) \vee g \frown skip)) =$
 $(more \wedge finite \wedge (\neg(f \frown skip) \vee g \frown skip))$
by *auto*
have 10: $\vdash (more \wedge finite \wedge (\neg(f \frown skip) \vee g \frown skip)) =$
 $(more \wedge finite \wedge ((f \frown skip) \longrightarrow (g \frown skip)))$
by *auto*
have 11: $\vdash (more \wedge finite \wedge (f \longrightarrow g) \frown skip) = (more \wedge finite \wedge ((f \frown skip) \longrightarrow (g \frown skip)))$

```

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:
   $\vdash \text{bf}(\Box (\neg f)) = (\neg((\# \text{True} \frown f) \frown \# \text{True}))$ 
by (simp add: bf-d-def always-d-def df-d-def sometimes-d-def schop-d-def)

```

```

lemma DiAndEmptyEqvAndEmpty:
   $\vdash (\text{di } f \wedge \text{empty}) = (f \wedge \text{empty})$ 
by (metis ChopEmptyAndEmpty di-d-def int-simps(17) inteq-reflection)

```

```

lemma DfAndEmptyEqvAndEmpty:
   $\vdash (\text{df } f \wedge \text{empty}) = (f \wedge \text{empty})$ 
by (metis ChopEmptyAndEmpty DiAndEmptyEqvAndEmpty Prop04 SChopEmptyAndEmpty df-d-def di-d-def)

```

```

lemma FiniteImpBfEqvBi:
 $\vdash \text{finite} \longrightarrow \text{bf } f = \text{bi } f$ 
proof -
  have 1:  $\vdash \text{bi } f = (\text{bf } f \wedge (\text{inf} \longrightarrow f))$ 
  by (simp add: BiEqvBfAndfinite)
  have 2:  $\vdash \text{finite} \longrightarrow (\text{bf } f \wedge (\text{inf} \longrightarrow f)) = \text{bf } f$ 
  unfolding finite-d-def by fastforce
  show ?thesis
  using 1 2 by fastforce
qed

```

```

lemma FiniteImpDfEqvDi:
 $\vdash \text{finite} \longrightarrow \text{df } f = \text{di } f$ 
proof -
  have 1:  $\vdash \text{di } f = (\text{df } f \vee (f \wedge \text{inf}))$ 
  by (simp add: DiEqvDfOrInf)
  have 2:  $\vdash \text{finite} \longrightarrow (\text{df } f \vee (f \wedge \text{inf})) = \text{df } f$ 
  unfolding finite-d-def by fastforce
  show ?thesis using 1 2 by fastforce
qed

```

15.5.3 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: $\vdash (\neg(empty \vee (bi\ (\neg f)) \frown skip)) = (\neg empty \wedge \neg((bi\ (\neg f)) \frown skip))$

by *auto*

have 4: $\vdash (\neg empty \wedge \neg((bi\ (\neg f)) \frown skip)) = (more \wedge \neg((bi\ (\neg f)) \frown skip))$

unfolding *empty-d-def* **by** *auto*

have 40: $\vdash (bi\ (\neg f)) = (\neg(di\ f))$

by (*meson NotDiEqvBiNot Prop11*)

have 41: $\vdash (\neg((bi\ (\neg f)) \frown skip)) = (empty \vee inf \vee ((di\ f) \frown skip))$

using 40 *NotNotSChopSkip[of LIFT (di f)]* **by** (*metis inteq-reflection*)

have 5: $\vdash (more \wedge \neg((bi\ (\neg f)) \frown skip)) = (more \wedge (empty \vee inf \vee (di\ f) \frown skip))$

using 41 **by** *auto*

have 6: $\vdash (more \wedge (empty \vee inf \vee (di\ f) \frown skip)) =$

$(more \wedge (inf \vee (di\ f) \frown skip))$

unfolding *empty-d-def* **by** *fastforce*

show *?thesis* **unfolding** *finite-d-def* **using** 1 2 3 4 5

by *fastforce*

qed

lemma *DsDi*:

$\vdash ds\ f = (finite \longrightarrow (di\ f) \frown skip)$

proof –

have 1: $\vdash ds\ f = (more \wedge (finite \longrightarrow (di\ f) \frown skip))$ **by** (*rule DsMoreDi*)

have 2: $\vdash (di\ f) \frown skip \longrightarrow more$ **by** (*simp add: SChopSkipImpMore*)

hence 3: $\vdash (more \wedge (finite \longrightarrow (di\ f) \frown skip)) =$

$((more \wedge inf) \vee (more \wedge (di\ f) \frown skip))$

unfolding *finite-d-def* **by** *fastforce*

have 4: $\vdash ((more \wedge inf) \vee (more \wedge (di\ f) \frown skip)) =$

$(inf \vee (di\ f) \frown skip)$

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*:

$\vdash ds\ f = (finite \longrightarrow (df\ f) \frown skip)$

proof –

have 1: $\vdash finite \longrightarrow (di\ f) = (df\ f)$

using *FiniteImpDfEqvDi* **by** *fastforce*

have 2: $\vdash ds\ f = (finite \longrightarrow (di\ f) \frown skip)$

by (simp add: DsDi)
 have 3: $\vdash (finite \longrightarrow (di\ f) \frown skip) = (finite \longrightarrow (df\ f) \frown skip)$
 by (metis 1 2 FiniteImpAnd inteq-reflection schop-d-def)
 show ?thesis
 by (metis 2 3 inteq-reflection)
 qed

lemma *BsEqvNotDsNot*:

$\vdash bs\ f = (\neg(ds\ (\neg\ f)))$

proof –

have 1: $\vdash ds\ (\neg\ f) = (more \wedge (finite \longrightarrow di\ (\neg\ f) \frown skip))$

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 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 BsEqvRule)

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: $\vdash more \wedge bi f \longrightarrow (f \wedge inf) \vee (more \wedge bi f \wedge finite)$
by (*smt (z3) BiElim OrFiniteInf Prop10 intI inteq-reflection unl-lift2*)
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: $\vdash more \wedge bi f \wedge finite \longrightarrow (bi f) \frown skip$
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: $\vdash \neg(\text{bs } (\neg f)) \longrightarrow \text{inf} \vee \neg(\text{bf } (\neg f))$ **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: $\vdash (\text{empty} \vee \text{bi } (\text{bi } f \vee \text{bi } g) \frown \text{skip}) = (\text{empty} \vee (\text{bi } f \vee \text{bi } g) \frown \text{skip})$

by *auto*

have 8: $\vdash (\text{empty} \vee (\text{bi } f \vee \text{bi } g) \frown \text{skip}) = \text{bs}(\text{bi } f \vee \text{bi } g)$

by (*metis BiBiOrEqvBi BsEqvEmptyOrBiSChopSkip inteq-reflection*)

from 1 2 4 8 show ?thesis by (*metis inteq-reflection*)

qed

lemma *BsOrBsEqvBsBfOrBf*:

$\vdash (\text{bs } f \vee \text{bs } g) = \text{bs}(\text{bf } f \vee \text{bf } g)$

proof –

have 1: $\vdash (\text{bs } f \vee \text{bs } g) = \text{bs}(\text{bi } f \vee \text{bi } g)$

by (*simp add: BsOrBsEqvBsBiOrBi*)

have 2: $\vdash (\text{bi } f \vee \text{bi } g) \frown \text{skip} = (\text{bf } f \vee \text{bf } g) \frown \text{skip}$

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*:

$\vdash \text{di } f \vee (\text{ds } f \wedge \text{finite}) = \text{di } f$

proof –

have 1: $\vdash \text{di } f \longrightarrow \text{di } f \vee \text{ds } f$ by *auto*

have 2: $\vdash \text{di } f \longrightarrow \text{di } f$ by *auto*

have 3: $\vdash \text{ds } f \longrightarrow \text{inf} \vee \text{di } f$ by (*rule DsImpDi*)

have 4: $\vdash \text{di } f \vee \text{ds } f \longrightarrow \text{inf} \vee \text{di } f$ using 2 3 by *auto*

from 1 4 show ?thesis unfolding *finite-d-def* by *fastforce*

qed

lemma *DiAndDsEqvDs*:

$\vdash (\text{di } f \wedge (\text{ds } f \wedge \text{finite})) = (\text{ds } f \wedge \text{finite})$

proof –

have 1: $\vdash \text{di } f \wedge \text{ds } f \longrightarrow \text{ds } f$ by *auto*

have 2: $\vdash \text{ds } f \longrightarrow \text{ds } f$ by *auto*

have 3: $\vdash \text{ds } f \longrightarrow \text{inf} \vee \text{di } f$ by (*rule DsImpDi*)

have 4: $\vdash (\text{ds } f \wedge \text{finite}) \longrightarrow \text{di } f \wedge (\text{ds } f \wedge \text{finite})$

using 2 3 unfolding *finite-d-def* by *auto*

from 1 4 show ?thesis by *auto*

qed

lemma *DiFiniteOrInf*:

$\vdash \text{di } f = ((f \wedge \text{finite}) \vee (\text{di } f) \frown \text{skip} \vee (\text{di } f \wedge \text{inf}))$

proof –

have 1: $\vdash (\text{di } f) = ((\text{di } f \wedge \text{finite}) \vee (\text{di } f \wedge \text{inf}))$

by (*simp add: OrFiniteInf*)

have 2: $\vdash (\text{di } f \wedge \text{finite}) = (\text{di } f) \frown \text{finite}$

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*)

$\text{inteq-reflection sometimes-d-def}$
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: $\vdash (di\ f) \frown \text{finite} = (di\ f) \frown (\# \text{True} \frown \text{skip} \vee \text{empty})$
by (*simp add: 4 RightSChopEqvSChop*)
have 6: $\vdash (di\ f) \frown (\# \text{True} \frown \text{skip}) = (di\ f) \frown \text{skip}$
unfolding di-d-def
by (*metis* 2 3 *ChopAssoc FmoreEqvSkipChopFinite SkipFiniteEqvFiniteSkip di-d-def fmore-d-def*
inteq-reflection schop-d-def)
have 7: $\vdash (di\ f) \frown \text{finite} = (f \frown \text{finite}) \frown \text{finite}$
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: $\vdash f \frown (\# \text{True} \frown \text{skip}) = (di\ f) \frown \text{skip}$
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: $\vdash (di\ f \wedge \text{finite}) = ((f \wedge \text{finite}) \vee (di\ f) \frown \text{skip})$
by (*metis* 10 11 2 3 7 8 *FmoreEqvSkipChopFinite SChopOrEqvRule WPowerstarEqv*
WPowerstar-skip-finite fmore-d-def int-eq)
have 14: $\vdash ((di\ f \wedge \text{finite}) \vee (di\ f \wedge \text{inf})) =$
 $((f \wedge \text{finite}) \vee (di\ f) \frown \text{skip} \vee (di\ f \wedge \text{inf}))$
using 13 **by** *fastforce*
show ?thesis **using** 1 14 **by** *fastforce*
qed

lemma *DiFiniteEqv*:

$\vdash (di\ f \wedge \text{finite}) = ((f \wedge \text{finite}) \vee (di\ f) \frown \text{skip})$

proof –

have 1: $\vdash (di\ f \wedge \text{finite}) =$

$((f \wedge \text{finite}) \vee (di\ f) \frown \text{skip} \vee (di\ f \wedge \text{inf})) \wedge \text{finite}$

using *DiFiniteOrInf* **by** (*metis int-simps(1) inteq-reflection*)

have 2: $\vdash (((f \wedge \text{finite}) \vee (di\ f) \frown \text{skip} \vee (di\ f \wedge \text{inf})) \wedge \text{finite}) =$

$((f \wedge \text{finite}) \vee ((di\ f) \frown \text{skip} \wedge \text{finite}))$

by (*metis* (*no-types, opaque-lifting*) *ChopAndFiniteDist ChopEmpty DiEqvOrDiChopSkipB*

DiFiniteOrInf OrSChopEqv SChopAssoc inteq-reflection itl-def(9))

have 3: $\vdash ((di\ f) \frown \text{skip} \wedge \text{finite}) = ((di\ f) \frown \text{skip})$

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: $\vdash ((finite \longrightarrow di\ (f \wedge di\ g) \frown skip) \vee (finite \longrightarrow di\ (g \wedge di\ f) \frown skip)) =$

$(ds(f \wedge di\ g) \vee ds(g \wedge di\ f))$
using 9 10 **by** fastforce
have 12: $\vdash ((finite \longrightarrow di(f \wedge di\ g) \frown skip) \vee (finite \longrightarrow di(g \wedge di\ f) \frown skip)) =$
 $(finite \longrightarrow di(f \wedge di\ g) \frown skip \vee di(g \wedge di\ f) \frown skip)$
by fastforce
from 4 5 8 11 12 81 **show** ?thesis **by** (metis inteq-reflection)
qed

lemma *SChopFiniteEqvSChopTrueAndFinite*:

$\vdash g \frown finite = (g \frown \#True \wedge finite)$
by (meson FiniteImportSChopRight Prop04 RightSChopEqvSChop int-simps(17) lift-and-com)

lemma *BsEqvBiMoreImpChop*:

$\vdash (bs\ f) = (bi(more \wedge finite \longrightarrow f \frown skip) \wedge finite)$

proof –

have 1: $\vdash (bs\ f \vee inf) = (empty \vee inf \vee (bi\ f \frown skip))$
unfolding bs-d-def **by** fastforce
have 2: $\vdash (empty \vee inf \vee (bi\ f \frown skip)) = ((\neg(\neg(bi\ f)) \frown skip))$
using NotNotSChopSkip **by** fastforce
have 3: $\vdash \neg(\neg(bi\ f)) \frown skip = (\neg(di\ (\neg f) \frown skip))$
by (simp add: bi-d-def)
have 4: $\vdash (\neg(di\ (\neg f) \frown skip)) = (\neg(((\neg f) \frown \#True) \frown skip))$
by (metis FiniteImpAnd FiniteImpDfEqvDi LeftChopImpChop Prop11 itl-def(15) lift-imp-neg schop-d-def)
have 5: $\vdash (\neg(((\neg f) \frown \#True) \frown skip)) = (\neg((\neg f) \frown (\#True \frown skip)))$
by (meson Prop11 SChopAssoc lift-imp-neg)
have 05: $\vdash \#True \frown skip = skip \frown finite$
by (metis DiamondSChopdef Prop10 SkipFiniteEqvFiniteSkip WPowerstar-ext WPowerstar-skip-finite
inteq-reflection itl-def(10) itl-def(9))
have 6: $\vdash (\neg((\neg f) \frown (\#True \frown skip))) = (\neg((\neg f) \frown (skip \frown finite)))$
by (metis 05 5 inteq-reflection)
have 7: $\vdash (\neg((\neg f) \frown (skip \frown finite))) = (\neg(((\neg f) \frown skip) \frown finite))$
using SChopAssoc **by** fastforce
have 07: $\vdash (\neg(((\neg f) \frown skip) \frown finite)) = ((\neg(((\neg f) \frown skip) \frown \#True)) \vee inf)$
using InfEqvNotFinite SChopFiniteEqvSChopTrueAndFinite **by** fastforce
have 8: $\vdash (\neg(((\neg f) \frown skip) \frown finite)) = ((\neg(di\ ((\neg f) \frown skip))) \vee inf)$
by (metis 07 FiniteImportSChopRight Prop10 WPowerstar-ext WPowerstar-skip-finite di-d-def
inteq-reflection lift-and-com schop-d-def)
have 9: $\vdash (\neg(di\ ((\neg f) \frown skip))) = bi\ (\neg((\neg f) \frown skip))$
using NotDiEqvBiNot **by** blast
have 10: $\vdash bi\ (\neg((\neg f) \frown skip)) = bi(empty \vee inf \vee (f \frown skip))$
using NotNotSChopSkip **using** BiEqvBi **by** blast
have 010: $\vdash (empty \vee inf \vee (f \frown skip)) = (\neg (more \wedge finite) \vee (f \frown skip))$
unfolding empty-d-def finite-d-def **by** fastforce
have 11: $\vdash bi(empty \vee inf \vee (f \frown skip)) = bi(\neg (more \wedge finite) \vee (f \frown skip))$
by (simp add: 010 BiEqvBi)
have 12: $\vdash (\neg (more \wedge finite) \vee (f \frown skip)) = (more \wedge finite \longrightarrow f \frown skip)$ **by** auto
have 13: $\vdash bi(\neg (more \wedge finite) \vee (f \frown skip)) = bi(more \wedge finite \longrightarrow f \frown skip)$
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: $\vdash (\neg (((\neg f) \frown \text{skip}) \frown \text{finite})) = (\text{bi}(\text{more} \wedge \text{finite} \longrightarrow f \frown \text{skip}) \vee \text{inf})$
using 8 9 10 11 13 by (*metis inteq-reflection*)
have 16: $\vdash (\text{bs } f \vee \text{inf}) = (\text{bi}(\text{more} \wedge \text{finite} \longrightarrow f \frown \text{skip}) \vee \text{inf})$
using 14 15 by *fastforce*
have 17: $\vdash \text{bs } f \longrightarrow \text{finite}$
by (*metis AndChopB EmptyImpFinite FiniteChopSkipImpFinite Prop02 bs-d-def lift-imp-trans schop-d-def*)
have 18: $\vdash ((\text{bs } f \vee \text{inf}) \wedge \text{finite}) = \text{bs } f$
using 17 unfolding *finite-d-def* **by** *fastforce*
have 19: $\vdash ((\text{bi}(\text{more} \wedge \text{finite} \longrightarrow f \frown \text{skip}) \vee \text{inf}) \wedge \text{finite}) =$
 $(\text{bi}(\text{more} \wedge \text{finite} \longrightarrow f \frown \text{skip}) \wedge \text{finite})$
unfolding *finite-d-def* **by** *fastforce*
have 20: $\vdash ((\text{bs } f \vee \text{inf}) \wedge \text{finite}) = ((\text{bi}(\text{more} \wedge \text{finite} \longrightarrow f \frown \text{skip}) \vee \text{inf}) \wedge \text{finite})$
using 16 by *fastforce*
show ?thesis using 16 18 19 20 by (*metis inteq-reflection*)
qed

lemma NotSChopInf:

$\vdash (\neg (f \frown \text{inf})) = (\text{finite} \vee \neg(f \frown \# \text{True}))$
proof –
have 1: $\vdash (\neg (f \frown \text{inf})) = (\text{finite} \vee (\text{bf } (\neg f)))$
by (*metis NotNotSChopInf int-simps(4) inteq-reflection*)
have 2: $\vdash \text{bf } (\neg f) = (\neg (f \frown \# \text{True}))$
unfolding *bf-d-def df-d-def* **by** *auto*
show ?thesis using 1 2 Prop06 by *blast*
qed

lemma BoxMoreStateEqvBsFinState:

$\vdash (\Box(\text{more} \longrightarrow \neg(\text{init } w)) \wedge \text{finite}) = \text{bs}(\neg(\text{fin}(\text{init } w)))$
proof –
have 1: $\vdash \Box(\text{more} \longrightarrow \neg(\text{init } w)) = (\neg(\Diamond(\neg(\text{more} \longrightarrow \neg(\text{init } w)))))$
by (*simp add: always-d-def*)
have 01: $\vdash (\neg(\text{more} \longrightarrow \neg(\text{init } w))) = (\text{init } w \wedge \text{more})$ **by** *auto*
hence 2: $\vdash \neg(\Diamond(\neg(\text{more} \longrightarrow \neg(\text{init } w)))) = (\neg(\# \text{True} \frown (\text{init } w \wedge \text{more})))$
by (*metis DiamondSChopdef TrueW int-simps(3) int-simps(6) inteq-reflection*)
have 3: $\vdash \text{more} = (\# \text{True} \frown \text{skip} \vee \text{inf})$
by (*metis FmoreEqvSkipChopFinite MoreAndInfEqvInf OrFiniteInf SkipFiniteEqvFiniteSkip fmore-d-def int-simps(17) inteq-reflection schop-d-def*)
have 4: $\vdash (\text{init } w \wedge \text{more}) = (\text{init } w \wedge ((\# \text{True} \frown \text{skip} \vee \text{inf})))$
using 3 by *auto*
have 5: $\vdash (\text{init } w \wedge (\# \text{True} \frown \text{skip} \vee \text{inf})) =$
 $((\text{init } w \wedge \text{empty}) \frown (\# \text{True} \frown \text{skip}) \vee (\text{init } w \wedge \text{empty}) \frown \text{inf})$
by (*meson Prop04 SChopOrEqv StateAndEmptySChop*)
have 6: $\vdash (\text{init } w \wedge \text{more}) = ((\text{init } w \wedge \text{empty}) \frown (\# \text{True} \frown \text{skip}) \vee (\text{init } w \wedge \text{empty}) \frown \text{inf})$
using 4 5 by *fastforce*
have 7: $\vdash (\# \text{True} \frown (\text{init } w \wedge \text{more})) =$
 $(\# \text{True} \frown ((\text{init } w \wedge \text{empty}) \frown (\# \text{True} \frown \text{skip}) \vee (\text{init } w \wedge \text{empty}) \frown \text{inf}))$
using 6 by (*simp add: RightSChopEqvSChop*)
have 8: $\vdash (\# \text{True} \frown ((\text{init } w \wedge \text{empty}) \frown (\# \text{True} \frown \text{skip}))) =$
 $((\# \text{True} \frown (\text{init } w \wedge \text{empty})) \frown (\# \text{True} \frown \text{skip}))$
using *SChopAssoc* **by** *blast*

have 9: $\vdash (((\#True \wedge (init\ w \wedge empty)) \wedge (\#True \wedge skip)) \wedge$
 $(((((\#True \wedge (init\ w \wedge empty)) \wedge \#True) \wedge skip) \wedge$
using *SChopAssoc* **by** *blast*
have 10: $\vdash (\#True \wedge (init\ w \wedge more)) =$
 $(((((\#True \wedge (init\ w \wedge empty)) \wedge \#True) \wedge skip) \vee \#True \wedge ((init\ w \wedge empty) \wedge inf)) \wedge$
using *7 8 9 by (metis SChopOrEqv inteq-reflection)*
hence 11: $\vdash (\neg(\#True \wedge (init\ w \wedge more))) =$
 $(\neg(((\#True \wedge (init\ w \wedge empty)) \wedge \#True) \wedge skip) \vee \#True \wedge ((init\ w \wedge empty) \wedge inf)) \wedge$
by *auto*
have 011: $\vdash (\neg(((\#True \wedge (init\ w \wedge empty)) \wedge \#True) \wedge skip) \vee \#True \wedge ((init\ w \wedge empty) \wedge inf)) \wedge$
 $(\neg(((\#True \wedge (init\ w \wedge empty)) \wedge \#True) \wedge skip) \wedge \neg(\#True \wedge ((init\ w \wedge empty) \wedge inf)))$
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: $\vdash (\neg(((\#True \wedge (init\ w \wedge empty)) \wedge \#True) \wedge skip) \vee \#True \wedge ((init\ w \wedge empty) \wedge inf)) \wedge$
 $((empty \vee inf \vee (bi(\Box (\neg(init\ w \wedge empty)))) \wedge skip) \wedge (finite \vee bi(\Box (\neg(init\ w \wedge empty)))))$
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: $\vdash (empty \vee inf \vee (bi(\Box (\neg(init\ w \wedge empty)))) \wedge skip) =$

$(\text{empty} \vee \text{inf} \vee (\text{bi}(\Box (\neg(\text{init } w) \vee \neg \text{empty})) \frown \text{skip}))$
by *auto*
have 175: $\vdash (\text{finite} \vee \text{bi}(\Box (\neg(\text{init } w \wedge \text{empty})))) =$
 $(\text{finite} \vee \text{bi}(\Box (\neg(\text{init } w) \vee \neg \text{empty})))$
using 173 by *fastforce*
have 181: $\vdash (\neg(\text{init } w) \vee \neg \text{empty}) = (\neg \text{empty} \vee \neg(\text{init } w))$
by *auto*
hence 18: $\vdash \Box (\neg(\text{init } w) \vee \neg \text{empty}) = \Box (\neg \text{empty} \vee \neg(\text{init } w))$
by (*simp add: BoxEqvBox*)
have 191: $\vdash (\neg \text{empty} \vee \neg(\text{init } w)) = (\text{empty} \longrightarrow \neg(\text{init } w))$
by *auto*
hence 19: $\vdash \Box (\neg \text{empty} \vee \neg(\text{init } w)) = \Box(\text{empty} \longrightarrow \neg(\text{init } w))$
by (*simp add: BoxEqvBox*)
have 20: $\vdash \Box(\text{empty} \longrightarrow \neg(\text{init } w)) = \text{fin } (\neg(\text{init } w))$
by (*simp add: fin-d-def*)
have 21: $\vdash (\text{fin } (\neg(\text{init } w)) \wedge \text{finite}) = ((\neg(\text{fin } (\text{init } w))) \wedge \text{finite})$
by (*metis FinNotStateEqvNotFinState Initprop(2) inteq-reflection*)
have 22: $\vdash (\text{bi}(\Box (\neg(\text{init } w) \vee \neg \text{empty})) \wedge \text{finite}) = (\text{bi } (\neg(\text{fin } (\text{init } w)))) \wedge \text{finite}$
by (*metis 181 191 20 21 BiAndEqv BiFiniteEqvFinite inteq-reflection*)
hence 23: $\vdash (\text{bi}(\Box (\neg(\text{init } w) \vee \neg \text{empty}))) \frown \text{skip} = (\text{bi } (\neg(\text{fin } (\text{init } w)))) \frown \text{skip}$
by (*simp add: LeftChopEqvChop schop-d-def*)
hence 24: $\vdash (\text{empty} \vee \text{inf} \vee (\text{bi}(\Box (\neg(\text{init } w) \vee \neg \text{empty}))) \frown \text{skip}) =$
 $(\text{empty} \vee \text{inf} \vee (\text{bi } (\neg(\text{fin } (\text{init } w)))) \frown \text{skip})$
by *auto*
have 241: $\vdash (\text{bi } (\neg(\text{fin } (\text{init } w)))) \frown \text{skip} = \text{bf } (\neg \text{fin } (\text{init } w)) \frown \text{skip}$
by (*metis (no-types, opaque-lifting) EmptyChop FiniteImpAnd FiniteImpBfEqvBi LeftChopEqvChop Prop04 schop-d-def*)
hence 25: $\vdash (\text{empty} \vee \text{inf} \vee (\text{bi } (\neg(\text{fin } (\text{init } w)))) \frown \text{skip}) = (\text{inf} \vee \text{bs}(\neg(\text{fin } (\text{init } w))))$
unfolding *bs-d-def* **by** *auto*
have 26: $\vdash (\Box(\text{more} \longrightarrow \neg(\text{init } w)) \wedge \text{finite}) =$
 $((\text{empty} \vee \text{inf} \vee (\text{bi}(\Box (\neg(\text{init } w \wedge \text{empty}))) \frown \text{skip})) \wedge (\text{finite} \vee \text{bi}(\Box (\neg(\text{init } w \wedge \text{empty}))))$
 $\wedge \text{finite})$
by (*metis (no-types, opaque-lifting) 01 10 16 TrueSChopEqvDiamond always-d-def int-simps(4) inteq-reflection*)
have 27: $\vdash (((\text{empty} \vee \text{inf} \vee (\text{bi}(\Box (\neg(\text{init } w \wedge \text{empty}))) \frown \text{skip})) \wedge (\text{finite} \vee \text{bi}(\Box (\neg(\text{init } w \wedge \text{empty}))))$
 $\wedge \text{finite}) =$
 $((\text{empty} \vee \text{inf} \vee (\text{bi}(\Box (\neg(\text{init } w \wedge \text{empty}))) \frown \text{skip})) \wedge \text{finite})$
by *fastforce*
have 28: $\vdash (\text{bi}(\Box (\neg(\text{init } w \wedge \text{empty}))) \frown \text{skip}) \longrightarrow \text{finite}$
by (*metis LenOneEqvSkip SChopImpFinite inteq-reflection len-k-finite*)
have 29: $\vdash \text{empty} \longrightarrow \text{finite}$
by (*simp add: EmptyImpFinite*)
have 29: $\vdash (((\text{empty} \vee \text{inf} \vee (\text{bi}(\Box (\neg(\text{init } w \wedge \text{empty}))) \frown \text{skip})) \wedge \text{finite}) =$
 $((\text{empty} \vee (\text{bi}(\Box (\neg(\text{init } w \wedge \text{empty}))) \frown \text{skip})) \wedge \text{finite})$
using 28 29 unfolding *finite-d-def* **by** *fastforce*
have 30: $\vdash (((\text{empty} \vee (\text{bi}(\Box (\neg(\text{init } w \wedge \text{empty}))) \frown \text{skip})) \wedge \text{finite}) = \text{bs}(\neg(\text{fin } (\text{init } w))))$
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

15.5.4 First occurrence

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*:

```

 $\vdash \triangleright \#True = empty$ 
proof -
have 1:  $\vdash \triangleright \#True = (\#True \wedge bs(\neg \#True))$ 
by (simp add: first-d-def)
have 2:  $\vdash bs(\neg \#True) = (empty \vee (bi(\neg \#True)) \frown skip)$ 
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:  $\vdash bs(\neg \#True) = empty$ 

```

using 2 4 by fastforce
 from 1 5 show ?thesis by fastforce
 qed

lemma FstFalse:

⊢ $\neg(\triangleright \#False)$

proof –

have 1: $\triangleright \#False = (\#False \wedge bs \#True)$ by (simp add: first-d-def)

from 1 show ?thesis by auto

qed

lemma FstChopFalseEqvFalse:

⊢ $\neg(\triangleright f \frown \#False)$

by (simp add: Valid-def itl-defs first-d-def bs-d-def)

lemma FstEmpty:

⊢ $\triangleright empty = empty$

proof –

have 1: $\triangleright empty = (empty \wedge bs (\neg empty))$ by (simp add: first-d-def)

have 2: $\triangleright bs (\neg empty) = (empty \vee bi (\neg empty) \frown skip)$ by (simp add: bs-d-def)

from 1 2 show ?thesis by fastforce

qed

lemma FstAndEmptyEqvAndEmpty:

⊢ $(\triangleright f \wedge empty) = (f \wedge empty)$

proof –

have 1: $\triangleright f \wedge empty = ((f \wedge bs (\neg f)) \wedge empty)$ by (simp add: first-d-def)

have 2: $\triangleright bs (\neg f) = (empty \vee bi (\neg f) \frown skip)$ by (simp add: bs-d-def)

from 1 2 show ?thesis by fastforce

qed

lemma FstEmptyOrEqvEmpty:

⊢ $\triangleright (empty \vee f) = empty$

proof –

have 1: $\triangleright (empty \vee f) = ((\triangleright empty \wedge bs (\neg f)) \vee (\triangleright f \wedge bs (\neg empty)))$ using FstWithOrEqv by blast

have 2: $\triangleright (\neg empty) = more$ by (simp add: empty-d-def)

hence 3: $\triangleright bs (\neg empty) = bs more$ using BsEqvRule by blast

have 4: $\triangleright bs more = empty$ using BsMoreEqvEmpty by blast

have 5: $\triangleright (\triangleright f \wedge bs (\neg empty)) = (\triangleright f \wedge empty)$ using 3 4 by fastforce

have 6: $\triangleright \triangleright empty = empty$ using FstEmpty by blast

hence 7: $\triangleright (\triangleright empty \wedge bs (\neg f)) = (empty \wedge bs (\neg f))$ by auto

have 8: $\triangleright (empty \wedge bs (\neg f)) = (empty \wedge (empty \vee bi (\neg f) \frown skip))$ by (simp add: bs-d-def)

have 9: $\triangleright (empty \wedge (empty \vee bi (\neg f) \frown skip)) = empty$ by auto

have 10: $\triangleright (empty \wedge bs (\neg f)) = empty$ using 8 9 by auto

have 11: $\triangleright ((\triangleright empty \wedge bs (\neg f)) \vee (\triangleright f \wedge bs (\neg empty))) =$

$(empty \vee (\triangleright f \wedge empty))$ using 7 10 5 by fastforce

have 12: $\triangleright (empty \vee (\triangleright f \wedge empty)) = empty$ 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: $\vdash \triangleright \text{more} = (\text{more} \wedge \text{bs } (\neg \text{more}))$ **by** (*simp add: first-d-def*)

have 2: $\vdash (\text{more} \wedge \text{bs } (\neg \text{more})) = (\text{more} \wedge (\text{empty} \vee \text{bi } (\neg \text{more}) \frown \text{skip}))$ **by** (*simp add: bs-d-def*)

have 3: $\vdash (\text{more} \wedge (\text{empty} \vee \text{bi } (\neg \text{more}) \frown \text{skip})) = (\text{more} \wedge \text{bi } (\neg \text{more}) \frown \text{skip})$

unfolding *empty-d-def* **by** *fastforce*

have 4: $\vdash (\text{more} \wedge ((\text{bi } (\neg \text{more})) \frown \text{skip})) = ((\text{bi } (\neg \text{more})) \frown \text{skip})$

using *SChopSkipImpMore* **by** *fastforce*

have 5: $\vdash ((\text{bi } (\neg \text{more})) \frown \text{skip}) = \text{bi } \text{empty} \frown \text{skip}$ **by** (*simp add: empty-d-def*)

have 6: $\vdash \text{bi } \text{empty} = \text{empty}$ **using** *BiEmptyEqvEmpty* **by** *auto*

hence 7: $\vdash \text{bi } \text{empty} \frown \text{skip} = \text{empty} \frown \text{skip}$

using *LeftSChopEqvSChop* **by** *blast*

have 8: $\vdash \text{empty} \frown \text{skip} = \text{skip}$ **using** *EmptySChop* **by** *blast*

from 1 2 3 4 5 7 8 **show** *?thesis* **by** (*metis int-eq*)

qed

lemma *FstEqvBsNotAndDi*:

$\vdash \triangleright f = (\text{bs } (\neg f) \wedge \text{di } f)$

proof –

have 1: $\vdash \text{bs } (\neg f) = (\neg(\text{ds } f))$ **by** (*simp add: ds-d-def*)

hence 2: $\vdash (\text{bs } (\neg f) \wedge \text{di } f) = (\neg(\text{ds } f) \wedge \text{di } f)$ **by** *auto*

have 3: $\vdash (\text{di } f \wedge \text{finite}) = ((\text{ds } f \wedge \text{finite}) \vee (f \wedge \text{finite}))$

using *OrDsEqvDi* **by** *fastforce*

have 03: $\vdash (\neg(\text{ds } f)) \longrightarrow \text{finite}$

by (*metis BsEqvBiMoreImpChop Prop11 Prop12 ds-d-def int-simps(4) lift-imp-trans*)

have 4: $\vdash (\neg(\text{ds } f) \wedge \text{di } f) = (\neg(\text{ds } f) \wedge ((\text{ds } f \wedge \text{finite}) \vee (f \wedge \text{finite})))$

using 03 3 **by** *fastforce*

have 5: $\vdash (\neg(\text{ds } f) \wedge ((\text{ds } f \wedge \text{finite}) \vee (f \wedge \text{finite}))) =$

$(\neg(\text{ds } f) \wedge f \wedge \text{finite})$ **by** *auto*

have 6: $\vdash (\neg(\text{ds } f) \wedge f \wedge \text{finite}) = (f \wedge \text{bs } (\neg f))$ **using** 1

using 03 **by** *fastforce*

from 2 4 5 6 **show** *?thesis* **by** (*metis first-d-def int-eq*)

qed

lemma *FstOrDiEqvDi*:

$\vdash (\triangleright f \vee \text{di } f) = \text{di } f$

proof –

have 1: $\vdash (\triangleright f \vee \text{di } f) = ((f \wedge \text{bs } (\neg f)) \vee \text{di } f)$ **by** (*simp add: first-d-def*)

have 2: $\vdash ((f \wedge \text{bs } (\neg f)) \vee \text{di } f) = ((f \vee \text{di } f) \wedge (\text{bs } (\neg f) \vee \text{di } f))$ **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*:

$\vdash finite \longrightarrow di\ (f) = di\ (\triangleright f)$
proof –
have 1: $\vdash di\ (\triangleright f) = di\ (f \wedge bs\ (\neg f))$
by (*simp add: first-d-def*)
have 2: $\vdash di\ (f \wedge bs\ (\neg f)) \longrightarrow di\ f \wedge di\ (bs\ (\neg f))$
using *DiAndImpAnd* **by** *auto*
hence 3: $\vdash di\ (f \wedge bs\ (\neg f)) \longrightarrow di\ f$
by *auto*
have 03: $\vdash di\ (f \wedge bs\ (\neg f)) \longrightarrow di\ (f)$
by (*metis AndDiEqv DiImpDi Prop11 Prop12 inteq-reflection*)
have 4: $\vdash di\ (\triangleright f) \longrightarrow di\ (f)$ **using** 1 03

by *fastforce*
 have 5: $\vdash (di\ f) \wedge empty = (f \wedge empty)$
 by (*simp add: DiAndEmptyEqvAndEmpty*)
 have 6: $\vdash (\triangleright f \wedge empty) = (f \wedge empty)$
 using *FstAndEmptyEqvAndEmpty* by *auto*
 have 7: $\vdash di\ (f) \wedge empty \longrightarrow \triangleright f$
 using 5 6 by *fastforce*
 have 8: $\vdash \triangleright f \longrightarrow di\ (\triangleright f)$
 using *DiIntro* by *auto*
 have 9: $\vdash di\ (f) \wedge empty \longrightarrow di\ (\triangleright f)$
 using 7 8 using *lift-imp-trans* by *blast*
 hence 10: $\vdash empty \longrightarrow (di\ (f) \longrightarrow di\ (\triangleright f))$
 by *auto*

 have 11: $\vdash (di\ (f) \longrightarrow di\ (\triangleright f)) \frown skip \longrightarrow more \wedge finite$
 by (*metis FstImpFinite FstMoreEqvSkip Prop12 SChopImpFinite SChopSkipImpMore inteq-reflection*)
 have 12: $\vdash more \wedge finite \longrightarrow$
 $((di\ (f) \longrightarrow di\ (\triangleright f)) \frown skip =$
 $((di\ (f)) \frown skip \longrightarrow (di\ (\triangleright f)) \frown skip))$
 by (*simp add: MoreImpImpSChopSkipEqv*)
 have 13: $\vdash (more \wedge finite \wedge (di\ (f) \longrightarrow di\ (\triangleright f)) \frown skip) =$
 $(more \wedge finite \wedge ((di\ (f)) \frown skip \longrightarrow (di\ (\triangleright f)) \frown skip))$
 using 12 by *fastforce*
 have 14: $\vdash (di\ (f) \longrightarrow di\ (\triangleright f)) \frown skip =$
 $(more \wedge finite \wedge ((di\ (f)) \frown skip \longrightarrow (di\ (\triangleright f)) \frown skip))$
 using 11 13 by *fastforce*
 have 15: $\vdash (di\ (f) \wedge finite) = ((f \wedge finite) \vee (ds\ f \wedge finite))$
 using *OrDsEqvDi* by (*metis (no-types, opaque-lifting) int-eq lift-and-com*)
 have 16: $\vdash di\ (f) = (di\ (f) \wedge (bs\ (\neg f) \vee \neg(bs\ (\neg f))))$
 by *auto*
 have 17: $\vdash (di\ (f) \wedge (bs\ (\neg f) \vee \neg(bs\ (\neg f)))) =$
 $((di\ (f) \wedge bs\ (\neg f)) \vee (di\ (f) \wedge \neg(bs\ (\neg f))))$
 by *auto*
 have 171: $\vdash bs\ (\neg f) \longrightarrow finite$
 using *BsEqvBiMoreImpChop* by *fastforce*
 have 18: $\vdash (di\ (f) \wedge bs\ (\neg f)) = ((f \vee ds\ f) \wedge bs\ (\neg f))$
 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: $\vdash (di (f) \wedge bs (\neg f) \wedge ((di (f) \longrightarrow di (\triangleright f)) \neg skip)) \longrightarrow di (\triangleright f)$
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: $\vdash (di (f) \wedge \neg(bs (\neg f)) \wedge ((di (f) \longrightarrow di (\triangleright f)) \neg skip)) =$
 $(di (f) \wedge ds f \wedge finite \wedge ((di (f) \longrightarrow di (\triangleright f)) \neg skip))$
using 11 **by** *fastforce*
have 030: $\vdash finite \longrightarrow (di f) \neg skip = (di f); skip$
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: $\vdash finite \longrightarrow (di f) \neg skip = (di(f \wedge finite)); skip$
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: $\vdash ds f = (finite \longrightarrow (di (f)) \neg skip)$
by (*simp add: DsDi*)
hence 31: $\vdash (di (f) \wedge ds f \wedge finite \wedge ((di (f) \longrightarrow di (\triangleright f)) \neg skip)) =$
 $(di (f) \wedge finite \wedge ((di (f)) \neg skip) \wedge ((di (f) \longrightarrow di (\triangleright f)) \neg skip))$
by *auto*
have 32: $\vdash ((di (f) \longrightarrow di (\triangleright f)) \neg skip \longrightarrow ((di (f)) \neg skip \longrightarrow (di (\triangleright f)) \neg skip))$
using 14 **by** *auto*
hence 33: $\vdash di (f) \wedge finite \wedge ((di (f)) \neg skip) \wedge ((di (f) \longrightarrow di (\triangleright f)) \neg skip \longrightarrow$
 $di (f) \wedge finite \wedge ((di (f)) \neg skip) \wedge ((di (f)) \neg skip \longrightarrow (di (\triangleright f)) \neg skip))$
by *auto*
have 34: $\vdash di (f) \wedge finite \wedge ((di (f)) \neg skip) \wedge$
 $((di (f)) \neg skip \longrightarrow (di (\triangleright f)) \neg skip) \longrightarrow (di (\triangleright f)) \neg skip$
by *fastforce*
have 36: $\vdash (di (\triangleright f)) \neg skip \longrightarrow di(di (\triangleright f))$
by (*metis AndChopA ChopImpDi lift-imp-trans chop-d-def*)
have 37: $\vdash di(di (\triangleright f)) = di (\triangleright f)$
using *DiEqvDiDi* **by** *fastforce*
have 38: $\vdash di (f) \wedge finite \wedge (di f) \neg skip \wedge ((di (f)) \neg skip \longrightarrow (di (\triangleright f)) \neg skip) \longrightarrow di (\triangleright f)$
using 37 36
by (*metis 34 inteq-reflection lift-imp-trans*)
have 39: $\vdash di (f) \wedge finite \wedge \neg(bs (\neg f)) \wedge ((di (f) \longrightarrow di (\triangleright f)) \neg skip) \longrightarrow di (\triangleright f)$
using 29 31 33 38 36 37 **by** *fastforce*
hence 40: $\vdash \neg(bs (\neg f)) \wedge finite \wedge ((di (f) \longrightarrow di (\triangleright f)) \neg skip) \longrightarrow (di (f) \longrightarrow di (\triangleright f))$
by *fastforce*
have 41: $\vdash bs (\neg f) \wedge ((di (f) \longrightarrow di (\triangleright f)) \neg skip) \longrightarrow (di (f) \longrightarrow di (\triangleright f))$
using 27 **by** *fastforce*
have 42: $\vdash (\neg(bs (\neg f)) \vee bs (\neg f)) \wedge ((di (f) \longrightarrow di (\triangleright f)) \neg skip) \longrightarrow (di (f) \longrightarrow di (\triangleright f))$
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: $\vdash bs (\neg (\triangleright f)) = (empty \vee bi (\neg (\triangleright f)) \frown skip)$ by (simp add: bs-d-def)

have 2: $\vdash finite \longrightarrow bi (\neg (\triangleright f)) = bi (\neg f)$ using BiNotFstEqvBiNot by fastforce

hence 3: $\vdash bi (\neg (\triangleright f)) \frown skip = bi (\neg f) \frown skip$ using LeftSChopEqvSChop

by (simp add: FiniteImpAnd LeftChopEqvChop schop-d-def)

hence 4: $\vdash (empty \vee bi (\neg (\triangleright f)) \frown skip) = (empty \vee bi (\neg f) \frown skip)$ 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: $\vdash empty \wedge init w \longrightarrow empty \vee bi (\neg (init w)) \frown skip$ 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: $\vdash (\text{empty} \wedge (\text{init } w) \wedge \text{empty}) = (\text{empty} \wedge (\text{init } w))$
by *auto*
have 6: $\vdash (\text{empty} \wedge (\text{init } w)) = \triangleright(\text{init } w)$
using *FstState* **by** *fastforce*
from 1 2 4 5 6 **show** *?thesis* **by** *fastforce*
qed

lemma *FstStateImpFstStateOr*:

$\vdash \triangleright(\text{init } w) \longrightarrow \triangleright(\text{init } w \vee f)$

proof –

have 1: $\vdash \triangleright(\text{init } w) = (\text{empty} \wedge \text{init } w)$

using *FstState* **by** *blast*

have 2: $\vdash (\text{empty} \wedge \text{init } w) = (\text{empty} \wedge (\text{empty} \vee \text{bi } (\neg f) \neg \text{skip}) \wedge \text{init } w)$

by *auto*

have 3: $\vdash (\text{empty} \wedge (\text{empty} \vee \text{bi } (\neg f) \neg \text{skip}) \wedge \text{init } w) =$
 $(\text{empty} \wedge \text{bs } (\neg f) \wedge \text{init } w)$

by (*simp add: bs-d-def*)

have 4: $\vdash (\text{empty} \wedge \text{bs } (\neg f) \wedge \text{init } w) = (\text{empty} \wedge \text{init } w \wedge \text{bs } (\neg f))$

by *auto*

have 5: $\vdash (\text{empty} \wedge \text{init } w) = \triangleright(\text{init } w)$

using *FstState* **by** *fastforce*

hence 6: $\vdash (\text{empty} \wedge \text{init } w \wedge \text{bs } (\neg f)) = (\triangleright(\text{init } w) \wedge \text{bs } (\neg f))$

by *auto*

have 7: $\vdash \triangleright(\text{init } w) \wedge \text{bs } (\neg f) \longrightarrow (\triangleright(\text{init } w) \wedge \text{bs } (\neg f)) \vee (\triangleright f \wedge \text{bs } (\neg(\text{init } w)))$

by *auto*

have 8: $\vdash \triangleright(\text{init } w \vee f) = ((\triangleright(\text{init } w) \wedge \text{bs } (\neg f)) \vee (\triangleright f \wedge \text{bs } (\neg(\text{init } w))))$

using *FstWithOrEqv* **by** *blast*

show *?thesis* **using** 1 3 8 **by** *fastforce*

qed

lemma *FstLenSame*:

$(\forall \sigma. (\sigma \models \text{di } (\triangleright f \wedge \text{len}(i)) \wedge \text{di } (\triangleright f \wedge \text{len}(j))) \longrightarrow (i=j))$

by (*simp add: DiLenFstsem FstLenSamesem*)

lemma *FstLenSame-1*:

$\vdash \text{di } (\triangleright f \wedge \text{len}(i)) \wedge \text{di } (\triangleright f \wedge \text{len}(j)) \longrightarrow (\#i=\#j)$

using *FstLenSame Valid-def* **by** *fastforce*

lemma *FstAndLenSame*:

$(\forall \sigma. (\sigma \models \text{di } ((\triangleright f \wedge g1) \wedge \text{len}(i)) \wedge \text{di } ((\triangleright f \wedge g2) \wedge \text{len}(j))) \longrightarrow (i=j))$

using *linorder-neqE-nat* **by** (*simp add: DiLenFstAndsem*) *blast*

lemma *FstAndLenSame-1*:

$\vdash \text{di } ((\triangleright f \wedge g1) \wedge \text{len}(i)) \wedge \text{di } ((\triangleright f \wedge g2) \wedge \text{len}(j)) \longrightarrow (\#i=\#j)$

using *FstAndLenSame Valid-def* **by** *fastforce*

lemma *FstLenSameChop*:

$(\forall \sigma. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i));h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(j));h2) \longrightarrow (i=j))$

proof

```

fix  $\sigma$ 
show  $(\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i)); h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(j)); h2) \longrightarrow (i=j)$ 
proof
assume 0:  $(\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i)); h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(j)); h2)$ 
have 1:  $(\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i)); h1)$  using 0 by auto
have 2:  $(\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i)); h1) \longrightarrow$ 
 $(\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i)); \# \text{True})$  by (metis ChopImpDi Valid-def di-d-def unl-lift2)
have 3:  $(\sigma \models \text{di}((\triangleright f \wedge g1) \wedge \text{len}(i)))$  using 1 2 by (simp add: di-d-def)
have 4:  $(\sigma \models ((\triangleright f \wedge g2) \wedge \text{len}(j)); h2)$  using 0 by auto
have 5:  $(\sigma \models ((\triangleright f \wedge g2) \wedge \text{len}(j)); h2) \longrightarrow$ 
 $(\sigma \models ((\triangleright f \wedge g2) \wedge \text{len}(j)); \# \text{True})$  by (metis ChopImpDi Valid-def di-d-def unl-lift2)
have 6:  $(\sigma \models \text{di}((\triangleright f \wedge g2) \wedge \text{len}(j)))$  using 4 5 by (simp add: di-d-def)
have 7:  $(\sigma \models \text{di}((\triangleright f \wedge g1) \wedge \text{len}(i)) \wedge \text{di}((\triangleright f \wedge g2) \wedge \text{len}(j)))$  using 3 6 by auto
thus  $(i=j)$  using FstAndLenSame by blast
qed
qed

```

lemma *FstLenSameChop-1:*

```

 $\vdash ((\triangleright f \wedge g1) \wedge \text{len}(i)); h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(j)); h2 \longrightarrow (\#i=\#j)$ 
using FstLenSameChop Valid-def by fastforce

```

lemma *DiImpExistsOneDiLenAndFst:*

```

assumes nfinite  $\sigma$ 
 $(\sigma \models \text{di } f)$ 
shows  $(\exists! k. (\sigma \models \text{di}(\triangleright f \wedge \text{len}(k))))$ 
proof -
have 1:  $(\sigma \models \text{di}(\triangleright f))$ 
using assms DiEqvDiFst
by (metis (no-types, lifting) FiniteImpAnd finite-defs inteq-reflection unl-lift2)
have 2:  $(\sigma \models \triangleright f) = ((\sigma \models \triangleright f) \wedge (\exists k. (\sigma \models \text{len}(k))))$ 
using AndExistsLen[of TEMP  $\triangleright f$ ]
using assms len-defs nfinite-conv-nlength-enat by blast
have 3:  $((\sigma \models \triangleright f) \wedge (\exists k. (\sigma \models \text{len}(k)))) =$ 
 $(\exists k. (\sigma \models \triangleright f) \wedge (\sigma \models \text{len}(k)))$ 
by auto
have 4:  $(\sigma \models \text{di}(\triangleright f)) = (\exists k. (\sigma \models \text{di}(\triangleright f \wedge \text{len}(k))))$ 
using assms by (metis DiLensem itl-defs(10))
have 5:  $(\exists k. (\sigma \models \text{di}(\triangleright f \wedge \text{len}(k))))$ 
using 1 using 4 by auto
then obtain i where 6:  $(\sigma \models \text{di}(\triangleright f \wedge \text{len}(i)))$  by blast
from 5 obtain j where 7:  $(\sigma \models \text{di}(\triangleright f \wedge \text{len}(j)))$  by blast
have 8:  $(\sigma \models \text{di}(\triangleright f \wedge \text{len}(i))) \wedge (\sigma \models \text{di}(\triangleright f \wedge \text{len}(j)))$ 
using 6 7 by auto
hence 9:  $(\sigma \models \text{di}(\triangleright f \wedge \text{len}(i)) \wedge \text{di}(\triangleright f \wedge \text{len}(j)))$ 
by simp
hence 10:  $i=j$ 
using FstLenSame by blast
have 11:  $\bigwedge j. (\sigma \models \text{di}(\triangleright f \wedge \text{len}(j))) \longrightarrow (j=i)$ 
using 9 10 using FstLenSame by auto
thus  $(\exists! k. (\sigma \models \text{di}(\triangleright f \wedge \text{len } k)))$ 

```


using 11 5 by blast
qed

lemma *DiImpExistsOneDiLenAndFst-1:*

$\vdash \text{finite} \longrightarrow \text{di } f \longrightarrow (\exists! k. (\text{di}(\triangleright f \wedge \text{len}(k))))$

using *DiImpExistsOneDiLenAndFst* **unfolding** *Valid-def finite-defs* **by** *fastforce*

lemma *LFstAndDist-help:*

$(\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(k)); h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(k)); h2) =$
 $(\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k)); (h1 \wedge h2))$

using *LFixedAndDistr* **by** *fastforce*

lemma *LFstAndDist-help-1:*

$(\exists k. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(k)); h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(k)); h2)) =$
 $(\exists k. (\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k)); (h1 \wedge h2)))$

proof

assume 0: $\exists k. \sigma \models ((\triangleright f \wedge g1) \wedge \text{len } k) ; h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len } k) ; h2$

obtain *k* **where** 1: $\sigma \models ((\triangleright f \wedge g1) \wedge \text{len } k) ; h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len } k) ; h2$

using 0 **by** *auto*

hence 2: $(\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k)); (h1 \wedge h2))$

using *LFstAndDist-help* **by** *blast*

show $(\exists k. (\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k)); (h1 \wedge h2)))$

using 2 **by** *auto*

next

assume 3: $(\exists k. (\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k)); (h1 \wedge h2)))$

obtain *k* **where** 4: $(\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k)); (h1 \wedge h2))$

using 3 **by** *auto*

hence 5: $(\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(k)); h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(k)); h2)$

using *LFstAndDist-help* **by** *blast*

show $(\exists k. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(k)); h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(k)); h2))$

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: $(\sigma \models (\triangleright f \wedge g1); h1) = (\sigma \models (\triangleright f \wedge g1) \frown h1)$

using *FstImpFinite schop-d-def*

by (*metis (no-types, lifting) FstFstAndEqvFstAnd Prop10 inteq-reflection*)

have 02: $(\sigma \models (\triangleright f \wedge g2); h2) = (\sigma \models (\triangleright f \wedge g2) \frown h2)$

using *FstImpFinite schop-d-def*

by (*metis (no-types, lifting) FstFstAndEqvFstAnd Prop10 inteq-reflection*)

have 1: $(\sigma \models (\triangleright f \wedge g1) \frown h1) = (\exists i. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i)); h1))$

using *AndExistsLenChop[of TEMP $\triangleright f \wedge g1$]* **by** *fastforce*

have 2: $(\sigma \models (\triangleright f \wedge g2) \frown h2) = (\exists j. (\sigma \models ((\triangleright f \wedge g2) \wedge \text{len}(j)); h2))$

using *AndExistsLenChop[of TEMP $\triangleright f \wedge g2$]* **by** *fastforce*

have 3: $(\sigma \models (\triangleright f \wedge g1) \frown h1 \wedge (\triangleright f \wedge g2) \frown h2) =$

$((\exists i j. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i)); h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(j)); h2))$
 $)$

using 1 2 **by** *auto*
have 4: $(\exists i j. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i)); h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(j)); h2)) =$
 $(\exists k. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(k)); h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(k)); h2))$
using *FstLenSameChop* **by** *blast*
have 5: $(\exists k. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(k)); h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(k)); h2)) =$
 $(\exists k. (\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k)); (h1 \wedge h2)))$
using *LFstAndDist-help-1* **by** *blast*
have 6: $(\exists k. (\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k)); (h1 \wedge h2))) =$
 $(\sigma \models ((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \neg (h1 \wedge h2))$
using *AndExistsLenChop*[*of TEMP* $((\triangleright f \wedge g1) \wedge \triangleright f \wedge g2)]$ **by** *fastforce*
have 7: $(\sigma \models ((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)); (h1 \wedge h2)) =$
 $(\sigma \models (\triangleright f \wedge g1 \wedge g2); (h1 \wedge h2))$
by (*auto simp add: chop-defs*)
have 8: $(\sigma \models (\triangleright f \wedge g1 \wedge g2) \neg (h1 \wedge h2)) = (\sigma \models (\triangleright f \wedge g1 \wedge g2); (h1 \wedge h2))$
using *FstImpFinite schop-d-def*
by (*metis (no-types, lifting) FstFstAndEqvFstAnd Prop10 inteq-reflection*)
have 9: $(\sigma \models (\triangleright f \wedge g1) \neg h1 \wedge (\triangleright f \wedge g2) \neg h2) =$
 $(\sigma \models (\triangleright f \wedge g1 \wedge g2); (h1 \wedge h2))$
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 *s chop-d-def* **using** 01 02 9 **by** *auto*
qed

lemma *LFstAndDistr*:
 $\vdash ((\triangleright f \wedge g1); h1 \wedge (\triangleright f \wedge g2); h2) = (\triangleright f \wedge g1 \wedge g2); (h1 \wedge h2)$
using *LFstAndDistrsem* **by** *fastforce*

lemma *LFstAndDistrA*:
 $\vdash ((\triangleright f \wedge g1); h \wedge (\triangleright f \wedge g2); h) = (\triangleright f \wedge g1 \wedge g2); h$
proof –
have 1: $\vdash ((\triangleright f \wedge g1); h \wedge (\triangleright f \wedge g2); h) = (\triangleright f \wedge g1 \wedge g2); (h \wedge h)$ **using** *LFstAndDistr* **by** *blast*
have 2: $\vdash (\triangleright f \wedge g1 \wedge g2); (h \wedge h) = (\triangleright f \wedge g1 \wedge g2); h$ **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: $\vdash (empty \vee (\neg(\triangleright f \frown df(g))) \frown skip) = (empty \vee \neg(\neg((\neg(\triangleright f \frown df(g))) \frown skip)))$
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: $\vdash (empty \vee (finite \wedge \neg(\triangleright f \frown (df(g) \frown skip)))) = (empty \vee (finite \wedge (\neg(di(\triangleright f)) \vee (\triangleright f \frown (\neg(df(g) \frown skip))))))$
using NotFstChop
by (*metis FstImpFinite Prop10 int-simps(1) inteq-reflection schop-d-def*)
have 015: $\vdash (\neg(df(g) \frown skip)) = (empty \vee inf \vee bf(\neg g) \frown skip)$
by (*metis NotDfEqvBfNot NotSChopNotSkip inteq-reflection*)
have 16: $\vdash (\triangleright f \frown (\neg(df(g) \frown skip))) = (\triangleright f \frown (empty \vee inf \vee bf(\neg g) \frown skip))$
by (*simp add: 015 RightSChopEqvSChop*)
have 0160: $\vdash bf(\neg g) \frown skip = bi(\neg g) \frown skip$
by (*simp add: FiniteImpAnd FiniteImpBfEqvBi LeftChopEqvChop schop-d-def*)
have 016: $\vdash (empty \vee inf \vee bf(\neg g) \frown skip) = (inf \vee bs(\neg g))$
unfolding bs-d-def using 0160 by fastforce
have 017: $\vdash (\triangleright f \frown (\neg(df(g) \frown skip))) = (\triangleright f \frown (inf \vee bs(\neg g)))$

by (metis 016 16 inteq-reflection)
 have 018: $\vdash (\triangleright f \frown (\text{inf} \vee \text{bs}(\neg g))) = (\triangleright f \frown \text{inf} \vee \triangleright f \frown (\text{bs}(\neg g)))$
 by (simp add: SChopOrEqv)
 have 019: $\vdash (\text{finite} \wedge (\neg(\text{di}(\triangleright f)) \vee (\triangleright f \frown (\neg(\text{df}(g) \frown \text{skip})))) =$
 $((\text{finite} \wedge \neg(\text{di}(\triangleright f))) \vee (\text{finite} \wedge (\triangleright f \frown (\neg(\text{df}(g) \frown \text{skip}))))$
 by force
 have 020: $\vdash (\text{finite} \wedge (\triangleright f \frown (\neg(\text{df}(g) \frown \text{skip})))) = (\text{finite} \wedge (\triangleright f \frown \text{inf} \vee \triangleright f \frown (\text{bs}(\neg g))))$
 using 017 018 by fastforce
 have 021: $\vdash \triangleright f \frown \text{inf} \longrightarrow \text{inf}$
 by (metis AndChopB FiniteChopInfEqvInf Prop11 lift-imp-trans schop-d-def)
 have 022: $\vdash (\text{finite} \wedge (\triangleright f \frown \text{inf} \vee \triangleright f \frown (\text{bs}(\neg g)))) = (\text{finite} \wedge \triangleright f \frown (\text{bs}(\neg g)))$
 using 021 unfolding finite-d-def by fastforce
 have 023: $\vdash \triangleright f \frown (\text{bs}(\neg g)) \longrightarrow \text{finite}$
 by (metis AndChopB EmptyImpFinite FiniteChopSkipImpFinite Prop02 SChopImpFinite bs-d-def
 lift-imp-trans schop-d-def)
 have 024: $\vdash (\text{finite} \wedge \triangleright f \frown (\text{bs}(\neg g))) = \triangleright f \frown (\text{bs}(\neg g))$
 using 023 by fastforce
 have 025: $\vdash (\text{empty} \vee (\text{finite} \wedge (\neg(\text{di}(\triangleright f)) \vee (\triangleright f \frown (\neg(\text{df}(g) \frown \text{skip}))))) =$
 $(\text{empty} \vee (\text{finite} \wedge \neg(\text{di}(\triangleright f))) \vee \triangleright f \frown (\text{bs}(\neg g)))$
 by (metis 014 019 020 022 024 inteq-reflection)
 have 026: $\vdash \text{bs}(\neg(\triangleright f;g)) = (\text{empty} \vee (\text{finite} \wedge (\neg(\text{di}(\triangleright f)) \vee (\triangleright f \frown (\neg(\text{df}(g) \frown \text{skip})))))$
 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: $\vdash \triangleright(\triangleright f;g) = ((\triangleright f;g) \wedge \text{bs}(\neg(\triangleright f;g)))$
 by (simp add: first-d-def)
 have 2: $\vdash \text{bs}(\neg(\triangleright f;g)) = (\text{empty} \vee (\text{finite} \wedge \neg(\text{di}(\triangleright f))) \vee (\triangleright f; \text{bs}(\neg g)))$
 using BsNotFstChop by auto
 hence 3: $\vdash ((\triangleright f;g) \wedge \text{bs}(\neg(\triangleright f;g))) = ((\triangleright f;g) \wedge (\text{empty} \vee (\text{finite} \wedge \neg(\text{di}(\triangleright f))) \vee (\triangleright f; \text{bs}(\neg g))))$
 by auto
 have 4: $\vdash ((\triangleright f;g) \wedge (\text{empty} \vee (\text{finite} \wedge \neg(\text{di}(\triangleright f))) \vee (\triangleright f; \text{bs}(\neg g)))) =$
 $((\triangleright f;g) \wedge \text{empty}) \vee ((\triangleright f;g) \wedge \neg(\text{di}(\triangleright f)) \wedge \text{finite}) \vee ((\triangleright f;g) \wedge (\triangleright f; \text{bs}(\neg g)))$
 by auto
 have 5: $\vdash \neg((\triangleright f;g) \wedge \neg(\text{di}(\triangleright f)) \wedge \text{finite})$
 using ChopImpDi by fastforce
 hence 6: $\vdash (((\triangleright f;g) \wedge \text{empty}) \vee ((\triangleright f;g) \wedge \neg(\text{di}(\triangleright f)) \wedge \text{finite}) \vee ((\triangleright f;g) \wedge (\triangleright f; \text{bs}(\neg g)))) =$
 $((\triangleright f;g) \wedge \text{empty}) \vee ((\triangleright f;g) \wedge (\triangleright f; \text{bs}(\neg g)))$
 by auto
 have 7: $\vdash ((\triangleright f;g) \wedge (\triangleright f; (\text{bs}(\neg g)))) = ((\triangleright f; (g \wedge (\text{bs}(\neg g))))$
 using LFstAndDistrC by blast
 hence 8: $\vdash (((\triangleright f;g) \wedge \text{empty}) \vee ((\triangleright f;g) \wedge (\triangleright f; (\text{bs}(\neg g)))) =$
 $((\triangleright f;g) \wedge \text{empty}) \vee ((\triangleright f; (g \wedge (\text{bs}(\neg g))))$
 by auto
 have 9: $\vdash (((\triangleright f;g) \wedge \text{empty}) \vee ((\triangleright f; (g \wedge (\text{bs}(\neg g)))) = ((\triangleright f;g) \wedge \text{empty}) \vee \triangleright f; \triangleright g$
 by (simp add: first-d-def)

have 10: $\vdash ((\triangleright f;g) \wedge \text{empty}) = ((\triangleright f;\triangleright g) \wedge \text{empty})$
 using *FstChopEmptyEqvFstChopFstEmpty*
 by (*metis ChopEmptyAndEmpty SChopEmptyAndEmpty inteq-reflection*)
hence 11: $\vdash (((\triangleright f;g) \wedge \text{empty}) \vee \triangleright f;\triangleright g) = (((\triangleright f;\triangleright g) \wedge \text{empty}) \vee \triangleright f;\triangleright g)$
 by *auto*
have 12: $\vdash (((\triangleright f;\triangleright g) \wedge \text{empty}) \vee \triangleright f;\triangleright g) = \triangleright f;\triangleright g$
 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: $\vdash \triangleright f = (\triangleright f);\text{empty}$ **using** *ChopEmpty* **by** (*metis int-eq*)
hence 2: $\vdash \triangleright(\triangleright f) = \triangleright((\triangleright f);\text{empty})$ **using** *FstEqvRule* **by** *blast*
have 3: $\vdash \triangleright((\triangleright f);\text{empty}) = \triangleright f;\triangleright \text{empty}$ **using** *FstFstChopEqvFstChopFst* **by** *auto*
have 4: $\vdash \triangleright f;\triangleright \text{empty} = \triangleright f;\text{empty}$ **using** *FstEmpty* **using** *RightChopEqvChop* **by** *blast*
have 5: $\vdash \triangleright f;\text{empty} = \triangleright f$ **using** *ChopEmpty* **by** *blast*
from 2 3 4 5 **show** *?thesis* **by** *fastforce*
qed

lemma *FstCSEqvEmpty*:

$\vdash \triangleright(f^\star) = \text{empty}$

proof –

have 1: $\vdash \triangleright(f^\star) = \triangleright(\text{empty} \vee ((f \wedge \text{more});f^\star))$ **using** *ChopstarEqv FstEqvRule* **by** *blast*
from 1 **show** *?thesis* **using** *FstEmptyOrEqvEmpty* **by** *fastforce*
qed

lemma *FstIterFixFst*:

$\vdash \text{fpower}(\triangleright f) n = \triangleright(\text{fpower}(\triangleright f) n)$

proof

(*induct n*)

case 0

then show *?case*

proof –

have 1: $\vdash \text{fpower}(\triangleright f) 0 = \text{empty}$
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:  $\vdash (\triangleright f) ; \triangleright (fpower (\triangleright f) n) = \triangleright(\triangleright f; (fpower (\triangleright f) n))$ 
using FstFstChopEqvFstChopFst by fastforce
have 7:  $\vdash \triangleright(\triangleright f; (fpower (\triangleright f) n)) = \triangleright(fpower (\triangleright f) (Suc n))$ 
  by (metis 4 6 inteq-reflection)
from 4 5 6 7 show ?thesis by fastforce
qed
qed

```

lemma *DsImpNotFst*:

```

 $\vdash ds f \longrightarrow (\neg(\triangleright f))$ 
proof –
  have 1:  $\vdash (ds f \wedge \triangleright f) = (ds f \wedge (f \wedge bs (\neg f)))$  by (simp add: first-d-def)
  have 2:  $\vdash (ds f \wedge (f \wedge bs (\neg f))) = (ds f \wedge f \wedge \neg(ds f))$  using NotDsEqvBsNot by fastforce
  from 1 2 show ?thesis by fastforce
qed

```

lemma *FstLenAndEqvLenAnd*:

```

 $\vdash \triangleright(len(k) \wedge f) = (len(k) \wedge f)$ 
proof –
  have 1:  $\vdash len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow ds(len(k))$ 
    using DsAndImpElimL by fastforce
  have 01:  $\vdash ds(len(k) \wedge f) = (finite \longrightarrow di(len(k) \wedge f) \frown skip)$ 
    by (simp add: DsDi)
  have 02:  $\vdash len k \longrightarrow finite$ 
    by (simp add: len-k-finite)
  have 03:  $\vdash di(len(k) \wedge f) \frown skip \longrightarrow (di(len(k))) \frown skip$ 
    by (simp add: DiAndA LeftSCHopImpSCHop)
  hence 2:  $\vdash len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (di(len(k))) \frown skip$ 
    using 01 02 03 by fastforce
  hence 3:  $\vdash len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow ((len(k) \frown \# True) \frown skip)$ 
    unfolding di-d-def
    by (metis Prop10 inteq-reflection len-k-finite schop-d-def)
  hence 4:  $\vdash len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (len(k) \frown (\# True \frown skip))$ 
    using SCHopAssoc by fastforce
  hence 5:  $\vdash len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (len(k) \frown (skip \frown finite))$ 
    by (metis Prop10 SkipFiniteEqvFiniteSkip WPowerstar-ext WPowerstar-skip-finite int-simps(17)
      inteq-reflection schop-d-def)
  hence 6:  $\vdash len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (len(k) \frown (skip \frown finite)) \wedge len(k)$ 
    by auto
  hence 7:  $\vdash len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (len(k) \frown (skip \frown finite)) \wedge len(k) \frown empty$ 
    by (metis ChopEmpty Prop12 inteq-reflection len-k-finite lift-imp-trans schop-d-def)
  hence 8:  $\vdash len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (len(k) \frown ((skip \frown finite) \wedge empty))$ 
    by (metis LenSCHopAnd Prop11 lift-imp-trans)
  have 08:  $\vdash \neg((skip \frown finite) \wedge empty)$ 
    by (metis ChopEmptyAndEmpty FmoreEqvSkipChopFinite NotFmoreAndEmpty SCHopEmptyAndEmpty
      inteq-reflection lift-and-com)
  have 09:  $\vdash ((skip \frown finite) \wedge empty) = \# False$ 
    using 08 by (simp add: Prop11)
  have 010:  $\vdash len(k) \frown ((skip \frown finite) \wedge empty) = \# False$ 

```

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: $\vdash \text{ds } f = (\text{finite} \longrightarrow \text{di } f \frown \text{skip})$
 using DsDi by blast
 have 4: $\vdash \text{di } f \frown \text{skip} = (f \frown \# \text{True}) \frown \text{skip}$
 by (metis ChopTrueAndFiniteEqvAndFiniteChopFinite SChopFiniteEqvSChopTrueAndFinite
 di-d-def int-simps(1) inteq-reflection schop-d-def)
 have 5: $\vdash (f \frown \# \text{True}) \frown \text{skip} = f \frown (\# \text{True} \frown \text{skip})$
 by (meson Prop11 SChopAssoc)
 have 6: $\vdash f \frown (\# \text{True} \frown \text{skip}) = f \frown (\text{skip} \frown \text{finite})$
 by (metis ChopTrueAndFiniteEqvAndFiniteChopFinite DiSkipEqvMore DiamondSChopdef
 FmoreEqvSkipChopFinite RightSChopEqvSChop SkipFiniteEqvFiniteSkip di-d-def fmore-d-def
 inteq-reflection schop-d-def sometimes-d-def)
 have 7: $\vdash f \frown (\text{skip} \frown \text{finite}) = (f \frown \text{skip}) \frown \text{finite}$
 using SChopAssoc by blast

have 8: $\vdash (f \frown \text{skip}) \frown \text{finite} = (di(f \frown \text{skip}) \wedge \text{finite})$
by (*metis ChopTrueAndFiniteEqvAndFiniteChopFinite Prop11 di-d-def schop-d-def*)
have 9: $\vdash (\neg(ds\ f)) = (\neg(di(f \frown \text{skip}))) \wedge \text{finite}$
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: $\vdash \triangleright f \longrightarrow \neg(di\ f \frown \text{skip})$ **using** *FstImpNotDiChopSkip* **by** *blast*
have 2: $\vdash di\ f \wedge \text{finite} \longrightarrow f \vee (di\ f \frown \text{skip})$ **using** *DiEqvOrDiChopSkipB*
using *DiFiniteEqv* **by** *fastforce*
have 3: $\vdash \triangleright f \wedge di\ f \longrightarrow (f \vee (di\ f \frown \text{skip})) \wedge \neg(di\ f \frown \text{skip})$ **using** 1 2
using *FstAndElimL* **by** *fastforce*
have 4: $\vdash ((f \vee (di\ f \frown \text{skip})) \wedge \neg(di\ f \frown \text{skip})) = (f \wedge \neg(di\ f \frown \text{skip}))$ **by** *auto*
have 5: $\vdash \triangleright f \wedge di\ f \longrightarrow f \wedge \neg(di\ f \frown \text{skip})$ **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: $\vdash \text{finite} \longrightarrow di(\triangleright f \wedge g) = ((\triangleright f \wedge g) \vee di((\triangleright f \wedge g) \frown \text{skip}))$
using *DiEqvOrDiChopSkipA*
by (*metis (no-types, lifting) DiEqvDiFst FstFstAndEqvFstAnd FstImpFinite Prop10 inteq-reflection schop-d-def*)
have 4: $\vdash \text{finite} \longrightarrow di((\triangleright f \wedge g) \frown \text{skip}) = ((\triangleright f \wedge g) \frown \text{skip}) \frown \text{finite}$
unfolding *di-d-def*
by (*smt (z3) ChopTrueAndFiniteEqvAndFiniteChopFinite intI inteq-reflection schop-d-def unl-lift2*)
have 04: $\vdash \triangleright f \longrightarrow \text{finite}$
by (*simp add: FstImpFinite*)
have 5: $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow (\triangleright f \wedge g) \vee ((\triangleright f \wedge g) \frown \text{skip}) \frown \text{finite}$
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: $\vdash ((\triangleright f \wedge g) \frown \text{skip}) \frown \text{finite} \longrightarrow di(f \frown \text{skip})$
by (*metis AndChopA ChopImpDi lift-imp-trans schop-d-def*)
have 9: $\vdash \triangleright f \longrightarrow \neg(di(f \frown \text{skip}))$
using *FstImpNotDiChopSkipB* **by** *blast*

have 10: $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow ((\triangleright f \wedge g) \vee di(f \frown skip))$
using 5 8 by fastforce
have 11: $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow \neg(di(f \frown skip)) \wedge ((\triangleright f \wedge g) \vee di(f \frown skip))$
using 9 10 1 by fastforce
have 12: $\vdash (\neg(di(f \frown skip)) \wedge ((\triangleright f \wedge g) \vee di(f \frown skip))) = (\neg(di(f \frown skip)) \wedge ((\triangleright f \wedge g)))$
by auto
have 13: $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow (\triangleright f \wedge g)$
using 11 12 by auto
have 14: $\vdash (\triangleright f \wedge g) \longrightarrow \triangleright f$
by auto
hence 15: $\vdash (\triangleright f \wedge g) \longrightarrow di(\triangleright f \wedge g)$
using DiIntro by auto
have 16: $\vdash (\triangleright f \wedge g) \longrightarrow \triangleright f \wedge di(\triangleright f \wedge g)$
using 14 15 by auto
from 13 16 show ?thesis by fastforce
qed

lemma FstAndDiImpBsNotAndDi:

$\vdash (\triangleright f \wedge di g) \longrightarrow (bs (\neg(di f \wedge g)))$

proof –

have 1: $\vdash (\triangleright f \wedge di g) \wedge \neg(bs (\neg(di f \wedge g))) \longrightarrow ds(di f \wedge g)$
by (auto simp add: ds-d-def)
hence 2: $\vdash (\triangleright f \wedge di g) \wedge \neg(bs (\neg(di f \wedge g))) \longrightarrow ds(di f)$
using DsAndImp by fastforce
hence 3: $\vdash (\triangleright f \wedge di g) \wedge \neg(bs (\neg(di f \wedge g))) \longrightarrow di(di f) \frown skip$
using DsDi FstImpFinite by fastforce
hence 4: $\vdash (\triangleright f \wedge di g) \wedge \neg(bs (\neg(di f \wedge g))) \longrightarrow di f \frown skip$
using DiEqvDiDi by (metis int-eq)
hence 5: $\vdash (\triangleright f \wedge di g) \wedge \neg(bs (\neg(di f \wedge g))) \longrightarrow ds f$
using DsDi by fastforce
hence 6: $\vdash (\triangleright f \wedge di g) \wedge \neg(bs (\neg(di f \wedge g))) \longrightarrow \neg(\triangleright f)$
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*:

$\vdash \triangleright(\circ f) = \circ(\triangleright f)$

proof –

have 1: $\vdash \triangleright(\circ f) = \triangleright(\text{skip} ; f)$ **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: $\vdash \text{skip} ; \triangleright f = \circ(\triangleright f)$ **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*:

$\vdash \triangleright(\text{di } f \wedge \text{di } g) = \triangleright(\text{df } f \wedge \text{df } g)$

proof –

have 1: $\vdash \triangleright(\text{di } f \wedge \text{di } g) = \triangleright((\text{di } f \wedge \text{di } g) \wedge \text{finite})$
by (*simp add: FstEqvFstAndFinite*)
have 2: $\vdash ((\text{di } f \wedge \text{di } g) \wedge \text{finite}) = ((\text{df } f \wedge \text{df } g) \wedge \text{finite})$

```

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 *FstBoxStateEqvStateAndEmpty*:

```

 $\vdash \triangleright (\Box (init\ w)) = ((init\ w) \wedge empty)$ 
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:  $\vdash ((init\ w) \wedge empty) ; (\Box (init\ w))^* = ((init\ w) \wedge (\Box (init\ w))^*)$ 
using StateAndEmptyChop by blast
have 5:  $\vdash ((init\ w) \wedge (\Box (init\ w))^*) = ((init\ w) \wedge empty) ; (\Box (init\ w))^*$ 
using 4 by fastforce
have 6:  $\vdash \Box (init\ w) = ((init\ w) \wedge empty) ; (\Box (init\ w))^*$ 
using 3 5 by fastforce
have 7:  $\vdash ((init\ w) \wedge empty) ; (\Box (init\ w))^* = \triangleright (init\ w) ; (\Box (init\ w))^*$ 
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:  $\vdash \triangleright (init\ w) ; \triangleright ((\Box (init\ w))^*) = \triangleright (init\ w) ; empty$ 
using RightChopEqvChop FstCSEqvEmpty by blast
have 12:  $\vdash \triangleright (init\ w) ; empty = \triangleright (init\ w)$ 
using RightChopEqvChop ChopEmpty by blast
have 13:  $\vdash \triangleright (init\ w) = ((init\ w) \wedge empty)$ 
using FstState by fastforce
from 9 10 11 12 13 show ?thesis by fastforce
qed

```

lemma *FstAndFstStarEqvFst*:

```

 $\vdash (schopstar\ (\triangleright f) \wedge (\triangleright f)) = \triangleright f$ 
proof –
have 1:  $\vdash (schopstar\ (\triangleright f)) = (empty \vee (\triangleright f) \frown (schopstar\ (\triangleright f)))$ 
by (meson Prop11 Schopstar-unfoldl-eq)
have 2:  $\vdash ((schopstar\ (\triangleright f)) \wedge \triangleright f) = ((empty \vee (\triangleright f) \frown (schopstar\ (\triangleright f))) \wedge \triangleright f)$ 

```

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:*

$\vdash (df(\text{halt}(\text{init } w) \wedge f) \wedge df(\text{halt}(\text{init } w) \wedge g)) = df(\text{halt}(\text{init } w) \wedge f \wedge g)$

proof –

have 01: $\vdash (\text{halt}(\text{init } w) \wedge f \wedge \text{finite}) = (\triangleright (\text{fin}(\text{init } w)) \wedge f \wedge \text{finite})$
 using HaltStateEqvFstFinState by fastforce
 have 02: $\vdash (\text{halt}(\text{init } w) \wedge g \wedge \text{finite}) = (\triangleright (\text{fin}(\text{init } w)) \wedge g \wedge \text{finite})$
 using HaltStateEqvFstFinState by fastforce

have 03: $\vdash (\text{halt } (\text{init } w) \wedge f \wedge \text{finite}) \neg \# \text{True} = (\triangleright(\text{fin } (\text{init } w)) \wedge f \wedge \text{finite}) \neg \# \text{True}$
by (*simp add: 01 LeftSChopEqvSChop*)
have 04: $\vdash (\text{halt } (\text{init } w) \wedge g \wedge \text{finite}) \neg \# \text{True} = (\triangleright(\text{fin } (\text{init } w)) \wedge g \wedge \text{finite}) \neg \# \text{True}$
by (*simp add: 02 LeftSChopEqvSChop*)
have 05: $\vdash (\triangleright(\text{fin } (\text{init } w)) \wedge f \wedge \text{finite}) \neg \# \text{True} = (\triangleright(\text{fin } (\text{init } w)) \wedge f) \neg \# \text{True}$
by (*metis (no-types, lifting) FstFstAndEqvFstAnd FstImpFinite LeftSChopEqvSChop Prop11 Prop12 inteq-reflection*)
have 06: $\vdash (\triangleright(\text{fin } (\text{init } w)) \wedge g \wedge \text{finite}) \neg \# \text{True} = (\triangleright(\text{fin } (\text{init } w)) \wedge g) \neg \# \text{True}$
by (*metis (no-types, lifting) FstFstAndEqvFstAnd FstImpFinite LeftSChopEqvSChop Prop11 Prop12 inteq-reflection*)
have 07: $\vdash (\text{halt } (\text{init } w) \wedge f \wedge \text{finite}) \neg \# \text{True} = (\text{halt } (\text{init } w) \wedge f) \neg \# \text{True}$
unfolding *schop-d-def*
by (*metis (no-types, lifting) LeftChopEqvChop Prop11 Prop12 lift-and-com*)
have 08: $\vdash (\text{halt } (\text{init } w) \wedge g \wedge \text{finite}) \neg \# \text{True} = (\text{halt } (\text{init } w) \wedge g) \neg \# \text{True}$
unfolding *schop-d-def*
by (*metis (no-types, lifting) LeftChopEqvChop Prop11 Prop12 lift-and-com*)
have 1: $\vdash (\text{df}(\text{halt } (\text{init } w) \wedge f) \wedge \text{df}(\text{halt } (\text{init } w) \wedge g)) =$
 $(\text{df}(\triangleright(\text{fin } (\text{init } w)) \wedge f) \wedge \text{df}(\triangleright(\text{fin } (\text{init } w)) \wedge g))$
unfolding *df-d-def*
by (*metis 01 02 05 06 07 08 inteq-reflection lift-and-com*)
have 2: $\vdash (\text{df}(\triangleright(\text{fin } (\text{init } w)) \wedge f) \wedge \text{df}(\triangleright(\text{fin } (\text{init } w)) \wedge g)) =$
 $\text{df}(\triangleright(\text{fin } (\text{init } w)) \wedge f \wedge g)$
using *LFstAndDistrD*
by (*metis (no-types, lifting) FstFstAndEqvFstAnd FstImpFinite Prop10 df-d-def di-d-def inteq-reflection schop-d-def*)
have 08: $\vdash (\triangleright(\text{fin } (\text{init } w)) \wedge f \wedge g) = (\text{halt } (\text{init } w) \wedge f \wedge g \wedge \text{finite})$
using *HaltStateEqvFstFinState* **by** *fastforce*
have 09: $\vdash (\triangleright(\text{fin } (\text{init } w)) \wedge f \wedge g) \neg \# \text{True} = (\text{halt } (\text{init } w) \wedge f \wedge g \wedge \text{finite}) \neg \# \text{True}$
by (*simp add: 08 LeftSChopEqvSChop*)
have 010: $\vdash (\text{halt } (\text{init } w) \wedge f \wedge g \wedge \text{finite}) \neg \# \text{True} = (\text{halt } (\text{init } w) \wedge f \wedge g) \neg \# \text{True}$
unfolding *schop-d-def*
by (*metis (no-types, lifting) LeftChopEqvChop Prop11 Prop12 lift-and-com*)
have 3: $\vdash \text{df}(\triangleright(\text{fin } (\text{init } w)) \wedge f \wedge g) = \text{df}(\text{halt } (\text{init } w) \wedge f \wedge g)$
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*:

$\vdash ((\triangleright(\text{len}(5)) \wedge \triangleright(\text{len}(2))) ; (\text{len}(5) \vee \text{len}(2))) = \# \text{False}$

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: $\vdash (\text{len}(5) \wedge \text{len}(2)) = \# \text{False}$

by (*simp add: Valid-def len-defs*)

have 3: $\vdash ((\text{len}(5) \wedge \text{len}(2)); (\text{len}(5) \vee \text{len}(2))) = (\# \text{False}; (\text{len}(5) \vee \text{len}(2)))$

by (*simp add: 2 LeftChopEqvChop*)

have 4: $\vdash (\#False; (len(5) \vee len(2))) = \#False$
by (*simp add: Valid-def chop-defs*)
from 1 3 4 **show** ?thesis **by** fastforce
qed

lemma counter-ex-rhs:

$\vdash ((\triangleright (len(5)) ; (len(5) \vee len(2))) \wedge (\triangleright (len(2)) ; (len(5) \vee len(2)))) = len(7)$

proof –

have 1: $\vdash (\triangleright (len(5)) ; (len(5) \vee len(2))) =$
 $len(5); (len(5) \vee len(2))$
using FstLenEqvLen LeftChopEqvChop **by** blast
have 2: $\vdash (\triangleright (len(2)) ; (len(5) \vee len(2))) =$
 $len(2); (len(5) \vee len(2))$
using FstLenEqvLen LeftChopEqvChop **by** blast
have 3: $\vdash len(5); (len(5) \vee len(2)) =$
 $((len(5); len(5)) \vee (len(5); len(2)))$
by (*simp add: ChopOrEqv*)
have 4: $\vdash ((len(5); len(5)) \vee (len(5); len(2))) =$
 $(len(10) \vee len(7))$
using LenEqvLenChopLen inteq-reflection **by** fastforce
have 5: $\vdash len(2); (len(5) \vee len(2)) =$
 $((len(2); len(5)) \vee (len(2); len(2)))$
by (*simp add: ChopOrEqv*)
have 6: $\vdash ((len(2); len(5)) \vee (len(2); len(2))) =$
 $(len(7) \vee len(4))$
using LenEqvLenChopLen inteq-reflection **by** fastforce
have 7: $\vdash ((len(10) \vee len(7)) \wedge (len(7) \vee len(4))) =$
 $((len(7) \vee len(10)) \wedge (len(7) \vee len(4)))$
by fastforce
have 8: $\vdash ((len(7) \vee len(10)) \wedge (len(7) \vee len(4))) =$
 $(len(7) \vee (len(10) \wedge len(4)))$
by fastforce
have 9: $\vdash (len(10) \wedge len(4)) = \#False$
by (*simp add: Valid-def len-defs*)
have 10: $\vdash (len(7) \vee (len(10) \wedge len(4))) = len(7)$
using 9 **by** auto
have 11: $\vdash ((\triangleright (len(5)) ; (len(5) \vee len(2))) \wedge (\triangleright (len(2)) ; (len(5) \vee len(2)))) =$
 $(len(5); (len(5) \vee len(2))) \wedge len(2); (len(5) \vee len(2))$
using 1 2 **by** fastforce
have 12: $\vdash (len(5); (len(5) \vee len(2))) \wedge len(2); (len(5) \vee len(2)) = len(7)$
by (*metis 10 4 6 7 8 ChopOrEqv inteq-reflection*)
from 11 12 **show** ?thesis **by** fastforce
qed

end

16 Monitors

theory *Monitor*
imports *First*

begin

The RV monitors language is introduced plus the algebraic properties of the monitor operators.

16.1 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 \Rightarrow 'a formula*

where (*MON (FIRST f)*) = *LIFT*(\triangleright *f*)
| (*MON (a UPTO b)*) = *LIFT*(\triangleright ((*MON a*) \vee (*MON b*)))
| (*MON (a THRU b)*) = *LIFT*(\triangleright (*di*(*MON a*) \wedge *di*(*MON b*)))
| (*MON (a THEN b)*) = *LIFT*((*MON a*) \frown (*MON b*))
| (*MON (a WITH f)*) = *LIFT*((*MON a*) \wedge *f*)

syntax

-MON :: 'a monitor \Rightarrow lift ((\mathcal{M} -) [80] 80)

translations

-MON == CONST MON

definition *eq-d :: ('a :: world) monitor \Rightarrow 'a monitor \Rightarrow bool* ((*- \simeq -*) [84,84] 83)

where

eq-d a b \equiv (\vdash (\mathcal{M} a) = (\mathcal{M} b))

lemma *MonEqRefl:*

a \simeq 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*)

16.2 Derived Monitors

definition *HALT-d* :: $('a :: \text{world}) \text{ formula} \Rightarrow 'a \text{ monitor}$

where $\text{HALT-d } w \equiv \text{FIRST}(\text{LIFT}(\text{fin } (\text{init } w)))$

definition *LEN-d* :: $\text{nat} \Rightarrow ('a :: \text{world}) \text{ monitor}$

where

$\text{LEN-d } k \equiv \text{FIRST}(\text{LIFT}(\text{len } k))$

definition *EMPTY-d* :: $('a :: \text{world}) \text{ monitor}$

where

$\text{EMPTY-d} \equiv \text{FIRST}(\text{LIFT}(\text{empty}))$

definition *SKIP-d* :: $('a :: \text{world}) \text{ monitor}$

where

$\text{SKIP-d} \equiv \text{FIRST}(\text{LIFT}(\text{skip}))$

syntax

$\text{-HALT-d} :: \text{lift} \Rightarrow 'a \text{ monitor} \quad ((\text{HALT } -) [84] \ 83)$

$\text{-LEN-d} :: \text{nat} \Rightarrow 'a \text{ monitor} \quad ((\text{LEN } -) [84] \ 83)$

$\text{-EMPTY-d} :: 'a \text{ monitor} \quad ((\text{EMPTY}))$

$\text{-SKIP-d} :: 'a \text{ monitor} \quad ((\text{SKIP}))$

syntax (*ASCII*)

-*HALT-d* :: *lift* \Rightarrow '*a monitor* ((*HALT -*) [84] 83)
-*LEN-d* :: *nat* \Rightarrow '*a monitor* ((*LEN -*) [84] 83)
-*EMPTY-d* :: '*a monitor* ((*EMPTY*))
-*SKIP-d* :: '*a monitor* ((*SKIP*))

translations

-*HALT-d* \equiv *CONST HALT-d*
-*LEN-d* \equiv *CONST LEN-d*
-*EMPTY-d* \equiv *CONST EMPTY-d*
-*SKIP-d* \equiv *CONST SKIP-d*

definition *GUARD-d* :: ('*a::world*) *formula* \Rightarrow '*a monitor*

where

GUARD-d w \equiv (*EMPTY WITH LIFT*(*init w*))

primrec *TIMES-d* :: ('*a :: world*) *monitor* \Rightarrow *nat* \Rightarrow '*a 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 monitor* ((*GUARD -*) [84] 83)
-*TIMES-d* :: ['*a monitor, nat*] \Rightarrow '*a monitor* ((- *TIMES -*) [84,84] 83)

syntax (*ASCII*)

-*GUARD-d* :: *lift* \Rightarrow '*a monitor* ((*GUARD -*) [84] 83)
-*TIMES-d* :: ['*a monitor, nat*] \Rightarrow '*a monitor* ((- *TIMES -*) [84,84] 83)

translations

-*GUARD-d* \equiv *CONST GUARD-d*
-*TIMES-d* \equiv *CONST TIMES-d*

definition *FAIL-d* :: ('*a:: world*) *monitor*

where

FAIL-d \equiv *GUARD* (#*False*)

definition *ALWAYS-d* :: ('*a :: world*) *monitor* \Rightarrow '*a formula* \Rightarrow '*a monitor*

where

ALWAYS-d a w \equiv (*a WITH LIFT*((*bi* (*finite* \longrightarrow *fin* (*init w*))))))

definition *SOMETIME-d* :: ('*a :: world*) *monitor* \Rightarrow '*a formula* \Rightarrow '*a monitor*

where

SOMETIME-d a w \equiv (*a WITH LIFT*((*di* (*finite* \wedge *fin* (*init w*))))))

definition *LIMIT-d* :: ('*a :: world*) *formula* \Rightarrow '*a formula*

where

$LIMIT-d\ f \equiv LIFT(bs\ (\neg f))$

definition $UNTIL-d :: ('a :: world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ monitor$
where

$UNTIL-d\ w1\ w2 \equiv (HALT\ w2)\ WITH\ (LIFT(bm\ w1))$

syntax

- $FAIL-d :: 'a\ monitor \quad (FAIL)$
- $ALWAYS-d :: ['a\ monitor, lift] \Rightarrow 'a\ monitor\ ((- ALWAYS -)\ [84,84]\ 83)$
- $SOMETIME-d :: ['a\ monitor, lift] \Rightarrow 'a\ monitor\ ((- SOMETIME -)\ [84,84]\ 83)$
- $LIMIT-d :: lift \Rightarrow lift \quad ((Limit -)\ [84]\ 83)$
- $UNTIL-d :: [lift, lift] \Rightarrow 'a\ monitor \quad ((- UNTIL -)\ [84,84]\ 83)$

syntax (ASCII)

- $FAIL-d :: 'a\ monitor \quad (FAIL)$
- $ALWAYS-d :: ['a\ monitor, lift] \Rightarrow 'a\ monitor\ ((- ALWAYS -)\ [84,84]\ 83)$
- $SOMETIME-d :: ['a\ monitor, lift] \Rightarrow 'a\ monitor\ ((- SOMETIME -)\ [84,84]\ 83)$
- $LIMIT-d :: lift \Rightarrow lift \quad ((Limit -)\ [84]\ 83)$
- $UNTIL-d :: [lift, lift] \Rightarrow 'a\ monitor \quad ((- UNTIL -)\ [84,84]\ 83)$

translations

- $FAIL-d \Rightarrow CONST\ FAIL-d$
- $ALWAYS-d \Rightarrow CONST\ ALWAYS-d$
- $SOMETIME-d \Rightarrow CONST\ SOMETIME-d$
- $LIMIT-d \Rightarrow CONST\ LIMIT-d$
- $UNTIL-d \Rightarrow CONST\ UNTIL-d$

definition $WITHIN-d :: ('a :: world)\ monitor \Rightarrow 'a\ formula \Rightarrow 'a\ monitor$
where

$WITHIN-d\ a\ f \equiv (a\ WITH\ LIFT(Limit\ f))$

syntax

- $WITHIN-d :: ['a\ monitor, lift] \Rightarrow 'a\ monitor\ ((- WITHIN -)\ [84,84]\ 83)$

syntax (ASCII)

- $WITHIN-d :: ['a\ monitor, lift] \Rightarrow 'a\ monitor\ ((- WITHIN -)\ [84,84]\ 83)$

translations

- $WITHIN-d \Rightarrow CONST\ WITHIN-d$

definition $AND-d :: ('a :: world)\ monitor \Rightarrow 'a\ monitor \Rightarrow 'a\ monitor$
where

$AND-d\ a\ b \equiv (a\ WITH\ LIFT(\mathcal{M}\ b))$

definition $ITERATE-d :: ('a :: world)\ monitor \Rightarrow 'a\ monitor \Rightarrow 'a\ monitor$
where

$ITERATE-d\ a\ b \equiv (a\ WITH\ (LIFT\ (schopstar\ (\mathcal{M}\ b))))$

syntax

-AND-d :: ['a monitor, 'a monitor] \Rightarrow 'a monitor ((- AND -) [84,84] 83)
 -ITERATE-d :: ['a monitor, 'a monitor] \Rightarrow 'a monitor ((- ITERATE -) [84,84] 83)

syntax (ASCII)

-AND-d :: ['a monitor, 'a monitor] \Rightarrow 'a monitor ((- AND -) [84,84] 83)
 -ITERATE-d :: ['a monitor, 'a monitor] \Rightarrow 'a monitor ((- ITERATE -) [84,84] 83)

translations

-AND-d \Rightarrow CONST AND-d
 -ITERATE-d \Rightarrow CONST ITERATE-d

definition STAR-d :: ('a :: world) monitor \Rightarrow 'a formula \Rightarrow 'a monitor

where

STAR-d a f \equiv ((FIRST LIFT(\Diamond f)) ITERATE (a))

definition REPEAT-d :: ('a :: world) monitor \Rightarrow 'a formula \Rightarrow 'a monitor

where

REPEAT-d a w \equiv ((HALT w) ITERATE (a WITH LIFT(keep(\neg (init w))))))

syntax

-STAR-d :: ['a monitor, lift] \Rightarrow 'a monitor ((- STAR -) [84,84] 83)
 -REPEAT-d :: ['a monitor, lift] \Rightarrow 'a monitor ((- REPEATUNTIL -) [84,84] 83)

syntax (ASCII)

-STAR-d :: ['a monitor, lift] \Rightarrow 'a monitor ((- STAR -) [84,84] 83)
 -REPEAT-d :: ['a monitor, lift] \Rightarrow 'a monitor ((- REPEATUNTIL -) [84,84] 83)

translations

-STAR-d \Rightarrow CONST STAR-d
 -REPEAT-d \Rightarrow CONST REPEAT-d

16.3 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: $\vdash (\mathcal{M} (\text{FIRST } x)) = \triangleright x$ **by** simp

have 2: $\vdash \triangleright x = \triangleright (\triangleright x)$ **using** FstFixFst **by** fastforce

have 3: $\vdash \triangleright (\triangleright x) = \triangleright (\mathcal{M} (\text{FIRST } x))$ **by** simp

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

qed

next

case (mUPTO-d a1 a2)

```

then show ?case
proof -
have 1:  $\vdash (\mathcal{M} (a1 \text{ UPTO } a2)) = \triangleright (\mathcal{M} a1) \vee (\mathcal{M} a2)$ 
  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:  $\vdash \triangleright (\triangleright ((\mathcal{M} a1) \vee (\mathcal{M} a2))) = \triangleright (\mathcal{M} (a1 \text{ UPTO } a2))$ 
  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:  $\vdash (\mathcal{M} (a1 \text{ THRU } a2)) = \triangleright (di (\mathcal{M} a1) \wedge di (\mathcal{M} a2))$ 
  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:  $\vdash \triangleright (\triangleright (di (\mathcal{M} a1) \wedge di (\mathcal{M} a2))) = \triangleright (\mathcal{M} (a1 \text{ THRU } a2))$ 
  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:  $\vdash (\mathcal{M} (a1 \text{ THEN } a2)) = (\mathcal{M} a1) \frown (\mathcal{M} a2)$ 
  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:  $\vdash \triangleright ((\mathcal{M} a1) \frown (\mathcal{M} a2)) = \triangleright (\mathcal{M} (a1 \text{ THEN } a2))$ 
  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*:

$\vdash \mathcal{M}(GUARD \ \#False) = \#False$

proof –

```

    have 1:  $\vdash \mathcal{M}(GUARD \ \#False) = \mathcal{M}(EMPTY \text{ WITH } LIFT(init \ \#False))$  by (simp add: GUARD-d-def)
    have 2:  $\vdash \mathcal{M}(EMPTY \text{ WITH } LIFT(init \ \#False)) = (\mathcal{M}(EMPTY) \wedge (init \ \#False))$  by (simp )
    have 3:  $\vdash \#False = (init \ \#False)$  by (simp add: init-defs Valid-def)
    have 4:  $\vdash (\mathcal{M}(EMPTY) \wedge (init \ \#False)) = (\mathcal{M}(EMPTY) \wedge \#False)$  using 3 by auto
    have 5:  $\vdash (\mathcal{M}(EMPTY) \wedge \#False) = \#False$  by simp
    have 6:  $\vdash (\mathcal{M}(EMPTY) \wedge (init \ \#False)) = \#False$  using 4 5 by simp
    have 7:  $\vdash \mathcal{M}(EMPTY \text{ WITH } LIFT(init \ \#False)) = \#False$  using 2 6 by fastforce
    have 8:  $\vdash \mathcal{M}(GUARD \ \#False) = \#False$  using 1 7 by fastforce
    from 8 show ?thesis by auto

```

qed

lemma *MFirstFalseEqvFalse*:

$\vdash \mathcal{M}(FIRST \ LIFT \ \#False) = \#False$

proof –

```

    have 1:  $\vdash \mathcal{M}(FIRST \ LIFT \ \#False) = \triangleright \ \#False$  by (simp )
    have 2:  $\vdash \mathcal{M}(FIRST \ LIFT \ \#False) = \#False$  using FstFalse by fastforce
    from 2 show ?thesis by auto

```

qed

lemma *MFailAlt*:

$\vdash \mathcal{M} \ FAIL = \#False$

proof –

```

    have 1:  $\vdash \mathcal{M} \ FAIL = \mathcal{M} \ (GUARD \ (\#False))$  by (simp add: FAIL-d-def)
    have 2:  $\vdash \mathcal{M}(GUARD \ (\#False)) = \#False$  using MGuardFalseEqvFalse by auto
    from 1 2 show ?thesis by fastforce

```

qed

lemma *MFailEqvFirstFalseWithinEmpty*:

$FAIL \simeq ((FIRST \ LIFT \ \#False) \text{ WITHIN } empty)$

proof –

```

    have 1:  $\vdash \mathcal{M} \ ((FIRST \ LIFT \ \#False) \text{ WITHIN } (empty)) =$ 
       $\mathcal{M}((FIRST \ LIFT \ \#False) \text{ WITH } LIFT(Limit \ empty))$ 
      by (simp add: WITHIN-d-def)
    have 2:  $\vdash \mathcal{M}((FIRST \ LIFT \ \#False) \text{ WITH } LIFT(Limit \ empty)) =$ 
       $(\mathcal{M}(FIRST \ LIFT \ \#False) \wedge (Limit \ empty))$ 
      by (simp )
    have 3:  $\vdash \mathcal{M}((FIRST \ LIFT \ \#False) \text{ WITH } LIFT(Limit \ empty)) = \#False$ 
      using MFirstFalseEqvFalse by auto
    have 4:  $\vdash \mathcal{M} \ ((FIRST \ LIFT \ \#False) \text{ WITHIN } (empty)) = \#False$ 
      using 1 3 by fastforce

```


have 5: $\vdash \mathcal{M}(\text{FAIL}) = \#False$
using *MFailAlt* **by** *simp*
from 4 5 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MEEmptyAlt*:

$\vdash \mathcal{M} \text{ EMPTY} = \text{empty}$

proof –

have 1: $\vdash \mathcal{M}(\text{EMPTY}) = \mathcal{M}(\text{FIRST LIFT empty})$ **by** (*simp add: EMPTY-d-def*)
have 2: $\vdash \mathcal{M}(\text{FIRST LIFT empty}) = \triangleright \text{empty}$ **by** (*simp*)
have 3: $\vdash \triangleright \text{empty} = \text{empty}$ **using** *FstEmpty* **by** *auto*
from 1 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *MSkipAlt*:

$\vdash \mathcal{M} \text{ SKIP} = \text{skip}$

proof –

have 1: $\vdash \mathcal{M} \text{ SKIP} = \mathcal{M}(\text{FIRST LIFT skip})$ **by** (*simp add: SKIP-d-def*)
have 2: $\vdash \mathcal{M}(\text{FIRST LIFT skip}) = \triangleright \text{skip}$ **by** (*simp*)
have 3: $\vdash \triangleright \text{skip} = \text{skip}$ **using** *FstSkip* **by** *simp*
from 1 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *MGuardAlt*:

$\vdash \mathcal{M}(\text{GUARD}(w)) = (\text{empty} \wedge \text{init } w)$

proof –

have 1: $\vdash \mathcal{M}(\text{GUARD}(w)) = \mathcal{M}(\text{EMPTY WITH } (\text{LIFT } (\text{init } w)))$ **by** (*simp add: GUARD-d-def*)
have 2: $\vdash \mathcal{M}(\text{EMPTY WITH } (\text{LIFT } (\text{init } w))) = (\mathcal{M}(\text{EMPTY}) \wedge (\text{init } w))$ **by** (*simp*)
have 3: $\vdash (\mathcal{M}(\text{EMPTY}) \wedge (\text{init } w)) = (\text{empty} \wedge (\text{init } w))$ **using** *MEEmptyAlt* **by** *fastforce*
have 4: $\vdash (\text{empty} \wedge (\text{init } w)) = (\text{empty} \wedge \text{init } w)$ **by** *simp*
from 1 2 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *MLengthAlt*:

$\vdash \mathcal{M}(\text{LEN}(k)) = \text{len}(k)$

proof –

have 1: $\vdash \mathcal{M}(\text{LEN}(k)) = \mathcal{M}(\text{FIRST LIFT}(\text{len}(k)))$ **by** (*simp add: LEN-d-def*)
have 2: $\vdash \mathcal{M}(\text{FIRST LIFT}(\text{len}(k))) = \triangleright(\text{len}(k))$ **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 *MAlwaysAlt*:

$\vdash \mathcal{M}(a \text{ ALWAYS } w) = (\mathcal{M}(a) \wedge \Box (\text{init } w))$

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: $\vdash \mathcal{M}(a \text{ SOMETIME } w) = \mathcal{M}(a \text{ WITH LIFT}(di (finite \wedge fin (init w))))$

by (*simp add: SOMETIME-d-def*)

have 2: $\vdash \mathcal{M}(a \text{ WITH LIFT}(di (finite \wedge fin (init w)))) = (\mathcal{M}(a) \wedge (di (finite \wedge fin (init w))))$

by (*simp*)

have 3: $\vdash \mathcal{M}(a \text{ WITH LIFT}(di (finite \wedge fin (init w)))) = (\mathcal{M}(a) \wedge \Diamond (\text{init } w))$

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*:

$\vdash \mathcal{M}(a \text{ TIMES } k) = fpower (\mathcal{M}(a)) k$

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: $\vdash \text{empty} = fpower (\mathcal{M} a) 0$ 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*:

$\vdash \mathcal{M}(\text{HALT } w) = (\text{halt}(\text{init } w) \wedge \text{finite})$
 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: $\vdash \triangleright(\text{fin}(\text{init } w)) = (\text{halt}(\text{init } w) \wedge \text{finite})$ 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 MEmptyUpto:

$(\text{EMPTY UPTO } a) \simeq \text{EMPTY}$

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** MEmptyAlt **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** MEmptyAlt **by** auto

from 1 4 5 6 **show** ?thesis **using** MonEq **by** (metis int-eq)

qed

lemma MEmptyThru:

$(\text{EMPTY THRU } a) \simeq (a)$

proof –

have 1: $\vdash \mathcal{M}(\text{EMPTY THRU } a) = \triangleright(di(\mathcal{M} \text{ EMPTY}) \wedge di(\mathcal{M} a))$ **by** (simp)

have 2: $\vdash di(\mathcal{M} \text{ EMPTY}) = di \text{ empty}$ **using** MEmptyAlt DiEqvDi **by** blast

hence 3: $\vdash (di(\mathcal{M} \text{ EMPTY}) \wedge di(\mathcal{M} a)) = (di \text{ empty} \wedge di(\mathcal{M} a))$ **by** auto

hence 4: $\vdash (di \text{ empty} \wedge di(\mathcal{M} a)) = di(\mathcal{M} a)$ **using** DiEmpty **by** auto

have 5: $\vdash (di(\mathcal{M} \text{ EMPTY}) \wedge di(\mathcal{M} a)) = di(\mathcal{M} a)$ **using** 3 4 **by** fastforce

hence 6: $\vdash \triangleright(di(\mathcal{M} \text{ EMPTY}) \wedge di(\mathcal{M} a)) = \triangleright(di(\mathcal{M} a))$ **using** FstEqvRule **by** blast

have 7: $\vdash \triangleright(di(\mathcal{M} a)) = \triangleright(\mathcal{M} a)$ **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:

$(a \text{ THEN EMPTY}) \simeq (a \text{ WITH } (\text{LIFT } \text{finite}))$

proof –

have 1: $\vdash \mathcal{M}(a \text{ THEN EMPTY}) = (\mathcal{M} a) \frown (\mathcal{M} \text{ EMPTY})$ **by** (simp)

have 2: $\vdash (\mathcal{M} a) \frown (\mathcal{M} \text{ EMPTY}) = (\mathcal{M} a) \frown \text{empty}$ **by** (simp add: MEmptyAlt RightSChopEqvSChop)

have 3: $\vdash (\mathcal{M} a) \frown \text{empty} = ((\mathcal{M} a) \wedge \text{finite})$ **by** (simp add: ChopEmpty schop-d-def)

from 1 2 3 **show** ?thesis **using** MonEq **by** (metis MON.simps(5) inteq-reflection)

qed

lemma *MEEmptyThen*:

(*EMPTY THEN a*) \simeq *a*

proof –

have 1: $\vdash \mathcal{M}(\text{EMPTY THEN } a) = (\mathcal{M} \text{ EMPTY}) \frown (\mathcal{M} a)$ **by** (*simp*)

have 2: $\vdash (\mathcal{M} \text{ EMPTY}) \frown (\mathcal{M} a) = \text{empty} \frown (\mathcal{M} a)$ **by** (*simp add: MEmptyAlt LeftSChopEqvSChop*)

have 3: $\vdash \text{empty} \frown (\mathcal{M} a) = (\mathcal{M} a)$ **by** (*simp add: EmptySChop*)

from 1 2 3 **show** ?thesis **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MEEmptyIterate*:

(*EMPTY ITERATE b*) \simeq *EMPTY*

proof –

have 1: $\vdash \mathcal{M}(\text{EMPTY ITERATE } b) = \mathcal{M}(\text{EMPTY WITH LIFT } (\text{schopstar } (\mathcal{M} b)))$

by (*simp add: ITERATE-d-def*)

have 2: $\vdash \mathcal{M}(\text{EMPTY WITH LIFT } (\text{schopstar } (\mathcal{M} b))) = (\mathcal{M} \text{ EMPTY} \wedge (\text{schopstar } (\mathcal{M} b)))$

by (*simp*)

have 3: $\vdash (\mathcal{M} \text{ EMPTY} \wedge (\text{schopstar } (\mathcal{M} b))) = (\text{empty} \wedge (\text{schopstar } (\mathcal{M} b)))$

using *MEEmptyAlt* **by** *auto*

have 4: $\vdash (\text{empty} \wedge (\text{schopstar } (\mathcal{M} b))) = (\text{empty} \wedge (\text{empty} \vee (((\mathcal{M} b) \wedge \text{more}) \frown (\text{schopstar } (\mathcal{M} b)))))$

using *SChopstarEqv* **by** *fastforce*

have 5: $\vdash (\text{empty} \wedge (\text{empty} \vee (((\mathcal{M} b) \wedge \text{more}) \frown (\text{schopstar } (\mathcal{M} b)))) = \text{empty}$

by *auto*

have 6: $\vdash \mathcal{M}(\text{EMPTY ITERATE } b) = \mathcal{M} \text{ EMPTY}$

using 1 2 3 4 5 *MEEmptyAlt* **by** *fastforce*

from 6 **show** ?thesis **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MIterateIdemp*:

(*a ITERATE a*) \simeq (*a*)

proof –

have 1: $\vdash \mathcal{M}(a \text{ ITERATE } a) = \mathcal{M}(a \text{ WITH LIFT } (\text{schopstar } (\mathcal{M} a)))$ **by** (*simp add: ITERATE-d-def*)

have 2: $\vdash \mathcal{M}(a \text{ WITH LIFT } (\text{schopstar } (\mathcal{M} a))) = ((\mathcal{M} a) \wedge (\text{schopstar } (\mathcal{M} a)))$ **by** (*simp*)

have 3: $\vdash ((\mathcal{M} a) \wedge (\text{schopstar } (\mathcal{M} a))) = (\triangleright(\mathcal{M} a) \wedge (\text{schopstar } (\triangleright(\mathcal{M} a))))$ **using** *MFxFst*

by (*metis 2 integ-reflection*)

have 4: $\vdash (\triangleright(\mathcal{M} a) \wedge (\text{schopstar } (\triangleright(\mathcal{M} a)))) = \triangleright(\mathcal{M} a)$ **using** *FstAndFstStarEqvFst* **by** *fastforce*

have 5: $\vdash \triangleright(\mathcal{M} a) = \mathcal{M} a$ **using** *MFxFst* **by** *fastforce*

from 1 2 3 4 5 **show** ?thesis **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MUptoIdemp*:

(*a 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** *MFxFst* **by** *fastforce*

from 1 2 3 **show** ?thesis **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MThruIdemp*:

(*a 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: $\vdash \mathcal{M}(a \text{ THRU } b) = \triangleright(\text{di}(\mathcal{M} a) \wedge \text{di}(\mathcal{M} b))$ **by** (*simp*)
 have 2: $\vdash (\text{di}(\mathcal{M} a) \wedge \text{di}(\mathcal{M} b)) = (\text{di}(\mathcal{M} b) \wedge \text{di}(\mathcal{M} a))$ **by** *auto*
hence 3: $\vdash \triangleright(\text{di}(\mathcal{M} a) \wedge \text{di}(\mathcal{M} b)) = \triangleright(\text{di}(\mathcal{M} b) \wedge \text{di}(\mathcal{M} a))$ **using** *FstEqvRule* **by** *blast*
 have 4: $\vdash \triangleright(\text{di}(\mathcal{M} b) \wedge \text{di}(\mathcal{M} a)) = \mathcal{M}(b \text{ THRU } a)$ **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*:

$\vdash \triangleright f = \triangleright (f \wedge \text{finite})$

by (*metis FstDfEqvFst FstDiEqvFst di-d-def inteq-reflection itl-def(15) schop-d-def*)

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 *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 *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: $\vdash di(\triangleright (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) = di((di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \wedge \text{finite})$

using *DiAndFiniteEqvDiFst* **by** *fastforce*

have 3: $\vdash di((di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \wedge \text{finite}) = (di((\mathcal{M}\ a) \wedge \text{finite}) \wedge di((\mathcal{M}\ b) \wedge \text{finite}))$

by (*metis DfDfAndEqvDf DiAndFiniteEqvDiFst MFixFst df-d-def di-d-def inteq-reflection schop-d-def*)

have 4: $\vdash di(\triangleright (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) = (di((\mathcal{M}\ a) \wedge \text{finite}) \wedge di((\mathcal{M}\ b) \wedge \text{finite}))$

using 2 3 **by** *fastforce*

hence 5: $\vdash (di(\triangleright (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) \wedge di((\mathcal{M}\ c) \wedge \text{finite})) =$

$(di((\mathcal{M}\ a) \wedge \text{finite}) \wedge (di((\mathcal{M}\ b) \wedge \text{finite}) \wedge di((\mathcal{M}\ c) \wedge \text{finite})))$

by *auto*

have 6: $\vdash (di((\mathcal{M}\ b) \wedge \text{finite}) \wedge di((\mathcal{M}\ c) \wedge \text{finite})) = di((di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)) \wedge \text{finite})$

by (*metis DfDfAndEqvDf DfEqvDiAndFinite DiAndFiniteEqvDiFst MFixFst inteq-reflection*)

have 7: $\vdash di((di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)) \wedge \text{finite}) = di(\triangleright (di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)))$

by (*simp add: DiAndFiniteEqvDiFst*)

have 8: $\vdash (di((\mathcal{M}\ b) \wedge \text{finite}) \wedge di((\mathcal{M}\ c) \wedge \text{finite})) =$

$di(\triangleright (di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)))$

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: $\vdash (di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} c)))) =$
 $(di(((\mathcal{M} a) \vee (\mathcal{M} b)) \wedge \text{finite}) \wedge di(((\mathcal{M} a) \vee (\mathcal{M} c)) \wedge \text{finite})))$

using DiAndFiniteEqvDiFst by fastforce

have 3: $\vdash (di(((\mathcal{M} a) \vee (\mathcal{M} b)) \wedge \text{finite}) \wedge di(((\mathcal{M} a) \vee (\mathcal{M} c)) \wedge \text{finite})) =$
 $((di((\mathcal{M} a) \wedge \text{finite}) \vee di((\mathcal{M} b) \wedge \text{finite})) \wedge (di((\mathcal{M} a) \wedge \text{finite}) \vee di((\mathcal{M} c) \wedge \text{finite})))$

by (metis (no-types, lifting) DiDiAndEqvDi DiOrEqv FiniteOr inteq-reflection)

have 4: $\vdash ((di((\mathcal{M} a) \wedge \text{finite}) \vee di((\mathcal{M} b) \wedge \text{finite})) \wedge (di((\mathcal{M} a) \wedge \text{finite}) \vee di((\mathcal{M} c) \wedge \text{finite}))) =$
 $(di((\mathcal{M} a) \wedge \text{finite}) \vee (di((\mathcal{M} b) \wedge \text{finite}) \wedge di((\mathcal{M} c) \wedge \text{finite})))$

by auto

have 5: $\vdash (di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} c)))) =$
 $(di((\mathcal{M} a) \wedge \text{finite}) \vee (di((\mathcal{M} b) \wedge \text{finite}) \wedge di((\mathcal{M} c) \wedge \text{finite})))$

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: $\vdash (di((\mathcal{M} a) \wedge \text{finite}) \wedge (di((\mathcal{M} b) \wedge \text{finite}) \vee di((\mathcal{M} c) \wedge \text{finite}))) =$
 $(di((\mathcal{M} a) \wedge \text{finite}) \wedge di((\mathcal{M} b) \vee (\mathcal{M} c)) \wedge \text{finite})$ by (*metis 3 FiniteOr inteq-reflection*)
 using *DiOrEqv*
 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: $\vdash (\mathcal{M}(a \text{ WITH } f) \wedge \mathcal{M}(b \text{ WITH } f)) = \mathcal{M}((a \text{ WITH } f) \text{ WITH LIFT}(\mathcal{M}(b \text{ WITH } f)))$
 by simp
 have 8: $\vdash \mathcal{M}((a \text{ WITH } f) \text{ WITH LIFT}(\mathcal{M}(b \text{ WITH } f))) = \mathcal{M}((a \text{ WITH } f) \text{ AND } (b \text{ WITH } f))$
 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*:

$$(((\text{HALT } w) \text{ WITH } f) \text{ AND } ((\text{HALT } w) \text{ WITH } g)) \simeq ((\text{HALT } w) \text{ WITH LIFT}(f \wedge g))$$

proof –

have 1: $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ AND } ((\text{HALT } w) \text{ WITH } g)) =$
 $\mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ WITH LIFT}(\mathcal{M}((\text{HALT } w) \text{ WITH } g)))$
by (*simp add: AND-d-def*)
have 2: $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ WITH LIFT}(\mathcal{M}((\text{HALT } w) \text{ WITH } g))) =$
 $(\mathcal{M}(\text{HALT } w) \wedge f \wedge \mathcal{M}(\text{HALT } w) \wedge g)$
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: $\vdash (\mathcal{M}(\text{HALT } w) \wedge f \wedge g) = \mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \wedge g))$
by *auto*
from 1 2 3 4 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MHaltWithUptoHaltWithEqvHaltWithOr*:

$$(((\text{HALT } w) \text{ WITH } f) \text{ UPTO } ((\text{HALT } w) \text{ WITH } g)) \simeq ((\text{HALT } w) \text{ WITH LIFT}(f \vee g))$$

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: $\vdash \triangleright(\mathcal{M}(\text{HALT } w) \wedge (f \vee g)) = \triangleright(\mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \vee g)))$
by *simp*
have 6: $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH LIFT}(f \vee g))) = \triangleright(\mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \vee g)))$
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: $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ THRU } ((\text{HALT } w) \text{ WITH } g)) =$
 $\triangleright(\text{di}(\mathcal{M}(\text{HALT } w) \wedge f) \wedge \text{di}(\mathcal{M}(\text{HALT } w) \wedge g))$
by *simp*
have 2: $\vdash (\text{di}(\mathcal{M}(\text{HALT } w) \wedge f) \wedge \text{di}(\mathcal{M}(\text{HALT } w) \wedge g)) =$
 $(\text{di}((\text{halt}(\text{init } w) \wedge \text{finite}) \wedge f) \wedge \text{di}((\text{halt}(\text{init } w) \wedge \text{finite}) \wedge g))$
using *MHaltAlt* **by** (*metis DiDiAndEqvDi inteq-reflection*)
have 3: $\vdash (\text{di}((\text{halt}(\text{init } w) \wedge \text{finite}) \wedge f) \wedge \text{di}((\text{halt}(\text{init } w) \wedge \text{finite}) \wedge g)) =$
 $\text{di}((\text{halt}(\text{init } w) \wedge \text{finite}) \wedge f \wedge g)$
by (*metis FstDiamondStateEqvHalt LFstAndDistrD inteq-reflection*)

have 4: $\vdash di((halt(init\ w) \wedge finite) \wedge f \wedge g) = di(\mathcal{M}(HALT\ w) \wedge f \wedge g)$
by (*metis 3 MHaltAlt inteq-reflection*)
have 5: $\vdash (di(\mathcal{M}(HALT\ w) \wedge f) \wedge di(\mathcal{M}(HALT\ w) \wedge g)) = di(\mathcal{M}(HALT\ w) \wedge f \wedge g)$
using 2 3 4 **by** *fastforce*
have 6: $\vdash \triangleright(di(\mathcal{M}(HALT\ w) \wedge f) \wedge di(\mathcal{M}(HALT\ w) \wedge g)) = \triangleright(di(\mathcal{M}(HALT\ w) \wedge f \wedge g))$
using 5 *FstEqvRule* **by** *blast*
have 7: $\vdash \triangleright(di(\mathcal{M}(HALT\ w) \wedge f \wedge g)) = \triangleright(\mathcal{M}(HALT\ w) \wedge f \wedge g)$
using *FstDiEqvFst* **by** *fastforce*
have 8: $\vdash \triangleright(\mathcal{M}(HALT\ w) \wedge f \wedge g) = \triangleright(\mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)))$
by *simp*
have 9: $\vdash \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)) = \triangleright(\mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)))$
using *MFixFst* **by** *blast*
have 10: $\vdash \mathcal{M}((HALT\ w)\ WITH\ f)\ THRU\ ((HALT\ w)\ WITH\ g) = \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g))$
using 1 2 3 4 5 6 7 8 9 *int-eq* **by** *metis*
have 11: $\vdash \mathcal{M}((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g) = \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g))$
using *MHaltWithAndDistrib* **using** *eq-d-def* **by** *blast*
have 12: $\vdash \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)) = \mathcal{M}((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g)$
using 11 **by** *fastforce*
from 10 12 show ?thesis using MonEq by (metis int-eq)
qed

lemma *MThenAndDistrib*:

$(a\ THEN\ (b\ AND\ c)) \simeq ((a\ THEN\ b)\ AND\ (a\ THEN\ c))$

proof –

have 1: $\vdash \mathcal{M}(a\ THEN\ (b\ AND\ c)) = (\mathcal{M}(a)) \frown (\mathcal{M}(b\ AND\ c))$
by *simp*
have 2: $\vdash (\mathcal{M}(a)) \frown (\mathcal{M}(b\ AND\ c)) = (\mathcal{M}(a)) \frown (\mathcal{M}(b) \wedge \mathcal{M}(c))$
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: $\vdash ((\mathcal{M}(a)) \frown (\mathcal{M}(b))) \wedge ((\mathcal{M}(a)) \frown (\mathcal{M}(c))) =$
 $(\mathcal{M}(a\ THEN\ b) \wedge \mathcal{M}(a\ THEN\ c))$
by *simp*
have 7: $\vdash (\mathcal{M}(a\ THEN\ b) \wedge \mathcal{M}(a\ THEN\ c)) = \mathcal{M}((a\ THEN\ b)\ AND\ (a\ THEN\ c))$
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*:

$(a\ THEN\ (b\ UPTO\ c)) \simeq ((a\ THEN\ b)\ UPTO\ (a\ THEN\ c))$

proof –

have 1: $\vdash (\mathcal{M}(a\ THEN\ (b\ UPTO\ c))) = ((\mathcal{M}\ a) \frown (\triangleright((\mathcal{M}\ b) \vee (\mathcal{M}\ c))))$
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: $\vdash (\mathcal{M} a) \frown di(\mathcal{M} c) = (\mathcal{M} a) \frown ((\mathcal{M} c) \frown \# \text{True})$
 by (*metis DiAndFiniteEqvDiFst MFixFst di-d-def int-simps(20) inteq-reflection schop-d-def*)
 have 11: $\vdash (\mathcal{M} a) \frown ((\mathcal{M} c) \frown \# \text{True}) = ((\mathcal{M} a) \frown (\mathcal{M} c)) \frown \# \text{True}$
 by (*simp add: SCHopAssoc*)
 have 12: $\vdash ((\mathcal{M} a) \frown (\mathcal{M} c)) \frown \# \text{True} = di((\mathcal{M} a) \frown (\mathcal{M} c))$
 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:  $\vdash \triangleright (\mathcal{M} a) \neg (di(\mathcal{M} b) \wedge di(\mathcal{M} c)) = (di(\mathcal{M}(a \text{ THEN } b)) \wedge di(\mathcal{M}(a \text{ THEN } c)))$ 
    using 15 14 4 5 by fastforce
  have 17:  $\vdash \triangleright (\triangleright (\mathcal{M} a) \neg (di(\mathcal{M} b) \wedge di(\mathcal{M} c))) = \triangleright (di(\mathcal{M}(a \text{ THEN } b)) \wedge di(\mathcal{M}(a \text{ THEN } c)))$ 
    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

17 Monitor Example

```

theory MonitorExample
imports
  FOTheorems Monitor
begin

```

```

locale Test =
  fixes v :: state  $\Rightarrow$  nat
  fixes y :: state  $\Rightarrow$  bool
  fixes z :: state  $\Rightarrow$  nat
  fixes F2 :: nat statefun  $\Rightarrow$  temporal
  fixes F3 :: bool statefun  $\Rightarrow$  temporal
  fixes F4 :: nat statefun  $\Rightarrow$  temporal
  fixes F5 :: nat statefun  $\Rightarrow$  temporal
  fixes Init2 :: nat statefun  $\Rightarrow$  temporal
  fixes Init3 :: bool statefun  $\Rightarrow$  temporal
  fixes Mon1 :: state monitor
  fixes Mon2 :: state monitor
  fixes Mon3 :: state monitor
  fixes Mon4 :: state monitor
  fixes Mon5 :: state monitor
  fixes Mon6 :: state monitor
  defines F2  $\equiv (\lambda v. \text{TEMP } \square (\#0 \leq \$v))$ 
  defines F3  $\equiv (\lambda p. \text{TEMP } \square (\$p \vee \neg \$p))$ 
  defines F4  $\equiv (\lambda z. \text{TEMP } \$z = \#0 \wedge z \text{ gets } \$z + \#1)$ 
  defines F5  $\equiv (\lambda z. \text{TEMP } \text{fin}(\$z = \#4))$ 
  defines Init2  $\equiv (\lambda v. \text{TEMP } \$v = \#0)$ 
  defines Init3  $\equiv (\lambda p. \text{TEMP } \$p)$ 
  defines Mon1  $\equiv \text{FIRST}(F2 \ v)$ 
  defines Mon2  $\equiv \text{EMPTY UPTO Mon1}$ 
  defines Mon3  $\equiv \text{Mon1 WITH } (F2 \ v)$ 

```

```

defines Mon4  $\equiv$  Mon2 THEN Mon1
defines Mon5  $\equiv$  Mon3 THRU Mon4
defines Mon6  $\equiv$  (FIRST F4 z) WITH (F5 z)

```

lemma (*in Test*) *test*:

$\vdash \mathcal{M}(\text{Mon1}) = \text{empty}$

proof –

have 1: $\vdash \mathcal{M}(\text{Mon1}) = \triangleright(\Box (\#0 \leq \$v))$

using *F2-def Mon1-def* **by** *fastforce*

have 2: $\vdash \Box (\#0 \leq \$v)$

by (*simp add: Valid-def itl-defs*)

have 3: $\vdash \triangleright(\Box (\#0 \leq \$v)) = \text{empty}$

using 2 **by** (*metis FstTrue int-eq int-eq-true*)

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

qed

lemma (*in Test*) *test1*:

$\vdash \mathcal{M}(\text{Mon2}) = \text{empty}$

proof –

have 1: $\vdash \mathcal{M}(\text{Mon2}) = \mathcal{M}(\text{EMPTY UPTO Mon1})$

using *Mon2-def* **by** *fastforce*

have 2: $\vdash \mathcal{M}(\text{EMPTY UPTO Mon1}) = \triangleright(\mathcal{M}(\text{EMPTY}) \vee \mathcal{M}(\text{Mon1}))$

by *fastforce*

have 3: $\vdash \triangleright(\mathcal{M}(\text{EMPTY}) \vee \mathcal{M}(\text{Mon1})) = \triangleright(\text{empty} \vee \text{empty})$

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*:

$\vdash \mathcal{M}(\text{Mon3}) = \text{empty}$

proof –

have 1: $\vdash \mathcal{M}(\text{Mon3}) = \mathcal{M}(\text{Mon1 WITH } (F2\ v))$ **using** *Mon3-def* **by** *fastforce*

have 2: $\vdash \mathcal{M}(\text{Mon1 WITH } (F2\ v)) = (\mathcal{M}(\text{Mon1}) \wedge (F2\ v))$ **by** *fastforce*

have 3: $\vdash (\mathcal{M}(\text{Mon1}) \wedge (F2\ v)) = (\text{empty} \wedge (F2\ v))$ **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*:

$(nfinite\ s \wedge z\ (nnth\ s\ 0) = 0 \wedge (\forall\ i < nlength\ s.\ z\ (nnth\ s\ (Suc\ i)) = Suc(z\ (nnth\ s\ i))) \wedge$
 $z\ (nnth\ s\ (the-enat\ (nlength\ s))) = 4)$
 \implies
 $(nfinite\ s \wedge z\ (nnth\ s\ 0) = 0 \wedge (\forall\ i \leq nlength\ s.\ z\ (nnth\ s\ i) = i)$
 $\wedge z\ (nnth\ s\ (the-enat\ (nlength\ s))) = 4)$

proof –

assume $a0$: $nfinite\ s \wedge (z\ (nnth\ s\ 0) = 0 \wedge (\forall\ i < nlength\ s.\ z\ (nnth\ s\ (Suc\ i)) = Suc(z\ (nnth\ s\ i))) \wedge$
 $z\ (nnth\ s\ (the-enat\ (nlength\ s))) = 4)$

show $nfinite\ s \wedge (z\ (nnth\ s\ 0) = 0 \wedge (\forall\ i \leq nlength\ s.\ z\ (nnth\ s\ i) = i)$

$\wedge z\ (nnth\ s\ (the-enat\ (nlength\ s))) = 4)$

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 : $(\forall\ i \leq nlength\ s.\ z\ (nnth\ s\ i) = i)$

proof

fix i

show $i \leq nlength\ s \longrightarrow z\ (nnth\ s\ i) = i$

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*:

$(nfinite\ s \wedge z\ (nnth\ s\ 0) = 0 \wedge (\forall\ i \leq nlength\ s.\ z\ (nnth\ s\ i) = i)$

$\wedge z\ (nnth\ s\ (the-enat\ (nlength\ s))) = 4) \implies$

$(nfinite\ s \wedge z\ (nnth\ s\ 0) = 0 \wedge (\forall\ i < nlength\ s.\ z\ (nnth\ s\ (Suc\ i)) = Suc(z\ (nnth\ s\ i))) \wedge$

$z\ (nnth\ s\ (the-enat\ (nlength\ s))) = 4)$

proof –

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*:

$$\begin{aligned} & (nfinite\ s \wedge z\ (nnth\ s\ 0) = 0 \wedge (\forall\ i < nlength\ s.\ z\ (nnth\ s\ (Suc\ i)) = Suc(z\ (nnth\ s\ i))) \wedge \\ & \quad z\ (nnth\ s\ (the-enat\ (nlength\ s))) = 4) \\ & = \\ & (nfinite\ s \wedge z\ (nnth\ s\ 0) = 0 \wedge (\forall\ i \leq nlength\ s.\ z\ (nnth\ s\ i) = i) \\ & \quad \wedge z\ (nnth\ s\ (the-enat\ (nlength\ s))) = 4) \end{aligned}$$

using *test5-3 test5-4* **by** *blast*

lemma (in *Test*) *test5-6* :

$$\begin{aligned} & (nfinite\ s \wedge z\ (nnth\ s\ 0) = 0 \wedge (\forall\ i \leq nlength\ s.\ z\ (nnth\ s\ i) = i) \\ & \quad \wedge z\ (nnth\ s\ (the-enat\ (nlength\ s))) = 4) = \\ & (nlength\ s = 4 \wedge (\forall\ i \leq nlength\ s.\ z\ (nnth\ s\ i) = i)) \end{aligned}$$

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* :

$$\begin{aligned} & (s \models (\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4) \wedge finite) = \\ & \quad (nlength\ s = 4 \wedge (\forall\ i \leq nlength\ s.\ z\ (nnth\ s\ i) = i)) \end{aligned}$$

using *test5-6[of s] test5-5[of s] test5-2[of s]* **by** *presburger*

lemma (in *Test*) *test5-8-0*:

$$\begin{aligned} & (s \models \triangleright((\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4))) = \\ & \quad (s \models \triangleright(((\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4)) \wedge finite))) \end{aligned}$$

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*:

$$\begin{aligned} & (\vdash \triangleright(((\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4)) \wedge finite))) = \\ & \quad (\vdash \triangleright((\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4) \wedge finite))) \end{aligned}$$

proof –

$$\begin{aligned} & \textbf{have } 1: \vdash (((\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4)) \wedge finite) = \\ & \quad ((\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4) \wedge finite) \end{aligned}$$

by *auto*

show *?thesis* **by** (*metis 1 int-eq*)

qed

lemma (in *Test*) *test5-8-2*:

$$\begin{aligned} & (s \models \triangleright(((\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4)) \wedge finite))) = \\ & \quad (s \models \triangleright((\$z = \#0 \wedge z\ gets\ \$z + \#1) \wedge fin(\$z = \#4) \wedge finite))) \end{aligned}$$

using *test5-8-1* **unfolding** *Valid-def*

by (*simp add: Fstsem*)

lemma (in *Test*) *test5-8* :

$(s \models \triangleright ((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{finite})) =$
 $((s \models ((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{finite}) \wedge \text{nlength } s = 0 \vee$
 $0 < \text{nlength } s \wedge$
 $\text{nfinite } s \wedge$
 $(s \models ((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{finite}) \wedge$
 $(\forall ia. \text{enat } ia < \text{nlength } s \longrightarrow (\text{ntaken } ia \text{ } s \models \neg (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge$
 $\text{finite}))))$

using *Fstsem*[of *TEMP* $((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{finite}$]
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
 $0 < \text{nlength } s \wedge$
 $\text{nfinite } s \wedge$
 $(s \models ((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{finite}) \wedge$
 $(\forall ia. \text{enat } ia < \text{nlength } s \longrightarrow (\text{ntaken } ia \text{ } s \models \neg (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge$
 $\text{finite}))))$

proof –

assume *a0*: $s \models ((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{finite}$
show $0 < \text{nlength } s \wedge$
 $\text{nfinite } s \wedge$
 $(s \models ((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{finite}) \wedge$
 $(\forall ia. \text{enat } ia < \text{nlength } s \longrightarrow (\text{ntaken } ia \text{ } s \models \neg (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge$
 $\text{finite}))))$

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*: $(\forall ia. \text{enat } ia < \text{nlength } s \longrightarrow (\text{ntaken } ia \text{ } s \models \neg (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge$
 $\text{finite}))))$

proof

fix *ia*
show $\text{enat } ia < \text{nlength } s \longrightarrow$
 $(\text{ntaken } ia \text{ } s \models \neg (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{finite}))$
proof –
have *1*: $(\text{ntaken } ia \text{ } s \models \neg (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{finite})) =$
 $(\neg(\text{ntaken } ia \text{ } s \models (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{finite}))))$
by *auto*
have *2*: $(\text{ntaken } ia \text{ } s \models (((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{finite})) =$
 $(\text{nlength } (\text{ntaken } ia \text{ } s) = 4 \wedge (\forall i. \text{enat } i \leq \text{nlength } (\text{ntaken } ia \text{ } s) \longrightarrow z (\text{nnth } (\text{ntaken } ia \text{ } s) i) =$
 $i))$
using *test5-7*[of *ntaken ia s*] **by** *auto*

```

have 3: enat ia < nlength s  $\longrightarrow$ 
   $\neg(\text{nlength } (\text{ntaken } ia \ s) = 4 \wedge (\forall i. \text{enat } i \leq \text{nlength } (\text{ntaken } ia \ s) \longrightarrow z \ (\text{nnth } (\text{ntaken } ia \ s) \ i) =$ 
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

References

- [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] 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.
- [3] A. Lochbihler. Coinductive. *Archive of Formal Proofs*, feb 2010. <https://www.isa-afp.org/entries/Coinductive.html>, Formal proof development.
- [4] S. Merz. An Encoding of TLA in Isabelle. <http://www.pst.informatik.uni-muenchen.de/~merz/isabelle/>. Part of the Isabelle distribution., 1998.
- [5] 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](#).
- [6] 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.
- [7] B. Moszkowski and D. Guelev. An application of temporal projection to interleaving concurrency. *Formal Aspects of Computing*, 29(4):705–750, July 2017.
- [8] B. C. Moszkowski. Imperative reasoning in interval temporal logic. Technical report, University of Newcastle upon Tyne, 1996.

- [9] B. C. Moszkowski. A Complete Axiom System for Propositional Interval Temporal Logic with Infinite Time. *Logical Methods in Computer Science Journal*, 8(3), 2012.
- [10] 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.
- [11] 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.