

An encoding of Interval Temporal Logic in Isabelle/HOL

Antonio Cau

Ben Moszkowski

David Smallwood

April 9, 2021

Abstract

These Isabelle theories introduce the semantics and syntax of Finite and Infinite Interval Temporal Logic (ITL). The ITL proof system, as introduced in [5, 8], 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 [7], has been checked. Non-empty coinductive lists are used to denote intervals.

Contents

1	Extra operations on LLists	5
1.1	Auxiliary lemmas	5
1.2	lbutlast	6
1.3	kfilter	11
1.4	Transfer rules	48
2	Coinductive non-empty lists and their operations	48
2.1	Type definition	49
2.2	Code generator setup	50
2.3	Connection with ' <i>a llist</i> '	52
2.4	The <i>nlast</i> element <i>nlast</i>	64
2.5	<i>nset</i>	65
2.6	<i>nmap</i>	67
2.7	Appending two nonempty lazy lists <i>nappend</i>	68
2.8	Appending a nonempty lazy list to a lazy list <i>lappendn</i>	70
2.9	<i>nellist-all2</i>	70
2.10	From a nonempty lazy list to a lazy list <i>llist-of-nellist</i>	75
2.11	The length of a nonempty lazy list <i>nlength</i>	75
2.12	The <i>nth</i> element of a nonempty lazy list <i>nnth</i>	75
2.13	<i>ndropn</i>	78
2.14	<i>ntake</i>	86
2.15	<i>ntaken</i>	90
2.16	<i>nsubn</i>	94
2.17	<i>nzip</i>	96
2.18	<i>niterates</i>	101
2.19	Filtering non-empty lazy lists <i>nfilter</i>	102
2.20	<i>nfuse</i>	113

2.21	Setup for Lifting/Transfer	117
2.21.1	Relator and predicator properties	117
2.21.2	Transfer rules for the Transfer package	117
3	Extra operations on nellists	121
3.1	ndropns	121
3.2	nkfilter	124
4	Finite and Infinite ITL Semantics	163
4.1	Types of Formulas	163
4.2	Semantics of ITL	163
4.3	Abbreviations	165
4.4	Properties of Operators	173
4.5	Soundness Axioms	182
4.5.1	ChopAssoc	182
4.5.2	OrChopImp	186
4.5.3	ChopOrImp	186
4.5.4	EmptyChop	186
4.5.5	ChopEmpty	186
4.5.6	StateImpBi	186
4.5.7	NextImpNotNextNot	187
4.5.8	BiBoxChopImpChop	187
4.5.9	BoxInduct	187
4.5.10	ChopStarEqv	188
4.6	Quantification over State (Flexible) Variables	191
4.7	Omega	191
4.8	Temporal Quantifiers	194
5	Finite and Infinite ITL: Axioms and Rules	194
5.1	Rules	195
5.2	Axioms	195
5.3	Additional Lemmas	196
5.4	Quantification	197
5.5	Lemmas about <i>current-val</i>	197
5.6	Lemmas about <i>next-val</i>	198
5.7	Lemmas about <i>fin-val</i>	198
5.8	Lemmas about <i>penult-val</i>	199
5.9	Basic temporal variables properties	199
6	Finite and Infinite ITL theorems using Weak Chop	200
6.1	Propositional reasoning	200
6.2	State formulas	202
6.3	finite and inf properties	202
6.4	Basic Theorems	205
6.5	Further Properties Di and Bi	219
6.6	Properties of Da and Ba	225
6.7	Properties of Fin	233
6.8	Properties of Chopstar and Chopplus	260

6.9	Properties of Omega	289
6.10	Properties of While	296
6.11	Properties of Halt	304
6.12	Properties of Groups of chops	310
7	Finite and Infinite ITL theorems using strong chop	310
7.1	Strong Chop axioms	310
7.2	ITL operators in terms of SChop	312
7.3	Basic Theorems	313
7.4	Further Properties Df and Bf	319
7.5	Properties of SDa and SBa	327
7.6	Properties of SFin	336
7.7	Properties of SChopstar and SChopplus	363
7.8	Properties of Omega	385
7.9	Properties of SWhile	385
7.10	Properties of Halt	391
7.11	Properties of Groups of strong chops	396
8	Infinite and Finite Interval Temporal Algebra	396
8.1	Definition of Set of intervals and Operations on them	396
8.2	Simplification Lemmas	398
8.3	Algebraic Laws	403
8.3.1	Commutative Additive Monoid	403
8.3.2	Boolean algebra	403
8.3.3	multiplicative monoid	404
8.3.4	Subsumption order	407
8.3.5	Helper lemmas	407
8.3.6	Kleene Algebra	418
8.3.7	Omega Algebra	420
8.3.8	ITL specific Laws	421
8.4	Derived Laws	422
8.4.1	Helper Lemmas	422
8.4.2	ITL Axioms derived	429
8.5	Extra Laws	430
8.5.1	Boolean Laws	430
8.5.2	Chop	432
8.5.3	Next	433
8.5.4	SInit	437
8.5.5	SStar	441
8.5.6	Box and Diamond	443
8.5.7	Finite and Infinite	449
8.5.8	Omega	457
8.6	Link between Set of Intervals and ITL	465

9	Until operator for Finite and Infinite Intervals	471
9.1	Definitions	471
9.2	Axioms	472
9.2.1	NextUntil	472
9.2.2	UntilNextUntil	474
9.2.3	NotUntilFalse	476
9.2.4	UntilOrDist	476
9.2.5	UntilRightDistOr	476
9.2.6	UntilLeftDistAnd	476
9.2.7	UntilAndDist	476
9.2.8	untilNotImp	477
9.2.9	UntilUntil	477
9.2.10	UntilRightor	478
9.2.11	UntilRightAnd	478
9.2.12	DiamondEqvTrueUntil	479
9.2.13	TrueUntilImpNotUntil	479
9.2.14	WaitNotDistUntil	480
9.2.15	UntilInduction	481
9.3	Theorems	483
10	Pi operator for infinite and finite ITL	509
10.1	Definitions	509
10.2	Semantic Lemmas	510
10.3	Soundness of Axioms	514
10.3.1	PiK	514
10.3.2	PiDc	515
10.3.3	PiN	515
10.3.4	PiTrueEqvDiamond	515
10.3.5	PiOr	515
10.3.6	UPiFalseEqvBoxNot:	515
10.3.7	BoxEqvImpPiEqv	515
10.3.8	PiDiamondImpDiamond	516
10.3.9	PiAssoc	516
10.3.10	PiNotEqvDiamondAndNotPi	521
10.3.11	PiSChopDist	521
10.3.12	PiProp	525
10.3.13	PiNext	526
10.3.14	PiUntil	529
10.3.15	PiChopstar	540
10.3.16	TruePiEqv	547
10.3.17	BoxImpEqvPi	548
10.3.18	PiEqvDiamondUPi	548
10.3.19	PiEqvUntilPi	548
10.3.20	UPiEqvBoxOrPi	548
10.4	Theorems	548
11	First Order Finite and Infinite ITL theorems	559

1 Extra operations on LLists

the operations `kfilter`, `lleast`, `lbutlast` and `lidx` on coinductive lists are defined together with a library of lemmas.

theory *LListKFilter*

imports

Coinductive.Coinductive-List

begin

definition *kfilter* :: $('a \Rightarrow \text{bool}) \Rightarrow \text{nat} \Rightarrow 'a \text{ llist} \Rightarrow \text{nat llist}$

where *kfilter* *P n xs* = *lmap snd (lfilter (P ∘ fst) (lzip xs (iterates Suc n)))*

definition *lleast* :: $('a \Rightarrow \text{bool}) \Rightarrow 'a \text{ llist} \Rightarrow \text{nat}$

where *lleast* *P xs* = *(LEAST na. na < llength xs ∧ P (lnth xs na))*

primcorec *lbutlast* :: $'a \text{ llist} \Rightarrow 'a \text{ llist}$

where *lbutlast* *xs* =

(case xs of LNil ⇒ LNil |
(LCons x xs') ⇒ (if xs' = LNil then LNil else (LCons x (lbutlast xs'))))

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

definition *lidx* :: $\text{nat llist} \Rightarrow \text{bool}$

where *lidx* *xs* $\longleftrightarrow (\forall n. (\text{Suc } n) < \text{llength } xs \longrightarrow \text{lnth } xs \ n < \text{lnth } xs \ (\text{Suc } n))$

1.1 Auxiliary lemmas

lemma *enat-min*:

assumes $m \geq \text{enat } n'$

and $\text{enat } n < m - \text{enat } n'$

shows $\text{enat } n + \text{enat } n' < m$

using *assms*

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

lemma *enat-min-eq*:

assumes $m \geq \text{enat } n'$

and $\text{enat } n \leq m - \text{enat } n'$

shows $\text{enat } n + \text{enat } n' \leq m$

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*:

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

proof *auto*

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

proof (*coinduction arbitrary: xs ys*)

case (*Eq-llist xsa ysa*)

then show *?case*

proof –

```

assume  $a$ :  $l\text{length } xsa = l\text{length } ysa$ 
assume  $b$ :  $\forall i. \text{enat } i < l\text{length } ysa \longrightarrow l\text{nth } xsa \ i = l\text{nth } ysa \ i$ 
have 1:  $l\text{null } xsa = l\text{null } ysa$ 
  using  $a$  by auto
have 2:  $\neg l\text{null } xsa \longrightarrow$ 
   $\neg l\text{null } ysa \longrightarrow$ 
   $l\text{hd } xsa = l\text{hd } ysa$ 
  using  $b$  lhd-conv-lnth zero-enat-def by force
have 3:  $\neg l\text{null } xsa \longrightarrow$ 
   $\neg l\text{null } ysa \longrightarrow$ 
   $(\exists xs \ ys. ltl \ xsa = xs \wedge ltl \ ysa = ys \wedge l\text{length } xs = l\text{length } ys \wedge$ 
   $(\forall i. \text{enat } i < l\text{length } ys \longrightarrow l\text{nth } xs \ i = l\text{nth } ys \ i))$ 
  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:
   $(\exists x \in l\text{set } xs. P \ x) \longleftrightarrow (\exists i < l\text{length } xs. P \ (l\text{nth } xs \ i))$ 
by (metis in-lset-conv-lnth)

```

```

lemma exist-llength-gr-zero:
assumes  $(\exists x \in l\text{set } xs. P \ x)$ 
shows  $0 < l\text{length } xs$ 
by (metis assms gr-implies-not-zero gr-zeroI in-lset-conv-lnth)

```

1.2 lbutlast

```

lemma lbutlast-snoc [simp]:
   $l\text{butlast } (l\text{append } xs \ (L\text{Cons } x \ LNil)) = xs$ 
proof (cases lfinite xs)
case True
then show ?thesis
  proof (induct rule: lfinite-induct)
  case ( $LNil \ xs$ )
  then show ?case using l\text{list.collapse}(1) by fastforce
  next
  case ( $LCons \ xs$ )
  then show ?case
  by (metis LNil-eq-lappend-iff lappend-code(2) lbutlast-simps(2) lhd-LCons-ltl l\text{list.distinct}(1))
  qed
next
case False
then show ?thesis
  proof (coinduction arbitrary: xs)
  case ( $Eq\text{-l\text{list}} \ xsa$ )
  then show ?case
  proof –
  have 1:  $l\text{null } (l\text{butlast } (l\text{append } xsa \ (LCons \ x \ LNil))) = l\text{null } xsa$ 

```

```

    by (metis Eq-llist lappend-inf lbutlast.disc(2) lfinite.simps lhd-LCons-ltl lnull-imp-lfinite)
  have 2:  $\neg \text{lnull} (\text{lbutlast} (\text{lappend } xsa (\text{LCons } x \text{ LNil}))) \longrightarrow$ 
     $\neg \text{lnull } xsa \longrightarrow$ 
     $\text{lhd} (\text{lbutlast} (\text{lappend } xsa (\text{LCons } x \text{ LNil}))) = \text{lhd } xsa$ 
  by simp
  have 3:  $\neg \text{lnull} (\text{lbutlast} (\text{lappend } xsa (\text{LCons } x \text{ LNil}))) \longrightarrow$ 
     $\neg \text{lnull } xsa \longrightarrow$ 
     $(\exists xs. \text{ltl} (\text{lbutlast} (\text{lappend } xsa (\text{LCons } x \text{ LNil})))) =$ 
     $\text{lbutlast} (\text{lappend } xs (\text{LCons } x \text{ LNil})) \wedge \text{ltl } xsa = xs \wedge \neg \text{lfinite } xs$ 
  by (metis Eq-llist lappend-inf lbutlast-simps(2) lfinite-LNil lfinite-ltl lhd-LCons-ltl
    ltl-simps(2))
  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 (metis lbutlast.ctr(1) lbutlast.simps(4) llist.case-eq-if lnull-def lnull-ltlI)
  qed
next
case False
then show ?thesis
  proof (coinduction arbitrary: xs)
  case (Eq-llist xsa)
  then show ?case
    proof -
    have 1:  $\text{lnull} (\text{lbutlast} (\text{ltl } xsa)) = \text{lnull} (\text{ltl} (\text{lbutlast } xsa))$ 
      by (metis Eq-llist lbutlast-simps(2) lfinite-ltl lhd-LCons-ltl llist.discI(1)
        lnull-imp-lfinite ltl-simps(2))
    have 2:  $\neg \text{lnull} (\text{lbutlast} (\text{ltl } xsa)) \longrightarrow$ 
       $\neg \text{lnull} (\text{ltl} (\text{lbutlast } xsa)) \longrightarrow$ 
       $\text{lhd} (\text{lbutlast} (\text{ltl } xsa)) = \text{lhd} (\text{ltl} (\text{lbutlast } xsa))$ 
      by (metis lbutlast.disc-iff(1) lbutlast.simps(4) llist.simps(5) lnull-ltlI ltl-def
        not-tnull-conv)
    have 3:  $\neg \text{lnull} (\text{lbutlast} (\text{ltl } xsa)) \longrightarrow$ 
       $\neg \text{lnull} (\text{ltl} (\text{lbutlast } xsa)) \longrightarrow$ 
       $(\exists xs. \text{ltl} (\text{lbutlast} (\text{ltl } xsa)) = \text{lbutlast} (\text{ltl } xs) \wedge$ 
         $\text{ltl} (\text{ltl} (\text{lbutlast } xsa)) = \text{ltl} (\text{lbutlast } xs) \wedge \neg \text{lfinite } xs)$ 
      by (metis Eq-llist lbutlast.simps(4) lfinite-LNil lfinite-ltl llist.case-eq-if ltl-def)
    show ?thesis using 1 2 3 by auto
    qed
  qed
qed

```

qed
qed

lemma *lbutlast-not-lfinite*:
assumes $\neg \text{lfinite } xs$
shows $\text{lbutlast } xs = xs$
using *assms*
by (*coinduction arbitrary: xs*)
 (*metis lbutlast.disc-iff(1) lbutlast.simps(3) lbutlast-ltl lfinite-LNil lfinite-ltl*)

lemma *lbutlast-lfinite*:
 $\text{lfinite } (\text{lbutlast } xs) \longleftrightarrow \text{lfinite } xs$
proof
show $\text{lfinite } (\text{lbutlast } xs) \implies \text{lfinite } xs$
proof (*induct zs \equiv lbutlast xs rule: lfinite-induct*)
case *LNil*
then show ?*case* **using** *lbutlast-not-lfinite* **by** *fastforce*
next
case *LCons*
then show ?*case* **using** *lbutlast-not-lfinite* **by** *fastforce*
qed
show $\text{lfinite } xs \implies \text{lfinite } (\text{lbutlast } xs)$
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]*:
 $\text{llength } (\text{lbutlast } xs) = \text{epred } (\text{llength } xs)$
by (*coinduction arbitrary: xs rule: enat-coinduct*)
 (*simp,*
metis epred-enat-unfold lappend-code(1) lappend-lnull2 lbutlast-ltl llength-def llength-eq-0
llength-ltl llist.discI(1))

lemma *lbutlast-lappend*:
 $\text{lbutlast } (\text{lappend } xs \ ys) = (\text{if } ys = \text{LNil} \text{ then } \text{lbutlast } xs \text{ else } \text{lappend } xs \ (\text{lbutlast } ys))$
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 ys*)


```

case LNil
then show ?thesis by simp
next
case (LCons x21 x22)
then show ?thesis
  proof (cases xs)
  case LNil
  then show ?thesis by simp
  next
  case (LCons x21 x22)
  then show ?thesis using LCons.hyps(3) lappend-eq-LNil-iff by fastforce
  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
   $\neg$  lnull xs
shows (lappend (lbutlast xs) (LCons (llast xs) LNil)) = xs
using assms
proof (induct rule: lfinite-induct)
case (LNil xs)
then show ?case by simp
next
case (LCons xs)
then show ?case
  proof (cases xs)
  case lfinite-LNil
  then show ?thesis using LCons by simp
  next
  case (lfinite-LConsI xs x)
  then show ?thesis using LCons by (simp add: llast-LCons)
qed
qed

```

```

lemma lappend-lbutlast-llast-id-not-lfinite:
assumes  $\neg$  lfinite xs
   $\neg$  lnull xs
shows (lappend (lbutlast xs) (LCons (llast xs) LNil)) = xs
using assms
proof (coinduction arbitrary: xs)
case (Eq-llist xa)
then show ?case
  by (auto simp add: lbutlast-ltl llast-linfinite)
  (metis lfinite-LNil lfinite-ltl)
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-conv-ltake*:
 $\text{lbutlast } xs = \text{ltake } (\text{epred } (\text{llength } xs)) \text{ } xs$
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*)
(metis lfinite-LNil lfinite-ltl,
metis lbutlast.simps(4) lbutlast-not-lfinite lfinite-code(2) lhd-LCons-ltl
llength-eq-infty-conv-lfinite ltake-epred-ltl)
qed
qed

lemma *lmap-lbutlast*:
 $\text{lmap } f \text{ } (\text{lbutlast } xs) = \text{lbutlast } (\text{lmap } f \text{ } xs)$
by (*simp add: lbutlast-conv-ltake*)

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

lemma *in-lset-lbutlast-lappendI*:
 $x \in \text{lset } (\text{lbutlast } xs) \vee (\text{lfinite } xs \wedge x \in \text{lset}(\text{lbutlast } ys)) \implies$
 $x \in \text{lset } (\text{lbutlast } (\text{lappend } xs \text{ } ys))$

by (metis empty-iff in-lset-lappend-iff in-lset-lbutlastD lbutlast-lappend lset-LNil)

lemma *lnth-lbutlast*:

assumes $n < \text{length}(\text{lbutlast } xs)$
shows $\text{lnth } (\text{lbutlast } xs) \ n = \text{lnth } xs \ n$
proof (cases xs)
case $LNil$
then show ?thesis **by** simp
next
case $(LCons \ x21 \ x22)$
then show ?thesis **by** (metis assms lbutlast-conv-ltake length-lbutlast lnth-ltake)
qed

1.3 kfilter

lemma *kfilter-code* [simp, code]:

shows *kfilter-LNil*: $\text{kfilter } P \ n \ LNil = LNil$
and *kfilter-LCons*: $\text{kfilter } P \ n \ (LCons \ x \ xs) =$
 $(\text{if } P \ x \text{ then } LCons \ n \ (\text{kfilter } P \ (Suc \ n) \ xs) \text{ else } \text{kfilter } P \ (Suc \ n) \ xs)$
by (auto simp add: kfilter-def lzip.ctr(2))

lemma *lmap-fst-imp-a*:

assumes $\text{lnull } (\text{lfilter } P \ xs)$
shows $\text{lnull } (\text{lmap } fst \ (\text{lfilter } (P \circ fst) \ (\text{lzip } xs \ (\text{iterates } Suc \ n))))$
using *assms* *lset-lzipD1* **by** fastforce

lemma *lmap-fst-imp-b*:

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

lemma *lmap-snd-lnull*:

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

lemma *kfilter-lnull-conv*:

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

lemma *kfilter-not-lnull-conv*:

$\neg \text{lnull} (kfilter\ P\ n\ xs) \longleftrightarrow (\exists\ x \in \text{lset}\ xs.\ P\ x)$

by (*simp add: kfilter-lnull-conv*)

lemma *lmap-fst-lfilter*:

$lfilter\ P\ (lmap\ fst\ (lzip\ xs\ (iterates\ Suc\ n))) =$

$lmap\ fst\ (lfilter\ (P \circ fst)\ (lzip\ xs\ (iterates\ Suc\ n)))$

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

lemma *lmap-fst-lzip*:

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

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

lemma *lfilter-kfilter-1*:

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

$(lfilter\ P\ xs)$

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

lemma *lfilter-kfilter-snd-llength*:

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

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

by *simp*

lemma *kfilter-llength*:

$llength(kfilter\ P\ n\ xs) = llength(lfilter\ P\ xs)$

proof –

have 1: $llength(kfilter\ P\ n\ xs) = llength\ (lmap\ snd\ (lfilter\ (P \circ fst)\ (lzip\ xs\ (iterates\ Suc\ n))))$

by (*simp add: kfilter-def*)

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

$llength\ (lmap\ fst\ (lfilter\ (P \circ fst)\ (lzip\ xs\ (iterates\ Suc\ n))))$

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

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

$llength(lfilter\ P\ xs)$

by (*metis lfilter-kfilter-1*)

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

qed

lemma *ldropWhile-LEAST*:

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

shows $ldropWhile\ P\ xs = ldrown\ (LEAST\ n.\ n < llength\ xs \wedge (Not \circ P)\ (lnth\ xs\ n))\ xs$

proof –

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

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

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

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

by *atomize-elim*

(*metis (no-types, lifting) dual-order.strict-trans1 enat-ord-code(2) less-imp-le not-le-imp-less*

not-less-Least wellorder-Least-lemma(1))

thus *?thesis* **unfolding** *

```

proof (induct m arbitrary: xs)
case 0
  then show ?case
  by (cases xs) simp-all
next
case (Suc m)
then show ?case
  proof –
    have 1: ldropWhile P (ltl xs) = ldropn m (ltl xs) using Suc
    by (cases xs)
      (simp,
        metis Suc-le-mono ldrop-eSuc-ltl ldropn-Suc-conv-ldropn ldropn-eq-LNil llist.simps(3)
        lnth-Suc-LCons ltl-simps(2) not-le-imp-less)
    have 2: P (lnth xs 0)
    using Suc.prem(2) by auto
  show ?thesis
  by (metis 1 2 ldropWhile-LCons ldrop-eSuc-ltl lhd-LCons-ltl llist.collapse(1)
    lnth-0-conv-lhd ltl-simps(1))
qed
qed
qed

```

lemma ldropWhile-LEAST-not:

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

lemma ltakeWhile-LEAST:

assumes $\exists n < \text{length } xs. (Not \circ P) (\text{lnth } xs \ n)$
shows $(\text{ltakeWhile } P \ xs) = (\text{ltake } (\text{lleast } (Not \circ P) \ xs) \ xs)$

proof–

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

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

and *: $(\text{lleast } (Not \circ P) \ xs) = m$ **unfolding** lleast-def

by atomize-elim

(metis (no-types, lifting) Least-le dual-order.trans enat-ord-code(2) not-less not-less-iff-gr-or-eq
 wellorder-Least-lemma(1))

thus ?thesis **unfolding** *

proof (induct m arbitrary: xs)

case 0

then show ?case

proof –

have ltakeWhile P (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

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 (ltl xs) = ltake m (ltl xs)
    using Suc by (cases xs)
      (simp,
        metis Extended-Nat.eSuc-mono Suc-le-mono eSuc-enat llength-LCons lnth-Suc-LCons ltl-simps(2))
  have 2: P (lnth xs 0)
    using Suc by auto
  show ?thesis
    by (metis 1 2 eSuc-enat lhd-LCons-ltl lnth-0-conv-lhd ltake.ctr(1) ltakeWhile.code
      ltake-eSuc-LCons)
qed
qed
qed

```

```

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

```

```

lemma llength-LEAST:
assumes  $\exists n < \text{llength } xs. P (\text{lnth } xs \ n)$ 
shows (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  $\exists n < \text{llength } xs. (\text{Not} \circ P) (\text{lnth } xs \ n)$ 
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  $\exists n < \text{llength } xs. P (\text{lnth } xs \ n)$ 
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)) =

```

$\text{ldrop } (\text{llength } (\text{ltakeWhile } (P \circ \text{fst}) (\text{lzip } xs \text{ } ys))) (\text{lzip } xs \text{ } ys)$
by (*simp add: ltake-all*)
then show ?thesis
using *f1* **by** (*simp add: ldropWhile-eq-ldrop lmap-fst-lzip-conv-ltake lmap-snd-lzip-conv-ltake ltake-lzip*)
qed

lemma *lzip-ldropWhile-fst-iterates*:

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

lemma *ldropWhile-iterates-split*:

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

proof –

have 1: $\exists na. \text{enat } na < \text{llength } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \wedge$
 $(\text{Not} \circ (\text{Not} \circ P) \circ \text{fst}) (\text{lnth } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \text{ } na)$
using *lmap-fst-lzip[of xs n]*
by (*metis assms comp-apply llength-lmap lnth-lmap*)
have 2: $(\text{ldropWhile } ((\text{Not} \circ P) \circ \text{fst}) (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)))) =$
 $(\text{ldropn } (\text{LEAST } na. na < \text{llength } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \wedge$
 $((\text{Not} \circ (\text{Not} \circ P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \text{ } na)) (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n))))$
using 1 *ldropWhile-LEAST[of (lzip xs (iterates Suc n)) (Not ∘ P) ∘ fst]* **by** *auto*
have 3: $(\text{ldropn } (\text{LEAST } na. na < \text{llength } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \wedge$
 $((\text{Not} \circ (\text{Not} \circ P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \text{ } na)) (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)))) =$
 $(\text{ldropn } (\text{LEAST } na. na < \text{llength } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \text{ } na)) (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n))))$
by *auto*
have 4: $(\text{ldropn } (\text{LEAST } na. na < \text{llength } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \text{ } na)) (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)))) =$
 $(\text{lzip } (\text{ldropn } (\text{LEAST } na. na < \text{llength } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \text{ } na)) xs$
 $(\text{ldropn } (\text{LEAST } na. na < \text{llength } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \text{ } na)) (\text{iterates } \text{Suc } n))))$
using *ldropn-lzip by blast*
have 5: $(\text{ldropn } (\text{LEAST } na. na < \text{llength } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \text{ } na)) (\text{iterates } \text{Suc } n)) =$
 $(\text{iterates } \text{Suc } (n + (\text{LEAST } na. na < \text{llength } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \text{ } na))))$
by (*metis (no-types, lifting) funpow-Suc-conv ldropn-iterates*)
have 6: $(\text{LEAST } na. na < \text{llength } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \text{ } na)) =$
 $(\text{LEAST } na. na < \text{llength } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \wedge$
 $((P \circ \text{fst}))) (\text{lnth } (xs) \text{ } na, \text{lnth } (\text{iterates } \text{Suc } n) \text{ } na))$
by (*metis (no-types, hide-lams) comp-def fst-conv lmap-fst-lzip lnth-lmap*)
have 7: $\text{llength } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) = \text{llength } xs$
by *simp*
have 8: $(\text{LEAST } na. na < \text{llength } (\text{lzip } xs \text{ } (\text{iterates } \text{Suc } n)) \wedge$

```

    (( (P ∘ fst))) (lnth (xs) na, lnth (iterates Suc n) na)) =
    (LEAST na. na < llength xs ∧ P (lnth xs na) )
  by auto
have 9: (lzip (ldropn (LEAST na. na < llength (lzip xs (iterates Suc n)) ∧
    (( (P ∘ fst))) (lnth (lzip xs (iterates Suc n)) na)) xs)
    (ldropn (LEAST na. na < llength (lzip xs (iterates Suc n)) ∧
    (( (P ∘ fst))) (lnth (lzip xs (iterates Suc n)) na)) (iterates Suc n))) =
    (lzip (ldropn (LEAST na. na < llength xs ∧ P (lnth xs na) ) xs)
    (iterates Suc (n+(LEAST na. na < llength xs ∧ 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 ¬ lnull(kfilter P n xs)
  shows kfilter P n xs =
    (LCons (n+ (lleast P xs))
      (lmap snd (lfilter (P ∘ 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 ∘ fst) (lzip xs (iterates Suc n))) =
    lhd (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n)))
  by simp
  have 2: ltl(lfilter (P ∘ fst) (lzip xs (iterates Suc n))) =
    (lfilter (P ∘ fst) (ltl (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n)))))
  by (simp add: ltl-lfilter)
  have 3: lfilter (P ∘ fst) (lzip xs (iterates Suc n)) =
    (LCons (lhd(lfilter (P ∘ fst) (lzip xs (iterates Suc n))))
      (ltl(lfilter (P ∘ 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 ∘ fst) (lzip xs (iterates Suc n)))
  by (simp add: kfilter-def)
  have 5: ltl (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))) =
    ltl((lzip (ldropWhile (Not ∘ P) xs)
      (lmap snd (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n)))))
  by (metis lzip-ldropWhile-fst-iterates)
  have 6: ltl((lzip (ldropWhile (Not ∘ P) xs)
    (lmap snd (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))))) =
    (lzip (ltl (ldropWhile (Not ∘ P) xs)
      (ltl (lmap snd (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n)))))
    (ltl (lmap snd (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n)))))
  using lzip-ldropWhile-fst-iterates[of P xs n]
    ltl-lzip[of (ldropWhile (Not ∘ P) xs)
      (lmap snd (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n)))]
  by force
  have 7: lmap snd (

```



```

    (LCons (lhd (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))))
      (lfilter (P ∘ fst) (ltl (ldropWhile (Not ∘ P ∘ 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 ldrops-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 ldrops-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: ldrops-eq-ldrop)
have 16: ltl (ldrop (llength (ltakeWhile (Not ∘ P) xs)) xs) =
  ldrops (eSuc (llength (ltakeWhile (Not ∘ P) xs))) (xs)
  by (simp add: ldrops-eSuc-conv-ltl)
have 17: (llength (ltakeWhile (Not ∘ P) xs)) = (llength (ltake ?Least xs))
  using 8 ltakeWhile-LEAST[of xs Not ∘ P] unfolding lleast-def by auto
have 18: (llength (ltake ?Least xs)) = enat ?Least
  using 17 8 llength-ltakeWhile-LEAST-not unfolding lleast-def by fastforce
have 19: ldrops (eSuc (llength (ltakeWhile (Not ∘ P) xs))) (xs) = ldrops (Suc ?Least) (xs)
  using 17 18 by (simp add: eSuc-enat)
have 20: ltl (ldropWhile (Not ∘ P ∘ fst) (lzip xs (iterates Suc n))) =
  (lzip (ldrops (Suc ?Least) (xs)) (iterates Suc (Suc (n+?Least))))
  using 12 13 15 16 19 by auto
show ?thesis
using 2 3 4 10 20 9 by auto
qed

```

lemma *kfilter-eq-LCons*:

```

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

```

$(\text{zip } (\text{ldrop } (\text{Suc } (\text{lleast } P \text{ } xs)) \text{ } (xs))$
 $(\text{iterates } \text{Suc } (\text{Suc } (n + (\text{lleast } P \text{ } xs))))))$

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

lemma *kfilter-eq-LCons-1*:

$kfilter \ P \ n \ xs = LCons \ x \ xs' \implies$
 $x = (n + (\text{lleast } P \text{ } xs)) \wedge$
 $xs' = kfilter \ P \ (\text{Suc } (n + (\text{lleast } P \text{ } xs))) \ (\text{ldrop } (\text{Suc } (\text{lleast } P \text{ } xs)) \ xs)$
using *kfilter-eq-LCons*[of *P n xs x xs'*]
 $kfilter\text{-}def$ [of $P \ (\text{Suc } (n + (\text{lleast } P \text{ } xs)))$
 $(\text{ldrop } (\text{Suc } (\text{lleast } P \text{ } xs)) \ xs)$]
by *auto*

lemma *kfilter-eq-conv*:

$kfilter \ P \ n \ xs = LNil \vee$
 $kfilter \ P \ n \ xs =$
 $LCons \ (n + (\text{lleast } P \text{ } xs)) \ (kfilter \ P \ (\text{Suc } (n + (\text{lleast } P \text{ } xs))) \ (\text{ldrop } (\text{Suc } (\text{lleast } P \text{ } xs)) \ xs))$
proof (*cases kfilter P n xs*)
case *LNil*
then show *?thesis* **by** *simp*
next
case ($LCons \ x21 \ x22$)
then show *?thesis*
proof –
let *?Least* = $(\text{lleast } P \text{ } xs)$
have *1*: $x21 = n + ?Least$
using *kfilter-eq-LCons-1*[of *P n xs x21 x22*] **by** (*meson LCons*)
have *2*: $x22 = (kfilter \ P \ (\text{Suc } (n + ?Least)) \ (\text{ldrop } (\text{Suc } ?Least) \ xs))$
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 $\text{lnth } (kfilter \ P \ n \ xs) \ 0 = n + (\text{lleast } P \text{ } xs)$
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 $\text{lleast } P \ xs < \text{length } xs$
using *assms exist-lset-lnth*[of *xs P*] *kfilter-not-lnull-conv*[of *P n xs*]
 length-LEAST [of *xs P*] **by** *blast*

lemma *kfilter-upperbound*:

assumes $i < \text{length}(kfilter \ P \ n \ xs)$
shows $(\text{lnth } (kfilter \ P \ n \ xs) \ i) < n + \text{length } xs$
proof (*cases lfinite (kfilter P n xs)*)

```

case True
then show ?thesis using assms
  proof (induct zs≡kfilter P n xs arbitrary: xs n i rule: lfinite-induct)
  case (LNil xs)
  then show ?case
  using gr-implies-not-zero llength-lnull by blast
  next
  case (LCons xs)
  then show ?case
    proof –
      let ?Least = (lleast P xs)
      have 1: ∃ 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 ⟹ (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 ⟹ 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 ⟹ enat (lnth (kfilter P n xs) i) = enat (lnth xs' (i-1))
        by (simp add: 2 lnth-LCons')
      have 9: i>0 ∧ lnull xs' ⟹ enat (lnth (kfilter P n xs) i) < enat n + llength xs
        by (metis 2 LCons.prem1 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 ∧ ¬lnull xs' ⟹ (i-1) < llength xs'
        using 2 LCons.prem1 Suc-ile-eq by fastforce
      have 11: i>0 ∧ ¬lnull xs' ⟹
        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 ≤ (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) ≤ 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 ∧ ¬lnull xs' ⟹ enat (lnth (kfilter P n xs) i) < enat n + llength xs
        using 11 12 8 by auto
      have 15: i>0 ⟹ 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.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:  $\text{lnth}\ (kfilter\ P\ n\ xs)\ (Suc\ i) = \text{lnth}\ xs'\ i$ 
      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:  $\text{enat}\ i < \text{llength}\ (kfilter\ P\ (Suc\ (n + ?Least))\ (ldrop\ (Suc\ ?Least)\ xs))$ 
      by (metis 10 8 Extended-Nat.eSuc-mono Suc.prem1(1) eSuc-enat llength-LCons)
    have 12:  $\text{lnull}\ xs' \implies n \leq \text{lnth}\ (kfilter\ P\ n\ xs)\ (Suc\ i)$ 
      using 10 11 gr-implies-not-zero llength-eq-0 by blast
    have 13:  $\neg \text{lnull}\ xs' \implies (Suc\ (n + ?Least)) \leq \text{lnth}\ xs'\ i$ 
      using 10 11 Suc.hyps by blast
    have 14:  $\neg \text{lnull}\ xs' \implies n \leq \text{lnth}\ (kfilter\ P\ n\ xs)\ (Suc\ i)$ 
      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: (lnth xs ((lnth (kfilter P n xs) 0) - n)) =
  (lnth xs ?Least)
by (simp add: 1)
have 3: (lnth (lfilter P xs) 0) = lhd (ldropWhile (Not ∘ P) xs)
by (metis (full-types) 0.premis kfilter-llength lhd-conv-lnth lhd-lfilter llength-eq-0 not-less0)
have 4: ∃ n < llength xs. P (lnth xs n)
by (metis 0.premis dual-order.irrefl in-lset-conv-lnth kfilter-llength llength-eq-0 lnull-lfilter
  zero-enat-def)
have 5: (ldropWhile (Not ∘ P) xs) = ldropsn ?Least xs
using ldropsWhile-LEAST-not[of xs P ] 4 unfolding lleast-def by blast
have 6: lhd (ldropsn ?Least xs) = (lnth xs ?Least)
by (simp add: 4 lhd-ldropsn llength-LEAST)
show ?thesis using 2 3 5 6 by auto
qed
next
case (Suc i)
then show ?case
proof -
  let ?Least = lleast P xs
have 7: ∃ x xs'. (kfilter P n xs) = LCons x xs'
using Suc.premis gr-implies-not-zero kfilter-ldropWhile llength-lnull by blast
obtain x xs' where 8: (kfilter P n xs) = LCons x xs'
using 7 by auto
have 9: lnth (kfilter P n xs) (Suc i) = lnth xs' i
by (simp add: 8)
have 10: xs' = kfilter P (Suc (n + ?Least)) (ldrop (Suc ?Least) xs)
using kfilter-ldropWhile[of P n xs] by (simp add: 8 kfilter-eq-LCons-1)
have 11: enat i < llength (kfilter P (Suc (n + ?Least)) (ldrop (Suc ?Least) xs))
by (metis 10 8 Extended-Nat.eSuc-mono Suc.premis(1) eSuc-enat llength-LCons)
have 12: lnull xs' ⇒
  lnth xs (lnth (kfilter P n xs) (Suc i) - n) = lnth (lfilter P xs) (Suc i)
by (metis 10 11 llength-LNil llist.collapse(1) not-less-zero)
have 13: ¬lnull xs' ⇒
  lnth (ldrop (Suc ?Least) xs) (lnth xs' i - (Suc (n + ?Least))) =
  lnth (lfilter P (ldrop (Suc ?Least) xs)) i
by (metis (no-types, lifting) 10 11 Suc.hyps)
have 14: (lnth (lfilter P xs) (Suc i)) = (lnth (ltl(lfilter P xs)) i)
by (metis 8 kfilter-not-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 (ldropsn (LEAST n. n < llength xs ∧ (Not ∘ (Not ∘ P)) (lnth xs n) ) xs))
using ldropsWhile-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 (ldropsn (LEAST n. n < llength xs ∧ (Not ∘ (Not ∘ P)) (lnth xs n) ) xs)) =
  (ltl (ldropsn ?Least xs))
unfolding lleast-def by auto
have 18: (ltl (ldropsn ?Least xs)) =
  ldropsn (Suc ?Least) xs

```

```

  by (simp add: ldropsn-ltl ltl-ldropsn)
have 19: ldropsn (Suc ?Least) xs =
  ldropsn (Suc ?Least) xs
  by (simp add: ldropsn-enat)
have 20: lnth (lfilter P xs) (Suc i) = lnth (lfilter P (ldropsn (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)) (ldropsn (Suc ?Least) xs)) i
  using kfilter-lowerbound[of i P Suc (n+?Least) (ldropsn (Suc ?Least) xs) ] 11 by force
have 24: (Suc ?Least) > llength (ltake ?Least xs)
  by (simp add: min.strict-coboundedI1)
have 25: (ldropsn (Suc ?Least) (lappend (ltake ?Least xs) (ldropsn ?Least xs))) =
  ldropsn ((Suc ?Least) - llength (ltake ?Least xs) ) (ldropsn ?Least xs)
  by (simp add: ldropsn-lappend)
have 26: lappend (ltake (Suc ?Least) xs) (ldropsn (Suc ?Least) xs) = xs
  by (simp add: lappend-ltake-ldropsn)
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) (ldropsn (Suc ?Least) xs)) (lnth xs' i - n) =
  lnth (ldropsn (Suc ?Least) xs) (lnth xs' i - (Suc (n+?Least)))
  using lnth-lappend[of (ltake (Suc ?Least) xs) (ldropsn (Suc ?Least) xs) (lnth xs' i - n)]
  27 271 11 19
  by (metis (no-types, lifting) gr-implies-not-zero kfilter-LNil ldropsn-eq-LNil
    le-cases llength-lnull llength-ltake llist.disc(1) lnth-lappend2 min-def)
have 29: lnth xs (lnth xs' i - n) =
  lnth (ldropsn (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
qed

```

lemma in-kfilter-lset:

shows $x \in \text{lset} (\text{kfilter } P \ n \ xs) \longleftrightarrow x \in \{ n+i \mid i. i < \text{llength } xs \wedge P (\text{lnth } xs \ i) \}$
(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:  $\neg \text{lnull } (kfilter\ P\ (Suc\ (n+?Least))\ (ldrop\ (Suc\ ?Least)\ xs)) \implies$ 
 $y \in \{n + i \mid i. \text{enat } i < \text{llength } xs \wedge P\ (\text{lnth } xs\ i)\}$ 
using 7 9 by auto
show ?thesis
using 13 6 by blast
qed
qed
next
assume ?rhs
then obtain i where 15:  $x = n+i \wedge \text{enat } i < \text{llength } xs \wedge P\ (\text{lnth } xs\ i)$ 
by blast
thus ?lhs
proof (induct i arbitrary: xs n)
case 0
then show ?case
proof -
have 16:  $\text{lhd } (kfilter\ P\ n\ xs) = n+(\text{lleast } P\ xs)$ 
using kfilter-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
lset.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

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

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

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

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

lemma *ldistinct-kfilter*:

ldistinct(*kfilter P n xs*)

proof (*coinduction arbitrary: n xs*)

case (*ldistinct n1 xs1*)

then show *?case*

proof –

have 1: *lhd* (*kfilter P n1 xs1*) \notin *lset* (*ltl* (*kfilter P n1 xs1*))

proof –

have *f1*: *kfilter P n1 xs1* =

LCons (*n1* + *lleast P xs1*)

(*lmap snd*

(*lfilter* (*P* \circ *fst*)

(*lzip*

(*ldrop* (*enat* (*Suc* (*lleast P xs1*))) *xs1*)

(*iterates Suc* (*Suc* (*n1* + *lleast P xs1*))))))

by (*meson kfilter-ldropWhile ldistinct*)

then have *lmap snd*

(*lfilter* (*P* \circ *fst*)

(*lzip* (*ldrop* (*enat* (*Suc* (*lleast P xs1*))) *xs1*)

(*iterates Suc* (*Suc* (*n1* + *lleast P xs1*)))))) =

kfilter P (*Suc* (*n1* + *lleast P xs1*)) (*ldrop* (*enat* (*Suc* (*lleast P xs1*))) *xs1*)

using *kfilter-eq-LCons-1* **by** *blast*

then show *?thesis*

using *f1* **by** (*metis* (*no-types*) *in-lset-conv-lnth kfilter-lowerbound leD lessI lhd-LCons ltl-simps*(2))

qed

have 2: $((\exists n \ xs. \text{ltl } (\text{kfilter } P \ n1 \ xs1) = \text{kfilter } P \ n \ xs) \vee \text{ldistinct } (\text{ltl } (\text{kfilter } P \ n1 \ xs1)))$

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 $k < \text{length } xs$

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

```

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

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

lemma kfilter-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  $n \leq (\text{lnth } (\text{kfilter } P \ n \ xs) \ k)$ 
using assms by (simp add: kfilter-lnull-conv kfilter-lowerbound)

lemma kfilter-lnth-n-zero:
assumes  $k < \text{llength } (\text{kfilter } P \ n \ xs)$ 
shows  $(\text{lnth } (\text{kfilter } P \ n \ xs) \ k) - n = (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k)$ 
using assms
proof (induct k arbitrary: xs n)
case 0
then show ?case by (cases (kfilter P n xs))
  (simp,
    metis add.left-neutral add-left-cancel kfilter-ldropWhile kfilter-lnull-conv kfilter-lowerbound
    le-add-diff-inverse llength-eq-0 lnth-0 not-gr-zero zero-enat-def)
next
case (Suc k)
then show ?case
  proof –
    have 1:  $\text{lnull}(\text{kfilter } P \ n \ xs) \implies \text{lnth } (\text{kfilter } P \ n \ xs) \ (\text{Suc } k) - n = \text{lnth } (\text{kfilter } P \ 0 \ xs) \ (\text{Suc } k)$ 
      using Suc.premis gr-implies-not-zero llength-lnull 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:  $x = n + ?\text{Least}$ 
            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-lnull-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:  $y = ?\text{Least}$ 
            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:  $xs' = \text{kfilter } P \ (\text{Suc } (n + ?\text{Least})) \ (\text{ldrop } (\text{Suc } ?\text{Least}) \ xs)$ 
            using kfilter-ldropWhile[of P n xs]
            by (metis (no-types, lifting) 4 a0 kfilter-def ltl-simps(2))
          have 12:  $ys' = \text{kfilter } P \ (\text{Suc } (0 + ?\text{Least})) \ (\text{ldrop } (\text{Suc } ?\text{Least}) \ xs)$ 
            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: lnth (kfilter P 0 xs) (Suc k) = lnth ys' k
  by (simp add: 8)
have 15: ¬(∃ x ∈ lset (ldrop (Suc ?Least) xs). P x) ⇒
  lnth (kfilter P n xs) (Suc k) - n = lnth (kfilter P 0 xs) (Suc k)
  using 8 Suc by auto
  (metis (full-types) gr-implies-not-zero kfilter-eq-conv kfilter-not-lnull-conv
    ldrown-Suc-LCons ldrown-Suc-conv-ldrown ldrown-eq-LConsD llength-LNil llist.disc(2))
have 16: enat k < llength (kfilter P (Suc (n+?Least)) (ldrop (Suc ?Least) xs))
  by (metis (no-types, lifting) 12 8 Extended-Nat.eSuc-mono Suc.premis eSuc-enat
    kfilter-llength llength-LCons)
have 17: (∃ x ∈ lset (ldrop (Suc ?Least) xs). P x) ⇒
  lnth xs' k - (Suc (n+?Least)) = lnth (kfilter P 0 (ldrop (Suc ?Least) xs)) k
  using Suc.hyps using 11 16 by blast
have 18: enat k < llength (kfilter P (Suc (?Least)) (ldrop (Suc ?Least) xs))
  by (metis 16 kfilter-llength)
have 19: (∃ x ∈ lset (ldrop (Suc ?Least) xs). P x) ⇒
  lnth ys' k - (Suc (0+?Least)) =
  lnth (kfilter P 0 (ldrop (Suc ?Least) xs)) k
  using Suc.hyps by (simp add: 12 18)
have 20: (∃ x ∈ lset (ldrop (Suc ?Least) xs). P x) ⇒
  lnth (kfilter P n xs) (Suc k) - n = lnth xs' k - n
  using 13 by blast
have 21: (∃ x ∈ lset (ldrop (Suc ?Least) xs). P x) ⇒
  lnth xs' k - n = lnth (kfilter P 0 (ldrop (Suc ?Least) xs)) k + (Suc (?Least))
  using 11 16 17 kfilter-lowerbound by fastforce
have 22: (∃ x ∈ lset (ldrop (Suc ?Least) xs). P x) ⇒
  lnth (kfilter P 0 (ldrop (Suc ?Least) xs)) k + (Suc (?Least)) = lnth ys' k
  using 12 18 19 kfilter-lowerbound by fastforce
have 23: (∃ x ∈ lset (ldrop (Suc ?Least) xs). P x) ⇒
  lnth (kfilter P n xs) (Suc k) - n = lnth (kfilter P 0 xs) (Suc k)
  using 14 20 21 22 by linarith
show ?thesis
  using 15 23 by blast
qed
qed
show ?thesis
  using 1 2 by blast
qed
qed

```

lemma *kfilter-n-zero*:

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

proof –

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

have 2: $\bigwedge k. k < llength(kfilter P n xs) \longrightarrow$
 $lnth (kfilter P n xs) k = lnth (lmap (\lambda i. i+n) (kfilter P 0 xs)) k$

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

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

show *?thesis*

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

lemma *kfilter-n-zero-a:*

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

proof –

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

by (simp add: kfilter-llength)

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

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

by (simp add: kfilter-llength kfilter-lnth-n-zero)

show ?thesis

using 1 2 llist-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-less*:
assumes *lidx xs*
 $\text{Suc}(n+k) < \text{length } xs$
shows $\text{lnth } xs \ n < \text{lnth } xs \ (\text{Suc}(n+k))$
using *assms* **unfolding** *lidx-def*
proof (*induct k*)
case 0
then show ?*case* **by** *auto*
next
case (*Suc k*)
then show ?*case*
by (*metis Suc-ile-eq add-Suc-right less-imp-le less-trans*)
qed

lemma *lidx-less-eq*:
assumes *lidx xs*
 $k \leq j$
 $j < \text{length } xs$
shows $\text{lnth } xs \ k \leq \text{lnth } xs \ j$
using *assms*
proof (*cases k=j*)
case *True*
then show ?*thesis* **by** *simp*
next
case *False*
then show ?*thesis* **using** *assms*
by (*metis less-imp-Suc-add less-imp-le lidx-less order.not-eq-order-implies-strict*)
qed

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

lemma *lidx-expand-1*:

$lidx\ xs \longleftrightarrow lnull\ xs \vee llength\ xs = 1 \vee$
 $(1 < llength\ xs \wedge (\forall n. (Suc\ n) < llength\ xs \longrightarrow lnth\ xs\ n < lnth\ xs\ (Suc\ n)))$
by (metis One-nat-def enat-0-iff(1) iless-Suc-eq ldropn-0 ldropn-Suc-conv-ldropn ldropn-eq-LNil
 $lidx-def\ llength-LCons\ llength-LNil\ lnull-ldropn\ not-le-imp-less\ not-less-zero$
 $old.nat.distinct(2)\ one-eSuc\ one-enat-def\ order.not-eq-order-implies-strict$)

lemma $lidx-LCons$:

$lidx\ (LCons\ x\ xs) \longleftrightarrow$
 $lnull\ xs \vee$
 $(0 < llength\ xs \wedge$
 $(\forall n. (Suc\ n) < eSuc\ (llength\ xs) \longrightarrow lnth\ (LCons\ x\ xs)\ n < lnth\ (LCons\ x\ xs)\ (Suc\ n)))$
by (metis ldropn-0 lidx-def lidx-expand-1 llength-LCons llength-LNil lnull-def lnull-ldropn not-less
 $one-eSuc\ zero-enat-def$)

lemma $lidx-LCons-conv$:

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

proof –

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

\longleftrightarrow

$(0 < llength\ xs \wedge$
 $x < lhd\ xs \wedge$
 $(\forall n. 1 \leq n \wedge (n) < (llength\ xs) \longrightarrow lnth\ (LCons\ x\ xs)\ n < lnth\ (LCons\ x\ xs)\ (Suc\ n)))$

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

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

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

have 5: $(0 < llength\ xs \wedge$

$x < \text{lhd } xs \wedge$
 $(\forall n. 1 \leq n \wedge (n) < (\text{llength } xs) \longrightarrow \text{lnth } xs (n-1) < \text{lnth } xs (n))) \longleftrightarrow$
 $(0 < \text{llength } xs \wedge$
 $x < \text{lhd } xs \wedge$
 $(\forall n. 0 \leq (n-1) \wedge (\text{Suc } (n-1)) < (\text{llength } xs) \longrightarrow \text{lnth } xs (n-1) < \text{lnth } xs (\text{Suc } (n-1))))$
by (metis diff-Suc-1 le0 le-add1 le-add-diff-inverse plus-1-eq-Suc)
have 6: $(0 < \text{llength } xs \wedge$
 $x < \text{lhd } xs \wedge$
 $(\forall n. 0 \leq (n-1) \wedge (\text{Suc } (n-1)) < (\text{llength } xs) \longrightarrow \text{lnth } xs (n-1) < \text{lnth } xs (\text{Suc } (n-1)))) \longleftrightarrow$
 $(0 < \text{llength } xs \wedge$
 $x < \text{lhd } xs \wedge$
 $(\forall n. 0 \leq n \wedge (\text{Suc } n) < (\text{llength } xs) \longrightarrow \text{lnth } xs n < \text{lnth } xs (\text{Suc } n)))$
by (metis diff-Suc-1)
show ?thesis
using 2 3 4 5 6 **by** auto
qed

lemma *lidx-LCons-1* [simp]:
 $\text{lidx } (LCons \ x \ xs) \longleftrightarrow \text{lnull } xs \vee (0 < \text{llength } xs \wedge x < \text{lhd } xs \wedge \text{lidx } xs)$
unfolding *lidx-def*
by (auto simp add: enat-0-iff(1))
 (metis Suc-ile-eq i0-less lhd-conv-lnth llength-eq-0 lnth-0 zero-enat-def,
 metis Suc-ile-eq lnth-Suc-LCons,
 metis gr-implies-not-zero iless-Suc-eq lidx-LCons-conv llength-eq-0 lnth-Suc-LCons
 not-less-iff-gr-or-eq)

lemma *lidx-kfilter-expand*:
assumes $(\text{Suc } na) < \text{llength}(\text{kfilter } P \ n \ xs)$
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 $\text{lidx } (\text{kfilter } P \ n \ xs)$
unfolding *lidx-def*
using *lidx-kfilter-expand* **by** blast

lemma *lidx-kfilter-gr-eq*:
assumes
 $k \leq j$
 $j < \text{llength}(\text{kfilter } P \ n \ xs)$
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 $\forall j. k < j \wedge j < \text{llength}(\text{kfilter } P \ n \ xs) \longrightarrow$
 $\text{lnth } (\text{kfilter } P \ n \ xs) \ k < \text{lnth } (\text{kfilter } P \ n \ xs) \ j$
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  $\forall ns \ n. \exists na. ((n::\text{nat}) \notin \text{lset } ns \vee \text{lnth } ns \ na = n) \wedge (n \notin \text{lset } ns \vee \text{enat } na < \text{length } ns)$ 
by (meson in-lset-conv-lnth)
then show ?thesis using step
by (metis (no-types) leD less-nat-zero-code lidx-kfilter-gr-eq llist.set-intros(2) not-le-imp-less)
qed
qed
have  $3: i \notin \text{lset}(\text{kfilter } P \ 0 \ xs) \wedge i < \text{length } xs \longrightarrow \neg P(\text{lnth } xs \ i)$ 
      by (simp add: 1 in-kfilter-lset)
have  $4: i < \text{length } xs$ 
      using  $0$  assms(2) dual-order.strict-trans enat-ord-simps(2) by blast
show ?thesis
using  $1 \ 2 \ 3 \ 4$  assms(2) by blast
qed

```

```

lemma kfilter-n-not-before:
assumes  $0 < \text{length}(\text{kfilter } P \ n \ (\text{ldropn } n \ xs))$ 
          $n < \text{length } xs$ 
          $n \leq i$ 
          $i < \text{lnth}(\text{kfilter } P \ n \ (\text{ldropn } n \ xs)) \ 0$ 
shows  $\neg P(\text{lnth } xs \ i)$ 
proof -
have  $00: \neg \text{lnull}(\text{kfilter } P \ n \ (\text{ldropn } n \ xs))$ 
      by (metis assms(1) ldrop-enat less-numeral-extra(3) length-LNil llist.collapse(1))
have  $0: \text{lnth}(\text{kfilter } P \ n \ (\text{ldropn } n \ xs)) \ 0 < \text{length } xs$ 
      using assms kfilter-upperbound[of  $0 \ P \ n \ (\text{ldropn } n \ xs)$ ]
      by (metis lappend-ltake-enat-ldropn ldrop-enat length-lappend length-ltake min.strict-order-iff
         zero-enat-def)
have  $1: i \in \text{lset}(\text{kfilter } P \ n \ (\text{ldropn } n \ xs)) \implies$ 
       $\neg \text{lnull}(\text{kfilter } P \ n \ (\text{ldropn } n \ xs)) \implies$ 
       $i < \text{lnth}(\text{kfilter } P \ n \ (\text{ldropn } n \ xs)) \ 0 \implies$ 

```

```

    n < llength xs ==>
    n ≤ i ==>
    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 ∉ lset (kfilter P n (ldropn n xs)) ∧ n ≤ i ∧ i < llength xs → ¬ 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 ≤ i ∧ i < llength xs
    using assms 00 kfilter-ldropWhile[of P n ldropn n xs ]
    by (metis 0 enat-ord-simps(2) ldrop-enat less-trans)
show ?thesis
using 1 2 3 assms(1) assms(2) assms(4) kfilter-not-lnull-conv by auto
qed

```

lemma kfilter-not-after:

```

assumes 0 < llength(kfilter P 0 xs)
    lnth (kfilter P 0 xs) (k-1) < i
    llength (kfilter P 0 xs) = (enat k)
    i < llength xs
shows ¬ P (lnth xs i)
proof -
have 0: ¬ 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 ∉ 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 < llength (kfilter P n (ldropn n xs))
    n < llength xs
    lnth (kfilter P n (ldropn n xs)) (k-1) < i
    llength (kfilter P n (ldropn n xs)) = (enat k)
    i < llength xs
shows ¬ P (lnth xs i)
proof -

```

```

have 0:  $\neg \text{lnull } (\text{kfilter } P \ n \ (\text{ldropn } n \ xs))$ 
  using assms(1) by auto
have 1:  $0 < k$ 
  using assms(1) assms(4) by (simp add: zero-enat-def)
have 2:  $i \notin \text{lset}(\text{kfilter } P \ n \ (\text{ldropn } n \ xs))$ 
  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  $\text{lnth } (\text{kfilter } P \ 0 \ xs) \ (k) < i$ 
   $i < \text{lnth } (\text{kfilter } P \ 0 \ xs) \ (\text{Suc } k)$ 
   $(\text{Suc } k) < \text{llength } (\text{kfilter } P \ 0 \ xs)$ 
shows  $\neg P \ (\text{lnth } xs \ i)$ 
proof –
  have 0:  $\exists x \in \text{lset } xs. P \ x$ 
    using assms(3) gr-implies-not-zero kfilter-not-lnull-conv by fastforce
  have 1:  $\neg \text{lnull } (\text{kfilter } P \ 0 \ xs)$ 
    by (simp add: 0 kfilter-not-lnull-conv)
  have 2:  $\text{lnth } (\text{kfilter } P \ 0 \ xs) \ (\text{Suc } k) \leq \text{llength } xs$ 
    using kfilter-upperbound[of  $\text{Suc } k \ P \ 0 \ xs$ ]
    using 1 assms(3) zero-enat-def by auto
  have 3:  $i \notin \text{lset}(\text{kfilter } P \ 0 \ xs)$ 
    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 lidx-imp-lsorted:
assumes lidx xs
shows lsorted xs
using assms
by (metis (no-types, lifting) less-imp-le lhd-LCons-ltl lidx-LCons-1 lsorted-coinduct')

```

```

lemma lidx-imp-ldistinct:
assumes lidx xs
shows ldistinct xs
using assms
proof (coinduction arbitrary: xs)
case (ldistinct xsa)
then show ?case
  proof –
    have 1:  $\text{lhd } xsa \notin \text{lset } (\text{ltl } xsa)$ 
      by (metis empty-iff lappend-code(1) lappend-lnull2 ldistinct(1) ldistinct(2) leD lhd-LCons-ltl)

```

```

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

```

```

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

```

```

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

```

```

lemma lfilter-kfilter-ltake-lidx-a:
  assumes  $k < \text{llength}(\text{lfilter } P \ xs)$ 
  shows  $\text{lidx } (\text{ltake } k \ (\text{kfilter } P \ n \ xs))$ 
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  $k < \text{llength}(\text{lfilter } P \ xs)$ 
  shows  $\text{lidx } (\text{ldropn } k \ (\text{kfilter } P \ n \ xs))$ 
using assms unfolding lidx-def
proof auto
  fix na
  assume a0:  $\text{enat } k < \text{llength}(\text{lfilter } P \ xs)$ 
  assume a1:  $\text{enat } (\text{Suc } na) < \text{llength}(\text{kfilter } P \ n \ xs) - \text{enat } k$ 
  show  $\text{lnth } (\text{ldropn } k \ (\text{kfilter } P \ n \ xs)) \ na < \text{lnth } (\text{ldropn } k \ (\text{kfilter } P \ n \ xs)) \ (\text{Suc } na)$ 
proof -
  have 1:  $\text{enat } (k + (\text{Suc } na)) < \text{llength}(\text{kfilter } P \ n \ xs)$ 
  proof -
    have  $\text{Suc } na + k = k + \text{Suc } na$ 
    by presburger
    then show ?thesis
    by (metis (no-types) a1 ldropn-eq-LNil ldropn-ldropn leD leI llength-ldropn)
  qed
  have 2:  $\text{enat } (k + na) < \text{llength}(\text{kfilter } P \ n \ xs)$ 

```

```

    by (metis 1 Suc-ile-eq add-Suc-right less-imp-le)
  have 3: lnth (kfilter P n xs) (k + (na)) < lnth (kfilter P n xs) (k + (Suc na))
    using 1 kfilter-mono by auto
  show ?thesis by (metis 1 2 3 add.commute lnth-ldropn)
qed
qed

```

```

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

```

```

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

```

```

lemma lfilter-kfilter-ltake-lset-eq:
  assumes
    k < llength(lfilter P xs)
  shows lset (ltake k (kfilter P 0 xs)) =

```

$lset (kfilter P 0 (ltake ((lnth (kfilter P 0 xs) k)) xs))$
proof –
have 1: $(lnth (kfilter P 0 xs) k) < llength xs$
using *kfilter-llength kfilter-upperbound*
by (*metis assms gen-llength-def kfilter-llength kfilter-upperbound llength-code*)
have 2: $\exists x \in lset xs . P x$
using *assms gr-implies-not-zero llength-eq-0 lnull-lfilter* **by** *blast*
have 3: $\{i. i < llength(ltake ((lnth (kfilter P 0 xs) k)) xs) \wedge$
 $P (lnth (ltake ((lnth (kfilter P 0 xs) k)) xs) i)\} =$
 $\{i. i < (lnth (kfilter P 0 xs) k) \wedge P (lnth xs i)\}$
by (*auto simp add: lnth-ltake 1 order-less-subst2*)
have 5: $\{i. i < (lnth (kfilter P 0 xs) k) \wedge P (lnth xs i)\} =$
 $\{i. i < (lnth (kfilter P 0 xs) k) \wedge i \in lset(kfilter P 0 xs)\}$
by (*auto simp add: 1 kfilter-holds-c order-less-subst2*)
have 6: $\{i. i < (lnth (kfilter P 0 xs) k) \wedge i \in lset(kfilter P 0 xs)\} =$
 $\{i. i < (lnth (kfilter P 0 xs) k) \wedge i \in \{(lnth (kfilter P 0 xs) j) \mid j. j < llength(kfilter P 0 xs)\}\}$
by (*simp add: lset-conv-lnth*)
have 7: $\{i. i < (lnth (kfilter P 0 xs) k) \wedge i \in \{(lnth (kfilter P 0 xs) j) \mid j. j < llength(kfilter P 0 xs)\}\} =$
 $\{(lnth (kfilter P 0 xs) j) \mid j. j < k\}$
by (*auto simp add: assms lidx-kfilter-gr kfilter-llength*)
(metis kfilter-llength leD lidx-kfilter-gr-eq not-le-imp-less,
metis assms enat-ord-simps(2) less-trans)
have 8: $k < llength(kfilter P 0 xs)$
by (*simp add: assms kfilter-llength*)
have 9: $\{(lnth (kfilter P 0 xs) j) \mid j. j < k\} = lset(ltake k (kfilter P 0 xs))$
using *ltake-lset[of k (kfilter P 0 xs)]* 8 **by** *auto*
have 10: $lset (kfilter P 0 (ltake ((lnth (kfilter P 0 xs) k)) xs)) =$
 $\{i. i < (lnth (kfilter P 0 xs) k) \wedge$
 $P (lnth (ltake (lnth (kfilter P 0 xs) k) xs) i)\}$
by (*auto simp add: in-kfilter-lset*)
(meson 1 enat-ord-simps(2) less-trans)
have 11: $(lnth (kfilter P 0 xs) k) = llength(ltake (lnth (kfilter P 0 xs) k) xs)$
by (*simp add: 1 dual-order.strict-implies-order min-def*)
have 12: $\{i. i < (lnth (kfilter P 0 xs) k) \wedge$
 $P (lnth (ltake (lnth (kfilter P 0 xs) k) xs) i)\} =$
 $\{i. i < llength(ltake (lnth (kfilter P 0 xs) k) xs) \wedge$
 $P (lnth (ltake (lnth (kfilter P 0 xs) k) xs) i)\}$
using 11 **by** *auto*
show *?thesis*
using 10 12 3 5 6 7 9 **by** *auto*
qed

lemma *lfilter-kfilter-ldropn-lset-eq*:
assumes $k < llength(lfilter P xs)$
shows $lset(kfilter P (lnth (kfilter P 0 xs) k) (ldropn (lnth (kfilter P 0 xs) k) xs)) =$
 $lset(ldropn k (kfilter P 0 xs))$
proof –
have 1: $(lnth (kfilter P 0 xs) k) < llength xs$
using *kfilter-llength kfilter-upperbound*
by (*metis assms gen-llength-def kfilter-llength kfilter-upperbound llength-code*)

have 2: $\exists x \in \text{lset } xs . P x$
using *assms gr-implies-not-zero llength-eq-0 lnull-lfilter* **by** *blast*
have 10: $\text{lset}(\text{kfilter } P (\text{lnth } (\text{kfilter } P 0 xs) k) (\text{ldropn } (\text{lnth } (\text{kfilter } P 0 xs) k) xs)) =$
 $\{(\text{lnth } (\text{kfilter } P 0 xs) k) + i \mid i. i < \text{llength } xs - (\text{lnth } (\text{kfilter } P 0 xs) k) \wedge$
 $P (\text{lnth } xs (i + (\text{lnth } (\text{kfilter } P 0 xs) k))) \}$
using 1 *kfilter-ldropn-lset-b[of (lnth (kfilter P 0 xs) k) xs P (lnth (kfilter P 0 xs) k)]*
by *linarith*
have 5: $\{(\text{lnth } (\text{kfilter } P 0 xs) k) + i \mid i. i < \text{llength } xs - (\text{lnth } (\text{kfilter } P 0 xs) k) \wedge$
 $P (\text{lnth } xs (i + (\text{lnth } (\text{kfilter } P 0 xs) k))) \} =$
 $\{(\text{lnth } (\text{kfilter } P 0 xs) k) + i \mid i. i < \text{llength } xs - (\text{lnth } (\text{kfilter } P 0 xs) k) \wedge$
 $(i + (\text{lnth } (\text{kfilter } P 0 xs) k)) \in \text{lset}(\text{kfilter } P 0 xs) \}$
using *kfilter-holds-b[of - xs 0 P]* **using** 1 2
by *auto*
(metis ldropn-eq-LNil ldropn-ldropn leD llength-ldropn not-le-imp-less,
metis gen-llength-def in-lset-conv-lnth kfilter-upperbound llength-code)
have 51: $\{(\text{lnth } (\text{kfilter } P 0 xs) k) + i \mid i. i < \text{llength } xs - (\text{lnth } (\text{kfilter } P 0 xs) k) \wedge$
 $(i + (\text{lnth } (\text{kfilter } P 0 xs) k)) \in \text{lset}(\text{kfilter } P 0 xs) \} =$
 $\{(\text{lnth } (\text{kfilter } P 0 xs) k) + i \mid i.$
 $(\text{lnth } (\text{kfilter } P 0 xs) k) \leq i + (\text{lnth } (\text{kfilter } P 0 xs) k) \wedge$
 $i + (\text{lnth } (\text{kfilter } P 0 xs) k) < \text{llength } xs \wedge$
 $(i + (\text{lnth } (\text{kfilter } P 0 xs) k)) \in \text{lset}(\text{kfilter } P 0 xs) \}$
by *auto*
(metis 1 enat-min less-imp-le plus-enat-simps(1),
metis ldropn-eq-LNil ldropn-ldropn leD llength-ldropn not-le-imp-less)
have 52: $\{(\text{lnth } (\text{kfilter } P 0 xs) k) + i \mid i.$
 $(\text{lnth } (\text{kfilter } P 0 xs) k) \leq i + (\text{lnth } (\text{kfilter } P 0 xs) k) \wedge$
 $i + (\text{lnth } (\text{kfilter } P 0 xs) k) < \text{llength } xs \wedge$
 $(i + (\text{lnth } (\text{kfilter } P 0 xs) k)) \in \text{lset}(\text{kfilter } P 0 xs) \} =$
 $\{j. (\text{lnth } (\text{kfilter } P 0 xs) k) \leq j \wedge j < \text{llength } xs \wedge j \in \text{lset}(\text{kfilter } P 0 xs) \}$
by *(metis (no-types, lifting) add commute le-iff-add)*
have 53: $\{j. (\text{lnth } (\text{kfilter } P 0 xs) k) \leq j \wedge j < \text{llength } xs \wedge j \in \text{lset}(\text{kfilter } P 0 xs) \} =$
 $\{j. (\text{lnth } (\text{kfilter } P 0 xs) k) \leq j \wedge j < \text{llength } xs \wedge$
 $j \in \{ (\text{lnth } (\text{kfilter } P 0 xs) jj) \mid jj. jj < \text{llength}(\text{kfilter } P 0 xs) \} \}$
by *(simp add: lset-conv-lnth)*
have 54: $\{j. (\text{lnth } (\text{kfilter } P 0 xs) k) \leq j \wedge j < \text{llength } xs \wedge$
 $j \in \{ (\text{lnth } (\text{kfilter } P 0 xs) jj) \mid jj. jj < \text{llength}(\text{kfilter } P 0 xs) \} \} =$
 $\{ (\text{lnth } (\text{kfilter } P 0 xs) j) \mid j. k \leq j \wedge j < \text{llength } (\text{kfilter } P 0 xs) \}$
by *auto*
(metis assms kfilter-llength lidx-kfilter-gr not-less,
meson lidx-kfilter-gr-eq,
metis gen-llength-def kfilter-upperbound llength-code)
have 8: $k < \text{llength}(\text{kfilter } P 0 xs)$
by *(simp add: assms kfilter-llength)*
have 9: $\{ (\text{lnth } (\text{kfilter } P 0 xs) j) \mid j. k \leq j \wedge j < \text{llength } (\text{kfilter } P 0 xs) \} =$
 $\text{lset}(\text{ldropn } k (\text{kfilter } P 0 xs))$
using *ldropn-lset[of k (kfilter P 0 xs)]* **using** 8 **by** *blast*
show *?thesis*
using 10 5 51 52 53 54 9 **by** *auto*
qed

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

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

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

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

lemma *kfilter-lmap-shift-ldropn*:

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

proof –

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

have 2: $\bigwedge i. i < \text{llength } (\text{lmap } (\lambda s. \text{lnth } xs \ (s + (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k))))$
 $(\text{kfilter } P \ 0 \ (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)) \implies$
 $\text{lnth } (\text{lmap } (\lambda s. \text{lnth } xs \ (s + (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k))))$
 $(\text{kfilter } P \ 0 \ (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)) \ i =$
 $\text{lnth } xs \ ((\text{lnth } (\text{kfilter } P \ 0 \ (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)) \ i) +$
 $(\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k))$

by *simp*

have 3: $\bigwedge i. i < \text{llength } (\text{lmap } (\lambda s. \text{lnth } xs \ s))$
 $(\text{kfilter } P \ (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)) \implies$
 $\text{lnth } (\text{lmap } (\lambda s. \text{lnth } xs \ s))$
 $(\text{kfilter } P \ (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k)$
 $(\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)) \ i =$
 $\text{lnth } xs \ (\text{lnth } (\text{kfilter } P \ (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k)$
 $(\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)) \ i)$

by *simp*

have 4: $\bigwedge i. i < \text{llength } (\text{lmap } (\lambda s. \text{lnth } xs \ (s + (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k))))$
 $(\text{kfilter } P \ 0 \ (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)) \implies$
 $\text{lnth } xs \ ((\text{lnth } (\text{kfilter } P \ 0 \ (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)) \ i) +$
 $(\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k)) =$
 $\text{lnth } xs \ (\text{lnth } (\text{kfilter } P \ (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k)$
 $(\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)) \ i)$

using *kfilter-lnth-n-zero*[of - $P \ (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)$]
kfilter-lowerbound[of - $P \ (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs)$]

using 1 *diff-add* **by** *fastforce*

show *?thesis*

using *llist-eq-lnth-eq*[of $\text{lmap } (\lambda s. \text{lnth } xs \ (s + (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k)))$
 $(\text{kfilter } P \ 0 \ (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs))]$

using 1 2 3 4 **by** *presburger*

qed

lemma *lfilter-kfilter-ldropn*:

assumes $k < \text{llength}(\text{lfilter } P \ xs)$
shows $(\text{ldropn } k \ (\text{lfilter } P \ xs)) =$
 $(\text{lfilter } P \ (\text{ldropn } (\text{lnth } (\text{kfilter } P \ 0 \ xs) \ k) \ xs))$

proof –

have 1: $\exists x \in \text{lset } xs. P \ x$
using *assms gr-implies-not-zero llength-eq-0 lnull-lfilter* **by** *blast*
have 2: $(\text{lfilter } P \ xs) = \text{lmap } (\lambda s. \text{lnth } xs \ s) \ (\text{kfilter } P \ 0 \ xs)$

by (simp add: kfilter-lmap-lfilter)
 have 3: ldrop k (lfilter P xs) =
 ldrop k (lmap (λs. lnth xs s) (kfilter P 0 xs))
 by (simp add: 2)
 have 4: (ldrop k (lmap (λs. lnth xs s) (kfilter P 0 xs))) =
 (lmap (λs. lnth xs s) (ldrop k (kfilter P 0 xs)))
 using ldrop-lmap by blast
 have 6: (lfilter P (ldropn (lnth (kfilter P 0 xs) k) xs)) =
 lmap (λs. lnth (ldropn (lnth (kfilter P 0 xs) k) xs) s)
 (kfilter P 0 (ldropn (lnth (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
 (metis add.commute add.left-neutral kfilter-upperbound ldropn-eq-LNil ldropn-ldropn leD
 lnth-ldropn not-le-imp-less zero-enat-def)
 have 7: lmap (λs. lnth (ldropn (lnth (kfilter P 0 xs) k) xs) s)
 (kfilter P 0 (ldropn (lnth (kfilter P 0 xs) k) xs)) =
 lmap (λs. lnth xs (s + (lnth (kfilter P 0 xs) k)))
 (kfilter P 0 (ldropn (lnth (kfilter P 0 xs) k) xs))
 using 61 by auto
 have 8: lmap (λs. lnth xs (s + (lnth (kfilter P 0 xs) k)))
 (kfilter P 0 (ldropn (lnth (kfilter P 0 xs) k) xs)) =
 lmap (λs. lnth xs s)
 (kfilter P (lnth (kfilter P 0 xs) k) (ldropn (lnth (kfilter P 0 xs) k) xs))
 by (simp add: kfilter-lmap-shift-ldropn)
 have 9: lmap (λs. lnth xs s)
 (kfilter P (lnth (kfilter P 0 xs) k) (ldropn (lnth (kfilter P 0 xs) k) xs)) =
 lmap (λs. lnth xs s) (ldropn k (kfilter P 0 xs))
 by (simp add: assms kfilter-kfilter-ldropn)
 show ?thesis
 by (metis 4 7 8 9 kfilter-lmap-lfilter ldrop-enat)

qed

lemma lfilter-lnth-aa:

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

shows $P (\text{lnth } (\text{lfilter } P \ xs) \ n)$

using assms

by (meson in-lset-conv-lnth lfilter-id-conv lfilter-idem)

lemma exist-one-conv:

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

$(\exists k < \text{llength } xs. P (\text{lnth } xs \ k) \wedge$

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

by blast

lemma lfilter-llength-one-conv-a:

assumes $\text{llength}(\text{lfilter } P \ xs) = 1$

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

lemma lfilter-llength-one-conv-c:
 $(\exists k < \text{length } xs. P (\text{lnth } xs \ k) \wedge$
 $(\forall j < \text{length } xs. j \neq k \longrightarrow \neg P (\text{lnth } xs \ j))) \longleftrightarrow$
 $(\exists k < \text{length } xs. P (\text{lnth } xs \ k) \wedge$
 $(\forall j < \text{length } xs. j < k \vee k < j \longrightarrow \neg P (\text{lnth } xs \ j)))$
using antisym-conv3 **by** auto

lemma lfilter-llength-one-conv-d:
 $(\exists k < \text{length } xs. P (\text{lnth } xs \ k) \wedge$
 $(\forall j < \text{length } xs. j < k \vee k < j \longrightarrow \neg P (\text{lnth } xs \ j))) \longleftrightarrow$
 $(\exists k < \text{length } xs. P (\text{lnth } xs \ k) \wedge$
 $(\forall j. j < k \longrightarrow \neg P (\text{lnth } xs \ j)) \wedge$
 $(\forall j < \text{length } xs. k < j \longrightarrow \neg P (\text{lnth } xs \ j)))$
by (meson enat-ord-simps(2) less-trans)

lemma exist-one-lfilter-llength-one:
assumes $(\exists! i. i < \text{length } xs \wedge P (\text{lnth } xs \ i))$
shows $\text{length}(\text{lfilter } P \ xs) \leq 1$
using assms
proof auto
fix $i :: \text{nat}$
assume a1: $\forall y \ y'. \text{enat } y < \text{length } xs \wedge P (\text{lnth } xs \ y) \wedge \text{enat } y' < \text{length } xs \wedge P (\text{lnth } xs \ y') \longrightarrow$
 $y = y'$
show $\text{length} (\text{lfilter } P \ xs) \leq 1$
proof –
have f1: $\forall e \ ea. (e :: \text{enat}) \leq ea \vee ea < e$
by (meson not-le-imp-less)
have f3: $\text{length} (\text{kfilter } P \ 0 \ xs) = \text{gen-llength } 0 (\text{lfilter } P \ xs)$
by (metis kfilter-llength llength-code)
have f4: $\text{enat } 0 + \text{length } xs = \text{length } xs$
by (metis gen-llength-def llength-code)
have $\text{length } xs = \text{enat } 0 + \text{length } xs$
by (metis (full-types) gen-llength-def llength-code)
then have f5: $\neg 1 < \text{length} (\text{lfilter } P \ xs)$

```

proof –
  have g1:  $Suc\ 0 = 0 + Suc\ 0$ 
    by auto
  have g2:  $\bigwedge i. enat\ i < llength\ (kfilter\ P\ 0\ xs) \implies$ 
     $lnth\ xs\ (lnth\ (kfilter\ P\ 0\ xs)\ i - 0) = lnth\ (lfilter\ P\ xs)\ i$ 
    using lfilter-kfilter[of - P 0 xs] by auto
  have g3:  $\bigwedge k. enat\ k < llength\ (kfilter\ P\ 0\ xs) \implies$ 
     $lnth\ (kfilter\ P\ 0\ xs)\ k - 0 = lnth\ (kfilter\ P\ 0\ xs)\ k$ 
    using kfilter-mono[of - P 0 xs] by auto
  have g4:  $\bigwedge n. enat\ n < llength\ (lfilter\ P\ xs) \implies P\ (lnth\ (lfilter\ P\ xs)\ n)$ 
    using lfilter-lnth-aa[of - P xs] by auto
  have g5:  $enat\ (0 + Suc\ 0) = 1$ 
    using one-enat-def by auto
  have  $\neg\ 1 < gen-llength\ 0\ (lfilter\ P\ xs) \vee enat\ 0 < gen-llength\ 0\ (lfilter\ P\ xs)$ 
    using zero-enat-def by fastforce
  then show ?thesis
    by (metis g1 g2 g3 g4 g5 a1 f3 f4 kfilter-upperbound ldistinct-conv-lnth ldistinct-kfilter
      llength-code n-not-Suc-n)
  qed
show ?thesis
using f5 not-le by blast
qed
qed

```

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

assumes $\exists\ k < llength\ xs. P\ (lnth\ xs\ k) \wedge$
 $(\forall\ j < llength\ xs. j \neq k \longrightarrow \neg\ P\ (lnth\ xs\ j))$

shows $llength(lfilter\ P\ xs) = 1$

proof –

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

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

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

by (*metis* *assms* *gr-zeroI* *in-lset-conv-lnth* *llength-eq-0* *lnull-lfilter*)

show *?thesis*

by (*metis* 1 2 *One-nat-def* *Suc-ile-eq* *dual-order.antisym* *one-enat-def* *zero-enat-def*)

qed

lemma *lfilter-llength-one-conv*:

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

using *lfilter-llength-one-conv-a[of P xs]* *lfilter-llength-one-conv-b[of xs P]* **by** *blast*

lemma *lfilter-llength-one-conv-1*:

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

using *lfilter-llength-one-conv[of xs P]* *lfilter-llength-one-conv-c[of xs P]*

by *blast*

lemma *lfilter-llength-one-conv-2:*

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

lemma *lfilter-lappend-ltake:*

assumes $k < \text{llength } xs$
shows $\text{lfilter } P \ (\text{ltake } k \ xs) = \text{ltake } (\text{llength}(\text{lfilter } P \ (\text{ltake } k \ xs))) \ (\text{lfilter } P \ xs)$
proof –
have 1: $\text{lfilter } P \ xs =$
 $\text{lappend } (\text{lfilter } P \ (\text{ltake } (\text{the-enat } k) \ xs)) \ (\text{lfilter } P \ (\text{ldropn } (\text{the-enat } k) \ xs))$
by (*metis enat-ord-code*(4) *lappend-ltake-enat-ldropn lfilter-lappend-lfinite lfinite-ltake*)
have 2: $\text{ltake } (\text{llength}(\text{lfilter } P \ (\text{ltake } k \ xs)))$
 $(\text{lappend } (\text{lfilter } P \ (\text{ltake } (\text{the-enat } k) \ xs)) \ (\text{lfilter } P \ (\text{ldropn } (\text{the-enat } k) \ xs))) =$
 $\text{lfilter } P \ (\text{ltake } k \ xs)$
by (*metis assms enat-the-enat lfilter-idem lfinite-ltake llength-eq-inf-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:*

lfinite xs \implies
 $kfilter \ P \ n \ (\text{lappend } xs \ ys) =$
 $\text{lappend } (kfilter \ P \ n \ xs) \ (kfilter \ P \ (n + (\text{the-enat}(\text{llength } xs))) \ ys)$
unfolding *kfilter-def*
proof (*induct arbitrary: n rule: lfinite.induct*)
case *lfinite-LNil*
then show ?case **by** *simp*
next
case (*lfinite-LConsI xs x*)
then show ?case
proof –
have 1: $(\text{lzip } (\text{lappend } (\text{LCons } x \ xs) \ ys) \ (\text{iterates } \text{Suc } n)) =$
 $(\text{LCons } (x, n) \ (\text{lzip } (\text{lappend } xs \ ys) \ (\text{iterates } \text{Suc } (\text{Suc } n))))$
by (*simp add: lzip.ctr*(2))
have 3: $\text{lmap } \text{snd} \ (\text{lfilter } (P \circ \text{fst}) \ (\text{lzip } (\text{lappend } (\text{LCons } x \ xs) \ ys) \ (\text{iterates } \text{Suc } n))) =$
 $(\text{if } P \ x \ \text{then } (\text{LCons } n \ (\text{lmap } \text{snd} \ (\text{lfilter } (P \circ \text{fst}) \ (\text{lzip } (\text{lappend } xs \ ys) \ (\text{iterates } \text{Suc } (\text{Suc } n))))))$
 $\text{else } \text{lmap } \text{snd} \ (\text{lfilter } (P \circ \text{fst}) \ (\text{lzip } (\text{lappend } xs \ ys) \ (\text{iterates } \text{Suc } (\text{Suc } n))))$
using 1 **by** *auto*
have 4: $(\text{if } P \ x \ \text{then } (\text{LCons } n \ (\text{lmap } \text{snd} \ (\text{lfilter } (P \circ \text{fst}) \ (\text{lzip } (\text{lappend } xs \ ys) \ (\text{iterates } \text{Suc } (\text{Suc } n))))))$
 $\text{else } \text{lmap } \text{snd} \ (\text{lfilter } (P \circ \text{fst}) \ (\text{lzip } (\text{lappend } xs \ ys) \ (\text{iterates } \text{Suc } (\text{Suc } n)))) =$
 $(\text{if } P \ x \ \text{then}$
 $(\text{LCons } n \ (\text{lappend } (\text{lmap } \text{snd} \ (\text{lfilter } (P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } (\text{Suc } n))))$
 $(\text{lmap } \text{snd} \ (\text{lfilter } (P \circ \text{fst}) \ (\text{lzip } ys \ (\text{iterates } \text{Suc } ((\text{Suc } n) + \text{the-enat } (\text{llength } xs))))))))$
 $\text{else } (\text{lappend } (\text{lmap } \text{snd} \ (\text{lfilter } (P \circ \text{fst}) \ (\text{lzip } xs \ (\text{iterates } \text{Suc } (\text{Suc } n))))$
 $(\text{lmap } \text{snd} \ (\text{lfilter } (P \circ \text{fst}) \ (\text{lzip } ys \ (\text{iterates } \text{Suc } ((\text{Suc } n) + \text{the-enat } (\text{llength } xs))))))))$

```

using lfinite-LConsI.hyps(2) by auto
have 5: (lmap snd (lfilter (P ∘ fst) (lzip (LCons x xs) (iterates Suc n)))) =
  (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)))))))
  by (simp add: lzip.ctr(2))
have 6: (lmap snd (lfilter (P ∘ fst) (lzip ys (iterates Suc (n + the-enat (llength (LCons x xs)))))) =
  (lmap snd (lfilter (P ∘ fst) (lzip ys (iterates Suc ((Suc n) + the-enat (llength (xs)))))))
using eSuc-enat lfinite-LConsI.hyps(1) llength-eq-infty-conv-lfinite by force
have 7: (lappend (lmap snd (lfilter (P ∘ 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

```

1.4 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)
  (metis (full-types, hide-lams) llist-all2-conv-all-lnth)

```

```

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**

LListKFilter

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: $\bigwedge x. nlast0\ x = nlast\ x$

by (*simp add: nlast0.simps nlast-def*)

show ?thesis **using** 1 **by** (*rule ext*)

qed

lemmas *nlast-NNil* [*code*, *nitpick-simp*] = *nellist.sel(1)*

lemma *nlast-NCons* [*simp*, *code*, *nitpick-simp*]: *nlast* (*NCons* *x nell*) = *nlast nell*

by *simp*

declare *nellist.sel(2)* [*simp del*]

definition *nfirst* :: *'a nellist \Rightarrow 'a*

where *nfirst nell* = (case *nell* of (*NNil* *b*) \Rightarrow *b* | (*NCons* *b nell'*) \Rightarrow *b*)

primcorec *unfold-nellist* :: (*'a \Rightarrow bool*) \Rightarrow (*'a \Rightarrow 'b*) \Rightarrow (*'a \Rightarrow 'b*) \Rightarrow (*'a \Rightarrow 'a*) \Rightarrow *'a \Rightarrow 'b nellist*

where

$p\ a \implies \text{unfold-nellist } p\ g1\ g21\ g22\ a = \text{NNil } (g1\ a) \mid$
 $- \implies \text{unfold-nellist } p\ g1\ g21\ g22\ a =$
 $\text{NCons } (g21\ a) (\text{unfold-nellist } p\ g1\ g21\ g22\ (g22\ a))$

declare

$\text{unfold-nellist.ctr}(1) \text{ [simp]}$
 $\text{nellist.corec}(1) \text{ [simp]}$

2.2 Code generator setup

More lemmas about generated constants

lemma *ntl-unfold-nellist*:

$\text{ntl } (\text{unfold-nellist } \text{IS-NNIL } \text{NNIL } \text{NHD } \text{NTL } a) =$
 $(\text{if } \text{IS-NNIL } a \text{ then } \text{NNil } (\text{NNIL } a) \text{ else } \text{unfold-nellist } \text{IS-NNIL } \text{NNIL } \text{NHD } \text{NTL } (\text{NTL } a))$

by(*simp*)

lemma *is-NNil-ntl* [simp]:

$\text{is-NNil } \text{nell} \implies \text{is-NNil } (\text{ntl } \text{nell})$

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]:

$\text{unfold-nellist } \text{IS-NNIL } \text{NNIL } \text{NHD } \text{NTL } a = \text{NNil } b \longleftrightarrow \text{IS-NNIL } a \wedge b = \text{NNIL } a$

by(*auto simp add: unfold-nellist.code*)

lemma *NNil-eq-unfold-nellist* [simp]:

$\text{NNil } b = \text{unfold-nellist } \text{IS-NNIL } \text{NNIL } \text{NHD } \text{NTL } a \longleftrightarrow \text{IS-NNIL } a \wedge b = \text{NNIL } a$

by(*auto simp add: unfold-nellist.code*)

lemma *nmap-is-NNil*:

$\text{is-NNil } \text{nell} \implies \text{nmap } f\ \text{nell} = \text{NNil } (f\ (\text{nlast } \text{nell}))$

by(*clarsimp simp add: is-NNil-def*)

declare $\text{nellist.map-sel}(2) \text{ [simp]}$

lemma *ntl-nmap* [simp]:

$\text{ntl } (\text{nmap } f\ \text{nell}) = \text{nmap } f\ (\text{ntl } \text{nell})$

by(*cases nell simp-all*)

lemma *nmap-eq-NNil-conv*:

$\text{nmap } f\ \text{nell} = \text{NNil } y \longleftrightarrow (\exists y'. \text{nell} = \text{NNil } y' \wedge f\ y' = y)$

by(*cases nell simp-all*)

lemma *NNil-eq-nmap-conv*:

$\text{NNil } y = \text{nmap } f\ \text{nell} \longleftrightarrow (\exists y'. \text{nell} = \text{NNil } y' \wedge f\ y' = y)$

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 $\bigwedge nell\ y. [\neg is-NNil\ nell; y \in nset (ntl\ nell); P\ y\ (ntl\ nell)] \implies P\ y\ nell$
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*: $\bigwedge x'\ nell. [x \in nset\ nell; P\ nell] \implies P (NCons\ x'\ nell)$
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 nellist.collapse(1) nellist.map-disc-iff nellist.map-sel(1)*)
next
case (*find nell*)
then show ?*case* **by** (*metis 1 nellist.collapse(2)*)
next
case (*step nell*)
then show ?*case* **by** (*metis 2 nellist.collapse(2)*)
qed

lemma *nset-induct* [*consumes 1, case-names findnil find step, induct set: nset*]:
assumes *major*: $x \in nset\ nell$
and *findnil*: $\bigwedge nell. is-NNil\ nell \implies P (nlast\ nell)$
and *find*: $\bigwedge nell. P (NCons\ x\ nell)$
and *step*: $\bigwedge x'\ nell. [x \in nset\ nell; x \neq x'; P\ nell] \implies P (NCons\ x'\ nell)$
shows $P\ nell$

```

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 |
                                NCons x nell'  $\Rightarrow$  LCons x (llist-of-nellist nell'))

```

```

context
fixes
b :: 'a
begin
primcorec nellist-of-llist-a :: 'a llist  $\Rightarrow$  'a nellist where
  nellist-of-llist-a ll = (case ll of LNil  $\Rightarrow$  NNil b |
                           LCons x ll'  $\Rightarrow$  NCons x (nellist-of-llist-a ll'))
end

```

```

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

```

```

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

```

```

lemmas nellist-of-llist-a-LNil = nellist-of-llist-a-simps(1)
and nellist-of-llist-a-LCons = nellist-of-llist-a-simps(2)

```

```

lemma nlast-nellist-of-llist-a-lnull [simp]:
  lnull ll  $\implies$  nlast (nellist-of-llist-a b ll) = b
unfolding lnull-def by simp

```

```

lemma not-llnull-expand:
   $\neg$  lnull ll  $\longleftrightarrow$  ( $\exists$  b. ll = (LCons b LNil))  $\vee$  ( $\exists$  b ll'. ll = (LCons b ll')  $\wedge$   $\neg$  lnull ll')
by (metis lhd-LCons-ltl llist.disc(1) llist.disc(2) llist.expand)

```

```

lemma is-LEmpty-llength:
  ( $\exists$  b. ll = (LCons b LNil))  $\longleftrightarrow$  llength ll = 1
by (metis epred-llength llength-LCons llength-eq-0 llist.disc(1) ltl-simps(2) not-llnull-expand
      one-eSuc)

```

```

lemma is-LMore-llength:
  ( $\exists$  b ll'. ll = (LCons b ll')  $\wedge$   $\neg$  lnull ll')  $\longleftrightarrow$  1 < llength ll
by (metis dual-order.strict-implies-not-eq gr-implies-not-zero lhd-LCons-ltl llength-LCons)

```

llength-eq-0 lstrict-prefix-code(2) lstrict-prefix-code(4) lstrict-prefix-length-less one-eSuc)

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 | (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*) \Rightarrow (*LCons b LNil*) |

(*NCons b nell'*) \Rightarrow *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' \Longrightarrow b = b'*

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

proof(*unfold* (*ll = ll'*))

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*) \Rightarrow *NCons x (NNil a)* |

(*NCons x nell'*) \Rightarrow *NCons x (snocn nell' a)*)

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

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

shows *nhd-snocn: nhd(snocn nell a) = nfirst nell*

and *ntl-snocn: ntl(snocn nell a) = (if is-NNil nell then (NNil a) else snocn (ntl nell) a)*

by (*case-tac* [*!*] *nell*)

(*auto simp add: nfirst-def*)

lemma *is-NNil-snocn*:

is-NNil(snocn nell a) \longleftrightarrow False

by (*auto simp add: snocn-def*)

lemma *nmap-snocn-distrib*:

nmap f (snocn nell a) = snocn (nmap f nell) (f a)

proof (*coinduction arbitrary: nell rule: nellist.coinduct-strong*)

case (*Eq-nellist nella*)

then show *?case*

by (*auto simp add: nellist.case-eq-if nellist.map-sel(1)*)

qed

definition *nfinite* :: '*a* *nellist* \Rightarrow *bool*

where *nfinite nell \equiv lfinite (llist-of-nellist nell)*

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

assumes *nfinite nell*

and $\bigwedge y. P (NNil y)$

and $\bigwedge x nell. \llbracket nfinite nell; P nell \rrbracket \Longrightarrow P (NCons x nell)$

shows *P nell*

```

using assms
unfolding nfinite-def
proof (induct ll $\equiv$ llist-of-nellist nell arbitrary: nell rule: 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 nell  $\implies$  nfinite (NCons x nell)
unfolding nfinite-def
by auto

```

```

declare nfinite-NNil [iff]

```

```

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

```

```

lemma nfinite-NCons [simp]:
  nfinite (NCons x nell) = nfinite nell
by (simp add: nfinite-def)

```

```

lemma nfinite-ntl [simp]:
  nfinite (ntl nell) = nfinite nell
by (cases nell) simp-all

```

```

lemma nfinite-code [code]:
  nfinite (NNil x) = True
  nfinite (NCons x nell) = nfinite nell
by simp-all

```

```

lemma nfinite-imp-finite-nset:
  assumes nfinite nell
  shows finite (nset nell)
using assms
by (induct nell rule:nfinite-induct) simp-all

```

```

lemma nfinite-snocn [simp]:
  nfinite(snocn nell a)  $\longleftrightarrow$  nfinite nell
  (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
  assume ?lhs thus ?rhs
  proof (induct zs $\equiv$ snocn nell a arbitrary: nell rule: nfinite-induct)
  case (NNil y)
  then show ?case

```

```

by (metis is-NNil-snocn 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*:

```

 $\neg$  nfinite nell  $\implies$  snocn nell a = nell
proof (coinduction arbitrary: nell)
case (Eq-nellist nella)
then show ?case
  proof –
    have 1: is-NNil (snocn nella a) = is-NNil nella
      using Eq-nellist by auto
    have 2: (is-NNil (snocn nella a)  $\longrightarrow$  is-NNil nella  $\longrightarrow$  nlast (snocn nella a) = nlast nella)
      by auto
    have 3: ( $\neg$  is-NNil (snocn nella a)  $\longrightarrow$   $\neg$  is-NNil nella  $\longrightarrow$ 
      nhd (snocn nella a) = nhd nella  $\wedge$ 
      ( $\exists$  nell. ntl (snocn nella a) = snocn nell a  $\wedge$  ntl nella = nell  $\wedge$   $\neg$  nfinite nell))
      by (simp add: Eq-nellist nellist.case-eq-if)
    from 1 2 3 show ?thesis by blast
  qed
qed

```

lemma *nfinite-nmap* [simp]:

```

nfinite (nmap f nell) = nfinite nell (is ?lhs  $\longleftrightarrow$  ?rhs)
proof
assume ?lhs thus ?rhs
  proof (induct zs $\equiv$ nmap f nell arbitrary: nell rule: nfinite-induct)
  case (NNil y)
  then show ?case by (metis nellist.disc(1) nellist.map-disc-iff is-NNil-imp-nfinite)
  next
  case (NCons x nell)
  then show ?case by (metis nellist.sel(5) nfinite-ntl ntl-nmap)
  qed
next
assume ?rhs thus ?lhs
  by (induct rule: nfinite-induct) simp-all
qed

```

lemma *nset-snocn-nfinite* [simp]:

```

nfinite nell  $\implies$  nset(snocn nell a) = nset nell  $\cup$  {a}
by (induct rule: nfinite-induct) auto

```

lemma *nset-snocn1*:

$nset (snocn\ nell\ a) \subseteq nset\ nell \cup \{a\}$
proof (*cases nfinite nell*)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
then show *?thesis* **by** (*auto simp add: snocn-inf*)
qed

lemma *nset-snocn-conv*:
 $nset (snocn\ nell\ a) = (if\ nfinite\ nell\ then\ nset\ nell \cup \{a\}\ else\ nset\ nell)$
by (*simp add: snocn-inf*)

lemma *in-nset-snocn-iff*:
 $x \in nset (snocn\ nell\ a) \longleftrightarrow x \in nset\ nell \vee nfinite\ nell \wedge x = a$
by (*metis Un-iff empty-iff insert-iff nset-snocn-conv*)

lemma *llist-of-nellist-inverse-1*:
assumes $\neg nfinite\ nell$
shows $nellist\text{-}of\text{-}l\text{-}list\text{-}a\ b\ (l\text{-}list\text{-}of\text{-}nellist\ nell) = snocn\ nell\ b$
using *assms*
proof (*coinduction arbitrary: nell*)
case (*Eq-nellist nella*)
then show *?case*
proof –
have 1: $is\text{-}NNil\ (nellist\text{-}of\text{-}l\text{-}list\text{-}a\ b\ (l\text{-}list\text{-}of\text{-}nellist\ nella)) = is\text{-}NNil\ (snocn\ nella\ b)$
by *auto*
have 2: $(is\text{-}NNil\ (nellist\text{-}of\text{-}l\text{-}list\text{-}a\ b\ (l\text{-}list\text{-}of\text{-}nellist\ nella)) \longrightarrow is\text{-}NNil\ (snocn\ nella\ b)) \longrightarrow nlast\ (nellist\text{-}of\text{-}l\text{-}list\text{-}a\ b\ (l\text{-}list\text{-}of\text{-}nellist\ nella)) = nlast\ (snocn\ nella\ b)$
by *simp*
have 3: $(\neg is\text{-}NNil\ (nellist\text{-}of\text{-}l\text{-}list\text{-}a\ b\ (l\text{-}list\text{-}of\text{-}nellist\ nella)) \longrightarrow \neg is\text{-}NNil\ (snocn\ nella\ b)) \longrightarrow nhd\ (nellist\text{-}of\text{-}l\text{-}list\text{-}a\ b\ (l\text{-}list\text{-}of\text{-}nellist\ nella)) = nhd\ (snocn\ nella\ b) \wedge (\exists\ nell. ntl\ (nellist\text{-}of\text{-}l\text{-}list\text{-}a\ b\ (l\text{-}list\text{-}of\text{-}nellist\ nella)) = nellist\text{-}of\text{-}l\text{-}list\text{-}a\ b\ (l\text{-}list\text{-}of\text{-}nellist\ nell) \wedge ntl\ (snocn\ nella\ b) = snocn\ nell\ b \wedge \neg nfinite\ nell)$
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 $nellist\text{-}of\text{-}l\text{-}list\text{-}a\ b\ (l\text{-}list\text{-}of\text{-}nellist\ nell) = snocn\ nell\ b$
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.disc*(1) *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 $llist\text{-}of\text{-}nellist\ (nellist\text{-}of\text{-}llist\text{-}a\ b\ ll) = lappend\ ll\ (LCons\ b\ LNil)$
using *nellist-of-llist-a-inverse-1 nellist-of-llist-a-inverse-2* **by** *metis*

lemma *nellist-of-llist-inverse* [simp]:
assumes $\neg\ lnull\ ll$
shows $llist\text{-}of\text{-}nellist\ (nellist\text{-}of\text{-}llist\ ll) = ll$
using *assms* **by** *simp*

lemma *nmap-nellist-of-llist-a*:
 $nmap\ f\ (nellist\text{-}of\text{-}llist\text{-}a\ b\ ll) = nellist\text{-}of\text{-}llist\text{-}a\ (f\ b)\ (lmap\ f\ ll)$
by (*coinduction arbitrary: ll*) (*auto simp add: nmap-is-NNil*)

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

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

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

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

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

lemma *Domainp-cr-nellist* [transfer-domain-rule]:
 $Domainp\ cr\text{-}nellist = (\lambda ll.\ \neg\ (lnull\ ll))$
unfolding *cr-nellist-def* *Domainp-iff[abs-def]*
using *Domainp-help* **by** *fastforce*

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

*l*list-of-nellist nelly = *l*list-of-nellist nellz \implies nelly = nellz
by (coinduction arbitrary: nelly nellz)
 (metis *l*list-of-nellist-inverse-b)

lemma quotient-help-2:
 (\neg lnull (*l*list-of-nellist nell) \wedge nellist-of-*l*list (*l*list-of-nellist nell) = nell)
by (simp add: bi-unique-cr-nellist-help)

lemma quotient-help-nellist-1:
 cr-nellist ll nell \longrightarrow nellist-of-*l*list ll = nell
by (metis cr-nellist-def *l*list-of-nellist-inverse-b)

lemma quotient-help-nellist-2:
 (cr-nellist (*l*list-of-nellist nell) nell)
by (simp add: cr-nellist-def)

lemma quotient-help-3-nellist:
 ((\neg lnull llx \wedge llx = lly) =
 (cr-nellist llx (nellist-of-*l*list llx) \wedge
 cr-nellist lly (nellist-of-*l*list lly) \wedge
 nellist-of-*l*list llx =
 nellist-of-*l*list lly))
by (metis Domainp.DomainI Domainp-cr-nellist cr-nellist-def nellist-of-*l*list-inverse)

lemma Quotient-nellist:
 Quotient (λ llx lly. \neg lnull llx \wedge llx = lly)
 nellist-of-*l*list *l*list-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 \implies pcr-nellist A) (λ b. LCons b LNil) NNil
by (auto simp add: pcr-nellist-def OO-def cr-nellist-def rel-fun-def)

lemma *NCons-transfer* [transfer-rule]:

($A \implies \text{pcr-nellist } A \implies \text{pcr-nellist } A$) $LCons$ $NCons$
by (*auto simp add: pcr-nellist-def OO-def cr-nellist-def rel-fun-def*)

lemma *nmap-transfer* [transfer-rule]:

(($=$) $\implies \text{pcr-nellist } (=) \implies \text{pcr-nellist } (=)$) $lmap$ $nmap$
by (*auto simp add: cr-nellist-def nellist.pcr-cr-eq rel-fun-def lmap-llist-of-nellist*)

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

lemma *is-NNil-transfer* [transfer-rule]:

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

lemma *nfirst-transfer-a* [transfer-rule]:

($\text{pcr-nellist } (=) \implies (=)$) *lhd nfirst*
by (*simp add: cr-nellist-def nellist.pcr-cr-eq nfirst-def rel-fun-def llist-of-nellist.simps(2)*)

lemma *nhd-transfer-a1* [transfer-rule]:

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

lemma *ntl-transfer* [transfer-rule]:

($\text{pcr-nellist } A \implies \text{pcr-nellist } A$) ($\lambda ll. \text{if is-lfirst } ll \text{ then } ll \text{ else ltl } ll$) *ntl*
proof (*auto simp add: pcr-nellist-def cr-nellist-def OO-def rel-fun-def intro!: llist-all2-ltlI*
lfinite-LConsI dest: llist-all2-lnullD)
show $\bigwedge b\ y. \text{llist-all2 } A\ (LCons\ b\ LNil)\ (\text{llist-of-nellist } y) \implies$
 $\text{llist-all2 } A\ (LCons\ b\ LNil)\ (\text{llist-of-nellist } (ntl\ y))$
using *llist-all2-ltlI*
by (*metis eq-LConsD is-NNil-ntl llist-all2-LNil1 llist-of-nellist.code llist-of-nellist.simps(3)*
llist-of-nellist-eq-LNil ltl-llist-of-nellist nlast-ntl)
next
show $\bigwedge x\ y. \forall b. x \neq LCons\ b\ LNil \implies \text{llist-all2 } A\ x\ (\text{llist-of-nellist } y) \implies$
 $\text{llist-all2 } A\ (ltl\ x)\ (\text{llist-of-nellist } (ntl\ y))$
by (*metis lhd-LCons-ltl llist-all2-LNil2 llist-all2-lnullD llist-all2-ltlI*
llist-of-nellist.disc-iff ltl-llist-of-nellist ltl-llist-of-nellist1)
qed

lemma *nfinite-transfer* [transfer-rule]:

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

lemma *llist-of-nellist-transfer* [transfer-rule]:

($\text{pcr-nellist } (=) \implies (=)$) *id llist-of-nellist*
by (*simp add: pcr-nellist-def cr-nellist-def OO-def rel-fun-def llist.rel-eq*)

lemma *nellist-of-llist-a-transfer* [transfer-rule]:

((=) ==> (=) ==> pcr-nellist (=)) (λb ll. lappend ll (LCons b LNil)) nellist-of-llist-a
by (auto simp add: pcr-nellist-def cr-nellist-def OO-def rel-fun-def)

lemma *nlast-nellist-of-llist-a-lfinite* [simp]:

lfinite ll ==> nlast (nellist-of-llist-a b ll) = b
by(induct rule: lfinite.induct) simp-all

lemma *snocn-transfer* [transfer-rule]:

(pcr-nellist (A) ==> (A) ==> pcr-nellist (A)) (λ ll a. lappend ll (LCons a LNil)) snocn
unfolding rel-fun-def
by (auto simp add: pcr-nellist-def nellist.pcr-cr-eq cr-nellist-def OO-def)
 (metis LNil-transfer llist.rel-intros(2) llist-all2-lappendI llist-of-nellist-inverse
 nellist-of-llist-a-inverse)

lemma *nellist-all2-help-a*:

llist-all2 P (llist-of-nellist nella) (llist-of-nellist nellb) ==> nellist-all2 P nella nellb
by (coinduction arbitrary: nella nellb)
 (metis lhd-llist-of-nellist llist.disc(1) llist-all2-LCons-LCons llist-all2-LNil1
 llist-all2-LNil2 llist-all2-lhdD llist-all2-ltlI llist-of-nellist.disc-iff
 llist-of-nellist-eq-LNil ltl-llist-of-nellist ltl-simps(2))

lemma *nellist-all2-help-b*:

nellist-all2 P nella nellb ==> llist-all2 P (llist-of-nellist nella) (llist-of-nellist nellb)
proof (coinduction arbitrary: nella nellb)
case LNil
then show ?case
by simp
next
case LCons
then show ?case
by auto
 (metis llist-of-nellist.simps(2) nellist.case-eq-if nellist.rel-sel,
 metis llist-all2-LNil1 ltl-llist-of-nellist ltl-llist-of-nellist1 nellist.rel-sel)
qed

lemma *nellist-all2-transfer* [transfer-rule]:

((=) ==> pcr-nellist (=) ==> pcr-nellist (=) ==> (=)) 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 ⇒ 'a nellist ⇒ 'a nellist

is (λ llx lly. lappend llx lly)

by auto

lift-definition *nfilter* :: ('a ⇒ bool) ⇒ 'a nellist ⇒ 'a nellist

is λ P ll. (if lnull(lfilter P ll) then ll else lfilter P ll)

by auto

lift-definition *lappendn* :: 'a llist ⇒ 'a nellist ⇒ 'a nellist

is *lappend*
by *auto*

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

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

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

lift-definition *ndropn* :: nat \Rightarrow 'a nellist \Rightarrow 'a nellist
is λ n ll. ldropsn (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 λ n ll. ltake (eSuc n) ll
by *auto*

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

definition *nsubn* :: 'a nellist \Rightarrow nat \Rightarrow nat \Rightarrow 'a nellist
where *nsubn* nell i j = *ntaken* (j-i) (*ndropn* i nell)

definition *nfuse* :: 'a nellist \Rightarrow 'a nellist \Rightarrow 'a nellist
where *nfuse* nellx nelly = (if is-NNil nelly then nellx else nappend nellx (ntl nelly))

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 (=) ==> (=)*) (*λll. 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 ==> nlast (nmap f nell) = f (nlast nell)

by (*induct rule: nfinite-induct*)

(*auto simp add: nellist.map-sel(1)*)

lemma *nset-nlast*:

nfinite nell ==> nlast nell ∈ 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)) ∪ {nlast nell} = nset nell* (**is** *?lhs = ?rhs*)

proof(*intro set-eqI iffI*)

fix *x*

assume *x ∈ ?lhs*

thus *x ∈ ?rhs*

proof –

have 1: *nlast nell = x ==> x ∈ nset nell*

using *assms nset-nlast* **by** *blast*

have 2: *nlast nell = x ==> x ∈ nset nell*

unfolding *nlast0-nlast[symmetric]* **by** (*simp add: 1*)

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* **using** *ltl-llist-of-nellist1 nellist.set-sel(2)* **by** *force*

next

case (*step y*)

then show *?case* **by** (*metis in-nset-ntlD lbutlast.disc-iff(2) lbutlast-ltl ltl-llist-of-nellist*
ltl-llist-of-nellist1)

qed

show *?thesis*

using 2 3 *x ∈ lset (lbutlast (llist-of-nellist nell)) ∪ {nlast nell}* **by** *blast*

qed

```

next
  fix x
  assume  $x \in ?rhs$ 
  thus  $x \in ?lhs$ 
  proof(induct rule: nellist-set-induct)
    case (findnil nell)
    then show ?case
      by (cases nell) auto
    next
    case (find nell)
    thus ?case
      by (metis UnI1 lbutlast.disc-iff(2) llist.discI(1) 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-ntlD lbutlast-ntl ltl-llist-of-nellist nlast-ntl)
  qed
qed

lemma lset-llist-of-nellist-2:
  assumes  $\neg nfinite\ nell$ 
  shows  $lset\ (lbutlast\ (llist-of-nellist\ nell)) = nset\ nell$  (is  $?lhs = ?rhs$ )
  proof(intro set-eqI iffI)
    fix x
    assume  $x \in ?lhs$ 
    thus  $x \in ?rhs$ 
    proof -
      have  $\exists x \in lset\ (lbutlast\ (llist-of-nellist\ nell)) \implies x \in nset\ nell$ 
      proof (induct lbutlast (llist-of-nellist nell) arbitrary: nell rule: llist-set-induct)
        case find
        then show ?case using ltl-llist-of-nellist1 nellist.set-sel(2) by force
      next
        case (step y)
        then show ?case by (metis in-nset-ntlD lbutlast.disc-iff(2) lbutlast-ntl ltl-llist-of-nellist
          ltl-llist-of-nellist1)
      qed
    show ?thesis
      using  $\exists x \in lset\ (lbutlast\ (llist-of-nellist\ nell))$  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.disc-iff(2) lbutlast.simps(3) lhd-llist-of-nellist llist.disc(1)
    llist.set-sel(1) llist-of-nellist-not-lnull ltl-llist-of-nellist)
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 (=) ==> (=)) lset nset
by(auto simp add: cr-nellist-def nellist.pcr-cr-eq)

```

2.6 *nmap*

```

lemma nmap-eq-NCons-conv:
  nmap f nellx = NCons y nelly  $\longleftrightarrow$ 

```

$(\exists z \text{ nellz}. \text{nellx} = \text{NCons } z \text{ nellz} \wedge f z = y \wedge \text{nmap } f \text{ nellz} = \text{nelly})$
by *(cases nellx) simp-all*

lemma *NCons-eq-nmap-conv*:

$\text{NCons } y \text{ nelly} = \text{nmap } f \text{ nellx} \longleftrightarrow$
 $(\exists z \text{ nellz}. \text{nellx} = \text{NCons } z \text{ nellz} \wedge f z = y \wedge \text{nmap } f \text{ nellz} = \text{nelly})$
by *(cases nellx) auto*

2.7 Appending two nonempty lazy lists *nappend*

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

$\text{nappend } (\text{NNil } b) \text{ nell} = (\text{NCons } b \text{ nell})$
by *transfer auto*

lemma *nappend-NCons* [*simp, code, nitpick-simp*]:

$\text{nappend } (\text{NCons } a \text{ nellx}) \text{ nelly} = \text{NCons } a (\text{nappend } \text{nellx } \text{nelly})$
by *transfer auto*

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

$\text{nhd}(\text{nappend } \text{nellx } \text{nelly}) = (\text{if is-NNil } \text{nellx} \text{ then } \text{nlast } \text{nellx} \text{ else } \text{nhd } \text{nellx})$
by *(cases nellx) auto*

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

$\text{ntl}(\text{nappend } \text{nellx } \text{nelly}) = (\text{if is-NNil } \text{nellx} \text{ then } \text{nelly} \text{ else } \text{nappend } (\text{ntl } \text{nellx}) \text{ nelly})$
by *(cases nellx) auto*

lemma *is-NNil-nappend*:

$\text{is-NNil}(\text{nappend } \text{nellx } \text{nelly}) \longleftrightarrow \text{False}$
by *(cases nellx) auto*

lemma *nappend-assoc*:

$\text{nappend } (\text{nappend } \text{nellx } \text{nelly}) \text{ nellz} = \text{nappend } \text{nellx} (\text{nappend } \text{nelly } \text{nellz})$
by *transfer (auto simp add: split-beta lappend-assoc)*

lemma *nmap-nappend-distrib*:

$\text{nmap } f (\text{nappend } \text{nellx } \text{nelly}) = \text{nappend } (\text{nmap } f \text{ nellx}) (\text{nmap } f \text{ nelly})$
by *transfer (auto simp add: split-beta lmap-lappend-distrib)*

lemma *nlast-nappend*:

$\text{nlast } (\text{nappend } \text{nellx } \text{nelly}) = (\text{if } \text{nfinite } \text{nellx} \text{ then } \text{nlast } \text{nelly} \text{ else } \text{nlast } \text{nellx})$
by *transfer (auto simp add: llast-lappend)*

lemma *nfinite-nappend*:

$\text{nfinite } (\text{nappend } \text{nellx } \text{nelly}) \longleftrightarrow \text{nfinite } \text{nellx} \wedge \text{nfinite } \text{nelly}$
by *transfer auto*

lemma *nappend-inf*:

$\neg \text{nfinite } \text{nellx} \implies \text{nappend } \text{nellx } \text{nelly} = \text{nellx}$
by *transfer (auto simp add: lappend-inf)*

lemma *nappend-snocn-inf*:
assumes $\neg \text{nfinite nell}$
shows $\text{nappend nell (NNil a)} = \text{snocn nell a}$
using *assms*
by (*simp add: nappend-inf snocn-inf*)

lemma *nappend-snocn-finite*:
assumes nfinite nell
shows $\text{nappend nell (NNil a)} = \text{snocn nell a}$
using *assms*
by (*induct rule: nfinite-induct simp-all*)

lemma *nappend-snocn*:
 $\text{nappend nell (NNil a)} = \text{snocn nell a}$
by (*meson nappend-snocn-finite nappend-snocn-inf*)

lemma *split-nellist-first*:
assumes $x \in \text{nset nell}$
shows $\text{nell} = (\text{NNil } x) \vee (\exists \text{ ys. nell} = \text{nappend (NNil } x) \text{ ys}) \vee$
 $(\exists \text{ ys. nell} = (\text{nappend ys (NNil } x)) \wedge \text{nfinite ys} \wedge x \notin \text{nset ys}) \vee$
 $(\exists \text{ ys zs. nell} = (\text{nappend ys (NCons } x \text{ zs})) \wedge \text{nfinite ys} \wedge x \notin \text{nset ys})$
using *assms*
by *transfer*
(auto,
metis eq-LConsD lhd-lappend llist.disc(1) llist.expand split-llist-first)

lemma *split-nellist*:
assumes $x \in \text{nset nell}$
shows $\text{nell} = (\text{NNil } x) \vee (\exists \text{ ys. nell} = \text{nappend (NNil } x) \text{ ys}) \vee$
 $(\exists \text{ ys zs. nell} = \text{nappend ys (NCons } x \text{ zs}) \wedge \text{nfinite ys}) \vee$
 $(\exists \text{ ys. nell} = \text{nappend ys (NNil } x) \wedge \text{nfinite ys})$
using *assms*
by (*meson split-nellist-first*)

lemma *split-nellist-a*:
assumes $\text{nell} = (\text{NNil } x) \vee (\exists \text{ ys. nell} = \text{nappend (NNil } x) \text{ ys}) \vee$
 $(\exists \text{ ys zs. nell} = \text{nappend ys (NCons } x \text{ zs}) \wedge \text{nfinite ys}) \vee$
 $(\exists \text{ ys. nell} = \text{nappend ys (NNil } x) \wedge \text{nfinite ys})$
shows $x \in \text{nset nell}$
proof –
have 1: $\text{nell} = (\text{NNil } x) \implies x \in \text{nset nell}$
by *simp*
have 2: $(\exists \text{ ys. nell} = \text{nappend (NNil } x) \text{ ys}) \implies x \in \text{nset nell}$
by *auto*
have 3: $(\exists \text{ ys zs. nell} = \text{nappend ys (NCons } x \text{ zs}) \wedge \text{nfinite ys}) \implies x \in \text{nset nell}$
by *transfer auto*
have 4: $(\exists \text{ ys. nell} = \text{nappend ys (NNil } x) \wedge \text{nfinite ys}) \implies x \in \text{nset nell}$
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 $\exists n. \text{nlenght } nell = \text{enat } n$

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 $\longleftrightarrow (\exists n. \text{nlenght } nell = \text{enat } n)$

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

end

2.9 *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) ts $\longleftrightarrow (\exists b'. ts = NNil b' \wedge Q b b')$

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

lemma *nellist-all2-NNil2*:

nellist-all2 Q ts (NNil b') $\longleftrightarrow (\exists b. ts = NNil b \wedge Q b b')$

using *nellist.rel-sel*
by (*metis is-NNil-def nellist-all2-NNil*)

lemma *nellist-all2-NCons1*:

nellist-all2 P (NCons x ts) ts' \longleftrightarrow
($\exists x' ts''. ts' = NCons x' ts'' \wedge P x x' \wedge nellist-all2 P ts ts''$)

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 ts' (NCons x ts) \longleftrightarrow
($\exists x' ts''. ts' = NCons x' ts'' \wedge P x' x \wedge nellist-all2 P ts'' ts$)

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

lemma *not-lnull-llength*:

$\neg lnull\ ll \longleftrightarrow 1 \leq llength\ ll$

by (*metis gr-zeroI ileI1 llength-eq-0 not-one-le-zero one-eSuc*)

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 $\bigwedge_{nellix\ nellyy}.$

X nellix nellyy \implies

(is-NNil nellix = is-NNil nellyy) \wedge

(is-NNil nellix \longrightarrow is-NNil nellyy $\longrightarrow P$ (nlast nellix) (nlast nellyy)) \wedge

($\neg is-NNil\ nellix \longrightarrow \neg is-NNil\ nellyy \longrightarrow P$ (nhd nellix) (nhd nellyy)) \wedge

(X (ntl nellix) (ntl nellyy) \vee nellist-all2 P (ntl nellix) (ntl nellyy)))

shows *nellist-all2 P nellx nelly*

using *assms*

nellist.rel-coinduct[of ($\lambda\ nellx\ nelly. X\ nellx\ nelly \vee nellist-all2\ P\ nellx\ nelly$) 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 (*NNil*) *b b' where nellx = NNil b nelly = NNil b' P b b'*

| (NCons) x nellx' y nelly'

where *nellx = NCons x nellx' and nelly = NCons y nelly'*

and *P x y and nellist-all2 P nellx' nelly'*

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*:

$\text{nellist-all2 } P \text{ nellx nelly} \implies \text{nlength nellx} = \text{nlength nelly}$

by(*transfer*)(*auto dest: llist-all2-llengthD*)

lemma *nellist-all2-nfiniteD*:

$\text{nellist-all2 } P \text{ nellx nelly} \implies \text{nfinite nellx} = \text{nfinite nelly}$

by *transfer*

(*auto dest: llist-all2-lfiniteD*)

lemma *nellist-all2-nfinite1-nlastD*:

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

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*:

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

by(*metis nellist-all2-nfinite1-nlastD nellist-all2-nfiniteD*)

lemma *nellist-all2D-llist-all2-llist-of-nellist*:

$\text{nellist-all2 } P \text{ nellx nelly} \implies \text{llist-all2 } P \ (\text{llist-of-nellist nellx}) \ (\text{llist-of-nellist nelly})$

by *transfer*

(*simp add: nellist-all2-help-b*)

lemma *nellist-all2-is-NNilD*:

$\text{nellist-all2 } P \text{ nellx nelly} \implies \text{is-NNil nellx} \longleftrightarrow \text{is-NNil nelly}$

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

lemma *nellist-all2-nhdD*:

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

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

lemma *nellist-all2-ntlI*:

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

$\text{nellist-all2 } P \ (\text{ntl nellx}) \ (\text{ntl nelly})$

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

lemma *nellist-all2-refl*:

$\text{nellist-all2 } P \text{ nell nell} \longleftrightarrow$

$(\forall x \in \text{nset nell}. P \ x \ x) \wedge (\text{nfinite nell} \longrightarrow P \ (\text{nlast nell}) \ (\text{nlast nell}))$

by *transfer*

(*auto, metis in-lset-lappend-iff lappend-lbutlast-llast-id-lfinite lfinite-lappend*

llist.set-intros(1))

lemma *nellist-all2-refl*:

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

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

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

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

proof –

assume *a1*: *lfinite nelly*

assume *a2*: *llist-all2 P nellx nelly*

assume *a3*: $\neg \text{lnull } \text{nellx}$

assume *a4*: $\neg \text{lnull } \text{nelly}$

have *f5*: *lfinite (lappend (lbutlast nellx) (LCons (llast nellx) LNil))*

using *a3 a2 a1* **by** (*simp add: llist-all2-lfiniteD*)

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

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

have *f7*: *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 llist.disc(2) llist-all2-lappend1D(2)
llist-all2-lhdD2)

qed

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

nellist-all2 P nellx nelly \longleftrightarrow

nlength nellx = nlength nelly \wedge

$(\forall n. \text{enat } n \leq \text{nlength } \text{nellx} \longrightarrow P \ (\text{nnth } \text{nellx } n) \ (\text{nnth } \text{nelly } n))$

by *transfer*

(*auto simp add: llist-all2-llengthD,*

metis eSuc-epred iless-Suc-eq llength-eq-0 llist-all2-lnthD2 min.orderE the-enat.simps,

metis eSuc-epred iless-Suc-eq llength-eq-0 llist-all2-conv-all-lnth min.orderE the-enat.simps)

lemma *nellist-all2-nnthD*:

$\llbracket \text{nellist-all2 } P \ \text{nellx} \ \text{nelly}; \text{enat } n \leq \text{nlength } \text{nellx} \rrbracket \implies P \ (\text{nnth } \text{nellx } n) \ (\text{nnth } \text{nelly } n)$

by (*simp add: nellist-all2-conv-all-nnth*)

lemma *nellist-all2-nnthD2*:

$\llbracket \text{nellist-all2 } P \ \text{nellx} \ \text{nelly}; \text{enat } n \leq \text{nlength } \text{nelly} \rrbracket \implies P \ (\text{nnth } \text{nellx } n) \ (\text{nnth } \text{nelly } n)$

by (*simp add: nellist-all2-conv-all-nnth*)

lemmas *nellist-all2-eq = nellist.rel-eq*

lemma *nmap-eq-nmap-conv-nellist-all2*:

nmap f nellx = nmap f' nelly \longleftrightarrow

$nellist-all2 (\lambda x y. f x = f' y) nellx nelly$
by *transfer*
 $(clarsimp simp add: lmap-eq-lmap-conv-llist-all2)$

lemma *nellist-all2-trans*:

$\llbracket nellist-all2 P nellx nelly; nellist-all2 P nelly nellz; transp P \rrbracket$
 $\implies nellist-all2 P nellx nellz$

by *transfer (auto elim: llist-all2-trans dest: llist-all2-lfiniteD transpD)*

lemma *nellist-all2-nappendI*:

$\llbracket nellist-all2 P nellx nelly;$
 $\llbracket nfinite nellx; nfinite nelly; P (nlast nellx) (nlast nelly) \rrbracket$
 $\implies nellist-all2 P nellx' nelly' \rrbracket$
 $\implies nellist-all2 P (nappend nellx nellx') (nappend nelly nelly')$

by *transfer*

$(auto simp add: lnull-def lappend-eq-lappend-conv nellist-all2-conv-all-nnth-help1$
 $intro: llist-all2-lappendI)$

lemma *llist-all2-nellist-of-llistI*:

$nellist-all2 A nellx nelly \implies$
 $llist-all2 A (lbutlast(llist-of-nellist nellx)) (lbutlast(llist-of-nellist nelly))$

proof *(coinduction arbitrary: nellx nelly)*

case *LNil*

then show *?case*

by $(metis lbutlast.disc-iff(1) llist-of-nellist-inverse-a ltl-llist-of-nellist1$
 $nellist-all2-is-NNilD nellist-of-llist-a.disc(1))$

next

case *LCons*

then show *?case*

by $(metis lbutlast.disc(1) lbutlast.simps(3) lbutlast-rtl lhd-llist-of-nellist rtl-llist-of-nellist$
 $ltl-llist-of-nellist1 nellist-all2-nhdD nellist-all2-ntlI)$

qed

lemma *nellist-all2-nellist-of-llist-a [simp]*:

$nellist-all2 A (nellist-of-llist-a b llx) (nellist-of-llist-a c lly) \longleftrightarrow$
 $llist-all2 A llx lly \wedge (lfinite llx \longrightarrow A b c)$

proof *(cases lfinite llx)*

case *True*

then show *?thesis*

using *llist-all2-nellist-of-llistI*

by $(auto simp add: llist-all2-lappendI nellist-all2-help-a, fastforce,$
 $metis lbutlast-lfinite lbutlast-snoc llist-all2-lfiniteD nellist-all2-nfinite1-nlastD$
 $nellist-of-llist-a-inverse nfinite-def nlast-nellist-of-llist-a-lfinite)$

next

case *False*

then show *?thesis*

by $(metis lbutlast-snoc llist-all2-lappendI llist-all2-nellist-of-llistI nellist-all2-help-a$
 $nellist-of-llist-a-inverse)$

qed

2.10 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-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

2.11 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*)

2.12 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 | 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 \Rightarrow nnth (nmap f nell) n = f (nnth nell n)

by transfer

(metis co-enat.exhaust-sel iless-Suc-eq llength-eq-0 llength-lmap lnth-lmap min.orderE
the-enat.simps)

lemma *nhd-conv-nnth*:

\neg is-NNil nell \Rightarrow 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*:

$x \in \text{nset } nell \iff (\exists n. \text{enat } n \leq \text{nlength } nell \wedge \text{nnth } nell \ n = x)$

by transfer

(metis eSuc-epred iless-Suc-eq in-lset-conv-lnth llength-eq-0 min-absorb1 the-enat.simps)

lemma *nnth-beyond*:

nlength nell \leq enat n \Rightarrow

nnth nell n = nnth nell (the-enat (nlength nell))

by transfer

(metis enat.distinct(2) enat-the-enat min.idem min-absorb2 min-enat-simps(4))

lemma *exists-nnth-nset*:

$(\exists x \in \text{nset } nell. P \ x) = (\exists n. P \ (\text{nnth } nell \ n))$

by (auto,

metis in-nset-conv-nnth,
 metis enat-ord-code(4) enat-the-enat in-nset-conv-nnth min.strict-order-iff min-def neq-iff
 nnth-beyond order-refl)

lemma nset-conv-nnth:

nset nell = {nnth nell n | n. enat n ≤ nlength nell}

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

lemma nnth-nappend1:

enat n ≤ nlength nellx ⇒ 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.premis Suc-ile-eq by auto

qed

qed

lemma nnth-nappend2:

⌊nlength nellx = enat k; k < n⌋ ⇒ nnth (nappend nellx nelly) n = nnth nelly (n − 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:

nnth (nappend nellx nelly) n =

(if enat n ≤ nlength nellx then nnth nellx n else nnth nelly (n − Suc(the-enat(nlength nellx))))

by (cases nlength nellx)

(auto simp add: nnth-nappend1 nnth-nappend2)

lemma nnth-nlast:

nfinite nell ⇒ nlast nell = nnth nell (the-enat (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)*

2.13 ndropn

lemma *ndropn-0* [*simp*, *code*, *nitpick-simp*]:

ndropn 0 nell = nell

using *zero-enat-def* **by** *transfer auto*

lemma *ndropn-NNil* [*simp*, *code*]:

ndropn n (NNil b) = (NNil b)

by *transfer auto*

lemma *ndropn-Suc-NCons* [*simp*, *code*]:

ndropn (Suc n) (NCons x nell) = ndropn n nell

proof (*cases nfinite nell*)

case *True*

then show *?thesis*

by *transfer*

(*auto simp add: min-def not-lnull-conv Suc-ile-eq llength-eq-infty-conv-lfinite the-enat-eSuc ,
metis Extended-Nat.eSuc-mono eSuc-enat iless-Suc-eq leD,
metis antisym eSuc-enat enat-the-enat ile-eSuc llength-eq-infty-conv-lfinite n-not-Suc-n
the-enat.simps*)

next

case *False*

then show *?thesis*

by *transfer*

(*simp,
metis co.enat.sel(2) eSuc-infinity infinity-ileE ldropn-Suc-LCons llength-eq-infty-conv-lfinite
min.cobounded1 min-def the-enat.simps*)

qed

lemma *ndropn-Suc* [*nitpick-simp*]:

ndropn (Suc n) nell = (case nell of NNil b \Rightarrow NNil b | NCons x nell' \Rightarrow ndropn n nell')

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

lemma *ltl-power-NNil-help*:

($((\lambda ll. \text{if } \exists b. ll = LCons\ b\ LNil \text{ then } ll \text{ else } ltl\ ll) \rightsquigarrow n) (LCons\ b\ LNil) = LCons\ b\ LNil$)

by (*induction n*) *simp-all*

lemma *ntl-power-NNil*:

(*ntl* \rightsquigarrow *n*) (NNil *b*) = (NNil *b*)

by *transfer* (*auto simp add: lnull-def ltl-power-NNil-help*)

lemma *ntl-power-NCons*:

*ntl ((ntl \rightsquigarrow *n*) (NCons *x21* *x22*)) = (ntl \rightsquigarrow *n*) *x22**

by (*induction n*) (*transfer*, *auto*)

lemma *ntl-power-Suc* [*simp*]:

(*ntl* \rightsquigarrow (*Suc n*)) *nell* = (*case nell of NNil b \Rightarrow NNil b | NCons x nell' \Rightarrow (ntl \rightsquigarrow *n*) nell'*)

by (*cases nell*)
 (*simp-all add: ntl-power-NNil ntl-power-NCons*)

lemma *llist-of-nellist-ndropn* [*simp*]:
llist-of-nellist (*ndropn n nell*) =
ldropn (*the-enat* (*min* (*enat n*) ((*epred*(*llength*((*llist-of-nellist nell*)))))))
 (*llist-of-nellist nell*)
by *transfer auto*

lemma *ndropn-Suc-conv-ndropn*:
enat n < nlength nell \implies NCons (nnth nell n) (ndropn (Suc n) nell) = ndropn n nell
proof (*induct n arbitrary: nell*)
case 0
then show ?*case*
proof (*cases nell*)
case (*NNil x1*)
then show ?*thesis* **using** 0.*prems* **by** *auto*
next
case (*NCons x21 x22*)
then show ?*thesis* **by** *simp*
qed
next
case (*Suc n*)
then show ?*case*
proof (*cases nell*)
case (*NNil x1*)
then show ?*thesis* **using** *Suc.prems* **by** *auto*
next
case (*NCons x21 x22*)
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 x21 x22*)
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 x21 x22)
  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-implyes-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 ~ 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 x21 x22)
  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 x21 x22)
  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 x21 x22)
  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.commute enat-ord-simps(1) le-add1 ndropn-nnth nfinite-conv-nlength-enat nnth-NNil
 nnth-beyond)

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-def is-NNil-ndropn ndropn-nnth nnth-NNil nnth-beyond)

have 4: $\neg nfinite\ nell \implies$

$ndropn\ n\ nell = (NNil\ b) \longleftrightarrow$

$nnth\ nell\ (the-enat(nlength\ nell)) = b \wedge nlength\ nell \leq (enat\ n)$

by (metis enat-ile is-NNil-ndropn nellist.disc(1) nfinite-conv-nlength-enat)

show ?thesis

using 1 2 3 4 **by** fastforce

qed

lemma ntl-ndropn:

$ntl(ndropn\ n\ nell) = ndropn\ n\ (ntl\ nell)$

by (simp add: funpow-swap1 ndropn-ntl)

lemma nfinite-ndropn-a:

assumes nfinite nell

shows nfinite(ndropn n nell)

using assms

proof (induct n arbitrary: nell)

case 0

then show ?case **by** auto

next

case (Suc n)

then show ?case **by** (simp add: ndropn-ntl)

qed

lemma nfinite-ndropn-b:

assumes nfinite(ndropn n nell)

shows nfinite nell

using assms

proof (induct ys \equiv ndropn n nell arbitrary: n nell rule: nfinite-induct)

case (NNil y)

then show ?case **by** (metis enat-ile ndropn-eq-NNil nfinite-conv-nlength-enat)

next

case (NCons x nell)

then show ?case **by** (metis nellist.sel(5) nfinite-ntl ntl-ndropn)
qed

lemma nfinite-ndropn[simp]:
 nfinite(ndropn n nell) = nfinite nell
using nfinite-ndropn-a nfinite-ndropn-b **by** blast

lemma ndropn-ndropn:
 ndropn m (ndropn n nell) = ndropn (n+m) nell
proof (induct n arbitrary: nell)
case 0
then show ?case **by** simp
next
case (Suc n)
then show ?case
by (metis add-Suc ndropn-NNil ndropn-Suc nellist.case-eq-if)
qed

lemma ndropn-nlast:
 nfinite nell \implies ndropn (the-enat(nlength nell)) nell = (NNil (nlast nell))
by (metis add.left-neutral enat.simps(3) enat-the-enat ndropn-eq-NNil ndropn-nnth ndropn-nnth-a
 nfinite-conv-nlength-enat order-refl)

lemma ndropn-nfirst:
 nfirst (ndropn n nell) = (nnth nell n)
by transfer
 (metis lhd-ldropn llist-of-nellist-ndropn lnull-ldropn not-le-imp-less
 not-lnull-conv-llist-of-nellist)

lemma ndropn-all:
 nlength nell \leq enat n \implies ndropn n nell = (NNil (nlast nell))
by (metis enat-ile ndropn-eq-NNil ndropn-nlast nlength-eq-enat-nfiniteD)

lemma ndropn-nappend1:
 nfinite nellx \implies n \leq nlength nellx \implies
 ndropn (Suc(the-enat (nlength nellx) + n)) (nappend nellx nelly) = ndropn n nelly
proof (induct arbitrary: nelly n rule: nfinite-induct)
case (NNil y)
then show ?case **by** simp
next
case (NCons x nell)
then show ?case
by (metis ab-semigroup-add-class.add-ac(1) eSuc-enat nappend-NCons ndropn-Suc-NCons
 nfinite-nlength-enat nlength-NCons plus-1-eq-Suc the-enat.simps)
qed

lemma ndropn-nappend2:
 enat n \leq (nlength nellx) \implies ndropn n (nappend nellx nelly) = nappend (ndropn n nellx) nelly
by transfer
 (auto,

metis add-eq-0-iff-both-eq-0 eSuc-epred enat-the-enat iless-Suc-eq infinity-ileE leD
 llength-eq-0 min.cobounded1 min.cobounded2,
 metis co.enat.collapse enat-le-plus-same(1) epred-iadd1 iless-Suc-eq ldropn-lappend
 llength-eq-0 min-absorb1 order-trans the-enat.simps)

lemma ndropn-nappend3:

$nlength\ nellx < enat\ n \implies$

$ndropn\ n\ (nappend\ nellx\ nelly) = ndropn\ (n - (the-enat\ (eSuc(nlength\ nellx))))\)\ nelly$

proof (transfer, auto)

fix nellxa :: 'b llist **and** na :: nat **and** nellya :: 'c llist

assume a1: $\neg lnull\ nellxa$

assume a2: $llength\ nellxa + llength\ nellya \leq$

$enat\ (the-enat\ (min\ (enat\ na)\ (epred\ (llength\ nellxa + llength\ nellya))))$

have $llength\ nellxa \neq 0$

using a1 llength-eq-0 **by** blast

then show False

using a2 **by** (metis (no-types) co.enat.exhaust-sel enat-le-plus-same(1) ile0-eq iless-Suc-eq
 leD min.cobounded2 min-enat1-conv-enat the-enat.simps)

next

show $\bigwedge nellx\ n\ nelly.$

$\neg lnull\ nellx \implies$

$epred\ (llength\ nellx) < enat\ n \implies$

$\neg lnull\ nelly \implies$

$ldropn\ (the-enat\ (min\ (enat\ n)\ (epred\ (llength\ nellx + llength\ nelly))))\ (lappend\ nellx\ nelly) =$
 $ldropn\ (the-enat\ (min\ (enat\ (n - the-enat\ (llength\ nellx))\ (epred\ (llength\ nelly))))\ nelly$

proof (auto simp add: min-def)

show $\bigwedge nellx\ n\ nelly.$

$\neg lnull\ nellx \implies$

$epred\ (llength\ nellx) < enat\ n \implies$

$\neg lnull\ nelly \implies$

$enat\ n \leq epred\ (llength\ nellx + llength\ nelly) \implies$

$enat\ (n - the-enat\ (llength\ nellx)) \leq epred\ (llength\ nelly) \implies$

$ldropn\ n\ (lappend\ nellx\ nelly) = ldropn\ (n - the-enat\ (llength\ nellx))\ nelly$

by (metis eSuc-epred iless-Suc-eq ldropn-lappend leD llength-eq-0)

next

fix nellxa :: 'c llist **and** na :: nat **and** nellya :: 'c llist

assume a1: $\neg lnull\ nellxa$

assume a2: $epred\ (llength\ nellxa) < enat\ na$

assume a3: $\neg lnull\ nellya$

assume a4: $enat\ na \leq epred\ (llength\ nellxa + llength\ nellya)$

assume $\neg enat\ (na - the-enat\ (llength\ nellxa)) \leq epred\ (llength\ nellya)$

then have $lnull\ (ldropn\ na\ (lappend\ nellxa\ nellya))$

using a3 a2 a1

by (metis eSuc-epred iless-Suc-eq ldropn-lappend leD llength-eq-0 lnull-ldropn not-le-imp-less)

then have False

using a4 a1

by (metis eSuc-epred iless-Suc-eq leD llength-eq-0 llength-lappend lnull-lappend lnull-ldropn)

then show $ldropn\ na\ (lappend\ nellxa\ nellya) = ldropn\ (the-enat\ (epred\ (llength\ nellya)))\ nellya$

by meson

next

```

fix nellxa :: 'c llist and na :: nat and nellya :: 'c llist
assume a1: ¬ lnull nellxa
assume a2: epred (llength nellxa) < enat na
assume a3: ¬ lnull nellya
assume a4: ¬ enat na ≤ epred (llength nellxa + llength nellya)
assume enat (na - the-enat (llength nellxa)) ≤ epred (llength nellya)
then have ¬ lnull (ldropn na (lappend nellxa nellya))
using a3 a2 a1
by (metis co.enat.exhaust-sel iless-Suc-eq ldrops-lappend leD llength-eq-0 lnull-ldrops)
then have False
using a4 a1
by (metis (no-types) co.enat.exhaust-sel iless-Suc-eq llength-eq-0 llength-lappend
    lnull-lappend lnull-ldrops not-le-imp-less)
then show ldrops (the-enat (epred (llength nellxa + llength nellya))) (lappend nellxa nellya) =
    ldrops (na - the-enat (llength nellxa)) nellya
by meson
next
fix nellxa :: 'c llist and na :: nat and nellya :: 'c llist
assume a1: ¬ lnull nellxa
assume a2: ¬ enat (na - the-enat (llength nellxa)) ≤ epred (llength nellya)
assume a3: ¬ lnull nellya
assume a5: ¬ enat na ≤ epred (llength nellxa + llength nellya)
assume a4: epred (llength nellxa) < enat na
have f0: llength nellxa ≤ enat ((the-enat (epred (llength nellxa + llength nellya))))
  by (metis a3 a5 add commute enat-ile epred-iadd1 le-cases le-iff-add llength-eq-0 the-enat.simps)
have f1: the-enat (llength nellxa) ≤ enat ((the-enat (epred (llength nellxa + llength nellya))))
  by (metis (no-types) enat-ord-code(4) enat-the-enat f0 leD)
have f2: 0 < llength nellxa
  by (simp add: a1)
have f3: 0 < llength nellya
  by (simp add: a3)
have f5: enat ((the-enat (epred (llength nellxa + llength nellya)))) -
    the-enat (llength nellxa) = (the-enat (epred (llength nellya)))
proof simp
  have 1: the-enat (epred (llength nellxa + llength nellya)) =
    the-enat (epred (llength nellya + llength nellxa))
  by (simp add: add commute)
  have 2: the-enat (epred (llength nellya + llength nellxa)) =
    (the-enat (epred (llength nellya))) + the-enat (llength nellxa)
  by (metis a2 enat-ile epred-iadd1 f0 f3 gr-implies-not-zero min-def
    min-enat1-conv-enat plus-enat-simps(1) the-enat.simps)
  show the-enat (epred (llength nellxa + llength nellya)) - the-enat (llength nellxa) =
    the-enat (epred (llength nellya))
  by (simp add: 1 2)
qed
show ldrops (the-enat (epred (llength nellxa + llength nellya))) (lappend nellxa nellya) =
  ldrops (the-enat (epred (llength nellya))) nellya
  by (metis f0 f5 idiff-enat-enat ldrops-lappend2 the-enat.simps)
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)

2.14 *ntake*

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

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

by *transfer* *auto*

lemma *ntake-0* [*simp*]:

$ntake\ 0\ nell = (NNil\ (nfirst\ nell))$

by *transfer* (*auto* *simp* *add*: *ltake.ctr*(2))

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

$ntake\ (eSuc\ n)\ (NCons\ x\ nell) = (NCons\ x\ (ntake\ n\ nell))$

by *transfer* *auto*

lemma *ntake-Suc*:

$ntake\ (eSuc\ n)\ nell =$

$(case\ nell\ of\ (NNil\ b) \Rightarrow (NNil\ b) \mid (NCons\ x\ nell') \Rightarrow (NCons\ x\ (ntake\ n\ nell')))$

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* *x21* *x22*)

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 x21 x22)
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]:
 $ntake\ n\ (NCons\ x\ nell) =$
 $(case\ n\ of\ 0 \Rightarrow (NNil\ x) \mid$
 $(eSuc\ n') \Rightarrow (NCons\ x\ (ntake\ n'\ nell)))$
by (simp add: co.enat.case-eq-if)
 $(metis\ eSuc-epred\ nellist.simps(6)\ nfirst-def\ ntake-Suc-NCons)$

lemma nhd-ntake [simp]:
 $n \neq 0 \implies nhd(ntake\ n\ nell) = nhd\ nell$
unfolding nhd-def
by (simp add: nellist.case-eq-if)
 $(metis\ (no-types,\ lifting)\ co.enat.case-eq-if\ nellist.collapse(2)\ nellist.sel(3)\ ntake-NCons)$

lemma ntl-ntake:
 $n \neq 0 \implies ntl(ntake\ n\ nell) = ntake\ (epred\ n)\ (ntl\ nell)$
by (cases nell) (simp, metis eSuc-epred nellist.sel(5) ntake-Suc-NCons)

lemma ntl-ntake-0:
 $ntl(ntake\ 0\ nell) = (NNil\ (nfirst\ nell))$
by simp

lemma ntake-ntl:
 $ntake\ n\ (ntl\ nell) = ntl(ntake\ (Suc\ n)\ nell)$
by (simp add: enat-0-iff(1) ntl-ntake)

lemma nlength-ntake [simp]:
 $nlength\ (ntake\ n\ nell) = min\ n\ (nlength\ nell)$

by *transfer simp*

lemma *ntake-nmap [simp]*:

$ntake\ n\ (nmap\ f\ nell) = nmap\ f\ (ntake\ n\ nell)$

by *transfer simp*

lemma *ntake-ntake [simp]*:

$ntake\ n\ (ntake\ m\ nell) = ntake\ (\min\ n\ m)\ nell$

by *transfer simp*

lemma *nset-ntake*:

$nset\ (ntake\ n\ nell) \subseteq nset\ nell$

by *transfer (simp add: lset-ltake)*

lemma *ntake-all*:

$nlength\ nell \leq m \implies ntake\ m\ nell = nell$

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

lemma *nfinite-ntake [simp]*:

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

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

lemma *ntake-nappend1*:

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

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

lemma *ntake-nappend2*:

assumes $nlength\ nellx < n$

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

proof (*cases nellx*)

case (*NNil x1*)

then show *?thesis using assms*

by (*metis eSuc-le-iff eSuc-minus-1 idiff-0-right ileI1 nappend-NNil nlength-NNil ntake-Suc-NCons*)

next

case (*NCons x21 x22*)

then show *?thesis*

proof (*cases nfinite nellx*)

case *True*

then show *?thesis*

using *assms*

proof (*transfer*)

fix *nellxa :: 'a llist*

fix *na*

fix *nellya :: 'a llist*

assume *a0*: $\neg lnull\ nellxa \wedge nellxa = nellxa$

assume *a1*: *lfinite nellxa*

assume *a2*: *epred (llength nellxa) < na*

assume *a3*: $\neg lnull\ nellya \wedge nellya = nellya$

show $\neg lnull\ (ltake\ (eSuc\ na)\ (lappend\ nellxa\ nellya)) \wedge$

$ltake\ (eSuc\ na)\ (lappend\ nellxa\ nellya) =$


```

    lappend nellx (ltake (eSuc (na - epred (llength nellx) - 1)) nellya)
  proof -
    have 1:  $\neg$  lnull (ltake (eSuc na) (lappend nellx nellya))
      using a0 by force
    have 2: ltake (eSuc na) (lappend nellx nellya) =
      lappend (ltake (eSuc na) nellx) (ltake ((eSuc na) - llength nellx) nellya)
      by (meson ltake-lappend)
    have 21: llength nellx  $\leq$  (eSuc na)
      by (metis a0 a2 co.enat.exhaust-sel eSuc-ile-mono leD le-cases llength-eq-0)
    have 3: lappend (ltake (eSuc na) nellx) (ltake ((eSuc na) - llength nellx) nellya) =
      lappend nellx (ltake (eSuc na - llength nellx) nellya)
      using ltake-lappend2[of nellx (eSuc na) nellya] 21 2 by auto
    have 4: (eSuc na - llength nellx) = (eSuc (na - epred (llength nellx) - 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-infnty-conv-lfinite)
    have 5: (ltake (eSuc na - llength nellx) nellya) =
      (ltake (eSuc (na - epred (llength nellx) - 1)) nellya)
      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 enat m < n
  shows (nnth (ntake n nell) m) = (nnth nell m)
  using assms
  proof (induct m arbitrary: nell n)
    case 0
    then show ?case
    proof (cases n rule: enat-coexhaust)
      case 0
      then show ?thesis
      using 0.prem by auto
    next
      case (eSuc n')
      then show ?thesis
      by (metis 0.prem min.strict-order-iff ndropn-0 ndropn-nfirst nlast-NNil ntake-0 ntake-ntake
        zero-enat-def)
    qed
  next

```

```

case (Suc m)
then show ?case
  proof (cases n rule: enat-coexhaust)
    case 0
    then show ?thesis
    using Suc.prem by auto
    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

```

```

lemma ntake-nnth-a:
  assumes enat m ≤ n
           n ≤ nlength nell
           nfinite nell
  shows (nnth (ntake n nell) m) = (nnth nell m)
using asms
by transfer (auto simp add: lnth-ltake min.absorb1)

```

2.15 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 x21 x22*)
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 ntaken-0 ntaken-NNil ntaken-NNil*)
next
case (*NCons x21 x22*)
then show *?thesis*
proof (*cases n*)
case *0*
then show *?thesis*
by (*simp add: NCons*)
next
case (*Suc nat*)
then show *?thesis*
using *NCons by force*
qed
qed

lemma *NNil-eq-ntaken-iff [simp]*:

(NNil x) = ntaken n nell \longleftrightarrow nell = (NNil x) \vee (n = 0 \wedge nfirst nell = x)
by (*metis ntaken-eq-NNil-iff*)

lemma *ntaken-NCons [code, nitpick-simp]*:

$ntaken\ n\ (NCons\ x\ nell) = (case\ n\ of\ 0 \Rightarrow (NNil\ x) \mid (Suc\ n') \Rightarrow (NCons\ x\ (ntaken\ n'\ nell))\)$
by (cases n) (auto simp add: nfirst-def)

lemma *nhd-ntaken* [simp]:
 $n \neq 0 \implies nhd(ntaken\ n\ nell) = nhd\ nell$
by (cases nell)
 (simp-all add: Nitpick.case-nat-unfold ntaken-NCons)

lemma *ntl-ntaken*:
 $n \neq 0 \implies ntl(ntaken\ n\ nell) = ntaken\ (n-1)\ (ntl\ nell)$
by simp-all
 (metis Suc-pred nellist.exhaust-sel nellist.sel(4) nellist.sel(5) ntaken-NNil ntaken-Suc-NCons)

lemma *ntl-ntaken-0*:
 $ntl(ntaken\ 0\ nell) = (NNil\ (nfirst\ nell))$
by simp

lemma *ntaken-ntl*:
 $ntaken\ n\ (ntl\ nell) = ntl(ntaken\ (Suc\ n)\ nell)$
by (simp add: enat-0-iff(1) ntl-ntaken)

lemma *ntaken-nlength* [simp]:
 $nlength\ (ntaken\ n\ nell) = min\ n\ (nlength\ nell)$
by transfer simp

lemma *ntaken-nmap* [simp]:
 $ntaken\ n\ (nmap\ f\ nell) = nmap\ f\ (ntaken\ n\ nell)$
using enat-0-iff(2) **by** transfer auto

lemma *ntaken-ntaken* [simp]:
 $ntaken\ n\ (ntaken\ m\ nell) = ntaken\ (min\ n\ m)\ nell$
using enat-0-iff(2) **by** transfer auto

lemma *nset-ntaken*:
 $nset\ (ntaken\ n\ nell) \subseteq nset\ nell$
by transfer (simp add: lset-ltake)

lemma *ntaken-all*:
 $nlength\ nell \leq m \implies ntaken\ m\ nell = nell$
by transfer
 (auto simp add: zero-enat-def, metis eSuc-enat eSuc-epred eSuc-ile-mono llength-eq-0 ltake-all)

lemma *ntaken-nnth*:
assumes $k \leq m$
shows $(nnth\ (ntaken\ m\ nell)\ k) = (nnth\ nell\ k)$
using assms
by transfer
 (auto simp add: min-def lnth-ltake ltake-all,
 metis Suc-ile-eq co.enat.exhaust-sel dual-order.trans enat-ord-simps(1) iless-Suc-eq llength-eq-0)

lemma *ntaken-nnth-a*:

assumes $k > m$

shows $(nnth (ntaken m nell) k) = (nnth nell m)$

using *assms*

by *transfer*

(*auto simp add: min-def lnth-ltake ltake-all,*
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.trans enat-ord-simps(1) less-imp-le,
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.cobounded1 min.idem nfinite-ntaken nnth-beyond ntaken-nnth*)

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*:

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

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

apply *transfer*

proof *auto*

show $\bigwedge nellx\ n\ nelly.$

$\neg lnull\ nellx \implies$

$epred\ (llength\ nellx) < enat\ n \implies$

$\neg lnull\ nelly \implies$

$enat\ (Suc\ n) = 0 \implies False$

by (*metis Zero-not-Suc the-enat.simps the-enat-0*)

next

fix *nellxa* :: 'c llist

fix *na*

fix *nellya* :: 'c llist

assume *a0* : $\neg lnull\ nellxa$

assume *a1*: $epred\ (llength\ nellxa) < enat\ na$

assume *a2*: $\neg lnull\ nellya$

show $ltake\ (enat\ (Suc\ na))\ (lappend\ nellxa\ nellya) =$

$lappend\ nellxa\ (ltake\ (enat\ (Suc\ (na - Suc\ (the-enat\ (epred\ (llength\ nellxa))))))\ nellya)$

proof –

```

have 1: llength nellxa ≤ (enat (Suc na))
  by (metis a0 a1 co.enat.exhaust-sel eSuc-enat eSuc-ile-mono leD le-cases llength-eq-0)
have 2: ltake (enat (Suc na)) (lappend nellxa nellya) =
  lappend nellxa (ltake ((enat (Suc na)) - llength nellxa) nellya)
  using ltake-lappend2 using 1 by auto
have 3: Suc (the-enat (epred (llength nellxa))) = llength nellxa
  by (metis a0 a1 co.enat.exhaust-sel eSuc-enat enat-ord-code(4) enat-the-enat llength-eq-0
    not-less-iff-gr-or-eq)
have 4: ((enat (Suc na)) - llength nellxa) =
  (enat (Suc (na - Suc (the-enat (epred (llength nellxa))))))
  by (metis 3 Suc-diff-Suc a1 diff-Suc-1 diff-Suc-Suc enat.inject epred-enat idiff-enat-enat
    less-enatE)
show ?thesis
using 2 4 by presburger
qed
qed

```

lemma *ntaken-eq-ntaken-antimono*:

```

[[ ntaken n nellx = ntaken n nelly; m ≤ n ]] ⇒ ntaken m nellx = ntaken m nelly
by (metis min.orderE ntaken-ntaken)

```

2.16 nsubn

lemma *nsubn-nlength*:

```

nlength(nsubn nell i j) = min (j-i) (nlength nell - i)
by (simp add: nsubn-def)

```

lemma *nsubn-nnth*:

```

assumes i+k < j
shows nnth (nsubn nell i j) k = nnth nell (i+k)
using assms
unfolding nsubn-def
using ntaken-nnth[of k (j - i) (ndropn i nell)]
  ndropn-nnth[of i nell k]
by simp

```

lemma *ntaken-ndropn*:

```

ntaken n (ndropn k nell) = nsubn nell k (n+k)
by (simp add: nsubn-def)

```

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-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-def)

lemma *pref-ntaken-3-nlength*:

$nlength\ (ntaken\ i\ (ntaken\ (i+k)\ nell)) = nlength\ (ntaken\ i\ nell)$

by (metis le-add1 min.orderE ntaken-ntaken)

lemma *pref-ntaken-3-nnth*:

$nnth\ (ntaken\ i\ (ntaken\ (i+k)\ nell))\ m = nnth\ (ntaken\ i\ nell)\ m$

proof –

have 1: $m \leq i \longrightarrow (nnth\ (ntaken\ i\ (ntaken\ (i+k)\ nell))\ m) = (nnth\ (ntaken\ i\ nell)\ m)$

by (simp add: ntaken-nnth)

have 2: $m > i \longrightarrow (nnth\ (ntaken\ i\ (ntaken\ (i+k)\ nell))\ m) = (nnth\ (ntaken\ i\ nell)\ m)$

by (metis le-add1 ntaken-nnth ntaken-nnth-a)

show ?thesis

using 1 2 not-le **by** blast

qed

lemma *pref-ntaken-3*:

$(ntaken\ i\ (ntaken\ (i+k)\ nell)) = (ntaken\ i\ nell)$

by (metis le-add1 min.orderE ntaken-ntaken)

lemma *ntaken-ndropn-swap-nlength*:

assumes $ia + i \leq nlength\ nell$

shows $nlength\ (ntaken\ ia\ (ndropn\ i\ nell)) = nlength\ (ndropn\ i\ (ntaken\ (ia+i)\ nell))$

using *assms* ndropn-nlength[of $i\ (ntaken\ (ia+i)\ nell)$] ntaken-nlength[of $ia\ (ndropn\ i\ nell)$]

by auto

(metis add-diff-cancel-right' enat-minus-mono1 idiff-enat-enat min.orderE)

lemma *ntaken-ndropn-swap-nnth*:

assumes $m \leq ia$

$ia + i \leq nlength\ nell$

shows $nnth\ (ntaken\ ia\ (ndropn\ i\ nell))\ m = nnth\ (ndropn\ i\ (ntaken\ (ia+i)\ nell))\ m$

using *assms*

by (simp add: ntaken-nnth)

lemma *nellist-eq-nnth-eq*:

$(nellx = nelly) \longleftrightarrow nlength\ nellx = nlength\ nelly \wedge (\forall\ i \leq nlength\ nellx. nnth\ nellx\ i = nnth\ nelly\ i)$

by transfer

(metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0 llist-eq-lnth-eq min-absorb1 the-enat.simps)

lemma *ntaken-ndropn-swap*:

assumes $ia + i \leq nlength\ nell$

shows $(ntaken\ ia\ (ndropn\ i\ nell)) = (ndropn\ i\ (ntaken\ (ia+i)\ nell))$

using *assms* nellist-eq-nnth-eq[of $(ntaken\ ia\ (ndropn\ i\ nell))\ (ndropn\ i\ (ntaken\ (ia+i)\ nell))$]

using *ntaken-ndropn-swap-nlength* by (auto simp add: *ntaken-ndropn-swap-nnth*)

2.17 *nzip*

lemma *nzip-nhd*:

$\neg is-NNil\ nelly \wedge \neg is-NNil\ nelly \implies nhd\ (nzip\ nelly\ nelly) = (nhd\ nelly, nhd\ nelly)$

by *transfer*

(auto simp add: *lzip-eq-LCons-conv*,
metis *lzip-eq-LNil-conv*)

lemma *nzip-ntl*:

$\neg is-NNil\ nelly \wedge \neg is-NNil\ nelly \implies ntl(nzip\ nelly\ nelly) = nzip\ (ntl\ nelly)\ (ntl\ nelly)$

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*]:

$nzip\ (NNil\ b)\ nelly = (NNil\ (b, (nnth\ nelly\ 0)))$

$nzip\ nelly\ (NNil\ b) = (NNil\ ((nnth\ nelly\ 0),\ b))$

$nzip\ (NCons\ x\ nelly)\ (NCons\ y\ nelly) = NCons\ (x, y)\ (nzip\ nelly\ nelly)$

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*]:

$is-NNil\ (nzip\ nelly\ nelly) \longleftrightarrow (is-NNil\ nelly) \vee (is-NNil\ nelly)$

by *transfer*

(auto simp add: *lzip-eq-LCons-conv not-lnull-conv*,
metis *lzip-eq-LNil-conv*)

lemma *nzip-eq-NNil-conv*:

$nzip\ nelly\ nelly = (NNil\ (x, y)) \longleftrightarrow$

$((is-NNil\ nelly) \vee (is-NNil\ nelly)) \wedge (nnth\ nelly\ 0) = x \wedge (nnth\ nelly\ 0) = y$

by *auto*

(metis *is-NNil-nzip nellylist.disc(1)*,
metis *Pair-inject is-NNil-nzip nellylist.collapse(1) nellylist.disc(1) nlast-NNil nzip-simps(1)*
nzip-simps(2),
metis *Pair-inject is-NNil-def is-NNil-nzip nellylist.inject(1) nzip-simps(1) nzip-simps(2)*,
metis *nellylist.collapse(1) nnth-NNil nzip-simps(1)*,
metis *nellylist.collapse(1) nnth-NNil nzip-simps(2)*)

lemma *nzip-eq-NCons-conv*:

$nzip\ nelly\ nelly = (NCons\ z\ zs) \longleftrightarrow$

$(\exists x \text{ nellx}' y \text{ nelly}'. \text{nellx} = (\text{NCons } x \text{ nellx}') \wedge \text{nelly} = (\text{NCons } y \text{ nelly}') \wedge$
 $z = (x, y) \wedge \text{zs} = (\text{nzip } \text{nellx}' \text{ nelly}'))$
by (*cases nellx nelly rule: nellist.exhaust[case-product nellist.exhaust]*)
auto

lemma *nzip-nappend*:
 $\text{nlength } \text{nellx} = \text{nlength } \text{nellu}$
 $\implies \text{nzip } (\text{nappend } \text{nellx } \text{nelly}) (\text{nappend } \text{nellu } \text{nellv}) =$
 $\text{nappend } (\text{nzip } \text{nellx } \text{nellu}) (\text{nzip } \text{nelly } \text{nellv})$
by *transfer*
(simp,
meson co.enat.expand llength-eq-0 lzip-lappend)

lemma *nlength-nzip* [*simp*]:
 $\text{nlength } (\text{nzip } \text{nellx } \text{nelly}) = (\min (\text{nlength } \text{nellx}) (\text{nlength } \text{nelly}))$
by *transfer simp*

lemma *ntake-nzip*:
 $\text{ntake } n (\text{nzip } \text{nellx } \text{nelly}) = \text{nzip } (\text{ntake } n \text{ nellx}) (\text{ntake } n \text{ nelly})$
by *transfer*
(simp add: ltake-lzip)

lemma *ntaken-nzip*:
 $\text{ntaken } n (\text{nzip } \text{nellx } \text{nelly}) = \text{nzip } (\text{ntaken } n \text{ nellx}) (\text{ntaken } n \text{ nelly})$
by *transfer*
(simp add: enat-0-iff(1) ltake-lzip)

lemma *ndropn-nzip* [*simp*]:
 $n \leq \text{nlength } \text{nellx} \wedge n \leq \text{nlength } \text{nelly} \implies$
 $\text{ndropn } n (\text{nzip } \text{nellx } \text{nelly}) = \text{nzip } (\text{ndropn } n \text{ nellx}) (\text{ndropn } n \text{ nelly})$
by *transfer*
(auto simp add: min-def,
metis co.enat.exhaust-sel iless-Suc-eq leD llength-eq-0,
metis co.enat.exhaust-sel iless-Suc-eq leD llength-eq-0)

lemma *nzip-niterates*:
 $\text{nzip } (\text{niterates } f \ x) (\text{niterates } g \ y) = \text{niterates } (\lambda(x,y). (f \ x, g \ y)) (x, y)$
by *transfer*
(simp add: lzip-iterates)

lemma *nnth-nzip*:
assumes $n \leq \text{nlength } \text{nellx}$
 $n \leq \text{nlength } \text{nelly}$
shows $\text{nnth } (\text{nzip } \text{nellx } \text{nelly}) \ n = (\text{nnth } \text{nellx } \ n, \text{nnth } \text{nelly } \ n)$
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*:

$nset (nzip\ nellx\ nelly) =$
 $\{ (nnth\ nellx\ n, \text{nth}\ nelly\ n) \mid n. n \leq \min (nlength\ nellx) (nlength\ nelly) \}$
by transfer
 $(auto\ simp\ add: in-lset-conv-lnth,$
 $metis\ Pair-inject\ co.enat.exhaust-sel\ iless-Suc-eq\ llength-eq-0\ lnth-lzip\ min.orderE$
 $the-enat.simps,$
 $metis\ eSuc-epred\ iless-Suc-eq\ llength-eq-0\ lnth-lzip\ min-def-raw\ the-enat.simps)$

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 $nset-nzip-same [simp]$:
 $nset (nzip\ nellx\ nellx) = (\lambda x. (x, x)) \text{ ` } nset\ nellx$
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 $nzip-nmap2$:

$nzip\ nelly\ (nmap\ f\ nelly) = nmap\ (\lambda(x, y). (x, f\ y))\ (nzip\ nelly\ nelly)$
by *transfer*
(simp add: lzip-lmap2)

lemma *nmap-fst-nzip-conv-ntake*:

$nmap\ fst\ (nzip\ nelly\ nelly) = ntake\ (min\ (nlength\ nelly)\ (nlength\ nelly))\ nelly$
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*:

$nmap\ snd\ (nzip\ nelly\ nelly) = ntake\ (min\ (nlength\ nelly)\ (nlength\ nelly))\ nelly$
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*:

$nzip\ nelly\ nelly =$
 $nzip\ (ntake\ (min\ (nlength\ nelly)\ (nlength\ nelly))\ nelly)$
 $(ntake\ (min\ (nlength\ nelly)\ (nlength\ nelly))\ nelly)$
by *transfer*
(auto,
metis co.enat.exhaust-sel epred-min i0-lb llength-lzip ltake-all ltake-lzip order-refl)

lemma *nellist-all2-conv-nzip*:

$nellist-all2\ P\ nelly\ nelly \longleftrightarrow$
 $nlength\ nelly = nlength\ nelly \wedge (\forall (x, y) \in nset(nzip\ nelly\ nelly). P\ x\ y)$
using *nset-nzip[of nelly nelly]*
by *(auto simp add: nellist-all2-conv-all-nnth)*
blast

lemma *nellist-all2-all-nnthI*:

assumes $nlength\ nelly = nlength\ nelly$
 $\bigwedge n. enat\ n \leq nlength\ nelly \implies P\ (nnth\ nelly\ n)\ (nnth\ nelly\ n)$
shows $nellist-all2\ P\ nelly\ nelly$
using *assms* **by** *(simp add: nellist-all2-conv-all-nnth)*

lemma *nellist-all2-nsetD1*:

assumes $nellist-all2\ P\ nelly\ nelly$
 $x \in nset\ nelly$
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 $nellist-all2\ P\ nelly\ nelly$
 $y \in nset\ nelly$
shows $\exists x \in nset\ nelly. P\ x\ y$
using *assms*
by *(metis in-nset-conv-nnth nellist-all2-conv-all-nnth)*

```

lemma nellist-all2-nzipI:
  assumes nellist-all2 P nellx nelly
            nellist-all2 P' nellx' nelly'
  shows nellist-all2 (rel-prod P P') (nzip nellx nellx') (nzip nelly nelly')
using assms
proof (coinduction arbitrary: nellx nellx' nelly nelly')
case (ilist-all2 nx nx' ny ny')
then show ?case
  proof –
    have 1: is-NNil (nzip nx nx') = is-NNil (nzip ny ny')
      by (metis ilist-all2(1) ilist-all2(2) is-NNil-nzip nellist-all2-is-NNilD)
    have 2: (is-NNil (nzip nx nx')  $\implies$ 
      is-NNil (nzip ny ny')  $\implies$ 
      rel-prod P P' (nlast (nzip nx nx')) (nlast (nzip ny ny')))
      by (metis (no-types, lifting) enat-min-eq-0-iff ilist-all2(1) ilist-all2(2)
        is-NNil-nzip min-def-raw nellist.sel(1) nellist-all2-conv-all-nnth
        nzip-eq-NNil-conv order-refl rel-prod-inject zero-enat-def)
    have 3: ( $\neg$  is-NNil (nzip nx nx')  $\implies$ 
       $\neg$  is-NNil (nzip ny ny')  $\implies$ 
      ( $\exists$  nellx nellx' nelly nelly'.
        ntl (nzip nx nx') = nzip nellx nellx'  $\wedge$ 
        ntl (nzip ny ny') = nzip nelly nelly'  $\wedge$ 
        nellist-all2 P nellx nelly  $\wedge$  nellist-all2 P' nellx' nelly')  $\vee$ 
        nellist-all2 (rel-prod P P') (ntl (nzip nx nx')) (ntl (nzip ny ny')))))
      by (auto simp add: nzip-eq-NCons-conv dest: nellist-all2-nhdD intro: nellist-all2-ntlI)
      (meson ilist-all2(1) ilist-all2(2) nellist-all2-ntlI nzip-ntl)
    have 4:  $\neg$  is-NNil (nzip nx nx')  $\longrightarrow$ 
       $\neg$  is-NNil (nzip ny ny')  $\longrightarrow$ 
      rel-prod P P' (nhd (nzip nx nx')) (nhd (nzip ny ny')))
      by (simp add: ilist-all2(1) ilist-all2(2) nellist-all2-nhdD nzip-nhd)
    have 5: ( $\neg$  is-NNil (nzip nx nx')  $\longrightarrow$ 
       $\neg$  is-NNil (nzip ny ny')  $\longrightarrow$ 
      rel-prod P P' (nhd (nzip nx nx')) (nhd (nzip ny ny'))  $\wedge$ 
      ( $\exists$  nellx nellx' nelly nelly'.
        ntl (nzip nx nx') = nzip nellx nellx'  $\wedge$ 
        ntl (nzip ny ny') = nzip nelly nelly'  $\wedge$ 
        nellist-all2 P nellx nelly  $\wedge$  nellist-all2 P' nellx' nelly')  $\vee$ 
        nellist-all2 (rel-prod P P') (ntl (nzip nx nx')) (ntl (nzip ny ny')))))
      using 3 4 by blast
    show ?thesis
      using 1 2 3 4 by presburger
  qed
qed

```

```

lemma ndistinct-nzipI1:
  ndistinct nellx  $\implies$  ndistinct (nzip nellx nelly)
by transfer
  (simp add: ldistinct-lzipI1)

```

lemma *ndistinct-nzipI2*:
 $ndistinct\ nelly \implies ndistinct\ (nzip\ nellx\ nelly)$
by *transfer*
(simp add: ldistinct-lzipI2)

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  $\sim$  nn) (h x)) = is-NNil ((nmap f  $\sim$  nn) (niterates f x))
    by auto
  have 2: (is-NNil ((nmap f  $\sim$  nn) (h x))  $\longrightarrow$ 
    is-NNil ((nmap f  $\sim$  nn) (niterates f x))  $\longrightarrow$ 
    nlast ((nmap f  $\sim$  nn) (h x)) = nlast ((nmap f  $\sim$  nn) (niterates f x)))
    by simp
  have 3: ( $\neg$  is-NNil ((nmap f  $\sim$  nn) (h x))  $\longrightarrow$ 
     $\neg$  is-NNil ((nmap f  $\sim$  nn) (niterates f x))  $\longrightarrow$ 
    nhd ((nmap f  $\sim$  nn) (h x)) = nhd ((nmap f  $\sim$  nn) (niterates f x)))
    by simp
  have 4: ( $\neg$  is-NNil ((nmap f  $\sim$  nn) (h x))  $\longrightarrow$ 
     $\neg$  is-NNil ((nmap f  $\sim$  nn) (niterates f x))  $\longrightarrow$ 
    (ntl ((nmap f  $\sim$  nn) (h x)) = (nmap f  $\sim$  (Suc nn)) (h x)  $\wedge$ 
    ntl ((nmap f  $\sim$  nn) (niterates f x)) = (nmap f  $\sim$  (Suc nn)) (niterates f x)))
    by (metis  $\langle \wedge x. \neg \text{is-NNil } (h\ x) \rangle \langle \wedge x. \text{ntl } (h\ x) = \text{nmap } f\ (h\ x) \rangle$  funpow-simps-right(2)
    nellist.sel(5) niterates-nmap niterates-not-is-NNil ntl-funpow-nmap o-apply)
  show ?thesis using 1 2 3 4 by blast
qed
qed
}
thus ?thesis by auto
qed

```

lemma *nlength-niterates* [*simp*]:

nlength (*niterates* *f* *x*) = ∞

by *transfer auto*

lemma *ndropn-niterates*:

ndropn *n* (*niterates* *f* *x*) = *niterates* *f* ((*f* \sim *n*) *x*)

by *transfer*

(*simp add: ldrown-iterates*)

lemma *nnth-niterates* [*simp*]:

nnth (*niterates* *f* *x*) *n* = (*f* \sim *n*) *x*

by *transfer auto*

lemma *nset-niterates*:

nset (*niterates* *f* *x*) = { (*f* \sim *n*) *x* | *n. True* }

by *transfer*

(*metis lset-iterates*)

2.19 Filtering non-empty lazy lists *nfilter*

lemma *nfilter-NNil* [*simp*]:

shows *nfilter* *P* (*NNil* *b*) = *NNil* *b*

by *transfer auto*

lemma *nfilter-True [simp]*:
shows *nfilter* ($\lambda x. \text{True}$) *nell* = *nell*
by *transfer auto*

lemma *nfilter-False-finite*:
assumes *nfinite nell*
shows *nfilter* ($\lambda x. \text{False}$) *nell* = *nell*
using *assms* **by** *transfer auto*

lemma *nfilter-NCons [simp]*:
assumes $(\exists x \in \text{nset } nell. P x)$
shows *nfilter* *P* (*NCons* *x nell*) = (if *P* *x* then *NCons* *x* (*nfilter* *P nell*) else *nfilter* *P nell*)
using *assms* **by** *transfer auto*

lemma *nfilter-NCons-a [simp]*:
assumes $\neg(\exists x \in \text{nset } nell. P x)$
 $P x$
shows *nfilter* *P* (*NCons* *x nell*) = (*NNil* *x*)
using *assms* **by** *transfer auto*

lemma *nfilter-expand*:
assumes $\exists x \in \text{nset } nell. P x$
shows *nfilter* *P nell* =
 (if is-*NNil* *nell* then *nell*
 else
 (if $(\exists x \in \text{nset}(\text{ntl } nell). P x)$ then
 (if *P* (*nhd* *nell*) then (*NCons* (*nhd* *nell*) (*nfilter* *P* (*ntl* *nell*)))
 else (*nfilter* *P* (*ntl* *nell*))))
 else (*NNil* (*nhd* *nell*))))
using *assms* **by** (*cases nell*) *auto*

lemma *nset-nfilter*:
assumes $\exists x \in \text{nset } nell. P x$
shows *nset* (*nfilter* *P nell*) = *nset nell* $\cap \{xa. P xa\}$
using *assms*
by *transfer auto*

lemma *exist-conj*:
assumes $\exists x \in \text{nset } (nfilter\ Q\ xs). P\ x$
 $\exists x \in \text{nset } xs. Q\ x$
shows $\exists x \in \text{nset } xs. P\ x \wedge Q\ x$
using *assms*
by *transfer (auto split: if-split-asm)*

lemma *nfilter-nfilter [simp]*:
assumes $\exists x \in \text{nset } (nfilter\ Q\ xs). P\ x$
 $\exists x \in \text{nset } xs. Q\ x$
shows *nfilter* *P* (*nfilter* *Q xs*) = *nfilter* ($\lambda x. P\ x \wedge Q\ x$) *xs*
using *assms*

```

proof (transfer fixing: P Q)
fix xsa :: 'a llist
assume  $\neg \text{lnull } xsa \wedge xsa = xsa$ 
assume a1: Bex (lset (if lnull (lfilter Q xsa) then xsa else lfilter Q xsa)) P
assume Bex (lset xsa) Q
then have  $\neg \text{lnull } (\text{lfilter } Q \text{ xsa})$ 
by simp
then have  $\text{lfilter } P \text{ (if lnull (lfilter Q xsa) then xsa else lfilter Q xsa)} = \text{lfilter } P \text{ (lfilter Q xsa)} \wedge$ 
 $\neg \text{lnull } (\text{lfilter } P \text{ (lfilter Q xsa)}) \wedge$ 
 $\neg \text{lnull } (\text{if lnull (lfilter P (if lnull (lfilter Q xsa) then xsa else lfilter Q xsa))$ 
 $\text{then if lnull (lfilter Q xsa) then xsa else lfilter Q xsa}$ 
 $\text{else lfilter P (if lnull (lfilter Q xsa) then xsa else lfilter Q xsa)})$ 
using a1 by (auto split: if-split-asm)
then show  $\neg \text{lnull } (\text{if lnull (lfilter P (if lnull (lfilter Q xsa) then xsa else lfilter Q xsa))$ 
 $\text{then if lnull (lfilter Q xsa) then xsa else lfilter Q xsa}$ 
 $\text{else lfilter P (if lnull (lfilter Q xsa) then xsa else lfilter Q xsa)}) \wedge$ 
 $(\text{if lnull (lfilter P (if lnull (lfilter Q xsa) then xsa else lfilter Q xsa))}$ 
 $\text{then if lnull (lfilter Q xsa) then xsa else lfilter Q xsa}$ 
 $\text{else lfilter P (if lnull (lfilter Q xsa) then xsa else lfilter Q xsa)}) =$ 
 $(\text{if lnull (lfilter } (\lambda a. P \ a \wedge Q \ a) \text{ xsa) then xsa else lfilter } (\lambda a. P \ a \wedge Q \ a) \text{ xsa})$ 
using lfilter-lfilter by auto
qed

```

```

lemma length-nfilter-le [simp]:
  nlength (nfilter P xs)  $\leq$  nlength xs
by transfer (simp add: epred-le-epredI llength-lfilter-ile)

```

```

lemma nfilter-nnth:
  assumes  $(\exists x \in \text{nset } xs. P \ x)$ 
 $i \leq \text{nlength } (\text{nfilter } P \ xs)$ 
shows  $(\exists k \leq \text{nlength } xs. \text{nnth}(\text{nfilter } P \ xs) \ i = \text{nnth } xs \ k)$ 
using assms nset-nfilter[of xs P]
using in-nset-conv-nnth
by (metis Int-iff)

```

```

lemma nfilter-nappend1:
assumes  $\forall x \in \text{nset } xs. \neg P \ x$ 
 $\text{nfinite } xs$ 
 $\exists y \in \text{nset } ys. P \ y$ 
shows  $\text{nfilter } P \ (\text{nappend } xs \ ys) = \text{nfilter } P \ ys$ 
using assms by transfer auto

```

```

lemma nfilter-nappend2:
assumes  $\forall x \in \text{nset } xs. \neg P \ x$ 
 $\exists y \in \text{nset } ys. P \ y$ 
shows  $\text{nfilter } P \ (\text{nappend } ys \ xs) = \text{nfilter } P \ ys$ 
using assms
by transfer
  (auto split: if-split-asm,
   meson in-lset-lappend-iff,

```


metis lappend-LNil2 lappend-inf lfilter-empty-conv lfilter-lappend-lfinite)

lemma *nfilter-nappend [simp]*:
assumes $(\exists x \in \text{nset } (\text{nappend } xs \text{ } ys). P \ x)$
 $(\exists x \in \text{nset } xs. P \ x)$
 $(\exists x \in \text{nset } ys. P \ x)$
shows $\text{nfilter } P \ (\text{nappend } xs \text{ } ys) =$
 $(\text{if } \text{nfinite } xs \text{ then } \text{nappend } (\text{nfilter } P \ xs) \ (\text{nfilter } P \ ys)$
 $\text{else } (\text{nfilter } P \ xs))$
proof (*cases nfinite xs*)
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*:
assumes $\exists x \in \text{nset } (\text{nmap } f \ xs). P \ x$
shows $\text{nfilter } P \ (\text{nmap } f \ xs) = \text{nmap } f \ (\text{nfilter } (P \circ f) \ xs)$
using *assms* **by** *transfer (auto simp add: lfilter-lmap)*

lemma *nlenght-nfilter-nmap[simp]*:
assumes $\exists x \in \text{nset } (\text{nmap } f \ xs). P \ x$
shows $\text{nlenght } (\text{nfilter } P \ (\text{nmap } f \ xs)) = \text{nlenght}(\text{nfilter } (P \circ f) \ xs)$
using *assms* **by** (*simp add: nfilter-nmap*)

lemma *nfilter-is-subset [simp]*:
assumes $\exists x \in \text{nset } xs. P \ x$
shows $\text{nset } (\text{nfilter } P \ xs) \leq \text{nset } xs$
using *assms* **by** (*simp add: nset-nfilter*)

lemma *nfilter-cong[fundef-cong]*:
assumes $xs = ys$
 $(\bigwedge x. x \in \text{nset } ys \implies P \ x = Q \ x)$
shows $\text{nfilter } P \ xs = \text{nfilter } Q \ ys$
using *assms* **by** *transfer auto*

lemma *split-test*:
assumes $2 \leq \text{nlenght } ys$
 $0 < i$
 $i < \text{nlenght } ys$
shows $ys = \text{nappend } (\text{ntaken } (i-1) \ ys) \ (NCons \ (\text{nnth } ys \ i) \ (\text{ndropn } (i+1) \ ys))$
using *assms*
by *transfer*
(simp,
metis eSuc-enat eSuc-epred ileI1 iless-Suc-eq lappend-ltake-enat-ldropn ldropsn-Suc-conv-ldropn
llenght-eq-0 min.strict-order-iff min-def the-enat.simps)

lemma *nset-nappend*:

$nset (nappend\ xs\ ys) = (if\ nfinite\ xs\ then\ nset\ xs \cup nset\ ys\ else\ nset\ xs)$
by transfer (*simp add: lappend-inf*)

lemma *split-nellist-nappend*:

assumes $\exists\ i \leq nlength\ ys.\ nnth\ ys\ i = x \wedge P\ (nnth\ ys\ i) \wedge$
 $(\forall\ j.\ j \neq i \wedge j \leq nlength\ ys \longrightarrow \neg P(nnth\ ys\ j))$

shows $P\ x \wedge$

$(\ ys = (NNil\ x) \vee$
 $(\exists\ vs.\ ys = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs.\ \neg P\ v)) \vee$
 $(\exists\ us.\ ys = nappend\ us\ (NNil\ x) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us.\ \neg P\ u)) \vee$
 $(\exists\ us\ vs.\ ys = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
 $(\forall\ u \in nset\ us.\ \neg P\ u) \wedge (\forall\ v \in nset\ vs.\ \neg P\ v))$
 $)$

proof –

obtain i **where** $1: i \leq nlength\ ys \wedge nnth\ ys\ i = x \wedge P\ (nnth\ ys\ i) \wedge$
 $(\forall\ j.\ j \neq i \wedge j \leq nlength\ ys \longrightarrow \neg P(nnth\ ys\ j))$

using *assms* **by** *auto*

have $01: (\forall\ j.\ (j < i \vee i < j) \wedge j \leq nlength\ ys \longrightarrow \neg P(nnth\ ys\ j))$

using 1

by *auto*

have $2: i=0 \wedge nlength\ ys = 0 \implies P\ x \wedge ys = (NNil\ x)$

by (*metis 1 ndropn-0 ndropn-eq-NNil the-enat-0 zero-enat-def*)

have $3: i=0 \wedge nlength\ ys > 0 \implies P\ x \wedge ys = (NCons\ x\ (ntl\ ys)) \wedge (\forall\ v \in nset\ (ntl\ ys).\ \neg P\ v)$

using 1 *in-nset-conv-nnth[of - ntl ys]*

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 $4: i=0 \wedge nlength\ ys > 0 \implies P\ x \wedge (\exists\ vs.\ ys = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs.\ \neg P\ v))$

using 3 **by** *auto*

have $5: i > 0 \wedge i = nlength\ ys \wedge nfinite\ ys \implies P\ x \wedge ys = nappend\ (ntaken\ (i-1)\ ys)\ (NNil\ x)$

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 $6: i > 0 \wedge i = nlength\ ys \wedge nfinite\ ys \implies (\forall\ u \in nset\ (ntaken\ (i-1)\ ys).\ \neg P\ u)$

using 1

by *transfer*

(*auto,*
 $metis\ in-lset-conv-lnth\ lbutlast-conv-ltake\ less-imp-le\ llength-lbutlast\ lnth-ltake$
 $min.strict-order-iff\ the-enat.simps$)

have $7: i > 0 \wedge i = nlength\ ys \wedge nfinite\ ys \implies P\ x \wedge (\exists\ us.\ ys = nappend\ us\ (NNil\ x) \wedge$
 $nfinite\ us \wedge (\forall\ u \in nset\ us.\ \neg P\ u))$

using $5\ 6$ **by** *auto*

have $8: i > 0 \wedge i < nlength\ ys \implies$

$P\ x \wedge ys = nappend\ (ntaken\ (i-1)\ ys)\ (NCons\ x\ (ndropn\ (i+1)\ ys))$

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: $i > 0 \wedge i < \text{nlength } ys \implies \text{nfinite } (\text{ntaken } (i-1) \text{ } ys)$
using *enat-ord-code(4) nfinite-ntaken by blast*
have 10: $i > 0 \wedge i < \text{nlength } ys \implies (\forall u \in \text{nset } (\text{ntaken } (i-1) \text{ } ys). \neg P u)$
using *01 in-nset-conv-nnth[of - (ntaken (i-1) ys)]*
by *simp (metis Suc-pred le-imp-less-Suc ntaken-nnth)*
have 11: $i > 0 \wedge i < \text{nlength } ys \implies (\forall v \in \text{nset } (\text{ndropn } (i+1) \text{ } ys). \neg P v)$
using *1 01 in-nset-conv-nnth[of - (ndropn (i+1) ys)]*
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: $i > 0 \wedge i < \text{nlength } ys \implies (\exists us \text{ } vs . \text{ } ys = \text{nappend } us \text{ } (NCons \text{ } x \text{ } vs) \wedge \text{nfinite } us \wedge$
 $(\forall u \in \text{nset } us. \neg P u) \wedge (\forall v \in \text{nset } vs. \neg P v))$
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 $0 < \text{nlength } ys$
shows $ys = (NCons (\text{nnth } ys \text{ } 0) (\text{ntl } ys))$
using *assms by (metis ndropn-0 ndropn-Suc-conv-ndropn nellist.sel(5) zero-enat-def)*

lemma *nellist-split-2-last:*

assumes $0 < i$
 $i = \text{nlength } ys$
 $\text{nfinite } ys$
shows $ys = \text{nappend } (\text{ntaken } (i-1) \text{ } ys) (NNil (\text{nnth } ys \text{ } i))$
using *assms*
by *transfer*
(simp,
metis Suc-ile-eq co.enat.exhaust-sel eSuc-enat llength-eq-0 ltake-Suc-conv-snoc-lnth ltake-all
order-refl the-enat.simps)

lemma *nellist-split-3:*

assumes $0 < i$
 $i < \text{nlength } ys$
shows $ys = \text{nappend } (\text{ntaken } (i-1) \text{ } ys) (NCons (\text{nnth } ys \text{ } i) (\text{ndropn } (i+1) \text{ } ys))$
using *assms by transfer*
(auto,
metis eSuc-enat eSuc-epred ileI1 iless-Suc-eq lappend-ltake-enat-ldropn ldropn-Suc-conv-ldropn
less-imp-le llength-eq-0 min-def the-enat.simps)

lemma *NNil-eq-nfilterD:*

assumes $\exists x \in \text{nset } ys. P x$
 $(NNil \text{ } x) = \text{nfilter } P \text{ } ys$
shows $(\exists us \text{ } vs . (ys = (NNil \text{ } x) \vee$
 $(ys = (NCons \text{ } x \text{ } vs) \wedge (\forall v \in \text{nset } vs. \neg P v)) \vee$
 $(ys = \text{nappend } us \text{ } (NNil \text{ } x) \wedge \text{nfinite } us \wedge (\forall u \in \text{nset } us. \neg P u)) \vee$
 $(ys = \text{nappend } us \text{ } (NCons \text{ } x \text{ } vs) \wedge \text{nfinite } us \wedge$

$(\forall u \in \text{nset } us. \neg P u) \wedge (\forall v \in \text{nset } vs. \neg P v)$
 $) \wedge P x$
proof –
have 1: $(\exists i \leq \text{nlength } ys. P (\text{nnth } ys \ i))$
by (metis assms(1) in-nset-conv-nnth)
obtain i **where** 2: $i \leq \text{nlength } ys \wedge P (\text{nnth } ys \ i)$
using 1 **by** auto
have 3: $i = 0 \wedge \text{nlength } ys = 0 \implies P x \wedge ys = (\text{NNil } x)$
by (metis 2 assms(2) ndropn-0 ndropn-eq-NNil nfilter-NNil the-enat-0 zero-enat-def)
have 4: $i=0 \wedge \text{nlength } ys > 0 \implies P x \wedge ys = (\text{NCons } x (\text{ntl } ys)) \wedge (\forall v \in \text{nset } (\text{ntl } ys). \neg P v)$
using 2 assms nellist-split-2-first[of ys] nfilter-expand[of ys P]
by (metis nellist.distinct(1) nellist.sel(3) nlast-NNil)
have 5: $i=0 \wedge \text{nlength } ys > 0 \implies P x \wedge (\exists vs. ys = (\text{NCons } x \ vs) \wedge (\forall v \in \text{nset } vs. \neg P v))$
using 4 **by** auto
have 60: $0 < i \wedge i = \text{nlength } ys \wedge \text{nfinite } ys \implies x = (\text{nnth } ys \ i)$
using 2 assms nellist-split-2-last[of i ys]
proof simp
assume a1: $ys = \text{nappend } (\text{ntaken } (i - \text{Suc } 0) \ ys) (\text{NNil } (\text{nnth } ys \ i))$
assume a2: $P (\text{nnth } ys \ i)$
assume a3: $\text{NNil } x = \text{nfilter } P \ ys$
assume a4: $\exists x \in \text{nset } ys. P x$
have f5: $\forall as \ p \ asa. ((\exists a. (a::'a) \in \text{nset } as \wedge p \ a) \vee \neg \text{nfinite } as \vee (\forall a. a \notin \text{nset } asa \vee \neg p \ a))$
 $\vee \text{nfilter } p (\text{nappend } as \ asa) = \text{nfilter } p \ asa$
by (metis (full-types) nfilter-nappend1)
have f7: $\exists a. a \in \text{nset } (\text{NNil } (\text{nnth } ys \ i)) \wedge P a$
using a2 **by** auto
have f8: $\text{nfilter } P (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \ ys) (\text{NNil } (\text{nnth } ys \ i))) \neq$
 $\text{nappend } (\text{nfilter } P (\text{ntaken } (i - \text{Suc } 0) \ ys)) (\text{nfilter } P (\text{NNil } (\text{nnth } ys \ i)))$
using a3 a1 **by** (metis (full-types) is-NNil-nappend nellist.disc(1))
have f9: $\text{nfinite } (\text{ntaken } (i - \text{Suc } 0) \ ys)$
by simp
obtain $aaa :: 'a$ **where**
f9: $aaa \in \text{nset } ys \wedge P \ aaa$
using a4 **by** blast
then have $aaa \in \text{nset } (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \ ys) (\text{NNil } (\text{nnth } ys \ i)))$
using a1 **by** presburger
then have f10: $\forall a. a \notin \text{nset } (\text{ntaken } (i - \text{Suc } 0) \ ys) \vee \neg P a$
using f9 f8 f7 **by** (meson nfilter-nappend nfinite-ntaken)
then show ?thesis
using f9 f7 f5 a3 a1
proof –
have $\text{NNil } (\text{nnth } ys \ i) = \text{NNil } x$
using nfilter-NNil[of P (nnth ys i)] **by** (metis (no-types) f10 a1 a3 f5 f7 nfinite-ntaken)
then show ?thesis
by blast
qed
qed
have 6: $0 < i \wedge i = \text{nlength } ys \wedge \text{nfinite } ys \implies P x \wedge ys = \text{nappend } (\text{ntaken } (i-1) \ ys) (\text{NNil } x)$
using 2 60 assms nellist-split-2-last[of i ys]
by blast

have 7: $i > 0 \wedge i = \text{nlength } ys \wedge \text{nfinite } ys \implies (\forall u \in \text{nset } (\text{ntaken } (i-1) \text{ } ys). \neg P u)$
using *assms* 6 *nfilter-nappend*[of $(\text{ntaken } (i-1) \text{ } ys) - P]$
by (*metis is-NNil-nappend nellist.disc*(1) *nfinite-ntaken split-nellist-a*)
have 8: $i > 0 \wedge i = \text{nlength } ys \wedge \text{nfinite } ys \implies P x \wedge (\exists us. ys = \text{nappend } us \text{ } (\text{NNil } x) \wedge \text{nfinite } us \wedge (\forall u \in \text{nset } us. \neg P u))$
using 6 7 **by** *auto*
have 9: $i > 0 \wedge i < \text{nlength } ys \implies$
 $P x \wedge ys = \text{nappend } (\text{ntaken } (i-1) \text{ } ys) (\text{NCons } x (\text{ndropn } (i+1) \text{ } ys))$
using 2 *assms nellist-split-3*[of $i \text{ } ys$]
 nfilter-nappend1 [of $(\text{ntaken } (i-1) \text{ } ys) P (\text{NCons } x (\text{ndropn } (i+1) \text{ } ys))]$
 nfilter-nappend [of $(\text{ntaken } (i-1) \text{ } ys) - P]$
proof *simp*
assume *a1*: $\text{enat } i \leq \text{nlength } ys \wedge P (\text{nnth } ys \text{ } i)$
assume *a2*: $\text{NNil } x = \text{nfilter } P \text{ } ys$
assume *a3*: $\exists x \in \text{nset } ys. P x$
assume *a4*: $ys = \text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ } ys) (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys))$
have *f5*: $P (\text{nnth } (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ } ys) (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys)))) i$
using *a1 a4* **by** *auto*
have *f6*: $\text{NNil } x = \text{nfilter } P (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ } ys) (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys)))$
using *a2 a4* **by** *auto*
have *f7*: $\forall as \text{ } p \text{ } asa. ((\exists a. (a::'a) \in \text{nset } as \wedge p a) \vee \neg \text{nfinite } as \vee (\forall a. a \notin \text{nset } asa \vee \neg p a)) \vee$
 $\text{nfilter } p (\text{nappend } as \text{ } asa) = \text{nfilter } p \text{ } asa$
by (*metis (full-types) nfilter-nappend1*)
have $\text{nfilter } P (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ } ys) (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys))) =$
 $\text{nappend } (\text{nfilter } P (\text{ntaken } (i - \text{Suc } 0) \text{ } ys)) (\text{nfilter } P (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys)))$
 \longrightarrow
 $\text{nappend } (\text{nfilter } P (\text{ntaken } (i - \text{Suc } 0) \text{ } ys)) (\text{nfilter } P (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys))) =$
 $\text{NNil } x$
using *f6* **by** *presburger*
then have $\text{nfilter } P (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ } ys) (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys))) \neq$
 $\text{nappend } (\text{nfilter } P (\text{ntaken } (i - \text{Suc } 0) \text{ } ys)) (\text{nfilter } P (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys)))$
by (*metis (no-types) is-NNil-nappend nellist.disc*(1))
then have *f9*: $(\forall a. a \notin \text{nset } (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys)) \vee \neg P a) \vee$
 $\text{nfilter } P (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ } ys) (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys))) =$
 $\text{nfilter } P (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys)) \vee$
 $(\forall a. a \notin \text{nset } (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ } ys) (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys))))$
 $\vee \neg P a)$
using *nfilter-nappend*[of $(\text{ntaken } (i - \text{Suc } 0) \text{ } ys) (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys)) P]$
 nfinite-ntaken [of $(i - \text{Suc } 0) \text{ } ys]$
by (*metis f7*)
obtain *aaa* :: $'a$ **where** *f10*: $aaa \in \text{nset } ys \wedge P \text{ } aaa$
using *a3* **by** *blast*
then have *f11*: $aaa \in \text{nset } (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ } ys) (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys)))$
using *a4* **by** *presburger*
have *f12*: $\text{nnth } (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ } ys) (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys))) i \in$
 $\text{nset } (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys))$
using *a4* **by** *fastforce*
then have $\text{NNil } x = \text{nfilter } P (\text{NCons } (\text{nnth } ys \text{ } i) (\text{ndropn } (\text{Suc } i) \text{ } ys))$
using *f11 f10 f9 f6 f5* **by** (*metis (no-types)*)
then have $\text{NNil } x = \text{nfilter } P (\text{NCons } (\text{nnth } (\text{nappend } (\text{ntaken } (i - \text{Suc } 0) \text{ } ys)$

($NCons$ ($nnth$ ys i) ($ndropn$ (Suc i) ys)))
 i) ($ndropn$ (Suc i) ys))

using a_4 **by** *presburger*

then have $f13$: $\forall a. a \notin nset$ ($ndropn$ (Suc i) ys) $\vee \neg P$ a

using $f5$ **by** (*metis* (*full-types*) *nellist.distinct*(1) *nfilter-NCons*)

have $nfilter$ P ($NCons$ ($nnth$ ($nappend$ ($ntaken$ ($i - Suc$ 0) ys)
 $(NCons$ ($nnth$ ys i) ($ndropn$ (Suc i) ys)))
 i) ($ndropn$ (Suc i) ys)) = $NNil$ x

using ($NNil$ $x = nfilter$ P ($NCons$ ($nnth$ ys i) ($ndropn$ (Suc i) ys))) a_4 **by** *presburger*

then have $x = nnth$ ($nappend$ ($ntaken$ ($i - Suc$ 0) ys) ($NCons$ ($nnth$ ys i) ($ndropn$ (Suc i) ys))) i

using $f13$ $f5$ **by** *simp*

then have $f14$: $x = nnth$ ys i

using a_4 **by** *presburger*

then have $ys = nappend$ ($ntaken$ ($i - Suc$ 0) ys) ($NCons$ x ($ndropn$ (Suc i) ys))

using a_4 **by** *blast*

then show P $x \wedge ys = nappend$ ($ntaken$ ($i - Suc$ 0) ys) ($NCons$ x ($ndropn$ (Suc i) ys))

using $f14$ $a1$ **by** *blast*

qed

have 10: $i > 0 \wedge i < nlength$ $ys \implies (\forall u \in nset$ ($ntaken$ ($i-1$) ys). $\neg P$ u)

using 9 *assms* *nfilter-nappend*[of ($ntaken$ ($i-1$) ys) - P]

by (*metis* *is-NNil-nappend* *nellist.disc*(1) *nellist.set-intros*(2) *nfinite-ntaken*)

have 11: $i > 0 \wedge i < nlength$ $ys \implies (\forall v \in nset$ ($ndropn$ ($i+1$) ys). $\neg P$ v)

using 9 10 *assms* *nfilter-nappend*[of ($ntaken$ ($i-1$) ys) - P]

nfilter-nappend1[of ($ntaken$ ($i-1$) ys) P ($NCons$ x ($ndropn$ ($i+1$) ys))]

by (*metis* *nellist.distinct*(1) *nellist.set-intros*(2) *nfilter-NCons* *nfinite-ntaken*)

have 12: $i > 0 \wedge i < nlength$ $ys \implies$

P $x \wedge (\exists us\ vs. ys = nappend$ us ($NCons$ x vs) $\wedge nfinite$ $us \wedge$

$(\forall u \in nset$ $us. \neg P$ u) $\wedge (\forall v \in nset$ $vs. \neg P$ v))

using 9 10 11 **using** *assms*(1) **by** *fastforce*

show *?thesis*

by (*metis* 12 2 3 5 8 *dual-order.order-iff-strict* *neq0-conv* *nlength-eq-enat-nfiniteD*
zero-enat-def)

qed

lemma *nfilter-eq-NNilD*:

assumes $\exists x \in nset$ $ys. P$ x

$nfilter$ P $ys = (NNil$ $x)$

shows $(\exists us\ vs. (ys = (NNil$ $x) \vee$

$(ys = (NCons$ x $vs) \wedge (\forall v \in nset$ $vs. \neg P$ $v))) \vee$

$(ys = nappend$ us ($NNil$ x) $\wedge nfinite$ $us \wedge (\forall u \in nset$ $us. \neg P$ $u))) \vee$

$(ys = nappend$ us ($NCons$ x $vs) \wedge nfinite$ $us \wedge$

$(\forall u \in nset$ $us. \neg P$ $u) \wedge (\forall v \in nset$ $vs. \neg P$ $v)))$

$) \wedge P$ $x)$

using *assms* *NNil-eq-nfilterD*[of ys P x] **by** *simp*

lemma *nfilter-eq-NNil-iff*:

assumes $\exists x \in nset$ $ys. P$ x

shows $(nfilter$ P $ys = (NNil$ $x)) \longleftrightarrow$

$(\exists us\ vs. (ys = (NNil$ $x) \vee$

$(ys = (NCons$ x $vs) \wedge (\forall v \in nset$ $vs. \neg P$ $v))) \vee$

$(ys = nappend\ us\ (NNil\ x) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us. \neg P\ u)) \vee$
 $(ys = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
 $(\forall\ u \in nset\ us. \neg P\ u) \wedge (\forall\ v \in nset\ vs. \neg P\ v))$
 $) \wedge P\ x)$

proof –

have 1: $(nfilter\ P\ ys = (NNil\ x)) \implies$
 $(\exists\ us\ vs. (ys = (NNil\ x) \vee$
 $(ys = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs. \neg P\ v)) \vee$
 $(ys = nappend\ us\ (NNil\ x) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us. \neg P\ u)) \vee$
 $(ys = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
 $(\forall\ u \in nset\ us. \neg P\ u) \wedge (\forall\ v \in nset\ vs. \neg P\ v))$
 $) \wedge P\ x)$

using *assms nfilter-eq-NNilD*[of *ys P x*] **by** *simp*

have 2: $(\exists\ us\ vs. (ys = (NNil\ x) \vee$
 $(ys = (NCons\ x\ vs) \wedge (\forall\ v \in nset\ vs. \neg P\ v)) \vee$
 $(ys = nappend\ us\ (NNil\ x) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us. \neg P\ u)) \vee$
 $(ys = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge$
 $(\forall\ u \in nset\ us. \neg P\ u) \wedge (\forall\ v \in nset\ vs. \neg P\ v))$
 $) \wedge P\ x) \implies (nfilter\ P\ ys = (NNil\ x))$

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 $\exists\ x \in nset\ ys. P\ x$

$(NCons\ x\ xs) = nfilter\ P\ ys$

shows $(\exists\ us\ vs.$

$(ys = (NCons\ x\ vs) \vee$

$ys = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us. \neg P\ u)) \wedge$

$(\exists\ x \in nset\ vs. P\ x) \wedge P\ x \wedge xs = nfilter\ P\ vs)$

using *assms* **by** *transfer (auto split: if-split-asm simp add: NCons-eq-nfilterD-help)*

lemma *nfilter-eq-NConsD*:

assumes $\exists\ x \in nset\ ys. P\ x$

$nfilter\ P\ ys = (NCons\ x\ xs)$
shows $(\exists\ us\ vs.$
 $(\ ys = (NCons\ x\ vs) \vee$
 $\ ys = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us. \neg P\ u)) \wedge$
 $(\exists\ x \in nset\ vs. P\ x) \wedge P\ x \wedge xs = nfilter\ P\ vs)$
using *assms* $NCons\text{-}eq\text{-}nfilterD[of\ ys\ P\ x\ xs]$ **by** *simp*

lemma *nfilter-eq-NCons-iff*:

assumes $\exists\ x \in nset\ ys. P\ x$

shows $nfilter\ P\ ys = (NCons\ x\ xs) \longleftrightarrow$

$(\exists\ us\ vs.$

$(\ ys = (NCons\ x\ vs) \vee$

$\ ys = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us. \neg P\ u)) \wedge$

$(\exists\ x \in nset\ vs. P\ x) \wedge P\ x \wedge xs = nfilter\ P\ vs)$

proof –

have 1: $nfilter\ P\ ys = (NCons\ x\ xs) \implies$

$(\exists\ us\ vs.$

$(\ ys = (NCons\ x\ vs) \vee$

$\ ys = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us. \neg P\ u)) \wedge$

$(\exists\ x \in nset\ vs. P\ x) \wedge P\ x \wedge xs = nfilter\ P\ vs)$

using *assms* $nfilter\text{-}eq\text{-}NConsD[of\ ys\ P\ x\ xs]$ **by** *simp*

have 2: $(\exists\ us\ vs.$

$(\ ys = (NCons\ x\ vs) \vee$

$\ ys = nappend\ us\ (NCons\ x\ vs) \wedge nfinite\ us \wedge (\forall\ u \in nset\ us. \neg P\ u)) \wedge$

$(\exists\ x \in nset\ vs. P\ x) \wedge P\ x \wedge xs = nfilter\ P\ vs) \implies nfilter\ P\ ys = (NCons\ x\ xs)$

using $nfilter\text{-}nappendI[of\ -\ P]$ $nfilter\text{-}NCons[of\ -\ P\ x]$

by (*auto*, *meson* $nfilter\text{-}NCons$, *metis*)

show *?thesis* **using** 1 2 **by** *blast*

qed

lemma *nfilter-id-conv*:

assumes $(\exists\ x \in nset\ xs. P\ x)$

shows $(nfilter\ P\ xs = xs) = (\forall\ x \in nset\ xs. P\ x)$ (**is** *?lhs* = *?rhs*)

using *assms* **by** *transfer* (*auto* *simp* *add*: $lfilter\text{-}id\text{-}conv$)

lemma *nfilter-idem*:

assumes $(\exists\ x \in nset\ xs. P\ x)$

shows $nfilter\ P\ (nfilter\ P\ xs) = nfilter\ P\ xs$

using *assms* **by** *transfer* *auto*

lemma *nfilter-ndropn-nlength*:

assumes $k \leq nlength\ nell$

$\exists\ x \in nset\ ((ndropn\ k\ nell)). P\ x$

shows $nlength(nfilter\ P\ ((ndropn\ k\ nell))) \leq nlength\ (nfilter\ P\ (nell))$

using *assms*

by *transfer*

(*auto* *simp* *add*: *min-def* *dest*: *in-lset-ldropnD*,

metis *co.enat.exhaust-sel* *epred-le-epredI* *iless-Suc-eq* $lfilter\text{-}ldropn\text{-}llength\ llength\text{-}eq\text{-}0$)

2.20 *nfuse*

lemma *nfuse-nlength*:

shows $nlength\ (nfuse\ xs\ ys) = (nlength\ xs) + (nlength\ ys)$

unfolding *nfuse-def*

by (*cases ys*) (*simp, simp add: eSuc-plus-1*)

lemma *nfuse-nnth*:

assumes $i \leq nlength\ (nfuse\ xs\ ys)$

$nlast\ xs = nfirst\ ys$

shows $(i \leq nlength\ xs \longrightarrow nnth\ (nfuse\ xs\ ys)\ i = nnth\ xs\ i)$

\wedge

$(nlength\ xs < i \wedge i \leq nlength\ (nfuse\ xs\ ys) \longrightarrow$

$nnth\ (nfuse\ xs\ ys)\ i = nnth\ ys\ (i - (the-enat\ (nlength\ xs))))$

unfolding *nfuse-def*

by (*cases ys*)

(*auto simp add: nnth-nappend,*

metis Suc-diff-Suc enat-iless enat-ord-simps(2) ndropn-Suc-NCons ndropn-nfirst the-enat.simps)

lemma *nfuse-nnth-a*:

assumes $j \leq nlength\ ys$

$nlast\ xs = nfirst\ ys$

$nfinite\ xs$

shows $nnth\ (nfuse\ xs\ ys)\ ((the-enat\ (nlength\ xs)) + j) = (nnth\ ys\ j)$

using *assms* **unfolding** *nfuse-def*

by (*cases is-NNil ys*)

(*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 $nlast\ xs = nfirst\ ys$

shows $nfuse\ xs\ ys =$

(*if nfinite xs then*

(*if is-NNil xs then ys else nappend (ntaken (the-enat(epred(nlength xs))) xs) ys*)

else xs)

proof (*cases ys*)

case (*NNil x1*)

then show *?thesis*

proof (*cases nfinite xs*)

case *True*

then show *?thesis*

proof (*cases xs*)

case (*NNil x1*)

then show *?thesis* **using** *assms*

by (*simp add: nfuse-def*)

(*metis nappend-snocn nellist.collapse(1) nellist.collapse(2) nhd-nappend nhd-snocn*)

next

```

case (NCons x21 x22)
then show ?thesis unfolding nfuse-def 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-def)
qed
next
case (NCons x21 x22)
then show ?thesis
proof (cases nfinite xs)
case True
then show ?thesis
unfolding nfuse-def
proof auto
show is-NNil ys  $\implies$  is-NNil xs  $\implies$  xs = ys
by (simp add: NCons)
show nfinite xs  $\implies$  is-NNil ys  $\implies$   $\neg$  is-NNil xs  $\implies$ 
  xs = nappend (ntaken (the-enat (epred (nlength xs))) xs) ys
using assms NCons True by simp
show  $\neg$  is-NNil ys  $\implies$  is-NNil xs  $\implies$  nappend xs (ntl ys) = ys
using assms NCons True
by (metis nappend-NNil nellist.collapse(1) nellist.sel(5) nnth-0 ntaken-0 ntaken-nlast)
show nfinite xs  $\implies$   $\neg$  is-NNil ys  $\implies$   $\neg$  is-NNil xs  $\implies$ 
  nappend xs (ntl ys) = nappend (ntaken (the-enat (epred (nlength xs))) xs) ys
using assms NCons True
by (cases xs)
  ( 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-def)
qed
qed

lemma nfuse-leftneutral :
  nfuse (NNil (nfirst xs)) xs = xs
by (simp add: nfuse-nappend)

lemma nfuse-rightneutral :

```

$nfuse\ xs\ (NNil\ (nlast\ xs)) = xs$
unfolding *nfuse-def*
by *simp*

lemma *nfirst-nfuse* :
assumes $nlast\ xs = nfirst\ ys$
shows $nfirst\ (nfuse\ xs\ ys) = nfirst\ xs$
using *assms* **unfolding** *nfuse-def* **by** *transfer auto*

lemma *nlast-nfuse* :
assumes $nlast\ xs = nfirst\ ys$
 $nfinite\ xs$
shows $nlast\ (nfuse\ xs\ ys) = nlast\ ys$
using *assms* **unfolding** *nfuse-def* **by** *transfer (auto, metis lhd-LCons-ltl llast-LCons llast-lappend)*

lemma *nfuseassoc* :
assumes $(nlast\ xs) = (nfirst\ ys)$
 $(nlast\ ys) = (nfirst\ zs)$
shows $(nfuse\ xs\ (nfuse\ ys\ zs)) = (nfuse\ (nfuse\ xs\ ys)\ zs)$
using *assms* **unfolding** *nfuse-def*
by *transfer (auto simp add: lappend-assoc, simp add: lappend-ltl)*

lemma *ntaken-nfuse* :
assumes $nlast\ xs = nfirst\ ys$
 $nfinite\ xs$
shows $(ntaken\ (the-enat\ (nlength\ xs))\ (nfuse\ xs\ ys)) = xs$
proof (*cases is-NNil ys*)
case *True*
then show *?thesis* **using** *assms* **by** (*metis ndropn-eq-NNil ndropn-nlast nfuse-def ntaken-all*)
next
case *False*
then show *?thesis* **using** *assms*
by (*metis add.left-neutral enat-le-plus-same(2) nfinite-nlength-enat nfuse-def ntaken-all*
 $ntaken-nappend1\ the-enat.simps$)
qed

lemma *ndropn-nfuse* :
assumes $nlast\ xs = nfirst\ ys$
 $nfinite\ xs$
shows $(ndropn\ (the-enat\ (nlength\ xs))\ (nfuse\ xs\ ys)) = ys$
proof (*cases is-NNil ys*)
case *True*
then show *?thesis* **using** *assms* **by** (*metis ndropn-nlast nfuse-def 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*
 $nfuse-def\ nfuse-leftneutral\ nlength-code\ the-enat.simps$)
qed

lemma *nfuse-ntaken-ndropn-nlength* :
assumes $n \leq \text{nlength } xs$
shows $\text{nlength } (\text{nfuse } (\text{ntaken } n \text{ } xs) (\text{ndropn } n \text{ } xs)) = \text{nlength } xs$
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-nth* :
assumes $n \leq \text{nlength } xs$
 $i \leq \text{nlength } xs$
shows $\text{nnth } (\text{nfuse } (\text{ntaken } n \text{ } xs) (\text{ndropn } n \text{ } xs)) \text{ } i = \text{nnth } xs \text{ } i$
using *assms*
nfuse-nnth[*of* $i \text{ } (\text{ntaken } n \text{ } xs) (\text{ndropn } n \text{ } xs)$]
nfuse-ntaken-ndropn-nlength[*of* $n \text{ } xs$]
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 $n \leq \text{nlength } xs$
shows $\text{nfuse } (\text{ntaken } n \text{ } xs) (\text{ndropn } n \text{ } xs) = xs$
using *assms*
by (*simp add: nfuse-ntaken-ndropn-nlength nfuse-ntaken-ndropn-nth nellist-eq-nnth-eq*)

lemma *nsubn-nfuse*:
assumes $(\text{enat } k) \leq n$
 $(\text{enat } n) \leq m$
 $(\text{enat } m) \leq \text{nlength } xs$
shows $\text{nfuse } (\text{nsubn } xs \text{ } k \text{ } n) (\text{nsubn } xs \text{ } n \text{ } m) = (\text{nsubn } xs \text{ } k \text{ } m)$
using *assms*
proof –
have 1: $\text{nlast}(\text{nsubn } xs \text{ } k \text{ } n) = (\text{nnth } xs \text{ } n)$
by (*metis assms(1) enat-ord-simps(1) le-add-diff-inverse2 nsubn-def ntaken-ndropn-nlast*)
have 2: $\text{nfirst}(\text{nsubn } xs \text{ } n \text{ } m) = (\text{nnth } xs \text{ } n)$
by (*simp add: ndropn-nfirst nsubn-def ntaken-ndropn-nfirst-a*)
have 3: $\text{nlength } (\text{nsubn } xs \text{ } k \text{ } n) = n - k$
using *assms nsubn-nlength*[*of* $xs \text{ } k \text{ } n$]
by (*metis dual-order.trans enat-minus-mono1 idiff-enat-enat min-absorb1*)
have 4: $\text{nlength } (\text{nsubn } xs \text{ } n \text{ } m) = m - n$
by (*metis assms(3) enat-minus-mono1 idiff-enat-enat min-def nsubn-nlength*)
have 5: $\text{nlength } (\text{nsubn } xs \text{ } k \text{ } m) = m - k$
by (*metis assms(3) enat-minus-mono1 idiff-enat-enat min-absorb1 nsubn-nlength*)
have 6: $\text{nlength}(\text{nfuse } (\text{nsubn } xs \text{ } k \text{ } n) (\text{nsubn } xs \text{ } n \text{ } m)) = \text{nlength}(\text{nsubn } xs \text{ } k \text{ } m)$
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: $(\forall i. i \leq \text{nlength}(\text{nsubn } xs \text{ } k \text{ } m) \longrightarrow (\text{nnth } (\text{nfuse } (\text{nsubn } xs \text{ } k \text{ } n) (\text{nsubn } xs \text{ } n \text{ } m)) \text{ } i) = (\text{nnth } xs \text{ } (k+i)))$
proof
fix i
show $i \leq \text{nlength}(\text{nsubn } xs \text{ } k \text{ } m) \longrightarrow$

```

      (nnth (nfuse (nsubn xs k n) (nsubn xs n m)) i) = (nnth xs (k+i))
proof -
  have 41: nlength (nsubn xs k m) = (m-k)
    using 5 by blast
  have 42: i ≤ nlength (nsubn xs k m) → nnth (nfuse (nsubn xs k n) (nsubn xs n m)) i =
    (if i ≤ n - k then (nnth (nsubn xs k n) i)
     else (nnth (nsubn xs n m) (i-(n-k))))
    by (simp add: 1 2 3 6 nfuse-nnth)
  have 43: i ≤ (m-k) → nnth (nfuse (nsubn xs k n) (nsubn xs n m)) i =
    (if i ≤ (n-k) then (nnth xs (k+i)) else (nnth xs (n+(i-(n-k)))))
    using assms 42 5 unfolding nsubn-def
    by (auto simp add: ntaken-nnth)
    (metis enat-ord-simps(1) min.bounded-iff,
     metis enat-ord-simps(1) min.bounded-iff)
  have 44: i ≤ (m-k) → nnth (nfuse (nsubn xs k n) (nsubn xs n m)) i =
    (nnth xs (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: (∀ i. i ≤ nlength(nsubn xs k m) → (nnth (nsubn xs k m) i) = (nnth xs (k+i)))
  using assms by (simp add: nsubn-def ntaken-nnth)
show ?thesis
  by (simp add: 6 7 8 nellist-eq-nnth-eq)
qed

```

2.21 Setup for Lifting/Transfer

2.21.1 Relator and predicator properties

abbreviation *nellist-all* == *pred-nellist*

2.21.2 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 ==> *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 ==> *nellist-all2* A ==> *nellist-all2* A) *NCons* *NCons*

unfolding *rel-fun-def* **by** *simp*
declare *NCons-transfer* [*transfer-rule*]

lemma *case-nellist-transfer* [*transfer-rule*]:
 $((A \text{ ===> } C) \text{ ===> } (A \text{ ===> } \textit{nellist-all2 } A \text{ ===> } C) \text{ ===> } \textit{nellist-all2 } A \text{ ===> } C)$
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*]:
 $((A \text{ ===> } (=)) \text{ ===> } (A \text{ ===> } C) \text{ ===> } (A \text{ ===> } C) \text{ ===> } (A \text{ ===> } A) \text{ ===> } A \text{ ===> } \textit{nellist-all2 } C)$

unfold-nellist unfold-nellist

proof(*rule rel-funI*) +

fix *IS-NNIL1* :: '*a* \Rightarrow bool **and** *IS-NNIL2*

NLAST1 NLAST2 NHD1 NHD2 NTL1 NTL2 x y

assume *rel*: (*A* ===> (=)) *IS-NNIL1 IS-NNIL2* (*A* ===> *C*) *NLAST1 NLAST2*

(*A* ===> *C*) *NHD1 NHD2* (*A* ===> *A*) *NTL1 NTL2*

and *A x y*

show *nellist-all2 C* (*unfold-nellist IS-NNIL1 NLAST1 NHD1 NTL1 x*)

(*unfold-nellist IS-NNIL2 NLAST2 NHD2 NTL2 y*)

using $\langle A \ x \ y \rangle$ **using** *rel* **by** (*coinduction arbitrary: x y*) (*auto 4 4 elim: rel-funE*)

qed

lemma *corec-nellist-transfer* [*transfer-rule*]:
 $((A \text{ ===> } (=)) \text{ ===> } (A \text{ ===> } C) \text{ ===> } (A \text{ ===> } C) \text{ ===> } (A \text{ ===> } (=)) \text{ ===> } (A \text{ ===> } \textit{nellist-all2 } C))$

$\text{===> } (A \text{ ===> } A) \text{ ===> } A \text{ ===> } \textit{nellist-all2 } C$) *corec-nellist corec-nellist*

by (*simp add: nellist.corec-transfer*)

lemma *ntl-transfer2* [*transfer-rule*]:

(*nellist-all2 A* ===> *nellist-all2 A*) *ntl ntl*

unfolding *ntl-def[abs-def]* **by** *transfer-prover*

declare *ntl-transfer* [*transfer-rule*]

lemma *nset-transfer2* [*transfer-rule*]:

(*nellist-all2 A* ===> *rel-set A*) *nset nset*

by (*simp add: nellist.set-transfer*)

lemma *nmap-transfer4* [*transfer-rule*]:

$((A \text{ ===> } B) \text{ ===> } \textit{nellist-all2 } A \text{ ===> } \textit{nellist-all2 } B)$ *nmap nmap*

by (*simp add: nellist.map-transfer*)

declare *nmap-transfer* [*transfer-rule*]

lemma *is-NNil-transfer2* [*transfer-rule*]:

(*nellist-all2 A* ===> (=)) *is-NNil is-NNil*

by(*auto dest: nellist-all2-is-NNilD*)

declare *is-NNil-transfer* [*transfer-rule*]

lemma *snocn-transfer2* [*transfer-rule*]:

```

  (nellist-all2 A ==> A ==> nellist-all2 A ) snocn snocn
unfolding rel-fun-def
by (metis nappend-snocn nellist-all2-NNil nellist-all2-nappendI)
declare snocn-transfer [transfer-rule]

lemma nappend-transfer [transfer-rule]:
  (nellist-all2 A ==> ( nellist-all2 A ) ==> nellist-all2 A ) nappend nappend
by(auto intro: nellist-all2-nappendI elim: rel-funE)
declare nappend.transfer [transfer-rule]

lemma lappendn-transfer [transfer-rule]:
  (llist-all2 A ==> nellist-all2 A ==> nellist-all2 A ) lappendn lappendn
unfolding rel-fun-def
by transfer(auto intro: llist-all2-lappendI)
declare lappendn.transfer [transfer-rule]

lemma nellist-of-llist-a-transfer2 [transfer-rule]:
  (A ==> llist-all2 A ==> nellist-all2 A ) nellist-of-llist-a nellist-of-llist-a
by (simp add: rel-funI)
declare nellist-of-llist-a-transfer [transfer-rule]

lemma nlength-transfer [transfer-rule]:
  (nellist-all2 A ==> (=)) nlength nlength
by(auto dest: nellist-all2-nlengthD)
declare nlength.transfer [transfer-rule]

lemma ndropn-transfer [transfer-rule]:
  ((=) ==> nellist-all2 A ==> nellist-all2 A ) ndropn ndropn
unfolding rel-fun-def
by transfer (auto intro: llist-all2-ldropnI simp add: llist-all2-ldropnI llist-all2-llengthD )
declare ndropn.transfer [transfer-rule]

lemma ntake-transfer [transfer-rule]:
  ((=) ==> nellist-all2 A ==> nellist-all2 A ) ntake ntake
unfolding rel-fun-def
by transfer (auto simp add: llist-all2-ltakeI)
declare ntake.transfer [transfer-rule]

lemma ntaken-transfer [transfer-rule]:
  ((=) ==> nellist-all2 A ==> nellist-all2 A ) ntaken ntaken
unfolding rel-fun-def
by transfer (auto simp add: llist-all2-ltakeI)
declare ntaken.transfer [transfer-rule]

lemma nzip-transfer [transfer-rule]:
  (nellist-all2 A ==> nellist-all2 B ==> nellist-all2 (rel-prod A B)) nzip nzip
by (auto intro: nellist-all2-nzipI)

lemma niterates-transfer [transfer-rule]:

```

```

  ((A ==> A) ==> A ==> nellist-all2 A) niterates niterates
unfolding rel-fun-def
apply transfer
using iterates-transfer unfolding rel-fun-def
by blast

```

```

lemma nfilter-transfer [transfer-rule]:
  ( (A ==> (=)) ==> nellist-all2 A ==> nellist-all2 A) nfilter nfilter
unfolding rel-fun-def
by transfer
  (auto intro: llist-all2-lfilterI dest: llist-all2-lfiniteD llist-all2-lsetD1,
    meson llist-all2-lsetD2)
declare nfilter.transfer [transfer-rule]

```

```

lemma 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` and `nidx` are defined for nellists together with a library of lemmas.

```
theory NELListKFilter
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: enat nellx ≤ nlength (ndropns ia) ⟹
      is-NNil (ndropn nellx (ndropns ia)) = is-NNil (ndropns (ndropn nellx ia))
    by (simp add: is-NNil-ndropn ndropns-nlength)
    have 2: enat nellx ≤ nlength (ndropns ia) ⟹
      (is-NNil (ndropn nellx (ndropns ia)) ⟶
       is-NNil (ndropns (ndropn nellx ia)) ⟶
       nlast (ndropn nellx (ndropns ia)) = nlast (ndropns (ndropn nellx ia)))
    by (metis dual-order.antisym ndropn-eq-NNil ndropns.disc(2) ndropns.simps(3) ndropns-nlength
      ndropns-nnth nellist.case-eq-if nellist.collapse(1) the-enat.simps)
    have 3: enat nellx ≤ nlength (ndropns ia) ⟹
      (¬ is-NNil (ndropn nellx (ndropns ia)) ⟶
       ¬ is-NNil (ndropns (ndropn nellx ia)) ⟶
       nhd (ndropn nellx (ndropns ia)) = nhd (ndropns (ndropn nellx ia)))
    by (metis Nat.add-0-right ndropn-nnth ndropns.disc(1) ndropns-nlength ndropns-nnth nhd-conv-nnth
      nhd-ndropns)
  qed

```

```

have 4: enat nellx ≤ nlength (ndropns ia) ==>
  (¬ is-NNil (ndropn nellx (ndropns ia)) →
   ¬ is-NNil (ndropns (ndropn nellx ia)) →
   (∃ nell i.
    ntl (ndropn nellx (ndropns ia)) = ndropn i (ndropns nell) ∧
    ntl (ndropns (ndropn nellx ia)) = ndropns (ndropn i nell) ∧
    enat i ≤ nlength (ndropns nell)))
by (metis Suc-ile-eq is-NNil-ndropn ndropn-Suc-conv-ndropn ndropns-code(2) ndropns-nlength
    nellist.sel(5) order.order-iff-strict)
show ?thesis
using 1 2 3 4 Eq-nellist by blast
qed
qed

```

```

lemma ndropns-nfilter-nnth:
assumes i ≤ nlength (nfilter P (ndropns nell))
  ∃ nelly ∈ nset(ndropns nell). P nelly
shows P (nnth (nfilter P (ndropns nell)) i)
using assms using nset-nfilter[of ndropns nell P]
by (metis (full-types) Int-iff in-nset-conv-nnth mem-Collect-eq )

```

```

lemma nnth-zero-ndropn:
  nnth (ndropn n nell) 0 = nnth nell n
by simp

```

```

lemma in-nset-ndropns-nhd:
  x ∈ nset nell ⟷ ( ∃ ys. x=(nnth ys 0) ∧ ys ∈ nset(ndropns nell))
by auto
  (metis in-nset-conv-nnth ndropns-nlength nmap-first-ndropns nnth-nmap,
   metis Nat.add-0-right in-nset-conv-nnth in-nset-ndropns ndropn-nnth)

```

```

lemma nset-ndropns-nhd:
  nset nell = {(nnth ys 0) | ys. ys ∈ nset(ndropns nell) }
by auto
  (meson in-nset-ndropns-nhd,
   metis in-nset-conv-nnth in-nset-ndropns nnth-zero-ndropn)

```

```

lemma nellist-all2-ndropnsI:
  nellist-all2 A xs ys ==> nellist-all2 (nellist-all2 A) (ndropns xs) (ndropns ys)
by (coinduction arbitrary: xs ys)
  (auto simp add: nellist.case-eq-if dest: nellist-all2-nhdD nellist-all2-is-NNilD
   intro: nellist-all2-ntII)

```

```

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

context

includes *nellist.lifting*

begin

lift-definition *nkfilter* :: (*'a* \Rightarrow *bool*) \Rightarrow *nat* \Rightarrow *'a* *nellist* \Rightarrow *nat* *nellist*
is $\lambda P n ll.$ (if *lnull*(*kfilter* *P* *n* *ll*) then (*iterates* *Suc* *n*) else *kfilter* *P* *n* *ll*)
by *simp*

lift-definition *nleast* :: (*'a* \Rightarrow *bool*) \Rightarrow *'a* *nellist* \Rightarrow *nat*
is $\lambda P ll.$ *lleast* *P* *ll*
by *auto*

lift-definition *nidx* :: *nat* *nellist* \Rightarrow *bool*
is $\lambda ll.$ *lidx* *ll*
by *auto*

3.2 nkfilter

lemma *nkfilter-NNil* [*simp*]:
 shows $P b \Longrightarrow nkfilter\ P\ n\ (NNil\ b) = (NNil\ n)$
by *transfer auto*

lemma *nkfilter-NCons* [*simp*]:
 assumes $(\exists x \in nset\ nell.\ P\ x)$
 shows $nkfilter\ P\ n\ (NCons\ x\ nell) =$
 $(if\ P\ x\ then\ NCons\ n\ (nkfilter\ P\ (Suc\ n)\ nell)\ else\ nkfilter\ P\ (Suc\ n)\ nell)$
using *assms*
by *transfer*
 $(auto\ simp\ add:\ kfilter-lnull-conv)$

lemma *nkfilter-NCons-a* [*simp*]:
 assumes $\neg(\exists x \in nset\ nell.\ P\ x)$
 $P\ x$
 shows $nkfilter\ P\ n\ (NCons\ x\ nell) = (NNil\ n)$
using *assms*
by *transfer*
 $(auto\ simp\ add:\ kfilter-lnull-conv)$

lemma *nkfilter-nlength*:
 assumes $\exists x \in nset\ nell.\ P\ x$
 shows $nlength(nkfilter\ P\ n\ nell) = nlength(nfilter\ P\ nell)$
using *assms*
by *transfer*
 $(auto\ simp\ add:\ kfilter-lnull-conv\ kfilter-llength)$

lemma *nkfilter-upperbound*:
 assumes $\exists x \in nset\ nell.\ P\ x$
 $i \leq nlength\ (nkfilter\ P\ n\ nell)$

shows $\text{nnth } (\text{nkfilter } P \ n \ \text{nell}) \ i \leq n + \text{nlength } \text{nell}$
using *assms*
by *transfer*
(auto,
simp add: *kfilter-lnull-conv*,
metis *co.enat.exhaust-sel iadd-Suc-right iless-Suc-eq kfilter-upperbound llength-eq-0*
min-absorb1 the-enat.simps)

lemma *nkfilter-lowerbound*:
assumes $\exists x \in \text{nset } \text{nell}. P \ x$
 $i \leq \text{nlength } (\text{nkfilter } P \ n \ \text{nell})$
shows $n \leq \text{nnth } (\text{nkfilter } P \ n \ \text{nell}) \ i$
using *assms*
by *transfer*
(auto,
metis *co.enat.collapse iless-Suc-eq kfilter-lnth-n-zero-a llength-eq-0 min-absorb1 the-enat.simps*)

lemma *nkfilter-mono*:
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 *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 $\exists x \in \text{nset } \text{nell}. P \ x$
 $i \leq \text{nlength } (\text{nkfilter } P \ n \ \text{nell})$
shows $(\text{nnth } \text{nell } ((\text{nnth } (\text{nkfilter } P \ n \ \text{nell}) \ i) - n)) = (\text{nnth } (\text{nfilter } P \ \text{nell}) \ i)$
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*: $x \in \text{lset } \text{nella}$
assume *a2*: $P \ x$
assume *a3*: $\text{enat } ia \leq \text{epred } (\text{llength } (\text{kfilter } Pa \ na \ \text{nella}))$
assume *a4*: $\neg \text{lnull } \text{nella}$
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*: $\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{kfilter } Pa \ na \ \text{nella}))) = \text{enat } ia$
using *f5 a3* **by** *blast*
then have *f8*: $(\text{enat } ia < \text{llength } (\text{kfilter } Pa \ 0 \ \text{nella})) =$
 $(\text{enat } (\text{the-enat } (\text{min } (\text{enat } ia) (\text{epred } (\text{llength } (\text{kfilter } Pa \ na \ \text{nella})))))) <$
 $\text{llength } (\text{kfilter } Pa \ na \ \text{nella}))$
by (simp add: *kfilter-llength*)

have f9: $\forall n p na as. \neg enat\ n < llength\ (kfilter\ p\ na\ (as::'a\ llist)) \vee$
 $lnth\ (kfilter\ p\ na\ as)\ n - na = lnth\ (kfilter\ p\ 0\ as)\ n$
using kfilter-lnth-n-zero **by** blast
have eSuc (epred (llength nella)) = enat 0 + llength nella
using a4 **by** (metis co.enat.exhaust-sel gen-llength-def llength-code llength-eq-0)
then have enat (the-enat (min (enat ia) (epred (llength (kfilter Pa na nella))))) <
 $llength\ (kfilter\ Pa\ na\ nella) \longrightarrow$
 $enat\ (lnth\ (kfilter\ Pa\ na\ nella)$
 $(the-enat\ (min\ (enat\ ia)\ (epred\ (llength\ (kfilter\ Pa\ na\ nella)))) - na)$
 $< eSuc\ (epred\ (llength\ nella))$
using f9 f8 f7 f6 the-enat.simps **by** presburger
then have f10: enat (the-enat (min (enat ia) (epred (llength (kfilter Pa na nella))))) <
 $llength\ (kfilter\ Pa\ na\ nella) \longrightarrow$
 $enat\ (lnth\ (kfilter\ Pa\ na\ nella)$
 $(the-enat\ (min\ (enat\ ia)\ (epred\ (llength\ (kfilter\ Pa\ na\ nella)))) - na)$
 $\leq eSuc\ (epred\ (llength\ nella))$
using iless-Suc-eq **by** blast
have f11: $\forall n p na as. \neg enat\ n < llength\ (kfilter\ p\ na\ as) \vee lnth\ as\ (lnth\ (kfilter\ p\ na\ as)\ n - na) =$
 $(lnth\ (lfilter\ p\ as)\ n::'a)$
using lfilter-kfilter **by** blast
have f12: the-enat (min (enat ia) (epred (llength (lfilter Pa nella)))) =
 $the-enat\ (min\ (enat\ ia)\ (epred\ (llength\ (kfilter\ Pa\ na\ nella))))$
by (simp add: kfilter-llength)
have $\neg lnull\ (kfilter\ Pa\ na\ nella)$
using a1 a2 kfilter-not-llnull-conv **by** auto
then have enat (the-enat (min (enat ia) (epred (llength (kfilter Pa na nella))))) <
 $llength\ (kfilter\ Pa\ na\ nella)$
using f7 a3 **by** (metis co.enat.exhaust-sel iless-Suc-eq llength-eq-0 the-enat.simps)
then show lnth nella
 $(the-enat$
 $(min\ (enat\ (lnth\ (kfilter\ Pa\ na\ nella)$
 $(the-enat\ (min\ (enat\ ia)\ (epred\ (llength\ (kfilter\ Pa\ na\ nella)))) - na))$
 $(epred\ (llength\ nella)))) =$
 $lnth\ (lfilter\ Pa\ nella)\ (the-enat\ (min\ (enat\ ia)\ (epred\ (llength\ (lfilter\ Pa\ nella))))$
using f12 f11 f10 f5 the-enat.simps **by** presburger
qed

lemma in-nkfilter-nset:

assumes $\exists x \in nset\ nell. P\ x$

shows $x \in nset(nkfilter\ P\ n\ nell) \longleftrightarrow x \in \{ n+i \mid i. i \leq nlength\ nell \wedge P\ (nnth\ nell\ i) \}$

using assms

by transfer

(auto simp add: kfilter-llnull-conv min-def,
metis co.enat.exhaust-sel gen-llength-def iless-Suc-eq kfilter-holds kfilter-llength-n-zero
kfilter-lnth-n-zero kfilter-lowerbound kfilter-upperbound ldistinct-Ex1 ldistinct-kfilter
le-add-diff-inverse llength-code llength-eq-0,
metis add.commute co.enat.exhaust-sel iless-Suc-eq kfilter-holds-not-a llength-eq-0)

lemma nkfilter-nset:

assumes $\exists x \in nset\ nell. P\ x$

shows $nset(nkfilter\ P\ n\ nell) = \{ n+i \mid i. i \leq nlength\ nell \wedge P\ (nnth\ nell\ i) \}$
using *assms in-nkfilter-nset[of nell P - n]* **by** *blast*

lemma *nlength-nkfilter-le [simp]*:

assumes $\exists x \in nset\ nell. P\ x$
shows $nlength\ (nkfilter\ P\ n\ nell) \leq nlength\ nell$
using *assms*
by *transfer*
(auto simp add: kfilter-lnull-conv epred-le-epredI kfilter-llength llength-lfilter-ile)

lemma *nkfilter-nleast*:

assumes $\exists x \in nset\ xs. P\ x$
shows $nnth\ (nkfilter\ P\ n\ xs)\ 0 = n+(nleast\ P\ xs)$
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 $\exists x \in nset\ nell. P\ x$
shows $ndistinct(nkfilter\ P\ n\ nell)$
using *assms*
by *transfer*
(auto simp add: kfilter-lnull-conv ldistinct-kfilter)

lemma *nkfilter-ndropn-nset*:

assumes $\exists x \in nset\ (ndropn\ k\ nell). P\ x$
 $k \leq nlength\ nell$
shows $nset(nkfilter\ P\ n\ (ndropn\ k\ nell)) = \{ n+i \mid i. i \leq nlength\ nell - k \wedge P\ (nnth\ nell\ (k+i)) \}$
using *assms nkfilter-nset[of (ndropn k nell) P n]*
ndropn-nnth[of - nell k] **by** *auto*

lemma *nkfilter-ndropn-nset-b*:

assumes $\exists x \in nset\ (ndropn\ k\ nell). P\ x$
 $k \leq nlength\ nell$
shows $nset(nkfilter\ P\ n\ (ndropn\ k\ nell)) = \{ n+ i \mid i. i \leq nlength\ nell - k \wedge P\ (nnth\ nell\ (i+k)) \}$
proof –
have *1*: $\{ n+i \mid i. i \leq nlength\ nell - k \wedge P\ (nnth\ nell\ (k+i)) \} =$
 $\{ n+ i \mid i. i \leq nlength\ nell - k \wedge P\ (nnth\ nell\ (i+k)) \}$
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 $P\ (nnth\ nell\ ((nnth\ (nkfilter\ P\ n\ nell)\ k) - n))$
 $k \leq nlength\ (nfilter\ P\ nell)$
shows $(\exists x \in nset\ nell. P\ x)$
using *assms using exists-nnth-nset* **by** *blast*

lemma *nkfilter-nlength-n-zero*:

assumes $(\exists x \in \text{nset } \text{nell}. P x)$
shows $\text{nlength}(\text{nkfilter } P \ n \ \text{nell}) = \text{nlength}(\text{nkfilter } P \ 0 \ \text{nell})$
using *assms* **by** (*simp add: nkfilter-nlength*)

lemma *nkfilter-nnth-n-zero*:

assumes $(\exists x \in \text{nset } \text{nell}. P x)$
 $k \leq \text{nlength}(\text{nkfilter } P \ n \ \text{nell})$
shows $(\text{nnth } (\text{nkfilter } P \ n \ \text{nell}) \ k) - n = (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)$
using *assms*
by *transfer*
 (*auto split: if-split-asm,*
metis kfilter-lnull-conv,
metis kfilter-lnull-conv,
metis co.enat.exhaust-sel iless-Suc-eq kfilter-llength-n-zero kfilter-lnth-n-zero llength-eq-0
min.orderE the-enat.simps)

lemma *nkfilter-n-zero*:

assumes $(\exists x \in \text{nset } \text{nell}. P x)$
shows $(\text{nkfilter } P \ n \ \text{nell}) = \text{nmap } (\lambda i. i+n) (\text{nkfilter } P \ 0 \ \text{nell})$
proof –
have 1: $\text{nlength}(\text{nkfilter } P \ n \ \text{nell}) = \text{nlength } (\text{nmap } (\lambda i. i+n) (\text{nkfilter } P \ 0 \ \text{nell}))$
using *assms nkfilter-nlength-n-zero* **by** *fastforce*
have 2: $\bigwedge k. k \leq \text{nlength } (\text{nkfilter } P \ n \ \text{nell}) \longrightarrow$
 $\text{nnth } (\text{nkfilter } P \ n \ \text{nell}) \ k = \text{nnth } (\text{nmap } (\lambda i. i+n) (\text{nkfilter } P \ 0 \ \text{nell})) \ k$
using 1 *assms nkfilter-nnth-n-zero[of nell P - n]*
by (*metis diff-add nkfilter-lowerbound nlength-nmap nnth-nmap*)
show ?thesis **by** (*simp add: 1 2 nellist-eq-nnth-eq*)
qed

lemma *nkfilter-n-zero-a*:

assumes $(\exists x \in \text{nset } \text{nell}. P x)$
shows $(\text{nkfilter } P \ 0 \ \text{nell}) = \text{nmap } (\lambda i. i-n) (\text{nkfilter } P \ n \ \text{nell})$
proof –
have 1: $\text{nlength}(\text{nkfilter } P \ 0 \ \text{nell}) = \text{nlength } (\text{nmap } (\lambda i. i-n) (\text{nkfilter } P \ n \ \text{nell}))$
by (*metis assms nkfilter-nlength-n-zero nlength-nmap*)
have 2: $\bigwedge k. k \leq \text{nlength}(\text{nkfilter } P \ 0 \ \text{nell}) \longrightarrow$
 $\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k = \text{nnth } (\text{nmap } (\lambda i. i-n) (\text{nkfilter } P \ n \ \text{nell})) \ k$
by (*simp add: 1 assms nkfilter-nnth-n-zero*)
show ?thesis **by** (*simp add: 1 2 nellist-eq-nnth-eq*)
qed

lemma *nkfilter-holds*:

assumes $(\exists x \in \text{nset } \text{nell}. P x)$
 $y \in \text{nset}(\text{nkfilter } P \ n \ \text{nell})$
shows $P (\text{nnth } \text{nell } (y-n))$
using *assms in-nkfilter-nset[of nell P]* **by** *force*

lemma *nkfilter-holds-not*:

assumes $(\exists x \in \text{nset } \text{nell}. P x)$
 $y \in \{i+n \mid i. i \leq \text{nlength } \text{nell}\} - (\text{nset } (\text{nkfilter } P \ n \ \text{nell}))$

shows $\neg P (nnth\ nell\ (y-n))$
using *assms nkfilter-nset[of nell P n]* **by** *auto*

lemma *nkfilter-holds-a:*

assumes $(\exists x \in nset\ nell.\ P\ x)$
 $i \leq nlength\ nell$
 $(i+n) \in nset(nkfilter\ P\ n\ nell)$

shows $P (nnth\ nell\ i)$

using *assms nkfilter-holds[of nell P i+n n]* **by** *simp*

lemma *nkfilter-holds-not-a:*

assumes $(\exists x \in nset\ nell.\ P\ x)$
 $i \leq nlength\ nell$
 $P (nnth\ nell\ i)$

shows $(i+n) \in nset(nkfilter\ P\ n\ nell)$

using *assms by (simp add: in-nkfilter-nset)*

lemma *nkfilter-holds-b:*

assumes $(\exists x \in nset\ nell.\ P\ x)$
 $i \leq nlength\ nell$

shows $(i+n) \in nset(nkfilter\ P\ n\ nell) = P (nnth\ nell\ i)$

using *assms by (meson nkfilter-holds-a nkfilter-holds-not-a)*

lemma *nkfilter-holds-c:*

assumes $(\exists x \in nset\ nell.\ P\ x)$
 $n \leq i$
 $i-n \leq nlength\ nell$

shows $i \in nset(nkfilter\ P\ n\ nell) = P (nnth\ nell\ (i-n))$

using *assms*

by *(metis diff-add idiff-enat-enat nkfilter-holds-b)*

lemma *nkfilter-holds-not-b:*

assumes $(\exists x \in nset\ nell.\ P\ x)$
 $n \leq i$
 $i-n \leq nlength\ nell$

shows $i \notin nset (nkfilter\ P\ n\ nell) = (\neg P (nnth\ nell\ (i-n)))$

using *assms*

by *(simp add: nkfilter-holds-c)*

lemma *nkfilter-disjoint-nset-coset:*

assumes $(\exists x \in nset\ nell.\ P\ x)$

shows $(\{i+n|i.\ i \leq nlength\ nell\} - nset(nkfilter\ P\ n\ nell)) \cap nset (nkfilter\ P\ n\ nell) = \{\}$

using *assms by (simp add: inf commute)*

lemma *nidx-less:*

assumes *nidx nell*

$Suc(n+k) \leq nlength\ nell$

shows $nnth\ nell\ n < nnth\ nell\ (Suc(n+k))$

using *assms*

apply *transfer*

```

proof auto
fix nella :: nat llist and na :: nat and ka :: nat
assume a1: lidx nella
assume a2: enat (Suc (na + ka)) ≤ epred (llength nella)
assume a3:  $\neg$  lnull nella
have f4: min na (na + ka) = na
by simp
have f5:  $\forall n$  na. min (enat n) (enat na) = enat (min n na)
using min-enat-simps(1) by blast
have f6:  $\forall e$  ea. if (e::enat) ≤ ea then min e ea = e else min e ea = ea
by (simp add: min-def-raw)
have enat (na + ka) ≤ epred (llength nella)
using a2 Suc-ile-eq by force
then have min (enat na) (epred (llength nella)) = enat na
by (metis f4 f6 min.bounded-iff min-enat-simps(1))
then show lnth nella (the-enat (min (enat na) (epred (llength nella)))) <
lnth nella (the-enat (min (enat (Suc (na + ka))) (epred (llength nella))))
using f6 a3 a2 a1
by (metis (no-types) co.enat.exhaust-sel iless-Suc-eq lidx-less llength-eq-0 the-enat.simps)
qed

```

```

lemma nidx-less-eq:
assumes nidx nell
k ≤ j
j ≤ nlength nell
shows nnth nell k ≤ nnth nell j
using assms
by transfer
(auto, metis co.enat.exhaust-sel dual-order.trans enat-ord-simps(1) iless-Suc-eq lidx-less-eq
llength-eq-0 min.orderE the-enat.simps)

```

```

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-gr-first:
assumes nidx nell
0 < i
i ≤ nlength nell
shows (nnth nell 0) < nnth nell i
using assms nidx-less[of nell 0 i-1]
by transfer auto

```

```

lemma nidx-expand:
nidx nell  $\longleftrightarrow$  ( $\forall i$ . (Suc i) ≤ nlength nell  $\longrightarrow$  nnth nell i < nnth nell (Suc i))
by transfer
(auto simp add: min-def Suc-ile-eq,

```

metis Extended-Nat.eSuc-mono co.enat.exhaust-sel eSuc-enat lidx-def llength-eq-0,
metis Suc-ile-eq co.enat.exhaust-sel iless-Suc-eq less-imp-le lidx-def llength-eq-0)

lemma *nidx-nkfilter-expand:*

assumes $(\exists x \in \text{nset } \text{nell}. P x)$
 $(\text{Suc } i) \leq \text{nlength}(\text{nkfilter } P \text{ n nell})$
shows $\text{nnth } (\text{nkfilter } P \text{ n nell}) \ i < \text{nnth } (\text{nkfilter } P \text{ n nell}) \ (\text{Suc } i)$
using *assms* **by** (*simp add: nkfilter-mono*)

lemma *nidx-nkfilter:*

assumes $(\exists x \in \text{nset } \text{nell}. P x)$
shows *nidx* $(\text{nkfilter } P \text{ n nell})$
using *assms* *nidx-nkfilter-expand*[of *nell* *P* - *n*] *nidx-expand*[of $(\text{nkfilter } P \text{ n nell})$]
by *blast*

lemma *nidx-nkfilter-gr-eq:*

assumes $(\exists x \in \text{nset } \text{nell}. P x)$
 $k \leq j$
 $j \leq \text{nlength } (\text{nkfilter } P \text{ n nell})$
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 $(\exists x \in \text{nset } \text{nell}. P x)$
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 $(\exists x \in \text{nset } \text{nell}. P x)$
 $k \leq \text{nlength } (\text{nkfilter } P \text{ n nell})$
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 $(\exists x \in \text{nset } \text{nell}. P x)$
 $i < (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ 0)$
shows $\neg P \ (\text{nnth } \text{nell } i)$
using *assms*
by *transfer*
(auto simp add: min-def split: if-split-asm,
meson kfilter-not-before llength-eq-0 not-gr-zero,
metis dual-order.strict-trans kfilter-not-before less-enatE llength-eq-0 not-gr-zero
not-le-imp-less the-enat.simps,
meson le-less-linear less-enatE less-nat-zero-code,
meson le-less-linear less-enatE not-less0)

lemma *nkfilter-n-not-before:*

```

assumes ( $\exists x \in \text{nset } (\text{ndropn } n \text{ nell}). P x$ )
           $n \leq \text{nlength nell}$ 
           $n \leq i$ 
           $i < (\text{nnth } (\text{nkfilter } P n (\text{ndropn } n \text{ nell})) 0)$ 
shows  $\neg P (\text{nnth nell } i)$ 
using assms
apply transfer
unfolding min-def
using zero-enat-def
proof (auto split: if-split-asm)
show  $\bigwedge n \text{ nell } P i x.$ 
           $\text{enat } 0 = 0 \implies$ 
           $\neg \text{lnull nell} \implies$ 
           $\text{enat } n \leq \text{epred } (\text{llength nell}) \implies$ 
           $n \leq i \implies$ 
           $\neg \text{lnull } (\text{kfilter } P n (\text{ldropn } n \text{ nell})) \implies$ 
           $i < \text{lnth } (\text{kfilter } P n (\text{ldropn } n \text{ nell})) 0 \implies$ 
           $x \in \text{lset } (\text{ldropn } n \text{ nell}) \implies P x \implies \text{enat } i \leq \text{epred } (\text{llength nell}) \implies P (\text{lnth nell } i) \implies \text{False}$ 
by (metis co.enat.exhaust-sel iless-Suc-eq kfilter-n-not-before llength-eq-0 not-gr-zero)
next
fix na :: nat and nella :: 'b llist and Pa :: 'b  $\Rightarrow$  bool and ia :: nat and x :: 'b
assume a1: ia < lnth (kfilter Pa na (ldropn na nella)) 0
assume a2:  $\neg \text{lnull } (\text{kfilter Pa na (ldropn na nella))$ 
assume a3: enat na  $\leq$  epred (llength nella)
assume a4:  $\neg \text{lnull nella}$ 
assume a5:  $\neg \text{enat ia} \leq \text{epred } (\text{llength nella})$ 
assume a6: Pa (lnth nella (the-enat (epred (llength nella))))
have f7:  $\forall p n \text{ bs}. \text{lnull } (\text{kfilter } p n (\text{bs}::'b \text{ llist})) \vee \text{lnth } (\text{kfilter } p n \text{ bs}) 0 = n + \text{lleast } p \text{ bs}$ 
by (meson kfilter-lnth-zero)
then have f8: lnth (kfilter Pa na (ldropn na nella)) 0 = na + lleast Pa (ldropn na nella)
using a2 by blast
have f9: 0 < llength (kfilter Pa na (ldropn na nella))
using a2 by simp
have f10: na  $\leq$  the-enat (epred (llength nella))
by (metis a3 a5 enat-ord-code(4) enat-ord-simps(1) enat-the-enat less-imp-le)
have f11: the-enat (epred (llength nella)) < lnth (kfilter Pa na (ldropn na nella)) 0
by (metis a1 a5 dual-order.strict-trans enat-ord-simps(2) enat-ord-simps(3) enat-the-enat not-le-imp-less)
then show False using kfilter-n-not-before[of Pa na nella (the-enat (epred (llength nella)))]
using f10 f11
by (metis a3 a4 a6 co.enat.exhaust-sel f9 iless-Suc-eq llength-eq-0)
qed

```

lemma *nkfilter-not-after:*

```

assumes ( $\exists x \in \text{nset nell}. P x$ )
           $\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k < i$ 
           $\text{nlength}(\text{nkfilter } P 0 \text{ nell}) = (\text{enat } k)$ 
           $i \leq \text{nlength nell}$ 
shows  $\neg P (\text{nnth nell } i)$ 
using assms

```

by *transfer*

(*auto simp add: min-def split: if-split-asm,*
metis co.enat.exhaust-sel diff-Suc-1 eSuc-enat iless-Suc-eq kfilter-not-after llength-eq-0
not-gr-zero)

lemma *nkfilter-n-not-after:*

assumes $(\exists x \in \text{nset } (\text{ndropn } n \text{ nell}). P x)$
 $n \leq \text{nlength } \text{nell}$
 $\text{nnth } (\text{nkfilter } P \ n \ (\text{ndropn } n \ \text{nell})) \ k < i$
 $\text{nlength}(\text{nkfilter } P \ n \ (\text{ndropn } n \ \text{nell})) = (\text{enat } k)$
 $i \leq \text{nlength } \text{nell}$

shows $\neg P (\text{nnth } \text{nell } i)$

using *assms*

by *transfer*

(*auto split: if-split-asm,*
metis diff-Suc-1 eSuc-enat eSuc-epred iless-Suc-eq kfilter-n-not-after llength-eq-0 min.orderE
not-gr-zero the-enat.simps)

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 *ndistinct-Ex1:*

assumes *ndistinct nell*

$x \in \text{nset } \text{nell}$

shows $\exists !i. i \leq \text{nlength } \text{nell} \wedge (\text{nnth } \text{nell } i) = x$

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 min.orderE the-enat.simps)

lemma *nidx-nset-eq:*

assumes *nidx nellx*

nidx nelly

$\text{nset } \text{nellx} = \text{nset } \text{nelly}$

shows $\text{nellx} = \text{nelly}$

using *assms*

by *transfer*

(simp add: bi-unique-cr-nellist-help lid_x-lset-eq nid_x.rep-eq)

lemma *ntaken-nset*:

assumes $k \leq \text{nlength } nell$

shows $\text{nset } (\text{ntaken } k \text{ } nell) = \{ (\text{nnth } nell \ i) \mid i. i \leq k \}$

using *assms*

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

(meson ntaken-nnth,
metis dual-order.irrefl enat-ord-simps(2) le-less-trans not-le-imp-less ntaken-nnth)

lemma *ndropn-nset*:

assumes $k \leq \text{nlength } nell$

shows $\text{nset } (\text{ndropn } k \text{ } nell) = \{ (\text{nnth } nell \ i) \mid i. k \leq i \wedge i \leq \text{nlength } nell \}$

using *assms*

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

(metis enat-ile enat-ord-simps(1) le-add1 le-cases nnth-beyond,
metis enat-ile le-cases le-iff-add ndropn-nlength ndropn-nnth nnth-beyond)

lemma *nfilter-nkfilter-ntaken-nid_x-a*:

assumes $\exists x \in \text{nset } nell. P \ x$

$k \leq \text{nlength } (\text{nfilter } P \text{ } nell)$

shows $\text{nid}_x (\text{ntaken } k (\text{nkfilter } P \text{ } nell))$

using *assms*

by (simp add: nid_x-expand nid_x-nkfilter-gr ntaken-nnth)

lemma *nfilter-nkfilter-ndropn-nid_x-a*:

assumes $\exists x \in \text{nset } nell. P \ x$

$k \leq \text{nlength } (\text{nfilter } P \text{ } nell)$

shows $\text{nid}_x (\text{ndropn } k (\text{nkfilter } P \text{ } nell))$

using *assms*

by *transfer*

(auto simp add: kfilter-lnull-conv,
metis (no-types, lifting) gr-implies-not-zero kfilter-llength leI lfilter-kfilter-ldropn-lid_x-a
lid_x-def llength-eq-0 lnull-ldropn)

lemma *nfilter-nkfilter-ntaken-nid_x-b*:

assumes $P (\text{nnth } nell \ ((\text{nnth } (\text{nkfilter } P \ 0 \text{ } nell) \ k)))$

$k \leq \text{nlength } (\text{nfilter } P \text{ } nell)$

shows $\text{nid}_x (\text{nkfilter } P \ 0 (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \text{ } nell) \ k) \text{ } nell))$

using *assms* nid_x-nkfilter[of (ntaken (nnth (nkfilter P 0 nell) k) nell) P 0]

by (metis nfinite-ntaken nset-nlast ntaken-nlast)

lemma *nfilter-nkfilter-ntaken-nid_x-b-1*:

assumes $\exists x \in \text{nset } nell. P \ x$

$k \leq \text{nlength } (\text{nfilter } P \text{ } nell)$

shows $\text{nid}_x (\text{nkfilter } P \ 0 (\text{ntaken } (\text{nnth } (\text{nkfilter } P \ 0 \text{ } nell) \ k) \text{ } nell))$

using *assms* nfilter-nkfilter-ntaken-nid_x-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-nid_x-b-2*:

assumes P ($\text{nnth nell } ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - \text{n})$)
 $k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
shows $\text{nidx } (\text{nkfilter } P \text{ n } (\text{ntaken } (\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) \text{ nell}))$
using $\text{assms nidx-nkfilter}[of (\text{ntaken } (\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) \text{ nell}) \text{ P n}]$
by ($\text{metis diff-le-self exists-nnth-nset ntaken-nnth}$)

lemma $\text{nfilter-nkfilter-ntaken-nidx-b-3}$:

assumes $\exists x \in \text{nset nell}. P x$
 $k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
shows $\text{nidx } (\text{nkfilter } P \text{ n } (\text{ntaken } (\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) \text{ nell}))$
using $\text{assms nfilter-nkfilter-ntaken-nidx-b-2}[of P \text{ nell n k}] \text{ nkfilter-holds}$
by ($\text{metis in-nset-conv-nnth nkfilter-nlength}$)

lemma $\text{nfilter-nkfilter-ndropn-nidx-b}$:

assumes P ($\text{nnth nell } ((\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ k}))$)
 $k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
shows $\text{nidx } (\text{nkfilter } P \text{ 0 } (\text{ndropn } (\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ k}) \text{ nell}))$
using $\text{assms nidx-nkfilter}[of (\text{ndropn } (\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ k}) \text{ nell}) \text{ P 0}]$
by ($\text{metis diff-add diff-zero in-nset-conv-nnth ndropn-nnth zero-enat-def zero-le}$)

lemma $\text{nfilter-nkfilter-ndropn-nidx-b-1}$:

assumes $\exists x \in \text{nset nell}. P x$
 $k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
shows $\text{nidx } (\text{nkfilter } P \text{ 0 } (\text{ndropn } (\text{nnth } (\text{nkfilter } P \text{ 0 nell}) \text{ k}) \text{ nell}))$
using $\text{assms nfilter-nkfilter-ndropn-nidx-b}[of P \text{ nell k}] \text{ nkfilter-holds}[of \text{nell P}]$
by ($\text{metis in-nset-conv-nnth nkfilter-nlength nkfilter-nnth-n-zero}$)

lemma $\text{nfilter-nkfilter-ndropn-nidx-b-2}$:

assumes P ($\text{nnth nell } ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - \text{n})$)
 $k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
shows $\text{nidx } (\text{nkfilter } P ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - \text{n})$
 $(\text{ndropn } ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - \text{n}) \text{ nell}))$

proof –

have 1: $\text{enat } (\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - \text{n} \leq \text{nlength nell}$
using $\text{nkfilter-upperbound}[of \text{nell P}] \text{ nkfilter-nnth-n-zero}[of \text{nell P k n}] \text{ assms}$
by ($\text{metis gen-nlength-def idiff-enat-enat nfilter-nkfilter-ntaken-nlength-0}$
 $\text{nkfilter-nlength nlength-code}$)
have 2: $\text{nset } (\text{ndropn } ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - \text{n}) \text{ nell}) =$
 $\{ (\text{nnth nell } i) \mid i. ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - \text{n}) \leq i \wedge i \leq \text{nlength nell} \}$
using $\text{ndropn-nset}[of ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - \text{n}) \text{ nell}] \text{ 1 by simp}$
have 3: $\exists x \in \text{nset } (\text{ndropn } ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - \text{n}) \text{ nell}). P x$
using 1 2 $\text{assms}(1)$ **by** auto
show $?thesis$ **using** 3
 $\text{nidx-nkfilter}[of (\text{ndropn } ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - \text{n}) \text{ nell}) \text{ P } ((\text{nnth } (\text{nkfilter } P \text{ n nell}) \text{ k}) - \text{n})]$
by blast

qed

lemma $\text{nfilter-nkfilter-ndropn-nidx-b-3}$:

assumes $\exists x \in \text{nset nell}. P x$
 $k \leq \text{nlength } (\text{nfilter } P \text{ nell})$

shows $\text{nidx} (\text{nkfilter } P ((\text{nnth} (\text{nkfilter } P \text{ n nell}) k) - n) \text{ (ndropn } ((\text{nnth} (\text{nkfilter } P \text{ n nell}) k) - n) \text{ nell}))$
using *assms nfilter-nkfilter-ndropn-nidx-b-2*
by (*metis in-nset-conv-nnth nkfilter-holds nkfilter-nlength*)

lemma *ntaken-nkfilter-ntaken-nset-eq*:

assumes $P (\text{nnth nell } ((\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k)))$
 $k \leq \text{nlength} (\text{nfilter } P \text{ nell})$

shows $\text{nset}(\text{ntaken } k (\text{nkfilter } P \text{ 0 nell})) =$
 $\text{nset}(\text{nkfilter } P \text{ 0 } (\text{ntaken } ((\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k)) \text{ nell}))$

proof –

have 1: $((\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k)) \leq \text{nlength nell}$

using *assms nkfilter-upperbound[of nell P k 0]*

by (*metis diff-zero gen-nlength-def nfilter-nkfilter-ntaken-nlength-0 nkfilter-nlength nlength-code*)

have 2: $\exists x \in \text{nset nell}. P x$

using *assms(1) exists-nnth-nset* **by** *blast*

have 3: $\{i. i \leq \text{nlength}(\text{ntaken } (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k) \text{ nell}) \wedge$
 $P (\text{nnth} (\text{ntaken } (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k) \text{ nell}) i) \}$
 $= \{i. i \leq (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k) \wedge P (\text{nnth nell } i)\}$

by (*auto simp add: ntaken-nnth 1 order-subst2*)

have 4: $\exists x \in \text{nset}(\text{ntaken } (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k) \text{ nell}). P x$

using 1 *assms(1) ntaken-nset* **by** *fastforce*

have 5: $\{i. i \leq (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k) \wedge P (\text{nnth nell } i)\} =$
 $\{i. i \leq (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k) \wedge i \in \text{nset}(\text{nkfilter } P \text{ 0 nell})\}$

using 4 1 2 *nkfilter-holds-b*

by (*metis (mono-tags, hide-lams) add-cancel-left-right enat-ord-simps(1) order.trans*)

have 6: $\{i. i \leq (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k) \wedge i \in \text{nset}(\text{nkfilter } P \text{ 0 nell})\} =$
 $\{i. i \leq (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k) \wedge$
 $i \in \{ (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) j) \mid j. j \leq \text{nlength} (\text{nkfilter } P \text{ 0 nell}) \}\}$

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

have 7: $\{i. i \leq (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k) \wedge$
 $i \in \{ (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) j) \mid j. j \leq \text{nlength} (\text{nkfilter } P \text{ 0 nell}) \}\} =$
 $\{(\text{nnth} (\text{nkfilter } P \text{ 0 nell}) j) \mid j. j \leq k\}$

by (*auto simp add: assms 2 nidx-nkfilter-less-eq nkfilter-nlength*)

(*metis 2 dual-order.antisym le-cases nidx-nkfilter-gr-eq nkfilter-nlength,*

metis assms(2) dual-order.trans enat-ord-simps(1))

have 8: $k \leq \text{nlength} (\text{nkfilter } P \text{ 0 nell})$

by (*simp add: 2 assms(2) nkfilter-nlength*)

have 9: $\{(\text{nnth} (\text{nkfilter } P \text{ 0 nell}) j) \mid j. j \leq k\} = \text{nset}(\text{ntaken } k (\text{nkfilter } P \text{ 0 nell}))$

using *ntaken-nset[of k (nkfilter P 0 nell)]* 8 **by** *auto*

have 91: $\text{min} (\text{enat } (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k)) (\text{nlength nell}) = (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k)$

by (*simp add: 1 min-def*)

have 10: $\text{nset}(\text{nkfilter } P \text{ 0 } (\text{ntaken } ((\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k)) \text{ nell})) =$
 $\{i. i \leq (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k) \wedge$
 $P (\text{nnth} (\text{ntaken } (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k) \text{ nell}) i) \}$

using *nkfilter-nset[of (ntaken (nnth (nkfilter P 0 nell) k) nell) P 0]*

1 4 91 *ntaken-nlength[of (nnth (nkfilter P 0 nell) k) nell]*

by *auto*

have 11: $\text{nlength}((\text{ntaken } ((\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k)) \text{ nell})) = (\text{nnth} (\text{nkfilter } P \text{ 0 nell}) k)$

by (*simp add: 1 min.absorb1*)

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 nell k]*
nkfilter-holds[of nell P (nnth (nkfilter P 0 nell) k) 0]
by (*metis in-nset-conv-nnth nkfilter-nlength nkfilter-nnth-n-zero*)

lemma *ndropn-nkfilter-ndropn-nset-eq:*

assumes $P \ (\text{nnth } \text{nell} \ (\ (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)) \)$
 $k \leq \text{nlength } (\text{nfilter } P \ \text{nell})$
shows $\text{nset}(\text{ndropn } k \ (\text{nkfilter } P \ 0 \ \text{nell})) =$
 $\text{nset}(\text{nkfilter } P \ (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ (\text{ndropn } ((\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)) \ \text{nell}))$

proof –

have 1: $(\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \leq \text{nlength } \text{nell}$
using *nkfilter-upperbound[of nell P k 0] assms*
by (*metis diff-zero gen-nlength-def nfilter-nkfilter-ntaken-nlength-0 nkfilter-nlength nlength-code*)
have 2: $\exists x \in \text{nset } \text{nell}. P \ x$
using *assms(1) exists-nnth-nset by blast*
have 4: $\exists x \in \text{nset}(\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \ \text{nell}). P \ x$
using 1 *assms(1) ndropn-nset by fastforce*
have 10: $\text{nset}(\text{nkfilter } P \ (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)$
 $(\text{ndropn } ((\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)) \ \text{nell})) =$
 $\{(\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) + i \mid i. i \leq \text{nlength } \text{nell} - (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \wedge$
 $P \ (\text{nnth } \text{nell} \ (i + (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k))) \}$
using 1
nkfilter-ndropn-nset-b[of (nnth (nkfilter P 0 nell) k) nell P (nnth (nkfilter P 0 nell) k)]
4 by *linarith*
have 5: $\{(\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) + i \mid i. i \leq \text{nlength } \text{nell} - (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \wedge$
 $P \ (\text{nnth } \text{nell} \ (i + (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k))) \} =$
 $\{(\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) + i \mid i. i \leq \text{nlength } \text{nell} - (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k) \wedge$
 $(i + (\text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k)) \in \text{nset}(\text{nkfilter } P \ 0 \ \text{nell})\}$
proof (*auto simp add: add.commute*)
fix $i :: \text{nat}$
assume *a1: enat i ≤ nlength nell – enat (nnth (nkfilter P 0 nell) k)*
assume *a2: P (nnth nell (i + nnth (nkfilter P 0 nell) k))*
have *f0: enat (i + (nnth (nkfilter P 0 nell) k)) ≤ nlength nell*
using 1 *a1 enat-min-eq by auto*
show $i + \text{nnth } (\text{nkfilter } P \ 0 \ \text{nell}) \ k \in \text{nset } (\text{nkfilter } P \ 0 \ \text{nell})$
by (*metis Nat.add-0-right a2 f0 in-nset-conv-nnth nkfilter-holds-b*)

```

next
fix i
assume b0: enat i ≤ nlength nell - enat (nnth (nkfilter P 0 nell) k)
assume b1: i + nnth (nkfilter P 0 nell) k ∈ nset (nkfilter P 0 nell)
show P (nnth nell (i + nnth (nkfilter P 0 nell) k))
using nkfilter-holds[of nell P ] 2 by (metis b1 minus-nat.diff-0)
qed
have 51: {(nnth (nkfilter P 0 nell) k) + i | i. i ≤ nlength nell - (nnth (nkfilter P 0 nell) k) ∧
  (i + (nnth (nkfilter P 0 nell) k)) ∈ nset(nkfilter P 0 nell)} =
  {(nnth (nkfilter P 0 nell) k) + i | i.
    (nnth (nkfilter P 0 nell) k) ≤ i + (nnth (nkfilter P 0 nell) k) ∧
    i + (nnth (nkfilter P 0 nell) k) ≤ nlength nell ∧
    (i + (nnth (nkfilter P 0 nell) k)) ∈ nset(nkfilter P 0 nell)}
by auto
  (metis 2 add.left-neutral in-nset-conv-nnth nkfilter-upperbound zero-enat-def,
    metis add-diff-cancel-right' enat-minus-mono1 idiff-enat-enat)
have 52: {(nnth (nkfilter P 0 nell) k) + i | i.
  (nnth (nkfilter P 0 nell) k) ≤ i + (nnth (nkfilter P 0 nell) k) ∧
  i + (nnth (nkfilter P 0 nell) k) ≤ nlength nell ∧
  (i + (nnth (nkfilter P 0 nell) k)) ∈ nset(nkfilter P 0 nell)} =
  {j. (nnth (nkfilter P 0 nell) k) ≤ j ∧ j ≤ nlength nell ∧ j ∈ nset(nkfilter P 0 nell)}
by (metis (no-types, lifting) add.commute diff-add)
have 53 : {j. (nnth (nkfilter P 0 nell) k) ≤ j ∧ j ≤ nlength nell ∧ j ∈ nset(nkfilter P 0 nell)} =
  {j. (nnth (nkfilter P 0 nell) k) ≤ j ∧ j ≤ nlength nell ∧ j ∈
    {(nnth (nkfilter P 0 nell) jj) | jj. jj ≤ nlength (nkfilter P 0 nell)}}
by (simp add: nset-conv-nnth)
have 54: {j. (nnth (nkfilter P 0 nell) k) ≤ j ∧ j ≤ nlength nell ∧ j ∈
  {(nnth (nkfilter P 0 nell) jj) | jj. jj ≤ nlength (nkfilter P 0 nell)}} =
  {(nnth (nkfilter P 0 nell) j) | j. k ≤ j ∧ j ≤ nlength(nkfilter P 0 nell)}
using assms 2
by (auto,
  metis dual-order.antisym le-cases nidx-less-eq nidx-nkfilter nkfilter-nlength,
  meson nidx-nkfilter-gr-eq,
  metis gen-nlength-def nkfilter-upperbound nlength-code)
have 8: k ≤ nlength (nkfilter P 0 nell)
by (simp add: 2 assms(2) nkfilter-nlength)
have 9: {(nnth (nkfilter P 0 nell) j) | j. k ≤ j ∧ j ≤ nlength(nkfilter P 0 nell)} =
  nset(ndropn k (nkfilter P 0 nell))
by (simp add: 8 ndropn-nset)
show ?thesis
using 10 5 51 52 53 54 9 by auto
qed

```

lemma *ndropn-nkfilter-ndropn-nset-eq-1:*

```

assumes ∃ x ∈ nset nell. P x
  k ≤ nlength (nkfilter P nell)
shows nset(ndropn k (nkfilter P 0 nell)) =
  nset(nkfilter P (nnth (nkfilter P 0 nell) k) (ndropn ((nnth (nkfilter P 0 nell) k)) nell))
using assms ndropn-nkfilter-ndropn-nset-eq[of P nell k]
by (metis diff-zero in-nset-conv-nnth nkfilter-holds nkfilter-nlength)

```

lemma *nkfilter-nkfilter-ntaken*:
assumes $\exists x \in \text{nset } \text{nell}. P x$
 $k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
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: nkfilter-nkfilter-ntaken-nidx-a nkfilter-nkfilter-ntaken-nidx-b-1 nidx-nset-eq*
ntaken-nkfilter-ntaken-nset-eq-1)

lemma *nkfilter-nkfilter-ndropn*:
assumes $\exists x \in \text{nset } \text{nell}. P x$
 $k \leq \text{nlength } (\text{nfilter } P \text{ nell})$
shows $\text{ndropn } k (\text{nkfilter } P 0 \text{ nell}) =$
 $(\text{nkfilter } P (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) (\text{ndropn } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \text{ nell}))$
proof –
have 1: $P (\text{nnth } \text{nell } ((\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k)))$
using *nkfilter-holds[of nell P] assms*
by (*metis diff-zero in-nset-conv-nnth nkfilter-nlength*)
show *?thesis*
by (*metis 1 assms(1) assms(2) diff-zero ndropn-nkfilter-ndropn-nset-eq*
nkfilter-nkfilter-ndropn-nidx-a nkfilter-nkfilter-ndropn-nidx-b-3 nidx-nset-eq)
qed

lemma *nkfilter-nmap-nfilter*:
assumes $\exists x \in \text{nset } \text{nell}. P x$
shows $\text{nmap } (\lambda n. \text{nnth } \text{nell } n) (\text{nkfilter } P 0 \text{ nell}) = \text{nfilter } P \text{ nell}$
using *assms nellist-eq-nnth-eq[of nmap (\lambda n. 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 $P (\text{nnth } \text{nell } ((\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k)))$
 $k \leq \text{nlength } (\text{nkfilter } P 0 \text{ nell})$
shows $\text{ntaken } k (\text{nfilter } P \text{ nell}) =$
 $\text{nfilter } P (\text{ntaken } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \text{ nell})$
proof –
have 1: $\exists x \in \text{nset } \text{nell}. P x$
using *assms(1) exists-nnth-nset by blast*
have 2: $\text{nfilter } P \text{ nell} = \text{nmap } (\lambda n. \text{nnth } \text{nell } n) (\text{nkfilter } P 0 \text{ nell})$
using 1 *assms by (simp add: nkfilter-nmap-nfilter)*
have 3: $\text{ntaken } k (\text{nfilter } P \text{ nell}) = \text{ntaken } k (\text{nmap } (\lambda n. \text{nnth } \text{nell } n) (\text{nkfilter } P 0 \text{ nell}))$
using 2 *by simp*
have 4: $\text{ntaken } k (\text{nmap } (\lambda n. \text{nnth } \text{nell } n) (\text{nkfilter } P 0 \text{ nell})) =$
 $\text{nmap } (\lambda n. \text{nnth } \text{nell } n) (\text{ntaken } k (\text{nkfilter } P 0 \text{ nell}))$
by *simp*
have 5: $\exists x \in \text{nset } (\text{ntaken } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \text{ nell}). P x$
using *assms by (metis nfinite-ntaken nset-nlast ntaken-nlast)*
have 6: $(\text{nfilter } P (\text{ntaken } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \text{ nell})) =$
 $\text{nmap } (\lambda s. \text{nnth } (\text{ntaken } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) k) \text{ nell}) s)$

```

      (nkfilter P 0 (ntaken (nnth (nkfilter P 0 nell) k) nell))
    using 5 by (simp add: nkfilter-nmap-nfilter)
  have 7: nmap (λn. nnth nell n) (ntaken k (nkfilter P 0 nell)) =
    nmap (λn. nnth nell n) (nkfilter P 0 (ntaken (nnth (nkfilter P 0 nell) k) nell))
    by (simp add: 1 2 assms(2) nkfilter-nkfilter-ntaken)
  have 70: enat k ≤ nlength (nfilter P nell)
    using 2 assms by auto
  have 71: (Λz. z ∈ nset (nkfilter P 0 (ntaken (nnth (nkfilter P 0 nell) k) nell)) ⇒
    (λn. nnth nell n) z = (λs. nnth (ntaken (nnth (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: enat k ≤ nlength (nkfilter P 0 nell)
    assume a2: Λk j. [k ≤ j; enat j ≤ nlength (nkfilter P 0 nell)] ⇒
      nnth (nkfilter P 0 nell) k ≤ nnth (nkfilter P 0 nell) j
    assume a3: z ∈ nset (nkfilter P 0 (ntaken (nnth (nkfilter P 0 nell) k) nell))
    assume {nnth (nkfilter P 0 nell) i | i. i ≤ k} =
      nset (nkfilter P 0 (ntaken (nnth (nkfilter P 0 nell) k) nell))
    then have ∃ n. z = nnth (nkfilter P 0 nell) n ∧ n ≤ k
    using a3 by blast
    then have z ≤ nnth (nkfilter P 0 nell) k
    using a2 a1 by meson
    then show nnth nell z = nnth (ntaken (nnth (nkfilter P 0 nell) k) nell) z
    by (simp add: ntaken-nnth)
  qed
  have 8: nmap (λn. nnth nell n)
    (nkfilter P 0 (ntaken (nnth (nkfilter P 0 nell) k) nell)) =
    nmap (λs. 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 ∃ x ∈ nset nell. P x
      k ≤ 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 ∃ x ∈ nset nell. P x
  shows   nmap (λs. nnth nell (s+n)) (nkfilter P 0 nell) =
      nmap (λs. 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*

$nellist-eq-nnth-eq[of\ nmap\ (\lambda s. nnth\ nell\ (s+(nnth\ (nkfilter\ P\ 0\ nell)\ k)))$
 $(nkfilter\ P\ 0\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell))$
 $nmap\ (\lambda s. nnth\ nell\ s)\ (nkfilter\ P\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)$
 $(ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell))]$

using *nkfilter-n-zero[of (ndropn (nnth (nkfilter P 0 nell) k) nell) P*
 $(nnth\ (nkfilter\ P\ 0\ nell)\ k)]$

by *simp*

lemma *nfilter-nkfilter-ndropn:*

assumes $P\ (nnth\ nell\ ((nnth\ (nkfilter\ P\ 0\ nell)\ k))\)$
 $k \leq nlength\ (nkfilter\ P\ 0\ nell)$
shows $ndropn\ k\ (nfilter\ P\ nell) =$
 $nfilter\ P\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)$

proof —

have 1: $\exists x \in nset\ nell. P\ x$
using *assms* **using** *exists-nnth-nset* **by** *blast*
have 2: $(nfilter\ P\ nell) = nmap\ (\lambda n. nnth\ nell\ n)\ (nkfilter\ P\ 0\ nell)$
by (*simp add: 1 nkfilter-nmap-nfilter*)
have 3: $ndropn\ k\ (nfilter\ P\ nell) = ndropn\ k\ (nmap\ (\lambda n. nnth\ nell\ n)\ (nkfilter\ P\ 0\ nell))$
by (*simp add: 2*)
have 4: $ndropn\ k\ (nmap\ (\lambda n. nnth\ nell\ n)\ (nkfilter\ P\ 0\ nell)) =$
 $nmap\ (\lambda n. nnth\ nell\ n)\ (ndropn\ k\ (nkfilter\ P\ 0\ nell))$
using *ndropn-nmap* **by** *blast*
have 5: $\exists x \in nset\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell). P\ x$

```

  by (metis add.commute add.left-neutral assms(1) in-nset-conv-nnth ndropn-nnth zero-enat-def
    zero-le)
have 6:  $nfilter\ P\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell) =$ 
   $nmap\ (\lambda s. nnth\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)\ s)$ 
   $(nkfilter\ P\ 0\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell))$ 
  by (metis 5 nkfilter-nmap-nfilter)
have 7:  $nmap\ (\lambda s. nnth\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)\ s)$ 
   $(nkfilter\ P\ 0\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)) =$ 
   $nmap\ (\lambda s. nnth\ nell\ (s+(nnth\ (nkfilter\ P\ 0\ nell)\ k)))$ 
   $(nkfilter\ P\ 0\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell))$ 
  by (simp add: add.commute)
have 8:  $nmap\ (\lambda s. nnth\ nell\ (s+(nnth\ (nkfilter\ P\ 0\ nell)\ k)))$ 
   $(nkfilter\ P\ 0\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)) =$ 
   $nmap\ (\lambda s. nnth\ nell\ s)$ 
   $(nkfilter\ P\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell))$ 
  using 5 assms(2) nkfilter-nmap-shift-ndropn by fastforce
have 9:  $nmap\ (\lambda s. nnth\ nell\ s)$ 
   $(nkfilter\ P\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)) =$ 
   $nmap\ (\lambda s. nnth\ nell\ s)$ 
   $(ndropn\ k\ (nkfilter\ P\ 0\ nell))$ 
  by (simp add: 1 2 assms(2) nkfilter-nkfilter-ndropn)
show ?thesis using 3 4 6 7 8 9 by auto
qed

```

```

lemma nfilter-nkfilter-ndropn-1:
assumes  $\exists x \in nset\ nell. P\ x$ 
   $k \leq nlength\ (nkfilter\ P\ 0\ nell)$ 
shows  $ndropn\ k\ (nfilter\ P\ nell) =$ 
   $nfilter\ P\ (ndropn\ (nnth\ (nkfilter\ P\ 0\ nell)\ k)\ nell)$ 
using assms nfilter-nkfilter-ndropn
by (metis in-nset-conv-nnth minus-nat.diff-0 nkfilter-holds)

```

```

lemma nkfilter-nnth-aa:
assumes  $\exists x \in nset\ nell. P\ x$ 
   $n \leq nlength\ (nfilter\ P\ nell)$ 
shows  $P\ (nnth\ (nfilter\ P\ nell)\ n)$ 
using assms in-nset-conv-nnth nkfilter-holds[of nell P - 0] nkfilter-nfilter[of nell P - 0]
by (metis nkfilter-nlength)

```

```

lemma nfilter-nlength-zero-conv-a:
assumes  $\exists x \in nset\ nell. P\ x$ 
   $nlength(nfilter\ P\ nell) = 0$ 
shows  $(\exists k \leq nlength\ nell. P\ (nnth\ nell\ k) \wedge$ 
   $(\forall j \leq nlength\ nell. j \neq k \longrightarrow \neg P\ (nnth\ nell\ j)))$ 
using assms
apply transfer
proof (auto simp add: epred-llength min-def split: if-split-asm)
  fix nella :: 'a llist and Pa :: 'a  $\Rightarrow$  bool and x :: 'a
  assume a1:  $\neg lnull\ nella$ 
  assume a2:  $Pa\ x$ 

```

assume $a3: x \in \text{lset } nella$
assume $a4: \text{lnull } (\text{lfilter } Pa \text{ } nella)$
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 $(\text{metis } a2 \ a3 \ \text{lfilter-LNil} \ \text{llist.disc}(1) \ \text{llist.exhaust-sel} \ \text{lnull-lfilter})$
have $2: \text{llength } nella = \text{eSuc } (\text{llength } (\text{ltl } nella))$
by $(\text{metis } 1 \ \text{llength-LCons})$
have $3: \neg \text{lnull } (\text{lfilter } Pa \text{ } nella)$
by $(\text{meson } a2 \ a3 \ \text{lnull-lfilter})$
show $?thesis$
by $(\text{metis } 2 \ 3 \ a4 \ \text{iless-Suc-eq} \ \text{lfilter-llength-one-conv-a} \ \text{lhs-LCons-ltl} \ \text{llength-LCons} \ \text{llength-LNil}$
 $\text{llist.collapse}(1) \ \text{one-eSuc})$
qed
qed

lemma $nfilter-nlength-zero-conv-c:$

$(\exists k \leq nlength \text{ nell. } P (\text{nnth } \text{nell } k) \wedge$
 $(\forall j \leq nlength \text{ nell. } j \neq k \longrightarrow \neg P (\text{nnth } \text{nell } j))) =$
 $(\exists k \leq nlength \text{ nell. } P (\text{nnth } \text{nell } k) \wedge$
 $(\forall j \leq nlength \text{ nell. } j < k \vee k < j \longrightarrow \neg P (\text{nnth } \text{nell } j)))$

using antisym-conv3 **by** auto

lemma $nfilter-nlength-zero-conv-d:$

$(\exists k \leq nlength \text{ nell. } P (\text{nnth } \text{nell } k) \wedge$
 $(\forall j \leq nlength \text{ nell. } j < k \vee k < j \longrightarrow \neg P (\text{nnth } \text{nell } j))) \longleftrightarrow$
 $(\exists k \leq nlength \text{ nell. } P (\text{nnth } \text{nell } k) \wedge$
 $(\forall j. j < k \longrightarrow \neg P (\text{nnth } \text{nell } j)) \wedge$
 $(\forall j \leq nlength \text{ nell. } k < j \longrightarrow \neg P (\text{nnth } \text{nell } j)))$

by $(\text{meson } \text{enat-ord-simps}(2) \ \text{le-cases} \ \text{le-less-trans})$

lemma $nfilter-nlength-zero-conv-b:$

assumes $(\exists k \leq nlength \text{ nell. } P (\text{nnth } \text{nell } k) \wedge$
 $(\forall j \leq nlength \text{ nell. } j \neq k \longrightarrow \neg P (\text{nnth } \text{nell } j)))$
shows $(\exists x \in \text{nset } \text{nell. } P \ x) \wedge nlength(nfilter \ P \ \text{nell}) = 0$

using assms

by transfer

$(\text{auto split: if-split-asm,}$
 $\text{metis co.enat.exhaust-sel} \ \text{iless-Suc-eq} \ \text{in-lset-conv-lnth} \ \text{llength-eq-0} \ \text{min.orderE} \ \text{the-enat.simps,}$
 $\text{metis co.enat.exhaust-sel} \ \text{co.enat.sel}(2) \ \text{iless-Suc-eq} \ \text{lfilter-llength-one-conv-b} \ \text{llength-eq-0}$
 $\text{min.orderE} \ \text{one-eSuc} \ \text{the-enat.simps})$

lemma $nfilter-nlength-zero-conv:$

$((\exists x \in \text{nset } \text{nell. } P \ x) \wedge nlength(nfilter \ P \ \text{nell}) = 0) \longleftrightarrow$
 $(\exists k \leq nlength \text{ nell. } P (\text{nnth } \text{nell } k) \wedge$
 $(\forall j \leq nlength \text{ nell. } j \neq k \longrightarrow \neg P (\text{nnth } \text{nell } j)))$

using $nfilter-nlength-zero-conv-a[\text{of } \text{nell } P] \ nfilter-nlength-zero-conv-b[\text{of } \text{nell } P]$

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.prem1(2) Suc-le-D ndropn-Suc-NCons ndropn-is-NNil*)

have 2: $P x21 \wedge \neg \text{is-NNil } x22 \implies$

$\text{nfilter } P (\text{ndropn } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) 0) \text{ nell}) = \text{nfilter } P (\text{ndropn } (\text{Suc } j) \text{ nell})$

using *Suc NCons*

by *simp*

(*metis Suc.prem1(1) dual-order.strict-trans1 nkfilter-not-before nnth-0 zero-less-Suc*)

have 3: $\neg P x21 \wedge \neg \text{is-NNil } x22 \implies$

$\text{nfilter } P (\text{ndropn } (\text{nnth } (\text{nkfilter } P 0 \text{ nell}) 0) \text{ nell}) = \text{nfilter } P (\text{ndropn } (\text{Suc } j) \text{ nell})$

using *Suc NCons by (simp add: nkfilter-nleast)*

show ?thesis

using 1 2 3 **by** *blast*

qed


```

    qed
qed

lemma nfilter-nappend-ntaken:
  assumes  $\exists x \in \text{nset } (\text{ntaken } k \text{ nell}). P x$ 
     $k \leq \text{nlength } \text{nell}$ 
  shows  $\text{nfilter } P (\text{ntaken } k \text{ nell}) =$ 
     $\text{ntaken } (\text{the-enat } (\text{nlength}(\text{nfilter } P (\text{ntaken } k \text{ nell})))) (\text{nfilter } P \text{ nell})$ 
using assms
apply transfer
proof auto
  show  $\bigwedge k \text{ nell } P x.$ 
     $\neg \text{lnull } \text{nell} \implies$ 
     $\text{enat } k \leq \text{epred } (\text{llength } \text{nell}) \implies$ 
     $x \in \text{lset } (\text{ltake } (\text{enat } (\text{Suc } k)) \text{ nell}) \implies$ 
     $P x \implies$ 
     $\forall x \in \text{lset } \text{nell}. \neg P x \implies$ 
     $\text{lfilter } P (\text{ltake } (\text{enat } (\text{Suc } k)) \text{ nell}) =$ 
     $\text{ltake } (\text{enat } (\text{Suc } (\text{the-enat } (\text{epred } (\text{llength } (\text{lfilter } P (\text{ltake } (\text{enat } (\text{Suc } k)) \text{ nell}))))))) \text{ nell}$ 
  by (metis dual-order.strict-trans1 in-lset-conv-lnth llength-ltake lprefix-lnthD ltake-is-lprefix
    min.cobounded2)
next
  fix k
  fix nell :: 'a llist
  fix P
  fix x
  fix xa
  assume a0:  $\neg \text{lnull } \text{nell}$ 
  assume a1:  $\text{enat } k \leq \text{epred } (\text{llength } \text{nell})$ 
  assume a2:  $x \in \text{lset } (\text{ltake } (\text{enat } (\text{Suc } k)) \text{ nell})$ 
  assume a3:  $P x$ 
  assume a4:  $xa \in \text{lset } \text{nell}$ 
  assume a5:  $P xa$ 
  show  $\text{lfilter } P (\text{ltake } (\text{enat } (\text{Suc } k)) \text{ nell}) =$ 
     $\text{ltake } (\text{enat } (\text{Suc } (\text{the-enat } (\text{epred } (\text{llength } (\text{lfilter } P (\text{ltake } (\text{enat } (\text{Suc } k)) \text{ nell}))))))) (\text{lfilter } P \text{ nell})$ 
  proof (cases  $k = \text{epred } (\text{llength } \text{nell})$ )
  case True
  then show ?thesis
  proof -
    have f1:  $\text{eSuc } (\text{enat } k) = \text{llength } \text{nell}$ 
    by (simp add: True a0)
    then have  $\text{llength } (\text{lfilter } P \text{ nell}) \neq 0$ 
    by (metis (no-types) a2 a3 eSuc-enat llength-eq-0 lnull-lfilter ltake-all order-refl)
    then show ?thesis
    using f1 by (metis True co.enat.exhaust-sel eSuc-enat enat-the-enat epred-le-epredI infinity-ileE
      llength-lfilter-ile ltake-all order-refl)
  qed
next
  case False
  then show ?thesis

```

```

proof –
have  $f1$ :  $l\text{length } nell \neq 0$ 
using  $a0$   $l\text{length-eq-0}$  by  $\text{blast}$ 
have  $g1$ :  $\neg l\text{null } (l\text{filter } P (l\text{take } (enat (Suc k)) nell))$ 
by ( $\text{meson } a2 a3 l\text{null-lfilter}$ )
then show  $?thesis$ 
using  $f1$ 
proof –
have  $f1$ :  $enat k < e\text{pred } (l\text{length } nell)$ 
using  $\text{False } a1 \text{ order.not-eq-order-implies-strict}$  by  $\text{blast}$ 
have  $f2$ :  $\forall e. e = 0 \vee e = e\text{Suc } (e\text{pred } e)$ 
by ( $\text{metis co.enat.exhaust-sel}$ )
then have  $g2$ :  $enat (Suc k) < l\text{length } nell$ 
using  $f1$  by ( $\text{metis (no-types) Suc-ile-eq } (l\text{length } nell \neq 0) \text{ iless-Suc-eq}$ )
then show  $?thesis$ 
using  $f2$ 
using  $l\text{filter-lappend-ltake}[of \text{ Suc } k]$ 
proof –
have  $e\text{Suc } (enat k) = \min (enat (Suc k)) (l\text{length } nell)$ 
by ( $\text{metis } (enat (Suc k) < l\text{length } nell) e\text{Suc-enat min.strict-order-iff}$ )
then have  $e\text{Suc } (enat k) = l\text{length } (l\text{take } (enat (Suc k)) nell)$ 
by  $\text{simp}$ 
then show  $?thesis$ 
by ( $\text{metis } g1 g2 \text{ co.enat.exhaust-sel } e\text{Suc-enat } e\text{Suc-infinity } enat\text{-the-enat } infinity\text{-ileE}$ 
 $l\text{filter-lappend-ltake } l\text{length-eq-0 } l\text{length-lfilter-ile}$ )
qed
qed
qed
qed
qed

```

```

lemma  $n\text{append-ntaken-ndropn}$ :
assumes  $(Suc k) \leq n\text{length } nell$ 
shows  $n\text{append } (ntaken k nell) (ndropn (Suc k) nell) = nell$ 
using  $\text{assms}$ 
by  $\text{transfer } (\text{simp add: min-absorb1})$ 

```

```

lemma  $n\text{append-nfilter-nfinite}$ :
assumes  $\exists x \in n\text{set } (nellx). P x$ 
 $\exists x \in n\text{set } (nelly). P x$ 
 $n\text{finite } nellx$ 
shows  $n\text{filter } P (n\text{append } nellx nelly) = n\text{append } (n\text{filter } P nellx) (n\text{filter } P nelly)$ 
using  $\text{assms}$ 
by  $\text{transfer auto}$ 

```

```

lemma  $n\text{filter-nappend-ndropn}$ :
assumes  $\exists x \in n\text{set } (ndropn (Suc k) nell). P x$ 
 $\exists x \in n\text{set } (ntaken k nell). P x$ 
 $(Suc k) \leq n\text{length } nell$ 
shows  $n\text{filter } P (ndropn (Suc k) nell) =$ 

```

$\text{ndropn } (\text{the-enat } (\text{eSuc}(\text{nlength}(\text{nfilter } P \text{ (ntaken } k \text{ nell)})))) (\text{nfilter } P \text{ nell})$
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: $\text{ndropn } (\text{the-enat } (\text{eSuc}(\text{nlength}(\text{nfilter } P \text{ (ntaken } k \text{ nell)}))))$
 $\quad (\text{nappend } (\text{nfilter } P \text{ (ntaken } k \text{ nell)}) (\text{nfilter } P \text{ (ndropn } (\text{Suc } k) \text{ nell)}))$
 $\quad = (\text{nfilter } P \text{ (ndropn } (\text{Suc } k) \text{ nell)})$
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 $\exists x \in \text{nset } (\text{ntaken } k \text{ nell}). P \ x$
 $k \leq \text{nlength } \text{nell}$
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 $\bigwedge k \text{ nell } P \ n \ x.$
 $\neg \text{lnull } \text{nell} \implies$
 $\text{enat } k \leq \text{epred } (\text{llength } \text{nell}) \implies$
 $x \in \text{lset } (\text{ltake } (\text{enat } (\text{Suc } k)) \text{ nell}) \implies$
 $P \ x \implies$
 $\forall x \in \text{lset } \text{nell}. \neg P \ x \implies$
 $\text{kfilter } P \ n \text{ (ltake } (\text{enat } (\text{Suc } k)) \text{ nell)} =$
 $\text{ltake } (\text{enat } (\text{Suc } (\text{the-enat } (\text{epred } (\text{llength } (\text{kfilter } P \ n \text{ (ltake } (\text{enat } (\text{Suc } k)) \text{ nell})))))) (\text{iterates } \text{Suc } n)$
by (*meson lset-ltake subsetD*)
next
fix $ka :: \text{nat}$ **and** $\text{nella} :: 'a \text{ llist}$ **and** $Pa :: 'a \Rightarrow \text{bool}$ **and** $na :: \text{nat}$ **and** $x :: 'a$ **and** $xa :: 'a$
assume *a1: Pa x*
assume *a2: Pa xa*
assume *a3: xa ∈ lset nella*
assume *a4: x ∈ lset (ltake (enat (Suc ka)) 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: $\text{eSuc } \infty = \infty$*
by *simp*
have *f8: $(\text{enat } (\text{Suc } (\text{the-enat } (\text{epred } (\text{llength } (\text{kfilter } Pa \ na \text{ (ltake } (\text{enat } (\text{Suc } ka)) \text{ nella})))))) =$*
 $(\text{llength } (\text{kfilter } Pa \ na \text{ (ltake } (\text{enat } (\text{Suc } ka)) \text{ nella})))$
proof –

```

have f1:  $\forall n. \neg \infty \leq \text{enat } n$ 
by (meson infinity-ileE)
have  $\neg \text{lnull } (\text{kfilter } Pa \text{ na } (\text{ltake } (\text{enat } (\text{Suc } ka)) \text{ nella}))$ 
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  $\text{ltake } (\text{enat } (\text{Suc } ka)) \text{ nella} = \text{nella} \longrightarrow$ 
     $\text{ltake } (\text{llength } (\text{lfilter } Pa (\text{ltake } (\text{enat } (\text{Suc } ka)) \text{ nella}))) (\text{kfilter } Pa \text{ na } nella) =$ 
     $\text{kfilter } Pa \text{ na } (\text{ltake } (\text{enat } (\text{Suc } ka)) \text{ nella}) \wedge$ 
     $\text{epred } (\text{llength } (\text{lfilter } Pa (\text{ltake } (\text{enat } (\text{Suc } ka)) \text{ nella}))) \neq \infty$ 
  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  $\text{ltake } (\text{enat } (\text{Suc } ka)) \text{ nella} \neq \text{nella}$ 
    then have  $\text{enat } (\text{Suc } ka) < \text{llength } nella \wedge \min (\text{enat } (\text{Suc } ka)) (\text{llength } nella) \neq \infty \vee$ 
       $\text{enat } (\text{Suc } ka) < \text{llength } nella \wedge \text{llength } (\text{lfilter } Pa (\text{ltake } (\text{enat } (\text{Suc } ka)) \text{ nella})) \neq \text{eSuc } \infty$ 
    by (metis (no-types) enat-ord-code(4) ltake-all min.orderE not-le-imp-less not-less-iff-gr-or-eq) }
    ultimately have  $\text{enat } (\text{Suc } ka) < \text{llength } nella \wedge \min (\text{enat } (\text{Suc } ka)) (\text{llength } nella) \neq \infty \vee$ 
       $\text{enat } (\text{Suc } ka) < \text{llength } nella \wedge \text{llength } (\text{lfilter } Pa (\text{ltake } (\text{enat } (\text{Suc } ka)) \text{ nella})) \neq \text{eSuc } \infty \vee$ 
       $\text{ltake } (\text{llength } (\text{lfilter } Pa (\text{ltake } (\text{enat } (\text{Suc } ka)) \text{ nella}))) (\text{kfilter } Pa \text{ na } nella) =$ 
       $\text{kfilter } Pa \text{ na } (\text{ltake } (\text{enat } (\text{Suc } ka)) \text{ nella}) \wedge$ 
       $\text{epred } (\text{llength } (\text{lfilter } Pa (\text{ltake } (\text{enat } (\text{Suc } ka)) \text{ nella}))) \neq \infty$ 
    by fastforce }
    ultimately show  $\text{kfilter } Pa \text{ na } (\text{ltake } (\text{enat } (\text{Suc } ka)) \text{ nella}) =$ 
       $\text{ltake } (\text{enat } (\text{Suc } (\text{the-enat } (\text{epred } (\text{llength } (\text{kfilter } Pa \text{ na } (\text{ltake } (\text{enat } (\text{Suc } ka)) \text{ nella}))))))$ 
       $(\text{kfilter } Pa \text{ na } nella)$ 
    using f6 f5 a4 a1
    kfilter-lappend-ltake[of  $(\text{enat } (\text{Suc } ka)) \text{ nella } Pa \text{ na}$ ]
    by (metis enat-ord-code(4) kfilter-llength)
  }
qed

lemma nfilter-ndropns-nmap-help-1:
  assumes  $\exists x \in \text{nset } nell. P x$ 
     $j \leq \text{nnth}(\text{nkfilter } P \ 0 \ nell) (\text{Suc } 0)$ 
     $\text{nnth } (\text{nkfilter } P \ 0 \ nell) \ 0 < j$ 
     $(\text{Suc } 0) \leq \text{nlength}(\text{nfilter } P \ nell)$ 
  shows  $(\text{nfilter } P (\text{ndropn } (\text{nnth } (\text{nkfilter } P \ 0 \ nell) (\text{Suc } 0)) \ nell)) =$ 
     $(\text{nfilter } P (\text{ndropn } j \ nell))$ 
  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  $\implies$ 
      nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc 0)) nell) =
      nfilter P (ndropn (Suc j) nell)
    by (metis NCons Suc.premis(2) Suc-le-D ndropn-Suc-NCons ndropn-is-NNil)
    have 2: P x21  $\wedge \neg$  is-NNil x22  $\implies$ 
      nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc 0)) nell) =
      nfilter P (ndropn (Suc j) nell)
    using Suc NCons
    proof simp
    assume a1: P x21  $\wedge \neg$  is-NNil x22
    assume a2: enat (Suc 0)  $\leq$  nlength (nfilter P (NCons x21 x22))
    assume a3: nell = NCons x21 x22
    assume a4: Suc j  $\leq$  nnth (nkfilter P 0 (NCons x21 x22)) (Suc 0)
    have f5:  $\forall as\ p\ a\ n. ((\exists a. (a::'a) \in \text{nset } as \wedge p\ a) \vee \neg p\ a) \vee$ 
      nkfilter p n (NCons a as) = NNil n
    by (metis (no-types) nkfilter-NCons-a)
    have f6:  $\forall as\ p\ n. (\forall a. (a::'a) \notin \text{nset } as \vee \neg p\ a) \vee$ 
      nlength (nkfilter p n as) = nlength (nfilter p as)
    by (meson nkfilter-nlength)
    have f7:  $\forall p\ as. (\forall a. (a::'a) \notin \text{nset } as \vee \neg p\ a) = (\forall a. a \notin \text{nset } as \vee \neg p\ a)$ 
    by meson
    have f8: nlength (nkfilter P 0 (NCons x21 x22)) =
      nlength (nfilter P (NCons x21 x22))
    by (meson a1 nellist.set-intros(2) nkfilter-nlength)
    have f9:  $\forall p\ as. (\forall a. (a::'a) \notin \text{nset } as \vee \neg p\ a) = (\forall a. a \notin \text{nset } as \vee \neg p\ a)$ 
    by meson
    have f10:  $(\exists a. a \in \text{nset } x22 \wedge P\ a) \longrightarrow$ 
      nkfilter P 0 (NCons x21 x22) = NCons 0 (nkfilter P (Suc 0) x22)
    using a1 by (simp add: Bex-def-raw)
    then have f11:  $(\exists a. a \in \text{nset } x22 \wedge P\ a) \longrightarrow$ 
      nnth (nkfilter P (Suc 0) x22) 0 – Suc 0 = nnth (nkfilter P 0 x22) 0
    using f9 f8 f7 a2 by (metis Suc-ile-eq iless-Suc-eq nkfilter-nnth-n-zero nlength-NCons)
    have f14: j < nnth (nkfilter P 0 (NCons x21 x22)) (Suc 0)
    using a4 Suc-le-eq by blast
    have  $\exists a. a \in \text{nset } x22 \wedge P\ a$ 
    using f8 f5 a2 a1 by (metis Suc-ile-eq gr-implies-not-zero nlength-NNil)
    then have  $(\exists a. a \in \text{nset } x22 \wedge P\ a) \wedge \text{nfilter } P (\text{ndropn } (\text{nnth } (\text{nkfilter } P\ 0\ x22)\ 0)\ x22) =$ 
      nfilter P (ndropn j x22)
    using f14 f11 a4
  proof –

```

```

have j ≤ nnth (nkfilter P 0 x22) 0
using (∃ a. a ∈ nset x22 ∧ P a) f10 f11 f14 by fastforce
then show ?thesis
by (simp add: Bex-def-raw (∃ a. a ∈ nset x22 ∧ P a) nfilter-ndropns-nmap-help-0)
qed
then show nfilter P (ndropn (nnth (nkfilter P 0 (NCons x21 x22)) (Suc 0)) (NCons x21 x22)) =
      nfilter P (ndropn j x22)
using f11 a4
by (metis One-nat-def Suc-le-D diff-Suc-1 f10 ndropn-Suc-NCons nnth-Suc-NCons)
qed
have 3: ¬P x21 ∧ ¬ is-NNil x22 ⇒
      nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc 0)) nell) =
      nfilter P (ndropn (Suc j) nell)
using Suc NCons
proof simp
  assume a1: ∧nell. [∃ a∈nset nell. P a; j ≤ nnth (nkfilter P 0 nell) (Suc 0);
    nnth (nkfilter P 0 nell) 0 < j; enat (Suc 0) ≤ nlength (nfilter P nell)] ⇒
    nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc 0)) nell) =
    nfilter P (ndropn j nell)
  assume a2: enat (Suc 0) ≤ nlength (nfilter P x22)
  assume a3: Suc j ≤ nnth (nkfilter P (Suc 0) x22) (Suc 0)
  assume a4: ∃ x∈nset x22. P x
  assume a5: nnth (nkfilter P (Suc 0) x22) 0 < Suc j
  have f12: nnth (nkfilter P 0 x22) (Suc 0) =
    ( (nnth (nkfilter P (Suc 0) x22) (Suc 0)) - (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 ≤ nnth (nkfilter P 0 x22) (Suc 0)
  using a3 by linarith
  have f14: enat 0 ≤ nlength (nfilter P x22)
  using a2 by (metis One-nat-def one-enat-def order.trans zero-enat-def zero-le-one)
  have f15: nlength (nkfilter P (Suc 0) x22) = nlength (nfilter P x22)
  by (simp add: a4 nkfilter-nlength)
  then have Suc 0 ≤ nnth (nkfilter P (Suc 0) x22) 0
  by (simp add: a4 f14 nkfilter-lowerbound)
  then have Suc (nnth (nkfilter P 0 x22) 0) ≤ j
  by (metis a4 a5 add-Suc leD nkfilter-nleast not-less-eq-eq)
  then have nfilter P (ndropn (nnth (nkfilter P 0 x22) (Suc 0)) x22) =
    nfilter P (ndropn j x22)
  by (simp add: Suc.IH Suc-le-lessD a2 a4 f13)
  then show nfilter P (ndropn (nnth (nkfilter P (Suc 0) x22) (Suc 0)) (NCons x21 x22)) =
    nfilter P (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  $\exists x \in \text{nset } nell. P x$ 
     $j \leq \text{nnth}(\text{nkfilter } P \ 0 \ nell) \ (\text{Suc } i)$ 
     $\text{nnth}(\text{nkfilter } P \ 0 \ nell) \ i < j$ 
     $(\text{Suc } i) \leq \text{nlength}(\text{nfilter } P \ nell)$ 
  shows  $(\text{nfilter } P \ (\text{ndropn}(\text{nnth}(\text{nkfilter } P \ 0 \ nell) \ (\text{Suc } i)) \ nell)) =$ 
     $(\text{nfilter } P \ (\text{ndropn } j \ nell))$ 
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  $\implies$ 
           $\text{nfilter } P \ (\text{ndropn}(\text{nnth}(\text{nkfilter } P \ 0 \ nell) \ (\text{Suc } i)) \ nell) =$ 
             $\text{nfilter } P \ (\text{ndropn} \ (\text{Suc } j) \ nell)$ 
          by (metis NCons Suc.premis(2) Suc-le-D ndropn-Suc-NCons ndropn-is-NNil)
        have 2:  $\neg \text{is-NNil } x22 \wedge i = 0 \implies$ 
           $\text{nfilter } P \ (\text{ndropn}(\text{nnth}(\text{nkfilter } P \ 0 \ nell) \ (\text{Suc } i)) \ nell) =$ 
             $\text{nfilter } P \ (\text{ndropn} \ (\text{Suc } j) \ nell)$ 
          using Suc nfilter-ndropns-nmap-help-1[of nell P ] by metis
        have 3:  $P \ x21 \wedge \neg \text{is-NNil } x22 \wedge 0 < i \implies$ 
           $\text{nfilter } P \ (\text{ndropn}(\text{nnth}(\text{nkfilter } P \ 0 \ nell) \ (\text{Suc } i)) \ nell) = \text{nfilter } P \ (\text{ndropn} \ (\text{Suc } j) \ nell)$ 
          using Suc NCons
        proof simp
        assume a1:  $\bigwedge nell \ i. [\exists a \in \text{nset } nell. P \ a; j \leq \text{nnth}(\text{nkfilter } P \ 0 \ nell) \ (\text{Suc } i);$ 
           $\text{nnth}(\text{nkfilter } P \ 0 \ nell) \ i < j; \text{enat} \ (\text{Suc } i) \leq \text{nlength} \ (\text{nfilter } P \ nell)] \implies$ 
           $\text{nfilter } P \ (\text{ndropn}(\text{nnth}(\text{nkfilter } P \ 0 \ nell) \ (\text{Suc } i)) \ nell) =$ 
           $\text{nfilter } P \ (\text{ndropn } j \ nell)$ 
        assume a2:  $P \ x21 \wedge \neg \text{is-NNil } x22 \wedge 0 < i$ 
        assume a3:  $\text{enat} \ (\text{Suc } i) \leq \text{nlength} \ (\text{nfilter } P \ (\text{NCons } x21 \ x22))$ 
        assume a4:  $nell = \text{NCons } x21 \ x22$ 
        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: nlength (nkfilter P 0 (NCons x21 x22)) = nlength (nfilter P (NCons x21 x22))
  using a4 by (metis (no-types) Suc(2) nkfilter-nlength)
have f2: nkfilter P 0 (NCons x21 x22) = NCons 0 (nkfilter P (Suc 0) x22)
  by (metis a2 a3 dual-order.antisym enat.inject f1 nat.distinct(1) nkfilter-NCons
    nkfilter-NCons-a nlength-NNil zero-enat-def zero-le)
have f3: nnth (nkfilter P 0 (NCons x21 x22)) (Suc i) =
  nnth ( (nkfilter P (Suc 0) x22)) (i)
  by (simp add: f2)
have f4: enat i ≤ nlength (nkfilter P (Suc 0) x22)
  by (metis Suc-ile-eq a3 f1 f2 iless-Suc-eq nlength-NCons)
have f5: nnth ( (nkfilter P (Suc 0) x22)) (i) = (Suc 0) + nnth ( (nkfilter P 0 x22)) (i)
  using nkfilter-nnth-n-zero[of x22 P i Suc 0 ]
  using a5 f0 f3 f4 by linarith
have f6: ndropn (nnth (nkfilter P 0 (NCons x21 x22)) (Suc i)) (NCons x21 x22) =
  ndropn ((Suc 0) + nnth ( (nkfilter P 0 x22)) (i)) (NCons x21 x22)
  using f3 f5 by presburger
have f7: ndropn ((Suc 0) + nnth ( (nkfilter P 0 x22)) (i)) (NCons x21 x22) =
  ndropn (nnth ( (nkfilter P 0 x22)) (i)) x22
  using ndropn-Suc-NCons by auto
have f8: j ≤ nnth (nkfilter P 0 x22) (i)
  using a5 f3 f5 by linarith
have f11: nnth (nkfilter P 0 (NCons x21 x22)) i = nnth ( (nkfilter P (Suc 0) x22)) (i-1)
  by (metis One-nat-def Suc-pred a2 f2 nnth-Suc-NCons)
have f12: enat (i - 1) ≤ nlength (nkfilter P (Suc 0) x22)
  by (metis One-nat-def Suc-ile-eq Suc-pred a2 f4 less-imp-le)
have f13: (Suc 0) ≤ nnth ( (nkfilter P (Suc 0) x22)) (i-1)
  by (meson f0 f12 nkfilter-lowerbound)
have f14: nnth ( (nkfilter P (Suc 0) x22)) (i-1) = (Suc 0) + nnth ( (nkfilter P 0 x22)) (i-1)
  using nkfilter-nnth-n-zero[of x22 P i-1 Suc 0 ] f12 f0 f13 by (metis le-add-diff-inverse)
have f9: nnth (nkfilter P 0 x22) (i-1) < j
  using a6 f11 f14 by linarith
have f10: i ≤ nlength (nfilter P x22)
  by (metis f0 f4 nkfilter-nlength)
show ?thesis using a1[of x22 i-1]
  by (metis Suc-diff-1 a2 f0 f10 f3 f5 f7 f8 f9)
qed
qed
have 4: ¬P x21 ∧ ¬ is-NNil x22 ∧ 0 < i ⇒
  nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc i)) nell) =
  nfilter P (ndropn (Suc j) nell)
using Suc NCons
proof simp
  assume a0: ¬ P x21 ∧ ¬ is-NNil x22 ∧ 0 < i
  assume a2: ∧nell i. [∃ a∈nset nell. P a; j ≤ nnth (nkfilter P 0 nell) (Suc i);
    nnth (nkfilter P 0 nell) i < j; enat (Suc i) ≤ nlength (nfilter P nell)] ⇒
    nfilter P (ndropn (nnth (nkfilter P 0 nell) (Suc i)) nell) =
    nfilter P (ndropn j nell)
  assume a1: ∃ x∈nset x22. P x
  assume a4: Suc j ≤ nnth (nkfilter P (Suc 0) x22) (Suc i)

```



```

assume a5:  $\text{nnth } (\text{nkfilter } P \text{ (Suc 0) } x22) \text{ } i < \text{Suc } j$ 
assume a3:  $\text{enat } (\text{Suc } i) \leq \text{nlength } (\text{nfilter } P \text{ } x22)$ 
assume a6:  $\text{nell} = \text{NCons } x21 \text{ } x22$ 
show  $\text{nfilter } P \text{ (ndropn } (\text{nnth } (\text{nkfilter } P \text{ (Suc 0) } x22) \text{ (Suc } i)) \text{ (NCons } x21 \text{ } x22))} =$ 
 $\text{nfilter } P \text{ (ndropn } j \text{ } x22)$ 
proof –
  have f1:  $\text{nlength } (\text{nkfilter } P \text{ 0 (NCons } x21 \text{ } x22)) = \text{nlength } (\text{nfilter } P \text{ (NCons } x21 \text{ } x22))$ 
  by (metis NCons Suc.premis(1) nkfilter-nlength)
  have f2:  $\text{nkfilter } P \text{ 0 (NCons } x21 \text{ } x22) = (\text{nkfilter } P \text{ (Suc 0) } x22)$ 
  using NCons Suc.premis(2) a1 a5 by auto
  have f3:  $\text{nnth } (\text{nkfilter } P \text{ 0 (NCons } x21 \text{ } x22)) \text{ (Suc } i) =$ 
 $\text{nnth } (\text{nkfilter } P \text{ (Suc 0) } x22) \text{ (Suc } i)$ 
  by (simp add: f2)
  have f4:  $\text{enat } (\text{Suc } i) \leq \text{nlength } (\text{nkfilter } P \text{ (Suc 0) } x22)$ 
  using NCons Suc.premis(4) f1 f2 by auto
  have f5:  $\text{nnth } ( (\text{nkfilter } P \text{ (Suc 0) } x22)) \text{ (Suc } i) = (\text{Suc } 0) + \text{nnth } ( (\text{nkfilter } P \text{ 0 } x22)) \text{ (Suc } i)$ 
  by (metis One-nat-def Suc-le-D a1 a4 diff-Suc-1 f4 nkfilter-nnth-n-zero plus-1-eq-Suc)
  have f6:  $\text{ndropn } (\text{nnth } (\text{nkfilter } P \text{ (Suc 0) } x22) \text{ (Suc } i)) \text{ (NCons } x21 \text{ } x22) =$ 
 $\text{ndropn } ( (\text{Suc } 0) + \text{nnth } ( (\text{nkfilter } P \text{ 0 } x22)) \text{ (Suc } i)) \text{ (NCons } x21 \text{ } x22)$ 
  by (simp add: f5)
  have f7:  $\text{ndropn } ( (\text{Suc } 0) + \text{nnth } ( (\text{nkfilter } P \text{ 0 } x22)) \text{ (Suc } i)) \text{ (NCons } x21 \text{ } x22) =$ 
 $\text{ndropn } ( \text{nnth } ( (\text{nkfilter } P \text{ 0 } x22)) \text{ (Suc } i)) \text{ (x22)}$ 
  by simp
  have f8:  $j \leq \text{nnth } (\text{nkfilter } P \text{ 0 } x22) \text{ (Suc } i)$ 
  using a4 f5 by force
  have f11:  $\text{nnth } (\text{nkfilter } P \text{ 0 (NCons } x21 \text{ } x22)) \text{ } i = \text{nnth } ( (\text{nkfilter } P \text{ (Suc 0) } x22)) \text{ (} i \text{)}$ 
  by (simp add: f2)
  have f12:  $\text{enat } (i) \leq \text{nlength } (\text{nkfilter } P \text{ (Suc 0) } x22)$ 
  using Suc-ile-eq f4 by auto
  have f13:  $(\text{Suc } 0) \leq \text{nnth } ( (\text{nkfilter } P \text{ (Suc 0) } x22)) \text{ (} i \text{)}$ 
  by (simp add: a1 f12 nkfilter-lowerbound)
  have f14:  $\text{nnth } ( (\text{nkfilter } P \text{ (Suc 0) } x22)) \text{ (} i \text{)} = (\text{Suc } 0) + \text{nnth } ( (\text{nkfilter } P \text{ 0 } x22)) \text{ (} i \text{)}$ 
  by (metis a1 f12 f13 le-add-diff-inverse nkfilter-nnth-n-zero)
  have f9:  $\text{nnth } (\text{nkfilter } P \text{ 0 } x22) \text{ (} i \text{)} < j$ 
  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 $\exists x \in \text{nset } (\text{ndropns nell}). P \text{ } x$

$(\text{Suc } i) \leq \text{nlength}(\text{nfilter } P \text{ (ndropns nell)})$

shows $\text{ndropn } (\text{Suc } i) \text{ (nmap } (\lambda s. \text{nnth } s \text{ 0}) \text{ (nfilter } P \text{ (ndropns nell)))) =$
 $(\text{nmap } (\lambda s. \text{nnth } s \text{ 0})$

(*nfilter* *P* (*ndropns* (*ndropn* (*Suc* (*nnth* (*nkfilter* *P* 0 (*ndropns nell*)) *i*)) *nell*))))

proof –

have 1: *ndropn* (*Suc i*) (*nmap* ($\lambda s. \text{nnth } s \ 0$) (*nfilter* *P* (*ndropns nell*))) =
nmap ($\lambda s. \text{nnth } s \ 0$) (*ndropn* (*Suc i*) (*nfilter* *P* (*ndropns nell*)))
by (*simp add: ndropn-nmap*)

have 2: (*Suc* (*nnth* (*nkfilter* *P* 0 (*ndropns nell*)) *i*)) \leq *nlength* (*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: (*nfilter* *P* (*ndropns* (*ndropn* (*Suc* (*nnth* (*nkfilter* *P* 0 (*ndropns nell*)) *i*)) *nell*))) =
(*nfilter* *P* (*ndropn* (*Suc* (*nnth* (*nkfilter* *P* 0 (*ndropns nell*)) *i*)) (*ndropns nell*)))
by (*simp add: 2 ndropn-ndropns*)

have 4: (*ndropn* (*Suc i*) (*nfilter* *P* (*ndropns nell*))) =
(*nfilter* *P* (*ndropn* (*nnth* (*nkfilter* *P* 0 (*ndropns nell*)) (*Suc i*)) (*ndropns nell*)))
by (*simp add: assms(1) assms(2) nfilter-nkfilter-ndropn-1 nkfilter-nlength*)

have 5: (*Suc* (*nnth* (*nkfilter* *P* 0 (*ndropns nell*)) *i*)) \leq (*nnth* (*nkfilter* *P* 0 (*ndropns nell*)) (*Suc i*))
by (*simp add: Suc-leI assms(1) assms(2) nidx-nkfilter-expand nkfilter-nlength*)

have 6: *i* < *nlength*(*nfilter* *P* (*ndropns nell*))
using *Suc-ile-eq assms(2)* **by** *blast*

have 7: *nnth* (*nkfilter* *P* 0 (*ndropns nell*)) *i* < (*Suc* (*nnth* (*nkfilter* *P* 0 (*ndropns nell*)) *i*))
by *simp*

have 8: (*nfilter* *P* (*ndropn* (*nnth* (*nkfilter* *P* 0 (*ndropns nell*)) (*Suc i*)) (*ndropns nell*))) =
(*nfilter* *P* (*ndropn* (*Suc* (*nnth* (*nkfilter* *P* 0 (*ndropns nell*)) *i*)) (*ndropns nell*)))
using 5 6 7 *assms*
nfilter-ndropns-nmap-help-j[of ndropns nell P (Suc (nnth (nkfilter P 0 (ndropns nell)) i)) i]
by *blast*

show ?thesis
using 1 3 4 8 **by** *auto*

qed

lemma *ndropns-nfilter-nnth-exist-ndropn*:

assumes $\exists x \in \text{nset } (\text{ndropns } \text{nell}). P \ x$
 $j \leq \text{nlength } (\text{nfilter } P \ (\text{ndropns } \text{nell}))$
shows $(\exists i. \text{enat } i \leq \text{nlength } \text{nell} \wedge \text{nnth } (\text{nfilter } P \ (\text{ndropns } \text{nell})) \ j = \text{ndropn } i \ \text{nell})$

proof –

have 1: *nnth* (*nfilter* *P* (*ndropns nell*)) *j* \in *nset* (*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 $(\exists \text{ys} \in \text{nset } (\text{ndropns } \text{xs}). P \ \text{ys})$
 $j \leq \text{nlength } (\text{nfilter } P \ (\text{ndropns } \text{xs}))$
shows $\text{nlength } ((\text{nnth } (\text{nfilter } P \ (\text{ndropns } \text{xs})) \ j)) \leq \text{nlength } \text{xs}$
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*
 $\exists x \in \text{nset } (\text{ndropns } (\text{ndropn } (\text{Suc } k) \ \text{nell})). P \ x$

$\exists x \in \text{nset } (\text{ntaken } k \text{ (ndropns nell)}). P x$
shows $(\text{nfilter } P \text{ (ndropns (ndropn (Suc } k \text{) nell))) =}$
 $(\text{ndropn (the-enat (eSuc(nlength(nfilter } P \text{ (ntaken } k \text{ (ndropns nell)))))) (nfilter } P \text{ (ndropns nell)))}$
using *assms*
 $\text{ndropn-ndropns[of Suc } k \text{ nell] ndropns-nlength[of nell] nfilter-nappend-ndropn[of } k \text{ ndropns nell } P]$
by *simp*

lemma *ndropns-nfilter-ndropn-a:*

assumes $k \leq \text{nlength}(\text{nfilter } P \text{ (ndropns nell)})$
 $\exists x \in \text{nset } (\text{ndropns nell}). P x$
shows $\text{ndropn } k \text{ (nfilter } P \text{ (ndropns nell)) =}$
 $\text{nfilter } P \text{ (ndropns (ndropn (nnth (nkfilter } P \text{ } 0 \text{ (ndropns nell)) } k \text{) nell))}$
using *assms ndropn-ndropns nfilter-nkfilter-ndropn-1[of ndropns nell } P k]*
nkfilter-upperbound[of ndropns nell } P k 0]
by (*metis (mono-tags, lifting) add.left-neutral nkfilter-nlength zero-enat-def*)

lemma *nfilter-nlength-imp:*

assumes $\exists x \in \text{nset } xs. P x \wedge Q x$
shows $\text{nlength } (\text{nfilter } (\lambda x. P x \wedge Q x) xs) \leq \text{nlength } (\text{nfilter } P xs)$
using *assms* **by** *transfer (auto simp add: lfilter-nlength-imp epred-le-epredI)*

lemma *nkfilter-chop:*

assumes $\text{nlast } xs = \text{nfirst } ys$
 $P (\text{nlast } xs)$
 $\text{nfinite } xs$
shows $\text{nkfilter } P k \text{ (nfuse } xs \text{ } ys) =$
 $\text{nfuse } (\text{nkfilter } P k xs) \text{ (nkfilter } P (k + (\text{the-enat } (\text{nlength } xs))) ys)$
using *assms*
proof (*auto simp add: nfuse-def*)
show $\text{nlast } xs = \text{nfirst } ys \implies$
 $P (\text{nfirst } ys) \implies$
 $\text{nfinite } xs \implies$
 $\text{is-NNil } ys \implies$
 $\neg \text{is-NNil } (\text{nkfilter } P (k + \text{the-enat } (\text{nlength } xs)) ys) \implies$
 $\text{nkfilter } P k xs = \text{nappend } (\text{nkfilter } P k xs) (\text{ntl } (\text{nkfilter } P (k + \text{the-enat } (\text{nlength } xs)) ys))$
by *transfer auto*
show $\text{nlast } xs = \text{nfirst } ys \implies$
 $P (\text{nfirst } ys) \implies$
 $\text{nfinite } xs \implies \neg \text{is-NNil } ys \implies$
 $\text{is-NNil } (\text{nkfilter } P (k + \text{the-enat } (\text{nlength } xs)) ys) \implies$
 $\text{nkfilter } P k (\text{nappend } xs \text{ (ntl } ys)) = \text{nkfilter } P k xs$
apply *transfer*
proof (*auto simp add: kfilter-lappend-lfinite lappend-lnull2*)
fix $xsa :: 'a \text{ llist}$ **and** $ysa :: 'a \text{ llist}$ **and** $Pa :: 'a \Rightarrow \text{bool}$ **and** $ka :: \text{nat}$ **and** $b :: \text{nat}$
assume *a1:* (*if* *lnull* (*kfilter* *Pa* (*ka* + *the-enat* (*epred* (*llength* *xsa*)))) *ysa*)
 $\text{then iterates Suc } (ka + \text{the-enat } (\text{epred } (\text{llength } xsa)))$
 $\text{else } kfilter Pa (ka + \text{the-enat } (\text{epred } (\text{llength } xsa))) ysa = LCons b LNil$
assume *a2:* $\neg \text{lnull } ysa$
assume *a3:* $Pa (\text{lhs } ysa)$
assume *a4:* $\neg \text{lnull } (kfilter Pa (ka + \text{the-enat } (\text{llength } xsa)) (\text{lhs } ysa))$

```

have f5: lnull (LNil::nat llist)
using llength-LNil llength-eq-0 by blast
have f6: ysa = LCons (lhd ysa) (ltl ysa)
  using a2 by auto
then have Pa (lhd ysa)
  using a3 by auto
then have False
  by (metis a1 a2 a4 eq-LConsD f6 kfilter-code(2) kfilter-lnull-conv llength-LNil llength-eq-0 llist.set-sel(1))
then show lappend (kfilter Pa ka xsa) (kfilter Pa (ka + the-enat (llength xsa)) (ltl ysa)) = iterates Suc
ka
by meson
next
fix xsa :: 'b llist and ysa :: 'b llist and Pa :: 'b  $\Rightarrow$  bool and ka :: nat and b :: nat
assume a1: (if lnull (kfilter Pa (ka + the-enat (epred (llength xsa)))) ysa)
  then iterates Suc (ka + the-enat (epred (llength xsa)))
  else kfilter Pa (ka + the-enat (epred (llength xsa))) ysa = LCons b LNil
assume a2:  $\neg$  lnull ysa
assume a3: Pa (lhd ysa)
assume a4:  $\neg$  lnull (kfilter Pa (ka + the-enat (llength xsa)) (ltl ysa))
have f5: lnull (LNil::nat llist)
using llength-LNil llength-eq-0 by blast
have f6: ysa = LCons (lhd ysa) (ltl ysa)
using a2 by auto
then have Pa (lhd ysa)
using a3 by auto
then have False
using f6 f5 a4 a1 by (metis kfilter-LCons kfilter-llength-n-zero llength-eq-0 ltl-simps(2) not-lnull-conv)
then show lappend (kfilter Pa ka xsa) (kfilter Pa (ka + the-enat (llength xsa)) (ltl ysa)) =
  kfilter Pa ka xsa
by meson
qed
next
show nlast xs = nfirst ys  $\Longrightarrow$ 
  P (nfirst ys)  $\Longrightarrow$ 
  nfinite xs  $\Longrightarrow$ 
 $\neg$  is-NNil ys  $\Longrightarrow$ 
 $\neg$  is-NNil (nkfilter P (k + the-enat (nlength xs)) ys)  $\Longrightarrow$ 
  nkfilter P k (nappend xs (ntl ys)) =
  nappend (nkfilter P k xs) (ntl (nkfilter P (k + the-enat (nlength xs)) ys))
apply transfer
proof (auto simp add: lappend-iterates kfilter-lnull-conv split-if-split-asm)
show f1:  $\bigwedge$  xs ys P k x xa.
   $\neg$  lnull xs  $\Longrightarrow$ 
   $\neg$  lnull ys  $\Longrightarrow$ 
  llast xs = lhd ys  $\Longrightarrow$ 
  P (lhd ys)  $\Longrightarrow$ 
  lfinite xs  $\Longrightarrow$ 
 $\forall b. ys \neq LCons b LNil \Longrightarrow$ 
 $\forall b. kfilter P (k + the-enat (epred (llength xs))) ys \neq LCons b LNil \Longrightarrow$ 
 $x \in lset ys \Longrightarrow P x \Longrightarrow \forall x \in lset xs. \neg P x \Longrightarrow P xa \Longrightarrow xa \in lset (ltl ys) \Longrightarrow$ 

```

```

    kfilter P k (lappend xs (ltl ys)) = iterates Suc k
by (metis in-lset-lappend-iff lappend-lbutlast-lld-id lbutlast-lfinite llist.set-intros(1))
next
show f2:  $\bigwedge xs\ ys\ P\ k\ x\ xa\ xb.$ 
   $\neg lnull\ xs \implies$ 
   $\neg lnull\ ys \implies$ 
   $llast\ xs = lhd\ ys \implies$ 
   $P\ (lhd\ ys) \implies$ 
   $lfinite\ xs \implies$ 
   $\forall b. ys \neq LCons\ b\ LNil \implies$ 
   $\forall b. kfilter\ P\ (k + the-enat\ (epred\ (llength\ xs)))\ ys \neq LCons\ b\ LNil \implies$ 
   $x \in lset\ ys \implies$ 
   $P\ x \implies$ 
   $xa \in lset\ xs \implies$ 
   $P\ xa \implies$ 
   $P\ xb \implies$ 
   $xb \in lset\ xs \implies$ 
   $kfilter\ P\ k\ (lappend\ xs\ (ltl\ ys)) =$ 
   $lappend\ (kfilter\ P\ k\ xs)\ (ltl\ (kfilter\ P\ (k + the-enat\ (epred\ (llength\ xs))))\ ys)$ 
proof -
fix xsa :: 'b llist and ysa :: 'b llist and Pa :: 'b  $\Rightarrow$  bool and
  ka :: nat and x :: 'b and xa :: 'b and xb :: 'b
assume a1:  $\neg lnull\ ysa$ 
assume a2:  $Pa\ (lhd\ ysa)$ 
assume a3:  $llast\ xsa = lhd\ ysa$ 
assume a4:  $\neg lnull\ xsa$ 
assume a5:  $lfinite\ xsa$ 
have f6:  $ysa = LCons\ (lhd\ ysa)\ (ltl\ ysa)$ 
using a1 by auto
have f7:  $LCons\ (lhd\ ysa)\ (ltl\ ysa) \neq LNil \wedge lhd\ (LCons\ (lhd\ ysa)\ (ltl\ ysa)) = lhd\ ysa \wedge$ 
   $ltl\ (LCons\ (lhd\ ysa)\ (ltl\ ysa)) = ltl\ ysa$ 
by auto
then have f8:  $kfilter\ Pa\ (the-enat\ (epred\ (llength\ xsa)) + ka)\ (LCons\ (lhd\ ysa)\ (ltl\ ysa)) =$ 
   $LCons\ (the-enat\ (epred\ (llength\ xsa)) + ka)$ 
   $(kfilter\ Pa\ (Suc\ (the-enat\ (epred\ (llength\ xsa)) + ka))\ (ltl\ ysa))$ 
using f6 a2 by (metis (no-types) kfilter-LCons)
have f9:  $\forall bs\ p\ n\ bsa. \neg lfinite\ (bs::'b\ llist) \vee kfilter\ p\ n\ (lappend\ bs\ bsa) =$ 
   $lappend\ (kfilter\ p\ n\ bs)\ (kfilter\ p\ (n + the-enat\ (llength\ bs))\ bsa)$ 
by (meson kfilter-lappend-lfinite)
have f10:  $lfinite\ (lbutlast\ xsa)$ 
using a5 by (meson lbutlast-lfinite)
have f11:  $lappend\ (kfilter\ Pa\ ka\ (lbutlast\ xsa))$ 
   $(kfilter\ Pa\ (ka + the-enat\ (llength\ (lbutlast\ xsa))))\ LNil =$ 
   $kfilter\ Pa\ ka\ (lbutlast\ xsa)$ 
by simp
have  $LCons\ (the-enat\ (epred\ (llength\ xsa)) + ka)\ LNil =$ 
   $kfilter\ Pa\ (the-enat\ (epred\ (llength\ xsa)) + ka)\ (LCons\ (lhd\ ysa)\ LNil)$ 
using a2 by simp
then have f12:  $lappend\ (lappend\ (kfilter\ Pa\ ka\ (lbutlast\ xsa))$ 
   $(kfilter\ Pa\ (ka + the-enat\ (llength\ (lbutlast\ xsa))))\ LNil))$ 

```

```

      (LCons (the-enat (epred (llength xsa)) + ka) LNil) = kfilter Pa ka xsa
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 xsa)))) ysa =
      kfilter Pa (Suc (the-enat (epred (llength xsa)) + ka)) (ltl ysa)
using f8 f6 by (simp add: add.commute)
then show kfilter Pa ka (lappend xsa (ltl ysa)) =
      lappend (kfilter Pa ka xsa) (ltl (kfilter Pa (ka + the-enat (epred (llength xsa)))) ysa))
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  $\bigwedge xs\ ys\ P\ k\ x\ xa\ xb.$ 
   $\neg lnull\ xs \implies$ 
   $\neg lnull\ ys \implies$ 
   $llast\ xs = lhd\ ys \implies$ 
   $P\ (lhd\ ys) \implies$ 
   $lfinite\ xs \implies$ 
   $\forall b. ys \neq LCons\ b\ LNil \implies$ 
   $\forall b. kfilter\ P\ (k + the-enat\ (epred\ (llength\ xs)))\ ys \neq LCons\ b\ LNil \implies$ 
   $x \in lset\ ys \implies$ 
   $P\ x \implies$ 
   $xa \in lset\ xs \implies$ 
   $P\ xa \implies$ 
   $P\ xb \implies$ 
   $xb \in lset\ (ltl\ ys) \implies$ 
   $kfilter\ P\ k\ (lappend\ xs\ (ltl\ ys)) =$ 
   $lappend\ (kfilter\ P\ k\ xs)\ (ltl\ (kfilter\ P\ (k + the-enat\ (epred\ (llength\ xs))))\ ys))$ 
by (simp add: f2)
qed
qed

lemma nleast-conv:
assumes  $\exists x \in nset\ xs. P\ x$ 
shows  $nleast\ P\ xs = (LEAST\ na. na \leq nlength\ xs \wedge P\ (nnth\ xs\ na))$ 
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  $nlast\ xs = nfirst\ ys$ 
   $P\ (nlast\ xs)$ 
   $nfinite\ xs$ 
shows  $nfilter\ P\ (nfuse\ xs\ ys) = nfuse\ (nfilter\ P\ xs)\ (nfilter\ P\ ys)$ 
proof (cases is-NNil ys)
case True
then show ?thesis by (metis nfilter-NNil nfuse-def nfuse-leftneutral)
next
case False
then show ?thesis
  proof (cases is-NNil (nfilter P ys))

```

```

case True
then show ?thesis unfolding nfuse-def using assms nfilter-nappend2[of ntl ys P xs]
nfilter-expand[of ys 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:  $\exists x \in \text{nset } xs. P \ x$ 
    using assms nset-nlast by blast
  have 2:  $\exists x \in \text{nset } (\text{ntl } ys). P \ x$ 
    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:  $\neg \text{is-NNil } (\text{nfilter } P \ ys) \implies$ 
     $\text{nfilter } P \ (\text{nappend } xs \ (\text{ntl } ys)) = \text{nappend } (\text{nfilter } P \ xs) \ (\text{nfilter } P \ (\text{ntl } ys))$ 
    by (simp add: 1 2 assms(3) nappend-nfilter-nfinite)
  have 4:  $\text{is-NNil } (\text{nfilter } P \ ys) \implies \text{nfilter } P \ xs = \text{nappend } (\text{nfilter } P \ xs) \ (\text{nfilter } P \ (\text{ntl } ys))$ 
    by (simp add: False)
  have 5:  $\text{nfilter } P \ (\text{nfuse } xs \ ys) = \text{nfilter } P \ (\text{nappend } xs \ (\text{ntl } ys))$ 
    unfolding nfuse-def
    by (metis (full-types) False nellist.collapse(1) nfilter-NNil)
  have 6:  $(\text{ntl } (\text{nfilter } P \ ys)) = (\text{nfilter } P \ (\text{ntl } ys))$ 
    using assms 2 False nfilter-expand[of ys P]
    by (metis in-nset-ntlD nellist.case-eq-if nellist.sel(5) nfirst-def)
  show ?thesis
    by (metis 3 5 6 False nfuse-def)
qed
qed
qed

```

lemma *nfilter-chop1*:

assumes $n \leq \text{nlength } xs$

$P \ (\text{nlast } (\text{ntaken } n \ xs))$

shows $\text{nfilter } P \ xs = \text{nfuse } (\text{nfilter } P \ (\text{ntaken } n \ xs)) \ (\text{nfilter } P \ (\text{ndropn } n \ xs))$

using *assms*

by (*metis nfuse-ntaken-ndropn ndropn-nfirst nfilter-chop nfinite-ntaken ntaken-nlast*)

lemma *nfilter-chop1-ntaken*:

assumes $n \leq \text{nlength } xs$

$P \ (\text{nlast } (\text{ntaken } n \ xs))$

shows $\text{ntaken } (\text{the-enat } (\text{nlength } (\text{nfilter } P \ (\text{ntaken } n \ xs)))) \ (\text{nfilter } P \ xs) =$
 $(\text{nfilter } P \ (\text{ntaken } n \ xs))$

using *assms nfilter-nappend-ntaken* **by** (*metis nfinite-ntaken nset-nlast*)

lemma *nfilter-nlast*:

assumes $n \leq \text{nlength } xs$

$P \ (\text{nlast } (\text{ntaken } n \ xs))$

```

shows   $nlast(nfilter\ P\ (ntaken\ n\ xs)) = (nlast(ntaken\ n\ xs))$ 
using  assms
proof  (induction n arbitrary: xs)
case  0
then show  ?case by simp
next
case  (Suc n)
then show  ?case
  proof  (cases xs)
  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   $P\ (nfirst(ndropn\ n\ xs))$ 
shows   $nfirst(nfilter\ P\ (ndropn\ n\ xs)) = (nfirst\ (ndropn\ n\ xs))$ 
using  assms
proof  (induction n arbitrary: xs)
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   $n \leq nlength\ xs$ 
            $P\ (nlast\ (ntaken\ n\ xs))$ 
shows   $ndropn\ (the-enat\ (nlength(nfilter\ P\ (ntaken\ n\ xs))))\ (nfilter\ P\ xs) =$ 
            $(nfilter\ P\ (ndropn\ n\ xs))$ 
proof  –
  have  1:  $(nfilter\ P\ xs) = nfuse\ (nfilter\ P\ (ntaken\ n\ xs))\ (nfilter\ P\ (ndropn\ n\ xs))$ 
    by (simp add: assms nfilter-chop1)
  have  2:  $nlast(nfilter\ P\ (ntaken\ n\ xs)) = nfirst\ (nfilter\ P\ (ndropn\ n\ xs))$ 
    by (metis assms(1) assms(2) ndropn-nfirst nfilter-nfirst nfilter-nlast ntaken-nlast)
  have  3:  $ndropn\ (the-enat\ (nlength(nfilter\ P\ (ntaken\ n\ xs))))$ 
            $(nfuse\ (nfilter\ P\ (ntaken\ n\ xs))\ (nfilter\ P\ (ndropn\ n\ xs))) =$ 
            $(nfilter\ P\ (ndropn\ n\ xs))$ 
    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 xs
    P (nlast (ntaken n xs))
  shows nkfilter P k xs = nfuse (nkfilter P k (ntaken n xs)) (nkfilter P (k+n) (ndropn n xs))
using assms nfuse-ntaken-ndropn[of n xs] nkfilter-chop[of (ntaken n xs) (ndropn n xs) P k]
by (metis min.orderE ndropn-nfirst nfinite-ntaken ntaken-nlast ntaken-nlength the-enat.simps)

```

```

lemma nkfilter-nlast:
  assumes n ≤ nlength xs
    P (nlast (ntaken n xs))
  shows nlast(nkfilter P k (ntaken n xs)) = k+n
using assms
proof (induction n arbitrary: k xs)
case 0
then show ?case by simp
next
case (Suc n)
then show ?case
proof (cases xs)
case (NNil x1)
then show ?thesis
using Suc.prem1(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:  $\bigwedge xs k. \llbracket \text{enat } n \leq \text{nlength } xs; P (\text{nlast } (\text{ntaken } n \text{ xs})) \rrbracket \implies$ 
    nlast (nkfilter P k (ntaken n xs)) = 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  $\exists a. a \in \text{nset } (\text{ntaken } n \text{ x22}) \wedge P a$ 
using a1 nfinite-ntaken nset-nlast by blast
then show nlast (nkfilter P k (NCons x21 (ntaken n x22))) = Suc (k + n)
using f3 a2 a1 by (simp add: Bex-def-raw)
qed
qed
qed

```

```

lemma nkfilter-nfirst:
  assumes P (nfirst(ndropn n xs))
  shows nfirst(nkfilter P k (ndropn n xs)) = k

```

```

using assms
proof (induction n arbitrary: k xs)
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 xs)
case (NNil x1)
then show ?thesis using Suc
by (metis ndropn-ndropn plus-1-eq-Suc)
next
case (NCons x21 x22)
then show ?thesis using Suc
by auto
qed
qed

```

```

lemma nkfilter-chop1-ndropn:
assumes  $n \leq \text{nlength } xs$ 
         $P (\text{nlast } (\text{ntaken } n \text{ } xs))$ 
shows  $\text{ndropn } (\text{the-enat } (\text{nlength}(\text{nkfilter } P \text{ } k \text{ } (\text{ntaken } n \text{ } xs)))) (\text{nkfilter } P \text{ } k \text{ } xs) =$ 
        $(\text{nkfilter } P \text{ } (k+n) (\text{ndropn } n \text{ } xs))$ 
proof -
have 1:  $(\text{nkfilter } P \text{ } k \text{ } xs) = \text{nfuse } (\text{nkfilter } P \text{ } k \text{ } (\text{ntaken } n \text{ } xs)) (\text{nkfilter } P \text{ } (k+n) (\text{ndropn } n \text{ } xs))$ 
by (simp add: assms nkfilter-chop1)
have 2:  $\text{nlast}(\text{nkfilter } P \text{ } k \text{ } (\text{ntaken } n \text{ } xs)) = \text{nfirst } (\text{nkfilter } P \text{ } (k+n) (\text{ndropn } n \text{ } xs))$ 
by (metis assms(1) assms(2) ndropn-nfirst nkfilter-nfirst nkfilter-nlast ntaken-nlast)
have 3:  $\text{ndropn } (\text{the-enat } (\text{nlength}(\text{nkfilter } P \text{ } k \text{ } (\text{ntaken } n \text{ } xs))))$ 
        $(\text{nfuse } (\text{nkfilter } P \text{ } k \text{ } (\text{ntaken } n \text{ } xs)) (\text{nkfilter } P \text{ } (k+n) (\text{ndropn } n \text{ } xs))) =$ 
        $(\text{nkfilter } P \text{ } (k+n) (\text{ndropn } n \text{ } xs))$ 
by (metis 2 assms(2) enat-the-enat infinity-ileE ndropn-nfuse nfinite-conv-nlength-enat
    nfinite-ntaken nlength-nkfilter-le nset-nlast)
show ?thesis by (simp add: 1 3)
qed

```

```

lemma nkfilter-chop1-ntaken:
assumes  $n \leq \text{nlength } xs$ 
         $P (\text{nlast } (\text{ntaken } n \text{ } xs))$ 
shows  $\text{ntaken } (\text{the-enat } (\text{nlength}(\text{nkfilter } P \text{ } k \text{ } (\text{ntaken } n \text{ } xs)))) (\text{nkfilter } P \text{ } k \text{ } xs) =$ 
        $(\text{nkfilter } P \text{ } k \text{ } (\text{ntaken } n \text{ } xs))$ 
proof -
have 1:  $(\text{nkfilter } P \text{ } k \text{ } xs) = \text{nfuse } (\text{nkfilter } P \text{ } k \text{ } (\text{ntaken } n \text{ } xs)) (\text{nkfilter } P \text{ } (k+n) (\text{ndropn } n \text{ } xs))$ 
by (simp add: assms nkfilter-chop1)
have 2:  $\text{nlast}(\text{nkfilter } P \text{ } k \text{ } (\text{ntaken } n \text{ } xs)) = \text{nfirst } (\text{nkfilter } P \text{ } (k+n) (\text{ndropn } n \text{ } xs))$ 
by (metis assms(1) assms(2) ndropn-nfirst nkfilter-nfirst nkfilter-nlast ntaken-nlast)
have 3:  $\text{ntaken } (\text{the-enat } (\text{nlength}(\text{nkfilter } P \text{ } k \text{ } (\text{ntaken } n \text{ } xs))))$ 
        $(\text{nfuse } (\text{nkfilter } P \text{ } k \text{ } (\text{ntaken } n \text{ } xs)) (\text{nkfilter } P \text{ } (k+n) (\text{ndropn } n \text{ } xs))) =$ 

```

```

      (nkfilter P k (ntaken n xs))
    by (metis 1 assms(1) assms(2) nfinite-ntaken nkfilter-nappend-ntaken nset-nlast)
  show ?thesis by (simp add: 1 3)
qed

```

end

end

4 Finite and Infinite ITL Semantics

```

theory Semantics
imports NELList 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 intervals = 'a nellist

type-synonym ('a,'b) formfun   = 'a intervals  $\Rightarrow$  'b
type-synonym 'a formula       = ('a,bool) formfun
type-synonym ('a,'b) stfun     = 'a  $\Rightarrow$  'b
type-synonym 'a stpred        = ('a,bool) stfun

```

```

instance
  fun :: (type,type) world ..

```

```

instance
  prod :: (type,type) world ..

```

```

instance
  sum :: (type,type) world ..

```

```

instance
  nellist :: (type) world ..

```

Pair, function, sum, and interval are instantiated to be of type class world. This allows use of the lifted Intensional logic for formulas, and standard logical connectives can therefore be used.

4.2 Semantics of ITL

The semantics of ITL is defined.

```

definition skip-d :: ('a :: world) formula

```

where *skip-d* $\equiv (\lambda s. \text{nlength } s = (\text{enat } (\text{Suc } 0)))$

definition *chop-d* $:: ('a :: \text{world}) \text{ formula} \Rightarrow ('a :: \text{world}) \text{ formula} \Rightarrow ('a :: \text{world}) \text{ formula}$

where *chop-d* *F1 F2* \equiv

($\lambda s.$
 $(\exists n \leq \text{nlength } s. ((\text{ntaken } n \ s) \models F1) \wedge ((\text{ndropn } n \ s) \models F2))$
 $\vee (\neg \text{nfinite } s \wedge (s \models F1))$
 $)$

definition *current-val-d* $:: ('a :: \text{world}, 'b) \text{ stfun} \Rightarrow ('a, 'b) \text{ formfun}$

where *current-val-d* *f* $= (\lambda s. ((\text{nfirst } s) \models f))$

definition *next-val-d* $:: ('a :: \text{world}, 'b) \text{ stfun} \Rightarrow ('a, 'b) \text{ formfun}$

where *next-val-d* *f* \equiv

($\lambda s. (\text{if } \text{nlength } s \neq (\text{enat } 0) \text{ then } ((\text{nnth } s \ 1) \models f) \text{ else } (\epsilon (x :: 'b). x = x))$
 $)$

definition *fin-val-d* $:: ('a :: \text{world}, 'b) \text{ stfun} \Rightarrow ('a, 'b) \text{ formfun}$

where *fin-val-d* *f* $\equiv \lambda s. ((\text{if } \text{nfinite } s \text{ then } ((\text{nlast } s) \models f) \text{ else } (\epsilon (x :: 'b). x = x)))$

definition *penult-val-d* $:: ('a :: \text{world}, 'b) \text{ stfun} \Rightarrow ('a, 'b) \text{ formfun}$

where *penult-val-d* *f* \equiv

($\lambda s.$
 $(\text{if } \text{nfinite } s$
 $\text{then } (\text{if } \text{nlength } s \neq (\text{enat } 0)$
 $\text{then } ((\text{nnth } s (\text{the-enat } (\text{epred } (\text{nlength } s)))) \models f)$
 $\text{else } (\epsilon (x :: 'b). x = x))$
 $\text{else } (\epsilon (x :: 'b). x = x)$
 $)$
 $)$

syntax

-skip-d $:: \text{lift} \quad ((\text{skip}))$
-chop-d $:: [\text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((-;-) [84, 84] \ 83)$
-current-val-d $:: \text{lift} \Rightarrow \text{lift} \quad ((\$-) [100] \ 99)$
-next-val-d $:: \text{lift} \Rightarrow \text{lift} \quad ((-\$) [100] \ 99)$
-fin-val-d $:: \text{lift} \Rightarrow \text{lift} \quad ((!-) [100] \ 99)$
-penult-val-d $:: \text{lift} \Rightarrow \text{lift} \quad ((-!) [100] \ 99)$
TEMP $:: \text{lift} \Rightarrow 'b \quad ((\text{TEMP } -))$

syntax (ASCII)

-skip-d $:: \text{lift} \quad ((\text{skip}))$
-chop-d $:: [\text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((-;-) [84, 84] \ 83)$
-current-val-d $:: \text{lift} \Rightarrow \text{lift} \quad ((\$-) [100] \ 99)$
-next-val-d $:: \text{lift} \Rightarrow \text{lift} \quad ((-\$) [100] \ 99)$
-fin-val-d $:: \text{lift} \Rightarrow \text{lift} \quad ((!-) [100] \ 99)$
-penult-val-d $:: \text{lift} \Rightarrow \text{lift} \quad ((-!) [100] \ 99)$

translations

$-skip-d \quad \Rightarrow \text{CONST } skip-d$
 $-chop-d \quad \Rightarrow \text{CONST } chop-d$
 $-current-val-d \Rightarrow \text{CONST } current-val-d$
 $-next-val-d \quad \Rightarrow \text{CONST } next-val-d$
 $-fin-val-d \quad \Rightarrow \text{CONST } fin-val-d$
 $-penult-val-d \Rightarrow \text{CONST } penult-val-d$
 $TEMP F \quad \rightarrow (F :: (- \text{ intervals}) \Rightarrow -)$

4.3 Abbreviations

Some standard temporal abbreviations, with their concrete syntax.

definition $infinite-d :: ('a :: world) \text{ formula}$
where $infinite-d \equiv LIFT(\#True; \#False)$

syntax

$-infinite-d \quad :: \text{ lift} \quad (inf)$

syntax (ASCII)

$-infinite-d \quad :: \text{ lift} \quad (inf)$

translations

$-infinite-d \Rightarrow \text{CONST } infinite-d$

definition $finite-d :: ('a :: world) \text{ formula}$

where $finite-d \equiv LIFT(\neg(inf))$

syntax

$-finite-d \quad :: \text{ lift} \quad (finite)$

syntax (ASCII)

$-finite-d \quad :: \text{ lift} \quad (finite)$

translations

$-finite-d \Rightarrow \text{CONST } finite-d$

definition $schop-d :: ('a :: world) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$

where $schop-d F1 F2 \equiv LIFT((F1 \wedge finite); F2)$

definition $sometimes-d :: ('a :: world) \text{ formula} \Rightarrow 'a \text{ formula}$

where $sometimes-d F \equiv LIFT(finite; F)$

definition $di-d :: ('a :: world) \text{ formula} \Rightarrow 'a \text{ formula}$

where $di-d F \equiv LIFT(F; \#True)$

definition $da-d :: ('a :: world) \text{ formula} \Rightarrow 'a \text{ formula}$

where $da-d F \equiv LIFT(finite; (F; \#True))$

definition $next-d :: ('a :: world) \text{ formula} \Rightarrow 'a \text{ formula}$

where $next-d F \equiv LIFT(skip; F)$

definition $prev-d :: ('a::world) formula \Rightarrow 'a formula$
where $prev-d F \equiv LIFT(F;skip)$

syntax

$-schop-d \quad :: [lift, lift] \Rightarrow lift ((- \frown -) [84,84] 83)$
 $-sometimes-d :: lift \Rightarrow lift ((\Diamond -) [88] 87)$
 $-di-d \quad \quad :: lift \Rightarrow lift ((di -) [88] 87)$
 $-da-d \quad \quad :: lift \Rightarrow lift ((da -) [88] 87)$
 $-next-d \quad \quad :: lift \Rightarrow lift ((\bigcirc -) [88] 87)$
 $-prev-d \quad \quad :: lift \Rightarrow lift ((prev -) [88] 87)$

syntax (ASCII)

$-schop-d \quad :: [lift, lift] \Rightarrow lift ((- \text{schop} -) [84,84] 83)$
 $-sometimes-d :: lift \Rightarrow lift ((<>-) [88] 87)$
 $-di-d \quad \quad :: lift \Rightarrow lift ((di -) [88] 87)$
 $-da-d \quad \quad :: lift \Rightarrow lift ((da -) [88] 87)$
 $-next-d \quad \quad :: lift \Rightarrow lift ((next -) [88] 87)$
 $-prev-d \quad \quad :: lift \Rightarrow lift ((prev -) [88] 87)$

translations

$-schop-d \quad \Rightarrow CONST \text{schop-d}$
 $-sometimes-d \Rightarrow CONST \text{sometimes-d}$
 $-di-d \quad \quad \Rightarrow CONST \text{di-d}$
 $-da-d \quad \quad \Rightarrow CONST \text{da-d}$
 $-next-d \quad \quad \Rightarrow CONST \text{next-d}$
 $-prev-d \quad \quad \Rightarrow CONST \text{prev-d}$

definition $df-d :: ('a::world) formula \Rightarrow 'a formula$
where $df-d F \equiv LIFT(F \frown \#True)$

definition $sda-d :: ('a::world) formula \Rightarrow 'a formula$
where $sda-d F \equiv LIFT(\#True \frown (F \frown \#True))$

definition $always-d :: ('a::world) formula \Rightarrow 'a formula$
where $always-d F \equiv LIFT(\neg(\Diamond(\neg F)))$

definition $bi-d :: ('a::world) formula \Rightarrow 'a formula$
where $bi-d F \equiv LIFT(\neg(di(\neg F)))$

definition $ba-d :: ('a::world) formula \Rightarrow 'a formula$
where $ba-d F \equiv LIFT(\neg(da(\neg F)))$

definition $wnext-d :: ('a::world) formula \Rightarrow 'a formula$
where $wnext-d F \equiv LIFT(\neg(\bigcirc(\neg F)))$

definition $wprev-d :: ('a::world) formula \Rightarrow 'a formula$
where $wprev-d F \equiv LIFT(\neg(prev(\neg F)))$

definition *more-d* :: ('a::world) formula
where *more-d* \equiv *LIFT*($\circ(\#True)$)

syntax

-df-d :: *lift* \Rightarrow *lift* ((df -) [88] 87)
-sda-d :: *lift* \Rightarrow *lift* ((sda -) [88] 87)
-always-d :: *lift* \Rightarrow *lift* ((\square -) [88] 87)
-bi-d :: *lift* \Rightarrow *lift* ((bi -) [88] 87)
-ba-d :: *lift* \Rightarrow *lift* ((ba -) [88] 87)
-wnext-d :: *lift* \Rightarrow *lift* ((wnext -) [88] 87)
-wprev-d :: *lift* \Rightarrow *lift* ((wprev -) [88] 87)
-more-d :: *lift* ((more))

syntax (ASCII)

-df-d :: *lift* \Rightarrow *lift* ((df -) [88] 87)
-sda-d :: *lift* \Rightarrow *lift* ((sda -) [88] 87)
-always-d :: *lift* \Rightarrow *lift* (([] -) [88] 87)
-bi-d :: *lift* \Rightarrow *lift* ((bi -) [88] 87)
-ba-d :: *lift* \Rightarrow *lift* ((ba -) [88] 87)
-wnext-d :: *lift* \Rightarrow *lift* ((wnext -) [88] 87)
-wprev-d :: *lift* \Rightarrow *lift* ((wprev -) [88] 87)
-more-d :: *lift* ((more))

translations

-df-d \Rightarrow *CONST* df-d
-sda-d \Rightarrow *CONST* sda-d
-always-d \Rightarrow *CONST* always-d
-bi-d \Rightarrow *CONST* bi-d
-ba-d \Rightarrow *CONST* ba-d
-wnext-d \Rightarrow *CONST* wnext-d
-wprev-d \Rightarrow *CONST* wprev-d
-more-d \Rightarrow *CONST* more-d

definition *bf-d* :: ('a::world) formula \Rightarrow 'a formula
where *bf-d* *F* \equiv *LIFT*($\neg(df(\neg F))$)

definition *sba-d* :: ('a::world) formula \Rightarrow 'a formula
where *sba-d* *F* \equiv *LIFT*($\neg(sda(\neg F))$)

definition *empty-d* :: ('a::world) formula
where *empty-d* \equiv *LIFT*($\neg(more)$)

definition *fmore-d* :: ('a::world) formula
where *fmore-d* \equiv *LIFT*(*more* \wedge *finite*)

definition $dm-d :: ('a::world) formula \Rightarrow 'a formula$
where $dm-d F \equiv LIFT(\#True;(more \wedge F))$

syntax

$-bf-d \quad :: lift \Rightarrow lift ((bf -) [88] 87)$
 $-sba-d \quad :: lift \Rightarrow lift ((sba -) [88] 87)$
 $-empty-d \quad :: lift \quad ((empty))$
 $-fmore-d \quad :: lift \quad ((fmore))$
 $-dm-d \quad :: lift \Rightarrow lift ((dm -) [88] 87)$

syntax (*ASCII*)

$-bf-d \quad :: lift \Rightarrow lift ((bf -) [88] 87)$
 $-sba-d \quad :: lift \Rightarrow lift ((sba -) [88] 87)$
 $-empty-d \quad :: lift \quad ((empty))$
 $-fmore-d \quad :: lift \quad ((fmore))$
 $-dm-d \quad :: lift \Rightarrow lift ((dm -) [88] 87)$

translations

$-bf-d \quad \Rightarrow CONST\ bf-d$
 $-sba-d \quad \Rightarrow CONST\ sba-d$
 $-empty-d \Rightarrow CONST\ empty-d$
 $-fmore-d \Rightarrow CONST\ fmore-d$
 $-dm-d \quad \Rightarrow CONST\ dm-d$

definition $bm-d :: ('a::world) formula \Rightarrow 'a formula$
where $bm-d F \equiv LIFT(\neg(dm(\neg F)))$

definition $init-d :: ('a::world) formula \Rightarrow 'a formula$
where $init-d F \equiv LIFT((empty \wedge F);\#True)$

definition $fin-d :: ('a::world) formula \Rightarrow 'a formula$
where $fin-d F \equiv LIFT(\Box(empty \longrightarrow F))$

definition $halt-d :: ('a::world) formula \Rightarrow 'a formula$
where $halt-d F \equiv LIFT(\Box(empty = F))$

definition $initonly-d :: ('a::world) formula \Rightarrow 'a formula$
where $initonly-d F \equiv LIFT(bi(empty = F))$

definition $keep-d :: ('a::world) formula \Rightarrow 'a formula$
where $keep-d F \equiv LIFT(ba(skip \longrightarrow F))$

definition $yields-d :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a formula$
where $yields-d F1 F2 \equiv LIFT(\neg(F1;(\neg F2)))$

definition $syields-d :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a formula$
where $syields-d F1 F2 \equiv LIFT(\neg(F1 \frown (\neg F2)))$

definition $ifthenelse-d :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a formula \Rightarrow 'a formula$

where *ifthenelse-d* $F\ G\ H \equiv \text{LIFT}((F \wedge G) \vee (\neg F \wedge H))$

primrec *power-d* $:: ('a::\text{world})\ \text{formula} \Rightarrow \text{nat} \Rightarrow 'a\ \text{formula}$

where *pow-0* $: (\text{power-d}\ F\ 0) = \text{LIFT}(\text{empty})$

| *pow-Suc*: $(\text{power-d}\ F\ (\text{Suc}\ n)) = \text{LIFT}((F \wedge \text{finite});(\text{power-d}\ F\ n))$

primrec *spower-d* $:: ('a::\text{world})\ \text{formula} \Rightarrow \text{nat} \Rightarrow 'a\ \text{formula}$

where *spow-0* $: (\text{spower-d}\ F\ 0) = \text{LIFT}(\text{empty})$

| *spow-Suc*: $(\text{spower-d}\ F\ (\text{Suc}\ n)) = \text{LIFT}(F \frown (\text{spower-d}\ F\ n))$

syntax

-bm-d $:: \text{lift} \Rightarrow \text{lift} \quad ((bm\ -)\ [88]\ 87)$
-init-d $:: \text{lift} \Rightarrow \text{lift} \quad ((init\ -)\ [88]\ 87)$
-fin-d $:: \text{lift} \Rightarrow \text{lift} \quad ((fin\ -)\ [88]\ 87)$
-halt-d $:: \text{lift} \Rightarrow \text{lift} \quad ((halt\ -)\ [88]\ 87)$
-initonly-d $:: \text{lift} \Rightarrow \text{lift} \quad ((initonly\ -)\ [88]\ 87)$
-keep-d $:: \text{lift} \Rightarrow \text{lift} \quad ((keep\ -)\ [88]\ 87)$
-yields-d $:: [\text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((-\ \text{yields}\ -)\ [88,88]\ 87)$
-syields-d $:: [\text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((-\ \text{syields}\ -)\ [88,88]\ 87)$
-ifthenelse-d $:: [\text{lift}, \text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((if_i\ -\ \text{then}\ -\ \text{else}\ -)\ [88,88,88]\ 87)$
-power-d $:: [\text{lift}, \text{nat}] \Rightarrow \text{lift} \quad ((power\ -)\ [88,88]\ 87)$
-spower-d $:: [\text{lift}, \text{nat}] \Rightarrow \text{lift} \quad ((spower\ -)\ [88,88]\ 87)$

syntax (ASCII)

-bm-d $:: \text{lift} \Rightarrow \text{lift} \quad ((bm\ -)\ [88]\ 87)$
-init-d $:: \text{lift} \Rightarrow \text{lift} \quad ((init\ -)\ [88]\ 87)$
-fin-d $:: \text{lift} \Rightarrow \text{lift} \quad ((fin\ -)\ [88]\ 87)$
-halt-d $:: \text{lift} \Rightarrow \text{lift} \quad ((halt\ -)\ [88]\ 87)$
-initonly-d $:: \text{lift} \Rightarrow \text{lift} \quad ((initonly\ -)\ [88]\ 87)$
-keep-d $:: \text{lift} \Rightarrow \text{lift} \quad ((keep\ -)\ [88]\ 87)$
-yields-d $:: [\text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((-\ \text{yields}\ -)\ [88,88]\ 87)$
-syields-d $:: [\text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((-\ \text{syields}\ -)\ [88,88]\ 87)$
-ifthenelse-d $:: [\text{lift}, \text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((if_i\ -\ \text{then}\ -\ \text{else}\ -)\ [88,88,88]\ 87)$
-power-d $:: [\text{lift}, \text{nat}] \Rightarrow \text{lift} \quad ((power\ -)\ [88,88]\ 87)$
-spower-d $:: [\text{lift}, \text{nat}] \Rightarrow \text{lift} \quad ((spower\ -)\ [88,88]\ 87)$

translations

-bm-d $\equiv \text{CONST}\ bm\text{-d}$
-init-d $\equiv \text{CONST}\ init\text{-d}$
-fin-d $\equiv \text{CONST}\ fin\text{-d}$
-halt-d $\equiv \text{CONST}\ halt\text{-d}$
-initonly-d $\equiv \text{CONST}\ initonly\text{-d}$
-keep-d $\equiv \text{CONST}\ keep\text{-d}$
-yields-d $\equiv \text{CONST}\ yields\text{-d}$
-syields-d $\equiv \text{CONST}\ syields\text{-d}$
-ifthenelse-d $\equiv \text{CONST}\ ifthenelse\text{-d}$
-power-d $\equiv \text{CONST}\ power\text{-d}$
-spower-d $\equiv \text{CONST}\ spower\text{-d}$

definition $len-d :: nat \Rightarrow ('a::world) formula$
where $len-d\ n \equiv LIFT(power\ skip\ n)$

definition $fpowerstar-d :: ('a::world) formula \Rightarrow 'a\ formula$
where $fpowerstar-d\ F \equiv LIFT(\exists\ k.\ power\ F\ k)$

definition $spowerstar-d :: ('a::world) formula \Rightarrow 'a\ formula$
where $spowerstar-d\ F \equiv LIFT(\exists\ k.\ spower\ F\ k)$

syntax

$-len-d \quad :: nat \Rightarrow lift \quad ((len\ -) [88] 87)$
 $-fpowerstar-d \quad :: lift \Rightarrow lift \quad ((fpowerstar\ -) [85] 85)$
 $-spowerstar-d \quad :: lift \Rightarrow lift \quad ((spowerstar\ -) [85] 85)$

syntax (*ASCII*)

$-len-d \quad :: nat \Rightarrow lift \quad ((len\ -) [88] 87)$
 $-fpowerstar-d \quad :: lift \Rightarrow lift \quad ((fpowerstar\ -) [85] 85)$
 $-spowerstar-d \quad :: lift \Rightarrow lift \quad ((spowerstar\ -) [85] 85)$

translations

$-len-d \quad \Rightarrow CONST\ len-d$
 $-fpowerstar-d \quad \Rightarrow CONST\ fpowerstar-d$
 $-spowerstar-d \quad \Rightarrow CONST\ spowerstar-d$

definition $powerstar-d :: ('a::world) formula \Rightarrow 'a\ formula$
where $powerstar-d\ F \equiv LIFT((\exists\ k.\ power\ F\ k);(empty \vee (F \wedge inf)))$

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 $schopstar-d\ F \equiv LIFT(spowerstar\ (F \wedge more))$

syntax

$-chopstar-d \quad :: lift \Rightarrow lift \quad ((-^*) [85] 85)$
 $-schopstar-d \quad :: lift \Rightarrow lift \quad ((schopstar\ -) [85] 85)$

syntax (*ASCII*)

-chopstar-d $:: \text{lift} \Rightarrow \text{lift} \quad ((\text{chopstar } -) [85] 85)$
 -schopstar-d $:: \text{lift} \Rightarrow \text{lift} \quad ((\text{schopstar } -) [85] 85)$

translations

-chopstar-d $\Rightarrow \text{CONST chopstar-d}$
 -schopstar-d $\Rightarrow \text{CONST schopstar-d}$

definition *sfin-d* $:: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$

where *sfin-d* $F \equiv \text{LIFT}(\neg (\text{fin } (\neg F)))$

definition *ifthen-d* $:: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$

where *ifthen-d* $F G \equiv \text{LIFT}(\text{if}_i F \text{ then } G \text{ else } \# \text{True })$

definition *while-d* $:: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$

where *while-d* $F G \equiv \text{LIFT}((F \wedge G)^* \wedge (\text{fin } (\neg F))))$

syntax

-ifthen-d $:: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{if}_i - \text{ then } -) [88, 88] 87)$
 -while-d $:: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{while } - \text{ do } -) [88, 88] 87)$
 -sfin-d $:: \text{lift} \Rightarrow \text{lift } ((\text{sfin } -) [88] 87)$

syntax (*ASCII*)

-ifthen-d $:: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{if}_i - \text{ then } -) [88, 88] 87)$
 -while-d $:: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{while } - \text{ do } -) [88, 88] 87)$
 -sfin-d $:: \text{lift} \Rightarrow \text{lift } ((\text{sfin } -) [88] 87)$

translations

-ifthen-d $\Rightarrow \text{CONST ifthen-d}$
 -while-d $\Rightarrow \text{CONST while-d}$
 -sfin-d $\Rightarrow \text{CONST sfin-d}$

definition *swhile-d* $:: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$

where *swhile-d* $F G \equiv \text{LIFT}(\text{schopstar}(F \wedge G) \wedge (\text{sfin } (\neg F))))$

definition *repeat-d* $:: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$

where *repeat-d* $F G \equiv \text{LIFT}(F; \text{while } (\neg G) \text{ do } F)$

syntax

-swhile-d $:: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{swhile } - \text{ do } -) [88, 88] 87)$
 -repeat-d $:: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{repeat } - \text{ until } -) [88, 88] 87)$

syntax (*ASCII*)

-swhile-d $:: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{swhile } - \text{ do } -) [88, 88] 87)$
 -repeat-d $:: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{repeat } - \text{ until } -) [88, 88] 87)$

translations

-*swhile-d* \Rightarrow *CONST while-d*

-*repeat-d* \Rightarrow *CONST repeat-d*

definition *srepeat-d* :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a formula

where *srepeat-d* F G \equiv *LIFT*(F \frown *swhile* (\neg G) *do* F)

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 λ 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 λ 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 λ s. s \models *keep*(*temporal-assign-d* v e)

definition *stable-d* :: ('a::world,'b) stfun \Rightarrow 'a formula

where *stable-d* v \equiv λ s. s \models *gets-d* v (*current-val-d* v)

definition *padded-d* :: ('a::world,'b) stfun \Rightarrow 'a formula

where *padded-d* v \equiv λ s. s \models (*stable-d* v);*skip* \vee *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 λ s. s \models (*temporal-assign-d* v e) \wedge (*padded-d* v)

syntax

-*srepeat-d* :: [*lift*,*lift*] \Rightarrow *lift* ((*srepeat* - *until* -) [88,88] 87)

-*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)

-*srepeat-d* :: [*lift*,*lift*] \Rightarrow *lift* ((*srepeat* - *until* -) [88,88] 87)

-*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)

$-stable-d \quad :: lift \Rightarrow lift \quad ((stable -) [51] 50)$
 $-padded-d \quad :: lift \Rightarrow lift \quad ((padded -) [51] 50)$
 $-padded-temp-assign-d :: [lift, lift] \Rightarrow lift \quad ((- <^\sim -) [50, 51] 50)$

translations

$-srepeat-d \quad \Rightarrow CONST \ srepeat-d$
 $-next-assign-d \quad \Rightarrow CONST \ next-assign-d$
 $-prev-assign-d \quad \Rightarrow CONST \ prev-assign-d$
 $-always-eq-d \quad \Rightarrow CONST \ always-eq-d$
 $-temporal-assign-d \quad \Rightarrow CONST \ temporal-assign-d$
 $-gets-d \quad \Rightarrow CONST \ gets-d$
 $-stable-d \quad \Rightarrow CONST \ stable-d$
 $-padded-d \quad \Rightarrow CONST \ padded-d$
 $-padded-temp-assign-d \Rightarrow CONST \ padded-temp-assign-d$

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 ifthenelse-d-def power-d-def len-d-def fpowerstar-d-def spower-d-def powerstar-d-def chopstar-d-def schopstar-d-def sfinit-d-def ifthen-d-def while-d-def swhile-d-def repeat-d-def srepeat-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 :*

$(w \models skip) = (nlength \ w = (enat \ 1))$

by (*simp add: itl-def*)

lemma *chop-defs :*

$(w \models F1 ; F2) =$
 $($
 $\quad (\exists n \leq nlength \ w. \ (ntaken \ n \ w) \models F1) \ \wedge \ ((ndropn \ n \ w) \models F2))$
 $\quad \vee \ (\neg \ nfinite \ w \wedge (w \models F1))$
 $)$

by (*simp add: itl-def*)

lemma *infinite-defs:*

$(w \models inf) = (\neg \ nfinite \ w)$

by (*simp add: itl-def*)

lemma *finite-defs :*

$(w \models finite) = (nfinite \ w)$

by (*simp add: itl-def*)

lemma *schop-defs :*

$(w \models F1 \frown F2) =$
 $($

```

    (  $\exists n \leq \text{nlength } w. ( (\text{ntaken } n \ w) \models F1 ) \wedge ( (\text{ndropn } n \ w) \models F2 ) )$ 
  )

by (auto simp add: itl-def chop-defs finite-defs )

lemma sometimes-defs :
  (  $w \models \Diamond F$  ) = (  $\exists n \leq \text{nlength } w. ( (\text{ndropn } n \ w) \models F )$  )
by (simp add: itl-def finite-defs chop-defs)

lemma always-defs :
  (  $w \models \Box F$  ) =
  (  $\forall n \leq \text{nlength } w. ( (\text{ndropn } n \ w) \models F )$  )
by (simp add: itl-def sometimes-defs)

lemma di-defs :
  (  $w \models \text{di } F$  ) =
  (  $(\exists n \leq \text{nlength } w. ( (\text{ntaken } n \ w) \models F )) \vee (\neg \text{nfinite } w \wedge (w \models F))$  )
by (simp add: itl-def )

lemma df-defs :
  (  $w \models \text{df } F$  ) =
  (  $\exists n \leq \text{nlength } w. ( (\text{ntaken } n \ w) \models F )$  )
by (simp add: df-d-def schop-defs)

lemma bi-defs :
  (  $w \models \text{bi } F$  ) =
  (  $(\forall n \leq \text{nlength } w. ( (\text{ntaken } n \ w) \models F )) \wedge (\text{nfinite } w \vee (w \models F))$  )
by (simp add: itl-def di-defs )

lemma bf-defs :
  (  $w \models \text{bf } F$  ) =
  (  $\forall n \leq \text{nlength } w. ( (\text{ntaken } n \ w) \models F )$  )
by (simp add: bf-d-def df-defs )

lemma da-defs :
  (  $w \models \text{da } F$  ) =
  (  $(\exists n \ \text{na}. ( n + \text{na} \leq \text{nlength } w \wedge ( (\text{nsubn } w \ n \ (n + \text{na})) \models F )) \vee (\neg \text{nfinite } w \wedge ( (\text{ndropn } n \ w) \models F )) )$ 
  )
proof
  (auto simp add: itl-def chop-defs nsubn-def)
  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:  $F (ntaken\ na\ (ndropn\ n\ w))$ 
have enat  $n \leq nlength\ w$ 
using a1 by (meson enat-ord-simps(1) le-iff-add order-subst2)
then show  $\exists n. enat\ n \leq nlength\ w \wedge$ 
   $((\exists na. enat\ na \leq nlength\ w - enat\ n \wedge F (ntaken\ na\ (ndropn\ n\ w)))$ 
   $\vee \neg nfinite\ w \wedge F (ndropn\ n\ w))$ 
using a2 a1 by (metis (no-types) add-diff-cancel-left' enat-minus-mono1 idiff-enat-enat)
next
show  $\bigwedge n. \neg nfinite\ w \implies$ 
   $F (ndropn\ n\ w) \implies$ 
   $\exists n. enat\ n \leq nlength\ w \wedge$ 
   $((\exists na. enat\ na \leq nlength\ w - enat\ n \wedge F (ntaken\ na\ (ndropn\ n\ w))) \vee F (ndropn\ n\ w))$ 
by (meson enat-ile le-cases nfinite-conv-nlength-enat)
qed

```

```

lemma ba-defs :
   $(w \models ba\ F) =$ 
   $(\forall n\ na. (enat\ (n + na) \leq nlength\ w \longrightarrow ( (nsubn\ w\ n\ (n+na)) \models F))$ 
   $\wedge (nfinite\ w \vee ( (ndropn\ n\ w) \models F)))$ 
by (simp add: ba-d-def da-defs )

```

```

lemma sda-defs :
   $(w \models sda\ F) =$ 
   $(\exists n\ na. (n+na \leq nlength\ w \wedge ( (nsubn\ w\ n\ (n+ na)) \models F)))$ 
proof
  (auto simp add: sda-d-def schop-defs nsubn-def)
show  $\bigwedge n\ na.$ 
   $enat\ n \leq nlength\ w \implies$ 
   $enat\ na \leq nlength\ w - enat\ n \implies F (ntaken\ na\ (ndropn\ n\ w)) \implies$ 
   $\exists n\ na. enat\ (n + na) \leq nlength\ w \wedge F (ntaken\ na\ (ndropn\ n\ w))$ 
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 :: nat and na :: nat
assume a1:  $F (ntaken\ na\ (ndropn\ n\ w))$ 
assume a2:  $enat\ (n + na) \leq nlength\ w$ 
have enat  $na \leq nlength\ w - enat\ n$ 
by (metis a2 add-diff-cancel-left' enat-minus-mono1 idiff-enat-enat)
then show  $\exists n. enat\ n \leq nlength\ w \wedge$ 
   $(\exists na. enat\ na \leq nlength\ w - enat\ n \wedge F (ntaken\ na\ (ndropn\ n\ w)))$ 
using a2 a1
by (metis add.right-neutral le-cases le-zero-eq ndropn-all ndropn-nlength nlength-NNil
  the-enat.simps the-enat-0)
qed

```

```

lemma sba-defs :
   $(w \models sba\ F) =$ 
   $(\forall n\ na. n+na \leq nlength\ w \longrightarrow ( (nsubn\ w\ n\ (n+na)) \models F))$ 
by (simp add: sba-d-def sda-defs)

```

lemma *next-defs* :

($w \models \circ F$) =
 ($\text{nlength } w \neq (\text{enat } 0) \wedge ((\text{ndropn } 1 \ w) \models F)$)

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* :

($w \models \text{prev } F$) =
 ($(\text{nlength } w \neq (\text{enat } 0) \wedge \text{nfinite } w \wedge$
 ($(\text{ntaken } (\text{the-enat}(\text{epred}(\text{nlength } w))) \ w) \models F)) \vee (\neg \text{nfinite } w \wedge (w \models F))$)

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* :

($w \models \text{wprev } F$) =
 ($(\text{nfinite } w \longrightarrow (\text{nlength } w = (\text{enat } 0) \vee ((\text{ntaken } (\text{the-enat}(\text{epred}(\text{nlength } w))) \ w) \models F)))$
 $\wedge (\text{nfinite } w \vee (w \models F))$)

by (*simp add: wprev-d-def prev-defs*)

lemma *more-defs* :

($w \models \text{more}$) =
 ($\text{nlength } w \neq (\text{enat } 0)$)

by (*simp add: more-d-def next-defs*)

lemma *fmore-defs* :

($w \models \text{fmore}$) =
 ($\text{nfinite } w \wedge (\text{nlength } w \neq (\text{enat } 0))$)

by (*auto simp add: fmore-d-def more-defs finite-defs*)

lemma *empty-defs* :

($w \models \text{empty}$) =

(*nlength w = (enat 0)*)
by (*simp add: empty-d-def more-defs*)

lemma *init-defs* :

(*w* \models *init F*) =
 ((*ntaken 0 w*) \models *F*)

by (*simp add: init-d-def chop-defs empty-defs min-def*)
 (*metis nlength-eq-enat-nfiniteD ntaken-eq-NNil-iff zero-enat-def zero-le*)

lemma *initalt-defs* :

(*w* \models *bi(empty \longrightarrow F)*) =
 ((*ntaken 0 w*) \models *F*)

by (*simp add: bi-defs empty-defs min-def*)
 (*metis nlength-eq-enat-nfiniteD zero-enat-def zero-le*)

lemma *fin-defs* :

(*w* \models *fin F*) =
 ((*nfinite w* \wedge ((*ndropn (the-enat(nlength w)) w*) \models *F*)) \vee (\neg *nfinite w*))

by (*simp add: fin-d-def empty-defs always-defs*)
 (*metis add.right-neutral enat.distinct(2) enat-add-sub-same le-iff-add nfinite-nlength-enat
 nlength-eq-enat-nfiniteD the-enat.simps zero-enat-def*)

lemma *finalt-defs* :

(*w* \models *#True;(F \wedge empty)*) =
 ((*nfinite w* \wedge ((*ndropn (the-enat(nlength w)) w*) \models *F*)) \vee (\neg *nfinite w*))

by (*simp add: chop-defs empty-defs*)
 (*metis add.right-neutral enat.distinct(2) enat-add-sub-same le-iff-add nfinite-nlength-enat
 the-enat.simps zero-enat-def*)

lemma *sfin-defs* :

(*w* \models *sfin F*) = (*nfinite w* \wedge ((*ndropn (the-enat(nlength w)) w*) \models *F*))

by (*auto simp add: sfin-d-def fin-defs*)

lemma *halt-defs* :

(*w* \models *halt(F)*) = ($\forall n. n \leq nlength w \longrightarrow (nlength w = n) = ((ndropn n w) \models F)$)

by (*simp add: halt-d-def empty-defs always-defs*)
 (*metis add.right-neutral dual-order.strict-iff-order enat-add-sub-same enat-ord-code(4)
 le-iff-add zero-enat-def*)

lemma *initonly-defs* :

(*w* \models *initonly(F)*) =
 (($\forall n. n \leq nlength w \longrightarrow (n = 0) = ((ntaken n w) \models F)$) \wedge
 (*nfinite w* $\vee (nlength w = 0) = F w$))

by (*simp add: min.absorb1 initonly-d-def bi-defs empty-defs zero-enat-def*)

lemma *ifthenelse-defs*:

(*w* \models *if_i F then G else H*) =
 (((*w* \models *F*) \wedge (*w* \models *G*)) \vee ((\neg (*w* \models *F*) \wedge (*w* \models *H*))))

by (*simp add: itl-def*)

lemma *len-defs* :

($w \models \text{len } n$) =
($\text{nlength } w = n$)

proof

(*simp add: len-d-def*)

show ($w \models (\text{power skip } n)$) = ($\text{nlength } w = n$)

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 chop-defs skip-defs finite-defs*)

(*metis One-nat-def Suc-eq-plus1 add.commute enat.distinct(1) enat-add-sub-same
le-iff-add of-nat-add of-nat-eq-enat*)

qed

qed

lemma *currentval-defs* :

($s \models \$v$) = ($v (\text{nfirst } s)$)

by (*simp add: itl-def*)

lemma *nextval-defs* :

($s \models v\$$) =
(*if* $\text{nlength } s \neq (\text{enat } 0)$ *then* ($v (\text{nnth } s \ 1)$) *else* ($\epsilon \ x. x=x$))

by (*simp add: itl-def*)

lemma *finval-defs* :

($s \models !v$) =
((*if* $\text{nfinite } s$ *then* ($v (\text{nlast } s)$) *else* ($\epsilon \ x. x=x$))
)

by (*simp add: itl-def*)

lemma *penultval-defs* :

($s \models v!$) =
(*if* $\text{nfinite } s$ *then*
(*if* $\text{nlength } s \neq (\text{enat } 0)$ *then* ($v (\text{nnth } s (\text{the-enat}(\text{epred}(\text{nlength } s))))$) *else* ($\epsilon \ x. x=x$)
else ($\epsilon \ x. x=x$)
)

by (*simp add: itl-def*)

lemma *next-assign-defs* :

($s \models v := e$) = ((*if* $\text{nlength } s \neq (\text{enat } 0)$ *then* $v (\text{nnth } s \ 1)$ *else* ($\epsilon \ x. x=x$)) = $e \ s$)

by (*auto simp: itl-def*)

lemma *prev-assign-defs* :

($s \models v =: e$) =
(*if* $\text{nfinite } s$ *then*

```

    (if nlength s ≠ (enat 0) then (v (nnth s (the-enat(epred(nlength s))))) = e s
      else ((ε x. x=x) = e s))
  else True
)
by (simp add: itl-def finite-defs)

lemma always-eqv-defs :
  (s ⊨ v ≈ e) =
  ( (∀ i. i ≤ nlength s → 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 ⊨ v ← e) =
  (if nfinite s then (v (nlast s)) = e s
    else True
  )
by (simp add: itl-def finite-defs)

lemma gets-defs :
  (s ⊨ v gets e) =
  ( (∀ i. i < nlength s → 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-def ntaken-ndropn-nlast)
show ∧i. ∀n na. (enat na ≤ nlength s - enat n →
  enat (n + na) ≤ nlength s →
  na = Suc 0 → v (nnth s (Suc n)) = e (ntaken (Suc 0) (ndropn n s))) ∧
  (¬ enat na ≤ nlength s - enat n →
  enat (n + na) ≤ nlength s →
  nlength s - enat n = enat (Suc 0) →
  v (nnth s (na + n)) = e (ntaken na (ndropn n s))) ⇒
  enat i < nlength s ⇒
  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 ∧n na.
  ∀i. enat i < nlength s → v (nnth s (Suc i)) = e (ntaken (Suc 0) (ndropn i s)) ⇒
  ¬ na ≤ Suc 0 ⇒ enat (n + na) ≤ nlength s ⇒ nlength s - enat n = enat (Suc 0) ⇒
  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 (∀ i. i < nlength s → v (nnth s (Suc i)) = v (nnth s i))
  i ≤ 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 ( $\forall i. i \leq \text{nlength } s \longrightarrow v (\text{nnth } s \ i) = v (\text{nfirst } s)$ )
         ( $i < \text{nlength } s$ )
shows  $v (\text{nnth } s \ (\text{Suc } i)) = v (\text{nnth } s \ i)$ 
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:
( $\forall i. i < \text{nlength } s \longrightarrow v (\text{nnth } s \ (\text{Suc } i)) = v (\text{nnth } s \ i)$ ) =
( $\forall i. i \leq \text{nlength } s \longrightarrow v (\text{nnth } s \ i) = v (\text{nfirst } s)$ )
using stable-defs-helpa[of s v] stable-defs-helpb[of s v]
by blast

```

```

lemma stable-defs:
( $s \models \text{stable } v$ ) =
( $\forall i. i \leq \text{nlength } s \longrightarrow (v (\text{nnth } s \ i)) = (v (\text{nfirst } s)))$ )
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-def 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)
(metis One-nat-def enat.simps(3) enat-add-sub-same enat-ord-simps(1) iless-Suc-eq le-iff-add
less-imp-le one-enat-def plus-1-eSuc(2),
meson iless-Suc-eq less-imp-le,
metis One-nat-def enat.simps(3) enat-add-sub-same enat-ord-simps(1) ile-eSuc nfinite-nlength-enat
one-enat-def plus-1-eSuc(2),
metis One-nat-def antisym-conv2 co.enat.sel(2) diff-le-self enat-add-sub-same enat-ord-code(4)
enat-ord-simps(1) epred-enat iless-Suc-eq one-enat-def plus-1-eSuc(2))
qed

lemma *padded-temporal-assign-defs :*

$(s \models v <\sim e) =$
 $((s \models \text{padded } v) \wedge$
 $(\text{if } n\text{finite } s \text{ then } (v (n\text{last } s)) = e \text{ } s \text{ else } \text{True}))$
 $)$

by (*auto simp add: padded-temp-assign-d-def padded-defs temporal-assign-defs*)

lemma *chop-nfuse-1 :*

$(\exists \sigma 1 \sigma 2. \sigma = \text{nfuse } \sigma 1 \sigma 2 \wedge n\text{finite } \sigma 1 \wedge$
 $(\sigma 1 \models f) \wedge (\sigma 2 \models g) \wedge$
 $(n\text{last } \sigma 1 = n\text{first } \sigma 2)) =$
 $((\exists i. 0 \leq i \wedge i \leq n\text{length } \sigma \wedge (ntaken \ i \ \sigma \models f) \wedge (ndropn \ i \ \sigma \models g)))$

by *auto*

(metis enat-le-plus-same(1) nfuse-nlength ndropn-nfuse nfinite-conv-nlength-enat ntaken-nfuse
the-enat.simps,
metis nfuse-ntaken-ndropn ndropn-nfirst nfinite-ntaken ntaken-nlast)

lemma *chop-nfuse-2 :*

$(\exists \sigma 1 \sigma 2. \sigma = \text{nfuse } \sigma 1 \sigma 2 \wedge n\text{finite } \sigma 1 \wedge$
 $(\sigma 1 \in X) \wedge (\sigma 2 \in Y) \wedge$
 $(n\text{last } \sigma 1 = n\text{first } \sigma 2)) =$
 $(\exists i. i \leq n\text{length } \sigma \wedge (ntaken \ i \ \sigma) \in X \wedge (ndropn \ i \ \sigma) \in Y)$

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:*

$(\sigma \models f;g) = ($
 $(\exists \sigma 1 \sigma 2. \sigma = \text{nfuse } \sigma 1 \sigma 2 \wedge n\text{finite } \sigma 1 \wedge$
 $(\sigma 1 \models f) \wedge (\sigma 2 \models g) \wedge (n\text{last } \sigma 1 = n\text{first } \sigma 2))$
 $\vee (\neg n\text{finite } \sigma \wedge (\sigma \models f))$
 $)$

by (simp add: chop-defs chop-nfuse-1)

lemmas *itl-defs = skip-defs chop-defs infinite-defs finite-defs schop-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 len-defs
currentval-defs nextval-defs finval-defs penultval-defs next-assign-defs prev-assign-defs
always-eqv-defs temporal-assign-defs gets-defs stable-defs padded-defs
padded-temporal-assign-defs*

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 $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge (\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge$

$f(\text{ntaken } (\min na n) \sigma) \wedge g(\text{ndropn } na (\text{ntaken } n \sigma))) \wedge h(\text{ndropn } n \sigma)) \vee$

$\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f(\text{ntaken } n \sigma) \wedge g(\text{ndropn } n \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \sigma))$

proof –

have 1: $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$

$f(\text{ntaken } n \sigma) \wedge$

$((\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g(\text{ntaken } na (\text{ndropn } n \sigma)) \wedge h(\text{ndropn } na (\text{ndropn } n \sigma)))$

\vee

$\neg \text{nfinite } (\text{ndropn } n \sigma) \wedge g(\text{ndropn } n \sigma))) \vee$

$\neg \text{nfinite } \sigma \wedge f \sigma)$

using *assms by auto*

have 2: $\neg \text{nfinite } \sigma \wedge f \sigma \implies$

$((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$

$(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f(\text{ntaken } (\min na n) \sigma) \wedge g(\text{ndropn } na (\text{ntaken } n \sigma)))$

$\wedge h(\text{ndropn } n \sigma)) \vee$

$\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f(\text{ntaken } n \sigma) \wedge g(\text{ndropn } n \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \sigma))$

by *simp*

have 3: $(\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$

$f(\text{ntaken } n \sigma) \wedge$

$((\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g(\text{ntaken } na (\text{ndropn } n \sigma)) \wedge h(\text{ndropn } na (\text{ndropn } n \sigma)))$

\vee

$\neg \text{nfinite } (\text{ndropn } n \sigma) \wedge g(\text{ndropn } n \sigma))) \implies$

$((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$

$(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f(\text{ntaken } (\min na n) \sigma) \wedge g(\text{ndropn } na (\text{ntaken } n \sigma)))$

$\wedge h(\text{ndropn } n \sigma)) \vee$

$\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f(\text{ntaken } n \sigma) \wedge g(\text{ndropn } n \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \sigma))$

proof –

assume 4: $(\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$

$f(\text{ntaken } n \sigma) \wedge$

$((\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n \sigma)))$
 \vee
 $\neg \text{nfinite } (\text{ndropn } n \sigma) \wedge g (\text{ndropn } n \sigma)))$
obtain n **where** 5: $\text{enat } n \leq \text{nlength } \sigma \wedge$
 $f (\text{ntaken } n \sigma) \wedge$
 $((\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n \sigma)))$
 \vee
 $\neg \text{nfinite } (\text{ndropn } n \sigma) \wedge g (\text{ndropn } n \sigma))$
using 4 **by** *auto*
have 6: $\text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ntaken } n \sigma)$
using 5 **by** *auto*
have 7: $\neg \text{nfinite } (\text{ndropn } n \sigma) \wedge g (\text{ndropn } n \sigma) \implies$
 $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \sigma)))$
 $\wedge h (\text{ndropn } n \sigma)) \vee$
 $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ntaken } n \sigma) \wedge g (\text{ndropn } n \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \sigma))$
using 6 *nfinite-ndropn-a* **by** *blast*
have 8: $(\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n$
 $\sigma))) \implies$
 $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \sigma)))$
 $\wedge h (\text{ndropn } n \sigma)) \vee$
 $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ntaken } n \sigma) \wedge g (\text{ndropn } n \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \sigma))$
proof –
assume 9: $(\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \sigma)) \wedge h (\text{ndropn } na$
 $(\text{ndropn } n \sigma)))$
show $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \sigma)))$
 $\wedge h (\text{ndropn } n \sigma)) \vee$
 $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ntaken } n \sigma) \wedge g (\text{ndropn } n \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \sigma))$
proof –
obtain na **where** 10: $\text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \sigma)) \wedge$
 $h (\text{ndropn } na (\text{ndropn } n \sigma))$
using 9 **by** *auto*
have 11: $h (\text{ndropn } (na+n) \sigma)$
by (*metis* 10 *add commute ndropn-ndropn*)
have 12: $na+n \leq \text{nlength } \sigma$
by (*metis* 10 6 *Groups.add-ac(2) dual-order.strict-implies-order enat.simps(3)*
 $\text{enat-add-sub-same enat-less-enat-plusI2 le-iff-add not-le-imp-less}$
 $\text{order.not-eq-order-implies-strict plus-enat-simps(1)})$
have 13: $g (\text{ndropn } n (\text{ntaken } (n+na) \sigma))$
by (*metis* 10 12 *add commute ntaken-ndropn-swap plus-enat-simps(1)*)
have 14: $f (\text{ntaken } (\text{min } n \ (n+na)) \sigma)$
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 $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f(\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g(\text{ndropn } na \ (\text{ntaken } n \ \sigma)))$
 $\wedge h(\text{ndropn } n \ \sigma)) \vee$
 $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f(\text{ntaken } n \ \sigma) \wedge g(\text{ndropn } n \ \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \ \sigma))$
 shows $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $f(\text{ntaken } n \ \sigma) \wedge$
 $((\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g(\text{ntaken } na \ (\text{ndropn } n \ \sigma)) \wedge h(\text{ndropn } na \ (\text{ndropn } n \ \sigma)))$
 \vee
 $\neg \text{nfinite } (\text{ndropn } n \ \sigma) \wedge g(\text{ndropn } n \ \sigma))) \vee$
 $\neg \text{nfinite } \sigma \wedge f \ \sigma)$
 proof –
 have 1: $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f(\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g(\text{ndropn } na \ (\text{ntaken } n \ \sigma)))$
 $\wedge h(\text{ndropn } n \ \sigma)) \vee$
 $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f(\text{ntaken } n \ \sigma) \wedge g(\text{ndropn } n \ \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \ \sigma))$
 using assms by auto
 have 2: $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f(\text{ntaken } n \ \sigma) \wedge g(\text{ndropn } n \ \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \ \sigma)$
 \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
 $\neg \text{nfinite } (\text{ndropn } n \ \sigma) \wedge g(\text{ndropn } n \ \sigma))) \vee$
 $\neg \text{nfinite } \sigma \wedge f \ \sigma)$
 using nfinite-ndropn by blast
 have 3: $(\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f(\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g(\text{ndropn } na \ (\text{ntaken } n \ \sigma)))$
 $\wedge h(\text{ndropn } n \ \sigma)) \implies$
 $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $f(\text{ntaken } n \ \sigma) \wedge$
 $((\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g(\text{ntaken } na \ (\text{ndropn } n \ \sigma)) \wedge h(\text{ndropn } na \ (\text{ndropn } n \ \sigma)))$
 \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: $na \leq n \wedge \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \ \sigma))$
using 6 **by** *auto*
have 8: $na \leq \text{nlength } \sigma$
by (*simp add: 7*)
have 9: $n - na \leq \text{nlength } \sigma - na$
by (*metis 5 enat-minus-mono1 idiff-enat-enat*)
have 10: $f (\text{ntaken } na \ \sigma)$
using 7 **by** *linarith*
have 11: $g (\text{ntaken } (n - na) (\text{ndropn } na \ \sigma))$
by (*simp add: 5 7 ntaken-ndropn-swap*)
have 12: $h (\text{ndropn } ((n - na) + na) \ \sigma)$
by (*simp add: 5 7*)
have 13: $h (\text{ndropn } (n - na) (\text{ndropn } na \ \sigma))$
by (*metis 12 add commute ndropn-ndropn*)
show ?thesis
using 10 11 13 7 9 **by** *auto*
qed
qed
show ?thesis
using 2 3 *assms* **by** *blast*
qed

lemma *ChopAssocSemHelp:*

$((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $f (\text{ntaken } n \ \sigma) \wedge$
 $(\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \ \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n \ \sigma)))$
 \vee
 $\neg \text{nfinite } (\text{ndropn } n \ \sigma) \wedge g (\text{ndropn } n \ \sigma))) \vee$
 $\neg \text{nfinite } \sigma \wedge f \ \sigma) =$
 $((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f (\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g (\text{ndropn } na (\text{ntaken } n \ \sigma)))$
 $\wedge h (\text{ndropn } n \ \sigma)) \vee$
 $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ntaken } n \ \sigma) \wedge g (\text{ndropn } n \ \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \ \sigma))$
using *ChopAssocSemHelpa[of σ f g h]* *ChopAssocSemHelpb[of σ f g h]* **by** *blast*

lemma *ChopAssocSemHelp1:*

$((\sigma \models f ; (g ; h)) = ((\sigma \models (f;g);h))$
proof –
have $(\sigma \models f ; (g ; h)) = ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $f (\text{ntaken } n \ \sigma) \wedge$
 $(\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge g (\text{ntaken } na (\text{ndropn } n \ \sigma)) \wedge h (\text{ndropn } na (\text{ndropn } n \ \sigma)))$
 \vee

```

       $\neg \text{nfinite } (\text{ndropn } n \ \sigma) \wedge g \ (\text{ndropn } n \ \sigma))) \vee$ 
       $\neg \text{nfinite } \sigma \wedge f \ \sigma)$ 
by (simp add: chop-defs)
also have ... =
      ( $(\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$ 
        ( $\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge f \ (\text{ntaken } (\text{min } na \ n) \ \sigma) \wedge g \ (\text{ndropn } na \ (\text{ntaken } n \ \sigma)))$ 
         $\wedge h \ (\text{ndropn } n \ \sigma)) \vee$ 
         $\neg \text{nfinite } \sigma \wedge ((\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge f \ (\text{ntaken } n \ \sigma) \wedge g \ (\text{ndropn } n \ \sigma)) \vee \neg \text{nfinite } \sigma \wedge f \ \sigma))$ 
      using ChopAssocSemHelp[of  $\sigma \ f \ g \ h$ ] by blast
also have ... =
      ( $\sigma \models (f;g);h$ ) by (simp add: chop-defs)
finally show ( $\sigma \models f ; (g ; h)$ ) = ( $\sigma \models (f;g);h$ ) .
qed

```

lemma *ChopAssocSem*:

($\sigma \models f ; (g ; h) = (f;g);h$)
using *ChopAssocSemHelp1*[*of* $f \ g \ h \ \sigma$] **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*)
 (*metis ndropn-0 nlength-eq-enat-nfiniteD zero-enat-def zero-le*)

4.5.5 ChopEmpty

lemma *ChopEmptySem*:

($\sigma \models f;\text{empty} = f$)
by (*simp add: chop-defs empty-defs min-def*)
 (*metis cancel-comm-monoid-add-class.diff-cancel enat-diff-cancel-left idiff-enat-enat*
nfinite-nlength-enat ntaken-all order-refl zero-enat-def)

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:*

$$(\sigma \models bi (f \longrightarrow f1) \wedge \Box (g \longrightarrow g1) \longrightarrow f;g \longrightarrow f1;g1)$$

by (*simp add: bi-defs always-defs chop-defs*)

fastforce

4.5.9 BoxInduct

lemma *box-induct-help-1 :*

$$\begin{aligned} \bigwedge j. \forall n. enat\ n \leq nlength\ \sigma \longrightarrow f\ (ndropn\ n\ \sigma) \longrightarrow \\ nlength\ \sigma - enat\ n = (enat\ 0) \vee f\ (ndropn\ (Suc\ 0)\ (ndropn\ n\ \sigma)) \implies \\ f\ \sigma \implies \\ enat\ j \leq nlength\ \sigma \implies \\ f\ (ndropn\ j\ \sigma) \end{aligned}$$

proof –

fix *j*

show $\forall n. enat\ n \leq nlength\ \sigma \longrightarrow f\ (ndropn\ n\ \sigma) \longrightarrow$

$$nlength\ \sigma - enat\ n = (enat\ 0) \vee f\ (ndropn\ (Suc\ 0)\ (ndropn\ n\ \sigma)) \implies$$

$$f\ \sigma \implies$$

$$enat\ j \leq nlength\ \sigma \implies$$

$$f\ (ndropn\ j\ \sigma)$$

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:*

$$(\sigma \models \Box (f \longrightarrow wnext\ f) \wedge f \longrightarrow \Box f)$$

proof

(*auto simp add: always-defs wnext-defs*)

show $\bigwedge n. \forall n. enat\ n \leq nlength\ \sigma \longrightarrow f\ (ndropn\ n\ \sigma) \longrightarrow$

$$nlength\ \sigma - enat\ n = enat\ 0 \vee f\ (ndropn\ (Suc\ 0)\ (ndropn\ n\ \sigma)) \implies$$

$$f\ \sigma \implies$$

$$enat\ n \leq nlength\ \sigma \implies$$

$$f\ (ndropn\ n\ \sigma)$$

using *box-induct-help-1* **by** *blast*

qed

4.5.10 ChopStarEqv

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 *powersem1*:

$(\sigma \models (\exists k. \text{power } f k) = (\text{empty} \vee (\exists k. \text{power } f (\text{Suc } k))))$

proof *auto*

show $\bigwedge x. \sigma \models (\text{power } f x) \implies \forall k. \neg (\sigma \models (f \wedge \text{finite}); \text{power } f k) \implies \sigma \models \text{empty}$

by (*metis not0-implies-Suc pow-0 pow-Suc*)

show $\sigma \models \text{empty} \implies \exists x. \sigma \models (\text{power } f x)$

by (*metis pow-0*)

show $\bigwedge k. \sigma \models ((f \wedge \text{finite}); \text{power } f k) \implies \exists x. \sigma \models (\text{power } f x)$

by (*metis pow-Suc*)

qed

lemma *spowersem1*:

$(\sigma \models (\exists k. \text{spower } f k) = (\text{empty} \vee (\exists k. \text{spower } f (\text{Suc } k))))$

proof *auto*

show $\bigwedge x. \sigma \models (\text{spower } f x) \implies \forall k. \neg (\sigma \models f \frown \text{spower } f k) \implies \sigma \models \text{empty}$

by (*metis not0-implies-Suc spow-0 spow-Suc*)

show $\sigma \models \text{empty} \implies \exists x. \sigma \models (\text{spower } f x)$

by (*metis spow-0*)

show $\bigwedge k. \sigma \models (f \frown \text{spower } f k) \implies \exists x. \sigma \models (\text{spower } f x)$

by (*metis spow-Suc*)

qed

lemma *powersem*:

$\vdash (\exists k. \text{power } f k) = (\text{empty} \vee (f \wedge \text{finite}); (\exists k. (\text{power } f k)))$

proof –

have 1: $\vdash (\exists k. \text{power } f k) = (\text{empty} \vee (\exists k. \text{power } f (\text{Suc } k)))$

using *powersem1* **by** *blast*

have 2: $\vdash (\exists k. \text{power } f (\text{Suc } k)) = (\exists k. (f \wedge \text{finite}); \text{power } f k)$

by *simp*

have 3: $\vdash (\exists k. (f \wedge \text{finite}); (\text{power } f k)) = (f \wedge \text{finite}); (\exists k. (\text{power } f k))$

using *ChopExist* **by** *blast*

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

qed

lemma *spowersem*:

$\vdash (\exists k. \text{spower } f \ k) = (\text{empty} \vee f \frown (\exists k. (\text{spower } f \ k)))$

proof –

have 1: $\vdash (\exists k. \text{spower } f \ k) = (\text{empty} \vee (\exists k. \text{spower } f \ (\text{Suc } k)))$

using *spowersem1* **by** *blast*

have 2: $\vdash (\exists k. \text{spower } f \ (\text{Suc } k)) = (\exists k. f \frown \text{spower } f \ k)$

by *simp*

have 3: $\vdash (\exists k. f \frown (\text{spower } f \ k)) = f \frown (\exists k. (\text{spower } f \ k))$

using *SChopExist* **by** *blast*

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

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 *PowerstarEqvSem*:

$(\sigma \models (\text{powerstar } f)) = (\text{empty} \vee f;(\text{powerstar } f))$

proof –

have 1: $(\sigma \models (\text{powerstar } f)) =$

$(\sigma \models (\exists k. \text{power } f \ k);(\text{empty} \vee f \wedge \text{inf}))$

by (*simp add: powerstar-d-def*)

have 2: $(\sigma \models (\exists k. \text{power } f \ k);(\text{empty} \vee f \wedge \text{inf})) =$

$(\sigma \models (\text{empty} \vee (f \wedge \text{finite});(\exists k. (\text{power } f \ k))));(\text{empty} \vee f \wedge \text{inf}))$

using *powersem* **by** (*metis inteq-reflection*)

have 3: $(\sigma \models (\text{empty} \vee (f \wedge \text{finite});(\exists k. (\text{power } f \ k))));(\text{empty} \vee f \wedge \text{inf})) =$

$(\sigma \models \text{empty};(\text{empty} \vee (f \wedge \text{inf})) \vee$

$((f \wedge \text{finite});(\exists k. (\text{power } f \ k))));(\text{empty} \vee f \wedge \text{inf}))$

by (*auto simp add: chop-defs*)

have 4: $(\sigma \models \text{empty};(\text{empty} \vee (f \wedge \text{inf})) \vee$

$((f \wedge \text{finite});(\exists k. (\text{power } f \ k))));(\text{empty} \vee f \wedge \text{inf})) =$

$(\sigma \models (\text{empty} \vee (f \wedge \text{inf}) \vee ((f \wedge \text{finite});(\exists k. (\text{power } f \ k))));(\text{empty} \vee f \wedge \text{inf})))$

using *PowerstarEqvSemhelp1*

by (*metis (mono-tags, lifting) inteq-reflection unl-lift2*)

have 5: $(\sigma \models (\text{empty} \vee (f \wedge \text{inf}) \vee ((f \wedge \text{finite});(\exists k. (\text{power } f \ k))));(\text{empty} \vee f \wedge \text{inf}))) =$

$(\sigma \models (\text{empty} \vee (f \wedge \text{inf});((\exists k. (\text{power } f \ k));(\text{empty} \vee f \wedge \text{inf})) \vee$

$((f \wedge \text{finite});(\exists k. (\text{power } f \ k))));(\text{empty} \vee f \wedge \text{inf})))$

using *PowerstarEqvSemhelp2*

by (*metis (mono-tags, lifting) inteq-reflection*)

have 51: $(\sigma \models ((f \wedge \text{finite});(\exists k. (\text{power } f \ k))));(\text{empty} \vee f \wedge \text{inf})) =$

$(f \wedge \text{finite});((\exists k. (\text{power } f \ k));(\text{empty} \vee f \wedge \text{inf})))$

by (auto simp add: ChopAssocSemHelp1)
 have 6: $(\sigma \models (\text{empty} \vee (f \wedge \text{inf}); (\exists k. (\text{power } f \ k)); (\text{empty} \vee f \wedge \text{inf})) \vee$
 $((f \wedge \text{finite}); (\exists k. (\text{power } f \ k)); (\text{empty} \vee f \wedge \text{inf}))) =$
 $(\sigma \models (\text{empty} \vee (f \wedge \text{inf}); (\exists k. (\text{power } f \ k)); (\text{empty} \vee f \wedge \text{inf})) \vee$
 $(f \wedge \text{finite}); (\exists k. (\text{power } f \ k)); (\text{empty} \vee f \wedge \text{inf})))$

 using 51 by auto
 have 7 : $(\sigma \models (\text{empty} \vee (f \wedge \text{inf}); (\exists k. (\text{power } f \ k)); (\text{empty} \vee f \wedge \text{inf})) \vee$
 $(f \wedge \text{finite}); (\exists k. (\text{power } f \ k)); (\text{empty} \vee f \wedge \text{inf}))) =$
 $(\sigma \models (\text{empty} \vee f; (\exists k. (\text{power } f \ k)); (\text{empty} \vee f \wedge \text{inf}))))$
 using PowerstarEqvSemhelp3 by fastforce
 have 8: $(\sigma \models (\text{empty} \vee f; (\exists k. (\text{power } f \ k)); (\text{empty} \vee f \wedge \text{inf})))) =$
 $(\sigma \models (\text{empty} \vee f; (\text{powerstar } f)))$
 by (simp add: powerstar-d-def)
 from 1 2 3 4 5 6 7 8 show ?thesis by fastforce
 qed

lemma FPowerstarEqvSem:

$(\sigma \models (\text{fpowerstar } f) = (\text{empty} \vee (f \wedge \text{finite}); (\text{fpowerstar } f)))$
proof –
 have 1: $(\sigma \models (\text{fpowerstar } f)) =$
 $(\sigma \models (\exists k. \text{power } f \ k))$
 by (simp add: fpowerstar-d-def)
 have 2: $(\sigma \models (\exists k. \text{power } f \ k)) =$
 $(\sigma \models (\text{empty} \vee (f \wedge \text{finite}); (\exists k. (\text{power } f \ k))))$
 using powersem by (metis inteq-reflection)
 from 1 2 show ?thesis by (simp add: fpowerstar-d-def)
 qed

lemma SPowerstarEqvSem:

$(\sigma \models (\text{spowerstar } f) = (\text{empty} \vee f \frown (\text{spowerstar } f)))$
proof –
 have 1: $(\sigma \models (\text{spowerstar } f)) =$
 $(\sigma \models (\exists k. \text{spower } f \ k))$
 by (simp add: spowerstar-d-def)
 have 2: $(\sigma \models (\exists k. \text{spower } f \ k)) =$
 $(\sigma \models (\text{empty} \vee f \frown (\exists k. (\text{spower } f \ k))))$
 using powersem by (metis inteq-reflection)
 from 1 2 show ?thesis by (simp add: spowerstar-d-def)
 qed

lemma powerchopsem:

$\vdash (\exists k. \text{power } (f \wedge \text{more}) \ k) =$
 $(\text{empty} \vee ((f \wedge \text{more}) \wedge \text{finite}); (\exists k. (\text{power } (f \wedge \text{more}) \ k)))$
 $)$
 using powersem by auto

lemma spowerchopsem:

$\vdash (\exists k. \text{spower } (f \wedge \text{more}) \ k) =$

```

      ( empty  $\vee$  (f  $\wedge$  more)  $\frown$  ( $\exists$  k. (spower (f  $\wedge$  more) k))
    )
using spowersem by auto

```

lemma *ChopstarEqvSem*:

```

  ( $\sigma \models f^* =$  (empty  $\vee$  (f  $\wedge$  more);  $f^*$ ) )
by (metis PowerstarEqvSem chopstar-d-def)

```

lemma *SChopstarEqvSem*:

```

  ( $\sigma \models$  (schopstar f) = (empty  $\vee$  (f  $\wedge$  more)  $\frown$  (schopstar f) ))
by (metis SPowerstarEqvSem schopstar-d-def)

```

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 Omega

lemma *FMoreSem* [*simp*, *mono*]:

```

  (w  $\models$  inf  $\wedge$  (f  $\wedge$  fmore); g) =
  ( $\neg$  nfinite w  $\wedge$  ( $\exists$  n. f ( (ntaken n w))  $\wedge$  n > 0  $\wedge$  g (ndropn n w)))
by (simp add: itl-defs)
  (metis enat.inject le-cases min.orderE neq0-conv nfinite-ntaken ntaken-all)

```

coinductive *omega-d* :: ('a::world) formula \Rightarrow 'a formula

for *F* **where**

```

  (s  $\models$  inf  $\wedge$  (F  $\wedge$  fmore); (omega-d F))  $\Longrightarrow$  (s  $\models$  (omega-d F))

```

syntax

```

-omega-d :: lift  $\Rightarrow$  lift      ((-  $\omega$ ) [85] 85)

```

syntax (*ASCII*)

```

-omega-d :: lift  $\Rightarrow$  lift      ((omega -) [85] 85)

```

translations

```

-omega-d       $\equiv$  CONST omega-d

```

lemma *OmegaIntros*:

```

   $\vdash$  inf  $\wedge$  (f  $\wedge$  fmore); (omega f)  $\longrightarrow$  (omega f)
using omega-d.intros by fastforce

```

lemma *OmegaCases*:

$(w \models (\text{omega } F)) \implies$

$(w \models \text{inf} \wedge (F \wedge \text{fmore}); (\text{omega } F) \implies P) \implies P$

using *omega-d.cases*[of *F w P*] **by** *auto*

lemma *OmegaUnrollSem*:

$(s \models (\text{omega } f)) = (s \models (f \wedge \text{fmore}); (\text{omega } f))$

using *omega-d.cases*[of *f*]

by (*metis chop-defs infinite-defs nfinite-ndropn-a omega-d.simps unl-lift2*)

lemma *OmegaCoinductSem*:

assumes $\bigwedge x. X x \implies x \models \text{inf} \wedge (F \wedge \text{fmore}); (X \vee \text{omega } F)$

shows $x \models X \longrightarrow \text{omega } F$

using *assms omega-d.coinduct*[of *X x F*]

by (*auto simp add: chop-defs*)

lemma *OmegaCoinduct*:

assumes $\vdash X \longrightarrow \text{inf} \wedge (F \wedge \text{fmore}); (X \vee (\text{omega } F))$

shows $\vdash X \longrightarrow (\text{omega } F)$

using *assms OmegaCoinductSem*[of *X F*]

by (*simp add: Valid-def*)

lemma *OmegaWeakCoinductSem*:

assumes $\bigwedge x. X x \implies x \models \text{inf} \wedge (F \wedge \text{fmore}); X$

shows $x \models X \longrightarrow \text{omega } F$

using *assms omega-d.coinduct*[of *X x F*]

by (*auto simp add: chop-defs*)

lemma *OmegacoWeakCoinduct*:

assumes $\vdash X \longrightarrow \text{inf} \wedge (F \wedge \text{fmore}); X$

shows $\vdash X \longrightarrow (\text{omega } F)$

using *assms OmegaWeakCoinductSem*[of *X F*]

by (*simp add: Valid-def*)

lemma *OmegaInducthelpsem*:

assumes $\neg \text{nfinite } w$

$g \ w$

$\forall n. \text{enat } n \leq \text{nlength } w \longrightarrow$

$g \ (\text{ndropn } n \ w) \longrightarrow$

$(\exists na. \text{enat } na \leq \text{nlength } w - \text{enat } n \longrightarrow$

$f \ (\text{ntaken } na \ (\text{ndropn } n \ w)) \wedge 0 < na \wedge$

$g \ (\text{ndropn } (n + na) \ w) \wedge \text{enat } na \leq \text{nlength } w - \text{enat } n \)$

shows $\exists n. (\text{enat } n \leq \text{nlength } w \longrightarrow$

$f \ (\text{ntaken } n \ w) \wedge$

$0 < n \wedge$

$g \ (\text{ndropn } n \ w) \wedge$

$(\forall na. \text{enat } na \leq \text{nlength } w - \text{enat } n \longrightarrow$

$g \ (\text{ndropn } (n + na) \ w) \longrightarrow$

$(\exists nb. (\text{enat } nb \leq \text{nlength } w - \text{enat } (n + na) \longrightarrow$

$$\begin{aligned}
& f \text{ (ntaken nb (ndropn (n + na) w)) } \wedge 0 < nb \wedge \\
& g \text{ (ndropn (n + na + nb) w)) } \wedge \\
& \text{enat nb} \leq \text{nlength w} - \text{enat (n + na)) } \wedge \text{enat n} \leq \text{nlength w}
\end{aligned}$$

proof –

have 1: $(\exists na. \text{enat na} \leq \text{nlength w} - \text{enat 0} \longrightarrow$

$$\begin{aligned}
& f \text{ (ntaken na (ndropn 0 w)) } \wedge 0 < na \wedge \\
& \text{enat 0} < \text{nlength w} - \text{enat 0} \wedge g \text{ (ndropn (0 + na) w))}
\end{aligned}$$

by (metis assms(1) assms(2) assms(3) i0-less idiff-enat-0-right ndropn-0
nlength-eq-enat-nfiniteD zero-enat-def)

obtain n1 **where** 2: $\text{enat n1} \leq \text{nlength w} - \text{enat 0} \wedge$

$$\begin{aligned}
& f \text{ (ntaken n1 (ndropn 0 w)) } \wedge \\
& 0 < n1 \wedge \text{enat 0} < \text{nlength w} - \text{enat 0} \wedge g \text{ (ndropn (0 + n1) w)}
\end{aligned}$$

using 1 **by** (metis assms(1) enat-ile idiff-enat-0-right le-cases nlength-eq-enat-nfiniteD)

have 3: $n1 \leq \text{nlength w} \longrightarrow f \text{ (ntaken n1 w)}$

using 2 **by** auto

have 4: $f \text{ (ntaken n1 w)}$

using 2 **by** auto

have 5: $n1 \leq \text{nlength w} \longrightarrow 0 < n1$

by (simp add: 2)

have 6: $\text{enat 0} < \text{nlength w}$

using 2 **by** auto

have 7: $n1 \leq \text{nlength w} \longrightarrow g \text{ (ndropn n1 w)}$

using 2 **by** auto

have 8: $n1 \leq \text{nlength w} \longrightarrow$

$$\begin{aligned}
& (\forall na. \text{enat na} \leq \text{nlength w} - \text{enat n1} \longrightarrow \\
& g \text{ (ndropn (n1 + na) w)} \longrightarrow \\
& (\exists nb. (\text{enat nb} \leq \text{nlength w} - \text{enat (n1 + na)} \longrightarrow \\
& f \text{ (ntaken nb (ndropn (n1 + na) w)) } \wedge 0 < nb \wedge g \text{ (ndropn (n1 + na + nb) w)) } \wedge \\
& \text{enat nb} \leq \text{nlength w} - \text{enat (n1 + na)) } \wedge
\end{aligned}$$

using assms

by (cases nfinite w)

(blast,

$$\begin{aligned}
& \text{metis enat-add-sub-same enat-le-plus-same(1) enat-le-plus-same(2) enat-ord-code(4)} \\
& \text{enat-the-enat nlength-eq-enat-nfiniteD not-less-iff-gr-or-eq order.not-eq-order-implies-strict}
\end{aligned}$$

have 9: $n1 \leq \text{nlength w}$

using 2 **by** auto

have 10: $(n1 \leq \text{nlength w} \longrightarrow$

$$\begin{aligned}
& f \text{ (ntaken n1 w) } \wedge 0 < n1 \wedge g \text{ (ndropn n1 w) } \wedge \\
& (\forall na. \text{enat na} \leq \text{nlength w} - \text{enat n1} \longrightarrow
\end{aligned}$$

$$\begin{aligned}
& g \text{ (ndropn (n1 + na) w)} \longrightarrow \\
& (\exists nb. (\text{enat nb} \leq \text{nlength w} - \text{enat (n1 + na)} \longrightarrow
\end{aligned}$$

$$\begin{aligned}
& f \text{ (ntaken nb (ndropn (n1 + na) w)) } \wedge 0 < nb \wedge g \text{ (ndropn (n1 + na + nb) w)) } \wedge \\
& \text{enat nb} \leq \text{nlength w} - \text{enat (n1 + na)) } \wedge n1 \leq \text{nlength w}
\end{aligned}$$

using 2 4 7 8 9 **by** blast

show ?thesis **using** 10 **by** blast

qed

lemma OmegaInducthelp:

$$\begin{aligned}
& \vdash (\text{inf} \wedge g \wedge \Box(g \longrightarrow (f \wedge \text{fmore});g)) \\
& \longrightarrow
\end{aligned}$$

```

      inf ∧ (f ∧ fmore);( (inf ∧ g ∧ □(g → (f ∧ fmore);g)) )
proof –
  have 1: ⊢(inf ∧ g ∧ □(g → (f ∧ fmore);g)) → inf
    by (simp add: intI)
  have 4: ⊢(inf ∧ g ∧ □(g → (f ∧ fmore);g)) → (f ∧ fmore);(inf ∧ g ∧ □(g → (f ∧ fmore);g))
    using OmegaInducthelpsem[of - g f]
    by (auto simp add: Valid-def itl-defs ndropn-ndropn
      zero-enat-def min-def dest:nlength-eq-enat-nfiniteD)
      metis
  have 5: ⊢(inf ∧ g ∧ □(g → (f ∧ fmore);g)) →
      inf ∧ (f ∧ fmore);(inf ∧ g ∧ □(g → (f ∧ fmore);g))
    using 1 4 by auto
  show ?thesis using 5 by blast
qed

```

4.8 Temporal Quantifiers

definition *exist-state-d* :: ('a statefun ⇒ temporal) ⇒ temporal (**binder** Eex 10)
where *exist-state-d* F ≡ (λs. (∃ x. s ⊨ F x))

syntax

-Eex :: [idts, lift] ⇒ lift ((∃∃∃ -./ -) [0,10] 10)

translations

-Eex v A == Eex v. A

definition *forall-state-d* :: ('a statefun ⇒ temporal) ⇒ temporal (**binder** Aall 10)
where *forall-state-d* F ≡ LIFT(¬(∃∃ x. ¬(F x)))

syntax

-Aall :: [idts, lift] ⇒ lift ((∃∀∀ -./ -) [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 [8]). 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* :
 $\vdash f;empty = f$
using *ChopEmptySem Valid-def* **by** *blast*

lemma *StateImpBi* :
 $\vdash init\ f \longrightarrow bi\ (init\ f)$
using *StateImpBiSem Valid-def* **by** *blast*

lemma *NextImpNotNextNot* :
 $\vdash \bigcirc f \longrightarrow \neg (\bigcirc (\neg f))$
using *NextImpNotNextNotSem Valid-def* **by** *blast*

lemma *BiBoxChopImpChop* :

$\vdash bi (f \longrightarrow f1) \wedge \Box(g \longrightarrow g1) \longrightarrow f;g \longrightarrow f1;g1$
using *BiBoxChopImpChopSem Valid-def* **by** *blast*

lemma *BoxInduct* :
 $\vdash \Box (f \longrightarrow wnext f) \wedge f \longrightarrow \Box f$
using *BoxInductSem Valid-def* **by** *blast*

lemma *ChopstarEqv* :
 $\vdash f^* = (empty \vee (f \wedge more); f^*)$
using *ChopstarEqvSem Valid-def* **by** *blast*

lemma *OmegaUnroll*:
 $\vdash f^\omega = (f \wedge fmore);f^\omega$
by (*simp add: OmegaUnrollSem intI*)

lemma *OmegaInduct*:
 $\vdash (inf \wedge g \wedge \Box(g \longrightarrow (f \wedge fmore);g)) \longrightarrow omega f$
by (*simp add: OmegaInducthelp OmegacoWeakCoinduct*)

5.3 Additional Lemmas

The following is again from [4, 2] but adapted for our need.

lemma *int-eq-true*:
assumes $\vdash P$
shows $\vdash P = \#True$
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 *EEIxI* :
 $\vdash F\ y \longrightarrow (\exists \exists\ x.\ F\ x)$
by (*auto simp add: exist-state-d-def Valid-def*)

lemma *EExE*:
assumes $\bigwedge x. \vdash F\ x \longrightarrow G$
shows $\vdash (\exists \exists\ x.\ F\ x) \longrightarrow G$
using *assms* **by** (*metis (mono-tags, lifting) Valid-def exist-state-d-def unl-lift2*)

lemma *EExVal*:
 $((\ w) \models (\exists \exists\ x.\ F\ x)) =$
 $(\exists\ x\ (val :: 'a\ nellist). (\ (val = (nmap\ x\ w) \wedge ((\ w) \models F\ x))))$
by (*simp add: exist-state-d-def*)

lemma *AAxDef*:
 $\vdash (\forall \forall\ x.\ F\ x) = (\neg(\exists \exists\ x.\ \neg(F\ x)))$
by (*simp add: Valid-def forall-state-d-def exist-state-d-def*)

lemma *ExEqvRule*:
assumes $\bigwedge x. \vdash (f\ x) = (g\ x)$
shows $\vdash (\exists\ x.\ f\ x) = (\exists\ x.\ g\ x)$
using *assms* **by** *fastforce*

5.5 Lemmas about *current-val*

lemma *current-const*: $\vdash \$(\#c) = \#c$
by (*simp add: current-val-d-def intI*)

lemma *current-fun1*: $\vdash \$ (f \langle x \rangle) = f \langle \$x \rangle$
by (*simp add: current-val-d-def intI*)

lemma *current-fun2*: $\vdash \$ (f \langle x, y \rangle) = f \langle \$x, \$y \rangle$
by (*auto simp: current-val-d-def intI*)

lemma *current-fun3*: $\vdash \$ (f \langle x, y, z \rangle) = f \langle \$x, \$y, \$z \rangle$
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 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 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 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 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 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 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 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 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 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 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 le-numeral-extra(3) nlast-NNil ntaken-0 ntaken-all 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 [7] 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* :

$\vdash ((init\ f) \wedge (init\ g)) = init(f \wedge g)$
 $\vdash (\neg (init\ f)) = init(\neg f)$
 $\vdash ((init\ f) \vee (init\ g)) = init(f \vee g)$
 $\vdash init\ \#True$

by (*auto simp: itl-defs*)

lemma *Finprop* :

$\vdash ((\#True;(f \wedge empty)) \wedge (\#True;(g \wedge empty))) = (\#True;((f \wedge g) \wedge empty))$
 $\vdash ((\#True;(f \wedge empty)) \vee (\#True;(g \wedge empty))) = (\#True;((f \vee g) \wedge empty))$
 $\vdash (\#True;((\#True) \wedge empty))$
 $\vdash finite \longrightarrow (\neg (\#True;(f \wedge empty))) = (\#True;(\neg f \wedge empty))$
 $\vdash (\neg (\#True;(f \wedge empty))) = ((\#True;(\neg f \wedge empty)) \wedge finite)$

using *nfinite-nlength-enat*

by (*auto simp: finalt-defs finite-defs,*

auto simp add: itl-defs nfinite-nlength-enat, force)

6.3 finite and inf properties

lemma *EmptyImpFinite*:

$\vdash empty \longrightarrow finite$

using *nlength-eq-enat-nfiniteD* **by** (*auto simp add: itl-defs*)

lemma *SkipChopFiniteImpFinite*:

$\vdash skip;finite \longrightarrow finite$

using *nfinite-ndropn nlength-eq-enat-nfiniteD*

by (*simp add: Valid-def itl-defs, force*)

lemma *FiniteChopSkipImpFinite*:

$\vdash finite;skip \longrightarrow finite$

using *nlength-eq-enat-nfiniteD*

by (*simp add: Valid-def itl-defs, force*)

lemma *FiniteChopSkipImpMore*:

$\vdash finite;skip \longrightarrow more$

using *nlength-eq-enat-nfiniteD*

by (*simp add: Valid-def itl-defs, force*)

lemma *FiniteAndMoreImpFiniteChopSkip*:

$\vdash finite \wedge more \longrightarrow finite;skip$

proof (*auto simp add: Valid-def itl-defs*)

fix *w* :: 'a nellist

assume *a1*: *nlength w* \neq *enat 0*

assume *a2*: *nfinite w*

have *f3*: $\forall n\ na. \neg (n::nat) \leq na \vee n - na = 0$

using *diff-is-0-eq'* **by** *satx*

obtain $nn :: 'a\ nelist \Rightarrow nat$ **where**
 $f4: nlength\ w = enat\ (nn\ w)$
using $a2$ **by** (*meson nfinite-nlength-enat*)
have $\forall n\ na. \neg (n::nat) \leq na \vee na - (na - n) = n$
using *diff-diff-cancel* **by** *satx*
then show $\exists n. enat\ n \leq nlength\ w \wedge nlength\ w - enat\ n = enat\ (Suc\ 0)$
using $f4\ f3\ a1$
by (*metis Suc-diff-eq-diff-pred Suc-diff-le diff-le-self enat-ord-simps(1) gr-zeroI idiff-enat-enat order-refl*)
qed

lemma *FiniteChopSkipEqvFiniteAndMore:*

$\vdash finite;skip = (finite \wedge more)$

by (*simp add: FiniteAndMoreImpFiniteChopSkip FiniteChopSkipImpFinite FiniteChopSkipImpMore Prop12 int-iffI*)

lemma *FiniteChopSkipEqvSkipChopFinite:*

$\vdash finite;skip = skip;finite$

by (*auto simp add: Valid-def itl-defs*)

(metis enat.distinct(1) enat-add-sub-same enat-le-plus-same(2) le-iff-add min.absorb1,
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 min.orderE
nfinite-nlength-enat)

lemma *FiniteAndEmptyEqvEmpty:*

$\vdash (finite \wedge empty) = empty$

by (*auto simp add: Valid-def itl-defs nlength-eq-enat-nfiniteD zero-enat-def*)

lemma *FiniteChopFiniteEqvFinite:*

$\vdash finite;finite = finite$

by (*auto simp add: Valid-def itl-defs (metis zero-enat-def zero-le)*)

lemma *InfChopInfEqvInf:*

$\vdash inf;inf = inf$

by (*simp add: Valid-def itl-defs*)

lemma *InfChopFiniteEqvInf:*

$\vdash inf;finite = inf$

by (*simp add: Valid-def itl-defs*)

lemma *FiniteChopInfEqvInf:*

$\vdash finite;inf = inf$

by (*auto simp add: Valid-def itl-defs (metis zero-enat-def zero-le)*)

lemma *InfEqvNotFinite:*

$\vdash inf = (\neg finite)$

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 integ-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 ChopAndInf EmptyImpFinite 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*)

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: $\vdash bi (f \wedge f1 \longrightarrow f)$ by (rule BiGen)

have 3: $\vdash bi (f \wedge f1 \longrightarrow f) \longrightarrow (f \wedge f1);g \longrightarrow f;g$ by (rule BiChopImpChop)

from 2 3 show ?thesis using MP by blast

qed

lemma AndChopB:

$\vdash (f \wedge f1);g \longrightarrow f1;g$

proof –

have 1: $\vdash f \wedge f1 \longrightarrow f1$ by auto

hence 2: $\vdash bi (f \wedge f1 \longrightarrow f1)$ by (rule BiGen)

have 3: $\vdash bi (f \wedge f1 \longrightarrow f1) \longrightarrow (f \wedge f1);g \longrightarrow f1;g$ by (rule BiChopImpChop)

from 2 3 show ?thesis using MP by blast

qed

lemma NextChop:

$\vdash (\bigcirc f);g = \bigcirc(f;g)$

proof –

have 1: $\vdash skip;(f;g) = (skip;f);g$ by (rule ChopAssoc)

show ?thesis by (metis 1 int-eq next-d-def)

qed

lemma BoxChopImpChop :

$\vdash \Box (g \longrightarrow g1) \longrightarrow f;g \longrightarrow f;g1$

proof –

have 1: $\vdash g \longrightarrow g$ by auto

hence 2: $\vdash bi (g \longrightarrow g)$ by (rule BiGen)

have 3: $\vdash bi (f \longrightarrow f) \wedge \Box(g \longrightarrow g1) \longrightarrow f;g \longrightarrow f;g1$ by (rule BiBoxChopImpChop)

from 2 3 show ?thesis by fastforce

qed

lemma LeftChopImpChop:

assumes $\vdash f \longrightarrow f1$

shows $\vdash f;g \longrightarrow f1;g$

proof –

have 1: $\vdash f \longrightarrow f1$ using assms by auto

hence 2: $\vdash bi (f \longrightarrow f1)$ by (rule BiGen)

have 3: $\vdash bi (f \longrightarrow f1) \longrightarrow f;g \longrightarrow f1;g$ by (rule BiChopImpChop)

from 2 3 show ?thesis using MP by blast

qed

lemma RightChopImpChop:

assumes $\vdash g \longrightarrow g1$

shows $\vdash f;g \longrightarrow f;g1$

proof –
have $1: \vdash g \longrightarrow g1$ **using** *assms* **by** *auto*
hence $2: \vdash \Box (g \longrightarrow g1)$ **by** (*rule BoxGen*)
have $3: \vdash \Box (g \longrightarrow g1) \longrightarrow f;g \longrightarrow f;g1$ **by** (*rule BoxChopImpChop*)
from $2\ 3$ **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *RightChopEqvChop*:
assumes $\vdash g = g1$
shows $\vdash (f;g) = (f;g1)$
using *assms* *RightChopImpChop*[*of g g1 f*] *RightChopImpChop*[*of g1 g f*]
by *fastforce*

lemma *InfRightChopEqvChop*:
assumes $\vdash inf \longrightarrow g = g1$
shows $\vdash inf \longrightarrow (f;g) = (f;g1)$
using *assms*
by (*auto simp add: Valid-def itl-defs*)

lemma *ChopOrEqv*:
 $\vdash f;(g \vee g1) = (f;g \vee f;g1)$
proof –
have $1: \vdash g \longrightarrow g \vee g1$ **by** *auto*
hence $2: \vdash f;g \longrightarrow f;(g \vee g1)$ **by** (*rule RightChopImpChop*)
have $3: \vdash g1 \longrightarrow g \vee g1$ **by** *auto*
hence $4: \vdash f;g1 \longrightarrow f;(g \vee g1)$ **by** (*rule RightChopImpChop*)
from $2\ 4$ **show** *?thesis* **by** (*meson ChopOrImp Prop02 Prop11*)
qed

lemma *OrChopEqv*:
 $\vdash (f \vee f1);g = (f;g \vee f1;g)$
proof –
have $1: \vdash f \longrightarrow f \vee f1$ **by** *auto*
hence $2: \vdash f;g \longrightarrow (f \vee f1);g$ **by** (*rule LeftChopImpChop*)
have $3: \vdash f1 \longrightarrow f \vee f1$ **by** *auto*
hence $4: \vdash f1;g \longrightarrow (f \vee f1);g$ **by** (*rule LeftChopImpChop*)
from $2\ 4$ **show** *?thesis*
by (*meson OrChopImp int-iffI Prop02*)
qed

lemma *OrChopImpRule*:
assumes $\vdash f \longrightarrow f1 \vee f2$
shows $\vdash f;g \longrightarrow (f1;g) \vee (f2;g)$
proof –
have $1: \vdash f \longrightarrow f1 \vee f2$ **using** *assms* **by** *auto*
hence $2: \vdash f;g \longrightarrow (f1 \vee f2);g$ **by** (*rule LeftChopImpChop*)
have $3: \vdash (f1 \vee f2);g = (f1;g \vee f2;g)$ **by** (*rule OrChopEqv*)
from $2\ 3$ **show** *?thesis* **by** *fastforce*
qed

lemma *LeftChopEqvChop*:
assumes $\vdash f = f1$
shows $\vdash f;g = (f1;g)$
proof –
have 1: $\vdash f = f1$ **using** *assms* **by** *auto*
hence 2: $\vdash f \longrightarrow f1$ **by** *auto*
hence 3: $\vdash f;g \longrightarrow f1;g$ **by** (*rule LeftChopImpChop*)
have $\vdash f1 \longrightarrow f$ **using** 1 **by** *auto*
hence 4: $\vdash f1;g \longrightarrow f;g$ **by** (*rule LeftChopImpChop*)
from 3 4 **show** *?thesis* **by** (*simp add: int-iffI*)
qed

lemma *OrChopEqvRule*:
assumes $\vdash f = (f1 \vee f2)$
shows $\vdash f;g = ((f1;g) \vee (f2;g))$
proof –
have 1: $\vdash f = (f1 \vee f2)$ **using** *assms* **by** *auto*
hence 2: $\vdash f;g = ((f1 \vee f2);g)$ **by** (*rule LeftChopEqvChop*)
have 3: $\vdash (f1 \vee f2);g = (f1;g \vee f2;g)$ **by** (*rule OrChopEqv*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *NextImpNext*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \bigcirc f \longrightarrow \bigcirc g$
proof –
have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence 2: $\vdash \Box (f \longrightarrow g)$ **by** (*rule BoxGen*)
have 3: $\vdash \Box (f \longrightarrow g) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **by** (*rule BoxChopImpChop*)
have 4: $\vdash (skip;f) \longrightarrow (skip;g)$ **by** (*metis* 2 3 *MP*)
from 4 **show** *?thesis* **by** (*metis next-d-def*)
qed

lemma *ChopOrImpRule*:
assumes $\vdash g \longrightarrow g1 \vee g2$
shows $\vdash f;g \longrightarrow (f;g1) \vee (f;g2)$
proof –
have 1: $\vdash g \longrightarrow g1 \vee g2$ **using** *assms* **by** *auto*
hence 2: $\vdash f;g \longrightarrow f;(g1 \vee g2)$ **by** (*rule RightChopImpChop*)
have 3: $\vdash f;(g1 \vee g2) = (f;g1 \vee f;g2)$ **by** (*rule ChopOrEqv*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *NextImpDist*:
 $\vdash \bigcirc (f \longrightarrow g) \longrightarrow \bigcirc f \longrightarrow \bigcirc g$
proof –
have 1: $\vdash (\neg (f \longrightarrow g)) = (f \wedge \neg g)$ **by** *auto*
hence 2: $\vdash skip;(\neg (f \longrightarrow g)) = skip;(f \wedge \neg g)$ **by** (*rule RightChopEqvChop*)
have 3: $\vdash f \longrightarrow g \vee (f \wedge \neg g)$ **by** *auto*
hence 4: $\vdash skip;f \longrightarrow (skip;g) \vee (skip;(f \wedge \neg g))$ **by** (*rule ChopOrImpRule*)

hence 5: $\vdash \neg (skip; (f \wedge \neg g)) \longrightarrow (skip; f) \longrightarrow (skip; g)$ **by** *auto*
have 6: $\vdash \neg (skip; (\neg(f \longrightarrow g))) \longrightarrow (skip; f) \longrightarrow (skip; g)$ **using** 2 5 **by** *fastforce*
hence 7: $\vdash \neg (\bigcirc(\neg(f \longrightarrow g))) \longrightarrow (\bigcirc f) \longrightarrow (\bigcirc g)$ **by** (*simp add: next-d-def*)
have 8: $\vdash \bigcirc(f \longrightarrow g) \longrightarrow \neg (\bigcirc(\neg(f \longrightarrow g)))$ **by** (*rule NextImpNotNextNot*)
from 7 8 **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *FiniteChopImpDiamond*:

$\vdash (f \wedge \text{finite}); g \longrightarrow \Diamond g$
proof –
have 1: $\vdash f \wedge \text{finite} \longrightarrow \text{finite}$ **by** *auto*
hence 2: $\vdash (f \wedge \text{finite}); g \longrightarrow \text{finite}; g$ **by** (*rule LeftChopImpChop*)
from 2 **show** *?thesis* **by** (*simp add: sometimes-d-def*)
qed

lemma *NowImpDiamond*:

$\vdash f \longrightarrow \Diamond f$
proof –
have 1: $\vdash \text{empty}; f = f$ **by** (*rule EmptyChop*)
have 2: $\vdash \text{empty} \longrightarrow \text{finite}$ **by** (*rule EmptyImpFinite*)
hence 3: $\vdash \text{empty}; f \longrightarrow \text{finite}; f$ **by** (*rule LeftChopImpChop*)
have 4: $\vdash f \longrightarrow \text{finite}; f$ **using** 1 3 **by** *fastforce*
from 4 **show** *?thesis* **by** (*simp add: sometimes-d-def*)
qed

lemma *BoxElim*:

$\vdash \Box f \longrightarrow f$
proof –
have 1: $\vdash \neg f \longrightarrow \Diamond (\neg f)$ **by** (*rule NowImpDiamond*)
hence 2: $\vdash \neg (\Diamond (\neg f)) \longrightarrow f$ **by** *auto*
from 2 **show** *?thesis* **by** (*metis always-d-def*)
qed

lemma *NextDiamondImpDiamond*:

$\vdash \bigcirc(\Diamond f) \longrightarrow \Diamond f$
proof –
have 1: $\vdash skip; (\text{finite}; f) = ((skip; \text{finite}); f)$ **by** (*rule ChopAssoc*)
hence 2: $\vdash (skip; \text{finite}); f = skip; (\text{finite}; f)$ **by** *auto*
hence 3: $\vdash (skip; \text{finite}); f = \bigcirc(\Diamond f)$ **by** (*simp add: next-d-def sometimes-d-def*)
have 4: $\vdash (skip; \text{finite}); f \longrightarrow \Diamond f$
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: $\vdash \Box (\Diamond (\neg f)) \longrightarrow \Diamond (\neg f)$ **by** (*rule NextDiamondImpDiamond*)
 have 5: $\vdash \neg \neg (\Diamond (\neg f)) \longrightarrow \Diamond (\neg f)$ **by** *auto*
 hence 6: $\vdash \Box (\neg \neg (\Diamond (\neg f))) \longrightarrow \Box (\Diamond (\neg f))$ **by** (*rule NextImpNext*)
 have 7: $\vdash \Box (\neg \neg (\Diamond (\neg f))) \longrightarrow \Diamond (\neg f)$ **using** 4 6 **by** *auto*
 hence 8: $\vdash \Box (\neg (\Box f)) \longrightarrow \Diamond (\neg f)$ **by** (*simp add: always-d-def*)
 hence 9: $\vdash \neg (\Diamond (\neg f)) \longrightarrow \neg (\Box (\neg (\Box f)))$ **by** *auto*
 hence 10: $\vdash \Box f \longrightarrow \text{wnext} (\Box f)$ **by** (*simp add: always-d-def wnext-d-def*)
 from 3 10 **show** *?thesis* **by** *fastforce*
qed

lemma *BoxImpBoxRule*:

assumes $\vdash f \longrightarrow g$
 shows $\vdash \Box f \longrightarrow \Box g$
proof –
 have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
 hence 2: $\vdash \neg g \longrightarrow \neg f$ **by** *auto*
 hence 3: $\vdash \Box (\neg g \longrightarrow \neg f)$ **by** (*rule BoxGen*)
 have 4: $\vdash \Box (\neg g \longrightarrow \neg f) \longrightarrow (\text{finite}; (\neg g)) \longrightarrow (\text{finite}; (\neg f))$ **by** (*rule BoxChopImpChop*)
 have 5: $\vdash (\text{finite}; (\neg g)) \longrightarrow (\text{finite}; (\neg f))$ **using** 3 4 *MP* **by** *blast*
 hence 6: $\vdash \Diamond (\neg g) \longrightarrow \Diamond (\neg f)$ **by** (*simp add: sometimes-d-def*)
 hence 7: $\vdash \neg (\Diamond (\neg f)) \longrightarrow \neg (\Diamond (\neg g))$ **by** *auto*
 from 7 **show** *?thesis* **by** (*simp add: always-d-def*)
qed

lemma *BoxImpDist*:

$\vdash \Box (f \longrightarrow g) \longrightarrow \Box f \longrightarrow \Box g$
proof –
 have 1: $\vdash (f \longrightarrow g) \longrightarrow (\neg g \longrightarrow \neg f)$
 by *auto*
 hence 2: $\vdash \Box (f \longrightarrow g) \longrightarrow \Box (\neg g \longrightarrow \neg f)$
 by (*rule BoxImpBoxRule*)
 have 3: $\vdash \Box ((\neg g) \longrightarrow \neg f) \longrightarrow (\text{finite}; (\neg g)) \longrightarrow (\text{finite}; (\neg f))$
 by (*rule BoxChopImpChop*)
 have 4: $\vdash \Box (f \longrightarrow g) \longrightarrow (\text{finite}; (\neg g)) \longrightarrow (\text{finite}; (\neg f))$
 using 2 3 *lift-imp-trans* **by** *blast*
 hence 5: $\vdash \Box (f \longrightarrow g) \longrightarrow \Diamond (\neg g) \longrightarrow \Diamond (\neg f)$
 by (*simp add: sometimes-d-def*)
 hence 6: $\vdash \Box (f \longrightarrow g) \longrightarrow \neg (\Diamond (\neg f)) \longrightarrow \neg (\Diamond (\neg g))$
 by *auto*
 from 6 **show** *?thesis* **by** (*simp add: always-d-def*)
qed

lemma *DiamondEmptyEqvFinite*:

$\vdash \Diamond \text{empty} = \text{finite}$
proof –
 have 1: $\vdash \text{finite}; \text{empty} = \text{finite}$ **by** (*rule ChopEmpty*)
 from 1 **show** *?thesis* **by** (*simp add: sometimes-d-def*)
qed

lemma *NextEqvNext*:
assumes $\vdash f = g$
shows $\vdash \bigcirc f = \bigcirc g$
proof –
have $1: \vdash f = g$ **using** *assms* **by** *auto*
hence $2: \vdash \text{skip}; f = \text{skip}; g$ **by** (*rule RightChopEqvChop*)
from 1 **show** *?thesis* **by** (*metis 2 next-d-def*)
qed

lemma *NextAndNextImpNextRule*:
assumes $\vdash (f \wedge g) \longrightarrow h$
shows $\vdash (\bigcirc f \wedge \bigcirc g) \longrightarrow \bigcirc h$
using *assms*
by (*simp add: Valid-def itl-defs*)

lemma *NextAndNextEqvNextRule*:
assumes $\vdash (f \wedge g) = h$
shows $\vdash (\bigcirc f \wedge \bigcirc g) = \bigcirc h$
using *assms*
by (*simp add: NextAndNextImpNextRule NextImpNext Prop11 Prop12*)

lemma *WeakNextEqvWeakNext*:
assumes $\vdash f = g$
shows $\vdash \text{wnext } f = \text{wnext } g$
using *assms* **using** *inteq-reflection* **by** *force*

lemma *DiamondImpDiamond*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \Diamond f \longrightarrow \Diamond g$
using *assms* **by** (*simp add: RightChopImpChop sometimes-d-def*)

lemma *DiamondEqvDiamond*:
assumes $\vdash f = g$
shows $\vdash \Diamond f = \Diamond g$
using *assms* **using** *int-eq* **by** *force*

lemma *BoxEqvBox*:
assumes $\vdash f = g$
shows $\vdash \Box f = \Box g$
using *assms* **using** *inteq-reflection* **by** *force*

lemma *BoxAndBoxImpBoxRule*:
assumes $\vdash f \wedge g \longrightarrow h$
shows $\vdash \Box f \wedge \Box g \longrightarrow \Box h$
using *assms* **by** (*auto simp: itl-defs Valid-def*)

lemma *BoxAndBoxEqvBoxRule*:
assumes $\vdash (f \wedge g) = h$
shows $\vdash (\Box f \wedge \Box g) = \Box h$
using *assms* *BoxAndBoxImpBoxRule* *BoxImpBoxRule* **by** (*metis int-iffD1 int-iffD2 int-iffI Prop12*)

lemma *ImpBoxRule*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \Box f \longrightarrow \Box g$
using *assms* **by** (*simp add: BoxImpBoxRule*)

lemma *WnextEqvEmptyOrNext*:
 $\vdash \text{wnext } f = (\text{empty} \vee \bigcirc f)$
by (*auto simp: Valid-def itl-defs*)

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)

```

```

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: $\vdash f \wedge (\Box(\neg g)) \longrightarrow \text{more} \wedge \text{wnext } f \wedge \text{wnext}(\Box(\neg g))$

by (auto simp: Valid-def itl-defs)

hence 10: $\vdash f \wedge (\Box(\neg g)) \longrightarrow \text{wnext } f \wedge \text{wnext}(\Box(\neg g))$

by auto

hence 11: $\vdash f \wedge (\Box(\neg g)) \longrightarrow \text{wnext } (f \wedge \Box(\neg g))$

by (auto simp: Valid-def itl-defs)

hence 12: $\vdash \Box(f \wedge (\Box(\neg g))) \longrightarrow \text{wnext } (f \wedge \Box(\neg g))$

by (rule BoxGen)

have 13: $\vdash \Box(f \wedge (\Box(\neg g))) \longrightarrow \text{wnext } (f \wedge \Box(\neg g)) \wedge f \wedge (\Box(\neg g)) \longrightarrow \Box(f \wedge (\Box(\neg g)))$

by (rule BoxInduct)

hence 14: $\vdash \Box(f \wedge (\Box(\neg g))) \longrightarrow \text{wnext } (f \wedge \Box(\neg g)) \longrightarrow ((f \wedge (\Box(\neg g))) \longrightarrow \Box(f \wedge (\Box(\neg g))))$

by fastforce

have 15: $\vdash ((f \wedge (\Box (\neg g))) \longrightarrow \Box(f \wedge (\Box (\neg g))))$
using 12 14 *MP* **by** *blast*
have 16: $\vdash \Box(f \wedge (\Box (\neg g))) \longrightarrow (f \wedge (\Box (\neg g)))$
by (*rule BoxElim*)
have 17: $\vdash \Box(f \wedge (\Box (\neg g))) = (f \wedge (\Box (\neg g)))$
using 16 15 **by** *fastforce*
have 18: $\vdash (f \wedge (\Box (\neg g))) \longrightarrow \text{more}$
using 9 **by** *auto*
hence 19: $\vdash \Box(f \wedge (\Box (\neg g))) \longrightarrow \Box \text{ more}$
by (*rule ImpBoxRule*)
have 20: $\vdash \text{finite} = (\neg(\Box \text{ more}))$
by (*metis DiamondEmptyEqvFinite InfEqvNotFinite always-d-def empty-d-def finite-d-def int-eq*)
have 21: $\vdash \text{finite} \longrightarrow \neg(f \wedge (\Box (\neg g)))$
using 17 19 20 **by** *fastforce*
hence 22: $\vdash \text{finite} \longrightarrow \neg f \vee \neg(\Box (\neg g))$
by *auto*
have 23: $\vdash (\neg(\Box (\neg g))) = \Diamond g$
by (*auto simp: always-d-def*)
from 22 23 **show** *?thesis* **by** *fastforce*
qed

lemma *DiamondIntroB*:

assumes $\vdash (f \wedge \neg g) \longrightarrow \bigcirc (f \wedge \neg g)$
shows $\vdash f \wedge \text{finite} \longrightarrow \Diamond g$
proof –
have 1: $\vdash (f \wedge \neg g) \longrightarrow \bigcirc (f \wedge \neg g)$ **using** *assms* **by** *auto*
hence 2: $\vdash \text{finite} \longrightarrow \neg(f \wedge \neg g)$ **by** (*rule NextLoop*)
hence 3: $\vdash f \wedge \text{finite} \longrightarrow g$ **by** *auto*
have 4: $\vdash g \longrightarrow \Diamond g$ **by** (*rule NowImpDiamond*)
from 3 4 **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *NextContra* :

assumes $\vdash (f \wedge \neg g) \longrightarrow (\bigcirc f \wedge \neg(\bigcirc g))$
shows $\vdash f \wedge \text{finite} \longrightarrow g$
proof –
have 1: $\vdash (f \wedge \neg g) \longrightarrow (\bigcirc f \wedge \neg(\bigcirc g))$ **using** *assms* **by** *auto*
hence 2: $\vdash \neg(f \longrightarrow g) \longrightarrow \bigcirc(\neg(f \longrightarrow g))$ **by** (*auto simp: itl-defs Valid-def*)
hence 3: $\vdash \text{finite} \longrightarrow \neg\neg(f \longrightarrow g)$ **by** (*rule NextLoop*)
from 3 **show** *?thesis* **by** *auto*
qed

lemma *DiamondDiamondEqvDiamond*:

$\vdash \Diamond(\Diamond f) = \Diamond f$
proof –
have 1: $\vdash \text{finite}; \text{finite} = \text{finite}$ **by** (*simp add: FiniteChopFiniteEqvFinite*)
hence 2: $\vdash (\text{finite}; \text{finite}); f = \text{finite}; f$ **using** *LeftChopEqvChop* **by** *blast*
have 3: $\vdash (\text{finite}; \text{finite}); f = \text{finite}; (\text{finite}; f)$ **using** *ChopAssoc* **by** *fastforce*
from 2 3 **show** *?thesis* **by** (*metis integ-reflection sometimes-d-def*)
qed

lemma *WeakNextDiamondInduct*:

assumes $\vdash \text{wnext } (\Diamond f) \longrightarrow f$

shows $\vdash \text{finite} \longrightarrow f$

proof –

have 1: $\vdash \text{wnext } (\Diamond f) \longrightarrow f$ **using** *assms* **by** *blast*

hence 2: $\vdash \neg f \longrightarrow \neg(\text{wnext } (\Diamond f))$ **by** *fastforce*

hence 3: $\vdash \neg f \longrightarrow \bigcirc(\neg(\Diamond f))$ **by** (*simp add: wnext-d-def*)

have 4: $\vdash f \longrightarrow \Diamond f$ **by** (*rule NowImpDiamond*)

hence 5: $\vdash \neg(\Diamond f) \longrightarrow \neg f$ **by** *auto*

have 6: $\vdash \neg f \longrightarrow \bigcirc(\neg f)$ **using** 3 5 **using** *NextImpNext lift-imp-trans* **by** *blast*

hence 7: $\vdash \text{finite} \longrightarrow \neg\neg f$ **by** (*rule NextLoop*)

from 7 **show** *?thesis* **by** *auto*

qed

lemma *EmptyNextInducta*:

assumes $\vdash \text{empty} \longrightarrow f$

$\vdash \bigcirc f \longrightarrow f$

shows $\vdash \text{finite} \longrightarrow f$

proof –

have 1: $\vdash \text{empty} \longrightarrow f$ **using** *assms* **by** *auto*

have 2: $\vdash \bigcirc f \longrightarrow f$ **using** *assms* **by** *blast*

have 3: $\vdash (\text{empty} \vee \bigcirc f) \longrightarrow f$ **using** 1 2 **by** *fastforce*

have 4: $\vdash \text{wnext } f = (\text{empty} \vee \bigcirc f)$ **by** (*rule WnextEqvEmptyOrNext*)

hence 5: $\vdash \text{wnext } f \longrightarrow f$ **using** 3 **by** *fastforce*

hence 6: $\vdash \neg f \longrightarrow \neg(\text{wnext } f)$ **by** *auto*

hence 7: $\vdash \neg f \longrightarrow \bigcirc(\neg f)$ **by** (*auto simp: wnext-d-def*)

hence 8: $\vdash \text{finite} \longrightarrow \neg\neg f$ **by** (*rule NextLoop*)

from 8 **show** *?thesis* **by** *auto*

qed

lemma *EmptyNextInductb*:

assumes $\vdash \text{empty} \wedge f \longrightarrow g$

$\vdash \bigcirc(f \longrightarrow g) \wedge f \longrightarrow g$

shows $\vdash f \wedge \text{finite} \longrightarrow g$

proof –

have 1: $\vdash \text{empty} \wedge f \longrightarrow g$ **using** *assms* **by** *auto*

have 2: $\vdash \bigcirc(f \longrightarrow g) \wedge f \longrightarrow g$ **using** *assms* **by** *blast*

have 3: $\vdash (\text{empty} \vee \bigcirc(f \longrightarrow g)) \wedge f \longrightarrow g$ **using** 1 2 **by** *fastforce*

hence 4: $\vdash \text{wnext } (f \longrightarrow g) \wedge f \longrightarrow g$ **using** *WnextEqvEmptyOrNext* **by** *fastforce*

hence 5: $\vdash \text{wnext } (f \longrightarrow g) \longrightarrow (f \longrightarrow g)$ **by** *fastforce*

hence 6: $\vdash \neg(f \longrightarrow g) \longrightarrow \neg(\text{wnext } (f \longrightarrow g))$ **by** *fastforce*

hence 7: $\vdash \neg(f \longrightarrow g) \longrightarrow \bigcirc(\neg(f \longrightarrow g))$ **by** (*simp add: wnext-d-def*)

hence 8: $\vdash \text{finite} \longrightarrow \neg\neg(f \longrightarrow g)$ **by** (*rule NextLoop*)

from 8 **show** *?thesis* **by** *auto*

qed

lemma *FinImpFin*:

assumes $\vdash f \longrightarrow g$

shows $\vdash \text{fin } f \longrightarrow \text{fin } g$
using *ImpBoxRule*[of *LIFT* (*empty* \longrightarrow *f*) *LIFT* (*empty* \longrightarrow *g*)] *assms*
 fin-d-def [of *f*] fin-d-def [of *g*] **by** *fastforce*

lemma *FinEqvFin*:
assumes $\vdash f = g$
shows $\vdash \text{fin } f = \text{fin } g$
using *assms* **by** (*simp add: FinImpFin Prop11*)

lemma *FinAndFinImpFinRule*:
assumes $\vdash f \wedge g \longrightarrow h$
shows $\vdash \text{fin } f \wedge \text{fin } g \longrightarrow \text{fin } h$
using *assms* **by** (*auto simp add: itl-defs Valid-def*)

lemma *FinAndFinEqvFinRule*:
assumes $\vdash (f \wedge g) = h$
shows $\vdash (\text{fin } f \wedge \text{fin } g) = \text{fin } h$
using *assms*
by (*simp add: FinAndFinImpFinRule FinImpFin Prop11 Prop12*)

lemma *HaltEqvHalt*:
assumes $\vdash f = g$
shows $\vdash \text{halt } f = \text{halt } g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash (\text{empty} = f) = (\text{empty} = g)$ **by** *auto*
hence 3: $\vdash \Box(\text{empty} = f) = \Box(\text{empty} = g)$ **by** (*rule BoxEqvBox*)
from 3 **show** ?thesis **by** (*simp add: halt-d-def*)
qed

lemma *BiImpDiImpDi*:
 $\vdash \text{bi } (f \longrightarrow g) \longrightarrow \text{di } f \longrightarrow \text{di } g$
proof –
have 1: $\vdash \text{bi } (f \longrightarrow g) \longrightarrow (f; \# \text{True}) \longrightarrow (g; \# \text{True})$ **by** (*rule BiChopImpChop*)
from 1 **show** ?thesis **by** (*simp add: di-d-def*)
qed

lemma *DiImpDi*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \text{di } f \longrightarrow \text{di } g$
proof –
have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence 2: $\vdash f; \# \text{True} \longrightarrow g; \# \text{True}$ **by** (*rule LeftChopImpChop*)
from 2 **show** ?thesis **by** (*simp add: di-d-def*)
qed

lemma *BiImpBiRule*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \text{bi } f \longrightarrow \text{bi } g$
proof –

have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence 2: $\vdash \neg g \longrightarrow \neg f$ **by** *auto*
hence 3: $\vdash di(\neg g) \longrightarrow di(\neg f)$ **by** (*rule DiImpDi*)
hence 4: $\vdash \neg(di(\neg f)) \longrightarrow \neg(di(\neg g))$ **by** *auto*
from 4 **show** *?thesis* **by** (*simp add: bi-d-def*)
qed

lemma *DiEqvDi*:
assumes $\vdash f = g$
shows $\vdash di f = di g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash f; \#True = g; \#True$ **by** (*rule LeftChopEqvChop*)
from 2 **show** *?thesis* **by** (*simp add: di-d-def*)
qed

lemma *BiEqvBi*:
assumes $\vdash f = g$
shows $\vdash bi f = bi g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash (\neg f) = (\neg g)$ **by** *auto*
hence 3: $\vdash di(\neg f) = di(\neg g)$ **by** (*rule DiEqvDi*)
hence 4: $\vdash \neg(di(\neg f)) = \neg(di(\neg g))$ **by** *auto*
from 4 **show** *?thesis* **by** (*simp add: bi-d-def*)
qed

lemma *LeftChopChopImpChopRule*:
assumes $\vdash (f; g) \longrightarrow g$
shows $\vdash (f; g); h \longrightarrow (g; h)$
proof –
have 1: $\vdash (f; g) \longrightarrow g$ **using** *assms* **by** *blast*
hence 2: $\vdash (f; g); h \longrightarrow g; h$ **by** (*rule LeftChopImpChop*)
have 3: $\vdash f; (g; h) = (f; g); h$ **by** (*rule ChopAssoc*)
from 2 3 **show** *?thesis* **by** *auto*
qed

lemma *AndChopCommute* :
 $\vdash (f \wedge f1); g = (f1 \wedge f); g$
proof –
have 1: $\vdash (f \wedge f1) = (f1 \wedge f)$ **by** *auto*
from 1 **show** *?thesis* **by** (*rule LeftChopEqvChop*)
qed

lemma *BiAndChopImport*:
 $\vdash bi f \wedge (f1; g) \longrightarrow (f \wedge f1); g$
proof –
have 1: $\vdash f \longrightarrow (f1 \longrightarrow f \wedge f1)$ **by** *auto*
hence 2: $\vdash bi f \longrightarrow bi (f1 \longrightarrow f \wedge f1)$ **by** (*rule BiImpBiRule*)
have 3: $\vdash bi (f1 \longrightarrow (f \wedge f1)) \longrightarrow f1; g \longrightarrow (f \wedge f1); g$ **by** (*rule BiChopImpChop*)

from 2 3 show ?thesis using MP by fastforce
qed

lemma *StateAndChopImport*:

$\vdash (init\ w) \wedge (f;g) \longrightarrow ((init\ w) \wedge f);g$

proof –

have 1: $\vdash (init\ w) \longrightarrow bi\ (init\ w)$ **by** (rule *StateImpBi*)

hence 2: $\vdash (init\ w) \wedge (f;g) \longrightarrow bi\ (init\ w) \wedge (f;g)$ **by** auto

have 3: $\vdash bi\ (init\ w) \wedge (f;g) \longrightarrow ((init\ w) \wedge f);g$ **by** (rule *BiAndChopImport*)

from 2 3 show ?thesis using MP by fastforce

qed

6.5 Further Properties Di and Bi

lemma *ImpDi*:

$\vdash f \longrightarrow di\ f$

proof –

have 1: $\vdash f; empty = f$ **by** (rule *ChopEmpty*)

have 2: $\vdash empty \longrightarrow \#True$ **by** auto

hence 3: $\vdash f; empty \longrightarrow f; \#True$ **by** (rule *RightChopImpChop*)

have 4: $\vdash f \longrightarrow f; \#True$ **using** 1 3 **by** fastforce

from 4 show ?thesis **by** (simp add: di-d-def)

qed

lemma *DiState*:

$\vdash di\ (init\ w) = (init\ w)$

proof –

have 0: $\vdash (init\ (\neg w)) \longrightarrow bi\ (init\ (\neg w))$ **using** *StateImpBi* **by** fastforce

hence 1: $\vdash \neg (init\ w) \longrightarrow bi\ (\neg (init\ w))$ **using** *Initprop(2)* **by** (metis *inteq-reflection*)

hence 2: $\vdash (\neg (init\ w)) \longrightarrow \neg (di\ (\neg \neg (init\ w)))$ **by** (simp add: bi-d-def)

have 3: $\vdash (\neg (init\ w) \longrightarrow \neg (di\ (\neg \neg (init\ w)))) \longrightarrow$
 $(di\ (\neg \neg (init\ w)) \longrightarrow (init\ w))$ **by** auto

have 4: $\vdash di\ (\neg \neg (init\ w)) \longrightarrow (init\ w)$ **using** 2 3 MP **by** blast

have 5: $\vdash (init\ w) \longrightarrow \neg \neg (init\ w)$ **by** auto

hence 6: $\vdash di\ (init\ w) \longrightarrow di\ (\neg \neg (init\ w))$ **by** (rule *DiImpDi*)

have 7: $\vdash di\ (init\ w) \longrightarrow (init\ w)$ **using** 6 4 **using** *lift-imp-trans* **by** metis

have 8: $\vdash (init\ w) \longrightarrow di\ (init\ w)$ **by** (rule *ImpDi*)

from 7 8 show ?thesis **by** fastforce

qed

lemma *StateChop*:

$\vdash (init\ w); f \longrightarrow (init\ w)$

by (metis *ChopAssoc Prop12 RightChopImpChop TrueW init-d-def int-simps(13) int-simps(17) inteq-reflection*)

lemma *StateChopExportA*:

$\vdash ((init\ w) \wedge f); g \longrightarrow (init\ w)$

using *DiState AndChopA StateChop* **by** fastforce

lemma *StateAndChop*:

$\vdash ((init\ w) \wedge f); g = ((init\ w) \wedge (f; g))$
by (*simp add: AndChopB StateAndChopImport StateChopExportA Prop11 Prop12*)

lemma *StateAndChopImpChopRule*:

assumes $\vdash (init\ w) \wedge f \longrightarrow f1$
shows $\vdash (init\ w) \wedge (f; g) \longrightarrow (f1; g)$
proof –
have 1: $\vdash (init\ w) \wedge f \longrightarrow f1$ **using** *assms* **by** *auto*
hence 2: $\vdash ((init\ w) \wedge f); g \longrightarrow f1; g$ **by** (*rule LeftChopImpChop*)
have 3: $\vdash ((init\ w) \wedge f); g = ((init\ w) \wedge (f; g))$ **by** (*rule StateAndChop*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *StateImpChopEqvChop* :

assumes $\vdash (init\ w) \longrightarrow (f = f1)$
shows $\vdash (init\ w) \longrightarrow ((f; g) = (f1; g))$
proof –
have 1: $\vdash (init\ w) \longrightarrow (f = f1)$ **using** *assms* **by** *auto*
hence 2: $\vdash (init\ w) \wedge f \longrightarrow f1$ **by** *auto*
hence 3: $\vdash (init\ w) \wedge (f; g) \longrightarrow (f1; g)$ **by** (*rule StateAndChopImpChopRule*)
have 4: $\vdash (init\ w) \wedge f1 \longrightarrow f$ **using** 1 **by** *auto*
hence 5: $\vdash (init\ w) \wedge (f1; g) \longrightarrow (f; g)$ **by** (*rule StateAndChopImpChopRule*)
from 3 5 **show** *?thesis* **by** *fastforce*
qed

lemma *ChopEqvStateAndChop*:

assumes $\vdash f = (init\ w) \wedge f1$
shows $\vdash (f; g) = ((init\ w) \wedge (f1; g))$
proof –
have 1: $\vdash f = ((init\ w) \wedge f1)$ **using** *assms* **by** *auto*
hence 2: $\vdash f; g = (((init\ w) \wedge f1); g)$ **by** (*rule LeftChopEqvChop*)
have 3: $\vdash ((init\ w) \wedge f1); g = ((init\ w) \wedge (f1; g))$ **by** (*rule StateAndChop*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *DiIntro*:

$\vdash f \longrightarrow di\ f$
proof –
have 1: $\vdash f; empty = f$ **by** (*rule ChopEmpty*)
have 2: $\vdash empty \longrightarrow \#True$ **by** *auto*
hence 3: $\vdash \Box(empty \longrightarrow \#True)$ **by** (*rule BoxGen*)
have 4: $\vdash \Box(empty \longrightarrow \#True) \longrightarrow (f; empty \longrightarrow f; \#True)$ **by** (*rule BoxChopImpChop*)
have 5: $\vdash f; empty \longrightarrow f; \#True$ **using** 3 4 *MP* **by** *fastforce*
hence 6: $\vdash f; empty \longrightarrow di\ f$ **by** (*simp add: di-d-def*)
from 1 6 **show** *?thesis* **by** *fastforce*
qed

lemma *BiElim*:

$\vdash bi\ f \longrightarrow f$
proof –

have 1: $\vdash \neg f \longrightarrow di(\neg f)$ **by** (rule DiIntro)
have 2: $\vdash (\neg f \longrightarrow di(\neg f)) \longrightarrow (\neg(di(\neg f)) \longrightarrow f)$ **by** auto
have 3: $\vdash \neg (di(\neg f)) \longrightarrow f$ **using** 1 2 MP **by** blast
from 3 **show** ?thesis **by** (metis bi-d-def)
qed

lemma BiContraPosImpDist:

$\vdash bi(\neg g \longrightarrow \neg f) \longrightarrow (bi f) \longrightarrow (bi g)$

proof –

have 1: $\vdash bi(\neg g \longrightarrow \neg f) \longrightarrow (di(\neg g)) \longrightarrow (di(\neg f))$ **by** (rule BiImpDiImpDi)
hence 2: $\vdash bi(\neg g \longrightarrow \neg f) \longrightarrow (\neg(di(\neg g))) \longrightarrow (\neg(di(\neg f)))$ **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 $\vdash f = if_i(\text{init } w) \text{ then } f1 \text{ else } f2$
shows $\vdash f; g = if_i(\text{init } w) \text{ then } (f1; g) \text{ else } (f2; g)$

proof –

have 1: $\vdash f = if_i(\text{init } w) \text{ then } f1 \text{ else } f2$
using assms **by** auto
hence 2: $\vdash f = (((\text{init } w) \wedge f1) \vee ((\text{init } (\neg w)) \wedge f2))$
by (metis Initprop(2) ifthenelse-d-def inteq-reflection)
hence 3: $\vdash f; g = (((\text{init } w) \wedge f1); g \vee ((\text{init } (\neg w)) \wedge f2); g)$
by (rule OrChopEqvRule)
have 4: $\vdash ((\text{init } w) \wedge f1); g = ((\text{init } w) \wedge (f1; g))$
by (rule StateAndChop)
have 5: $\vdash ((\text{init } (\neg w)) \wedge f2); g = ((\text{init } (\neg w)) \wedge (f2; g))$
by (rule StateAndChop)
have 6: $\vdash f; g = (((\text{init } w) \wedge f1; g) \vee ((\text{init } (\neg w)) \wedge f2; g))$
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*:
 $\vdash (empty \vee f); g = (g \vee (f; g))$
proof –
have 1: $\vdash (empty \vee f); g = ((empty ; g) \vee (f; g))$ **by** (*rule OrChopEqv*)
have 2: $\vdash empty ; g = g$ **by** (*rule EmptyChop*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextChopEqv*:
 $\vdash (empty \vee \circ f); g = (g \vee \circ(f; g))$
proof –
have 1: $\vdash (empty \vee \circ f); g = (g \vee ((\circ f); g))$ **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 $\vdash f \longrightarrow empty \vee f1$
shows $\vdash f; g \longrightarrow g \vee (f1; g)$
proof –
have 1: $\vdash f \longrightarrow empty \vee f1$ **using** *assms* **by** *auto*
hence 2: $\vdash f; g \longrightarrow (empty \vee f1); g$ **by** (*rule LeftChopImpChop*)
have 3: $\vdash (empty \vee f1); g = (g \vee (f1; g))$ **by** (*rule EmptyOrChopEqv*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrChopEqvRule*:
assumes $\vdash f = (empty \vee f1)$
shows $\vdash f; g = (g \vee (f1; g))$
proof –
have 1: $\vdash f = (empty \vee f1)$ **using** *assms* **by** *auto*
hence 2: $\vdash f; g = ((empty \vee f1); g)$ **by** (*rule LeftChopEqvChop*)
have 3: $\vdash (empty \vee f1); g = (g \vee (f1; g))$ **by** (*rule EmptyOrChopEqv*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextChopImpRule*:
assumes $\vdash f \longrightarrow empty \vee \circ f1$
shows $\vdash f; g \longrightarrow g \vee \circ(f1; g)$
proof –
have 1: $\vdash f \longrightarrow empty \vee \circ f1$ **using** *assms* **by** *auto*
hence 2: $\vdash f; g \longrightarrow (empty \vee \circ f1); g$ **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)
(metis dual-order.order-iff-strict is-NNil-ndropn less-eqE ndropn-ndropn ndropn-nfirst
ndropn-nlength nlength-eq-enat-nfiniteD not-le ntaken-nnth)

lemma *BoxStateChopBoxEqvBox*:
 $\vdash \Box (\text{init } w); \Box (\text{init } w) = \Box (\text{init } w)$
proof –

have 1: $\vdash (\Box (init\ w)) = ((init\ w) \wedge (\text{empty} \vee \bigcirc(\Box (init\ w))))$
by (rule *BoxEqvAndEmptyOrNextBox*)
hence 2: $\vdash (\Box (init\ w); \Box (init\ w)) =$
 $((init\ w) \wedge ((\text{empty} \vee \bigcirc(\Box (init\ w))); \Box (init\ w)))$
by (metis *StateAndChop integ-reflection*)
have 3: $\vdash ((\text{empty} \vee \bigcirc(\Box (init\ w))); \Box (init\ w)) =$
 $(\Box (init\ w) \vee \bigcirc(\Box (init\ w); \Box (init\ w)))$
by (rule *EmptyOrNextChopEqv*)
have 4: $\vdash (\Box (init\ w); \Box (init\ w)) =$
 $((init\ w) \wedge (\Box (init\ w) \vee \bigcirc(\Box (init\ w); \Box (init\ w))))$
using 2 3 **by** *fastforce*
have 5: $\vdash \neg (\Box (init\ w)) \longrightarrow \neg (init\ w) \vee \neg (\bigcirc(\Box (init\ w)))$
by (rule *NotBoxImpNotOrNotNextBox*)
have 6: $\vdash (\Box (init\ w); \Box (init\ w)) \wedge \neg (\Box (init\ w)) \longrightarrow$
 $\bigcirc(\Box (init\ w); \Box (init\ w)) \wedge \neg (\bigcirc(\Box (init\ w)))$
using 4 5 **by** *fastforce*
hence 7: $\vdash \Box (init\ w); \Box (init\ w) \wedge \text{finite} \longrightarrow \Box (init\ w)$
by (rule *NextContra*)
have 8: $\vdash \Box (init\ w); \Box (init\ w) \wedge \text{inf} \longrightarrow \Box (init\ w)$
by (rule *BoxStateChopBoxAndInfImpBox*)
have 9: $\vdash \Box (init\ w); \Box (init\ w) \wedge (\text{finite} \vee \text{inf}) \longrightarrow \Box (init\ w)$
using 7 8 **by** *fastforce*
hence 10: $\vdash \Box (init\ w); \Box (init\ w) \longrightarrow \Box (init\ w)$
using *FiniteOrInfinite* **by** *fastforce*
have 11: $\vdash \Box (init\ w) = ((init\ w) \wedge \Box (init\ w))$
by (rule *BoxEqvAndBox*)
have 12: $\vdash \text{empty}; \Box (init\ w) = \Box (init\ w)$
by (rule *EmptyChop*)
have 13: $\vdash ((init\ w) \wedge \text{empty}); \Box (init\ w) = ((init\ w) \wedge (\text{empty}; \Box (init\ w)))$
by (rule *StateAndChop*)
have 14: $\vdash \Box (init\ w) = ((init\ w) \wedge \text{empty}); \Box (init\ w)$
using 11 12 13 **by** *fastforce*
have 15: $\vdash (init\ w) \wedge \text{empty} \longrightarrow \Box (init\ w)$
by (rule *StateAndEmptyImpBoxState*)
hence 16: $\vdash ((init\ w) \wedge \text{empty}); \Box (init\ w) \longrightarrow \Box (init\ w); \Box (init\ w)$
by (rule *LeftChopImpChop*)
have 17: $\vdash \Box (init\ w) \longrightarrow \Box (init\ w); \Box (init\ w)$
using 14 16 **by** *fastforce*
from 10 17 **show** ?thesis **by** *fastforce*
qed

lemma *NotBoxStateImpBoxYieldsNotBox*:

$\vdash \neg (\Box (init\ w)) \longrightarrow (\Box (init\ w)) \text{ yields } (\neg (\Box (init\ w)))$

proof –

have 1: $\vdash \Box (init\ w); \Box (init\ w) = \Box (init\ w)$ **by** (rule *BoxStateChopBoxEqvBox*)
have 2: $\vdash \Box (init\ w) = (\neg \neg (\Box (init\ w)))$ **by** *auto*
hence 3: $\vdash \Box (init\ w); \Box (init\ w) = \Box (init\ w); (\neg \neg (\Box (init\ w)))$ **by** (rule *RightChopEqvChop*)
have 4: $\vdash \neg (\Box (init\ w)) \longrightarrow \neg (\Box (init\ w); (\neg \neg (\Box (init\ w))))$ **using** 1 3 **by** *auto*
from 4 **show** ?thesis **by** (simp add: *yields-d-def*)
qed

lemma *StateEqvBi*:
 $\vdash (init\ w) = bi\ (init\ w)$
proof –
 have 1: $\vdash (init\ w) \longrightarrow bi\ (init\ w)$ **by** (rule *StateImpBi*)
 have 2: $\vdash bi\ (init\ w) \longrightarrow (init\ w)$ **by** (rule *BiElim*)
 from 1 2 **show** ?thesis **by** fastforce
qed

lemma *FiniteChopEqvDiamond*:
 $\vdash finite; f = \Diamond f$
by (simp add: sometimes-d-def)

6.6 Properties of Da and Ba

lemma *DaEqvDtDi*:
 $\vdash da\ f = \Diamond (di\ f)$
proof –
 have 1: $\vdash finite; (f; \#True) = finite; (f; \#True)$ **by** auto
 hence 2: $\vdash finite; (f; \#True) = finite; di\ f$ **by** (simp add: di-d-def)
 have 3: $\vdash finite; di\ f = \Diamond (di\ f)$ **by** (rule *FiniteChopEqvDiamond*)
 have 4: $\vdash finite; (f; \#True) = \Diamond (di\ f)$ **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: $\vdash finite; f = \Diamond f$ **by** (rule *FiniteChopEqvDiamond*)
 hence 2: $\vdash (finite; f); \#True = (\Diamond f); \#True$ **by** (rule *LeftChopEqvChop*)
 hence 3: $\vdash (finite; f); \#True = di\ (\Diamond f)$ **by** (simp add: di-d-def)
 have 4: $\vdash finite; (f; \#True) = (finite; f); \#True$ **by** (rule *ChopAssoc*)
 have 5: $\vdash finite; (f; \#True) = di\ (\Diamond f)$ **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*:

$\vdash di (\neg f) = (\neg (bi f))$

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*:
 $\vdash ba\ f \longrightarrow \Box f$
proof –
have 1: $\vdash ba\ f = bi(\Box f)$ **by** (rule *BaEqvBiBt*)
have 2: $\vdash bi(\Box f) \longrightarrow \Box f$ **by** (rule *BiElim*)
from 1 2 **show** ?thesis **using** lift-imp-trans **by** fastforce
qed

lemma *DiamondImpDa*:
 $\vdash \Diamond f \longrightarrow da\ f$
by (metis *DiIntro DiamondImpDiamond da-d-def di-d-def sometimes-d-def*)

lemma *DiImpDa*:
 $\vdash di\ f \longrightarrow da\ f$
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*:
 $\vdash ba\ f \wedge (g; g1) \longrightarrow (f \wedge g); (f \wedge g1)$
proof –
have 1: $\vdash ba\ f \longrightarrow bi\ f$ **by** (rule *BaImpBi*)
have 2: $\vdash bi\ f \wedge (g; g1) \longrightarrow (f \wedge g); g1$ **by** (rule *BiAndChopImport*)
have 3: $\vdash ba\ f \longrightarrow \Box f$ **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*)
qed

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*:

$\vdash ((init \ w) \wedge \text{empty}) ; f = ((init \ w) \wedge f)$

proof –

have 1: $\vdash ((init \ w) \wedge \text{empty}) ; f = ((init \ w) \wedge \text{empty} ; f)$ by (rule StateAndChop)

have 2: $\vdash \text{empty} ; f = f$ 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: $\vdash \circ((init\ w) \wedge f; g) = (\circ (init\ w) \wedge \circ(f; g))$ **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*:

$\vdash \text{fin } w \longrightarrow \Box(\text{fin } w)$

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: $\vdash \text{fin } w \longrightarrow \Box(\text{fin } w)$ **by** (*rule FinImpBox*)

hence 2: $\vdash \text{fin } w \wedge f;g \longrightarrow \Box(\text{fin } w) \wedge (f;g)$ **by** *auto*

have 3: $\vdash \Box(\text{fin } w) \wedge (f;g) \longrightarrow f;((\text{fin } w) \wedge g)$ **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 *inteq-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: $\vdash (((init\ w) \wedge empty) \vee \bigcirc (fin\ (init\ w))); f$
 $= (((init\ w) \wedge empty); f \vee (\bigcirc (fin\ (init\ w)))); f$
 by (rule *OrChopEqv*)
 have 4: $\vdash ((init\ w) \wedge empty); f = ((init\ w) \wedge f)$
 by (rule *StateAndEmptyChop*)
 have 5: $\vdash (\bigcirc (fin\ (init\ w))); f = \bigcirc((fin\ (init\ w)); f)$
 by (rule *NextChop*)
 from 2 3 4 5 show ?thesis by fastforce
 qed

lemma *FinChopEqvDiamond*:

$\vdash (fin\ (init\ w) \wedge finite); f = \Diamond((init\ w) \wedge f)$
proof –
 have 1: $\vdash (fin\ (init\ w) \wedge finite) = (finite; ((init\ w) \wedge empty))$
 by (metis *AndFinEqvChopAndEmpty int-simps(17) inteq-reflection lift-and-com*)
 hence 2: $\vdash (fin\ (init\ w) \wedge finite); f = (finite; ((init\ w) \wedge empty)); f$
 by (rule *LeftChopEqvChop*)
 have 3: $\vdash finite; ((init\ w) \wedge empty); f = (finite; ((init\ w) \wedge empty)); f$
 by (rule *ChopAssoc*)
 have 4: $\vdash finite; ((init\ w) \wedge empty); f = \Diamond((init\ w) \wedge empty); f$
 by (simp add: *sometimes-d-def*)
 have 5: $\vdash ((init\ w) \wedge empty); f = ((init\ w) \wedge f)$
 using *StateAndEmptyChop* by blast
 hence 6: $\vdash \Diamond((init\ w) \wedge empty); f = \Diamond((init\ w) \wedge f)$
 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*:

$\vdash (fin\ (init\ w) \wedge finite) \text{ yields } (init\ w)$
proof –
 have 1: $\vdash (fin\ (init\ w) \wedge finite); (\neg(init\ w)) = \Diamond((init\ w) \wedge \neg(init\ w))$
 by (rule *FinChopEqvDiamond*)
 have 2: $\vdash \neg(\Diamond((init\ w) \wedge \neg(init\ w)))$
 by (rule *NotDiamondAndNot*)
 have 3: $\vdash \neg((fin\ (init\ w) \wedge finite); (\neg(init\ w)))$
 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: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); \#True = (f \wedge \text{finite}); ((\text{init } w) \wedge \#True)$

by (*rule AndFinChopEqvStateAndChop*)

have 2: $\vdash ((\text{init } w) \wedge \#True) = (\text{init } w)$

by *auto*

hence 3: $\vdash ((f \wedge \text{finite}); ((\text{init } w) \wedge \#True)) = ((f \wedge \text{finite}); (\text{init } w))$

by (*rule RightChopEqvChop*)

have 4: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); \#True = (f \wedge \text{finite}); (\text{init } w)$

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: $\vdash \text{di } (\neg(f \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w))) = (\neg(\text{bi } (f \wedge \text{finite} \longrightarrow \text{fin}(\text{init } w))))$
 by (rule DiNotEqvNotBi)
 have 7: $\vdash \neg(\text{bi } (f \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w))) = (f \wedge \text{finite});(\text{init } (\neg w))$
 using 1 5 6 Initprop by fastforce
 hence 8: $\vdash \text{bi } (f \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w)) = (\neg((f \wedge \text{finite});(\neg(\text{init } w))))$
 by (metis Initprop(2) int-eq int-simps(7))
 from 8 show ?thesis by (simp add: yields-d-def)
 qed

lemma StateImpYields:

assumes $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w1)$
 shows $\vdash (\text{init } w) \longrightarrow ((f \wedge \text{finite}) \text{ yields } (\text{init } w1))$

proof –

have 1: $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w1)$
 using assms by auto
 hence 2: $\vdash (\text{init } w) \longrightarrow (f \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w1))$
 by auto
 hence 3: $\vdash (\text{init } w) \longrightarrow \text{bi } (f \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w1))$
 by (rule StateImpBiGen)
 have 4: $\vdash \text{bi } (f \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w1)) = (f \wedge \text{finite}) \text{ yields } (\text{init } w1)$
 by (rule BiImpFinEqvYieldsState)
 from 3 4 show ?thesis by fastforce

qed

lemma StateAndYieldsImpYields:

assumes $\vdash (\text{init } w) \wedge f \longrightarrow f1$
 shows $\vdash (\text{init } w) \wedge (f1 \text{ yields } g) \longrightarrow (f \text{ yields } g)$

proof –

have 1: $\vdash (\text{init } w) \wedge f \longrightarrow f1$ using assms by auto
 hence 2: $\vdash (\text{init } w) \wedge (f; (\neg g)) \longrightarrow f1; (\neg g)$ by (rule StateAndChopImpChopRule)
 hence 3: $\vdash (\text{init } w) \wedge \neg(f1; (\neg g)) \longrightarrow \neg(f; (\neg g))$ by auto
 from 3 show ?thesis by (simp add: yields-d-def)

qed

lemma AndYieldsA:

$\vdash f \text{ yields } g \longrightarrow (f \wedge f1) \text{ yields } g$

proof –

have 1: $\vdash f \wedge f1 \longrightarrow f$ by auto
 from 1 show ?thesis by (rule LeftYieldsImpYields)

qed

lemma *AndYieldsB*:

$\vdash f1 \text{ yields } g \longrightarrow (f \wedge f1) \text{ yields } g$
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) integ-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*:

$\vdash \text{fmore} ; (f \wedge \text{finite}) \longrightarrow \text{fmore}$

proof –

have 1: $\vdash \text{fmore}; (f \wedge \text{finite}) = \bigcirc(\text{finite}; (f \wedge \text{finite}))$

using *FmoreEqvSkipChopFinite* **by** (*metis NextChop inteq-reflection next-d-def*)

have 2: $\vdash \bigcirc(\text{finite}; (f \wedge \text{finite})) \longrightarrow \text{fmore}$

by (*metis ChopAndB FiniteChopFiniteEqvFinite FmoreEqvSkipChopFinite RightChopImpChop inteq-reflection next-d-def*)

have 3: $\vdash (\bigcirc \text{finite}; (f \wedge \text{finite})) \longrightarrow \text{fmore}$ **using** 1 2

by (*metis FmoreEqvSkipChopFinite inteq-reflection next-d-def*)

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

qed

lemma *ChopMoreImpMore*:

$\vdash f; \text{more} \longrightarrow \text{more}$

proof –

have 1: $\vdash (f \wedge \text{finite}) ; \text{more} \longrightarrow \Diamond \text{more}$

by (*rule FiniteChopImpDiamond*)

have 11: $\vdash (f \wedge \text{inf}) ; \text{more} \longrightarrow \text{more}$

by (*metis AndChopA MoreAndInfEqvInf PowerstarEqvSemhelp2 Prop12 inteq-reflection lift-and-com*)

have 2: $\vdash \Diamond \text{more} \longrightarrow \text{more}$

by (*metis FiniteChopMoreEqvMore NowImpDiamond inteq-reflection sometimes-d-def*)

have 3: $\vdash (f \wedge \text{finite}) ; \text{more} \longrightarrow \text{more}$

using 1 2 **by** *fastforce*

have 4: $\vdash f = ((f \wedge \text{finite}) \vee (f \wedge \text{inf}))$

by (*simp add: OrFiniteInf*)

hence 5: $\vdash f; \text{more} = ((f \wedge \text{finite}); \text{more} \vee (f \wedge \text{inf}); \text{more})$

by (*simp add: OrChopEqvRule*)

from 11 3 5 **show** ?thesis **by** *fastforce*

qed

lemma *MoreChopEqvNextDiamond*:

$\vdash \text{fmore} ; f = \bigcirc(\Diamond f)$

proof –

have 1: $\vdash \text{fmore} ; f = (\bigcirc \text{finite}); f$

by (*simp add: FmoreEqvSkipChopFinite LeftChopEqvChop next-d-def*)

have 2: $\vdash (\bigcirc \text{finite}); f = \bigcirc(\text{finite}; f)$

by (*rule NextChop*)

have 3: $\vdash \text{fmore} ; f = \bigcirc(\text{finite}; f)$

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: $\vdash fmore ; (\neg f) = \Box(\Diamond(\neg f))$ **by** (rule *MoreChopEqvNextDiamond*)

have 2: $\vdash \Box(\Diamond(\neg f)) = \Box(\neg(\Box f))$ **by** (auto simp: *always-d-def*)

have 3: $\vdash \Box(\neg(\Box f)) = (\neg (wnext(\Box f)))$ **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 $\vdash f \longrightarrow fmore$

shows $\vdash f; (g \wedge \text{finite}) \longrightarrow fmore$

proof –

have 1: $\vdash f \longrightarrow fmore$ **using** *assms* **by** *auto*

hence 2: $\vdash f; (g \wedge \text{finite}) \longrightarrow fmore ; (g \wedge \text{finite})$ **by** (rule *LeftChopImpChop*)

have 3: $\vdash fmore ; (g \wedge \text{finite}) \longrightarrow fmore$ **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*:

$\vdash \text{di } (f \vee g) = (\text{di } f \vee \text{di } g)$

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:  $\vdash \Box(bi\ f \longrightarrow f) \longrightarrow \Box(bi\ f) \longrightarrow \Box f$  by (rule BoxImpDist)
have 5:  $\vdash \Box(bi\ f) \longrightarrow \Box f$  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*:

```

 $\vdash f \longrightarrow da\ f$ 
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: $\vdash ba (f \wedge g) = \square(bi (f \wedge g))$
by (*rule BaEqvBtBi*)
have 2: $\vdash bi (f \wedge g) = (bi f \wedge bi g)$
by (*simp add: BiAndEqv*)
hence 3: $\vdash \square(bi (f \wedge g)) = \square(bi f \wedge bi g)$
using *BoxEqvBox* **by** *blast*
have 4: $\vdash \square(bi f \wedge bi g) = (\square(bi f) \wedge \square(bi g))$
by (*metis 2 BoxAndBoxEqvBoxRule inteq-reflection*)
have 5: $\vdash ba f = \square(bi f)$
by (*rule BaEqvBtBi*)
have 6: $\vdash ba g = \square(bi g)$
by (*rule BaEqvBtBi*)
from 1 3 4 5 6 **show** *?thesis* **by** *fastforce*
qed

lemma *BaImpBaEqvBa*:
 $\vdash ba (f = g) \longrightarrow (ba f = ba g)$
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: $\vdash ((ba f \longrightarrow ba g) \wedge (ba g \longrightarrow ba f)) = (ba f = ba g)$ **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: $\vdash \Diamond (\Diamond f) = finite;(finite;f)$
by (*simp add: sometimes-d-def*)
have 2: $\vdash finite;(finite;f) = (finite;finite);f$
by (*rule ChopAssoc*)
have 3: $\vdash (finite;finite);f = finite;f$
by (*simp add: LeftChopEqvChop FiniteChopFiniteEqvFinite*)
have 4: $\vdash finite;f = \Diamond f$
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: $\vdash ba\ (f \longrightarrow f1) \longrightarrow bi\ (f \longrightarrow f1)$ **by** (*rule BaImpBi*)
have 2: $\vdash bi\ (f \longrightarrow f1) \longrightarrow f; g \longrightarrow f1; g$ **by** (*rule BiChopImpChop*)
from 1 2 **show** ?thesis **by** *fastforce*

qed

lemma *BaRightChopImpChop*:

$\vdash ba\ (g \longrightarrow g1) \longrightarrow f; g \longrightarrow f; g1$

proof –

have 1: $\vdash ba\ (g \longrightarrow g1) \longrightarrow \Box(g \longrightarrow g1)$ **by** (*rule BaImpBt*)
have 2: $\vdash \Box(g \longrightarrow g1) \longrightarrow f; g \longrightarrow f; g1$ **by** (*rule BoxChopImpChop*)
from 1 2 **show** ?thesis **by** *fastforce*

qed

lemma *ChopAndBaImport*:

$\vdash (f; f1) \wedge ba\ g \longrightarrow (f \wedge g); (f1 \wedge g)$

proof –

have 1: $\vdash ba\ g \wedge (f; f1) \longrightarrow (g \wedge f); (g \wedge f1)$ **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*:

$\vdash ba\ f \wedge g; g1 \longrightarrow (f \wedge g); g1$

by (*meson BaAndChopImport ChopAndB lift-imp-trans*)

lemma *BaAndChopImportB*:

$\vdash ba\ f \wedge g; g1 \longrightarrow (f \wedge g); (ba\ f \wedge g1)$

proof –

have 1: $\vdash ba\ f = ba\ (ba\ f)$
by (*simp add: BaEqvBaBa*)
have 2: $\vdash ba\ (ba\ f) \wedge g; g1 \longrightarrow g; (ba\ f \wedge g1)$
by (*metis AndChopB BaAndChopImport lift-imp-trans*)
have 3: $\vdash ba\ f \wedge g; (ba\ f \wedge g1) \longrightarrow (f \wedge g); (ba\ f \wedge g1)$
by (*simp add: BaAndChopImportA*)
from 1 2 3 **show** ?thesis **by** *fastforce*

qed

lemma *BaImpBaImpBaAnd*:

$\vdash ba\ h \longrightarrow ba(g \longrightarrow ba\ h \wedge g)$

proof –

have 1: $\vdash ba\ h \longrightarrow (g \longrightarrow ba\ h \wedge g)$ **by** *fastforce*

hence 2: $\vdash ba(ba\ h) \longrightarrow ba(g \longrightarrow ba\ h \wedge g)$ **by** (rule *BaImpBa*)
have 3: $\vdash ba\ h = ba(ba\ h)$ **by** (rule *BaEqvBaBa*)
from 2 3 **show** ?thesis **by** fastforce
qed

lemma *BaChopImpChopBa*:

$\vdash ba\ f \longrightarrow g; g1 \longrightarrow g; ((ba\ f) \wedge g1)$

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 $\vdash \text{finite} \longrightarrow g = g1$

shows $\vdash \text{finite} \longrightarrow f;g = f;g1$

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, hide-lams) 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  $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow \text{fin}(\text{init } w1)$ 
 $\vdash (\text{init } w1) \wedge f1 \wedge \text{finite} \longrightarrow \text{fin}(\text{init } w2)$ 
shows  $\vdash (\text{init } w) \wedge (f; f1) \wedge \text{finite} \longrightarrow \text{fin}(\text{init } w2)$ 
proof –
have 1:  $\vdash (\text{init } w) \wedge (f; f1) \wedge \text{finite} \longrightarrow ((\text{init } w) \wedge f \wedge \text{finite}); (f1 \wedge \text{finite})$ 
using StateAndChopImport
by (metis ChopAndFiniteDist inteq-reflection)
have 2:  $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow \text{fin}(\text{init } w1) \wedge \text{finite}$ 
using assms by auto
hence 3:  $\vdash ((\text{init } w) \wedge f \wedge \text{finite}); (f1 \wedge \text{finite}) \longrightarrow (\text{fin}(\text{init } w1) \wedge \text{finite}); (f1 \wedge \text{finite})$ 
by (rule LeftChopImpChop)
have 4:  $\vdash (\text{fin}(\text{init } w1) \wedge \text{finite}); (f1 \wedge \text{finite}) = \Diamond((\text{init } w1) \wedge f1 \wedge \text{finite})$ 
by (rule FinChopEqvDiamond)
have 5:  $\vdash (\text{init } w1) \wedge f1 \wedge \text{finite} \longrightarrow \text{fin}(\text{init } w2)$ 
using assms by auto
hence 6:  $\vdash \Diamond((\text{init } w1) \wedge f1 \wedge \text{finite}) \longrightarrow \Diamond(\text{fin}(\text{init } w2))$ 
by (rule DiamondImpDiamond)
have 7:  $\vdash \Diamond(\text{fin}(\text{init } w2)) \longrightarrow \text{fin}(\text{init } w2)$ 
using DiamondFinImpFin by blast
from 1 3 4 6 7 show ?thesis by fastforce
qed

```

lemma *ChopRep*:

```

assumes  $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow f1 \wedge \text{fin}(\text{init } w1)$ 
 $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1$ 
shows  $\vdash (\text{init } w) \wedge (f; g) \wedge \text{finite} \longrightarrow ((f1 \wedge \text{finite}); g1)$ 
proof –
have 1:  $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow ((f1 \wedge \text{finite}) \wedge \text{fin}(\text{init } w1))$ 
using assms by auto
hence 2:  $\vdash (\text{init } w) \wedge ((f \wedge \text{finite}); (g \wedge \text{finite})) \longrightarrow$ 
 $((f1 \wedge \text{finite}) \wedge \text{fin}(\text{init } w1)); (g \wedge \text{finite})$ 
using StateAndChopImpChopRule by blast
have 3:  $\vdash ((f1 \wedge \text{finite}) \wedge \text{fin}(\text{init } w1)); (g \wedge \text{finite}) =$ 
 $(f1 \wedge \text{finite}); ((\text{init } w1) \wedge (g \wedge \text{finite}))$ 
using AndFinChopEqvStateAndChop by blast
have 4:  $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1$ 
using assms by auto
hence 5:  $\vdash (f1 \wedge \text{finite}); ((\text{init } w1) \wedge g \wedge \text{finite}) \longrightarrow (f1 \wedge \text{finite}); g1$ 

```

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: $\vdash (init\ w) \wedge f \wedge finite \longrightarrow f1 \wedge fin\ (init\ w1)$
 using *assms* by auto
 have 2: $\vdash (init\ w1) \wedge g \wedge finite \longrightarrow g1 \wedge fin\ (init\ w2)$
 using *assms* by auto
 have 3: $\vdash (init\ w) \wedge (f; g) \wedge finite \longrightarrow (f1 \wedge finite); (g1 \wedge fin\ (init\ w2))$
 using 1 2 by (rule *ChopRep*)
 have 4: $\vdash (f1 \wedge finite); (g1 \wedge fin\ (init\ w2)) \longrightarrow (f1 \wedge finite); g1$
 by (rule *ChopAndA*)
 have 5: $\vdash (f1 \wedge finite); (g1 \wedge fin\ (init\ w2)) \longrightarrow (f1 \wedge finite); fin\ (init\ w2)$
 by (rule *ChopAndB*)
 have 6: $\vdash (f1 \wedge finite); fin\ (init\ w2) \longrightarrow fin\ (init\ w2)$
 by (rule *ChopFinImpFin*)
 from 1 2 3 4 5 6 show ?thesis by (meson *Prop12* lift-imp-trans)
 qed

lemma *TrueChopMoreEqvMore*:

$\vdash \#True ; more = more$
 by (metis *ChopMoreImpMore EmptyImpFinite FiniteAndEmptyEqvEmpty FiniteChopMoreEqvMore*
LeftChopImpChop Prop09 int-eq-true int-iffI inteq-reflection)

lemma *FiniteChopFmoreEqvFmore*:

$\vdash finite;fmore = fmore$
 by (metis *TrueChopAndFiniteEqvAndFiniteChopFinite TrueChopMoreEqvMore fmore-d-def inteq-reflection*)

lemma *MoreChopLoop*:

assumes $\vdash f \longrightarrow fmore ; f$
 shows $\vdash finite \longrightarrow \neg f$
 proof –
 have 1: $\vdash f \longrightarrow fmore ; f$
 using *assms* by auto
 hence 11: $\vdash \Diamond (f) \longrightarrow \Diamond (fmore;f)$
 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: $\vdash \Diamond (f) \longrightarrow \bigcirc(\Diamond f)$

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 $\vdash f \wedge \neg g \longrightarrow (f \text{ more} ; (f \wedge \neg g))$
 shows $\vdash f \wedge \text{finite} \longrightarrow g$
 proof –
 have 1: $\vdash f \wedge \neg g \longrightarrow (f \text{ more} ; (f \wedge \neg g))$ using assms by auto
 hence 2: $\vdash \text{finite} \longrightarrow \neg (f \wedge \neg g)$ by (rule MoreChopLoop)
 from 2 show ?thesis by auto
 qed

lemma MoreChopLoopFinite:

assumes $\vdash f \wedge \text{finite} \longrightarrow f \text{ more} ; f$
 shows $\vdash \text{finite} \longrightarrow \neg f$
 proof –
 have 1: $\vdash f \wedge \text{finite} \longrightarrow f \text{ more} ; f$
 using assms by auto
 hence 11: $\vdash \Diamond (f \wedge \text{finite}) \longrightarrow \Diamond (f \text{ more}; f)$
 using DiamondImpDiamond by blast
 have 12: $\vdash \Diamond (f \text{ more}; f) = \text{finite}; (f \text{ more}; f)$
 by (simp add: sometimes-d-def)
 have 13: $\vdash \text{finite}; (f \text{ more}; f) = (\text{finite}; f \text{ more}); f$
 by (rule ChopAssoc)
 have 14: $\vdash \Diamond (f \text{ more}; f) = f \text{ more}; f$
 using FiniteChopFmoreEqvFmore 12 13 by (metis int-eq)
 have 2: $\vdash f \text{ more} ; f = \bigcirc (\Diamond f)$
 using MoreChopEqvNextDiamond by blast
 have 3: $\vdash \Diamond (f \wedge \text{finite}) \longrightarrow \bigcirc (\Diamond f)$
 using 11 14 2 by fastforce
 have 31: $\vdash \Diamond (f \wedge \text{finite}) = ((\Diamond f) \wedge \text{finite})$
 by (metis (no-types, lifting) 3 ChopAndB ChopAndNotChopImp DiamondDiamondEqvDiamond
 DiamondIntroC FiniteChopFiniteEqvFinite FiniteChopInfEqvInf Prop11 Prop12 finite-d-def
 inteq-reflection sometimes-d-def)
 have 32: $\vdash (\Diamond f) \wedge \text{finite} \longrightarrow \bigcirc (\Diamond f)$
 using 3 31 by fastforce
 hence 4: $\vdash \text{finite} \longrightarrow \neg (\Diamond f)$
 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:

$\vdash \text{more} ; f = (f\text{more};f) \vee \text{inf}$
by (*metis AndInfChopEqvAndInf MoreAndInfEqvInf OrChopEqv OrFiniteInf fmore-d-def int-eq*)

lemma *MoreChopLoopFiniteB*:

assumes $\vdash f \longrightarrow \text{more} ; f$

shows $\vdash \text{finite} \longrightarrow \neg f$

proof –

have 1: $\vdash f \longrightarrow \text{more} ; f$

using *assms* **by** *auto*

have 10: $\vdash f \longrightarrow (f\text{more};f) \vee \text{inf}$

using *MoreChopEqvFmoreOrInf assms* **by** *fastforce*

hence 100: $\vdash f \wedge \text{finite} \longrightarrow (f\text{more};f)$

by (*simp add: Prop13 finite-d-def*)

hence 11: $\vdash \Diamond (f \wedge \text{finite}) \longrightarrow \Diamond (f\text{more};f)$

using *DiamondImpDiamond* **by** *blast*

have 12: $\vdash \Diamond (f\text{more};f) = \text{finite};(f\text{more};f)$

by (*simp add: sometimes-d-def*)

have 13: $\vdash \text{finite};(f\text{more};f) = (\text{finite};f\text{more});f$

by (*rule ChopAssoc*)

have 14: $\vdash \Diamond (f\text{more};f) = f\text{more};f$

using *FiniteChopFmoreEqvFmore* 12 13 **by** (*metis int-eq*)

have 2: $\vdash f\text{more} ; f = \bigcirc(\Diamond f)$

using *MoreChopEqvNextDiamond* **by** *blast*

have 3: $\vdash \Diamond (f \wedge \text{finite}) \longrightarrow \bigcirc(\Diamond f)$

using 11 14 2 **by** *fastforce*

have 31: $\vdash \Diamond (f \wedge \text{finite}) = ((\Diamond f) \wedge \text{finite})$

by (*metis (no-types, hide-lams) ChopAndA ChopAndB ChopAndNotChopImp FiniteChopFiniteEqvFinite FiniteChopInfEqvInf Prop11 Prop12 finite-d-def inteq-reflection sometimes-d-def*)

have 32: $\vdash (\Diamond f) \wedge \text{finite} \longrightarrow \bigcirc(\Diamond f)$

using 3 31 **by** *fastforce*

hence 4: $\vdash \text{finite} \longrightarrow \neg (\Diamond f)$

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 $\vdash (f \wedge \neg g) \wedge \text{finite} \longrightarrow (f\text{more} ; (f \wedge \neg g))$

shows $\vdash f \wedge \text{finite} \longrightarrow g$

proof –

have 1: $\vdash (f \wedge \neg g) \wedge \text{finite} \longrightarrow (f\text{more} ; (f \wedge \neg g))$ **using** *assms* **by** *auto*

hence 2: $\vdash \text{finite} \longrightarrow \neg (f \wedge \neg g)$ **using** *MoreChopLoopFinite* **by** (*simp add: MoreChopLoopFinite*)

from 2 **show** *?thesis* **by** (*simp add: Valid-def*)

qed

lemma *MoreChopContraFiniteB*:

assumes $\vdash (f \wedge \neg g) \longrightarrow (\text{more} ; (f \wedge \neg g))$

shows $\vdash f \wedge \text{finite} \longrightarrow g$

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:  $\vdash h; (f \wedge \neg g) \longrightarrow \text{more}; (f \wedge \neg g)$  using 2 by (rule LeftChopImpChop)
have 5:  $\vdash f \wedge \neg g \longrightarrow \text{more}; (f \wedge \neg g)$  using 1 3 4 by fastforce
from 5 show ?thesis using MoreChopContraFiniteB by auto
qed

```

6.8 Properties of Chopstar and Chopplus

lemma *FPowerstardef*:

```

 $\vdash \text{fpowerstar } f = (\exists n. \text{power } f n)$ 
by (simp add: fpowerstar-d-def)

```

lemma *Powerstardef*:

```

 $\vdash \text{powerstar } f = (\text{fpowerstar } f); (\text{empty} \vee (f \wedge \text{inf}))$ 
by (simp add: fpowerstar-d-def powerstar-d-def)

```

lemma *Chopstardef*:

```

 $\vdash \text{chopstar } f = \text{powerstar } (f \wedge \text{more})$ 
by (simp add: chopstar-d-def)

```

lemma *AndEmptyChopAndEmptyEqvAndEmpty*:

```

 $\vdash (f \wedge \text{empty}); (f \wedge \text{empty}) = (f \wedge \text{empty})$ 

```

proof –

```

have 1:  $\vdash (f \wedge \text{empty}); (f \wedge \text{empty}) \longrightarrow (f \wedge \text{empty})$ 
  by (metis ChopAndB ChopEmpty int-eq)
have 2:  $\vdash (f \wedge \text{empty}) \longrightarrow (f \wedge \text{empty}); (f \wedge \text{empty})$ 
  by (auto simp add: Valid-def itl-defs)
  (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

```

lemma *PowerCommute*:

```

 $\vdash (f \wedge \text{finite}); \text{power } f n = \text{power } f n; (f \wedge \text{finite})$ 

```

proof

```

(induct n)
case 0
then show ?case
by (metis ChopEmpty EmptyChop inteq-reflection power-d.pow-0)
next
case (Suc n)
then show ?case
by (metis ChopAssoc inteq-reflection power-d.pow-Suc)
qed

```

lemma *ChopInductL*:

```

assumes  $\vdash g \vee f; h \longrightarrow h$ 
shows  $\vdash (\text{power } f n); g \longrightarrow h$ 
proof
(induct n)

```

```

case 0
then show ?case using EmptyChop assms
by (metis MP Prop12 int-eq int-iffD1 int-simps(33) pow-0)
next
case (Suc n)
then show ?case using assms
by (metis AndChopA ChopAssoc Prop05 Prop11 RightChopImpChop lift-imp-trans pow-Suc)
qed

```

lemma *ChopInductFiniteL:*

```

assumes  $\vdash g \vee (f \wedge \text{finite}); h \longrightarrow h$ 
shows  $\vdash (\text{power } f \ n); g \longrightarrow h$ 

```

proof

```

(induct n)
case 0
then show ?case using EmptyChop assms
by (metis MP Prop12 int-eq int-iffD1 int-simps(33) pow-0)
next
case (Suc n)
then show ?case using assms
by (metis ChopAndB ChopAssoc Prop05 Prop10 inteq-reflection lift-imp-trans pow-Suc)
qed

```

lemma *ChopInductFiniteMoreL:*

```

assumes  $\vdash g \vee ((f \wedge \text{more}) \wedge \text{finite}); h \longrightarrow h$ 
shows  $\vdash (\text{power } f \ n); g \longrightarrow h$ 

```

proof

```

(induct n)
case 0
then show ?case using assms by (metis ChopInductFiniteL pow-0)
next
case (Suc n)
then show ?case
proof -
have 1:  $\vdash \text{power } f \ (\text{Suc } n); g = ((f \wedge \text{finite}); \text{power } f \ n); g$ 
by simp
have 2:  $\vdash ((f \wedge \text{finite}); \text{power } f \ n); g = (f \wedge \text{finite}); ((\text{power } f \ n); g)$ 
by (meson ChopAssoc Prop11)
have 3:  $\vdash (f \wedge \text{finite}); ((\text{power } f \ n); g) \longrightarrow (f \wedge \text{finite}); h$ 
by (simp add: RightChopImpChop Suc.hyps)
have 31:  $\vdash f = ((f \wedge \text{more}) \vee (f \wedge \text{empty}))$ 
unfolding empty-d-def by fastforce
have 32:  $\vdash (f \wedge \text{finite}) = ((f \wedge \text{more}) \wedge \text{finite}) \vee ((f \wedge \text{empty}) \wedge \text{finite})$ 
using 31 by fastforce
have 4:  $\vdash (f \wedge \text{finite}); h = ((f \wedge \text{more}) \wedge \text{finite}); h \vee ((f \wedge \text{empty}) \wedge \text{finite}); h$ 
using 32 OrChopEqvRule by blast
have 5:  $\vdash ((f \wedge \text{more}) \wedge \text{finite}); h \longrightarrow h$ 
using assms by auto
have 6:  $\vdash ((f \wedge \text{empty}) \wedge \text{finite}); h \longrightarrow h$ 
by (metis AndChopA AndChopB EmptyChop inteq-reflection lift-imp-trans)

```

from 5 6 4 3 2 1 show ?thesis by fastforce
qed
qed

lemma *ChopInductInfL*:
assumes $\vdash g \vee f; h \longrightarrow h$
shows $\vdash ((\text{power } f \ n); (f \wedge \text{inf})); g \longrightarrow h$
proof
(induct n)
case 0
then show ?case **using** *assms*
by (*metis* (*no-types*, *lifting*) *AndInfChopEqvAndInf ChopAssoc ChopInductFiniteL PowerstarEqvSemhelp3 Prop03 Prop10 Prop12 inteq-reflection*)
next
case (*Suc n*)
then show ?case **using** *assms*
proof –
have $\vdash (f \wedge \text{finite}); (\text{power } f \ n; ((f \wedge \text{inf}); g)) = (\text{power } f \ (\text{Suc } n); (f \wedge \text{inf}); g)$
by (*metis ChopAssoc inteq-reflection pow-Suc*)
then show ?thesis
by (*metis* (*no-types*, *lifting*) *AndChopA ChopAndB ChopAssoc Prop03 Prop10 Suc assms int-eq lift-imp-trans*)

qed
qed

lemma *ChopInductInfMoreL*:
assumes $\vdash g \vee f; h \longrightarrow h$
shows $\vdash ((\text{power } f \ n); ((f \wedge \text{more}) \wedge \text{inf})); g \longrightarrow h$
using *ChopInductInfL*
by (*metis AndMoreAndInfEqvAndInf assms inteq-reflection*)

lemma *ChopInductR*:
assumes $\vdash g \vee h; f \longrightarrow h$
shows $\vdash g; (\text{power } f \ n) \longrightarrow h$
proof
(induct n)
case 0
then show ?case **using** *ChopEmpty assms*
by (*metis MP Prop12 int-iffD2 int-simps(33) inteq-reflection pow-0*)
next
case (*Suc n*)
then show ?case **using** *assms*
proof –
have 1: $\vdash \text{power } f \ n; (f \wedge \text{finite}) = \text{power } f \ (\text{Suc } n)$
using *PowerCommute* **by** *fastforce*
have 2: $\vdash h; (\text{finite} \wedge f) \longrightarrow h$
by (*metis* (*no-types*) *ChopAndA Prop05 assms inteq-reflection lift-and-com lift-imp-trans*)
then show ?thesis
by (*metis* 1 *AndChopB ChopAssoc Prop10 Suc int-eq lift-and-com lift-imp-trans*)
qed

qed

lemma *ChopInductInfR*:

assumes $\vdash g \vee h; f \longrightarrow h$

shows $\vdash g; ((\text{power } f \ n); (f \wedge \text{inf})) \longrightarrow h$

using *assms*

by (*metis ChopAndA ChopAssoc ChopInductR LeftChopImpChop Prop05 int-iffD1 lift-imp-trans*)

lemma *ChopExistPower*:

$\vdash (g; (\exists n. \text{power } f \ n)) = (\exists n. g; \text{power } f \ n)$

using *ChopExist* **by** *fastforce*

lemma *ExistChopPower*:

$\vdash (\exists n. (\text{power } f \ n); g) = (\exists n. \text{power } f \ n); g$

using *ExistChop* **by** *fastforce*

lemma *PowerStarCommute*:

$\vdash (f \wedge \text{finite}); (\exists n. \text{power } f \ n) = (\exists n. \text{power } f \ n); (f \wedge \text{finite})$

proof –

have 1: $\vdash (f \wedge \text{finite}); (\exists n. \text{power } f \ n) =$
 $(\exists n. (f \wedge \text{finite}); \text{power } f \ n)$

using *ChopExistPower* **by** *blast*

have 2: $\vdash (\exists n. (f \wedge \text{finite}); \text{power } f \ n) =$
 $(\exists n. (\text{power } f \ n); (f \wedge \text{finite}))$

using *PowerCommute* **by** *fastforce*

have 3: $\vdash (\exists n. (\text{power } f \ n); (f \wedge \text{finite})) =$
 $(\exists n. (\text{power } f \ n); (f \wedge \text{finite}))$

using *ExistChopPower* **by** *blast*

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

qed

lemma *PowerSucAndEmptyEqvAndEmpty*:

$\vdash (\text{power } (f \wedge \text{empty}) (\text{Suc } n)) = (f \wedge \text{empty})$

proof

(*induct n*)

case 0

then show *?case* **using** *ChopEmpty*

by (*metis (no-types, lifting) FiniteAndEmptyEqvEmpty Prop10 Prop12 int-iffD1 inteq-reflection*
pow-0 pow-Suc)

next

case (*Suc n*)

then show *?case*

by (*metis AndEmptyChopAndEmptyEqvAndEmpty EmptyImpFinite Prop10 Prop12 int-iffD1*
inteq-reflection pow-Suc)

qed

lemma *FiniteOr*:

$\vdash ((f \vee g) \wedge \text{finite}) = ((f \wedge \text{finite}) \vee (g \wedge \text{finite}))$

by *auto*

lemma *PowerOr*:

$\vdash (\text{power } (f \vee g) (\text{Suc } n)) = (((f \wedge \text{finite}); \text{power } (f \vee g) n) \vee ((g \wedge \text{finite}); \text{power } (f \vee g) n))$

by (*simp add: FiniteOr OrChopEqvRule*)

lemma *PowerEmptyOrMore*:

$\vdash (\text{power } ((f \wedge \text{empty}) \vee (f \wedge \text{more})) (\text{Suc } n)) = ((f \wedge \text{empty}); (\text{power } ((f \wedge \text{empty}) \vee (f \wedge \text{more})) n) \vee (f \wedge \text{fmore}); (\text{power } ((f \wedge \text{empty}) \vee (f \wedge \text{more})) n))$

proof –

have *f2*: $\vdash (f \wedge \text{empty}) = (\text{empty} \wedge f)$

by (*meson lift-and-com*)

have *f3*: $\vdash ((f \wedge \text{empty}) \wedge \text{finite}) = (\text{finite} \wedge f \wedge \text{empty})$

by (*meson lift-and-com*)

have *f4*: $\vdash (\text{empty} \wedge f) = (\text{finite} \wedge f \wedge \text{empty})$

using *FiniteAndEmptyEqvEmpty* **by** *auto*

have $\vdash ((f \wedge \text{more}) \wedge \text{finite}) = (f \wedge \text{fmore})$

by (*meson AndMoreAndFiniteEqvAndFmore*)

then show *?thesis*

using *f4 f3 f2* **by** (*metis PowerOr inteq-reflection*)

qed

lemma *PSEqvEmptyOrChopPS*:

$\vdash \text{powerstar } f = (\text{empty} \vee f; \text{powerstar } f)$

using *PowerstarEqvSem Valid-def* **by** *blast*

lemma *EmptyImpCS*:

$\vdash \text{empty} \longrightarrow f^*$

proof –

have *1*: $\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$ **by** (*rule ChopstarEqv*)

have *2*: $\vdash \text{empty} \longrightarrow \text{empty} \vee (f \wedge \text{more}); f^*$ **by** *auto*

from *1 2* **show** *?thesis* **by** *fastforce*

qed

lemma *CSEqvOrChopCS*:

$\vdash f^* = (\text{empty} \vee (f; f^*))$

proof –

have *1*: $\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$ **by** (*rule ChopstarEqv*)

have *2*: $\vdash (f \wedge \text{more}); f^* \longrightarrow f; f^*$ **by** (*rule AndChopA*)

have *3*: $\vdash f^* \longrightarrow \text{empty} \vee f; f^*$ **using** *1 2* **by** (*metis int-iffD1 Prop08*)

have *4*: $\vdash \text{empty} \longrightarrow f^*$ **by** (*rule EmptyImpCS*)

have *5*: $\vdash f \longrightarrow \text{empty} \vee (f \wedge \text{more})$ **by** (*auto simp: empty-d-def*)

have *6*: $\vdash f; f^* \longrightarrow f^* \vee (f \wedge \text{more}); f^*$ **using** *5* **by** (*rule EmptyOrChopImpRule*)

have *7*: $\vdash f^* \longrightarrow \text{empty} \vee (f \wedge \text{more}); f^*$ **using** *1* **by** *fastforce*

have *8*: $\vdash f; f^* \longrightarrow \text{empty} \vee (f \wedge \text{more}); f^*$ **using** *6 7* **by** *fastforce*

hence *9*: $\vdash f; f^* \longrightarrow f^*$ **using** *1* **by** *fastforce*

have *10*: $\vdash \text{empty} \vee f; f^* \longrightarrow f^*$ **using** *9 4* **by** *fastforce*

from *3 10* **show** *?thesis* **by** *fastforce*

qed

lemma *PowerChopCommute*:

$\vdash ((f \wedge \text{more}) \wedge \text{finite}); \text{power } (f \wedge \text{more}) \ n = \text{power } (f \wedge \text{more}) \ n; ((f \wedge \text{more}) \wedge \text{finite})$

using *PowerCommute* **by** *auto*

lemma *ChopExist*:

$\vdash (g; (\exists n. \text{power } (f \wedge \text{more}) \ n)) = (\exists n. g; \text{power } (f \wedge \text{more}) \ n)$

using *ChopExistPower* **by** *auto*

lemma *ExistChop*:

$\vdash (\exists n. (\text{power } (f \wedge \text{more}) \ n); g) = (\exists n. \text{power } (f \wedge \text{more}) \ n); g$

using *ExistChopPower* **by** *auto*

lemma *FPowerstarInductL*:

assumes $\vdash g \vee (f \wedge \text{finite}); h \longrightarrow h$

shows $\vdash (f\text{powerstar } f); g \longrightarrow h$

proof –

have 1: $\vdash (f\text{powerstar } f); g = (\exists n. \text{power } f \ n); g$

by (*simp add: fpowerstar-d-def LeftChopEqvChop*)

have 2: $\vdash (\exists n. \text{power } f \ n); g =$

$(\exists n. (\text{power } f \ n); g)$

using *ExistChopPower* **by** *fastforce*

have 3: $\bigwedge n. \vdash (\text{power } f \ n); g \longrightarrow h$

using *ChopInductFiniteL assms* **by** *blast*

have 4: $\vdash (\exists n. ((\text{power } f \ n)); g) \longrightarrow h$

using 3 **by** (*simp add: Valid-def*) *blast*

from 1 2 4 **show** *?thesis* **by** (*metis inteq-reflection*)

qed

lemma *FPowerstarInductMoreL*:

assumes $\vdash g \vee ((f \wedge \text{more}) \wedge \text{finite}); h \longrightarrow h$

shows $\vdash (f\text{powerstar } f); g \longrightarrow h$

proof –

have 1: $\vdash (f\text{powerstar } f); g = (\exists n. \text{power } f \ n); g$

by (*simp add: fpowerstar-d-def LeftChopEqvChop*)

have 2: $\vdash (\exists n. \text{power } f \ n); g =$

$(\exists n. (\text{power } f \ n); g)$

using *ExistChopPower* **by** *fastforce*

have 3: $\bigwedge n. \vdash (\text{power } f \ n); g \longrightarrow h$

using *ChopInductFiniteMoreL assms* **by** *blast*

have 4: $\vdash (\exists n. ((\text{power } f \ n)); g) \longrightarrow h$

using 3 **by** (*simp add: Valid-def*) *blast*

from 1 2 4 **show** *?thesis* **by** (*metis inteq-reflection*)

qed

lemma *PowerstarInductL*:

assumes $\vdash g \vee f; h \longrightarrow h$

shows $\vdash (\text{powerstar } f); g \longrightarrow h$

proof –

have 1: $\vdash (\text{powerstar } f); g = ((\exists n. \text{power } f \ n); (\text{empty} \vee (f \wedge \text{inf}))); g$

by (*simp add: powerstar-d-def LeftChopEqvChop*)

have 11: $\vdash ((\exists n. \text{power } f \ n); (\text{empty} \vee (f \wedge \text{inf}))); g =$
 $(\exists n. \text{power } f \ n); ((\text{empty} \vee (f \wedge \text{inf}))); g$
by (*meson ChopAssoc Prop11*)
have 2: $\vdash (\exists n. \text{power } f \ n); ((\text{empty} \vee (f \wedge \text{inf}))); g =$
 $(\exists n. ((\text{power } f \ n); ((\text{empty} \vee (f \wedge \text{inf}))); g))$
using *ExistChopPower* **by** *fastforce*
have 3: $\bigwedge n. \vdash (\text{power } f \ n); g \longrightarrow h$
using *ChopInductL assms* **by** *blast*
have 31: $\bigwedge n. \vdash ((\text{power } f \ n); (f \wedge \text{inf})); g \longrightarrow h$
using *ChopInductInfL assms* **by** *blast*
have 33: $\bigwedge n. \vdash ((\text{power } f \ n); g \vee ((\text{power } f \ n); (f \wedge \text{inf}))); g =$
 $((\text{power } f \ n); (\text{empty} \vee (f \wedge \text{inf}))); g$
by (*metis ChopEmpty ChopOrEqv OrChopEqv inteq-reflection*)
have 32: $\bigwedge n. \vdash ((\text{power } f \ n); (\text{empty} \vee (f \wedge \text{inf}))); g \longrightarrow h$
by (*metis 3 31 33 Prop02 inteq-reflection*)
have 4: $\vdash (\exists n. ((\text{power } f \ n); (\text{empty} \vee (f \wedge \text{inf}))); g) \longrightarrow h$
using 32 **by** *fastforce*
from 1 11 2 4 show ?thesis
by (*metis Semantics.ExistChop inteq-reflection*)
qed

lemma *ChopstarInductL:*

assumes $\vdash g \vee f; h \longrightarrow h$

shows $\vdash (\text{chopstar } f); g \longrightarrow h$

proof –

have 1: $\vdash (\text{chopstar } f); g = ((\exists n. \text{power } (f \wedge \text{more}) \ n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))); g$
by (*simp add: chopstar-d-def powerstar-d-def LeftChopEqvChop*)
have 11: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) \ n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))); g =$
 $(\exists n. \text{power } (f \wedge \text{more}) \ n); ((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))); g$
by (*meson ChopAssoc Prop11*)
have 2: $\vdash (\exists n. \text{power } (f \wedge \text{more}) \ n); ((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))); g =$
 $(\exists n. (\text{power } (f \wedge \text{more}) \ n); (((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))); g))$
using *ExistChopPower* **by** *fastforce*
have 21: $\vdash g \vee (f \wedge \text{more}); h \longrightarrow h$
using *AndChopA Prop03 Prop10 assms int-simps(33) inteq-reflection* **by** *fastforce*
have 3: $\bigwedge n. \vdash (\text{power } (f \wedge \text{more}) \ n); g \longrightarrow h$
using 21 *ChopInductL[of g LIFT(f \wedge more) h] assms* **by** *auto*
have 31: $\bigwedge n. \vdash ((\text{power } (f \wedge \text{more}) \ n); (f \wedge \text{more}) \wedge \text{inf}); g \longrightarrow h$
using *assms*
by (*metis (no-types, lifting) AndChopA ChopInductInfL Prop03 Prop10 int-eq-true int-simps(33) inteq-reflection lift-imp-trans*)
have 32: $\bigwedge n. \vdash ((\text{power } (f \wedge \text{more}) \ n); g \vee ((\text{power } (f \wedge \text{more}) \ n); (f \wedge \text{more}) \wedge \text{inf})); g =$
 $((\text{power } (f \wedge \text{more}) \ n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))); g$
by (*meson ChopAssoc ChopOrEqvRule EmptyOrChopEqv Prop04 Prop06*)
have 41: $\bigwedge n. \vdash ((\text{power } (f \wedge \text{more}) \ n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))); g \longrightarrow h$
by (*metis 3 31 32 Prop02 inteq-reflection*)
have 4: $\vdash (\exists n. ((\text{power } (f \wedge \text{more}) \ n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))); g) \longrightarrow h$
using 41 **by** *fastforce*
from 1 11 2 4 show ?thesis
by (*metis Semantics.ExistChop inteq-reflection*)

qed

lemma *ChopstarInductMoreL*:

assumes $\vdash g \vee (f \wedge \text{more}); h \longrightarrow h$

shows $\vdash (\text{chopstar } f); g \longrightarrow h$

proof –

have 1: $\vdash (\text{chopstar } f); g = ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))); g$

by (*simp add: chopstar-d-def powerstar-d-def LeftChopEqvChop*)

have 11: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))); g =$
 $(\exists n. \text{power } (f \wedge \text{more}) n); ((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))); g$

by (*meson ChopAssoc Prop11*)

have 2: $\vdash (\exists n. \text{power } (f \wedge \text{more}) n); ((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))); g =$
 $(\exists n. (\text{power } (f \wedge \text{more}) n); ((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))); g)$

using *ExistChopPower* **by** *fastforce*

have 3: $\bigwedge n. \vdash (\text{power } (f \wedge \text{more}) n); g \longrightarrow h$

using *ChopInductL assms* **by** (*metis*)

have 31: $\vdash (\exists n. (\text{power } (f \wedge \text{more}) n); g) \longrightarrow h$

using 3 **by** *fastforce*

have 32: $\bigwedge n. \vdash ((\text{power } (f \wedge \text{more}) n); ((f \wedge \text{more}) \wedge \text{inf})); g \longrightarrow h$

using *assms*

by (*metis (no-types, lifting) AndChopA ChopInductInfL int-eq-true inteq-reflection*)

have 33: $\vdash (\exists n. ((\text{power } (f \wedge \text{more}) n); ((f \wedge \text{more}) \wedge \text{inf}))); g \longrightarrow h$

using 32 **by** *fastforce*

have 34: $\vdash (\exists n. (\text{power } (f \wedge \text{more}) n); g) \vee$

$(\exists n. ((\text{power } (f \wedge \text{more}) n); ((f \wedge \text{more}) \wedge \text{inf}))); g \longrightarrow h$

using 31 33 **by** *fastforce*

have 35: $\vdash (\exists n. (\text{power } (f \wedge \text{more}) n); g \vee ((\text{power } (f \wedge \text{more}) n); ((f \wedge \text{more}) \wedge \text{inf}))); g$
 $\longrightarrow h$

using 34 **by** *fastforce*

have 36: $\bigwedge n. \vdash ((\text{power } (f \wedge \text{more}) n); g \vee ((\text{power } (f \wedge \text{more}) n); ((f \wedge \text{more}) \wedge \text{inf}))); g =$
 $((\text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))); g$

by (*meson ChopAssoc ChopOrEqvRule EmptyOrChopEqv Prop04 Prop06*)

have 4: $\vdash (\exists n. ((\text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))); g \longrightarrow h$

using 36 35 **by** *fastforce*

from 1 11 2 4 **show** *?thesis*

by (*metis Semantics.ExistChop inteq-reflection*)

qed

lemma *FPowerstarInductR*:

assumes $\vdash g \vee h; f \longrightarrow h$

shows $\vdash g; (\text{fpowerstar } f) \longrightarrow h$

proof –

have 1: $\vdash g; (\text{fpowerstar } f) = g; (\exists n. \text{power } f n)$

by (*simp add: fpowerstar-d-def*)

have 2: $\vdash (g; (\exists n. \text{power } f n)) = (\exists n. g; (\text{power } f n))$

using *ChopExistPower* **by** *blast*

have 3: $\bigwedge n. \vdash g; (\text{power } f n) \longrightarrow h$

using *ChopInductR assms* **by** *blast*

have 4: $\vdash (\exists n. g; (\text{power } f n)) \longrightarrow h$

using 3 **by** *fastforce*

from 1 2 4 show ?thesis by (metis inteq-reflection)
qed

lemma *PowerstarInductR*:

assumes $\vdash g \vee h; f \longrightarrow h$

shows $\vdash g; (\text{powerstar } f) \longrightarrow h$

proof –

have 1: $\vdash g; (\text{powerstar } f) = g; ((\exists n. \text{power } f \ n); (\text{empty} \vee (f \wedge \text{inf})))$

by (simp add: chopstar-d-def powerstar-d-def)

have 11: $\vdash g; ((\exists n. \text{power } f \ n); (\text{empty} \vee (f \wedge \text{inf}))) =$
 $(g; (\exists n. \text{power } f \ n)); (\text{empty} \vee (f \wedge \text{inf}))$

using *ChopAssoc* **by** blast

have 2: $\vdash (g; (\exists n. \text{power } f \ n)) = (\exists n. g; (\text{power } f \ n))$

using *ChopExistPower* **by** blast

hence 21: $\vdash (g; (\exists n. \text{power } f \ n)); (\text{empty} \vee (f \wedge \text{inf})) =$
 $(\exists n. g; (\text{power } f \ n)); (\text{empty} \vee (f \wedge \text{inf}))$

using *LeftChopEqvChop* **by** blast

have 3: $\bigwedge n. \vdash g; (\text{power } f \ n) \longrightarrow h$

using *ChopInductR* *assms* **by** blast

have 31: $\bigwedge n. \vdash g; ((\text{power } f \ n); (f \wedge \text{inf})) \longrightarrow h$

using *ChopInductInfR* *assms* **by** blast

have 32: $\bigwedge n. \vdash g; ((\text{power } f \ n); (\text{empty} \vee (f \wedge \text{inf}))) \longrightarrow h$
using 3 31

by (metis *ChopEmpty ChopOrEqv Prop02 inteq-reflection*)

have 4: $\vdash (\exists n. g; ((\text{power } f \ n); (\text{empty} \vee (f \wedge \text{inf})))) \longrightarrow h$
using 32 **by** fastforce

from 1 11 2 21 4 show ?thesis

by (metis *Semantics.ChopExist Semantics.ExistChop int-eq*)

qed

lemma *ChopstarInductR*:

assumes $\vdash g \vee h; f \longrightarrow h$

shows $\vdash g; (\text{chopstar } f) \longrightarrow h$

proof –

have 1: $\vdash g; (\text{chopstar } f) =$
 $g; ((\exists n. \text{power } (f \wedge \text{more}) \ n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})))$

by (simp add: chopstar-d-def powerstar-d-def)

have 11: $\vdash g; ((\exists n. \text{power } (f \wedge \text{more}) \ n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))) =$
 $(g; (\exists n. \text{power } (f \wedge \text{more}) \ n)); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))$

using *ChopAssoc* **by** blast

have 2: $\vdash (g; (\exists n. \text{power } (f \wedge \text{more}) \ n)); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) =$
 $((\exists n. g; \text{power } (f \wedge \text{more}) \ n)); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))$

by (simp add: *ChopExistPower LeftChopEqvChop*)

have 21: $\vdash g \vee h; (f \wedge \text{more}) \longrightarrow h$

using *ChopAndA* *assms* **by** fastforce

have 3: $\bigwedge n. \vdash g; (\text{power } (f \wedge \text{more}) \ n) \longrightarrow h$

using 21 *ChopInductR*[of $g \ h \ \text{LIFT}(f \wedge \text{more})$] *assms* **by** auto

have 31: $\bigwedge n. \vdash g; ((\text{power } (f \wedge \text{more}) \ n); (f \wedge \text{inf})) \longrightarrow h$

using *assms* 3

by (metis 21 *AndMoreAndInfEqvAndInf ChopInductInfR inteq-reflection*)

have 32: $\bigwedge n. \vdash g;((\text{power } (f \wedge \text{more}) \ n);(\text{empty} \vee (f \wedge \text{inf}))) \longrightarrow h$
using 3 31
by (metis ChopEmpty ChopOrEqv Prop02 inteq-reflection)
have 4: $\vdash (\exists n. g;((\text{power } (f \wedge \text{more}) \ n);(\text{empty} \vee (f \wedge \text{inf})))) \longrightarrow h$
using 32 **by** fastforce
from 1 11 2 4 **show** ?thesis
by (metis Semantics.ChopExist Semantics.ExistChop AndMoreAndInfEqvAndInf
inteq-reflection)
qed

lemma ChopstarInductMoreR:

assumes $\vdash g \vee h; (f \wedge \text{more}) \longrightarrow h$

shows $\vdash g; (\text{chopstar } f) \longrightarrow h$

proof –

have 1: $\vdash g; (\text{chopstar } f) = g; ((\exists n. \text{power } (f \wedge \text{more}) \ n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})))$

by (simp add: chopstar-d-def powerstar-d-def)

have 11: $\vdash g; ((\exists n. \text{power } (f \wedge \text{more}) \ n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))) =$
 $(g; (\exists n. \text{power } (f \wedge \text{more}) \ n)); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))$

using ChopAssoc **by** blast

have 2: $\vdash (g; (\exists n. \text{power } (f \wedge \text{more}) \ n)); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) =$
 $((\exists n. g; \text{power } (f \wedge \text{more}) \ n)); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))$

by (simp add: ChopExistPower LeftChopEqvChop)

have 3: $\bigwedge n. \vdash g; (\text{power } (f \wedge \text{more}) \ n) \longrightarrow h$

using ChopInductR assms **by** (metis)

have 31: $\bigwedge n. \vdash g; ((\text{power } (f \wedge \text{more}) \ n); (f \wedge \text{inf})) \longrightarrow h$

using assms

by (metis ChopInductInfR AndMoreAndInfEqvAndInf inteq-reflection)

have 32: $\bigwedge n. \vdash g; ((\text{power } (f \wedge \text{more}) \ n); (\text{empty} \vee (f \wedge \text{inf}))) \longrightarrow h$

using 3 31

by (metis ChopEmpty ChopOrEqv Prop02 inteq-reflection)

have 4: $\vdash (\exists n. g; ((\text{power } (f \wedge \text{more}) \ n); (\text{empty} \vee (f \wedge \text{inf})))) \longrightarrow h$

using 32 **by** fastforce

from 1 11 2 4 **show** ?thesis

by (metis Semantics.ChopExist Semantics.ExistChop AndMoreAndInfEqvAndInf
inteq-reflection)
qed

lemma FPSEqvEmptyOrFiniteChopFPS:

$\vdash \text{fpowerstar } f = (\text{empty} \vee (f \wedge \text{finite}); \text{fpowerstar } f)$

using FPowerstarEqvSem Valid-def **by** blast

lemma FPSAndMoreImpFPS:

$\vdash \text{fpowerstar } (f \wedge \text{more}) \longrightarrow \text{fpowerstar } f$

proof –

have 1: $\vdash ((f \wedge \text{more}) \wedge \text{finite}) \longrightarrow (f \wedge \text{finite})$

by auto

have 11: $\vdash ((f \wedge \text{more}) \wedge \text{finite}); \text{fpowerstar } f \longrightarrow (f \wedge \text{finite}); \text{fpowerstar } f$

using 1 LeftChopImpChop **by** blast

have 2: $\vdash \text{empty} \vee ((f \wedge \text{more}) \wedge \text{finite}); \text{fpowerstar } f \longrightarrow \text{fpowerstar } f$

using 11 FPSEqvEmptyOrFiniteChopFPS[of f]

by *fastforce*
have 3: $\vdash \text{fpowerstar } (f \wedge \text{more}); \text{empty} \longrightarrow \text{fpowerstar } f$
using 2 *FPowerstarInductL* **by** *blast*
from 2 3 **show** *?thesis* **by** (*metis ChopEmpty int-eq*)
qed

lemma *FPSImpAndMoreFPS*:

$\vdash \text{fpowerstar } f \longrightarrow \text{fpowerstar } (f \wedge \text{more})$
by (*meson ChopEmpty FPSeqvEmptyOrFiniteChopFPS FPowerstarInductMoreL int-iffD2 lift-imp-trans*)

lemma *FPSAndMoreEqvFPS*:

$\vdash \text{fpowerstar } (f \wedge \text{more}) = \text{fpowerstar } f$
using *FPSAndMoreImpFPS FPSImpAndMoreFPS* **by** *fastforce*

lemma *ChopstarImpPowerstar*:

$\vdash f^* \longrightarrow \text{powerstar } f$
by (*metis ChopEmpty ChopstarInductL PSeqvEmptyOrChopPS int-eq int-iffD2*)

lemma *PowerstarImpChopstar*:

$\vdash \text{powerstar } f \longrightarrow f^*$
by (*metis CSeqvOrChopCS ChopEmpty PowerstarInductL int-iffD2 inteq-reflection*)

lemma *ChopstarEqvPowerstar*:

$\vdash f^* = \text{powerstar } f$
using *ChopstarImpPowerstar PowerstarImpChopstar* **by** *fastforce*

lemma *CSAndMoreEqvAndMoreChop*:

$\vdash (f^* \wedge \text{more}) = (f \wedge \text{more}); f^*$
proof –
have 1: $\vdash (\text{empty} \vee (f \wedge \text{more}); f^*) \wedge \text{more} \longrightarrow (f \wedge \text{more}); f^*$
by (*auto simp: empty-d-def*)
have 2: $\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$
by (*rule ChopstarEqv*)
have 3: $\vdash f^* \wedge \text{more} \longrightarrow (f \wedge \text{more}); f^*$
using 1 2 **by** *fastforce*
have 4: $\vdash (f \wedge \text{more}); f^* \longrightarrow f^*$
using 2 **by** *fastforce*
have 5: $\vdash (f \wedge \text{more}) \longrightarrow \text{more}$
by *auto*
hence 6: $\vdash (f \wedge \text{more}); f^* \longrightarrow \text{more}$
by (*rule LeftChopImpMoreRule*)
have 7: $\vdash (f \wedge \text{more}); f^* \longrightarrow f^* \wedge \text{more}$
using 4 6 **by** *fastforce*
from 3 7 **show** *?thesis* **by** *fastforce*
qed

lemma *AndMoreCSeqvAndFmoreOrInf*:

$\vdash (f \wedge \text{more}); f^* = ((f \wedge \text{fmore}); f^* \vee (f \wedge \text{inf}))$
proof –
have 1: $\vdash (f \wedge \text{more}) = ((f \wedge \text{fmore}) \vee (f \wedge \text{inf}))$

by (metis AndMoreAndFiniteEqvAndFmore AndMoreAndInfEqvAndInf OrFiniteInf int-eq)
 hence 2: $\vdash (f \wedge \text{more}); f^* = ((f \wedge \text{fmore}) \vee (f \wedge \text{inf})); f^*$
 by (simp add: LeftChopEqvChop)
 have 3: $\vdash ((f \wedge \text{fmore}) \vee (f \wedge \text{inf})); f^* = ((f \wedge \text{fmore}); f^* \vee (f \wedge \text{inf}); f^*)$
 by (simp add: OrChopEqv)
 have 4: $\vdash (f \wedge \text{inf}); f^* = (f \wedge \text{inf})$
 using AndInfChopEqvAndInf by blast
 have 5: $\vdash ((f \wedge \text{fmore}); f^* \vee (f \wedge \text{inf}); f^*) = ((f \wedge \text{fmore}); f^* \vee (f \wedge \text{inf}))$
 using 3 4 by auto
 from 2 3 5 show ?thesis by fastforce
 qed

lemma *PowerAndMoreAndFinite:*

$\vdash ((\text{power } (f \wedge \text{more}) \ n) \wedge \text{finite}) = (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) \ n)$

proof

(induct n)

case 0

then show ?case using FiniteAndEmptyEqvEmpty by auto

next

case (Suc n)

then show ?case

proof –

have 1: $\vdash (\text{power } (f \wedge \text{more}) \ (\text{Suc } n) \wedge \text{finite}) =$
 $((f \wedge \text{more}) \wedge \text{finite}) ; \text{power } (f \wedge \text{more}) \ n \wedge \text{finite}$

by simp

have 2: $\vdash (((f \wedge \text{more}) \wedge \text{finite}) ; \text{power } (f \wedge \text{more}) \ n \wedge \text{finite}) =$
 $((((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{finite}) ; (\text{power } (f \wedge \text{more}) \ n \wedge \text{finite}))$

by (simp add: ChopAndFiniteDist)

have 3: $\vdash (((f \wedge \text{more}) \wedge \text{finite}) \wedge \text{finite}) = (((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite})$

by auto

have 4: $\vdash (((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}); (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) \ n) =$
 $\text{power } ((f \wedge \text{finite}) \wedge \text{more}) \ (\text{Suc } n)$

by simp

show ?thesis

by (metis 1 2 3 4 Suc.hyps integ-reflection)

qed

qed

lemma *CSAndFiniteDist:*

$\vdash ((\exists n. \text{power } (f \wedge \text{more}) \ n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{finite}) =$
 $((\exists n. \text{power } (f \wedge \text{more}) \ n) \wedge \text{finite})$

proof –

have 1: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) \ n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{finite}) =$
 $((\exists n. \text{power } (f \wedge \text{more}) \ n) \wedge \text{finite});$
 $((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{finite})$

using ChopAndFiniteDist by blast

have 11: $\vdash (((f \wedge \text{more}) \wedge \text{inf}) \wedge \neg \text{inf}) = \#False$

by *fastforce*
have 2: $\vdash ((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{finite}) = \text{empty}$
 unfolding *finite-d-def* using 11 *NotEmptyAndInf* by *auto*
 from 1 2 **show** ?thesis
 by (*metis ChopEmpty inteq-reflection*)
qed

lemma *CSAndFinite*:

$\vdash (f^* \wedge \text{finite}) = (f \wedge \text{finite})^*$
proof –
have 1: $\vdash (f^* \wedge \text{finite}) = ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{finite})$
 by (*simp add: chopstar-d-def powerstar-d-def intI*)
have 11: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{finite}) =$
 $((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{finite})$
 using *CSAndFiniteDist* by *blast*
have 2: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{finite}) =$
 $(\exists n. \text{power } (f \wedge \text{more}) n \wedge \text{finite})$
 by (*simp add: Valid-def*)
have 3: $\vdash (\exists n. \text{power } (f \wedge \text{more}) n \wedge \text{finite}) =$
 $(\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n))$
 using *PowerAndMoreAndFinite* by *fastforce*
have 12: $\vdash (((f) \wedge \text{inf}) \wedge \neg \text{inf}) = \#False$
 by *fastforce*
have 31: $\vdash (\text{empty} \vee ((f \wedge \text{finite}) \wedge \text{inf})) = \text{empty}$
 unfolding *finite-d-def* using 12 *NotEmptyAndInf* by *auto*
have 4: $\vdash (\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n)) = (f \wedge \text{finite})^*$
 using 31
 by (*metis ChopEmpty AndMoreAndInfEqvAndInf chopstar-d-def inteq-reflection powerstar-d-def*)
have 5: $\vdash (\exists n. \text{power } (f \wedge \text{more}) n \wedge \text{finite}) = (\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n))$
 by (*metis 3*)
have 6: $\vdash (f^* \wedge \text{finite}) = (\exists n. (\text{power } (f \wedge \text{more}) n) \wedge \text{finite})$
 using 1 11 by *fastforce*
show ?thesis
 using 4 5 6 by *fastforce*
qed

lemma *PowerchopAndMore*:

$\vdash ((\text{power } ((f \wedge \text{finite}) \wedge \text{more}) (\text{Suc } n)) \wedge \text{more}) =$
 $(\text{power } ((f \wedge \text{finite}) \wedge \text{more}) (\text{Suc } n))$
proof (*induct n*)
case 0
then show ?case
proof –
have 01: $\vdash (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) (\text{Suc } 0) \wedge \text{more}) =$
 $((((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}); \text{empty} \wedge \text{more})$

 by *simp*
have 02: $\vdash (((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}); \text{empty} \wedge \text{more}) =$
 $((((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}) \wedge \text{more})$

by (metis ChopEmpty inteq-reflection)
 have 03: $\vdash (((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}) \wedge \text{more}) =$
 $((((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}))$
 by auto
 have 04: $\vdash (((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}) =$
 $((((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite})); \text{empty}$

 by (simp add: ChopEmpty int-iffD1 int-iffD2 int-iffI)
 have 05: $\vdash (((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}); \text{empty} =$
 $\text{power } ((f \wedge \text{finite}) \wedge \text{more}) (\text{Suc } 0)$
 by simp
 show ?thesis
 by (metis 03 04 05 inteq-reflection)
 qed
 next
 case (Suc n)
 then show ?case
 by (metis LeftChopImpMoreRule Prop10 Prop11 Prop12 lift-and-com pow-Suc)
 qed

lemma PowerchopAndFmore:
 $\vdash ((\text{power } (f \wedge \text{more}) (\text{Suc } n)) \wedge \text{fmore}) = (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) (\text{Suc } n))$
proof –
 have 1: $\vdash ((\text{power } (f \wedge \text{more}) (\text{Suc } n)) \wedge \text{fmore}) =$
 $((\text{power } (f \wedge \text{more}) (\text{Suc } n)) \wedge \text{finite}) \wedge \text{more}$
 by (auto simp add: fmore-d-def)
 have 2: $\vdash (((\text{power } (f \wedge \text{more}) (\text{Suc } n)) \wedge \text{finite}) \wedge \text{more}) =$
 $((\text{power } ((f \wedge \text{finite}) \wedge \text{more}) (\text{Suc } n)) \wedge \text{more})$
 by (metis PowerAndMoreAndFinite inteq-reflection lift-and-com)
 have 3: $\vdash ((\text{power } ((f \wedge \text{finite}) \wedge \text{more}) (\text{Suc } n)) \wedge \text{more}) =$
 $(\text{power } ((f \wedge \text{finite}) \wedge \text{more}) (\text{Suc } n))$
 using PowerchopAndMore by blast
 show ?thesis
 by (metis 1 2 3 inteq-reflection)
 qed

lemma ExistPowerAndMoreExpand:
 $\vdash (\exists n. \text{power } (f \wedge \text{more}) n) = (\text{empty} \vee (\exists n. (\text{power } (f \wedge \text{more}) (\text{Suc } n))))$
 using powersem1[of LIFT(f \wedge more)] by auto

lemma CSAndFmoreDist:
 $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{fmore}) =$
 $((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{fmore})$
proof –
 have 1: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{fmore}) =$
 $((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{finite} \wedge \text{more})$
 by (metis fmore-d-def inteq-reflection lift-and-com)
 have 2: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{finite}) =$

$((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{finite})$
using *CSAndFiniteDist* **by** *blast*
have 3: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{finite} \wedge \text{more}) =$
 $((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{finite} \wedge \text{more})$
using 2 **by** *fastforce*
have 4: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{finite} \wedge \text{more}) =$
 $((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{fmore})$
by (*metis fmore-d-def inteq-reflection lift-and-com*)
show *?thesis*
by (*metis 1 3 4 inteq-reflection*)
qed

lemma *CSAndMoreEqvAndFMoreChop*:

$\vdash (f^* \wedge \text{fmore}) = (f \wedge \text{fmore}); (f \wedge \text{finite})^*$

proof –

have 1: $\vdash (f^* \wedge \text{fmore}) = ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{fmore})$
by (*simp add: chopstar-d-def powerstar-d-def intI*)
have 11: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{fmore}) =$
 $((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{fmore})$
using *CSAndFmoreDist* **by** *fastforce*
have 2: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{fmore}) =$
 $(\exists n. \text{power } (f \wedge \text{more}) n \wedge \text{fmore})$
by (*simp add: Valid-def*)
have 3: $\vdash (\exists n. \text{power } (f \wedge \text{more}) n \wedge \text{fmore}) =$
 $((\text{power } (f \wedge \text{more}) 0 \vee (\exists n. (\text{power } (f \wedge \text{more}) (\text{Suc } n)))) \wedge \text{fmore})$
using *ExistPowerAndMoreExpand* **by** *fastforce*
have 4: $\vdash ((\text{power } (f \wedge \text{more}) 0 \vee (\exists n. (\text{power } (f \wedge \text{more}) (\text{Suc } n)))) \wedge \text{fmore}) =$
 $((\text{power } (f \wedge \text{more}) 0 \wedge \text{fmore}) \vee ((\exists n. (\text{power } (f \wedge \text{more}) (\text{Suc } n)) \wedge \text{fmore})))$
by *auto*
have 5: $\vdash (((\text{power } (f \wedge \text{more}) 0 \wedge \text{fmore}) \vee ((\exists n. (\text{power } (f \wedge \text{more}) (\text{Suc } n)) \wedge \text{fmore}))) =$
 $((\exists n. (\text{power } (f \wedge \text{more}) (\text{Suc } n)) \wedge \text{fmore}))$
using *NotFmoreAndEmpty* **by** *fastforce*
have 6: $\vdash ((\exists n. (\text{power } (f \wedge \text{more}) (\text{Suc } n)) \wedge \text{fmore}) =$
 $(\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) (\text{Suc } n)))$
using *PowerchopAndFmore* **by** *fastforce*
have 7: $\vdash (\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) (\text{Suc } n))) =$
 $(\exists n. (((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}); (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n)))$
by (*simp*)
have 8: $\vdash (\exists n. (((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}); (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n))) =$
 $((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}; (\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n))$
by (*auto simp add: Valid-def itl-defs*)
have 9: $\vdash (((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}) = (f \wedge \text{fmore})$
by (*auto simp add: fmore-d-def*)
have 10: $\vdash (((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}); (\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n)) =$
 $(f \wedge \text{fmore}); (\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n))$
using 8 9 **by** (*simp add: LeftChopEqvChop*)
have 1011: $\vdash ((f \wedge \text{finite}) \wedge \text{inf}) = \# \text{False}$
unfolding *finite-d-def* **by** *fastforce*
have 101: $\vdash (\text{empty} \vee ((f \wedge \text{finite}) \wedge \text{inf})) = \text{empty}$
unfolding *finite-d-def* **using** 1011 *NotEmptyAndInf* **by** *auto*

have 11: $\vdash (\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n)) =$
 $(f \wedge \text{finite})^*$
using 101
by (*metis ChopEmpty AndMoreAndInfEqvAndInf chopstar-d-def inteq-reflection powerstar-d-def*)
hence 12: $\vdash (f \wedge \text{fmore}); (\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n)) =$
 $(f \wedge \text{fmore}); (f \wedge \text{finite})^*$
by (*simp add: RightChopEqvChop*)
from 1 11 2 3 4 5 6 7 8 10 12 show ?thesis
by (*metis CSAndFmoreDist inteq-reflection*)
qed

lemma CSAndMoreImpChopCS:

$\vdash f^* \wedge \text{more} \longrightarrow f; f^*$

proof –

have 1: $\vdash (f^* \wedge \text{more}) = (f \wedge \text{more}); f^*$ **by** (*rule CSAndMoreEqvAndMoreChop*)

have 2: $\vdash (f \wedge \text{more}); f^* \longrightarrow f; f^*$ **by** (*rule AndChopA*)

from 1 2 show ?thesis by fastforce

qed

lemma NotAndMoreEqvEmptyOr:

$\vdash \neg (f \wedge \text{more}) = (\text{empty} \vee \neg f)$

by (*auto simp: empty-d-def*)

lemma MoreAndEmptyOrEqvMoreAnd:

$\vdash (\text{more} \wedge (\text{empty} \vee \neg f)) = (\text{more} \wedge \neg f)$

by (*auto simp: empty-d-def*)

lemma CSMoreNotImpChopCSAndMore:

$\vdash f^* \wedge \text{more} \wedge \neg f \longrightarrow (f \wedge \text{more}); (f^* \wedge \text{more})$

proof –

have 1: $\vdash (f^* \wedge \text{more}) = (f \wedge \text{more}); f^*$

by (*rule CSAndMoreEqvAndMoreChop*)

have 2: $\vdash \text{empty} \vee \text{more}$

by (*auto simp: empty-d-def*)

hence 3: $\vdash f^* \longrightarrow \text{empty} \vee (f^* \wedge \text{more})$

by auto

hence 4: $\vdash (f \wedge \text{more}); f^* \longrightarrow (f \wedge \text{more}) \vee ((f \wedge \text{more}); (f^* \wedge \text{more}))$

by (*rule ChopEmptyOrImpRule*)

hence 5: $\vdash (f \wedge \text{more}); f^* \wedge \neg (f \wedge \text{more}) \longrightarrow ((f \wedge \text{more}); (f^* \wedge \text{more}))$

by fastforce

have 6: $\vdash (f \wedge \text{more}); f^* = ((f \wedge \text{more}); f^* \wedge \text{more})$ **using 1**

by auto

have 7: $\vdash ((f \wedge \text{more}); f^* \wedge \neg (f \wedge \text{more})) = ((f \wedge \text{more}); f^* \wedge \text{more} \wedge \neg (f \wedge \text{more}))$

using 6 by auto

have 8: $\vdash (f \wedge \text{more}); f^* \wedge \text{more} \wedge \neg f \longrightarrow (f \wedge \text{more}); (f^* \wedge \text{more})$

using 5 7 by auto

have 9: $\vdash (f^* \wedge \text{more} \wedge \neg f) = ((f^* \wedge \text{more}) \wedge (\text{more} \wedge \neg f))$

by auto

have 10: $\vdash ((f^* \wedge \text{more}) \wedge (\text{more} \wedge \neg f)) = ((f \wedge \text{more}); f^* \wedge (\text{more} \wedge \neg f))$

using 1 by fastforce
 from 1 8 9 10 show ?thesis by fastforce
 qed

lemma *ChopplusCommuteImpA*:

$\vdash f^*;f \longrightarrow f;f^*$

by (metis CSeqvOrChopCS ChopAndB ChopEmpty ChopstarInductL EmptyImpCS Prop02 Prop03 Prop10
 inteq-reflection)

lemma *ChopplusCommuteImpB*:

$\vdash f;f^* \longrightarrow f^*;f$

proof –

have f2: $\vdash (f^*;f);f \longrightarrow f^*;f$

by (metis CSeqvOrChopCS ChopplusCommuteImpA LeftChopImpChop Prop05 inteq-reflection)

have $\vdash f \longrightarrow f^*;f$

by (metis AndChopB EmptyChop EmptyImpCS Prop10 inteq-reflection)

then show ?thesis

using f2 ChopstarInductR Prop02 by blast

qed

lemma *ChopplusCommute*:

$\vdash f;f^* = f^*;f$

using ChopplusCommuteImpA ChopplusCommuteImpB by fastforce

lemma *CSeqvOrChopCSB*:

$\vdash f^* = (\text{empty} \vee (f^*;f))$

by (meson CSeqvOrChopCS ChopplusCommute Prop06)

lemma *CSAndMoreImpCSChop*:

$\vdash f^* \wedge \text{more} \longrightarrow f^*;f$

using CSAndMoreEqvAndMoreChop ChopplusCommute CSAndMoreImpChopCS by fastforce

lemma *PowerChopPower*:

$\vdash (\text{power } (f \wedge \text{more}) \ n); (\text{power } (f \wedge \text{more}) \ k) = (\text{power } (f \wedge \text{more}) \ (n+k))$

proof

(induct n arbitrary: k)

case 0

then show ?case using EmptyChopSem by auto

next

case (Suc n)

then show ?case

by (metis (no-types, lifting) ChopAssoc add-Suc inteq-reflection pow-Suc)

qed

lemma *CSChopCS*:

$\vdash f^* ; f^* = f^*$

proof –

have 1: $\vdash f^* ; f^* \longrightarrow f^*$

by (*meson CSeqvOrChopCSB ChopstarImpPowerstar ChopstarInductR PowerstarImpChopstar Prop02 Prop03*
lift-imp-trans)
have 2: $\vdash f^* \longrightarrow f^* ; f^*$
by (*metis ChopEmpty EmptyImpCS RightChopImpChop inteq-reflection*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *NotEmptyEqvMore*:
 $\vdash (\neg \text{empty}) = \text{more}$
by (*simp add: empty-d-def*)

lemma *NotCSImpMore*:
 $\vdash \neg (f^*) \longrightarrow \text{more}$
proof –
have 1: $\vdash \text{empty} \longrightarrow (f^*)$ **using** *EmptyImpCS* **by** *blast*
hence 2: $\vdash \neg \text{empty} \vee (f^*)$ **by** *fastforce*
from 2 **show** *?thesis* **using** 1 *NotEmptyEqvMore* **by** *fastforce*
qed

lemma *PowerAndInfB*:
 $\vdash ((f \wedge \text{more}); (\text{power } (f \wedge \text{more}) \ n)) \wedge \text{inf} =$
 $((f \wedge \text{inf}) \vee (f \wedge \text{fmore}); ((\text{power } (f \wedge \text{more}) \ n) \wedge \text{inf}))$
proof –
have 1: $\vdash ((f \wedge \text{more}); (\text{power } (f \wedge \text{more}) \ n)) \wedge \text{inf} =$
 $((f \wedge \text{more}) \wedge \text{inf}) \vee ((f \wedge \text{more}) \wedge \text{finite}); ((\text{power } (f \wedge \text{more}) \ n) \wedge \text{inf}))$
using *ChopAndInfB* **by** *blast*
have 2: $\vdash ((f \wedge \text{more}) \wedge \text{inf}) \vee ((f \wedge \text{more}) \wedge \text{finite}); ((\text{power } (f \wedge \text{more}) \ n) \wedge \text{inf}) =$
 $((f \wedge \text{inf}) \vee (f \wedge \text{fmore}); ((\text{power } (f \wedge \text{more}) \ n) \wedge \text{inf}))$
using *AndMoreAndInfEqvAndInf AndMoreAndFiniteEqvAndFmore*
by (*metis 1 inteq-reflection*)
show *?thesis*
by (*metis 1 2 int-eq*)
qed

lemma *CSAndInf*:
 $\vdash (f^* \wedge \text{inf}) = f^*; (f \wedge \text{inf})$
by (*meson AndChopA AndInfChopEqvAndInf CSeqvOrChopCSB ChopAndA ChopAndInf Prop03 Prop11 Prop12 lift-imp-trans*)

lemma *CSChopCSImpCS*:
 $\vdash (f^*; f^*) \longrightarrow f^*$
by (*simp add: CSChopCS int-iffD1*)

lemma *ImpChopPlus*:
 $\vdash f \longrightarrow f; f^*$
proof –
have 1: $\vdash f^* = (\text{empty} \vee f; f^*)$ **by** (*rule CSeqvOrChopCS*)
hence 2: $\vdash f; f^* = (f; \text{empty} \vee f; (f; f^*))$ **using** *ChopOrEqvRule* **by** *blast*

have 3: $\vdash f; \text{empty} = f$ **using** *ChopEmpty* **by** *blast*
 from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *ImpCS*:
 $\vdash f \longrightarrow f^*$
proof –
 have 1: $\vdash f \longrightarrow f; f^*$ **by** (*rule ImpChopPlus*)
 hence 2: $\vdash f \longrightarrow \text{empty} \vee f; f^*$ **by** *auto*
 from 2 **show** *?thesis* **using** *CSEqvOrChopCS* **by** *fastforce*
qed

lemma *CSChopImpCS*:
 $\vdash f^*; f \longrightarrow f^*$
proof –
 have 1: $\vdash f \longrightarrow f^*$ **by** (*rule ImpCS*)
 hence 2: $\vdash f^*; f \longrightarrow f^*; f^*$ **by** (*rule RightChopImpChop*)
 hence 3: $\vdash f^*; f \longrightarrow f^*; f^*$ **by** *auto*
 have 4: $\vdash f^*; f^* \longrightarrow f^*$ **by** (*rule CSChopCSImpCS*)
 from 2 3 4 **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *ChopPlusImpCS*:
 $\vdash f; f^* \longrightarrow f^*$
proof –
 have 1: $\vdash f; f^* \longrightarrow \text{empty} \vee f; f^*$ **by** *auto*
 from 1 **show** *?thesis* **using** *CSEqvOrChopCS* **by** *fastforce*
qed

lemma *CSChopEqvOrChopPlusChop*:
 $\vdash f^*; g = (g \vee (f; f^*) ; g)$
proof –
 have 1: $\vdash f^* = (\text{empty} \vee f; f^*)$ **by** (*rule CSEqvOrChopCS*)
 from 1 **show** *?thesis* **using** *EmptyOrChopEqvRule* **by** *blast*
qed

lemma *CSElim*:
 assumes $\vdash \text{empty} \longrightarrow g$
 $\vdash (f \wedge \text{more}) ; g \longrightarrow g$
 shows $\vdash f^* \longrightarrow g$
proof –
 have 1: $\vdash \text{empty} \vee (f \wedge \text{more}) ; g \longrightarrow g$
 using *assms* **using** *Prop02* **by** *blast*
 have 2: $\vdash (\text{chopstar } f) ; \text{empty} \longrightarrow g$
 using *ChopstarInductMoreL 1* **by** *blast*
 from 2 **show** *?thesis*
 by (*metis ChopEmpty inteq-reflection*)
qed

lemma *ChopstarImp*:

assumes $\vdash f;(\text{chopstar } g) \vee \text{empty} \longrightarrow (\text{chopstar } g)$
shows $\vdash (\text{chopstar } f) \longrightarrow (\text{chopstar } g)$
using *assms ChopstarInductL ChopEmpty*
by (*metis int-eq int-simps(33) lift-and-com*)

lemma *CSCSImpCS*:

$\vdash (f^*)^* \longrightarrow f^*$

proof –

have 1: $\vdash ((\text{chopstar } f);(\text{chopstar } f)) \vee \text{empty} \longrightarrow (\text{chopstar } f)$

by (*meson CSChopCSImpCS EmptyImpCS Prop02*)

from 1 **show** *?thesis* **using** *ChopstarImp* **by** *blast*

qed

lemma *CSImpCSCS*:

$\vdash f^* \longrightarrow (f^*)^*$

using *ImpCS* **by** *auto*

lemma *CSCSEqvCS*:

$\vdash (f^*)^* = f^*$

by (*simp add: CSCSImpCS CSImpCSCS int-iffI*)

lemma *RightEmptyOrChopEqv*:

$\vdash g;(\text{empty} \vee f) = (g \vee (g;f))$

proof –

have 1: $\vdash g;(\text{empty} \vee f) = (g;\text{empty} \vee g;f)$ **by** (*rule ChopOrEqv*)

have 2: $\vdash g;\text{empty} = g$ **by** (*rule ChopEmpty*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *RightEmptyOrChopEqvRule*:

assumes $\vdash f = (\text{empty} \vee f1)$

shows $\vdash g;f = (g \vee (g;f1))$

proof –

have 1: $\vdash f = (\text{empty} \vee f1)$ **using** *assms* **by** *auto*

hence 2: $\vdash g;f = g;(\text{empty} \vee f1)$ **by** (*rule RightChopEqvChop*)

have 3: $\vdash g;(\text{empty} \vee f1) = (g \vee (g;f1))$ **by** (*rule RightEmptyOrChopEqv*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *ChopPlusEqvOrChopChopPlus*:

$\vdash (f;f^*) = (f \vee f; (f;f^*))$

proof –

have 1: $\vdash f^* = (\text{empty} \vee f;f^*)$ **by** (*rule CSEqvOrChopCS*)

from 1 **show** *?thesis* **by** (*rule RightEmptyOrChopEqvRule*)

qed

lemma *CSAndEmptyEqvEmpty*:

$\vdash ((f^*) \wedge \text{empty}) = \text{empty}$

using *EmptyImpCS* **by** *fastforce*

lemma *NotAndMoreChopAndEmpty*:

$\vdash \neg((f \wedge \text{more});g) \wedge \text{empty}$

by (*metis AndChopA ChopEmpty LeftChopImpMoreRule Prop01 empty-d-def int-simps(14)*
int-simps(25) int-simps(4) inteq-reflection lift-and-com)

lemma *NotChopAndMoreAndEmpty*:

$\vdash \neg(f;(g \wedge \text{more})) \wedge \text{empty}$

by (*metis NotEmptyEqvMore Prop01 Prop05 Prop07 RightChopImpMoreRule empty-d-def int-iffD2*
int-simps(15) inteq-reflection lift-imp-neg)

lemma *ChopCSAndEmptyEqvAndEmpty*:

$\vdash ((f;f^*) \wedge \text{empty}) = (f \wedge \text{empty})$

proof –

have 1: $\vdash ((f;f^*) \wedge \text{empty}) = (f \wedge \text{empty});(f^* \wedge \text{empty})$

using *ChopAndEmptyEqvEmptyChopEmpty* **by** *blast*

have 2: $\vdash (f \wedge \text{empty});(f^* \wedge \text{empty}) = (f \wedge \text{empty});\text{empty}$

using *CSAndEmptyEqvEmpty* **using** *RightChopEqvChop* **by** *blast*

have 3: $\vdash (f \wedge \text{empty});\text{empty} = (f \wedge \text{empty})$

by (*rule ChopEmpty*)

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

qed

lemma *AndMoreChopAndMoreEqvAndMoreChop*:

$\vdash ((f \wedge \text{more});g \wedge \text{more}) = (f \wedge \text{more});g$

using *ChopImpDi DiAndB DiMoreEqvMore* **by** *fastforce*

lemma *ChopPlusEqv*:

$\vdash (f;f^*) = (f \vee (f \wedge \text{more}); (f;f^*))$

proof –

have 1: $\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$

by (*rule ChopstarEqv*)

have 2: $\vdash f^* = (\text{empty} \vee f;f^*)$

by (*rule CSEqvOrChopCS*)

hence 3: $\vdash (\text{empty} \vee f;f^*) = (\text{empty} \vee (f \wedge \text{more});f^*)$

using 1 2 **by** *fastforce*

have 4: $\vdash (f \wedge \text{more});(f^*) = (f \wedge \text{more});(\text{empty} \vee f;f^*)$

using 2 **using** *RightChopEqvChop* **by** *blast*

hence 5: $\vdash \text{empty} \vee f;f^* = \text{empty} \vee (f \wedge \text{more});(\text{empty} \vee f;f^*)$

using 3 4 **by** *fastforce*

have 6: $\vdash (f \wedge \text{more}); (\text{empty} \vee f;f^*) =$

$((f \wedge \text{more}); \text{empty} \vee (f \wedge \text{more}); (f;f^*))$

using *ChopOrEqv* **by** *blast*

have 7: $\vdash (f \wedge \text{more}); \text{empty} = (f \wedge \text{more})$

using *ChopEmpty* **by** *blast*

have 8: $\vdash (\text{empty} \vee f;f^*) =$

$(\text{empty} \vee (f \wedge \text{more}) \vee (f \wedge \text{more}); (f;f^*))$

using 5 6 7 **by** (*metis 2 3 inteq-reflection*)

have 9: $\vdash ((\text{empty} \vee f;f^*) \wedge \text{more}) = (f;f^* \wedge \text{more})$

by (*auto simp: empty-d-def*)

have 10: $\vdash ((\text{empty} \vee (f \wedge \text{more}) \vee (f \wedge \text{more}); (f;f^*)) \wedge \text{more}) =$

$((f \wedge \text{more}) \vee (f \wedge \text{more}); (f; f^*)) \wedge \text{more}$
by (*auto simp: empty-d-def*)
have 11: $\vdash ((f \wedge \text{more}) \vee (f \wedge \text{more}); (f; f^*)) \wedge \text{more} =$
 $((f \wedge \text{more}) \vee (f \wedge \text{more}); (f; f^*))$
using 10 6 7 *int-eq*
using *AndMoreChopAndMoreEqvAndMoreChop* **by** *fastforce*
have 12: $\vdash (f; f^* \wedge \text{more}) = ((f \wedge \text{more}) \vee (f \wedge \text{more}); (f; f^*))$
by (*metis 10 11 8 9 inteq-reflection*)
have 13: $\vdash (f; f^* \wedge \text{empty}) = (f \wedge \text{empty})$
by (*rule ChopCSAndEmptyEqvAndEmpty*)
have 14: $\vdash ((f \wedge \text{more}) \vee (f \wedge \text{more}); (f; f^*) \vee (f \wedge \text{empty})) =$
 $(f \vee (f \wedge \text{more}); (f; f^*))$
by (*auto simp: empty-d-def*)
have 15: $\vdash f; f^* = ((f; f^* \wedge \text{empty}) \vee (f; f^* \wedge \text{more}))$
by (*auto simp: empty-d-def*)
from 12 13 14 15 show ?thesis by fastforce
qed

lemma *ChopPlusImpChopPlus:*

assumes $\vdash f \longrightarrow g$
shows $\vdash f; f^* \longrightarrow g; g^*$
using *assms*
by (*metis AndChopB CSChopCS ChopImpChop ChopstarImp EmptyImpCS ImpCS Prop01 Prop02*
Prop05 Prop10 inteq-reflection)

lemma *ChopChopPlusImpChopPlus:*

$\vdash f; (f; f^*) \longrightarrow f; f^*$
proof –
have 1: $\vdash \text{empty} \vee \text{more}$ **by** (*auto simp: empty-d-def*)
hence 2: $\vdash f \longrightarrow \text{empty} \vee (f \wedge \text{more})$ **by** *auto*
hence 3: $\vdash f; (f; f^*) \longrightarrow (f; f^*) \vee (f \wedge \text{more}); (f; f^*)$ **by** (*rule EmptyOrChopImpRule*)
have 4: $\vdash f; f^* = (f \vee (f \wedge \text{more}); (f; f^*))$ **by** (*rule ChopPlusEqv*)
hence 5: $\vdash (f \wedge \text{more}); (f; f^*) \longrightarrow f; f^*$ **by** *auto*
from 3 5 show ?thesis using ChopPlusImpCS RightChopImpChop by blast
qed

lemma *CSImpCS:*

assumes $\vdash f \longrightarrow g$
shows $\vdash f^* \longrightarrow g^*$
proof –
have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence 2: $\vdash f; f^* \longrightarrow g; g^*$ **by** (*rule ChopPlusImpChopPlus*)
hence 3: $\vdash \text{empty} \vee f; f^* \longrightarrow \text{empty} \vee g; g^*$ **by** *auto*
from 2 3 show ?thesis using CSEqvOrChopCS by (metis inteq-reflection)
qed

lemma *ChopPlusIntro:*

assumes $\vdash f \longrightarrow g \vee (g \wedge \text{more}); f$
shows $\vdash f \wedge \text{finite} \longrightarrow g; g^*$
proof –

have 1: $\vdash f \wedge \neg g \longrightarrow (g \wedge \text{more}); f$ **using** *assms* **by** *auto*
have 2: $\vdash g; g^* = (g \vee (g \wedge \text{more}); (g; g^*))$ **by** (*rule ChopPlusEqv*)
have 3: $\vdash f \wedge \neg (g; g^*) \longrightarrow$
 $(g \wedge \text{more}); f \wedge \neg ((g \wedge \text{more}); (g; g^*))$ **using** 1 2 **by** *fastforce*
have 4: $\vdash g \wedge \text{more} \longrightarrow \text{more}$ **by** *auto*
from 3 4 **show** *?thesis* **using** *ChopContraB* **by** *blast*
qed

lemma *ChopPlusElim*:
assumes $\vdash f \longrightarrow g$
 $\vdash (f \wedge \text{more}); g \longrightarrow g$
shows $\vdash f; f^* \longrightarrow g$
proof –
have 1: $\vdash f \vee (f \wedge \text{more}); g \longrightarrow g$
using *assms Prop02* **by** *blast*
have 2: $\vdash f^*; f \longrightarrow g$
using *ChopstarInductMoreL 1* **by** *blast*
from 2 **show** *?thesis*
using *ChopplusCommute* **by** *fastforce*
qed

lemma *ChopPlusElimWithoutMore*:
assumes $\vdash f \longrightarrow g$
 $\vdash f; g \longrightarrow g$
shows $\vdash f; f^* \longrightarrow g$
proof –
have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *blast*
have 2: $\vdash (f; g) \longrightarrow g$ **using** *assms* **by** *blast*
have 3: $\vdash (f \wedge \text{more}); g \longrightarrow f; g$ **by** (*rule AndChopA*)
have 4: $\vdash (f \wedge \text{more}); g \longrightarrow g$ **using** 2 3 *lift-imp-trans* **by** *blast*
from 1 4 **show** *?thesis* **using** *ChopPlusElim* **by** *blast*
qed

lemma *ChopPlusEqvChopPlus*:
assumes $\vdash f = g$
shows $\vdash f; f^* = g; g^*$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash f \longrightarrow g$ **by** *auto*
hence 3: $\vdash f; f^* \longrightarrow g; g^*$ **by** (*rule ChopPlusImpChopPlus*)
have 4: $\vdash g \longrightarrow f$ **using** 1 **by** *auto*
hence 5: $\vdash g; g^* \longrightarrow f; f^*$ **by** (*rule ChopPlusImpChopPlus*)
from 3 5 **show** *?thesis* **by** *fastforce*
qed

lemma *CSEqvCS*:
assumes $\vdash f = g$
shows $\vdash f^* = g^*$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*

hence 2: $\vdash f;f^* = g;g^*$ **by** (rule *ChopPlusEqvChopPlus*)
 hence 3: $\vdash (\text{empty} \vee f;f^*) = (\text{empty} \vee g;g^*)$ **by** *auto*
 from 3 **show** *?thesis* **using** *CSEqvOrChopCS* **by** (metis *int-eq*)
qed

lemma *AndCSA*:

$\vdash (f \wedge g)^* \longrightarrow f^*$

proof –

have 1: $\vdash f \wedge g \longrightarrow f$ **by** *auto*

from 1 **show** *?thesis* **using** *CSImpCS* **by** *blast*

qed

lemma *AndCSB*:

$\vdash (f \wedge g)^* \longrightarrow g^*$

proof –

have 1: $\vdash f \wedge g \longrightarrow g$ **by** *auto*

from 1 **show** *?thesis* **using** *CSImpCS* **by** *blast*

qed

lemma *CSIntro*:

assumes $\vdash f \wedge \text{more} \longrightarrow (g \wedge \text{more}); f$

shows $\vdash f \wedge \text{finite} \longrightarrow g^*$

proof –

have 1: $\vdash f \wedge \text{more} \longrightarrow (g \wedge \text{more}); f$

using *assms* **by** *auto*

have 2: $\vdash \text{more} = (\neg \text{empty})$

by (auto *simp: empty-d-def*)

have 3: $\vdash f \wedge \neg \text{empty} \longrightarrow (g \wedge \text{more}); f$

using 1 2 **by** *fastforce*

have 4: $\vdash g^* = (\text{empty} \vee (g \wedge \text{more}); g^*)$

by (rule *ChopstarEqv*)

hence 41: $\vdash (\neg(\text{empty} \vee (g \wedge \text{more}); g^*)) = (\neg \text{empty} \wedge \neg((g \wedge \text{more}); g^*))$

by *fastforce*

have 411: $\vdash (\neg \text{empty} \wedge \neg((g \wedge \text{more}); g^*)) = (\text{more} \wedge \neg((g \wedge \text{more}); g^*))$

using *NotEmptyEqvMore* **by** *fastforce*

have 42: $\vdash \neg(g^*) = (\text{more} \wedge \neg((g \wedge \text{more}); g^*))$

using 4 41 411 **by** *fastforce*

have 43: $\vdash f \wedge \neg(g^*) \longrightarrow f \wedge \text{more} \wedge \neg((g \wedge \text{more}); g^*)$

using 42 **by** *fastforce*

have 44: $\vdash f \wedge \text{more} \wedge \neg((g \wedge \text{more}); g^*) \longrightarrow (g \wedge \text{more}); f \wedge \neg((g \wedge \text{more}); g^*)$

using 3 43 1 **by** *auto*

have 5: $\vdash f \wedge \neg(g^*) \longrightarrow$

$(g \wedge \text{more}); f \wedge \neg((g \wedge \text{more}); g^*)$

using 43 44 *lift-imp-trans* **by** *fastforce*

have 6: $\vdash g \wedge \text{more} \longrightarrow \text{more}$

by *auto*

from 5 6 **show** *?thesis* **using** *ChopContraB* **by** *blast*

qed

lemma *CSElimWithoutMore*:

assumes $\vdash \text{empty} \longrightarrow g$
 $\vdash f; g \longrightarrow g$
shows $\vdash f^* \longrightarrow g$
proof –
have 1: $\vdash \text{empty} \longrightarrow g$ **using** *assms* **by** *blast*
have 2: $\vdash f; g \longrightarrow g$ **using** *assms* **by** *blast*
have 3: $\vdash (f \wedge \text{more}); g \longrightarrow f; g$ **by** (*rule AndChopA*)
have 4: $\vdash (f \wedge \text{more}); g \longrightarrow g$ **using** 2 3 *lift-imp-trans* **by** *blast*
from 1 4 **show** *?thesis* **using** *CSElim* **by** *blast*
qed

lemma *ChopAssocB*:
 $\vdash (f;g);h = f;(g;h)$
using *ChopAssoc* **by** *fastforce*

lemma *CSChopEqvChopOrRule*:
assumes $\vdash f = (g^*; h)$
shows $\vdash f = ((g; f) \vee h)$
proof –
have 1: $\vdash f = (g^*; h)$ **using** *assms* **by** *auto*
have 2: $\vdash g^* = (\text{empty} \vee (g; g^*))$ **by** (*rule CSEqvOrChopCS*)
hence 3: $\vdash g^*; h = (h \vee ((g; g^*); h))$ **by** (*rule EmptyOrChopEqvRule*)
have 4: $\vdash (g; g^*); h = g; (g^*; h)$ **by** (*rule ChopAssocB*)
hence 41: $\vdash g^*; h = (h \vee g; (g^*; h))$ **using** 3 **by** *fastforce*
have 5: $\vdash g; f = g; (g^*; h)$ **using** 1 **by** (*rule RightChopEqvChop*)
hence 6: $\vdash (g^*; h) = (h \vee g; f)$ **using** 41 **by** *fastforce*
hence 61: $\vdash (g^*; h) = ((g; f) \vee h)$ **by** *auto*
from 1 61 **show** *?thesis* **by** *fastforce*
qed

lemma *CSChopIntroRule*:
assumes $\vdash f \wedge \neg h \longrightarrow g; f$
 $\vdash g \longrightarrow \text{more}$
shows $\vdash f \wedge \text{finite} \longrightarrow g^*; h$
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: $\vdash g; f \longrightarrow (g \wedge \text{more}); f$
by (*rule LeftChopImpChop*)
have 5: $\vdash f \longrightarrow (g \wedge \text{more}); f \vee h$
using 1 4 **by** *fastforce*
have 6: $\vdash g^* = (\text{empty} \vee (g \wedge \text{more}); g^*)$
by (*rule ChopstarEqv*)
hence 7: $\vdash (g^*); h = (h \vee ((g \wedge \text{more}); g^*); h)$
by (*rule EmptyOrChopEqvRule*)
have 8: $\vdash ((g \wedge \text{more}); g^*); h = (g \wedge \text{more}); (g^*; h)$

by (rule ChopAssocB)
 have 9: $\vdash (g^*); h = (h \vee (g \wedge \text{more}); (g^*; h))$
 using 7 8 by fastforce
 have 10: $\vdash f \wedge \neg (g^*; h) \longrightarrow (g \wedge \text{more}); f \wedge \neg ((g \wedge \text{more}); (g^*; h))$
 using 5 9 by fastforce
 have 11: $\vdash g \wedge \text{more} \longrightarrow \text{more}$
 by fastforce
 from 10 11 show ?thesis using ChopContraB by blast
 qed

lemma *DiamondAndEmptyEqvAndEmpty*:

$\vdash (\Diamond f \wedge \text{empty}) = (f \wedge \text{empty})$

by (metis ChopAndEmptyEqvEmptyChopEmpty EmptyChop FiniteAndEmptyEqvEmpty int-eq sometimes-d-def)

lemma *InitAndEmptyEqvAndEmpty*:

$\vdash ((\text{init } w) \wedge \text{empty}) = (w \wedge \text{empty})$

proof –

have 1: $\vdash ((\text{init } w) \wedge \text{empty}) = ((w \wedge \text{empty}); \# \text{True} \wedge \text{empty})$

by (metis init-d-def int-eq lift-and-com)

have 2: $\vdash ((w \wedge \text{empty}); \# \text{True} \wedge \text{empty}) = (w \wedge \text{empty}); (\# \text{True} \wedge \text{empty})$

by (meson AndChopA ChopAndA ChopAndEmptyEqvEmptyChopEmpty lift-imp-trans Prop11 Prop12)

have 3: $\vdash (w \wedge \text{empty}); (\# \text{True} \wedge \text{empty}) = (w \wedge \text{empty}); \text{empty}$

using RightChopEqvChop by fastforce

have 4: $\vdash (w \wedge \text{empty}); \text{empty} = (w \wedge \text{empty})$

using ChopEmpty by blast

from 1 2 3 4 show ?thesis by fastforce

qed

lemma *InitAndNotBoxInitImpNotEmpty*:

$\vdash \text{init } w \wedge \neg (\Box (\text{init } w)) \longrightarrow \neg \text{empty}$

proof –

have 1: $\vdash ((\text{init } w) \wedge \text{empty}) = (w \wedge \text{empty})$

by (rule InitAndEmptyEqvAndEmpty)

have 2: $\vdash (\neg (\Box (\text{init } w)) \wedge \text{empty}) = (\Diamond (\neg (\text{init } w)) \wedge \text{empty})$

by (auto simp: always-d-def)

have 3: $\vdash (\Diamond (\neg (\text{init } w)) \wedge \text{empty}) = (\neg (\text{init } w) \wedge \text{empty})$

by (simp add: DiamondAndEmptyEqvAndEmpty)

have 4: $\vdash (\neg (\text{init } w)) = (\text{init } (\neg w))$

using Initprop(2) by blast

have 5: $\vdash (\neg (\text{init } w) \wedge \text{empty}) = (\neg w \wedge \text{empty})$

using 4 InitAndEmptyEqvAndEmpty by (metis inteq-reflection)

have 6: $\vdash (\neg (\Box (\text{init } w)) \wedge \text{empty}) = (\neg w \wedge \text{empty})$

using 2 3 5 by fastforce

have 7: $\vdash \neg (\text{init } w \wedge \neg (\Box (\text{init } w)) \wedge \text{empty})$

using 1 6 by fastforce

from 7 show ?thesis by auto

qed

lemma *BoxImpTrueChopAndEmpty*:

$\vdash \Box f \longrightarrow \#True; (f \wedge \text{empty})$
using *BoxAndChopImport Finprop(3)* **by** *fastforce*

lemma *BoxInitAndMoreImpBoxInitAndMoreAndFinInit*:

$\vdash \Box (\text{init } w) \wedge \text{more} \longrightarrow (\Box (\text{init } w) \wedge \text{more}) \wedge \text{fin } (\text{init } w)$

proof –

have 1: $\vdash \text{fin } (\text{init } w) = \#True ; (\text{init } w \wedge \text{empty})$ **using** *FinEqvTrueChopAndEmpty* **by** *blast*

have 2: $\vdash \Box (\text{init } w) \longrightarrow \#True; (\text{init } w \wedge \text{empty})$ **by** (rule *BoxImpTrueChopAndEmpty*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *CSImpBox*:

assumes $\vdash f \longrightarrow \text{empty} \vee ((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}); f$

shows $\vdash (\text{init } w \wedge f) \wedge \text{finite} \longrightarrow \Box (\text{init } w)$

proof –

have 1: $\vdash f \longrightarrow \text{empty} \vee ((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}); f$

using *assms* **by** *auto*

have 2: $\vdash \text{init } w \wedge \neg (\Box (\text{init } w)) \longrightarrow \neg \text{empty}$

by (rule *InitAndNotBoxInitImpNotEmpty*)

have 3: $\vdash \text{init } w \wedge f \wedge \neg (\Box (\text{init } w)) \longrightarrow ((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}); f$

using 1 2 **by** *fastforce*

have 4: $\vdash \Box (\text{init } w) \wedge \text{more} \longrightarrow (\Box (\text{init } w) \wedge \text{more}) \wedge \text{fin } (\text{init } w)$

by (rule *BoxInitAndMoreImpBoxInitAndMoreAndFinInit*)

have 41: $\vdash (\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite} \longrightarrow$

$((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}) \wedge \text{fin } (\text{init } w)$

using 4 **by** *auto*

hence 5: $\vdash ((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}); f \longrightarrow$

$((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); f$

by (rule *LeftChopImpChop*)

have 6: $\vdash (((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); f =$

$((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}); (\text{init } w \wedge f)$

using *AndFinChopEqvStateAndChop* **by** *blast*

have 7: $\vdash \neg (\Box (\text{init } w)) \longrightarrow (\Box (\text{init } w)) \text{ yields } (\neg (\Box (\text{init } w)))$

by (rule *NotBoxStateImpBoxYieldsNotBox*)

have 8: $\vdash (\Box (\text{init } w)) \text{ yields } (\neg (\Box (\text{init } w))) \longrightarrow$

$((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}) \text{ yields } (\neg (\Box (\text{init } w)))$

using *AndYieldsA*

by (metis *AndMoreAndFiniteEqvAndFmore inteq-reflection*)

have 9: $\vdash ((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}); (\text{init } w \wedge f) \wedge$

$((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}) \text{ yields } (\neg (\Box (\text{init } w)))$

\longrightarrow

$((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}); ((\text{init } w \wedge f) \wedge \neg (\Box (\text{init } w)))$

by (rule *ChopAndYieldsImp*)

have 10: $\vdash (\text{init } w \wedge f) \wedge \neg (\Box (\text{init } w)) \longrightarrow$

$((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}); ((\text{init } w \wedge f) \wedge \neg (\Box (\text{init } w)))$

using 3 5 6 7 8 9 **by** *fastforce*

have 11: $\vdash ((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}); ((\text{init } w \wedge f) \wedge \neg (\Box (\text{init } w))) \longrightarrow$

$\text{more} ; ((\text{init } w \wedge f) \wedge \neg (\Box (\text{init } w)))$

by (metis 41 *LeftChopImpChop Prop12*)

have 12: $\vdash (\text{init } w \wedge f) \wedge \neg (\Box (\text{init } w)) \longrightarrow$

$more ; ((init\ w \wedge f) \wedge \neg(\Box (init\ w)))$
using 10 11 **by** fastforce
from 12 **show** ?thesis **using** MoreChopContraFiniteB **by** blast
qed

lemma BoxCSeqvBox:

$\vdash (init\ w \wedge (\Box (init\ w))^*) = \Box (init\ w)$
proof –
have 1: $\vdash \Box (init\ w); \Box (init\ w) \longrightarrow \Box (init\ w)$
by (simp add: BoxStateChopBoxEqvBox int-iffD1)
have 2: $\vdash (init\ w \wedge empty) \longrightarrow \Box (init\ w)$
by (simp add: StateAndEmptyImpBoxState)
have 3: $\vdash (init\ w \wedge empty) \vee \Box (init\ w); \Box (init\ w) \longrightarrow \Box (init\ w)$
using 1 2 **by** fastforce
have 4: $\vdash (init\ w \wedge empty); (\Box (init\ w))^* \longrightarrow \Box (init\ w)$
using ChopstarInductR 3 **by** blast
have 5: $\vdash init\ w \wedge (\Box (init\ w))^* \longrightarrow \Box (init\ w)$
using 4 StateAndEmptyChop **by** fastforce
have 11: $\vdash \Box (init\ w) \longrightarrow (init\ w)$
using BoxElim **by** blast
have 12: $\vdash \Box (init\ w) \longrightarrow (\Box (init\ w))^*$
by (rule ImpCS)
have 13: $\vdash \Box (init\ w) \longrightarrow init\ w \wedge (\Box (init\ w))^*$
using 11 12 **by** fastforce
from 5 13 **show** ?thesis **by** fastforce
qed

lemma BoxStateAndCSeqvCS:

$\vdash (\Box (init\ w) \wedge f^* \wedge finite) = (init\ w \wedge (\Box (init\ w) \wedge f)^* \wedge finite)$
proof –
have 1: $\vdash \Box (init\ w) \longrightarrow init\ w$
using BoxElim **by** blast
have 2: $\vdash (f^* \wedge more) = (f \wedge more); f^*$
by (rule CSAndMoreEqvAndMoreChop)
have 3: $\vdash (\Box (init\ w) \wedge ((f \wedge more); f^*)) =$
 $((\Box (init\ w) \wedge f \wedge more); (\Box (init\ w) \wedge f^*))$
by (rule BoxStateAndChopEqvChop)
have 4: $\vdash \Box (init\ w) \wedge f \wedge more \longrightarrow (\Box (init\ w) \wedge f) \wedge more$
by auto
hence 5: $\vdash (\Box (init\ w) \wedge f \wedge more); (\Box (init\ w) \wedge f^*) \longrightarrow$
 $((\Box (init\ w) \wedge f) \wedge more); (\Box (init\ w) \wedge f^*)$
by (rule LeftChopImpChop)
have 6: $\vdash (\Box (init\ w) \wedge f^*) \wedge more \longrightarrow$
 $((\Box (init\ w) \wedge f) \wedge more); (\Box (init\ w) \wedge f^*)$
using 2 3 5 **by** fastforce
hence 7: $\vdash (\Box (init\ w) \wedge f^*) \wedge finite \longrightarrow (\Box (init\ w) \wedge f)^*$
using CSIntro **by** blast
have 71: $\vdash init\ w \wedge \Box (init\ w) \wedge f^* \wedge finite \longrightarrow init\ w \wedge (\Box (init\ w) \wedge f)^* \wedge finite$
using 7 **by** fastforce
have 8: $\vdash \Box (init\ w) \wedge f^* \wedge finite \longrightarrow init\ w \wedge (\Box (init\ w) \wedge f)^* \wedge finite$

using 1 71 by fastforce
 have 11: $\vdash (\Box (init\ w) \wedge f)^* \longrightarrow (\Box (init\ w))^*$
 by (rule AndCSA)
 have 12: $\vdash (init\ w \wedge (\Box (init\ w))^*) = \Box (init\ w)$
 by (rule BoxCSEqvBox)
 have 13: $\vdash (\Box (init\ w) \wedge f)^* \longrightarrow f^*$
 by (rule AndCSB)
 have 14: $\vdash init\ w \wedge (\Box (init\ w) \wedge f)^* \longrightarrow init\ w \wedge (\Box (init\ w))^* \wedge f^*$
 using 11 13 by fastforce
 have 15: $\vdash init\ w \wedge (\Box (init\ w))^* \wedge f^* \longrightarrow \Box (init\ w) \wedge f^*$
 using 12 by auto
 have 16: $\vdash init\ w \wedge (\Box (init\ w) \wedge f)^* \longrightarrow \Box (init\ w) \wedge f^*$
 using 14 15 lift-imp-trans by blast
 from 8 16 show ?thesis by fastforce
 qed

lemma BaCSImpCS:

$\vdash ba\ (f \longrightarrow g) \wedge finite \longrightarrow f^* \longrightarrow g^*$
 proof –
 have 1: $\vdash f^* = (empty \vee (f \wedge more); f^*)$
 by (rule ChopstarEqv)
 have 2: $\vdash g^* = (empty \vee (g \wedge more); g^*)$
 by (rule ChopstarEqv)
 have 21: $\vdash \neg(g^*) = (\neg empty \wedge \neg((g \wedge more); g^*))$
 using 2 by fastforce
 hence 22: $\vdash \neg(g^*) = (more \wedge \neg((g \wedge more); g^*))$
 using NotEmptyEqvMore by fastforce
 have 3: $\vdash f^* \wedge \neg(g^*) \longrightarrow$
 $(empty \vee (f \wedge more); f^*) \wedge more \wedge \neg((g \wedge more); g^*)$
 using 1 22 by fastforce
 have 31: $\vdash ((empty \vee (f \wedge more); f^*) \wedge more) = ((f \wedge more); f^* \wedge more)$
 by (auto simp: empty-d-def)
 have 32: $\vdash f^* \wedge \neg(g^*) \longrightarrow (f \wedge more); f^* \wedge \neg((g \wedge more); g^*)$
 using 3 31 by fastforce
 have 4: $\vdash (f \longrightarrow g) \longrightarrow (f \wedge more \longrightarrow g \wedge more)$
 by auto
 hence 5: $\vdash ba\ (f \longrightarrow g) \longrightarrow ba\ (f \wedge more \longrightarrow g \wedge more)$
 by (rule BaImpBa)
 have 6: $\vdash ba\ (f \wedge more \longrightarrow g \wedge more) \longrightarrow$
 $(f \wedge more); f^* \longrightarrow (g \wedge more); f^*$
 by (rule BaLeftChopImpChop)
 have 7: $\vdash ba\ (f \longrightarrow g) \wedge (f \wedge more); f^* \longrightarrow (g \wedge more); f^*$
 using 5 6 by fastforce
 have 8: $\vdash (g \wedge more); f^* \wedge \neg((g \wedge more); g^*)$
 $\longrightarrow (g \wedge more); (f^* \wedge \neg(g^*))$
 by (rule ChopAndNotChopImp)
 have 9: $\vdash (g \wedge more); (f^* \wedge \neg(g^*)) \longrightarrow more; (f^* \wedge \neg(g^*))$
 by (rule AndChopB)
 have 10: $\vdash ba\ (f \longrightarrow g) \longrightarrow more; (f^* \wedge \neg(g^*)) \longrightarrow$
 $more; (ba\ (f \longrightarrow g) \wedge f^* \wedge \neg(g^*))$

by (rule BaChopImpChopBa)
 have 11: $\vdash ba (f \longrightarrow g) \wedge f^* \wedge \neg (g^*) \longrightarrow$
 $more ; (ba (f \longrightarrow g) \wedge f^* \wedge \neg (g^*))$
 using 32 7 8 9 10 by fastforce
 hence 12: $\vdash finite \longrightarrow \neg ((ba (f \longrightarrow g)) \wedge (f^*) \wedge (\neg (g^*)))$
 using MoreChopLoopFiniteB by blast
 from 12 show ?thesis by (simp add: Valid-def)
 qed

lemma BaCSEqvCS:

$\vdash ba (f = g) \wedge finite \longrightarrow (f^* = g^*)$
 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: $\vdash ba (f \longrightarrow g) \wedge finite \longrightarrow (f^* \longrightarrow g^*)$ by (rule BaCSImpCS)
 have 3: $\vdash ba (g \longrightarrow f) \wedge finite \longrightarrow (g^* \longrightarrow f^*)$ by (rule BaCSImpCS)
 have 4: $\vdash ba (f = g) \wedge finite \longrightarrow (f^* \longrightarrow g^*) \wedge (g^* \longrightarrow f^*)$ using 1 2 3 by fastforce
 have 5: $\vdash ((f^* \longrightarrow g^*) \wedge (g^* \longrightarrow f^*)) = (f^* = g^*)$ by auto
 from 4 5 show ?thesis by auto
 qed

lemma BaAndCSImport:

$\vdash ba f \wedge g^* \wedge finite \longrightarrow (f \wedge g)^*$
 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: $\vdash ba (g \longrightarrow f \wedge g) \wedge finite \longrightarrow g^* \longrightarrow (f \wedge g)^*$ by (rule BaCSImpCS)
 from 2 3 show ?thesis by fastforce
 qed

lemma CSSkipImpFinite:

$\vdash skip^* \longrightarrow finite$
 using CSElimWithoutMore EmptyImpFinite SkipChopFiniteImpFinite by blast

lemma FiniteImpCSSkip:

$\vdash finite \longrightarrow skip^*$
 using CSIntro
 by (metis (no-types, lifting) CSSkipImpFinite ChopAndB FiniteChopSkipEqvFiniteAndMore
 FiniteChopSkipEqvSkipChopFinite ImpChopPlus Prop10 Prop12 int-iffD1 inteq-reflection)

lemma CSSkipEqvFinite:

$\vdash skip^* = finite$
 using CSSkipImpFinite FiniteImpCSSkip by fastforce

6.9 Properties of Omega

lemma NotOmegaEmpty:

$\vdash \neg((empty)^\omega)$
 proof –
 have 1: $\vdash (empty)^\omega = (empty \wedge fmore);(empty)^\omega$

```

  by (simp add: OmegaUnroll)
have 2:  $\vdash (\text{empty} \wedge \text{fmore}) = \#False$ 
  using NotFmoreAndEmpty by auto
have 3:  $\vdash \#False;(\text{empty})^\omega = \#False$ 
  by (metis AndInfChopEqvAndInf int-eq int-simps(22))
from 1 2 3 show ?thesis
  by (metis TrueW int-simps(3) inteq-reflection)
qed

```

```

lemma NotOmegaFalse:
 $\vdash \neg((\#False)^\omega)$ 
by (metis ChopImpDi DiIntro NotDiFalse OmegaUnroll int-iffI int-simps(14)
    int-simps(19) inteq-reflection)

```

```

lemma NotOmegaInf:
 $\vdash \neg((\text{inf})^\omega)$ 
proof -
  have 1:  $\vdash (\text{inf})^\omega = (\text{inf} \wedge \text{fmore});(\text{inf})^\omega$ 
    by (simp add: OmegaUnroll)
  have 2:  $\vdash (\text{inf} \wedge \text{fmore}) = \#False$ 
    using FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite
      FmoreEqvSkipChopFinite InfEqvNotFinite by fastforce
  have 3:  $\vdash \#False;(\text{inf})^\omega = \#False$ 
    by (metis AndInfChopEqvAndInf int-eq int-simps(22))
from 1 2 3 show ?thesis
  by (metis TrueW int-simps(3) inteq-reflection)
qed

```

```

lemma OmegaLenPlusOneImpInf:
 $\vdash (\text{len}(\text{Suc } n))^\omega \longrightarrow \text{inf}$ 
by (metis AndChopB MoreChopLoop OmegaUnroll finite-d-def int-simps(32) inteq-reflection)

```

```

lemma InfImpOmegaLenPlusOne:
 $\vdash \text{inf} \longrightarrow (\text{len}(\text{Suc } n))^\omega$ 
proof -
  have 1:  $\vdash \text{inf} \wedge \#True \wedge \Box(\#True \longrightarrow (\text{len}(\text{Suc } n) \wedge \text{fmore});\#True) \longrightarrow (\text{len}(\text{Suc } n))^\omega$ 
    using OmegaInduct by blast
  have 2:  $\vdash \Box(\#True \longrightarrow (\text{len}(\text{Suc } n) \wedge \text{fmore});\#True) = \text{inf}$ 
    by (auto simp add: Valid-def itl-defs,
        metis enat-min-eq-0-iff ndropn-eq-NNil ndropn-nlast ndropn-nlength nfinite-nlength-enat
            nlength-NNil the-enat.simps zero-enat-def,
        metis le-cases min.orderE ndropn-nlength nfinite-ndropn-b nfinite-ntaken ntaken-all,
        metis linear min.order-iff ndropn-nlength nfinite-ndropn-b nfinite-ntaken ntaken-all
            ntaken-nlength,
        metis ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD)
from 1 2 show ?thesis by fastforce
qed

```

```

lemma OmegaLenPlusOneEqvInf:
 $\vdash (\text{len}(\text{Suc } n))^\omega = \text{inf}$ 

```

using *OmegaLenPlusOneImpInf InfImpOmegaLenPlusOne* **by** *fastforce*

lemma *OmegaSkipEqvInf*:

$\vdash (\text{skip})^\omega = \text{inf}$

proof –

have 1: $\vdash \text{skip} = (\text{len } 1)$

by (*simp add: len-d-def power-d-def*)

(*metis CSSkipEqvFinite ChopEmpty FiniteChopSkipEqvFiniteAndMore
FiniteChopSkipEqvSkipChopFinite ImpChopPlus Prop10 Prop12 inteq-reflection*)

have 2: $\vdash (\text{skip})^\omega = (\text{len } 1)^\omega$

using 1 **by** (*metis OmegaUnroll inteq-reflection*)

from 2 **show** *?thesis* **using** *OmegaLenPlusOneEqvInf* **by** *fastforce*

qed

lemma *OmegaTrueImpInf*:

$\vdash (\# \text{True})^\omega \longrightarrow \text{inf}$

by (*metis AndChopB MoreChopLoop OmegaUnroll finite-d-def int-simps(32) inteq-reflection*)

lemma *InfImpOmegaTrue*:

$\vdash \text{inf} \longrightarrow (\# \text{True})^\omega$

proof –

have 1: $\vdash \text{inf} \wedge \# \text{True} \wedge \Box(\# \text{True} \longrightarrow (\# \text{True} \wedge \text{fmore}); \# \text{True}) \longrightarrow \# \text{True}^\omega$

using *OmegaInduct* **by** *blast*

have 2: $\vdash \Box(\# \text{True} \longrightarrow (\# \text{True} \wedge \text{fmore}); \# \text{True}) = \text{inf}$

by (*auto simp add: Valid-def itl-defs*)

(*metis enat-min-eq-0-iff ndropn-eq-NNil ndropn-nlast ndropn-nlength nfinite-nlength-enat
nlength-NNil the-enat.simps zero-enat-def,
metis antisym-conv2 canonically-ordered-monoid-add-class.lessE enat.distinct(2) enat-add-sub-same
ileI1 less-numeral-extra(1) min-def nlength-eq-enat-nfiniteD one-eSuc one-enat-def zero-enat-def
zero-le,
metis ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *OmegaTrueEqvInf*:

$\vdash (\# \text{True})^\omega = \text{inf}$

using *OmegaTrueImpInf InfImpOmegaTrue* **by** *fastforce*

lemma *OmegaMoreImpInf*:

$\vdash (\text{more})^\omega \longrightarrow \text{inf}$

by (*metis AndChopB MoreChopLoop OmegaUnroll finite-d-def int-simps(32) inteq-reflection*)

lemma *InfImpOmegaMore*:

$\vdash \text{inf} \longrightarrow (\text{more})^\omega$

proof –

have 1: $\vdash \text{inf} \wedge \# \text{True} \wedge \Box(\# \text{True} \longrightarrow (\text{more} \wedge \text{fmore}); \# \text{True}) \longrightarrow \text{more}^\omega$

using *OmegaInduct* **by** *blast*

have 2: $\vdash \Box(\# \text{True} \longrightarrow (\text{more} \wedge \text{fmore}); \# \text{True}) = \text{inf}$

by (*auto simp add: Valid-def itl-defs*)

(*metis enat-min-eq-0-iff ndropn-eq-NNil ndropn-nlast ndropn-nlength nfinite-nlength-enat*)

```

    nlength-NNil the-enat.simps zero-enat-def,
    metis ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD,
    metis add.left-neutral canonically-ordered-monoid-add-class.lessE enat.distinct(2)
    enat-add-sub-same ileI1 less-numeral-extra(1) min-def nlength-eq-enat-nfiniteD one-eSuc
    one-enat-def order.not-eq-order-implies-strict zero-enat-def zero-le,
    metis ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD)
from 1 2 show ?thesis by fastforce
qed

lemma OmegaMoreEqvInf:
   $\vdash (more)^\omega = inf$ 
using OmegaMoreImpInf InfImpOmegaMore by fastforce

lemma OmegaFiniteImpInf:
   $\vdash (finite)^\omega \longrightarrow inf$ 
by (metis AndChopB ChopLoop OmegaUnroll TrueW finite-d-def int-simps(13) int-simps(32) inteq-reflection)

lemma InfImpOmegaFinite:
   $\vdash inf \longrightarrow (finite)^\omega$ 
proof –
  have 1:  $\vdash inf \wedge \#True \wedge \Box(\#True \longrightarrow (finite \wedge fmore); \#True) \longrightarrow finite^\omega$ 
    using OmegaInduct by blast
  have 2:  $\vdash \Box(\#True \longrightarrow (finite \wedge fmore); \#True) = inf$ 
    by (auto simp add: Valid-def itl-defs)
    (metis enat-min-eq-0-iff ndropn-eq-NNil ndropn-nlast ndropn-nlength nfinite-nlength-enat nlength-NNil
    the-enat.simps zero-enat-def,
    metis antisym-conv2 canonically-ordered-monoid-add-class.lessE enat.distinct(2) enat-add-sub-same
    ileI1 less-numeral-extra(1) min-def nlength-eq-enat-nfiniteD one-eSuc one-enat-def zero-enat-def
    zero-le,
    metis ndropn-nlength nfinite-ndropn-b nlength-eq-enat-nfiniteD)
from 1 2 show ?thesis by fastforce
qed

lemma OmegaFiniteEqvInf:
   $\vdash (finite)^\omega = inf$ 
using OmegaFiniteImpInf InfImpOmegaFinite by fastforce

lemma BoxStateAndInfImpOmegaBoxState:
   $\vdash inf \wedge \Box(init\ w) \longrightarrow (\Box(init\ w))^\omega$ 
proof –
  have 1:  $\vdash inf \wedge (inf \wedge \Box(init\ w)) \wedge$ 
     $\Box((inf \wedge \Box(init\ w)) \longrightarrow (\Box(init\ w) \wedge fmore); (inf \wedge \Box(init\ w))) \longrightarrow (\Box(init\ w))^\omega$ 
    using OmegaInduct by blast
  have 10:  $\vdash (skip; finite); inf = inf$ 
    by (metis AndChopA AndMoreAndFiniteEqvAndFmore FiniteChopInfEqvInf FiniteChopSkipEqvFinite-
    AndMore
    FiniteChopSkipEqvSkipChopFinite OmegaFiniteEqvInf OmegaUnroll Prop11 inteq-reflection)
  have 11:  $\vdash (inf \wedge \Box(init\ w)) = (\Box(init\ w) \wedge (skip; finite); inf)$ 
    using 10 by auto

```

have 12: $\vdash (inf \wedge (\Box (init\ w) \wedge fmore); \Box (init\ w)) = (\Box (init\ w) \wedge fmore); (\Box (init\ w) \wedge inf)$
by (*meson ChopAndInf Prop04 lift-and-com*)
have 2: $\vdash (inf \wedge \Box (init\ w)) = (inf \wedge (\Box (init\ w) \wedge fmore); \Box (init\ w))$
by (*metis 11 12 BoxStateAndChopEqvChop FmoreEqvSkipChopFinite inteq-reflection*)
have 3: $\vdash (inf \wedge (\Box (init\ w) \wedge fmore); \Box (init\ w)) = (\Box (init\ w) \wedge fmore); (inf \wedge \Box (init\ w))$
by (*metis ChopAndInf inteq-reflection lift-and-com*)
have 4: $\vdash inf \wedge \Box (init\ w) \longrightarrow \Box ((inf \wedge \Box (init\ w)) \longrightarrow (\Box (init\ w) \wedge fmore); (inf \wedge \Box (init\ w)))$
using 2 3 by (*metis (mono-tags, lifting) BoxGen intD intI inteq-reflection unl-lift2*)
from 1 4 show ?thesis by fastforce
qed

lemma *OmegaBoxStateImpBoxState:*

$\vdash (\Box (init\ w))^\omega \wedge inf \longrightarrow \Box (init\ w)$

proof –

have 1: $\vdash (\Box (init\ w))^\omega \longrightarrow init\ w$
by (*metis AndChopA BoxEqvAndEmptyOrNextBox OmegaUnroll Prop12 StateAndChop inteq-reflection*)
have 2: $\vdash (\Box (init\ w))^\omega \longrightarrow (\Box (init\ w) \wedge fmore); ((\Box (init\ w))^\omega)$
by (*simp add: OmegaUnroll int-iffD1*)
have 21: $\vdash (\Box (init\ w) \wedge fmore) \longrightarrow \Box (\Box (init\ w))$
by (*metis AndChopB BoxStateAndChopEqvChop FmoreEqvSkipChopFinite NextAndEqvNextAndNext Prop12 inteq-reflection next-d-def*)
have 22: $\vdash finite = (empty \vee fmore)$
by (*metis CSEqvOrChopCS CSSkipEqvFinite FmoreEqvSkipChopFinite int-eq*)
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: $\vdash \Box (\Box (init\ w)) \wedge fmore \longrightarrow \Box ((init\ w) \wedge empty) \vee (\Box (init\ w) \wedge fmore)$
using 23 24 by (*metis FmoreEqvSkipChopFinite NextAndEqvNextAndNext SkipChopEqvNext int-iffD2 inteq-reflection*)
have 26: $\bigwedge g. \vdash (\Box ((init\ w) \wedge empty) \vee (\Box (init\ w) \wedge fmore)); g = (\Box (init\ w \wedge g) \vee \Box ((\Box (init\ w) \wedge fmore); g))$
proof –
fix *g* :: 'a nellist \Rightarrow bool
have f1: $\vdash \Box (init\ w \wedge empty \vee \Box (init\ w) \wedge fmore); g = \Box ((init\ w \wedge empty \vee \Box (init\ w) \wedge fmore); g)$
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 $\vdash \Box (init\ w \wedge empty \vee \Box (init\ w) \wedge fmore); g = (\Box (init\ w \wedge g) \vee \Box ((\Box (init\ w) \wedge fmore); g))$
by (*metis (no-types, hide-lams) 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: $\vdash (skip; (\Box (init\ w) \wedge finite)); \Box (init\ w)^\omega = (skip; (init\ w \wedge \Box (init\ w)^\omega) \vee skip; ((\Box (init\ w) \wedge fmore); \Box (init\ w)^\omega))$

by (metis 26 28 SkipChopEqvNext int-eq)
 have 3: $\vdash (\Box(\text{init } w) \wedge \text{fmore});((\Box(\text{init } w))^\omega) \longrightarrow$
 $\quad (\bigcirc(\text{init } w \wedge (\Box(\text{init } w))^\omega) \vee \bigcirc((\Box(\text{init } w) \wedge \text{fmore});((\Box(\text{init } w))^\omega)))$
 by (metis 27 29 AndChopB LeftChopImpChop inteq-reflection next-d-def)
 have 4: $\vdash (\bigcirc(\text{init } w \wedge (\Box(\text{init } w))^\omega) \vee \bigcirc((\Box(\text{init } w) \wedge \text{fmore});((\Box(\text{init } w))^\omega))) \longrightarrow$
 $\quad \bigcirc(\Box(\text{init } w))^\omega$
 by (metis ChopAndB NextImpNext OmegaUnroll Prop02 Prop11 next-d-def)
 have 5: $\vdash (\Box(\text{init } w))^\omega \longrightarrow \bigcirc(\Box(\text{init } w))^\omega$
 using 2 3 4 by fastforce
 from 1 5 show ?thesis using BoxIntro by (metis Prop01 Prop05 inteq-reflection lift-and-com)
 qed

lemma OmegaIntro:

assumes $\vdash h \longrightarrow (f \wedge \text{fmore});h$

shows $\vdash h \wedge \text{inf} \longrightarrow f^\omega$

proof –

have 1: $\vdash h \longrightarrow (f \wedge \text{fmore});h$ using assms by auto

have 2: $\vdash \Box(h \longrightarrow (f \wedge \text{fmore});h)$ by (simp add: BoxGen assms)

from 1 2 show ?thesis using OmegaInduct by fastforce

qed

lemma OmegaImpRule:

assumes $\vdash f \longrightarrow g$

shows $\vdash f^\omega \wedge \text{inf} \longrightarrow g^\omega$

proof –

have 1: $\vdash (f \wedge \text{fmore}) \longrightarrow (g \wedge \text{fmore})$

using assms by auto

have 2: $\vdash (f \wedge \text{fmore});f^\omega \longrightarrow (g \wedge \text{fmore});f^\omega$

using 1 by (simp add: LeftChopImpChop)

have 3: $\vdash \Box(f^\omega \longrightarrow (f \wedge \text{fmore});f^\omega) \longrightarrow \Box(f^\omega \longrightarrow (g \wedge \text{fmore});f^\omega)$

by (metis 2 OmegaUnroll int-eq-true int-simps(13) inteq-reflection)

have 4: $\vdash (\text{inf} \wedge f^\omega \wedge \Box(f^\omega \longrightarrow (f \wedge \text{fmore});f^\omega)) \longrightarrow$
 $\quad \text{inf} \wedge f^\omega \wedge \Box(f^\omega \longrightarrow (g \wedge \text{fmore});f^\omega)$

using 3 by fastforce

have 5: $\vdash \text{inf} \wedge f^\omega \wedge \Box(f^\omega \longrightarrow (g \wedge \text{fmore});f^\omega) \longrightarrow g^\omega$

using OmegaInduct by blast

have 6: $\vdash f^\omega \longrightarrow (f \wedge \text{fmore});f^\omega$

by (simp add: OmegaUnroll int-iffD1)

have 7: $\vdash \Box(f^\omega \longrightarrow (f \wedge \text{fmore});f^\omega)$

using 6 by (simp add: BoxGen)

from 3 5 7 show ?thesis by fastforce

qed

lemma OmegaEqvRule:

assumes $\vdash f = g$

shows $\vdash f^\omega = g^\omega$

using assms using int-eq by force

lemma AndOmegaA:

$\vdash (f \wedge g)^\omega \wedge \text{inf} \longrightarrow f^\omega$

by (meson OmegaImpRule Prop12 int-iffD2 lift-and-com)

lemma AndOmegaB:

$\vdash (f \wedge g)^\omega \wedge \text{inf} \longrightarrow g^\omega$

by (meson OmegaImpRule Prop12 int-iffD2 lift-and-com)

lemma BaOmegaImpOmega:

$\vdash \text{ba } (f \longrightarrow g) \wedge \text{inf} \longrightarrow f^\omega \longrightarrow g^\omega$

proof –

have 1: $\vdash \text{ba } (f \longrightarrow g) \wedge (f \wedge \text{fmore}); f^\omega \longrightarrow ((f \longrightarrow g) \wedge (f \wedge \text{fmore})); ((f \longrightarrow g) \wedge f^\omega)$

using BaAndChopImport by fastforce

have 2: $\vdash (f \longrightarrow g) \wedge (f \wedge \text{fmore}) \longrightarrow (g \wedge \text{fmore})$

by auto

have 3: $\vdash (f \longrightarrow g) \wedge f^\omega \longrightarrow f^\omega$

by auto

have 4: $\vdash \text{ba } (f \longrightarrow g) \wedge (f \wedge \text{fmore}); f^\omega \longrightarrow (g \wedge \text{fmore}); f^\omega$

using 1 2 3

by (metis (no-types, lifting) AndChopB ChopAndB Prop10 int-eq lift-imp-trans)

have 5: $\vdash \text{ba } (f \longrightarrow g) \wedge (f \wedge \text{fmore}); f^\omega \longrightarrow ((f \longrightarrow g) \wedge (f \wedge \text{fmore})); (\text{ba}(f \longrightarrow g) \wedge f^\omega)$

using BaAndChopImportB by blast

have 6: $\vdash ((f \longrightarrow g) \wedge (f \wedge \text{fmore})); (\text{ba}(f \longrightarrow g) \wedge f^\omega) \longrightarrow ((g \wedge \text{fmore}); (\text{ba}(f \longrightarrow g) \wedge f^\omega))$

using 2 LeftChopImpChop by blast

have 7: $\vdash (\text{ba } (f \longrightarrow g) \wedge f^\omega) \longrightarrow (g \wedge \text{fmore}); (\text{ba } (f \longrightarrow g) \wedge f^\omega)$

using OmegaUnroll 5 6 by fastforce

have 8: $\vdash (\text{ba } (f \longrightarrow g) \wedge f^\omega) \wedge \text{inf} \longrightarrow g^\omega$

using 7 OmegaIntro by blast

from 8 show ?thesis by fastforce

qed

lemma BaOmegaEqvOmega:

$\vdash \text{ba } (f = g) \wedge \text{inf} \longrightarrow (f^\omega = g^\omega)$

proof –

have 0: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$ by fastforce

have 1: $\vdash \text{ba } (f = g) = (\text{ba } (f \longrightarrow g) \wedge \text{ba } (g \longrightarrow f))$ by (metis 0 BaAndEqv int-eq)

have 2: $\vdash \text{ba } (f \longrightarrow g) \wedge \text{inf} \longrightarrow (f^\omega \longrightarrow g^\omega)$ using BaOmegaImpOmega by blast

have 3: $\vdash \text{ba } (g \longrightarrow f) \wedge \text{inf} \longrightarrow (g^\omega \longrightarrow f^\omega)$ using BaOmegaImpOmega by blast

have 4: $\vdash \text{ba } (f = g) \wedge \text{inf} \longrightarrow (f^\omega \longrightarrow g^\omega) \wedge (g^\omega \longrightarrow f^\omega)$ using 1 2 3 by fastforce

have 5: $\vdash ((f^\omega \longrightarrow g^\omega) \wedge (g^\omega \longrightarrow f^\omega)) = (f^\omega = g^\omega)$ by auto

from 4 5 show ?thesis by auto

qed

lemma BaAndOmegaImport:

$\vdash \text{ba } f \wedge g^\omega \wedge \text{inf} \longrightarrow (f \wedge g)^\omega$

proof –

have 1: $\vdash f \longrightarrow (g \longrightarrow (f \wedge g))$ by auto

hence 2: $\vdash \text{ba } f \longrightarrow \text{ba } (g \longrightarrow f \wedge g)$ by (rule BaImpBa)

have 3: $\vdash \text{ba } (g \longrightarrow f \wedge g) \wedge \text{inf} \longrightarrow g^\omega \longrightarrow (f \wedge g)^\omega$ 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

6.10 Properties of While

lemma *WhileEqvIf*:

$\vdash ((while (init w) do f) \wedge finite) =$
 $(if_i (init w) then (f; (while (init w) do f)) else empty \wedge finite)$

proof –

have 1: $\vdash (while (init w) do f \wedge finite) =$
 $(((((init w) \wedge f)^* \wedge fin (\neg (init w))) \wedge finite)$

by (*simp add: while-d-def*)

have 2: $\vdash (init w \wedge f)^* = (empty \vee ((init w \wedge f); (init w \wedge f)^*))$

by (*rule CSEqvOrChopCS*)

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: $\vdash ((empty \wedge fin (\neg (init w)) \wedge finite) \vee$

$$((init\ w \wedge f); (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w)) \wedge finite) =$$

$$((\neg\ (init\ w) \wedge empty) \vee$$

$$((init\ w \wedge f); (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w)) \wedge finite))$$
using 23 33 **by** *auto*

have 4: $\vdash (init\ w \wedge f); (init\ w \wedge f)^* = (init\ w \wedge (f; (init\ w \wedge f)^*))$
by (*rule StateAndChop*)

have 41: $\vdash (((init\ w \wedge f); (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w)) \wedge finite) =$

$$(init\ w \wedge (f; (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w)) \wedge finite)$$

using 4 **by** *auto*

have 42: $\vdash (init\ w \wedge ((f; (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w)) \wedge finite)) =$

$$(init\ w \wedge ((f; (init\ w \wedge f)^*) \wedge fin\ (init\ (\neg\ w)) \wedge finite))$$

using *Initprop(2)* **by** (*metis StateAndEmptyChop int-eq*)

have 5: $\vdash ((f; ((init\ w \wedge f)^*)) \wedge (fin\ (init\ (\neg\ w))) \wedge finite)$

$$= ((f \wedge finite); ((init\ w \wedge f)^* \wedge (fin\ (init\ (\neg\ w))) \wedge finite))$$

using *ChopAndFin* **by** *fastforce*

hence 49: $\vdash (init\ w \wedge (f; (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w)) \wedge finite) =$

$$(init\ w \wedge (f \wedge finite); ((init\ w \wedge f)^* \wedge (fin\ (init\ (\neg\ w))) \wedge finite))$$

using 42 **by** *fastforce*

have 50: $\vdash (((init\ w \wedge f); (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w)) \wedge finite) =$

$$(init\ w \wedge (f \wedge finite); ((init\ w \wedge f)^* \wedge (fin\ (init\ (\neg\ w))) \wedge finite))$$

by (*meson 41 49 Prop04 lift-and-com*)

have 51: $\vdash (init\ w \wedge (f \wedge finite); ((init\ w \wedge f)^* \wedge (fin\ (init\ (\neg\ w))) \wedge finite)) =$

$$(init\ w \wedge (f \wedge finite); ((init\ w \wedge f)^* \wedge (fin\ (\neg\ (init\ w))) \wedge finite))$$

by (*metis (no-types) EmptyChop Initprop(2) inteq-reflection*)

have 52: $\vdash (((init\ w \wedge f); (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w)) \wedge finite) =$

$$(init\ w \wedge ((f \wedge finite); ((init\ w \wedge f)^* \wedge fin\ (\neg\ (init\ w)) \wedge finite)))$$

using 50 51 **by** *fastforce*

have 53: $\vdash ((\neg\ (init\ w) \wedge empty) \vee$

$$((init\ w \wedge f); (init\ w \wedge f)^*) \wedge fin\ (\neg\ (init\ w)) \wedge finite) =$$

$$((\neg\ (init\ w) \wedge empty) \vee$$

$$(init\ w \wedge ((f \wedge finite); ((init\ w \wedge f)^* \wedge fin\ (\neg\ (init\ w)) \wedge finite)))$$

using 52 34 **by** *auto*

have 6: $\vdash ((f \wedge finite); (((init\ w \wedge f)^* \wedge fin\ (\neg\ (init\ w))) \wedge finite)) =$

$$(f \wedge finite); (while\ (init\ w)\ do\ f \wedge finite)$$

by (*simp add: while-d-def*)

have 61: $\vdash (init\ w \wedge ((f \wedge finite); (((init\ w \wedge f)^* \wedge fin\ (\neg\ (init\ w))) \wedge finite))) =$

$$(init\ w \wedge ((f \wedge finite); (while\ (init\ w)\ do\ f \wedge finite)))$$

using 6 **by** *auto*

have 62: $\vdash ((\neg\ (init\ w) \wedge empty) \vee$

$$(init\ w \wedge ((f \wedge finite); (((init\ w \wedge f)^* \wedge fin\ (\neg\ (init\ w))) \wedge finite)))$$

$$= ((\neg\ (init\ w) \wedge empty) \vee$$

$$(init\ w \wedge ((f \wedge finite); (while\ (init\ w)\ do\ f \wedge finite)))$$

using 61 **by** *fastforce*

have 7: $\vdash (while\ (init\ w)\ do\ f \wedge finite)$

$$= (((\neg\ (init\ w) \wedge empty) \vee$$

$$(init\ w \wedge (f ; while\ (init\ w)\ do\ f) \wedge finite)))$$

proof –

have $\vdash (while\ (init\ w)\ do\ f \wedge finite) =$

$$(\neg\ init\ w \wedge empty \vee (init\ w \wedge f); (init\ w \wedge f)^* \wedge fin\ (\neg\ init\ w) \wedge finite)$$

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 *IfAndFiniteDist*:

$\vdash (\text{if}_i (\text{init } w) \text{ then } (f;g) \text{ else } \text{empty} \wedge \text{finite}) =$
 $(\text{if}_i (\text{init } w) \text{ then } ((f \wedge \text{finite});(g \wedge \text{finite})) \text{ else } \text{empty})$
proof –
have 1: $\vdash (\text{if}_i (\text{init } w) \text{ then } (f;g) \text{ else } \text{empty} \wedge \text{finite}) =$
 $((\text{init } w \wedge (f;g)) \vee (\neg(\text{init } w) \wedge \text{empty})) \wedge \text{finite}$
by (*auto simp: ifthenelse-d-def*)
have 2: $\vdash ((\text{init } w \wedge (f;g)) \vee (\neg(\text{init } w) \wedge \text{empty})) \wedge \text{finite} =$
 $((\text{init } w \wedge (f;g) \wedge \text{finite}) \vee (\neg(\text{init } w) \wedge \text{empty} \wedge \text{finite})))$
by *auto*
have 3: $\vdash (\text{init } w \wedge (f;g) \wedge \text{finite}) = (\text{init } w \wedge (f \wedge \text{finite});(g \wedge \text{finite}))$
using *ChopAndFiniteDist* **by** *fastforce*
have 4: $\vdash (\neg(\text{init } w) \wedge \text{empty} \wedge \text{finite}) = (\neg(\text{init } w) \wedge \text{empty})$
using *FiniteAndEmptyEqvEmpty* **by** *auto*
have 5: $\vdash ((\text{init } w \wedge (f;g) \wedge \text{finite}) \vee (\neg(\text{init } w) \wedge \text{empty} \wedge \text{finite}))) =$
 $((\text{init } w \wedge (f \wedge \text{finite});(g \wedge \text{finite})) \vee (\neg(\text{init } w) \wedge \text{empty}))$
by (*metis 2 3 4 inteq-reflection*)
have 6: $\vdash ((\text{init } w \wedge (f \wedge \text{finite});(g \wedge \text{finite})) \vee (\neg(\text{init } w) \wedge \text{empty})) =$
 $(\text{if}_i (\text{init } w) \text{ then } ((f \wedge \text{finite});(g \wedge \text{finite})) \text{ else } \text{empty})$
by (*auto simp: ifthenelse-d-def*)
from 1 2 5 6 **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *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 (rule ChopAssocB)
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 WhileChopEqvIfRule:

assumes $\vdash f = (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}); h$
shows $\vdash f = \text{if}_i \text{ (init } w) \text{ then } ((g \wedge \text{finite}); f) \text{ else } h$
proof –
have 1: $\vdash f = (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}); h$
using assms **by** auto
have 2: $\vdash (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}); h =$
 $\text{if}_i \text{ (init } w) \text{ then } ((g \wedge \text{finite}); ((\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}); h)) \text{ else } h$
by (rule WhileChopEqvIf)
have 3: $\vdash ((g \wedge \text{finite}); f) = ((g \wedge \text{finite}); ((\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}); h))$
using 1 **by** (rule RightChopEqvChop)
have 4: $\vdash ((g \wedge \text{finite}); ((\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}); h)) = ((g \wedge \text{finite}); f)$
using 3 **by** auto
have 5: $\vdash \text{if}_i \text{ (init } w) \text{ then } ((g \wedge \text{finite}); ((\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}); h)) \text{ else } h =$
 $\text{if}_i \text{ (init } w) \text{ then } ((g \wedge \text{finite}); f) \text{ else } h$
using 4 **using** inteq-reflection **by** fastforce
from 1 2 5 **show** ?thesis **by** fastforce
qed

lemma 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 *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 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 71: $\vdash ((\neg (\text{init } w) \wedge \text{empty}) \vee$

$((\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})) =$

$((\neg (\text{init } w) \wedge \text{empty}) \vee$

$(\text{init } w \wedge (f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\text{init } (\neg w)) \wedge \text{finite}))$

using 7

by (*metis (no-types, hide-lams) ChopEmpty Initprop(2) inteq-reflection*)

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: $\vdash \text{fin } (\text{init } (\neg w)) = \text{fin } (\neg (\text{init } w))$

using *FinEqvFin Initprop(2)* **by** *fastforce*

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: $\vdash (init\ w \wedge (f \wedge more); (init\ w \wedge f)^* \wedge fin\ (\neg (init\ w)) \wedge finite) =$
 $(init\ w \wedge ((f \wedge more) \wedge finite); ((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w)) \wedge finite))$
using 82 by fastforce
have 84: $\vdash ((\neg (init\ w) \wedge empty) \vee$
 $(init\ w \wedge (f \wedge more); (init\ w \wedge f)^* \wedge fin\ (init\ (\neg w)) \wedge finite)) =$
 $((\neg (init\ w) \wedge empty) \vee$
 $(init\ w \wedge ((f \wedge more) \wedge finite); ((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w)) \wedge finite)))$
by (metis 71 81 82 int-eq)
have 9: $\vdash ((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w)) \wedge finite) =$
 $((\neg (init\ w) \wedge empty) \vee$
 $(init\ w \wedge ((f \wedge more) \wedge finite);$
 $((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w)) \wedge finite)))$
using 84 61 by (metis 71 inteq-reflection)
have 10: $\vdash ((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w)) \wedge finite) =$
 $((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w))) \wedge finite$
by auto
hence 11: $\vdash ((\neg (init\ w) \wedge empty) \vee$
 $(init\ w \wedge ((f \wedge more) \wedge finite);$
 $((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w)) \wedge finite))) =$
 $((\neg (init\ w) \wedge empty) \vee$
 $(init\ w \wedge ((f \wedge more) \wedge finite);$
 $((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w)))) \wedge finite$
by (metis int-simps(1) inteq-reflection)
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 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 WhileEqEmptyOrChopWhile)
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: $\vdash f \wedge \text{finite} \longrightarrow (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite})$
using 4 5 *ChopContraB* **by** *blast*
from 6 **show** *?thesis* **by** (*simp add: Prop12*)
qed

lemma *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 *auto*
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** *auto*
have 4: $\vdash (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) \wedge \neg g \longrightarrow$
 $((((f \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})) \wedge$
 $\neg (((f \wedge \text{more}) \wedge \text{finite}); g))$
using 23 32 **by** *fastforce*
have 5: $\vdash (f \wedge \text{more}) \wedge \text{finite} \longrightarrow \text{more}$
by *auto*
from 4 5 **show** *?thesis* **using**
 $\text{ChopContraB}[of \text{LIFT}(\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) \text{LIFT}(g) \text{LIFT}(((f \wedge \text{more}) \wedge \text{finite}))]$
by *auto*
qed

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: $\vdash (f \longrightarrow g) \longrightarrow ((init\ w \wedge f) \longrightarrow (init\ w \wedge g))$

by *auto*

hence 2: $\vdash ba\ (f \longrightarrow g) \longrightarrow ba\ ((init\ w \wedge f) \longrightarrow (init\ w \wedge g))$

using *BaImpBa* **by** *blast*

have 3: $\vdash ba\ ((init\ w \wedge f) \longrightarrow (init\ w \wedge g)) \wedge finite \longrightarrow ((init\ w \wedge f)^* \longrightarrow (init\ w \wedge g)^*)$

by (*rule BaCSImpCS*)

have 4: $\vdash ba\ (f \longrightarrow g) \wedge finite \longrightarrow ((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w)) \longrightarrow (init\ w \wedge g)^* \wedge fin\ (\neg (init\ w)))$

using 2 3 **by** *fastforce*

from 4 **show** *?thesis* **by** (*simp add: while-d-def*)

qed

lemma *WhileImpWhile*:

assumes $\vdash f \longrightarrow g$

shows $\vdash (while\ (init\ w)\ do\ f) \wedge finite \longrightarrow (while\ (init\ w)\ do\ g)$

proof –

have 1: $\vdash f \longrightarrow g$

using *assms* **by** *auto*

hence 2: $\vdash ba\ (f \longrightarrow g)$

by (*rule BaGen*)

have 3: $\vdash ba\ (f \longrightarrow g) \wedge finite \longrightarrow (while\ (init\ w)\ do\ f) \longrightarrow (while\ (init\ w)\ do\ g)$

by (*rule BaWhileImpWhile*)

have 4: $\vdash ba\ (f \longrightarrow g) \longrightarrow (while\ (init\ w)\ do\ f \wedge finite) \longrightarrow (while\ (init\ w)\ do\ g)$

using 3 **by** (*auto simp: Valid-def*)

from 2 4 **show** *?thesis* **using** *MP* **by** *blast*

qed

6.11 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: $\vdash (\Box(empty = (init\ w)) \wedge more) =$
 $((empty = (init\ w)) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)$
by *auto*
have 5: $\vdash ((empty = (init\ w)) \wedge more) = (\neg(init\ w) \wedge more)$
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: $\vdash ((\neg(init\ w)) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)) =$
 $((\neg(init\ w)) \wedge \bigcirc(\Box(empty = (init\ w))))$
using 8 6 **by** *auto*
have 10: $\vdash \Box(empty = (init\ w)) = (((init\ w) \wedge empty) \vee (\Box(empty = (init\ w)) \wedge more))$
using 1 2 **by** *fastforce*
show *?thesis*
using 10 7 9 **by** *fastforce*
qed

lemma *HaltStateEqvIfStateThenEmptyElseNext:*

$\vdash halt(init\ w) = if_i(init\ w) \ then\ empty\ else\ (\bigcirc(halt(init\ w)))$

by (*metis BoxEmptyEqvIStateqvEmptyAndStateOrNotStateNext halt-d-def ifthenelse-d-def inteq-reflection lift-and-com*)

lemma *HaltChopEqv:*

$\vdash ((halt(init\ w)) ; f) = (if_i(init\ w) \ then\ (f) \ else\ (\bigcirc(halt(init\ w)); f))$

proof –

have 1: $\vdash halt(init\ w) =$
 $(if_i(init\ w) \ then\ empty\ else\ (\bigcirc(halt(init\ w))))$
by (*rule HaltStateEqvIfStateThenEmptyElseNext*)
hence 2: $\vdash ((halt(init\ w)); f) =$
 $(if_i(init\ w) \ then\ (empty; f) \ else\ (\bigcirc(halt(init\ w)); f))$

by (rule IfChopEqvRule)
 have 3: $\vdash \text{empty} ; f = f$
 by (rule EmptyChop)
 have 4: $\vdash (\bigcirc (\text{halt } (\text{init } w))) ; f = \bigcirc (\text{halt } (\text{init } w)) ; f$
 by (rule NextChop)
 from 2 3 4 show ?thesis by (metis inteq-reflection)
 qed

lemma AndHaltChopImp:

$\vdash \text{init } w \wedge (\text{halt } (\text{init } w)) ; f \longrightarrow f$
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: $\vdash \text{init } w \wedge \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc (\text{halt } (\text{init } w)) ; f) \longrightarrow f$
 by (auto simp: ifthenelse-d-def)
 from 1 2 show ?thesis by fastforce
 qed

lemma NotAndHaltChopImpNext:

$\vdash \neg (\text{init } w) \wedge (\text{halt } (\text{init } w)) ; f \longrightarrow \bigcirc (\text{halt } (\text{init } w)) ; f$
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: $\vdash \neg (\text{init } w) \wedge \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc (\text{halt } (\text{init } w)) ; f) \longrightarrow$
 $\quad \bigcirc (\text{halt } (\text{init } w)) ; f$
 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:

$\vdash wprev\ f = (empty \vee prev\ f)$

using nlength-eq-enat-nfiniteD by (auto simp add: Valid-def itl-defs)

lemma NotChopSkipEqvMoreAndNotChopSkip:

$\vdash (\neg f);skip = (more \wedge \neg(f;skip))$

proof –

have 1: $\vdash wprev\ f = (empty \vee prev\ f)$ using WprevEqvEmptyOrPrev by auto

hence 2: $\vdash (\neg(wprev\ f)) = (\neg(empty \vee prev\ f))$ by auto

have 3: $\vdash \neg(wprev\ f) = ((\neg f);skip)$ by (simp add: wprev-d-def prev-d-def)

have 31: $\vdash (empty \vee prev\ f) = (empty \vee (f;skip))$ by (simp add: prev-d-def)

have 32: $\vdash (empty \vee (f;skip)) = (\neg more \vee \neg\neg(f;skip))$ by (simp add: empty-d-def)

have 33: $\vdash (\neg more \vee \neg\neg(f;skip)) = (\neg(more \wedge \neg(f;skip)))$ by fastforce

have 34: $\vdash (empty \vee prev\ f) = (\neg(more \wedge \neg(f;skip)))$ using 31 32 33 by (metis int-eq)

have 4: $\vdash \neg(empty \vee prev\ f) = (more \wedge \neg(f;skip))$ using 34 by fastforce

from 2 3 4 show ?thesis by fastforce

qed

lemma HaltChopImpNotHaltChopNot:

$\vdash\ halt\ (init\ w); f \wedge finite \longrightarrow \neg(halt\ (init\ w); (\neg f))$

proof –

have 1: $\vdash halt\ (init\ w); f = if_i\ (init\ w)\ then\ f\ else\ (\bigcirc(halt\ (init\ w); f))$

by (rule HaltChopEqv)

have 2: $\vdash if_i\ (init\ w)\ then\ f\ else\ (\bigcirc(halt\ (init\ w); f)) \longrightarrow$
 $(((init\ w) \longrightarrow f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(halt\ (init\ w); f))))$

by (rule IfThenElseImp)

have 3: $\vdash halt\ (init\ w); (\neg f) =$
 $if_i\ (init\ w)\ then\ (\neg f)\ else\ (\bigcirc(halt\ (init\ w); (\neg f)))$

by (rule HaltChopEqv)

have 4: $\vdash if_i\ (init\ w)\ then\ (\neg f)\ else\ (\bigcirc(halt\ (init\ w); (\neg f))) \longrightarrow$
 $(((init\ w) \longrightarrow \neg f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(halt\ (init\ w); (\neg f)))))$

by (rule IfThenElseImp)

have 5: $\vdash halt\ (init\ w); f \wedge halt\ (init\ w); (\neg f) \longrightarrow$
 $(((init\ w) \longrightarrow f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(halt\ (init\ w); f)))) \wedge$
 $(((init\ w) \longrightarrow \neg f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(halt\ (init\ w); (\neg f)))))$

using 1 2 3 4 by fastforce

have 6: $\vdash (((init\ w) \longrightarrow f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(halt\ (init\ w); f)))) \wedge$
 $(((init\ w) \longrightarrow \neg f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(halt\ (init\ w); (\neg f))))) \longrightarrow$
 $(\bigcirc(halt\ (init\ w); f)) \wedge (\bigcirc(halt\ (init\ w); (\neg f)))$

by auto

have 7: $\vdash halt\ (init\ w); f \wedge halt\ (init\ w); (\neg f) \longrightarrow$
 $(\bigcirc(halt\ (init\ w); f)) \wedge (\bigcirc(halt\ (init\ w); (\neg f)))$

using 5 6 lift-imp-trans by blast

have 8: $\vdash ((\bigcirc(halt\ (init\ w); f)) \wedge (\bigcirc(halt\ (init\ w); (\neg f)))) =$
 $\bigcirc(halt\ (init\ w); f \wedge halt\ (init\ w); (\neg f))$

using NextAndEqvNextAndNext by fastforce

have 9: $\vdash \text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); (\neg f) \longrightarrow$
 $\quad \circ (\text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); (\neg f))$
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: $\vdash (\text{halt } (\text{init } w)); f \wedge (\text{halt } (\text{init } w)); g \wedge \text{finite} \longrightarrow$
 $\quad (\text{halt } (\text{init } w)); f \wedge (\text{halt } (\text{init } w)) \text{ yields } g$

by *auto*

have 3: $\vdash (\text{halt } (\text{init } w)); f \wedge (\text{halt } (\text{init } w)) \text{ yields } g \longrightarrow$
 $\quad (\text{halt } (\text{init } w)); (f \wedge g)$

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: $\vdash (\text{halt } (\text{init } w) \wedge f); f1 \longrightarrow$
 $\quad (\text{halt } (\text{init } w) \wedge f); (\neg g) \vee ((\text{halt } (\text{init } w) \wedge f); (f1 \wedge g))$

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: $\vdash (\text{halt } (\text{init } w) \wedge f); f1 \longrightarrow$
 $\quad \text{halt } (\text{init } w); (\neg g) \vee ((\text{halt } (\text{init } w) \wedge f); (f1 \wedge g))$

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: $\vdash (\text{halt } (\text{init } w) \wedge f); f1 \wedge \text{finite} \longrightarrow$
 $\quad \neg(\text{halt } (\text{init } w); g) \vee ((\text{halt } (\text{init } w) \wedge f); (f1 \wedge g))$

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: $\vdash \text{halt } (\text{init } w); f = \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt } (\text{init } w); f))$
by (rule *HaltChopEqv*)

hence 61: $\vdash (\text{halt } (\text{init } w); f \wedge \neg (\text{init } w)) =$
 $((\text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt } (\text{init } w); f)))) \wedge \neg (\text{init } w)$
using 6 by auto

have 62: $\vdash (\text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt } (\text{init } w); f))) \wedge$
 $\neg (\text{init } w) \longrightarrow (\bigcirc(\text{halt } (\text{init } w); f))$
by (auto simp: *ifthenelse-d-def*)

have 63: $\vdash \text{halt } (\text{init } w); f \wedge \neg (\text{init } w) \longrightarrow (\bigcirc(\text{halt } (\text{init } w); f))$
using 61 62 by fastforce

have 7: $\vdash (\text{halt } (\text{init } w); f) \wedge (\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)) \longrightarrow$
 $\bigcirc((\text{halt } (\text{init } w)); f)$
using 51 63 using *lift-imp-trans* by blast

have 8: $\vdash \Box(\neg (\text{init } w)) \longrightarrow \text{empty} \vee \bigcirc(\Box(\neg (\text{init } w)))$
by (metis *BoxImpYields WeakNextBoxImpMoreYields WnextEqvEmptyOrNext fmore-d-def int-eq*)

hence 9: $\vdash ((\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f))) \longrightarrow$
 $\neg (\text{halt } (\text{init } w); f) \vee \bigcirc((\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)))$
by (rule *EmptyOrNextChopImpRule*)

hence 10: $\vdash ((\text{halt } (\text{init } w); f) \wedge (\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f))) \longrightarrow$
 $\bigcirc((\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)))$
by fastforce

have 11: $\vdash (\text{halt } (\text{init } w); f \wedge (\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f))) \longrightarrow$
 $\bigcirc((\text{halt } (\text{init } w); f) \wedge \bigcirc((\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f))))$
using 7 10 by fastforce

have 12: $\vdash \bigcirc((\text{halt } (\text{init } w); f) \wedge \bigcirc((\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f))))$
 $\longrightarrow \bigcirc(((\text{halt } (\text{init } w); f) \wedge ((\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f))))$
using *NextAndEqvNextAndNext* by fastforce

```

have 13:  $\vdash (\text{halt } (\text{init } w)); f \wedge (\Box (\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)) \longrightarrow$ 
 $\quad \circ(((\text{halt } (\text{init } w)); f) \wedge ((\Box (\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f))))$ 
using 11 12 by fastforce
hence 14:  $\vdash \text{finite} \longrightarrow \neg ((\text{halt } (\text{init } w)); f \wedge (\Box (\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)))$ 
using NextLoop by blast
hence 15:  $\vdash (\text{halt } (\text{init } w)); f \wedge \text{finite} \longrightarrow \neg ((\Box (\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)))$ 
by auto
from 15 show ?thesis by (simp add: yields-d-def)
qed

```

6.12 Properties of Groups of chops

lemma *NestedChopImpChop*:

```

assumes  $\vdash \text{init } w \wedge f \longrightarrow g; (\text{init } w1 \wedge f1)$ 
 $\quad \vdash \text{init } w1 \wedge f1 \longrightarrow g1; (\text{init } w2 \wedge f2)$ 
shows  $\vdash \text{init } w \wedge f \longrightarrow g; (g1; (\text{init } w2 \wedge f2))$ 
proof –
have 1:  $\vdash \text{init } w \wedge f \longrightarrow g; (\text{init } w1 \wedge f1)$  using assms(1) by auto
have 2:  $\vdash \text{init } w1 \wedge f1 \longrightarrow g1; (\text{init } w2 \wedge f2)$  using assms(2) by auto
hence 3:  $\vdash g; (\text{init } w1 \wedge f1) \longrightarrow g; (g1; (\text{init } w2 \wedge f2))$  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)
 $(\text{metis AndChopA ChopAndA ChopAndFiniteDist Prop11 Prop12 inteq-reflection})$ 
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 OrSChopImp :
 $\vdash (f \vee g) \frown h \longrightarrow f \frown h \vee g \frown h$
unfolding schop-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 schop-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 schop-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 SChopstarEqv :
 $\vdash (\text{s chopstar } f) = (\text{empty} \vee (f \wedge \text{more}) \frown (\text{s chopstar } f))$
using SChopstarEqvSem Valid-def **by** blast

lemma AndMoreSChopEqvAndFmoreChop:
 $\vdash (f \wedge \text{more}) \frown g = (f \wedge \text{fmore}); g$
by (*simp add: LeftChopEqvChop AndMoreAndFiniteEqvAndFmore schop-d-def*)

lemma SOmegaUnroll:
 $\vdash f^\omega = (f \wedge \text{more}) \frown f^\omega$
using OmegaUnroll AndMoreSChopEqvAndFmoreChop **by** fastforce

lemma SOmegaInduct:

$\vdash (inf \wedge g \wedge \Box(g \longrightarrow (f \wedge more) \frown g)) \longrightarrow \omega f$
by (*metis AndMoreAndFiniteEqvAndFmore OmegaInduct int-eq schop-d-def*)

lemma *FiniteBfGen*:
assumes $\vdash finite \longrightarrow f$
shows $\vdash bf\ f$
using *assms*
by (*simp add: Valid-def itl-defs*)

lemma *BfGen*:
assumes $\vdash f$
shows $\vdash bf\ f$
using *assms*
by (*metis EmptyImpFinite FiniteAndEmptyEqvEmpty FiniteBfGen Prop09 int-eq-true inteq-reflection*)

7.2 ITL operators in terms of SChop

lemma *NextSChopdef*:
 $\vdash \bigcirc f = skip \frown f$
by (*metis FiniteChopSkipEqvSkipChopFinite NowImpDiamond Prop10 SkipChopFiniteImpFinite inteq-reflection lift-imp-trans next-d-def schop-d-def sometimes-d-def*)

lemma *DiamondSChopdef*:
 $\vdash \Diamond f = \#True \frown f$
by (*simp add: schop-d-def sometimes-d-def*)

lemma *FiniteSChopdef*:
 $\vdash finite = \Diamond empty$
by (*simp add: DiamondEmptyEqvFinite int-iffD1 int-iffD2 int-iffI*)

lemma *ChopSChopdef*:
 $\vdash f;g = ((f \frown g) \vee (f \wedge inf))$
by (*metis AndInfChopEqvAndInf OrChopEqv OrFiniteInf inteq-reflection schop-d-def*)

lemma *PowerSpowerdef*:
 $\vdash power\ f\ n = spower\ f\ n$
proof
 (*induct n*)
case 0
then show ?case **by** *auto*
next
case (*Suc n*)
then show ?case
by (*metis PowerCommute inteq-reflection pow-Suc schop-d-def spow-Suc*)
qed

lemma *SChopstarFPowerstardef*:
 $\vdash schopstar\ f = fpowerstar\ f$
proof –

have 1: $\vdash \text{schopstar } f = (\exists k. \text{spower } (f \wedge \text{more}) \ k)$
by (*simp add: schopstar-d-def spowerstar-d-def*)
have 2: $\vdash \text{fpowerstar } f = \text{fpowerstar } (f \wedge \text{more})$
using *FPSAndMoreEqvFPS* **by** *fastforce*
have 3: $\vdash \text{fpowerstar } (f \wedge \text{more}) = (\exists k. \text{power } (f \wedge \text{more}) \ k)$
by (*simp add: fpowerstar-d-def*)
have 4: $\bigwedge n. \vdash \text{spower } (f \wedge \text{more}) \ n = \text{power } (f \wedge \text{more}) \ n$
using *PowerSpowerdef* **by** *fastforce*
from 1 2 3 4 **show** *?thesis* **by** *fastforce*
qed

lemma *SFinprop* :

$\vdash ((\# \text{True} \neg (f \wedge \text{empty})) \wedge (\# \text{True} \neg (g \wedge \text{empty}))) = (\# \text{True} \neg ((f \wedge g) \wedge \text{empty}))$
 $\vdash ((\# \text{True} \neg (f \wedge \text{empty})) \vee (\# \text{True} \neg (g \wedge \text{empty}))) = (\# \text{True} \neg ((f \vee g) \wedge \text{empty}))$
 $\vdash \text{finite} \longrightarrow (\neg (\# \text{True} \neg (f \wedge \text{empty}))) = (\# \text{True} \neg (\neg f \wedge \text{empty}))$
 $\vdash (\neg (\# \text{True} \neg (f \wedge \text{empty}))) = ((\# \text{True} \neg (\neg f \wedge \text{empty})) \vee \text{inf})$

by (*auto simp add: Valid-def itl-defs*)

(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* :

$\vdash \text{bf } (f \longrightarrow f1) \longrightarrow f \neg g \longrightarrow f1 \neg g$

proof –

have 1: $\vdash g \longrightarrow g$ **by** *auto*

hence 2: $\vdash \square (g \longrightarrow g)$ **by** (*rule BoxGen*)

have 3: $\vdash \text{bf } (f \longrightarrow f1) \wedge \square (g \longrightarrow g) \longrightarrow f \neg g \longrightarrow f1 \neg g$ **by** (*rule BfBoxSCHopImpSCHop*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *BiImpBf*:

$\vdash \text{bi } f \longrightarrow \text{bf } f$

unfolding *bi-d-def bf-d-def di-d-def df-d-def schop-d-def*

by (*simp add: AndChopA*)

lemma *BiSCHopImpSCHop* :

$\vdash \text{bi } (f \longrightarrow f1) \longrightarrow f \neg g \longrightarrow f1 \neg g$

proof –

have 1: $\vdash g \longrightarrow g$

by *auto*

hence 2: $\vdash \square (g \longrightarrow g)$

by (*rule BoxGen*)

have 3: $\vdash \text{bi } (f \longrightarrow f1) \wedge \square (g \longrightarrow g) \longrightarrow f \neg g \longrightarrow f1 \neg g$

using *BiImpBf BfBoxSChopImpSChop* using *BfSChopImpSChop* by *fastforce*
 from 2 3 show *?thesis* by *fastforce*
 qed

lemma *AndSChopA*:

⊢ $(f \wedge f1) \frown g \longrightarrow f \frown g$
 proof –
 have 1: ⊢ $f \wedge f1 \longrightarrow f$ by *auto*
 hence 2: ⊢ $\text{bf } (f \wedge f1 \longrightarrow f)$ by (rule *BfGen*)
 have 3: ⊢ $\text{bf } (f \wedge f1 \longrightarrow f) \longrightarrow (f \wedge f1) \frown g \longrightarrow f \frown g$ by (rule *BfSChopImpSChop*)
 from 2 3 show *?thesis* using *MP* by *blast*
 qed

lemma *AndSChopB*:

⊢ $(f \wedge f1) \frown g \longrightarrow f1 \frown g$
 proof –
 have 1: ⊢ $f \wedge f1 \longrightarrow f1$ by *auto*
 hence 2: ⊢ $\text{bf } (f \wedge f1 \longrightarrow f1)$ by (rule *BfGen*)
 have 3: ⊢ $\text{bf } (f \wedge f1 \longrightarrow f1) \longrightarrow (f \wedge f1) \frown g \longrightarrow f1 \frown g$ by (rule *BfSChopImpSChop*)
 from 2 3 show *?thesis* using *MP* by *blast*
 qed

lemma *NextSChop*:

⊢ $(\bigcirc f) \frown g = \bigcirc(f \frown g)$
 proof –
 have 1: ⊢ $\text{skip} \frown (f \frown g) = (\text{skip} \frown f) \frown g$ by (rule *SChopAssoc*)
 from 1 show *?thesis* using *NextSChopdef* by (metis *inteq-reflection*)
 qed

lemma *BoxSChopImpSChop* :

⊢ $\Box (g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$
 proof –
 have 1: ⊢ $g \longrightarrow g$ by *auto*
 hence 2: ⊢ $\text{bf } (g \longrightarrow g)$ by (rule *BfGen*)
 have 3: ⊢ $\text{bf } (f \longrightarrow f) \wedge \Box (g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ by (rule *BfBoxSChopImpSChop*)
 from 2 3 show *?thesis* by *fastforce*
 qed

lemma *LeftSChopImpSChop*:

assumes ⊢ $f \longrightarrow f1$
 shows ⊢ $f \frown g \longrightarrow f1 \frown g$
 proof –
 have 1: ⊢ $f \longrightarrow f1$ using *assms* by *auto*
 hence 2: ⊢ $\text{bf } (f \longrightarrow f1)$ by (rule *BfGen*)
 have 3: ⊢ $\text{bf } (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ by (rule *BfSChopImpSChop*)

from 2 3 show ?thesis using MP by blast
qed

lemma *RightSChopImpSChop*:

assumes $\vdash g \longrightarrow g1$

shows $\vdash f \frown g \longrightarrow f \frown g1$

proof –

have 1: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*

hence 2: $\vdash \Box (g \longrightarrow g1)$ **by** (*rule BoxGen*)

have 3: $\vdash \Box (g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **by** (*rule BoxSChopImpSChop*)

from 2 3 **show** ?thesis **using** MP **by** blast

qed

lemma *RightSChopEqvSChop*:

assumes $\vdash g = g1$

shows $\vdash (f \frown g) = (f \frown g1)$

proof –

have 1: $\vdash g = g1$ **using** *assms* **by** *auto*

have 2: $(\vdash g \longrightarrow g1) \implies (\vdash f \frown g \longrightarrow f \frown g1)$ **by** (*rule RightSChopImpSChop*)

have 3: $(\vdash g1 \longrightarrow g) \implies (\vdash f \frown g1 \longrightarrow f \frown g)$ **by** (*rule RightSChopImpSChop*)

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

qed

lemma *BoxRightSChopEqvSChop*:

$\vdash \Box (g = g1) \longrightarrow (f \frown g) = (f \frown g1)$

proof –

have 0: $\vdash (g = g1) = (\vdash (g \longrightarrow g1) \wedge (\vdash g1 \longrightarrow g))$

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 *AndSChopCommute* :
 $\vdash (f \wedge f1) \frown g = (f1 \wedge f) \frown g$
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: $\vdash \Box(\text{empty} \longrightarrow \#True)$ **by** (rule *BoxGen*)
 have 4: $\vdash \Box(\text{empty} \longrightarrow \#True) \longrightarrow (f; \text{empty} \longrightarrow f; \#True)$ **by** (rule *BoxChopImpChop*)
 have 5: $\vdash f \frown \text{empty} \longrightarrow f \frown \#True$ **using** 3 4 **MP** **by** (simp add: *RightSChopImpSChop*)
 hence 6: $\vdash f \frown \text{empty} \longrightarrow df\ f$ **by** (simp add: *df-d-def*)
 from 1 6 **show** ?thesis **using** *AndFiniteImpDf* **by** blast
 qed

lemma *BfElim*:

$\vdash bf\ f \wedge finite \longrightarrow f$

proof –

have 1: $\vdash \neg f \wedge finite \longrightarrow df\ (\neg f)$ **by** (rule *DfIntro*)
 have 2: $\vdash (\neg f \wedge finite \longrightarrow df\ (\neg f)) \longrightarrow (\neg(df\ (\neg f)) \longrightarrow \neg(\neg f \wedge finite))$ **by** simp
 have 21: $\vdash \neg(\neg f \wedge finite) = (f \vee inf)$ **by** (simp add: *Valid-def finite-d-def*)
 have 3: $\vdash \neg(df\ (\neg f)) \longrightarrow f \vee inf$ **using** 1 2 21 **by** fastforce
 from 3 **show** ?thesis **by** (simp add: *Prop13 bf-d-def finite-d-def*)
 qed

lemma *BfContraPosImpDist*:

$\vdash bf\ (\neg g \longrightarrow \neg f) \longrightarrow (bf\ f) \longrightarrow (bf\ g)$

proof –

have 1: $\vdash bf\ (\neg g \longrightarrow \neg f) \longrightarrow (df\ (\neg g)) \longrightarrow (df\ (\neg f))$ **by** (rule *BfImpDfImpDf*)
 hence 2: $\vdash bf\ (\neg g \longrightarrow \neg f) \longrightarrow (\neg(df\ (\neg f))) \longrightarrow (\neg(df\ (\neg g)))$ **by** auto
 from 2 **show** ?thesis **by** (metis *bf-d-def*)
 qed

lemma *BfImpDist*:

$\vdash bf\ (f \longrightarrow g) \longrightarrow (bf\ f) \longrightarrow (bf\ g)$

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: $\vdash bf\ (\neg(\neg\ g \longrightarrow \neg\ f) \longrightarrow \neg(f \longrightarrow g))$ **by** (rule *BfGen*)
 have 4: $\vdash bf\ (\neg(\neg\ g \longrightarrow \neg\ f) \longrightarrow \neg(f \longrightarrow g))$
 \longrightarrow
 $bf\ (f \longrightarrow g) \longrightarrow bf\ (\neg\ g \longrightarrow \neg\ f)$ **by** (rule *BfContraPosImpDist*)
 have 5: $\vdash bf\ (f \longrightarrow g) \longrightarrow bf\ (\neg\ g \longrightarrow \neg\ f)$ **using** 3 4 **MP** **by** blast
 have 6: $\vdash bf\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow (bf\ f) \longrightarrow (bf\ g)$ **by** (rule *BfContraPosImpDist*)
 from 5 6 **show** ?thesis **using** *lift-imp-trans* **by** blast
 qed

lemma *FiniteImpBfImpBfRule*:

assumes $\vdash finite \longrightarrow (f \longrightarrow g)$

shows $\vdash bf\ f \longrightarrow bf\ g$

proof –

have 1: $\vdash finite \longrightarrow f \longrightarrow g$ **using** *assms* **by** auto
 have 2: $\vdash bf(f \longrightarrow g)$ **using** 1 **by** (simp add: *FiniteBfGen*)
 have 3: $\vdash bf(f \longrightarrow g) \longrightarrow bf\ f \longrightarrow bf\ g$ **using** *BfImpDist* **by** blast
 from 2 3 **show** ?thesis **by** fastforce

qed

lemma *FiniteImpBfEqvRule*:

assumes $\vdash \text{finite} \longrightarrow (f = g)$

shows $\vdash \text{bf } f = \text{bf } g$

proof –

have 1: $\vdash \text{finite} \longrightarrow (f = g)$ **using** *assms* **by** *blast*

have 2: $\vdash \text{finite} \longrightarrow (f \longrightarrow g)$ **using** 1 **by** *auto*

have 3: $\vdash \text{bf } f \longrightarrow \text{bf } g$ **by** (*simp add: 2 FiniteImpBfImpBfRule*)

have 4: $\vdash \text{finite} \longrightarrow (g \longrightarrow f)$ **using** 1 **by** *auto*

have 5: $\vdash \text{bf } g \longrightarrow \text{bf } f$ **by** (*simp add: 4 FiniteImpBfImpBfRule*)

from 3 5 **show** *?thesis* **by** *fastforce*

qed

lemma *IfSCHopEqvRule*:

assumes $\vdash f = \text{if}_i (\text{init } w) \text{ then } f1 \text{ else } f2$

shows $\vdash f \frown g = \text{if}_i (\text{init } w) \text{ then } (f1 \frown g) \text{ else } (f2 \frown g)$

proof –

have 1: $\vdash f = \text{if}_i (\text{init } w) \text{ then } f1 \text{ else } f2$

using *assms* **by** *auto*

hence 2: $\vdash f = (((\text{init } w) \wedge f1) \vee ((\text{init } (\neg w)) \wedge f2))$

unfolding *ifthenelse-d-def* **by** (*metis Initprop(2) int-eq*)

hence 3: $\vdash f \frown g = (((\text{init } w) \wedge f1) \frown g \vee ((\text{init } (\neg w)) \wedge f2) \frown g)$

by (*rule OrSCHopEqvRule*)

have 4: $\vdash ((\text{init } w) \wedge f1) \frown g = ((\text{init } w) \wedge (f1 \frown g))$

by (*rule StateAndSCHop*)

have 5: $\vdash ((\text{init } (\neg w)) \wedge f2) \frown g = ((\text{init } (\neg w)) \wedge (f2 \frown g))$

by (*rule StateAndSCHop*)

have 6: $\vdash f \frown g = (((\text{init } w) \wedge f1 \frown g) \vee ((\text{init } (\neg w)) \wedge f2 \frown g))$

using 3 4 5 **by** *fastforce*

from 6 **show** *?thesis* **unfolding** *ifthenelse-d-def* **by** (*metis Initprop(2) 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*:

$\vdash (\text{empty} \vee f) \frown g = (g \vee (f \frown g))$

proof –

have 1: $\vdash (\text{empty} \vee f) \frown g = ((\text{empty} \frown g) \vee (f \frown g))$ **by** (*rule OrSCHopEqv*)

have 2: $\vdash \text{empty} \frown g = g$ **by** (*rule EmptySCHop*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *EmptyOrNextSChopEqv*:

$\vdash (\text{empty} \vee \circ f) \frown g = (g \vee \circ(f \frown g))$

proof –

have 1: $\vdash (\text{empty} \vee \circ f) \frown g = (g \vee ((\circ f) \frown g))$ **by** (rule *EmptyOrSChopEqv*)

have 2: $\vdash (\circ f) \frown g = \circ(f \frown g)$ **by** (rule *NextSChop*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *EmptyOrSChopImpRule*:

assumes $\vdash f \longrightarrow \text{empty} \vee f1$

shows $\vdash f \frown g \longrightarrow g \vee (f1 \frown g)$

proof –

have 1: $\vdash f \longrightarrow \text{empty} \vee f1$ **using** assms **by** auto

hence 2: $\vdash f \frown g \longrightarrow (\text{empty} \vee f1) \frown g$ **by** (rule *LeftSChopImpSChop*)

have 3: $\vdash (\text{empty} \vee f1) \frown g = (g \vee (f1 \frown g))$ **by** (rule *EmptyOrSChopEqv*)

from 2 3 **show** ?thesis **by** fastforce

qed

lemma *EmptyOrSChopEqvRule*:

assumes $\vdash f = (\text{empty} \vee f1)$

shows $\vdash f \frown g = (g \vee (f1 \frown g))$

proof –

have 1: $\vdash f = (\text{empty} \vee f1)$ **using** assms **by** auto

hence 2: $\vdash f \frown g = ((\text{empty} \vee f1) \frown g)$ **by** (rule *LeftSChopEqvSChop*)

have 3: $\vdash (\text{empty} \vee f1) \frown g = (g \vee (f1 \frown g))$ **by** (rule *EmptyOrSChopEqv*)

from 2 3 **show** ?thesis **by** fastforce

qed

lemma *EmptyOrNextSChopImpRule*:

assumes $\vdash f \longrightarrow \text{empty} \vee \circ f1$

shows $\vdash f \frown g \longrightarrow g \vee \circ(f1 \frown g)$

proof –

have 1: $\vdash f \longrightarrow \text{empty} \vee \circ f1$ **using** assms **by** auto

hence 2: $\vdash f \frown g \longrightarrow (\text{empty} \vee \circ f1) \frown g$ **by** (rule *LeftSChopImpSChop*)

have 3: $\vdash (\text{empty} \vee \circ f1) \frown g = (g \vee \circ(f1 \frown g))$ **by** (rule *EmptyOrNextSChopEqv*)

from 2 3 **show** ?thesis **by** fastforce

qed

lemma *EmptyOrNextSChopEqvRule*:

assumes $\vdash f = (\text{empty} \vee \circ f1)$

shows $\vdash f \frown g = (g \vee \circ(f1 \frown g))$

proof –

have 1: $\vdash f = (\text{empty} \vee \circ f1)$ **using** assms **by** auto

hence 2: $\vdash f \frown g = ((\text{empty} \vee \circ f1) \frown g)$ **by** (rule *LeftSChopEqvSChop*)

have 3: $\vdash (\text{empty} \vee \circ f1) \frown g = (g \vee \circ(f1 \frown g))$ **by** (rule *EmptyOrNextSChopEqv*)

from 2 3 **show** ?thesis **by** fastforce

qed

lemma *SChopEmptyOrImpRule*:

assumes $\vdash g \longrightarrow \text{empty} \vee g1$

shows $\vdash f \frown g \wedge \text{finite} \longrightarrow f \vee (f \frown g1)$

proof –

have 1: $\vdash g \longrightarrow \text{empty} \vee g1$ **using** *assms* **by** *auto*

hence 2: $\vdash f \frown g \longrightarrow (f \frown \text{empty}) \vee (f \frown g1)$ **by** (*rule SChopOrImpRule*)

have 3: $\vdash \text{finite} \longrightarrow f \frown \text{empty} = f$ **by** (*rule SChopEmpty*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *BoxStateSChopBoxAndInfImpBox*:

$\vdash \Box (\text{init } w) \frown \Box (\text{init } w) \wedge \text{inf} \longrightarrow \Box (\text{init } w)$

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: $\vdash ((init\ w) \wedge empty) \frown \Box (init\ w) \longrightarrow \Box (init\ w) \frown \Box (init\ w)$
 by (rule *LeftSChopImpSChop*)
 have 17: $\vdash \Box (init\ w) \longrightarrow \Box (init\ w) \frown \Box (init\ w)$
 using 14 16 by *fastforce*
 from 10 17 show ?thesis by *fastforce*
 qed

lemma *NotBoxStateImpBoxSYieldsNotBox*:

$\vdash \neg(\Box (init\ w)) \longrightarrow (\Box (init\ w))\ syields\ (\neg(\Box (init\ w)))$
proof –
 have 1: $\vdash \Box (init\ w) \frown \Box (init\ w) = \Box (init\ w)$ by (rule *BoxStateSChopBoxEqvBox*)
 have 2: $\vdash \Box (init\ w) = (\neg \neg(\Box (init\ w)))$ by *auto*
 hence 3: $\vdash \Box (init\ w) \frown \Box (init\ w) = \Box (init\ w) \frown (\neg \neg(\Box (init\ w)))$ by (rule *RightSChopEqvSChop*)
 have 4: $\vdash \neg(\Box (init\ w)) \longrightarrow \neg(\Box (init\ w) \frown (\neg \neg(\Box (init\ w))))$ using 1 3 by *auto*
 from 4 show ?thesis by (simp add: *syields-d-def*)
 qed

lemma *StateEqvBf*:

$\vdash (init\ w) = bf\ (init\ w)$
proof –
 have 1: $\vdash (init\ w) \longrightarrow bf\ (init\ w)$ by (rule *StateImpBf*)
 have 2: $\vdash bf\ (init\ w) \wedge finite \longrightarrow (init\ w)$ by (rule *BfElim*)
 from 1 2 show ?thesis
 by (metis (no-types) *DfState Initprop(2) bf-d-def int-simps(4) inteq-reflection*)
 qed

lemma *TrueSChopEqvDiamond*:

$\vdash \#True \frown f = \Diamond f$
 using *DiamondSChopdef* by *fastforce*

lemma *BfAndEqvBfAndBf*:

$\vdash bf(f \wedge g) = (bf\ f \wedge bf\ g)$
proof –
 have 1: $\vdash f \wedge g \longrightarrow f$ by *auto*
 have 2: $\vdash bf(f \wedge g) \longrightarrow bf\ f$ by (simp add: 1 *BfImpBfRule*)
 have 3: $\vdash f \wedge g \longrightarrow g$ by *auto*
 have 4: $\vdash bf(f \wedge g) \longrightarrow bf\ g$ by (simp add: 3 *BfImpBfRule*)
 have 5: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ by *auto*
 have 6: $\vdash bf\ f \longrightarrow bf\ (g \longrightarrow f \wedge g)$ by (simp add: 5 *BfImpBfRule*)
 have 7: $\vdash bf\ (g \longrightarrow f \wedge g) \longrightarrow (bf\ g \longrightarrow bf(f \wedge g))$ by (simp add: *BfImpDist*)
 have 8: $\vdash bf\ f \wedge bf\ g \longrightarrow bf\ (f \wedge g)$ using 6 7 by *fastforce*
 from 2 4 8 show ?thesis by *fastforce*
 qed

lemma *BfEqvBfImpAndBfImp*:

$\vdash bf(f = g) = (bf\ (f \longrightarrow g) \wedge bf(g \longrightarrow f))$

```

proof -
  have 1:  $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$  by auto
  have 2:  $\vdash \text{bf}(f = g) = \text{bf}((f \longrightarrow g) \wedge (g \longrightarrow f))$  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*:

```

 $\vdash \text{bf}(f = f1) \longrightarrow f \frown g = f1 \frown g$ 
proof -
  have 1:  $\vdash \text{bf}(f = f1) = (\text{bf}(f \longrightarrow f1) \wedge \text{bf}(f1 \longrightarrow f))$  by (simp add: BfEqvBfImpAndBfImp)
  have 2:  $\vdash \text{bf}(f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$  by (simp add: BfSChopImpSChop)
  have 3:  $\vdash \text{bf}(f1 \longrightarrow f) \longrightarrow f1 \frown g \longrightarrow f \frown g$  by (simp add: BfSChopImpSChop)
  from 1 2 3 show ?thesis by fastforce
qed

```

lemma *BfEqvDfEqvDf*:

```

 $\vdash \text{bf}(f = g) \longrightarrow (\text{df } f = \text{df } g)$ 
proof -
  have 1:  $\vdash \text{bf}(f = g) \longrightarrow (f \frown \#True) = (g \frown \#True)$ 
    using BfEqvImpSChopEqvSChop by fastforce
  from 1 show ?thesis by (simp add: df-d-def)
qed

```

lemma *FiniteImpEqvDfImpRule*:

```

assumes  $\vdash \text{finite} \longrightarrow f = g$ 
shows  $\vdash \text{df } f = \text{df } g$ 
proof -
  have 1:  $\vdash \text{finite} \longrightarrow f = g$  using assms by auto
  have 2:  $\vdash \text{bf}(f = g)$  using 1 by (simp add: FiniteBfGen)
  have 3:  $\vdash \text{bf}(f = g) \longrightarrow (\text{df } f = \text{df } g)$  by (simp add: BfEqvDfEqvDf)
  from 2 3 show ?thesis by fastforce
qed

```

lemma *DfEmpty*:

```

 $\vdash \text{df } \text{empty}$ 
proof -
  have 1:  $\vdash \#True$  by auto
  have 2:  $\vdash \text{empty} \frown \#True = \#True$  by (rule EmptySChop)
  have 3:  $\vdash \text{empty} \frown \#True$  using 1 2 by auto
  from 3 show ?thesis by (simp add: df-d-def)
qed

```

lemma *BfImpDf*:

```

 $\vdash \text{bf } f \longrightarrow \text{df } f$ 

```

```

proof –
  have 1:  $\vdash f \longrightarrow (\text{empty} \longrightarrow f)$  by auto
  have 2:  $\vdash \text{bf } f \longrightarrow \text{bf}(\text{empty} \longrightarrow f)$  by (simp add: 1 BfImpBfRule)
  have 3:  $\vdash \text{bf}(\text{empty} \longrightarrow f) \longrightarrow \text{df empty} \longrightarrow \text{df } f$  by (simp add: BfImpDfImpDf)
  have 4:  $\vdash \text{bf } f \longrightarrow \text{df empty} \longrightarrow \text{df } f$  using 2 3 lift-imp-trans by blast
  have 5:  $\vdash \text{df empty}$  by (simp add: DfEmpty)
  from 4 5 show ?thesis by fastforce
qed

```

7.5 Properties of SDa and SBa

```

lemma SDaEqvDtDf:
 $\vdash \text{sda } f = \Diamond (\text{df } f)$ 
proof –
  have 1:  $\vdash \# \text{True} \frown (f \frown \# \text{True}) = \# \text{True} \frown (f \frown \# \text{True})$  by auto
  hence 2:  $\vdash \# \text{True} \frown (f \frown \# \text{True}) = \# \text{True} \frown \text{df } f$  by (simp add: df-d-def)
  have 3:  $\vdash \# \text{True} \frown (\text{df } f) = \Diamond (\text{df } f)$  by (simp add: TrueSChopEqvDiamond)
  have 4:  $\vdash \# \text{True} \frown (f \frown \# \text{True}) = \Diamond (\text{df } f)$  using 2 3 by fastforce
  from 4 show ?thesis by (simp add: sda-d-def)
qed

```

```

lemma SDaEqvDfDt:
 $\vdash \text{sda } f = \text{df } (\Diamond f)$ 
proof –
  have 1:  $\vdash \# \text{True} \frown f = \Diamond f$  by (rule TrueSChopEqvDiamond)
  hence 2:  $\vdash (\# \text{True} \frown f) \frown \# \text{True} = (\Diamond f) \frown \# \text{True}$  by (rule LeftSChopEqvSChop)
  hence 3:  $\vdash (\# \text{True} \frown f) \frown \# \text{True} = \text{df } (\Diamond f)$  by (simp add: df-d-def)
  have 4:  $\vdash \# \text{True} \frown (f \frown \# \text{True}) = (\# \text{True} \frown f) \frown \# \text{True}$  by (rule SChopAssoc)
  have 5:  $\vdash \# \text{True} \frown (f \frown \# \text{True}) = \text{df } (\Diamond f)$  using 3 4 by fastforce
  from 5 show ?thesis by (simp add: sda-d-def)
qed

```

```

lemma DtDfEqvDfDt:
 $\vdash \Diamond (\text{df } f) = \text{df } (\Diamond f)$ 
by (meson Prop04 SDaEqvDfDt SDaEqvDtDf)

```

```

lemma SBaEqvBfBt:
 $\vdash \text{sba } f = \text{bf } (\Box f)$ 
proof –
  have 1:  $\vdash \text{sda } (\neg f) = \text{df } (\Diamond (\neg f))$  by (rule SDaEqvDfDt)
  have 2:  $\vdash \Diamond (\neg f) = (\neg (\Box f))$  by (rule DiamondNotEqvNotBox)
  hence 3:  $\vdash \text{df } (\Diamond (\neg f)) = \text{df } (\neg (\Box f))$  by (rule DfEqvDf)
  have 4:  $\vdash \text{sda } (\neg f) = \text{df } (\neg (\Box f))$  using 1 3 by fastforce
  hence 5:  $\vdash (\neg (\text{sda } (\neg f))) = (\neg (\text{df } (\neg (\Box f))))$  by auto
  hence 6:  $\vdash (\neg (\text{sda } (\neg f))) = \text{bf } (\Box f)$  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*:
 $\vdash \text{ba } f \longrightarrow \text{bf } f$
proof –
have 1: $\vdash \text{ba } f = \Box(\text{bi } f)$ **by** (*rule BaEqvBtBi*)
have 2: $\vdash \Box(\text{bi } f) \longrightarrow \text{bi } f$ **by** (*rule BoxElim*)
have 3: $\vdash \text{bi } f \longrightarrow \text{bf } f$ **by** (*simp add: BiImpBf*)
from 1 2 3 **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *SBaImpBt*:
 $\vdash \text{sba } f \wedge \text{finite} \longrightarrow \Box f$
proof –
have 1: $\vdash \text{sba } f = \text{bf}(\Box f)$ **by** (*rule SBaEqvBfBt*)
have 2: $\vdash \text{bf}(\Box f) \wedge \text{finite} \longrightarrow \Box f$ **by** (*rule BfElim*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *DiamondImpSDa*:
 $\vdash \Diamond f \wedge \text{finite} \longrightarrow \text{sda } f$
by (*metis AndFiniteImpDf SDaEqvDfDt inteq-reflection*)

lemma *DfImpSDa*:
 $\vdash \text{df } f \longrightarrow \text{sda } f$
using *NowImpDiamond SDaEqvDtDf* **by** *fastforce*

lemma *BoxAndSChopImport*:
 $\vdash \Box h \wedge f \frown g \longrightarrow f \frown (h \wedge g)$
proof –
have 1: $\vdash h \longrightarrow g \longrightarrow (h \wedge g)$ **by** *auto*
hence 2: $\vdash \Box h \longrightarrow \Box(g \longrightarrow (h \wedge g))$ **by** (*rule ImpBoxRule*)
have 3: $\vdash \Box(g \longrightarrow (h \wedge g)) \longrightarrow f \frown g \longrightarrow f \frown (h \wedge g)$ **by** (*rule BoxSChopImpSChop*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *SBaAndSChopImport*:
 $\vdash \text{sba } f \wedge \text{finite} \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown (f \wedge g1)$
proof –
have 1: $\vdash \text{sba } f \longrightarrow \text{bf } f$ **by** (*rule SBaImpBf*)
have 2: $\vdash \text{bf } f \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown g1$ **by** (*rule BfAndSChopImport*)
have 3: $\vdash \text{sba } f \wedge \text{finite} \longrightarrow \Box f$ **by** (*rule SBaImpBt*)
have 4: $\vdash \Box f \wedge (f \wedge g) \frown g1 \longrightarrow (f \wedge g) \frown (f \wedge g1)$ **by** (*rule BoxAndSChopImport*)
from 1 2 3 4 **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndSChopImport*:
 $\vdash \text{ba } f \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown (f \wedge g1)$
proof –
have 1: $\vdash \text{ba } f \longrightarrow \text{bi } f$ **by** (*rule BaImpBi*)
have 2: $\vdash \text{bi } f \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown g1$ **by** (*rule BiAndSChopImport*)

have 3: $\vdash ba\ f \longrightarrow \Box f$ **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*:

$\vdash (\neg (f \frown g)) = f \text{ yields } (\neg g)$

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: yields-d-def)

qed

lemma *NotDfFalse*:

$\vdash \neg (df \# False)$

proof –

have 1: $\vdash (init \# True) \longrightarrow bf (init \# True)$ **by** (rule *StateImpBf*)

hence 2: $\vdash \# True \longrightarrow bf \# True$ **by** (simp add: BfGen)

have 3: $\vdash \# True$ **by** auto

have 4: $\vdash bf \# True$ **using** 2 3 **MP** **by** auto

hence 5: $\vdash \neg (df (\neg \# True))$ **by** (simp add: bf-d-def)

have 6: $\vdash (\neg \# True) = \# False$ **by** auto

hence 7: $\vdash df (\neg \# True) = df \# False$ **by** (rule *DfEqvDf*)

from 5 7 **show** ?thesis **by** auto

qed

lemma *StateAndEmptySChop*:

$\vdash ((init\ w) \wedge empty) \frown f = ((init\ w) \wedge f)$

proof –

have 1: $\vdash ((init\ w) \wedge empty) \frown f = ((init\ w) \wedge empty \frown f)$ **by** (rule *StateAndSChop*)

have 2: $\vdash empty \frown f = f$ **by** (rule *EmptySChop*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *StateAndNextSChop*:

$\vdash ((init\ w) \wedge \bigcirc f) \frown g = ((init\ w) \wedge \bigcirc(f \frown g))$

proof –

have 1: $\vdash ((init\ w) \wedge \bigcirc f) \frown g = ((init\ w) \wedge (\bigcirc f) \frown g)$ **by** (rule *StateAndSChop*)

have 2: $\vdash (\bigcirc f) \frown g = \bigcirc(f \frown g)$ **by** (rule *NextSChop*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *NextStateAndSChop*:

$\vdash \circ(((init\ w) \wedge f) \frown g) = (\circ (init\ w) \wedge \circ(f \frown g))$
proof –
have 1: $\vdash ((init\ w) \wedge f) \frown g = ((init\ w) \wedge f \frown g)$ **by** (rule *StateAndSChop*)
hence 2: $\vdash \circ(((init\ w) \wedge f) \frown g) = \circ((init\ w) \wedge f \frown g)$ **by** (rule *NextEqvNext*)
have 3: $\vdash \circ((init\ w) \wedge f \frown g) = (\circ (init\ w) \wedge \circ(f \frown g))$ **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*:

$\vdash df(\circ f) = \circ(df\ f)$

proof –

have 1: $\vdash (\circ f) \frown \#True = \circ(f \frown \#True)$ **by** (rule *NextSChop*)
from 1 **show** ?thesis **by** (simp add: df-d-def)

qed

lemma *DfNextState*:

$\vdash df(\circ (init\ w)) = \circ (init\ w)$

proof –

have 1: $\vdash df(\circ (init\ w)) = \circ(df\ (init\ w))$ **by** (rule *DfNext*)
have 2: $\vdash df\ (init\ w) = (init\ w)$ **by** (rule *DfState*)
hence 3: $\vdash \circ(df\ (init\ w)) = \circ (init\ w)$ **by** (rule *NextEqvNext*)
from 1 3 **show** ?thesis **by** fastforce

qed

lemma *DfStateAndNextStateEqvStateAndNextState*:

$\vdash df(init\ w \wedge \circ(init\ w1)) = (init\ w \wedge \circ(init\ w1))$

proof –

have 1: $\vdash (init\ w \wedge \circ(init\ w1)) \frown \#True = (init\ w \wedge \circ((init\ w1) \frown \#True))$
using *StateAndNextSChop* **by** blast
have 2: $\vdash df(init\ w \wedge \circ(init\ w1)) = (init\ w \wedge \circ((init\ w1) \frown \#True))$
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: $\vdash \circ(df(init\ w1)) = \circ(init\ w1)$
 by (simp add: 3 NextEqvNext)
 from 2 5 show ?thesis by (metis df-d-def int-eq)
 qed

lemma *StateImpBfGen*:

assumes $\vdash (init\ w) \longrightarrow f$
 shows $\vdash (init\ w) \longrightarrow bf\ f$
 proof –
 have 1: $\vdash (init\ w) \longrightarrow f$ using assms by auto
 hence 2: $\vdash \neg f \longrightarrow \neg (init\ w)$ by auto
 hence 3: $\vdash df(\neg f) \longrightarrow df(\neg (init\ w))$ by (rule DfImpDf)
 hence 4: $\vdash df(\neg f) \longrightarrow df(init(\neg w))$ by (metis Initprop(2) inteq-reflection)
 have 5: $\vdash df(init(\neg w)) = (init(\neg w))$ by (rule DfState)
 have 6: $\vdash df(\neg f) \longrightarrow \neg (init\ w)$ using 4 5 using Initprop(2) by fastforce
 hence 7: $\vdash (init\ w) \longrightarrow \neg(df(\neg f))$ by auto
 from 7 show ?thesis by (simp add: bf-d-def)
 qed

lemma *SChopAndNotSChopImp*:

$\vdash f \frown g \wedge \neg(f \frown g1) \longrightarrow f \frown (g \wedge \neg g1)$
 proof –
 have 1: $\vdash g \longrightarrow (g \wedge \neg g1) \vee g1$ by auto
 hence 2: $\vdash f \frown g \longrightarrow f \frown ((g \wedge \neg g1) \vee g1)$ by (rule RightSChopImpSChop)
 have 3: $\vdash f \frown ((g \wedge \neg g1) \vee g1) \longrightarrow (f \frown (g \wedge \neg g1)) \vee (f \frown g1)$ by (rule SChopOrImp)
 have 4: $\vdash f \frown g \longrightarrow f \frown (g \wedge \neg g1) \vee f \frown g1$ using 2 3 MP by fastforce
 from 4 show ?thesis by auto
 qed

lemma *SChopAndSYieldsImp*:

$\vdash f \frown g \wedge f\ syields\ g1 \longrightarrow f \frown (g \wedge g1)$
 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*:

$\vdash f \frown g \wedge f\ syields\ (g \longrightarrow g1) \longrightarrow f \frown g1$
 proof –
 have 1: $\vdash f \frown g \wedge f\ syields\ (g \longrightarrow g1) \longrightarrow f \frown (g \wedge (g \longrightarrow g1))$ 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*:

$\vdash f \frown (g \wedge \text{sf}in\ w) \longrightarrow \text{sf}in\ w$

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 min.absorb1*)

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*)

$\text{inteq-reflection lift-and-com})$
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: $\vdash (\text{sf}in\ g \wedge \text{empty}) = (g \wedge \text{empty})$
using *SFinAndEmpty* **by** *metis*
have 5: $\vdash (\text{empty} \wedge \text{sf}in\ g) = (g \wedge \text{empty})$
using 3 4 **by** *auto*
hence 6: $\vdash f \frown (\text{empty} \wedge \text{sf}in\ g) = f \frown (g \wedge \text{empty})$
using *RightSChopEqvSChop* **by** *blast*
from 1 2 5 **show** *?thesis* **by** (*metis inteq-reflection lift-and-com*)
qed

lemma *AndSFinEqvSChopStateAndEmpty*:
 $\vdash ((f \wedge \text{finite}) \wedge \text{sf}in\ (\text{init}\ w)) = f \frown ((\text{init}\ w) \wedge \text{empty})$
using *AndSFinEqvSChopAndEmpty* **by** *blast*

lemma *DiamondEqvEmptyOrNextDiamond*:
 $\vdash \Diamond f = (f \vee \bigcirc(\Diamond f))$
proof –
have 1: $\vdash \Box(\neg f) = ((\neg f) \wedge \text{wnext}(\Box(\neg f)))$
by (*simp add: BoxEqvAndWnextBox*)
have 2: $\vdash (\neg \Diamond f) = ((\neg f) \wedge \text{wnext}(\Box(\neg f)))$
using 1 **by** (*simp add: always-d-def*)
have 3: $\vdash \Diamond f = (f \vee \neg(\text{wnext}(\Box(\neg f))))$
using 2 **by** *auto*
have 4: $\vdash (\neg(\text{wnext}(\Box(\neg f)))) = \bigcirc(\neg\Box(\neg f))$
by (*simp add: wnext-d-def*)
have 5: $\vdash \neg\Box(\neg f) = \Diamond f$
by (*simp add: always-d-def*)
have 6: $\vdash \bigcirc(\neg\Box(\neg f)) = \bigcirc(\Diamond f)$
using 5 **using** *inteq-reflection* **by** *force*
from 3 4 6 **show** *?thesis* **by** *fastforce*
qed

lemma *SFinStateEqvStateAndEmptyOrNextSFinState*:
 $\vdash \text{sf}in\ (\text{init}\ w) = (((\text{init}\ w) \wedge \text{empty}) \vee \bigcirc(\text{sf}in\ (\text{init}\ w)))$
proof –
have 01: $\vdash \text{sf}in\ (\text{init}\ w) = \# \text{True} \frown ((\text{init}\ w) \wedge \text{empty})$
by (*simp add: SFinEqvTrueSChopAndEmpty*)
have 02: $\vdash \# \text{True} \frown ((\text{init}\ w) \wedge \text{empty}) = \Diamond((\text{init}\ w) \wedge \text{empty})$
by (*simp add: TrueSChopEqvDiamond*)
have 03: $\vdash \Diamond((\text{init}\ w) \wedge \text{empty}) = (((\text{init}\ w) \wedge \text{empty}) \vee \bigcirc(\text{sf}in\ (\text{init}\ w)))$
using *DiamondEqvEmptyOrNextDiamond* 02 01 **by** (*metis inteq-reflection*)
from 01 02 03 **show** *?thesis* **by** *fastforce*
qed

lemma *SFinSChopEqvOr*:
 $\vdash (\text{sf}in\ (\text{init}\ w)) \frown f = (((\text{init}\ w) \wedge f) \vee \bigcirc((\text{sf}in\ (\text{init}\ w)) \frown f))$

proof –

have 1: $\vdash \text{sfm } (\text{init } w) = (((\text{init } w) \wedge \text{empty}) \vee \bigcirc (\text{sfm } (\text{init } w)))$
by (rule *SFinStateEqvStateAndEmptyOrNextSFinState*)
hence 2: $\vdash (\text{sfm } (\text{init } w)) \frown f = (((\text{init } w) \wedge \text{empty}) \vee \bigcirc (\text{sfm } (\text{init } w))) \frown f$
by (rule *LeftSChopEqvSChop*)
have 3: $\vdash (((\text{init } w) \wedge \text{empty}) \vee \bigcirc (\text{sfm } (\text{init } w))) \frown f$
 $= (((\text{init } w) \wedge \text{empty}) \frown f \vee (\bigcirc (\text{sfm } (\text{init } w))) \frown f)$
by (rule *OrSChopEqv*)
have 4: $\vdash ((\text{init } w) \wedge \text{empty}) \frown f = ((\text{init } w) \wedge f)$
by (rule *StateAndEmptySChop*)
have 5: $\vdash (\bigcirc (\text{sfm } (\text{init } w))) \frown f = \bigcirc ((\text{sfm } (\text{init } w)) \frown f)$
by (rule *NextSChop*)
from 2 3 4 5 **show** ?thesis **by** fastforce
qed

lemma *SFinSChopEqvDiamond*:

$\vdash (\text{sfm } (\text{init } w)) \frown f = \Diamond ((\text{init } w) \wedge f)$

proof –

have 1: $\vdash (\text{sfm } (\text{init } w)) = (\# \text{True} \frown ((\text{init } w) \wedge \text{empty}))$
by (simp add: *SFinEqvTrueSChopAndEmpty*)
hence 2: $\vdash (\text{sfm } (\text{init } w)) \frown f = (\# \text{True} \frown ((\text{init } w) \wedge \text{empty})) \frown f$
by (rule *LeftSChopEqvSChop*)
have 3: $\vdash \# \text{True} \frown ((\text{init } w) \wedge \text{empty}) \frown f = (\# \text{True} \frown ((\text{init } w) \wedge \text{empty})) \frown f$
by (rule *SChopAssoc*)
have 4: $\vdash \# \text{True} \frown ((\text{init } w) \wedge \text{empty}) \frown f = \Diamond ((\text{init } w) \wedge \text{empty}) \frown f$
using *TrueSChopEqvDiamond* **by** blast
have 5: $\vdash ((\text{init } w) \wedge \text{empty}) \frown f = ((\text{init } w) \wedge f)$
using *StateAndEmptySChop* **by** blast
hence 6: $\vdash \Diamond ((\text{init } w) \wedge \text{empty}) \frown f = \Diamond ((\text{init } w) \wedge f)$
by (rule *DiamondEqvDiamond*)
from 2 3 4 6 **show** ?thesis **by** fastforce
qed

lemma *SFinSYields*:

$\vdash (\text{sfm } (\text{init } w)) \text{ syields } (\text{init } w)$

proof –

have 1: $\vdash (\text{sfm } (\text{init } w)) \frown (\neg(\text{init } w)) = \Diamond ((\text{init } w) \wedge \neg(\text{init } w))$
by (rule *SFinSChopEqvDiamond*)
have 2: $\vdash \neg(\Diamond ((\text{init } w) \wedge \neg(\text{init } w)))$
by (rule *NotDiamondAndNot*)
have 3: $\vdash \neg((\text{sfm } (\text{init } w)) \frown (\neg(\text{init } w)))$
using 1 2 **by** fastforce
from 3 **show** ?thesis **by** (simp add: *syields-d-def*)
qed

lemma *SFinEqvFinAndFinite*:

$\vdash (\text{finite} \wedge \text{fin } f) = \text{sfm } f$

by (metis *AndFinEqvChopAndEmpty DiamondSChopdef SFinEqvTrueSChopAndEmpty int-simps(20)*
inteq-reflection sometimes-d-def)

lemma *AndFiniteImpAndSFinStateOrSFinNotState*:
 $\vdash f \wedge \text{finite} \longrightarrow (f \wedge \text{sfin } (\text{init } w)) \vee (f \wedge \text{sfin } (\neg (\text{init } w)))$
proof –
have 1: $\vdash (\neg \text{fin } (\text{init } (\neg w)) \wedge \text{finite}) = (\text{fin } (\text{init } (\neg \neg w)) \wedge \text{finite})$
using *FinNotStateEqvNotFinState* **by** *blast*
have 2: $\vdash (\text{fin } (\text{init } (\neg \neg w)) \wedge \text{finite}) = (\text{fin } (\text{init } (w)) \wedge \text{finite})$
by *simp*
have 3: $\vdash f \wedge \text{finite} \longrightarrow (f \wedge \text{finite}) \wedge \text{fin } (\text{init } w) \vee (f \wedge \text{finite}) \wedge \text{fin } (\neg \text{init } w)$
by (*simp add: ImpAndFinStateOrFinNotState*)
have 4: $\vdash (\text{finite} \wedge \text{fin } (\text{init } w)) = \text{sfin}(\text{init } w)$
using *SFinEqvFinAndFinite*[of *LIFT*(*init w*)] **by** *fastforce*
have 5: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)) = (f \wedge \text{sfin } (\text{init } w))$
using 4 **by** *auto*
have 6: $\vdash (\text{finite} \wedge \text{fin } (\neg(\text{init } w))) = \text{sfin}(\neg(\text{init } w))$
using *SFinEqvFinAndFinite*[of *LIFT*($\neg(\text{init } w)$)] **by** *fastforce*
have 7: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\neg(\text{init } w))) = (f \wedge \text{sfin } (\neg(\text{init } w)))$
using 6 **by** *auto*
show ?thesis
using 3 5 7 **by** *fastforce*
qed

lemma *AndSFinSChopEqvStateAndSChop*:
 $\vdash (f \wedge \text{sfin } (\text{init } w)) \frown g = f \frown ((\text{init } w) \wedge g)$
proof –
have 1: $\vdash (\text{sfin } (\text{init } w)) \text{syields } (\text{init } w)$
by (*rule SFinSYields*)
have 2: $\vdash f \wedge \text{sfin } (\text{init } w) \longrightarrow \text{sfin } (\text{init } w)$
by *auto*
hence 3: $\vdash (\text{sfin } (\text{init } w)) \text{syields } (\text{init } w) \longrightarrow$
 $(f \wedge \text{sfin } (\text{init } w)) \text{syields } (\text{init } w)$
using *LeftSYieldsImpSYields* **by** *metis*
have 4: $\vdash (f \wedge \text{sfin } (\text{init } w)) \text{syields } (\text{init } w)$
using 1 3 *MP* **by** *fastforce*
have 5: $\vdash (f \wedge \text{sfin } (\text{init } w)) \frown g \wedge (f \wedge \text{sfin } (\text{init } w)) \text{syields } (\text{init } w)$
 $\longrightarrow (f \wedge \text{sfin } (\text{init } w)) \frown (g \wedge (\text{init } w))$
by (*rule SChopAndSYieldsImp*)
have 6: $\vdash (f \wedge \text{sfin } (\text{init } w)) \frown g \longrightarrow (f \wedge \text{sfin } (\text{init } w)) \frown (g \wedge (\text{init } w))$
using 4 5 **by** *fastforce*
have 7: $\vdash (f \wedge \text{sfin } (\text{init } w)) \frown (g \wedge (\text{init } w)) \longrightarrow f \frown (g \wedge (\text{init } w))$
by (*rule AndSChopA*)
have 8: $\vdash g \wedge (\text{init } w) \longrightarrow (\text{init } w) \wedge g$
by *auto*
hence 9: $\vdash f \frown (g \wedge (\text{init } w)) \longrightarrow f \frown ((\text{init } w) \wedge g)$
by (*rule RightSChopImpSChop*)
have 10: $\vdash (f \wedge \text{sfin } (\text{init } w)) \frown g \longrightarrow f \frown ((\text{init } w) \wedge g)$
using 6 7 9 **by** *fastforce*
have 11: $\vdash (f \wedge \text{finite}) \longrightarrow (f \wedge \text{sfin } (\text{init } w)) \vee (f \wedge \text{sfin } (\neg (\text{init } w)))$
using *AndFiniteImpAndSFinStateOrSFinNotState* **by** *blast*
hence 12: $\vdash f \frown ((\text{init } w) \wedge g) \longrightarrow$
 $((f \wedge \text{sfin } (\text{init } w)) \frown g) \vee ((f \wedge \text{sfin } (\neg (\text{init } w))) \frown g) \longrightarrow f \frown ((\text{init } w) \wedge g)$

by (metis FiniteImp LeftChopImpChop inteq-reflection schop-d-def)
 have 13: $\vdash ((f \wedge \text{sf}in \text{ (init } w)) \vee (f \wedge \text{sf}in (\neg (\text{init } w)))) \frown ((\text{init } w) \wedge g)$
 $=$
 $((f \wedge \text{sf}in (\text{init } w)) \frown ((\text{init } w) \wedge g) \vee (f \wedge \text{sf}in (\neg (\text{init } w))) \frown ((\text{init } w) \wedge g))$
 by (rule OrSChopEqv)
 have 14: $\vdash (f \wedge \text{sf}in (\text{init } (\neg w))) \frown ((\text{init } w) \wedge g) \longrightarrow \Diamond (\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g)$
 using SFinSChopEqvDiamond
 by (metis SChopImpSChop Prop12 int-iffD1 inteq-reflection lift-and-com)
 have 141: $\vdash \neg (\Diamond (\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g)) \longrightarrow$
 $\neg ((f \wedge \text{sf}in (\text{init } (\neg w))) \frown ((\text{init } w) \wedge g))$
 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: $\vdash \neg ((f \wedge \text{sf}in (\text{init } (\neg w))) \frown ((\text{init } w) \wedge g))$
 using 15 141 by fastforce
 have 1511: $\vdash (f \wedge \text{sf}in (\neg (\text{init } w))) \frown ((\text{init } w) \wedge g) \longrightarrow \#False$
 using 151 by (metis Initprop(2) int-simps(14) inteq-reflection)
 have 152: $\vdash (f \wedge \text{sf}in (\text{init } w)) \frown ((\text{init } w) \wedge g) \vee (f \wedge \text{sf}in (\neg (\text{init } w))) \frown ((\text{init } w) \wedge g) \longrightarrow$
 $(f \wedge \text{sf}in (\text{init } w)) \frown ((\text{init } w) \wedge g)$
 using 1511 by fastforce
 have 16: $\vdash f \frown ((\text{init } w) \wedge g) \longrightarrow (f \wedge \text{sf}in (\text{init } w)) \frown ((\text{init } w) \wedge g)$
 using 12 13 152 by fastforce
 have 17: $\vdash (f \wedge \text{sf}in (\text{init } w)) \frown ((\text{init } w) \wedge g) \longrightarrow (f \wedge \text{sf}in (\text{init } w)) \frown g$
 by (rule SChopAndB)
 have 18: $\vdash f \frown ((\text{init } w) \wedge g) \longrightarrow (f \wedge \text{sf}in (\text{init } w)) \frown g$
 using 16 17 by fastforce
 from 10 18 show ?thesis by fastforce
 qed

lemma DfAndSFinEqvSChopState:

$\vdash df (f \wedge \text{sf}in (\text{init } w)) = f \frown (\text{init } w)$

proof –

have 1: $\vdash (f \wedge \text{sf}in (\text{init } w)) \frown \#True = f \frown ((\text{init } w) \wedge \#True)$

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: $\vdash (f \wedge \text{sf}in (\text{init } w)) \frown \#True = f \frown (\text{init } w)$

using 1 3 by auto

from 4 show ?thesis by (simp add: df-d-def)

qed

lemma SFinNotStateEqvNotSFinState:

$\vdash \text{finite} \longrightarrow (\neg (\text{sf}in (\text{init } w))) = (\text{sf}in (\text{init } (\neg w)))$

using SFinEqvTrueSChopAndEmpty

by (metis InitAndEmptyEqvAndEmpty SFinprop(3) inteq-reflection)

lemma *BfImpSFinEqvSYieldsState*:

$\vdash \text{bf } (f \longrightarrow \text{sfin } (\text{init } w)) = f \text{ syields } (\text{init } w)$
proof –
have 1: $\vdash \text{df } (f \wedge \text{sfin } (\text{init } (\neg w))) = f \frown (\text{init } (\neg w))$
by (rule *DfAndSFinEqvSChopState*)
have 2: $\vdash \text{finite} \longrightarrow (f \wedge \text{sfin } (\text{init } (\neg w))) = (f \wedge \neg(\text{sfin } (\text{init } w)))$
using *SFinNotStateEqvNotSFinState* **by** *fastforce*
have 3: $\vdash (f \wedge \neg(\text{sfin } (\text{init } w))) = (\neg(f \longrightarrow \text{sfin } (\text{init } w)))$
by *auto*
have 4: $\vdash \text{finite} \longrightarrow (f \wedge \text{sfin } (\text{init } (\neg w))) = (\neg(f \longrightarrow \text{sfin } (\text{init } w)))$
using 2 3 **by** *fastforce*
hence 5: $\vdash \text{df } (f \wedge \text{sfin } (\text{init } (\neg w))) = \text{df } (\neg(f \longrightarrow \text{sfin } (\text{init } w)))$
by (metis *DfEqvNotBfNot FiniteImpAnd df-d-def inteq-reflection schop-d-def*)
have 6: $\vdash \text{df } (\neg(f \longrightarrow \text{sfin } (\text{init } w))) = (\neg(\text{bf } (f \longrightarrow \text{sfin } (\text{init } w))))$
by (rule *DfNotEqvNotBf*)
have 7: $\vdash \neg(\text{bf } (f \longrightarrow \text{sfin } (\text{init } w))) = f \frown (\text{init } (\neg w))$
using 1 5 6 *Initprop* **by** *fastforce*
hence 8: $\vdash \text{bf } (f \longrightarrow \text{sfin } (\text{init } w)) = (\neg(f \frown (\neg(\text{init } w))))$
by (metis *Initprop(2) int-eq int-simps(7)*)
from 8 **show** ?thesis **by** (simp add: *syields-d-def*)
qed

lemma *StateImpSYields*:

assumes $\vdash (\text{init } w) \wedge f \longrightarrow \text{sfin } (\text{init } w1)$
shows $\vdash (\text{init } w) \longrightarrow (f \text{ syields } (\text{init } w1))$
proof –
have 1: $\vdash (\text{init } w) \wedge f \longrightarrow \text{sfin } (\text{init } w1)$ **using** *assms* **by** *auto*
hence 2: $\vdash (\text{init } w) \longrightarrow (f \longrightarrow \text{sfin } (\text{init } w1))$ **by** *auto*
hence 3: $\vdash (\text{init } w) \longrightarrow \text{bf } (f \longrightarrow \text{sfin } (\text{init } w1))$ **using** *StateImpBfGen* **by** *auto*
have 4: $\vdash \text{bf } (f \longrightarrow \text{sfin } (\text{init } w1)) = f \text{ syields } (\text{init } w1)$ **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

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*:

$\vdash \Box f = \#True \text{ syields } f$
proof –
 have 1: $\vdash \#True \frown (\neg f) = \Diamond(\neg f)$ **by** (rule *TrueSChopEqvDiamond*)
 hence 2: $\vdash (\neg(\#True \frown (\neg f))) = (\neg(\Diamond(\neg f)))$ **by** *auto*
 have 3: $\vdash \Box f = (\neg(\Diamond(\neg f)))$ **by** (simp add: *always-d-def*)
 have 4: $\vdash \Box f = (\neg(\#True \frown (\neg f)))$ **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: $\vdash \Box g \longrightarrow f \text{ yields } g$ **by** (*rule BoxImpSYields*)
 from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *SYieldsAndSYieldsEqvSYieldsAnd*:

$\vdash ((f \text{ yields } g) \wedge (f \text{ yields } g1)) = f \text{ yields } (g \wedge g1)$

proof –
 have 1: $\vdash f \frown (\neg g \vee \neg g1) = ((f \frown (\neg g)) \vee (f \frown (\neg g1)))$ **by** (*rule SChopOrEqv*)
 hence 2: $\vdash ((f \frown (\neg g)) \vee (f \frown (\neg g1))) = f \frown (\neg g \vee \neg g1)$ **by** *auto*
 have 3: $\vdash (\neg g \vee \neg g1) = (\neg (g \wedge g1))$ **by** *auto*
 hence 4: $\vdash f \frown (\neg g \vee \neg g1) = f \frown (\neg (g \wedge g1))$ **by** (*rule RightSChopEqvSChop*)
 have 5: $\vdash (f \frown (\neg g)) \vee (f \frown (\neg g1)) = f \frown (\neg (g \wedge g1))$ **using** 2 4 **by** *fastforce*
 hence 6: $\vdash (\neg (f \frown (\neg g)) \wedge \neg (f \frown (\neg g1))) = (\neg (f \frown (\neg (g \wedge g1))))$ **using** 1 4 **by** *fastforce*
 from 6 **show** *?thesis* **by** (*simp add: syields-d-def*)
qed

lemma *SYieldsAndSYieldsImpAndSYieldsAnd*:

$\vdash (f \text{ yields } g) \wedge (f1 \text{ yields } g1) \longrightarrow (f \wedge f1) \text{ yields } (g \wedge g1)$

proof –
 have 1: $\vdash f \text{ yields } g \longrightarrow (f \wedge f1) \text{ yields } g$
 by (*rule AndSYieldsA*)
 have 2: $\vdash f1 \text{ yields } g1 \longrightarrow (f \wedge f1) \text{ yields } g1$
 by (*rule AndSYieldsB*)
 have 3: $\vdash ((f \wedge f1) \text{ yields } g \wedge (f \wedge f1) \text{ yields } g1) = (f \wedge f1) \text{ yields } (g \wedge g1)$
 by (*rule SYieldsAndSYieldsEqvSYieldsAnd*)
 from 1 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *SYieldsSYieldsEqvSChopSYields*:

$\vdash f \text{ yields } (g \text{ yields } h) = (f \frown g) \text{ yields } h$

proof –
 have 1: $\vdash f \frown (g \frown (\neg h)) = (f \frown g) \frown (\neg h)$ **by** (*rule SChopAssoc*)
 hence 2: $\vdash f \frown (g \frown (\neg h)) = (f \frown g) \frown (\neg h)$ **by** *auto*
 have 3: $\vdash g \frown (\neg h) = (\neg \neg (g \frown (\neg h)))$ **by** *auto*
 hence 4: $\vdash f \frown (g \frown (\neg h)) = f \frown (\neg \neg (g \frown (\neg h)))$ **by** (*rule RightSChopEqvSChop*)
 have 5: $\vdash f \frown (\neg \neg (g \frown (\neg h))) = (f \frown g) \frown (\neg h)$ **using** 2 4 **by** *auto*
 hence 6: $\vdash f \frown (\neg (g \text{ yields } h)) = (f \frown g) \frown (\neg h)$ **by** (*simp add: syields-d-def*)
 hence 7: $\vdash (\neg (f \frown (\neg (g \text{ yields } h)))) = (\neg ((f \frown g) \frown (\neg h)))$ **by** *auto*
 from 7 **show** *?thesis* **by** (*simp add: syields-d-def*)
qed

lemma *EmptyYields*:

$\vdash \text{empty} \text{ yields } f = f$

proof –
 have 1: $\vdash \text{empty} \frown (\neg f) = (\neg f)$ **by** (*rule EmptySChop*)
 hence 2: $\vdash (\neg (\text{empty} \frown (\neg f))) = f$ **by** *auto*
 from 2 **show** *?thesis* **by** (*simp add: syields-d-def*)

qed

lemma *NextSYields*:

$\vdash (\bigcirc f) \text{ syields } g = \text{wnext } (f \text{ syields } g)$

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*:

$\vdash \text{skip} \text{ syields } f = \text{wnext } f$

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*:

$\vdash \bigcirc f \longrightarrow \text{skip} \text{ syields } f$

proof –

have 1: $\vdash \bigcirc f \longrightarrow \text{wnext } f$ **using** *WnextEqvEmptyOrNext* **by** *fastforce*

have 2: $\vdash \text{skip} \text{ syields } f = \text{wnext } f$ **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 $\vdash f = \text{if}_i (\text{init } w) \text{ then } g \text{ else } h$
shows $\vdash \text{df } f = \text{if}_i (\text{init } w) \text{ then } (\text{df } g) \text{ else } (\text{df } h)$
proof –
have 1: $\vdash f = \text{if}_i (\text{init } w) \text{ then } g \text{ else } h$ **using** *assms* **by** *auto*
hence 2: $\vdash f \frown \#True = \text{if}_i (\text{init } w) \text{ then } (g \frown \#True) \text{ else } (h \frown \#True)$ **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: $\vdash \neg \neg f \longrightarrow \neg (\text{sba } (\neg f) \wedge \text{finite})$ **by** *fastforce*

have 3: $\vdash f = (\neg \neg f)$ **by** *simp*

have 4: $\vdash \text{sda } f = (\neg (\text{sba } (\neg f)))$ **by** (*rule SDaEqvNotSBaNot*)

from 2 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *SBaGen*:

assumes $\vdash f$

shows $\vdash \text{sba } f$

proof –

have 1: $\vdash f$ **using** *assms* **by** *auto*

hence 2: $\vdash \Box f$ **by** (*rule BoxGen*)

hence 3: $\vdash \text{bf } (\Box f)$ **by** (*rule BfGen*)

have 4: $\vdash \text{sba } f = \text{bf } (\Box f)$ **by** (*rule SBaEqvBfBt*)

from 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *SBaImpDist*:

$\vdash \text{sba } (f \longrightarrow g) \longrightarrow \text{sba } f \longrightarrow \text{sba } g$

proof –

have 1: $\vdash \text{bf } (f \longrightarrow g) \longrightarrow (\text{bf } f \longrightarrow \text{bf } g)$

by (*rule BfImpDist*)

hence 2: $\vdash \Box(\text{bf } (f \longrightarrow g) \longrightarrow (\text{bf } f \longrightarrow \text{bf } g))$

by (*rule BoxGen*)

have 3: $\vdash \Box(\text{bf } (f \longrightarrow g) \longrightarrow (\text{bf } f \longrightarrow \text{bf } g))$

\longrightarrow

$(\Box(\text{bf } (f \longrightarrow g)) \longrightarrow (\Box(\text{bf } f) \longrightarrow \Box(\text{bf } g)))$

by (*meson 2 BoxImpDist MP lift-imp-trans Prop01 Prop05 Prop09*)

have 4: $\vdash \Box(\text{bf } (f \longrightarrow g)) \longrightarrow (\Box(\text{bf } f) \longrightarrow \Box(\text{bf } g))$

using 2 3 *MP* **by** *fastforce*

have 5: $\vdash \text{sba } (f \longrightarrow g) = \Box(\text{bf } (f \longrightarrow g))$

by (*rule SBaEqvBtBf*)

have 6: $\vdash \text{sba } f = \Box(\text{bf } f)$

by (*rule SBaEqvBtBf*)

have 7: $\vdash \text{sba } g = \Box(\text{bf } g)$

by (*rule SBaEqvBtBf*)

from 4 5 6 7 **show** *?thesis* **by** *fastforce*

qed

lemma *SBaAndEqv*:

$\vdash \text{sba } (f \wedge g) = (\text{sba } f \wedge \text{sba } g)$

proof –

have 1: $\vdash \text{sba } (f \wedge g) = \Box(\text{bf } (f \wedge g))$

by (*rule SBaEqvBtBf*)

have 2: $\vdash \text{bf } (f \wedge g) = (\text{bf } f \wedge \text{bf } g)$

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 SBaImpSBaEqvSBa 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)
qed

from 1 2 show ?thesis by fastforce
qed

lemma BaRightSChopImpSChop:

$\vdash \text{ba } (g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$

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 AndSChopAndCommutate)

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 AndSChopAndCommutate)

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*:

$\vdash (\neg (ba\ f)) \frown g \longrightarrow \neg (ba\ f)$

proof –

have 1: $\vdash (\neg (ba\ f)) \frown g \longrightarrow df\ (\neg (ba\ f))$ **by** (rule *SChopImpDf*)

have 2: $\vdash df\ (\neg (ba\ f)) \longrightarrow \neg (ba\ f)$ **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: $\vdash \text{sfm } (\text{init } w) \longrightarrow f \text{ syields } (\text{sfm } (\text{init } w))$
by (rule *SFinImpSYieldsSFin*)
have 2: $\vdash (f \frown g) \wedge (\text{sfm } (\text{init } w)) \longrightarrow (f \frown g) \wedge f \text{ syields } (\text{sfm } (\text{init } w))$
using 1 **by** *fastforce*
have 3: $\vdash f \frown g \wedge f \text{ syields } (\text{sfm } (\text{init } w)) \longrightarrow$
 $f \frown (g \wedge (\text{sfm } (\text{init } w)))$
using *SChopAndSYieldsImp* **by** *blast*
have 4: $\vdash (f \frown g) \wedge (\text{sfm } (\text{init } w)) \longrightarrow f \frown (g \wedge \text{sfm } (\text{init } w))$
using 2 3 **by** (metis (mono-tags, lifting) *lift-imp-trans*)
from 4 **show** ?thesis
by (simp add: *Prop12 SChopAndA SChopSFinExportA int-iffI*)
qed

lemma *SChopAndNotSFin*:

$\vdash (f \frown g \wedge \neg (\text{sfm } (\text{init } w)) \wedge \text{finite}) = f \frown (g \wedge \neg (\text{sfm } (\text{init } w)) \wedge \text{finite})$

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: $\vdash (\text{init } w \wedge \text{finite} \wedge (\text{init } w \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w1)));$

$(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) =$
 $(init\ w \wedge (finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1))));$
 $(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)))$
by (*simp add: StateAndChop*)
have 05: $\vdash init\ w \wedge finite \wedge ((init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge finite);$
 $(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \longrightarrow$
 $(init\ w \wedge finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)));$
 $(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2))$
by (*metis 04 AndChopCommute ChopAndB StateAndEmptyChop int-eq*)
have 06: $\vdash init\ w \wedge finite \wedge ((init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge finite);$
 $(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \longrightarrow$
 $(init\ w \wedge finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)));$
 $(init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite$
by (*meson 03 05 Prop12*)
have 1: $\vdash (init\ w) \wedge finite \wedge$
 $((init\ w) \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge$
 $((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2))$
 \longrightarrow
 $((init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow sfin\ (init\ w1))) \wedge$
 $((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite$
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: $\vdash ((init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow sfin\ (init\ w1))) \wedge$
 $((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite$
 \longrightarrow
 $(sfin\ (init\ w1)) \wedge ((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite$
using 2 *LeftSChopImpSChop* **by** *blast*
have 4: $\vdash (sfin\ (init\ w1)) \wedge ((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2)) =$
 $\Diamond((init\ w1) \wedge ((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2)))$
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: $\vdash init\ w \wedge finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge (init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2))$
 $\longrightarrow sfin\ (init\ w1) \wedge ((init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite)$
by (*meson 1 3 lift-imp-trans*)
have 53: $\vdash init\ w \wedge finite \wedge (init\ w \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge (init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2))$
 $\longrightarrow \Diamond(init\ w1 \wedge (init\ w1 \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite)$
by (*metis 52 SFinSChopEqvDiamond inteq-reflection*)
have 6: $\vdash (init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow sfin\ (init\ w1)) \wedge$
 $((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2))$
 $\longrightarrow sfin\ (init\ w2)$
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, hide-lams*) 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 $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow f1 \wedge \text{sfin } (\text{init } w1)$
 $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1 \wedge \text{sfin } (\text{init } w2)$
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 $\vdash f \wedge \neg g \longrightarrow (\text{more} \frown (f \wedge \neg g))$
 shows $\vdash f \wedge \text{finite} \longrightarrow g$
 proof –
 have 1: $\vdash f \wedge \neg g \longrightarrow (\text{more} \frown (f \wedge \neg g))$ using *assms* by *auto*
 hence 2: $\vdash \text{finite} \longrightarrow \neg (f \wedge \neg g)$ by (rule *MoreSChopLoop*)
 from 2 show ?thesis by *fastforce*
 qed

lemma *MoreSChopLoopFinite*:

assumes $\vdash f \wedge \text{finite} \longrightarrow \text{more} \frown f$
 shows $\vdash \text{finite} \longrightarrow \neg f$
 proof –
 have 1: $\vdash f \wedge \text{finite} \longrightarrow \text{more} \frown f$
 using *assms* by *auto*
 hence 11: $\vdash \Diamond (f \wedge \text{finite}) \longrightarrow \Diamond (\text{more} \frown f)$
 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: $\vdash \Diamond (f \wedge \text{finite}) \longrightarrow \bigcirc (\Diamond f)$
 using 11 14 2 by *fastforce*
 have 31: $\vdash \Diamond (f \wedge \text{finite}) = ((\Diamond f) \wedge \text{finite})$
 by (metis (no-types, lifting) *DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty SFinAndSChop SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection lift-and-com*)
 have 32: $\vdash (\Diamond f) \wedge \text{finite} \longrightarrow \bigcirc (\Diamond f)$
 using 3 31 by *fastforce*
 hence 4: $\vdash \text{finite} \longrightarrow \neg (\Diamond f)$
 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 $\vdash (f \wedge \neg g) \wedge \text{finite} \longrightarrow (\text{more} \frown (f \wedge \neg g))$
 shows $\vdash f \wedge \text{finite} \longrightarrow g$

proof –
have 1: $\vdash (f \wedge \neg g) \wedge \text{finite} \longrightarrow (more \frown (f \wedge \neg g))$ **using** *assms* **by** *auto*
hence 2: $\vdash \text{finite} \longrightarrow \neg (f \wedge \neg g)$ **by** (*simp add: MoreSChopLoopFinite*)
from 2 **show** ?thesis **by** (*simp add: Valid-def*)
qed

lemma *SChopLoop*:
assumes $\vdash f \longrightarrow g \frown f$
 $\vdash g \longrightarrow fmore$
shows $\vdash \text{finite} \longrightarrow \neg f$
proof –
have 1: $\vdash f \longrightarrow g \frown f$ **using** *assms* **by** *auto*
have 2: $\vdash g \longrightarrow more$ **using** *assms* **by** (*simp add: Prop12 fmore-d-def*)
hence 3: $\vdash g \frown f \longrightarrow more \frown f$ **by** (*rule LeftSChopImpSChop*)
have 4: $\vdash f \longrightarrow more \frown f$ **using** 1 3 **by** *fastforce*
from 4 **show** ?thesis **using** *MoreSChopLoop* **by** *auto*
qed

lemma *SChopLoopB*:
assumes $\vdash f \longrightarrow g \frown f$
 $\vdash g \longrightarrow more$
shows $\vdash \text{finite} \longrightarrow \neg f$
proof –
have 1: $\vdash f \longrightarrow g \frown f$ **using** *assms* **by** *auto*
have 2: $\vdash g \longrightarrow more$ **using** *assms* **by** *auto*
hence 3: $\vdash g \frown f \longrightarrow more \frown f$ **by** (*rule LeftSChopImpSChop*)
have 4: $\vdash f \longrightarrow more \frown f$ **using** 1 3 **by** *fastforce*
from 4 **show** ?thesis **using** *MoreSChopLoop* **by** *blast*
qed

lemma *SChopContra*:
assumes $\vdash f \wedge \neg g \longrightarrow h \frown f \wedge \neg (h \frown g)$
 $\vdash h \longrightarrow fmore$
shows $\vdash f \wedge \text{finite} \longrightarrow g$
proof –
have 1: $\vdash f \wedge \neg g \longrightarrow h \frown f \wedge \neg (h \frown g)$ **using** *assms* **by** *auto*
have 2: $\vdash h \longrightarrow more$ **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: $\vdash h \frown (f \wedge \neg g) \longrightarrow more \frown (f \wedge \neg g)$ **using** 2 **by** (*rule LeftSChopImpSChop*)
have 5: $\vdash f \wedge \neg g \longrightarrow more \frown (f \wedge \neg g)$ **using** 1 3 4 **by** *fastforce*
from 5 **show** ?thesis **using** *MoreSChopContra* **by** *auto*
qed

lemma *SChopContraB*:
assumes $\vdash f \wedge \neg g \longrightarrow h \frown f \wedge \neg (h \frown g)$
 $\vdash h \longrightarrow more$
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: $\vdash h \longrightarrow more$ **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:  $\vdash h \frown (f \wedge \neg g) \longrightarrow more \frown (f \wedge \neg g)$  using 2 by (rule LeftSChopImpSChop)
have 5:  $\vdash f \wedge \neg g \longrightarrow more \frown (f \wedge \neg g)$  using 1 3 4 by fastforce
from 5 show ?thesis using MoreSChopContra by blast
qed

```

7.7 Properties of SChopstar and SChopplus

lemma *AndEmptySChopAndEmptyEqvAndEmpty*:

$\vdash (f \wedge empty) \frown (f \wedge empty) = (f \wedge empty)$

by (metis (no-types, lifting) *AndEmptyChopAndEmptyEqvAndEmpty EmptyImpFinite Prop10 Prop12 int-iffD1*

inteq-reflection schop-d-def)

lemma *SPowerImpFinite*:

$\vdash spower\ f\ n \longrightarrow finite$

proof

(*induct n*)

case 0

then show ?case

using *EmptyImpFinite* by auto

next

case (*Suc n*)

then show ?case

by (metis *DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty Prop10 SChopSFinExportA*
SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection spow-Suc)

qed

lemma *SPowerCommute*:

$\vdash f \frown spower\ f\ n = spower\ f\ n \frown (f \wedge finite)$

proof

(*induct n*)

case 0

then show ?case

by (metis *ChopEmptySem EmptySChop intI inteq-reflection schop-d-def spow-0*)

next

case (*Suc n*)

then show ?case

by (metis *SChopAssoc inteq-reflection spow-Suc*)

qed

lemma *SChopInductL*:

assumes $\vdash g \vee f \frown h \longrightarrow h$

shows $\vdash (spower\ f\ n) \frown g \longrightarrow h$

using *assms*

by (metis *ChopInductFiniteL PowerSpowerdef Prop10 SPowerImpFinite inteq-reflection schop-d-def*)

lemma *SChopInductMoreL*:

assumes $\vdash g \vee (f \wedge more) \frown h \longrightarrow h$

shows $\vdash (spower\ f\ n) \frown g \longrightarrow h$

```

proof
  (induct n)
  case 0
  then show ?case using assms by (metis SChopInductL spow-0)
  next
  case (Suc n)
  then show ?case
  proof –
  have 1:  $\vdash \text{spower } f \text{ (Suc } n) \frown g = (f \frown \text{spower } f \text{ } n) \frown g$ 
    by simp
  have 2:  $\vdash (f \frown \text{spower } f \text{ } n) \frown g = f \frown ((\text{spower } f \text{ } n) \frown g)$ 
    by (meson SChopAssoc Prop11)
  have 3:  $\vdash f \frown ((\text{spower } f \text{ } n) \frown g) \longrightarrow f \frown h$ 
    by (simp add: RightSChopImpSChop Suc.hyps)
  have 31:  $\vdash f = ((f \wedge \text{more}) \vee (f \wedge \text{empty}))$ 
    unfolding empty-d-def by fastforce
  have 4:  $\vdash f \frown h = ((f \wedge \text{more}) \frown h \vee ((f \wedge \text{empty})) \frown h)$ 
    using 31 by (simp add: OrSChopEqvRule)
  have 5:  $\vdash ((f \wedge \text{more})) \frown h \longrightarrow h$ 
    using assms by auto
  have 6:  $\vdash ((f \wedge \text{empty})) \frown h \longrightarrow h$ 
    by (metis AndSChopB EmptySChop inteq-reflection)
  from 5 6 4 3 2 1 show ?thesis by fastforce
qed
qed

```

```

lemma SChopImpFinite:
assumes  $\vdash f \longrightarrow \text{finite}$ 
shows  $\vdash g \frown f \longrightarrow \text{finite}$ 
using assms
by (metis DiamondImpDiamond FiniteChopEqvDiamond FiniteChopFiniteEqvFinite SChopImpDiamond
    inteq-reflection lift-imp-trans)

```

```

lemma SChopInductR:
assumes  $\vdash g \vee h \frown f \longrightarrow h$ 
shows  $\vdash g \frown (\text{spower } f \text{ } n) \longrightarrow h$ 
proof
  (induct n)
  case 0
  then show ?case using assms
    by (metis ChopEmpty MP Prop11 Prop12 int-simps(33) lift-imp-trans schop-d-def spow-0)
  next
  case (Suc n)
  then show ?case
  proof –
  have 1:  $\vdash g \frown (\text{spower } f \text{ (Suc } n)) = g \frown (f \frown (\text{spower } f \text{ } n))$ 
    by simp
  have 2:  $\vdash g \frown (f \frown (\text{spower } f \text{ } n)) = g \frown ((\text{spower } f \text{ } n) \frown (f \wedge \text{finite}))$ 
    using SPowerCommute by (simp add: SPowerCommute RightSChopEqvSChop)
  have 3:  $\vdash g \frown ((\text{spower } f \text{ } n) \frown (f \wedge \text{finite})) =$ 

```

```

      (g  $\frown$  (spower f n))  $\frown$  (f  $\wedge$  finite)
    using SChopAssoc by blast
  have 4:  $\vdash$  (g  $\frown$  (spower f n))  $\frown$  (f  $\wedge$  finite)  $\longrightarrow$  h  $\frown$  (f  $\wedge$  finite)
    using LeftSChopImpSChop Suc.hyps by blast
  have 5:  $\vdash$  h  $\frown$  (f  $\wedge$  finite)  $\longrightarrow$  h
    using assms by (metis Prop03 Prop10 SChopAndA inteq-reflection lift-imp-trans)
  from 1 2 3 4 5 show ?thesis by fastforce
qed
qed

```

lemma *SChopExistSPower*:

```

 $\vdash$  (g  $\frown$  ( $\exists$  n. spower f n)) = ( $\exists$  n. g  $\frown$  spower f n)
using SChopExist by fastforce

```

lemma *ExistSChopSPower*:

```

 $\vdash$  ( $\exists$  n. (spower f n)  $\frown$  g) = ( $\exists$  n. spower f n)  $\frown$  g
using ExistSChop by fastforce

```

lemma *SPowerStarCommute*:

```

 $\vdash$  f  $\frown$  ( $\exists$  n. spower f n) = ( $\exists$  n. spower f n)  $\frown$  (f  $\wedge$  finite)
proof -
  have 1:  $\vdash$  f  $\frown$  ( $\exists$  n. spower f n) =
    ( $\exists$  n. f  $\frown$  spower f n)
    using SChopExistSPower by blast
  have 2:  $\vdash$  ( $\exists$  n. f  $\frown$  spower f n) =
    ( $\exists$  n. (spower f n)  $\frown$  (f  $\wedge$  finite) )
    using SPowerCommute by fastforce
  have 3:  $\vdash$  ( $\exists$  n. (spower f n)  $\frown$  (f  $\wedge$  finite)) =
    ( $\exists$  n. (spower f n))  $\frown$  (f  $\wedge$  finite)
    using ExistSChopSPower by blast
  from 1 2 3 show ?thesis by fastforce
qed

```

lemma *SPowerSucAndEmptyEqvAndEmpty*:

```

 $\vdash$  (spower (f  $\wedge$  empty) (Suc n)) = (f  $\wedge$  empty)
proof
  (induct n)
  case 0
  then show ?case
    by (metis PowerSpowerdef PowerSucAndEmptyEqvAndEmpty inteq-reflection)
  next
  case (Suc n)
  then show ?case
    by (metis AndEmptySChopAndEmptyEqvAndEmpty inteq-reflection spow-Suc)
qed

```

lemma *SPowerOr*:

```

 $\vdash$  (spower (f  $\vee$  g) (Suc n)) = ( (f  $\frown$  spower (f  $\vee$  g) n)  $\vee$ 
    (g  $\frown$  spower (f  $\vee$  g) n))
by (simp add: FiniteOr OrSChopEqvRule)

```

lemma *PowerEmptyOrMore*:

$\vdash (\text{spower } (f \wedge \text{empty}) \vee (f \wedge \text{more})) (\text{Suc } n) =$
 $((f \wedge \text{empty}) \frown (\text{spower } (f \wedge \text{empty}) \vee (f \wedge \text{more})) n) \vee$
 $(f \wedge \text{more}) \frown (\text{power } (f \wedge \text{empty}) \vee (f \wedge \text{more})) n)$

using *SPowerOr*

by (*metis PowerSpowerdef inteq-reflection*)

lemma *SPSEqvEmptyOrSchopSPS*:

$\vdash \text{spowerstar } f = (\text{empty} \vee f \frown \text{spowerstar } f)$

by (*simp add: spowerstar-d-def spowersem*)

lemma *EmptyImpSCS*:

$\vdash \text{empty} \longrightarrow \text{schopstar } f$

proof –

have 1: $\vdash \text{schopstar } f = (\text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f)$ **by** (*rule SC ChopstarEqv*)

have 2: $\vdash \text{empty} \longrightarrow \text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f$ **by** *auto*

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *SCSEqvOrSchopSCS*:

$\vdash \text{schopstar } f = (\text{empty} \vee (f \frown \text{schopstar } f))$

proof –

have 1: $\vdash \text{schopstar } f = (\text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f)$
by (*rule SC ChopstarEqv*)

have 2: $\vdash (f \wedge \text{more}) \frown \text{schopstar } f \longrightarrow f \frown \text{schopstar } f$
by (*rule AndSCHopA*)

have 3: $\vdash \text{schopstar } f \longrightarrow \text{empty} \vee f \frown \text{schopstar } f$
using 1 2 **by** (*metis int-iffD1 Prop08*)

have 4: $\vdash \text{empty} \longrightarrow \text{schopstar } f$
by (*rule EmptyImpSCS*)

have 5: $\vdash f \longrightarrow \text{empty} \vee (f \wedge \text{more})$
by (*auto simp: empty-d-def*)

have 6: $\vdash f \frown \text{schopstar } f \longrightarrow \text{schopstar } f \vee (f \wedge \text{more}) \frown \text{schopstar } f$
using 5 **by** (*rule EmptyOrSchopImpRule*)

have 7: $\vdash \text{schopstar } f \longrightarrow \text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f$
using 1 **by** *fastforce*

have 8: $\vdash f \frown \text{schopstar } f \longrightarrow \text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f$
using 6 7 **by** *fastforce*

hence 9: $\vdash f \frown \text{schopstar } f \longrightarrow \text{schopstar } f$
using 1 **by** *fastforce*

have 10: $\vdash \text{empty} \vee f \frown \text{schopstar } f \longrightarrow \text{schopstar } f$
using 9 4 **by** *fastforce*

from 3 10 **show** *?thesis* **by** *fastforce*

qed

lemma *SPowerSchopCommute*:

$\vdash ((f \wedge \text{more}) \frown \text{spower } (f \wedge \text{more}) n = \text{spower } (f \wedge \text{more}) n \frown ((f \wedge \text{more}) \wedge \text{finite}))$

using *SPowerCommute* **by** *auto*

lemma *SChopExist*:

$\vdash (g \frown (\exists n. \text{spower } (f \wedge \text{more}) n)) = (\exists n. g \frown \text{spower } (f \wedge \text{more}) n)$

using *SChopExistSPower* **by** *auto*

lemma *ExistSChop*:

$\vdash (\exists n. (\text{spower } (f \wedge \text{more}) n) \frown g) = (\exists n. \text{spower } (f \wedge \text{more}) n) \frown g$

using *ExistSChopSPower* **by** *auto*

lemma *SPowerstarInductL*:

assumes $\vdash g \vee f \frown h \longrightarrow h$

shows $\vdash (\text{spowerstar } f) \frown g \longrightarrow h$

proof –

have 1: $\vdash (\text{spowerstar } f) \frown g = ((\exists n. \text{spower } f n)) \frown g$

by (*simp add: spowerstar-d-def LeftChopEqvChop*)

have 2: $\vdash (\exists n. \text{spower } f n) \frown g =$

$(\exists n. (\text{spower } f n) \frown g)$

using *ExistSChopSPower* **by** *fastforce*

have 3: $\bigwedge n. \vdash (\text{spower } f n) \frown g \longrightarrow h$

using *SChopInductL assms* **by** *blast*

have 4: $\vdash (\exists n. (\text{spower } f n) \frown g) \longrightarrow h$

using 3 **by** (*simp add: Valid-def*) *fastforce*

from 1 2 4 **show** *?thesis* **by** (*metis inteq-reflection*)

qed

lemma *SChopstarInductL*:

assumes $\vdash g \vee f \frown h \longrightarrow h$

shows $\vdash (\text{schopstar } f) \frown g \longrightarrow h$

proof –

have 1: $\vdash (\text{schopstar } f) \frown g = (\exists n. \text{spower } (f \wedge \text{more}) n) \frown g$

by (*simp add: schopstar-d-def spowerstar-d-def LeftChopEqvChop*)

have 2: $\vdash (\exists n. \text{spower } (f \wedge \text{more}) n) \frown g =$

$(\exists n. (\text{spower } (f \wedge \text{more}) n) \frown g)$

using *ExistSChopSPower* **by** *fastforce*

have 21: $\vdash g \vee (f \wedge \text{more}) \frown h \longrightarrow h$

using *AndSChopA assms* **by** *fastforce*

have 3: $\bigwedge n. \vdash (\text{spower } (f \wedge \text{more}) n) \frown g \longrightarrow h$

using 21 *SChopInductL*[of *g LIFT(f ∧ more) h*] **by** *auto*

have 4: $\vdash (\exists n. (\text{spower } (f \wedge \text{more}) n) \frown g) \longrightarrow h$

using 3 **by** (*simp add: Valid-def*) *fastforce*

from 1 2 4 **show** *?thesis* **by** (*metis inteq-reflection*)

qed

lemma *SChopstarInductMoreL*:

assumes $\vdash g \vee (f \wedge \text{more}) \frown h \longrightarrow h$

shows $\vdash (\text{schopstar } f) \frown g \longrightarrow h$

proof –

have 1: $\vdash (\text{schopstar } f) \frown g = (\exists n. \text{spower } (f \wedge \text{more}) n) \frown g$

by (*simp add: schopstar-d-def spowerstar-d-def LeftChopEqvChop*)

have 2: $\vdash (\exists n. \text{spower } (f \wedge \text{more}) n) \frown g =$

$(\exists n. (\text{spower } (f \wedge \text{more}) n) \frown g)$

```

  using ExistSChopSPower by fastforce
have 3:  $\bigwedge n. \vdash (\text{spower } (f \wedge \text{more}) n) \frown g \longrightarrow h$ 
  using SChopInductL assms by (metis)
have 4:  $\vdash (\exists n. (\text{spower } (f \wedge \text{more}) n) \frown g) \longrightarrow h$ 
  using 3 by fastforce
from 1 2 4 show ?thesis
  by (metis integ-reflection)
qed

lemma SPowerstarInductR:
  assumes  $\vdash g \vee h \frown f \longrightarrow h$ 
  shows  $\vdash g \frown (\text{spowerstar } f) \longrightarrow h$ 
proof -
  have 1:  $\vdash g \frown (\text{spowerstar } f) = g \frown ((\exists n. \text{spower } f n))$ 
    by (simp add: spowerstar-d-def)
  have 2:  $\vdash (g \frown (\exists n. \text{spower } f n)) = (\exists n. g \frown (\text{spower } f n))$ 
    using SChopExistSPower by blast
  have 3:  $\bigwedge n. \vdash g \frown (\text{spower } f n) \longrightarrow h$ 
    using SChopInductR assms by blast
  have 4:  $\vdash (\exists n. g \frown (\text{spower } f n)) \longrightarrow h$ 
    using 3 by (simp add: Valid-def) fastforce
  from 1 2 4 show ?thesis by (metis integ-reflection)
qed

lemma SChopstarInductR:
  assumes  $\vdash g \vee h \frown f \longrightarrow h$ 
  shows  $\vdash g \frown (\text{schopstar } f) \longrightarrow h$ 
proof -
  have 1:  $\vdash g \frown (\text{schopstar } f) =$ 
     $g \frown ((\exists n. \text{spower } (f \wedge \text{more}) n))$ 
    by (simp add: schopstar-d-def spowerstar-d-def)
  have 2:  $\vdash (g \frown (\exists n. \text{spower } (f \wedge \text{more}) n)) =$ 
     $((\exists n. g \frown \text{spower } (f \wedge \text{more}) n))$ 
    using SChopExistSPower by fastforce
  have 21:  $\vdash h \frown (f \wedge \text{more}) \longrightarrow h$ 
    using assms by (metis Prop03 Prop10 SChopAndA integ-reflection lift-imp-trans)
  have 22:  $\vdash g \longrightarrow h$ 
    using assms by auto
  have 23:  $\vdash g \vee h \frown (f \wedge \text{more}) \longrightarrow h$ 
    using 21 22 Prop02 by blast
  have 3:  $\bigwedge n. \vdash g \frown (\text{spower } (f \wedge \text{more}) n) \longrightarrow h$ 
    using 23 SChopInductR[of  $g$   $h$   $\text{LIFT}(f \wedge \text{more})$ ] by auto
  have 4:  $\vdash (\exists n. g \frown (\text{spower } (f \wedge \text{more}) n)) \longrightarrow h$ 
    using 3 by (simp add: Valid-def) fastforce
  from 1 2 4 show ?thesis by (metis integ-reflection)
qed

lemma SChopstarInductMoreR:
  assumes  $\vdash g \vee h \frown (f \wedge \text{more}) \longrightarrow h$ 
  shows  $\vdash g \frown (\text{schopstar } f) \longrightarrow h$ 

```


proof –
have 1: $\vdash g \neg (\text{schopstar } f) = g \neg ((\exists n. \text{spower } (f \wedge \text{more}) n))$
by (*simp add: schopstar-d-def spowerstar-d-def*)
have 2: $\vdash (g \neg (\exists n. \text{spower } (f \wedge \text{more}) n)) =$
 $((\exists n. g \neg \text{spower } (f \wedge \text{more}) n))$
using *SChopExistSPower* **by** *fastforce*
have 3: $\bigwedge n. \vdash g \neg (\text{spower } (f \wedge \text{more}) n) \longrightarrow h$
using *SChopInductR assms* **by** (*metis*)
have 4: $\vdash (\exists n. g \neg (\text{spower } (f \wedge \text{more}) n)) \longrightarrow h$
using 3 **by** (*simp add: Valid-def*) *fastforce*
from 1 2 4 **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *SChopstarImpSPowerstar*:
 $\vdash \text{schopstar } f \longrightarrow \text{spowerstar } f$
by (*metis (mono-tags, lifting) PowerSpowerdef SChopstarFPowerstardef fpowerstar-d-def intI*
intensional-rews(3) intensional-rews(6) inteq-reflection spowerstar-d-def)

lemma *SPowerstarImpSChopstar*:
 $\vdash \text{spowerstar } f \longrightarrow \text{schopstar } f$
by (*metis (mono-tags, lifting) PowerSpowerdef SChopstarFPowerstardef fpowerstar-d-def intI*
intensional-rews(3) intensional-rews(6) inteq-reflection spowerstar-d-def)

lemma *SChopstarEqvSPowerstar*:
 $\vdash \text{schopstar } f = \text{spowerstar } f$
using *SChopstarImpSPowerstar SPowerstarImpSChopstar* **by** *fastforce*

lemma *SCSAndMoreEqvAndMoreSChop*:
 $\vdash (\text{schopstar } f \wedge \text{more}) = (f \wedge \text{more}) \neg \text{schopstar } f$

proof –
have 1: $\vdash (\text{empty} \vee (f \wedge \text{more}) \neg \text{schopstar } f) \wedge \text{more} \longrightarrow (f \wedge \text{more}) \neg \text{schopstar } f$
by (*auto simp: empty-d-def*)
have 2: $\vdash \text{schopstar } f = (\text{empty} \vee (f \wedge \text{more}) \neg \text{schopstar } f)$
by (*rule SChopstarEqv*)
have 3: $\vdash \text{schopstar } f \wedge \text{more} \longrightarrow (f \wedge \text{more}) \neg \text{schopstar } f$
using 1 2 **by** *fastforce*
have 4: $\vdash (f \wedge \text{more}) \neg \text{schopstar } f \longrightarrow \text{schopstar } f$
using 2 **by** *fastforce*
have 5: $\vdash (f \wedge \text{more}) \longrightarrow \text{more}$
by *auto*
hence 6: $\vdash (f \wedge \text{more}) \neg \text{schopstar } f \longrightarrow \text{more}$
by (*rule LeftSChopImpMoreRule*)
have 7: $\vdash (f \wedge \text{more}) \neg \text{schopstar } f \longrightarrow \text{schopstar } f \wedge \text{more}$
using 4 6 **by** *fastforce*
from 3 7 **show** *?thesis* **by** *fastforce*
qed

lemma *SPowerAndMoreAndFinite*:
 $\vdash ((\text{spower } (f \wedge \text{more}) n) \wedge \text{finite}) = (\text{spower } (f \wedge \text{more}) n)$
by (*meson Prop10 Prop11 SPowerImpFinite*)

lemma *SCSAndFinite*:

$\vdash (\text{schopstar } f \wedge \text{finite}) = \text{schopstar } f$

proof –

have 1: $\vdash (\text{schopstar } f \wedge \text{finite}) = ((\exists n. \text{spower } (f \wedge \text{more}) n) \wedge \text{finite})$

by (*simp add: chopstar-d-def spowerstar-d-def intI*)

have 2: $\vdash ((\exists n. \text{spower } (f \wedge \text{more}) n) \wedge \text{finite}) =$
 $(\exists n. \text{spower } (f \wedge \text{more}) n \wedge \text{finite})$

by (*simp add: Valid-def*)

have 3: $\vdash (\exists n. \text{spower } (f \wedge \text{more}) n \wedge \text{finite}) =$
 $(\exists n. (\text{spower } (f \wedge \text{more}) n))$

using *SPowerAndMoreAndFinite* **by** *fastforce*

have 4: $\vdash (\exists n. (\text{spower } (f \wedge \text{more}) n)) = \text{schopstar } f$

by (*simp add: chopstar-d-def spowerstar-d-def*)

from 1 2 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *SPowerchopAndFmore*:

$\vdash ((\text{spower } (f \wedge \text{more}) (\text{Suc } n)) \wedge \text{fmore}) = (\text{spower } (f \wedge \text{more}) (\text{Suc } n))$

by (*metis (no-types, lifting) FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite*
FmoreEqvSkipChopFinite LeftSChopImpMoreRule Prop10 Prop12 SPowerImpFinite int-iffD1
inteq-reflection lift-and-com spower-d.simps(2))

lemma *ExistSPowerAndMoreExpand*:

$\vdash (\exists n. \text{spower } (f \wedge \text{more}) n) = (\text{empty} \vee (\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n))))$

using *spowersem1[of LIFT(f \wedge more)]* **by** *auto*

lemma *SCSAndMoreEqvAndFMoreSChop*:

$\vdash (\text{schopstar } f \wedge \text{fmore}) = (f \wedge \text{more}) \frown \text{schopstar } f$

proof –

have 1: $\vdash (\text{schopstar } f \wedge \text{fmore}) = ((\exists n. \text{spower } (f \wedge \text{more}) n) \wedge \text{fmore})$

by (*simp add: chopstar-d-def spowerstar-d-def intI*)

have 2: $\vdash ((\exists n. \text{spower } (f \wedge \text{more}) n) \wedge \text{fmore}) =$
 $(\exists n. \text{spower } (f \wedge \text{more}) n \wedge \text{fmore})$

by (*simp add: Valid-def*)

have 3: $\vdash (\exists n. \text{spower } (f \wedge \text{more}) n \wedge \text{fmore}) =$
 $((\text{spower } (f \wedge \text{more}) 0 \vee (\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n)))) \wedge \text{fmore})$

using *ExistSPowerAndMoreExpand* **by** *fastforce*

have 4: $\vdash ((\text{spower } (f \wedge \text{more}) 0 \vee (\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n)))) \wedge \text{fmore}) =$
 $((\text{spower } (f \wedge \text{more}) 0 \wedge \text{fmore}) \vee ((\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n))) \wedge \text{fmore}))$

by *auto*

have 5: $\vdash (((\text{spower } (f \wedge \text{more}) 0 \wedge \text{fmore}) \vee ((\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n))) \wedge \text{fmore}))) =$
 $((\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n))) \wedge \text{fmore})$

using *NotFmoreAndEmpty* **by** *fastforce*

have 6: $\vdash ((\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n))) \wedge \text{fmore}) =$
 $(\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n)))$

using *SPowerchopAndFmore* **by** *fastforce*

have 7: $\vdash (\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n))) =$
 $(\exists n. ((f \wedge \text{more}) \frown (\text{spower } (f \wedge \text{more}) n)))$

by (*simp*)

have 8: $\vdash (\exists n. ((f \wedge \text{more}) \frown (\text{spower } (f \wedge \text{more}) n))) =$
 $(f \wedge \text{more}) \frown (\exists n. (\text{spower } (f \wedge \text{more}) n))$
using *SChopExist* **by** *fastforce*
show *?thesis unfolding schopstar-d-def spowerstar-d-def*
by (*metis* 2 3 4 5 6 7 8 *int-eq*)
qed

lemma *SCSAndMoreImpSChopSCS*:

$\vdash \text{s chopstar } f \wedge \text{more} \longrightarrow f \frown \text{s chopstar } f$

proof –

have 1: $\vdash (\text{s chopstar } f \wedge \text{more}) = (f \wedge \text{more}) \frown \text{s chopstar } f$ **by** (*rule SCSAndMoreEqvAndMoreSChop*)

have 2: $\vdash (f \wedge \text{more}) \frown \text{s chopstar } f \longrightarrow f \frown \text{s chopstar } f$ **by** (*rule AndSChopA*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *SCSMoreNotImpSChopSCSAndMore*:

$\vdash \text{s chopstar } f \wedge \text{more} \wedge \neg f \longrightarrow (f \wedge \text{more}) \frown (\text{s chopstar } f \wedge \text{more})$

proof –

have 1: $\vdash (\text{s chopstar } f \wedge \text{more}) = (f \wedge \text{more}) \frown \text{s chopstar } f$
by (*rule SCSAndMoreEqvAndMoreSChop*)

have 2: $\vdash \text{empty} \vee \text{more}$

by (*auto simp: empty-d-def*)

hence 3: $\vdash \text{s chopstar } f \longrightarrow \text{empty} \vee (\text{s chopstar } f \wedge \text{more})$
by *auto*

hence 4: $\vdash (f \wedge \text{more}) \frown \text{s chopstar } f \longrightarrow (f \wedge \text{more}) \vee ((f \wedge \text{more}) \frown (\text{s chopstar } f \wedge \text{more}))$
using *SChopEmptyOrImpRule*

by (*metis* 1 *AndMoreAndFiniteEqvAndFmore SCSAndMoreEqvAndFMoreSChop inteq-reflection*)

hence 5: $\vdash (f \wedge \text{more}) \frown \text{s chopstar } f \wedge \neg(f \wedge \text{more}) \longrightarrow ((f \wedge \text{more}) \frown (\text{s chopstar } f \wedge \text{more}))$
by *fastforce*

have 6: $\vdash (f \wedge \text{more}) \frown \text{s chopstar } f = ((f \wedge \text{more}) \frown \text{s chopstar } f \wedge \text{more})$ **using** 1
by *auto*

have 7: $\vdash ((f \wedge \text{more}) \frown \text{s chopstar } f \wedge \neg(f \wedge \text{more})) =$
 $((f \wedge \text{more}) \frown \text{s chopstar } f \wedge \text{more} \wedge \neg(f \wedge \text{more}))$

using 6 **by** *auto*

have 8: $\vdash (f \wedge \text{more}) \frown \text{s chopstar } f \wedge \text{more} \wedge \neg f \longrightarrow (f \wedge \text{more}) \frown (\text{s chopstar } f \wedge \text{more})$
using 5 7 **by** *auto*

have 9: $\vdash (\text{s chopstar } f \wedge \text{more} \wedge \neg f) = ((\text{s chopstar } f \wedge \text{more}) \wedge (\text{more} \wedge \neg f))$
by *auto*

have 10: $\vdash ((\text{s chopstar } f \wedge \text{more}) \wedge (\text{more} \wedge \neg f)) =$
 $((f \wedge \text{more}) \frown \text{s chopstar } f \wedge (\text{more} \wedge \neg f))$

using 1 **by** *fastforce*

from 1 8 9 10 **show** *?thesis* **by** *fastforce*

qed

lemma *SChopplusCommuteImpA*:

$\vdash \text{s chopstar } f \frown (f \wedge \text{finite}) \longrightarrow f \frown \text{s chopstar } f$

by (*metis SChopstarEqvSPowerstar SPowerStarCommute int-iffD1 inteq-reflection spowerstar-d-def*)

lemma *SChopplusCommuteImpB*:

$\vdash f \frown \text{s chopstar } f \longrightarrow \text{s chopstar } f \frown (f \wedge \text{finite})$

by (*metis* *SChopstarEqvSPowerstar* *SPowerStarCommutate* *int-iffD2* *inteq-reflection* *spowerstar-d-def*)

lemma *SChopplusCommutate*:

$\vdash f \frown \text{schopstar } f = \text{schopstar } f \frown (f \wedge \text{finite})$

using *SChopplusCommutateImpA* *SChopplusCommutateImpB* **by** *fastforce*

lemma *SCSEqvOrChopSCSB*:

$\vdash \text{schopstar } f = (\text{empty} \vee (\text{schopstar } f \frown (f \wedge \text{finite})))$

by (*meson* *SCSEqvOrSChopSCS* *SChopplusCommutate* *Prop06*)

lemma *SCSAndMoreImpSCSSChop*:

$\vdash \text{schopstar } f \wedge \text{more} \longrightarrow \text{schopstar } f \frown (f \wedge \text{finite})$

using *SCSAndMoreEqvAndMoreSChop* *SChopplusCommutate* *SCSAndMoreImpSChopSCS* **by** *fastforce*

lemma *SPowerSChopSPower*:

$\vdash (\text{spower } (f \wedge \text{more}) \ n) \frown (\text{spower } (f \wedge \text{more}) \ k) = (\text{spower } (f \wedge \text{more}) \ (n+k))$

proof

(*induct* *n* *arbitrary*: *k*)

case *0*

then show ?*case* **by** (*metis* *EmptySChop* *add.left-neutral* *spow-0*)

next

case (*Suc* *n*)

then show ?*case*

by (*metis* *PowerChopPower* *PowerSpowerdef* *SPowerAndMoreAndFinite* *inteq-reflection* *schop-d-def*)

qed

lemma *SCSSChopSCS*:

$\vdash \text{schopstar } f \frown \text{schopstar } f = \text{schopstar } f$

proof –

have *1*: $\vdash \text{schopstar } f \frown \text{schopstar } f \longrightarrow \text{schopstar } f$

by (*metis* *Prop02* *Prop03* *SChopstarEqv* *SChopstarEqvSPowerstar* *SChopstarInductMoreL* *SPowerstarImpSChopstar* *inteq-reflection*)

have *2*: $\vdash \text{schopstar } f \longrightarrow \text{schopstar } f \frown \text{schopstar } f$

by (*metis* (*no-types*, *lifting*) *AndSFinEqvSChopAndEmpty* *DiamondEmptyEqvFinite* *EmptyImpSCS* *FiniteAndEmptyEqvEmpty* *SCSAndFinite* *SChopImpSChop* *SChopstarEqvSPowerstar* *SFinEqvTrueSChopAndEmpty* *SPowerstarImpSChopstar* *TrueSChopEqvDiamond* *inteq-reflection*)

from *1* *2* **show** ?*thesis* **by** *fastforce*

qed

lemma *NotSCSImpMore*:

$\vdash \neg (\text{schopstar } f) \longrightarrow \text{more}$

proof –

have *1*: $\vdash \text{empty} \longrightarrow \text{schopstar } f$ **using** *EmptyImpSCS* **by** *blast*

hence *2*: $\vdash \neg \text{empty} \vee \text{schopstar } f$ **by** *fastforce*

from *2* **show** ?*thesis* **using** *1* *NotEmptyEqvMore* **by** *fastforce*

qed

lemma *NotSCSAndInf*:

$\vdash \neg (\text{schopstar } f \wedge \text{inf})$

using *InfEqvNotFinite* *SCSAndFinite* **by** *fastforce*

lemma *SCSSChopSCSImpSCS*:

$\vdash (\text{schopstar } f \frown \text{schopstar } f) \longrightarrow \text{schopstar } f$
by (*simp add: SCSSChopSCS int-iffD1*)

lemma *ImpSChopPlus*:

$\vdash f \wedge \text{finite} \longrightarrow f \frown \text{schopstar } f$

proof –

have 1: $\vdash \text{schopstar } f = (\text{empty} \vee f \frown \text{schopstar } f)$ **by** (*rule SCSEqvOrSChopSCS*)
hence 2: $\vdash f \frown \text{schopstar } f = (f \frown \text{empty} \vee f \frown (f \frown \text{schopstar } f))$ **using** *SChopOrEqvRule* **by** *blast*
have 3: $\vdash \text{finite} \longrightarrow f \frown \text{empty} = f$ **using** *SChopEmpty* **by** *blast*
from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *ImpSCS*:

$\vdash f \wedge \text{finite} \longrightarrow \text{schopstar } f$

proof –

have 1: $\vdash f \wedge \text{finite} \longrightarrow f \frown \text{schopstar } f$ **by** (*rule ImpSChopPlus*)
hence 2: $\vdash f \wedge \text{finite} \longrightarrow \text{empty} \vee f \frown \text{schopstar } f$ **by** *auto*
from 2 **show** *?thesis* **using** *SCSEqvOrSChopSCS* **by** *fastforce*

qed

lemma *SCSSChopImpSCS*:

$\vdash \text{schopstar } f \frown (f \wedge \text{finite}) \longrightarrow \text{schopstar } f$

proof –

have 1: $\vdash f \wedge \text{finite} \longrightarrow \text{schopstar } f$ **by** (*rule ImpSCS*)
hence 2: $\vdash \text{schopstar } f \frown (f \wedge \text{finite}) \longrightarrow \text{schopstar } f \frown \text{schopstar } f$ **by** (*rule RightSChopImpSChop*)
hence 3: $\vdash \text{schopstar } f \frown (f \wedge \text{finite}) \longrightarrow \text{schopstar } f \frown \text{schopstar } f$ **by** *auto*
have 4: $\vdash \text{schopstar } f \frown \text{schopstar } f \longrightarrow \text{schopstar } f$ **by** (*rule SCSSChopSCSImpSCS*)
from 2 3 4 **show** *?thesis* **using** *lift-imp-trans* **by** *blast*

qed

lemma *SChopPlusImpSCS*:

$\vdash f \frown \text{schopstar } f \longrightarrow \text{schopstar } f$

proof –

have 1: $\vdash f \frown \text{schopstar } f \longrightarrow \text{empty} \vee f \frown \text{schopstar } f$ **by** *auto*
from 1 **show** *?thesis* **using** *SCSEqvOrSChopSCS* **by** *fastforce*

qed

lemma *SCSSChopEqvOrSChopPlusSChop*:

$\vdash \text{schopstar } f \frown g = (g \vee (f \frown \text{schopstar } f) \frown g)$

proof –

have 1: $\vdash \text{schopstar } f = (\text{empty} \vee f \frown \text{schopstar } f)$ **by** (*rule SCSEqvOrSChopSCS*)
from 1 **show** *?thesis* **using** *EmptyOrSChopEqvRule* **by** *blast*

qed

lemma *SCSElim*:

assumes $\vdash \text{empty} \longrightarrow g$
 $\vdash (f \wedge \text{more}) \frown g \longrightarrow g$
shows $\vdash \text{schopstar } f \longrightarrow g$

proof –
have 1: $\vdash \text{empty} \vee (f \wedge \text{more}) \frown g \longrightarrow g$
using *assms* **using** *Prop02* **by** *blast*
have 2: $\vdash (\text{schopstar } f) \frown \text{empty} \longrightarrow g$
using *SChopstarInductMoreL 1* **by** *blast*
from 2 **show** *?thesis*
by (*metis AndSFinEqvSChopAndEmpty DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty SCSAndFinite*
SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection)
qed

lemma *SChopstarImp*:
assumes $\vdash f \frown (\text{schopstar } g) \vee \text{empty} \longrightarrow (\text{schopstar } g)$
shows $\vdash (\text{schopstar } f) \longrightarrow (\text{schopstar } g)$
using *assms SChopstarInductL*[of *LIFT(empty) f LIFT(schopstar g)*]
SChopEmpty[of *LIFT(schopstar f)*]
by (*metis ChopEmpty SCSAndFinite int-eq int-simps(33) lift-and-com chop-d-def*)

lemma *SCSSCSImpSCS*:
 $\vdash \text{schopstar } (\text{schopstar } f) \longrightarrow \text{schopstar } f$
proof –
have 1: $\vdash ((\text{schopstar } f) \frown (\text{schopstar } f)) \vee \text{empty} \longrightarrow (\text{schopstar } f)$
by (*meson SCSSChopSCSImpSCS EmptyImpSCS Prop02*)
from 1 **show** *?thesis* **using** *SChopstarImp* **by** *blast*
qed

lemma *SCSImpSCSSCS*:
 $\vdash \text{schopstar } f \longrightarrow \text{schopstar } (\text{schopstar } f)$
using *ImpSCS* **by** (*metis SCSAndFinite inteq-reflection*)

lemma *SCSSCSEqvSCS*:
 $\vdash \text{schopstar } (\text{schopstar } f) = \text{schopstar } f$
by (*simp add: SCSSCSImpSCS SCSImpSCSSCS int-iffI*)

lemma *RightEmptyOrSChopEqv*:
 $\vdash g \frown (\text{empty} \vee f) = ((g \wedge \text{finite}) \vee (g \frown f))$
proof –
have 1: $\vdash g \frown (\text{empty} \vee f) = (g \frown \text{empty} \vee g \frown f)$ **by** (*rule SChopOrEqv*)
have 2: $\vdash \text{finite} \longrightarrow g \frown \text{empty} = g$ **by** (*rule SChopEmpty*)
from 1 2 **show** *?thesis* **by** (*simp add: RightEmptyOrChopEqv chop-d-def*)
qed

lemma *RightEmptyOrSChopEqvRule*:
assumes $\vdash f = (\text{empty} \vee f1)$
shows $\vdash g \frown f = ((g \wedge \text{finite}) \vee (g \frown f1))$
proof –
have 1: $\vdash f = (\text{empty} \vee f1)$ **using** *assms* **by** *auto*
hence 2: $\vdash g \frown f = g \frown (\text{empty} \vee f1)$ **by** (*rule RightSChopEqvSChop*)
have 3: $\vdash g \frown (\text{empty} \vee f1) = ((g \wedge \text{finite}) \vee (g \frown f1))$ **by** (*rule RightEmptyOrSChopEqv*)
from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *SChopPlusEqvOrSChopSChopPlus*:

$\vdash (f \frown \text{schopstar } f) = ((f \wedge \text{finite}) \vee f \frown (f \frown \text{schopstar } f))$

proof –

have 1: $\vdash \text{schopstar } f = (\text{empty} \vee f \frown \text{schopstar } f)$ **by** (rule *SCSEqvOrSChopSCS*)

from 1 **show** ?thesis **by** (rule *RightEmptyOrSChopEqvRule*)

qed

lemma *SCSAndEmptyEqvEmpty*:

$\vdash (\text{schopstar } f \wedge \text{empty}) = \text{empty}$

using *EmptyImpSCS* **by** fastforce

lemma *NotAndMoreSChopAndEmpty*:

$\vdash \neg(((f \wedge \text{more}) \frown g) \wedge \text{empty})$

by (metis *LeftSChopImpMoreRule Prop05 Prop12 Prop13 empty-d-def int-iffD1 int-simps(15) inteq-reflection lift-and-com*)

lemma *NotSChopAndMoreAndEmpty*:

$\vdash \neg((f \frown (g \wedge \text{more})) \wedge \text{empty})$

by (simp add: *NotChopAndMoreAndEmpty chop-d-def*)

lemma *SChopSCSAndEmptyEqvAndEmpty*:

$\vdash ((f \frown \text{schopstar } f) \wedge \text{empty}) = (f \wedge \text{empty})$

proof –

have 1: $\vdash ((f \frown \text{schopstar } f) \wedge \text{empty}) = (f \wedge \text{empty}) \frown (\text{schopstar } f \wedge \text{empty})$

using *SChopAndEmptyEqvEmptySChopEmpty* **by** blast

have 2: $\vdash (f \wedge \text{empty}) \frown (\text{schopstar } f \wedge \text{empty}) = (f \wedge \text{empty}) \frown \text{empty}$

using *SCSAndEmptyEqvEmpty* **using** *RightSChopEqvSChop* **by** blast

have 3: $\vdash (f \wedge \text{empty}) \frown \text{empty} = (f \wedge \text{empty})$

by (metis *AndChopA AndEmptySChopAndEmptyEqvAndEmpty ChopEmpty Prop11 SC chopAndB inteq-reflection chop-d-def*)

show ?thesis

using 2 3 *SChopAndEmptyEqvEmptySChopEmpty* **by** fastforce

qed

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 *AndFmoreOrAndEmptyEqvAndFinite*:

$\vdash ((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{empty})) = (f \wedge \text{finite})$

proof –

have 1: $\vdash (f \wedge \text{more} \wedge \text{finite}) \longrightarrow (f \wedge \text{finite})$

by fastforce

have 2: $\vdash (f \wedge \text{empty}) \longrightarrow (f \wedge \text{finite})$

using *FiniteAndEmptyEqvEmpty* **by** auto

have 3: $\vdash (f \wedge \text{finite}) \longrightarrow ((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{empty}))$

by (simp add: *empty-d-def intI*)

show ?thesis **by** (meson 1 2 3 *Prop02 int-iffI*)

qed

lemma *SChopPlusEqv*:

$\vdash (f \frown \text{schopstar } f) = ((f \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f))$

proof –

have 1: $\vdash \text{schopstar } f = (\text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f)$

by (*rule* *SChopstarEqv*)

have 2: $\vdash \text{schopstar } f = (\text{empty} \vee f \frown \text{schopstar } f)$

by (*rule* *SCSEqvOrSChopSCS*)

hence 3: $\vdash (\text{empty} \vee f \frown \text{schopstar } f) = (\text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f)$

using 1 2 **by** *fastforce*

have 4: $\vdash (f \wedge \text{more}) \frown \text{schopstar } f = (f \wedge \text{more}) \frown (\text{empty} \vee f \frown \text{schopstar } f)$

using 2 **using** *RightSChopEqvSChop* **by** *blast*

hence 5: $\vdash \text{empty} \vee f \frown \text{schopstar } f = \text{empty} \vee (f \wedge \text{more}) \frown (\text{empty} \vee f \frown \text{schopstar } f)$

using 3 4 **by** *fastforce*

have 6: $\vdash (f \wedge \text{more}) \frown (\text{empty} \vee f \frown \text{schopstar } f) =$

$((f \wedge \text{more}) \frown \text{empty} \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f))$

using *SChopOrEqv* **by** *blast*

have 7: $\vdash (f \wedge \text{more}) \frown \text{empty} = (f \wedge \text{more} \wedge \text{finite})$

by (*metis* *AndMoreAndFiniteEqvAndFmore ChopEmpty fmore-d-def inteq-reflection schop-d-def*)

have 8: $\vdash (\text{empty} \vee f \frown \text{schopstar } f) =$

$(\text{empty} \vee (f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f))$

using 5 6 7 **by** (*metis* 2 3 *inteq-reflection*)

have 9: $\vdash (f \frown \text{schopstar } f \wedge \text{more}) = ((\text{empty} \vee f \frown \text{schopstar } f) \wedge \text{more})$

by (*auto simp: empty-d-def*)

have 91: $\vdash (f \frown \text{schopstar } f \wedge \text{more}) =$

$((\text{empty} \vee (f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f)) \wedge \text{more})$

using 8 9 **by** *fastforce*

have 10: $\vdash ((\text{empty} \vee (f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f)) \wedge \text{more}) =$

$((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f)) \wedge \text{more})$

by (*auto simp: empty-d-def*)

have 101: $\vdash (f \frown \text{schopstar } f \wedge \text{more}) =$

$((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f)) \wedge \text{more})$

by (*metis* 10 91 *inteq-reflection*)

have 11: $\vdash (((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f)) \wedge \text{more}) =$

$((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f))$

using 10 6 7 *int-eq*

using *AndMoreSChopAndMoreEqvAndMoreSChop* **by** *fastforce*

have 12: $\vdash (f \frown \text{schopstar } f \wedge \text{more}) = ((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f))$

using 101 11 **by** *fastforce*

have 13: $\vdash (f \frown \text{schopstar } f \wedge \text{empty}) = (f \wedge \text{empty})$

by (*rule* *SChopSCSAndEmptyEqvAndEmpty*)

have 14: $\vdash ((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f) \vee (f \wedge \text{empty})) =$

$((f \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f))$

using *AndFmoreOrAndEmptyEqvAndFinite*[of *f*] **by** *auto*

have 15: $\vdash f \frown \text{schopstar } f = ((f \frown \text{schopstar } f \wedge \text{more}) \vee (f \frown \text{schopstar } f \wedge \text{empty}))$

by (*auto simp: empty-d-def*)

have 16: $\vdash ((f \frown \text{schopstar } f \wedge \text{more}) \vee (f \frown \text{schopstar } f \wedge \text{empty})) =$

$((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f)) \vee (f \wedge \text{empty})$

using 12 13 **by** (*metis* 15 *int-eq*)

have 17: $\vdash f \frown \text{schopstar } f =$
 $(((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f)) \vee (f \wedge \text{empty}))$
by (metis 15 16 inteq-reflection)
show ?thesis **using** 17 14 **by** fastforce
qed

lemma *SChopSChopPlusImpSChopPlus*:
 $\vdash f \frown (f \frown \text{schopstar } f) \longrightarrow f \frown \text{schopstar } f$
proof –
have 1: $\vdash \text{empty} \vee \text{more}$
by (auto simp: empty-d-def)
hence 2: $\vdash f \longrightarrow \text{empty} \vee (f \wedge \text{more})$
by auto
hence 3: $\vdash f \frown (f \frown \text{schopstar } f) \longrightarrow (f \frown \text{schopstar } f) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f)$
by (rule EmptyOrSChopImpRule)
have 4: $\vdash f \frown \text{schopstar } f = ((f \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f))$
by (rule SChopPlusEqv)
hence 5: $\vdash (f \wedge \text{more}) \frown (f \frown \text{schopstar } f) \longrightarrow f \frown \text{schopstar } f$
by auto
from 3 5 **show** ?thesis **using** *SChopPlusImpSCS RightSChopImpSChop* **by** blast
qed

lemma *SCSImpSCS*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \text{schopstar } f \longrightarrow \text{schopstar } g$
using *assms*
by (metis *AndSChopB EmptyImpSCS Prop02 Prop10 SChopPlusImpSCS SChopstarImp*
inteq-reflection lift-imp-trans)

lemma *SChopPlusImpSChopPlus*:
assumes $\vdash f \longrightarrow g$
shows $\vdash f \frown \text{schopstar } f \longrightarrow g \frown \text{schopstar } g$
using *assms* **by** (simp add: *SCSImpSCS SChopImpSChop*)

lemma *SChopPlusIntro*:
assumes $\vdash f \longrightarrow (g \wedge \text{finite}) \vee (g \wedge \text{more}) \frown f$
shows $\vdash f \wedge \text{finite} \longrightarrow g \frown \text{schopstar } g$
proof –
have 1: $\vdash f \wedge \neg (g \wedge \text{finite}) \longrightarrow (g \wedge \text{more}) \frown f$
using *assms* **by** auto
have 2: $\vdash g \frown \text{schopstar } g = ((g \wedge \text{finite}) \vee (g \wedge \text{more}) \frown (g \frown \text{schopstar } g))$
by (rule *SChopPlusEqv*)
have 3: $\vdash f \wedge \neg (g \frown \text{schopstar } g) \longrightarrow$
 $(g \wedge \text{more}) \frown f \wedge \neg ((g \wedge \text{more}) \frown (g \frown \text{schopstar } g))$ **using** 1 2
by fastforce
have 4: $\vdash g \wedge \text{more} \longrightarrow \text{more}$
by auto
from 3 4 **show** ?thesis **using** *SChopContraB* **by** blast
qed

lemma *SChopPlusElim*:
assumes $\vdash f \longrightarrow g$
 $\vdash (f \wedge \text{more}) \frown g \longrightarrow g$
shows $\vdash f \frown \text{schopstar } f \longrightarrow g$
proof –
have 1: $\vdash f \vee (f \wedge \text{more}) \frown g \longrightarrow g$
using *assms Prop02* **by** *blast*
have 2: $\vdash \text{schopstar } f \frown f \longrightarrow g$
using *SChopstarInductMoreL 1* **by** *blast*
from 2 **show** *?thesis*
using *SChopplusCommute* **by** (*metis Prop10 Prop12 SChopAndA inteq-reflection*)
qed

lemma *SChopPlusElimWithoutMore*:
assumes $\vdash f \longrightarrow g$
 $\vdash f \frown g \longrightarrow g$
shows $\vdash f \frown \text{schopstar } f \longrightarrow g$
proof –
have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *blast*
have 2: $\vdash (f \frown g) \longrightarrow g$ **using** *assms* **by** *blast*
have 3: $\vdash (f \wedge \text{more}) \frown g \longrightarrow f \frown g$ **by** (*rule AndSChopA*)
have 4: $\vdash (f \wedge \text{more}) \frown g \longrightarrow g$ **using** 2 3 *lift-imp-trans* **by** *blast*
from 1 4 **show** *?thesis* **using** *SChopPlusElim* **by** *blast*
qed

lemma *SChopPlusEqvSChopPlus*:
assumes $\vdash f = g$
shows $\vdash f \frown \text{schopstar } f = g \frown \text{schopstar } g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash f \longrightarrow g$ **by** *auto*
hence 3: $\vdash f \frown \text{schopstar } f \longrightarrow g \frown \text{schopstar } g$ **by** (*rule SChopPlusImpSChopPlus*)
have 4: $\vdash g \longrightarrow f$ **using** 1 **by** *auto*
hence 5: $\vdash g \frown \text{schopstar } g \longrightarrow f \frown \text{schopstar } f$ **by** (*rule SChopPlusImpSChopPlus*)
from 3 5 **show** *?thesis* **by** *fastforce*
qed

lemma *SCSEqvSCS*:
assumes $\vdash f = g$
shows $\vdash \text{schopstar } f = \text{schopstar } g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash f \frown \text{schopstar } f = g \frown \text{schopstar } g$ **by** (*rule SChopPlusEqvSChopPlus*)
hence 3: $\vdash (\text{empty} \vee f \frown \text{schopstar } f) = (\text{empty} \vee g \frown \text{schopstar } g)$ **by** *auto*
from 3 **show** *?thesis* **using** *SCSEqvOrSChopSCS* **by** (*metis int-eq*)
qed

lemma *AndSCSA*:
 $\vdash \text{schopstar } (f \wedge g) \longrightarrow \text{schopstar } f$
proof –

have 1: $\vdash f \wedge g \longrightarrow f$ **by** *auto*
from 1 **show** *?thesis* **using** *SCSImpSCS* **by** *blast*
qed

lemma *AndSCSB*:

$\vdash \text{schopstar } (f \wedge g) \longrightarrow \text{schopstar } g$
proof –
have 1: $\vdash f \wedge g \longrightarrow g$ **by** *auto*
from 1 **show** *?thesis* **using** *SCSImpSCS* **by** *blast*
qed

lemma *SCSIntro*:

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: $\vdash f \wedge \text{more} \longrightarrow (g \wedge \text{more}) \frown f$
using *assms* **by** *auto*
have 2: $\vdash \text{more} = (\neg \text{empty})$
by (*auto simp: empty-d-def*)
have 3: $\vdash f \wedge \neg \text{empty} \longrightarrow (g \wedge \text{more}) \frown f$
using 1 2 **by** *fastforce*
have 4: $\vdash \text{schopstar } g = (\text{empty} \vee (g \wedge \text{more}) \frown \text{schopstar } g)$
by (*rule SChopstarEqv*)
hence 41: $\vdash (\neg(\text{empty} \vee (g \wedge \text{more}) \frown \text{schopstar } g)) = (\neg \text{empty} \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g))$
by *fastforce*
have 411: $\vdash (\neg \text{empty} \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g)) = (\text{more} \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g))$
using *NotEmptyEqvMore* **by** *fastforce*
have 42: $\vdash \neg(\text{schopstar } g) = (\text{more} \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g))$
using 4 41 411 **by** *fastforce*
have 43: $\vdash f \wedge \neg(\text{schopstar } g) \longrightarrow f \wedge \text{more} \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g)$
using 42 **by** *fastforce*
have 44: $\vdash f \wedge \text{more} \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g) \longrightarrow$
 $(g \wedge \text{more}) \frown f \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g)$
using 3 43 1 **by** *auto*
have 5: $\vdash f \wedge \neg(\text{schopstar } g) \longrightarrow$
 $(g \wedge \text{more}) \frown f \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g)$
using 43 44 *lift-imp-trans* **by** *fastforce*
have 6: $\vdash g \wedge \text{more} \longrightarrow \text{more}$
by *auto*
from 5 6 **show** *?thesis* **using** *SChopContraB* **by** *blast*
qed

lemma *SCSElimWithoutMore*:

assumes $\vdash \text{empty} \longrightarrow g$
 $\vdash f \frown g \longrightarrow g$
shows $\vdash \text{schopstar } f \longrightarrow g$
proof –
have 1: $\vdash \text{empty} \longrightarrow g$ **using** *assms* **by** *blast*
have 2: $\vdash f \frown g \longrightarrow g$ **using** *assms* **by** *blast*
have 3: $\vdash (f \wedge \text{more}) \frown g \longrightarrow f \frown g$ **by** (*rule AndSChopA*)

have 4: $\vdash (f \wedge \text{more}) \frown g \longrightarrow g$ using 2 3 lift-imp-trans by blast
 from 1 4 show ?thesis using SCSElim by blast
 qed

lemma *SChopAssocB*:

$\vdash (f \frown g) \frown h = f \frown (g \frown h)$
 using *SChopAssoc* by fastforce

lemma *SCSChopEqvSCHopOrRule*:

assumes $\vdash f = (\text{schopstar } g \frown h)$
 shows $\vdash f = ((g \frown f) \vee h)$
 proof –
 have 1: $\vdash f = (\text{schopstar } g \frown h)$ using assms by auto
 have 2: $\vdash \text{schopstar } g = (\text{empty} \vee (g \frown \text{schopstar } g))$ by (rule *SCSEqvOrSCHopSCS*)
 hence 3: $\vdash \text{schopstar } g \frown h = (h \vee ((g \frown \text{schopstar } g) \frown h))$ by (rule *EmptyOrSCHopEqvRule*)
 have 4: $\vdash (g \frown \text{schopstar } g) \frown h = g \frown (\text{schopstar } g \frown h)$ by (rule *SChopAssocB*)
 hence 41: $\vdash \text{schopstar } g \frown h = (h \vee g \frown (\text{schopstar } g \frown h))$ using 3 by fastforce
 have 5: $\vdash g \frown f = g \frown (\text{schopstar } g \frown h)$ using 1 by (rule *RightSCHopEqvSCHop*)
 hence 6: $\vdash (\text{schopstar } g \frown h) = (h \vee g \frown f)$ using 41 by fastforce
 hence 61: $\vdash (\text{schopstar } g \frown h) = ((g \frown f) \vee h)$ by auto
 from 1 61 show ?thesis by fastforce
 qed

lemma *SCSChopIntroRule*:

assumes $\vdash f \wedge \neg h \longrightarrow g \frown f$
 $\vdash g \longrightarrow \text{more}$
 shows $\vdash f \wedge \text{finite} \longrightarrow \text{schopstar } g \frown h$
 proof –
 have 1: $\vdash f \wedge \neg h \longrightarrow g \frown f$
 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: $\vdash g \frown f \longrightarrow (g \wedge \text{more}) \frown f$
 by (rule *LeftSCHopImpSCHop*)
 have 5: $\vdash f \longrightarrow (g \wedge \text{more}) \frown f \vee h$
 using 1 4 by fastforce
 have 6: $\vdash \text{schopstar } g = (\text{empty} \vee (g \wedge \text{more}) \frown \text{schopstar } g)$
 by (rule *SCHopstarEqv*)
 hence 7: $\vdash (\text{schopstar } g) \frown h = (h \vee ((g \wedge \text{more}) \frown \text{schopstar } g) \frown h)$
 by (rule *EmptyOrSCHopEqvRule*)
 have 8: $\vdash ((g \wedge \text{more}) \frown \text{schopstar } g) \frown h = (g \wedge \text{more}) \frown (\text{schopstar } g \frown h)$
 by (rule *SChopAssocB*)
 have 9: $\vdash (\text{schopstar } g) \frown h = (h \vee (g \wedge \text{more}) \frown (\text{schopstar } g \frown h))$
 using 7 8 by fastforce
 have 10: $\vdash f \wedge \neg (\text{schopstar } g \frown h) \longrightarrow (g \wedge \text{more}) \frown f \wedge \neg ((g \wedge \text{more}) \frown (\text{schopstar } g \frown h))$
 using 5 9 by fastforce
 have 11: $\vdash g \wedge \text{more} \longrightarrow \text{more}$
 by fastforce

from 10 11 show ?thesis using SChopContraB by blast
qed

lemma BoxImpTrueSChopAndEmpty:

$\vdash \Box f \wedge \text{finite} \longrightarrow \# \text{True} \neg (f \wedge \text{empty})$

by (metis BoxAndSChopImport DiamondEmptyEqvFinite TrueSChopEqvDiamond inteq-reflection)

lemma BoxInitAndMoreImpBoxInitAndMoreAndSFinInit:

$\vdash \Box (\text{init } w) \wedge \text{more} \wedge \text{finite} \longrightarrow (\Box (\text{init } w) \wedge \text{more}) \wedge \text{sfin } (\text{init } w)$

proof –

have 1: $\vdash \text{sfin } (\text{init } w) = \# \text{True} \neg (\text{init } w \wedge \text{empty})$ **using SFinEqvTrueSChopAndEmpty by blast**

have 2: $\vdash \Box (\text{init } w) \wedge \text{finite} \longrightarrow \# \text{True} \neg (\text{init } w \wedge \text{empty})$ **by (rule BoxImpTrueSChopAndEmpty)**

from 1 2 show ?thesis by fastforce

qed

lemma SCSImpBox:

assumes $\vdash f \longrightarrow \text{empty} \vee ((\Box (\text{init } w) \wedge \text{more}) \neg f$

shows $\vdash (\text{init } w \wedge f) \wedge \text{finite} \longrightarrow \Box (\text{init } w)$

proof –

have 1: $\vdash f \longrightarrow \text{empty} \vee ((\Box (\text{init } w) \wedge \text{more}) \neg f$

using assms by auto

have 2: $\vdash \text{init } w \wedge \neg (\Box (\text{init } w)) \longrightarrow \neg \text{empty}$

by (rule InitAndNotBoxInitImpNotEmpty)

have 3: $\vdash \text{init } w \wedge f \wedge \neg (\Box (\text{init } w)) \longrightarrow ((\Box (\text{init } w) \wedge \text{more}) \neg f$

using 1 2 by fastforce

have 4: $\vdash \Box (\text{init } w) \wedge \text{more} \wedge \text{finite} \longrightarrow (\Box (\text{init } w) \wedge \text{more}) \wedge \text{sfin } (\text{init } w)$

by (rule BoxInitAndMoreImpBoxInitAndMoreAndSFinInit)

have 41: $\vdash (\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite} \longrightarrow ((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfin } (\text{init } w)$

using 4 by auto

hence 5: $\vdash ((\Box (\text{init } w) \wedge \text{more}) \neg f \longrightarrow (((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfin } (\text{init } w)) \neg f$

by (metis FiniteImp LeftChopImpChop inteq-reflection schop-d-def)

have 6: $\vdash (((\Box (\text{init } w) \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfin } (\text{init } w)) \neg f =$

$((\Box (\text{init } w) \wedge \text{more}) \neg (\text{init } w \wedge f))$

using AndSFinSChopEqvStateAndSChop

by (metis (no-types, lifting) 41 Prop10 Prop12 int-eq schop-d-def)

have 7: $\vdash \neg (\Box (\text{init } w)) \longrightarrow (\Box (\text{init } w)) \text{syields } (\neg (\Box (\text{init } w)))$

by (rule NotBoxStateImpBoxSYieldsNotBox)

have 8: $\vdash (\Box (\text{init } w)) \text{syields } (\neg (\Box (\text{init } w))) \longrightarrow$

$((\Box (\text{init } w) \wedge \text{more}) \text{syields } (\neg (\Box (\text{init } w))))$

using AndSYieldsA by (metis)

have 9: $\vdash ((\Box (\text{init } w) \wedge \text{more}) \neg (\text{init } w \wedge f) \wedge ((\Box (\text{init } w) \wedge \text{more}) \text{syields } (\neg (\Box (\text{init } w))))$

\longrightarrow

$((\Box (\text{init } w) \wedge \text{more}) \neg ((\text{init } w \wedge f) \wedge \neg (\Box (\text{init } w))))$

by (rule SChopAndSYieldsImp)

have 10: $\vdash (\text{init } w \wedge f) \wedge \neg (\Box (\text{init } w)) \longrightarrow$

$((\Box (\text{init } w) \wedge \text{more}) \neg ((\text{init } w \wedge f) \wedge \neg (\Box (\text{init } w))))$

using 3 5 6 7 8 9 by fastforce

have 11: $\vdash ((\Box (\text{init } w) \wedge \text{more}) \neg ((\text{init } w \wedge f) \wedge \neg (\Box (\text{init } w)))) \longrightarrow$

$\text{more} \neg ((\text{init } w \wedge f) \wedge \neg (\Box (\text{init } w)))$

using AndSChopB by blast

have 12: $\vdash (init\ w \wedge f) \wedge \neg(\Box (init\ w)) \longrightarrow$
 $more \frown ((init\ w \wedge f) \wedge \neg(\Box (init\ w)))$
using 10 11 **by** *fastforce*
from 12 **show** *?thesis* **using** *MoreSChopContra* **by** *blast*
qed

lemma *BoxSCSEqvBox*:

$\vdash (init\ w \wedge schopstar (\Box (init\ w))) = (\Box (init\ w) \wedge finite)$

proof –

have 1: $\vdash \Box (init\ w) \frown (\Box (init\ w) \wedge finite) \longrightarrow \Box (init\ w) \wedge finite$
by (*metis* *BoxStateAndChopEqvChop* *FiniteChopFiniteEqvFinite* *int-iffD2* *inteq-reflection* *s chop-d-def*)
have 2: $\vdash (init\ w \wedge empty) \longrightarrow \Box (init\ w) \wedge finite$
using *EmptyImpFinite* *StateAndEmptyImpBoxState* **by** *fastforce*
have 3: $\vdash (init\ w \wedge empty) \vee \Box (init\ w) \frown (\Box (init\ w) \wedge finite) \longrightarrow \Box (init\ w) \wedge finite$
using 1 2 **by** *fastforce*
have 4: $\vdash (init\ w \wedge empty) \frown schopstar (\Box (init\ w)) \longrightarrow \Box (init\ w) \wedge finite$
using *SChopstarInductR* 3
by (*metis* (*no-types*, *lifting*) *BoxBoxImpBox* *BoxEqvBoxBox* *BoxStateSChopBoxEqvBox* *Prop02* *Prop12*
SCSAndFinite *SCSImpSCSSCS* *SChopImpFinite* *StateAndEmptyImpBoxState* *int-eq*)
have 5: $\vdash init\ w \wedge schopstar (\Box (init\ w)) \longrightarrow \Box (init\ w) \wedge finite$
using 4 *StateAndEmptySChop* **by** *fastforce*
have 11: $\vdash \Box (init\ w) \longrightarrow (init\ w)$
using *BoxElim* **by** *blast*
have 12: $\vdash \Box (init\ w) \wedge finite \longrightarrow schopstar (\Box (init\ w))$
by (*rule* *ImpSCS*)
have 13: $\vdash \Box (init\ w) \wedge finite \longrightarrow init\ w \wedge schopstar (\Box (init\ w))$
using 11 12 **by** *fastforce*
from 5 13 **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateAndSCSEqvSCS*:

$\vdash (\Box (init\ w) \wedge schopstar f) = (init\ w \wedge schopstar (\Box (init\ w) \wedge f))$

proof –

have 1: $\vdash \Box (init\ w) \longrightarrow init\ w$
using *BoxElim* **by** *blast*
have 2: $\vdash (s chopstar f \wedge more) = (f \wedge more) \frown schopstar f$
by (*rule* *SCSAndMoreEqvAndMoreSChop*)
have 21: $\vdash (f \wedge more) \frown schopstar f = (finite \wedge (f \wedge more) \frown schopstar f)$
by (*metis* 2 *AndMoreAndFiniteEqvAndFmore* *SCSAndMoreEqvAndFMoreSChop* *inteq-reflection*
lift-and-com)
have 22: $\vdash (\Box (init\ w) \wedge (f \wedge more) \frown schopstar f) =$
 $(\Box (init\ w) \wedge finite \wedge (f \wedge more) \frown schopstar f)$

using 21 **by** *auto*
have 23: $\vdash ((\Box (init\ w) \wedge schopstar f) \wedge finite) = (\Box (init\ w) \wedge schopstar f)$
using *SCSAndFinite* **by** *fastforce*
have 24: $\vdash (\Box (init\ w) \wedge f \wedge more) \frown (\Box (init\ w) \wedge schopstar f) =$
 $((\Box (init\ w) \wedge f \wedge more) \frown (\Box (init\ w) \wedge schopstar f) \wedge finite)$
by (*meson* 23 *Prop10* *Prop12* *SChopImpFinite* *int-iffD2*)
have 3: $\vdash (\Box (init\ w) \wedge ((f \wedge more) \frown schopstar f)) =$

$((\Box (init\ w) \wedge f \wedge more) \frown (\Box (init\ w) \wedge schopstar\ f))$
by (*metis 22 24 BoxStateAndSChopEqvSChop int-eq*)
have 4: $\vdash \Box (init\ w) \wedge f \wedge more \longrightarrow (\Box (init\ w) \wedge f) \wedge more$
by *auto*
hence 5: $\vdash (\Box (init\ w) \wedge f \wedge more) \frown (\Box (init\ w) \wedge schopstar\ f) \longrightarrow$
 $((\Box (init\ w) \wedge f) \wedge more) \frown (\Box (init\ w) \wedge schopstar\ f)$
by (*rule LeftSChopImpSChop*)
have 6: $\vdash (\Box (init\ w) \wedge schopstar\ f) \wedge more \longrightarrow$
 $((\Box (init\ w) \wedge f) \wedge more) \frown (\Box (init\ w) \wedge schopstar\ f)$
using 2 3 5 by *fastforce*
hence 7: $\vdash (\Box (init\ w) \wedge schopstar\ f) \wedge finite \longrightarrow schopstar (\Box (init\ w) \wedge f)$
using *SCSIntro* **by** *blast*
have 70: $\vdash \Box (init\ w) \wedge schopstar\ f \longrightarrow schopstar (\Box (init\ w) \wedge f)$
using *SCSAndFinite 7* **using** *Valid-def* **by** *fastforce*
have 71: $\vdash init\ w \wedge \Box (init\ w) \wedge schopstar\ f \longrightarrow init\ w \wedge schopstar (\Box (init\ w) \wedge f)$
using 70 *SCSAndFinite* **by** *fastforce*
have 8: $\vdash \Box (init\ w) \wedge schopstar\ f \longrightarrow init\ w \wedge schopstar (\Box (init\ w) \wedge f)$
using 1 71 by *fastforce*
have 11: $\vdash schopstar (\Box (init\ w) \wedge f) \longrightarrow schopstar (\Box (init\ w))$
by (*rule AndSCSA*)
have 12: $\vdash (init\ w \wedge schopstar (\Box (init\ w))) = (\Box (init\ w) \wedge finite)$
by (*rule BoxSCSEqvBox*)
have 13: $\vdash schopstar (\Box (init\ w) \wedge f) \longrightarrow schopstar\ f$
by (*rule AndSCSB*)
have 14: $\vdash init\ w \wedge schopstar (\Box (init\ w) \wedge f) \longrightarrow init\ w \wedge schopstar (\Box (init\ w)) \wedge schopstar\ f$
using 11 13 by *fastforce*
have 15: $\vdash init\ w \wedge schopstar (\Box (init\ w)) \wedge schopstar\ f \longrightarrow \Box (init\ w) \wedge schopstar\ f$
using 12 by *auto*
have 16: $\vdash init\ w \wedge schopstar (\Box (init\ w) \wedge f) \longrightarrow \Box (init\ w) \wedge schopstar\ f$
using 14 15 *lift-imp-trans* **by** *blast*
from 8 16 show ?thesis by *fastforce*
qed

lemma SBaSCSImpSCS:

$\vdash sba\ (f \longrightarrow g) \longrightarrow schopstar\ f \longrightarrow schopstar\ g$

proof –

have 1: $\vdash schopstar\ f = (empty \vee (f \wedge more) \frown schopstar\ f)$

by (*rule SChopstarEqv*)

have 2: $\vdash schopstar\ g = (empty \vee (g \wedge more) \frown schopstar\ g)$

by (*rule SChopstarEqv*)

have 21: $\vdash (\neg(schopstar\ g)) = (\neg empty \wedge \neg((g \wedge more) \frown schopstar\ g))$

using 2 by *fastforce*

hence 22: $\vdash (\neg(schopstar\ g)) = (more \wedge \neg((g \wedge more) \frown schopstar\ g))$

using *NotEmptyEqvMore* **by** *fastforce*

have 3: $\vdash schopstar\ f \wedge \neg(schopstar\ g) \longrightarrow$

$(empty \vee (f \wedge more) \frown schopstar\ f) \wedge more \wedge \neg((g \wedge more) \frown schopstar\ g)$

using 1 22 by *fastforce*

have 31: $\vdash ((empty \vee (f \wedge more) \frown schopstar\ f) \wedge more) = ((f \wedge more) \frown schopstar\ f \wedge more)$

by (*auto simp: empty-d-def*)

have 32: $\vdash schopstar\ f \wedge \neg(schopstar\ g) \longrightarrow$

$(f \wedge \text{more}) \frown \text{schopstar } f \wedge \neg ((g \wedge \text{more}) \frown \text{schopstar } g)$
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: $\vdash \text{sba } (f \longrightarrow g) \longrightarrow \text{sba } (f \wedge \text{more} \longrightarrow g \wedge \text{more})$
by (*rule SBaImpSBa*)
have 6: $\vdash \text{sba } (f \wedge \text{more} \longrightarrow g \wedge \text{more}) \longrightarrow$
 $(f \wedge \text{more}) \frown \text{schopstar } f \longrightarrow (g \wedge \text{more}) \frown \text{schopstar } f$
by (*rule SBaLeftSCHopImpSCHop*)
have 7: $\vdash \text{sba } (f \longrightarrow g) \wedge (f \wedge \text{more}) \frown \text{schopstar } f \longrightarrow (g \wedge \text{more}) \frown \text{schopstar } f$
using 5 6 **by** *fastforce*
have 8: $\vdash (g \wedge \text{more}) \frown \text{schopstar } f \wedge \neg ((g \wedge \text{more}) \frown \text{schopstar } g)$
 $\longrightarrow (g \wedge \text{more}) \frown (\text{schopstar } f \wedge \neg (\text{schopstar } g))$
by (*rule SCHopAndNotSCHopImp*)
have 9: $\vdash (g \wedge \text{more}) \frown (\text{schopstar } f \wedge \neg (\text{schopstar } g)) \longrightarrow$
 $\text{more} \frown (\text{schopstar } f \wedge \neg (\text{schopstar } g))$
by (*rule AndSCHopB*)
have 10: $\vdash \text{sba } (f \longrightarrow g) \wedge \text{finite} \longrightarrow \text{more} \frown (\text{schopstar } f \wedge \neg (\text{schopstar } g)) \longrightarrow$
 $\text{more} \frown (\text{sba } (f \longrightarrow g) \wedge \text{schopstar } f \wedge \neg (\text{schopstar } g))$
using *SBaSCHopImpSCHopSBa* **by** *fastforce*
have 11: $\vdash \text{sba } (f \longrightarrow g) \wedge \text{finite} \wedge \text{schopstar } f \wedge \neg (\text{schopstar } g) \longrightarrow$
 $\text{more} \frown (\text{sba } (f \longrightarrow g) \wedge \text{schopstar } f \wedge \neg (\text{schopstar } g))$
using 32 7 8 9 10 **by** *fastforce*
have 12: $\vdash \text{sba } (f \longrightarrow g) \wedge \text{schopstar } f \wedge \neg (\text{schopstar } g) \longrightarrow$
 $\text{more} \frown (\text{sba } (f \longrightarrow g) \wedge \text{schopstar } f \wedge \neg (\text{schopstar } g))$
using 11 *SCSAndFinite* **by** *fastforce*
hence 12: $\vdash \text{finite} \longrightarrow \neg ((\text{sba } (f \longrightarrow g)) \wedge (\text{schopstar } f) \wedge (\neg (\text{schopstar } g)))$
using *MoreSCHopLoop* **by** *blast*
have 13: $\vdash (\text{sba } (f \longrightarrow g)) \wedge \text{finite} \wedge (\text{schopstar } f) \longrightarrow (\text{schopstar } g)$
using 12 **by** *fastforce*
have 14: $\vdash (\text{sba } (f \longrightarrow g)) \wedge (\text{schopstar } f) \longrightarrow (\text{schopstar } g)$
using *SCSAndFinite* 13 **by** *fastforce*
from 14 **show** ?thesis **by** *fastforce*
qed

lemma *SBaSCSEqvSCS*:

$\vdash \text{sba } (f = g) \longrightarrow (\text{schopstar } f = \text{schopstar } g)$

proof –

have 0: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$

by *fastforce*

have 1: $\vdash \text{sba } (f = g) = (\text{sba } (f \longrightarrow g) \wedge \text{sba } (g \longrightarrow f))$

by (*metis 0 SBaAndEqv int-eq*)

have 2: $\vdash \text{sba } (f \longrightarrow g) \longrightarrow (\text{schopstar } f \longrightarrow \text{schopstar } g)$

by (*rule SBaSCSImpSCS*)

have 3: $\vdash \text{sba } (g \longrightarrow f) \longrightarrow (\text{schopstar } g \longrightarrow \text{schopstar } f)$

by (*rule SBaSCSImpSCS*)

have 4: $\vdash \text{sba } (f = g) \longrightarrow (\text{schopstar } f \longrightarrow \text{schopstar } g) \wedge (\text{schopstar } g \longrightarrow \text{schopstar } f)$

using 1 2 3 **by** *fastforce*

have 5: $\vdash ((\text{schopstar } f \longrightarrow \text{schopstar } g) \wedge (\text{schopstar } g \longrightarrow \text{schopstar } f)) =$
 $(\text{schopstar } f = \text{schopstar } g)$

by auto
 from 4 5 show ?thesis by auto
 qed

lemma *SBaAndSCSImpImport*:

$\vdash \text{ sba } f \wedge \text{ schopstar } g \longrightarrow \text{ schopstar } (f \wedge g)$

proof –

have 1: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ by auto

hence 2: $\vdash \text{ sba } f \longrightarrow \text{ sba } (g \longrightarrow f \wedge g)$ by (rule *SBaImpSBa*)

have 3: $\vdash \text{ sba } (g \longrightarrow f \wedge g) \longrightarrow \text{ schopstar } g \longrightarrow \text{ schopstar } (f \wedge g)$ by (rule *SBaSCSImpSCS*)

from 2 3 show ?thesis by fastforce

qed

lemma *SCSSkipImpFinite*:

$\vdash \text{ schopstar skip} \longrightarrow \text{ finite}$

by (simp add: *EmptyImpFinite SCSElim SChopImpFinite*)

lemma *FiniteImpSCSSkip*:

$\vdash \text{ finite} \longrightarrow \text{ schopstar skip}$

using *AndMoreAndFiniteEqvAndFmore ChopAndB ChopEmpty*

FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite FmoreEqvSkipChopFinite SCSIIntro

by (metis (no-types, hide-lams) *EmptySChop FiniteAndEmptyEqvEmpty Prop10 int-eq schop-d-def*)

lemma *SCSSkipEqvFinite*:

$\vdash \text{ schopstar skip} = \text{ finite}$

using *SCSSkipImpFinite FiniteImpSCSSkip* by fastforce

7.8 Properties of Omega

lemma *SOmegaIntro*:

assumes $\vdash h \longrightarrow (f \wedge \text{ more}) \frown h$

shows $\vdash h \wedge \text{ inf} \longrightarrow f^\omega$

proof –

have 1: $\vdash h \longrightarrow (f \wedge \text{ more}) \frown h$ using *assms* by auto

have 2: $\vdash \Box (h \longrightarrow (f \wedge \text{ more}) \frown h)$ by (simp add: *BoxGen assms*)

from 1 2 show ?thesis using *SOmegaInduct* by fastforce

qed

7.9 Properties of SWhile

lemma *SWhileEqvIf*:

$\vdash \text{ swhile } (\text{ init } w) \text{ do } f = \text{ if}_i (\text{ init } w) \text{ then } (f \frown (\text{ swhile } (\text{ init } w) \text{ do } f)) \text{ else empty}$

proof –

have 1: $\vdash \text{ swhile } (\text{ init } w) \text{ do } f = (\text{ schopstar } ((\text{ init } w) \wedge f) \wedge \text{ sfin } (\neg (\text{ init } w)))$

by (simp add: *swhile-d-def*)

have 2: $\vdash \text{ schopstar } (\text{ init } w \wedge f) = (\text{ empty } \vee ((\text{ init } w \wedge f) \frown \text{ schopstar } (\text{ init } w \wedge f)))$

by (rule *SCSEqvOrSChopSCS*)

have 21: $\vdash (\text{ schopstar } ((\text{ init } w) \wedge f) \wedge \text{ sfin } (\neg (\text{ init } w))) =$

$((\text{ empty } \vee ((\text{ init } w \wedge f) \frown \text{ schopstar } (\text{ init } w \wedge f))) \wedge \text{ sfin } (\neg (\text{ init } w)))$

using 2 by fastforce

have 22: $\vdash ((\text{empty} \vee ((\text{init } w \wedge f) \frown \text{schopstar } (\text{init } w \wedge f))) \wedge \text{sfin } (\neg (\text{init } w))) =$
 $((\text{empty} \wedge \text{sfin } (\neg (\text{init } w))) \vee$
 $((\text{init } w \wedge f) \frown \text{schopstar } (\text{init } w \wedge f)) \wedge \text{sfin } (\neg (\text{init } w))))$
by auto
have 3: $\vdash (\text{empty} \wedge \text{sfin } (\neg (\text{init } w))) = (\neg (\text{init } w) \wedge \text{empty})$
by (metis Prop04 SFinAndEmpty lift-and-com)
have 4: $\vdash (\text{init } w \wedge f) \frown \text{schopstar } (\text{init } w \wedge f) = (\text{init } w \wedge (f \frown \text{schopstar } (\text{init } w \wedge f)))$
by (rule StateAndSCHop)
have 41: $\vdash ((\text{init } w \wedge f) \frown \text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w))) =$
 $(\text{init } w \wedge ((f \frown \text{schopstar } (\text{init } w \wedge f)) \wedge \text{sfin } (\neg (\text{init } w))))$
using 4 by auto
have 42: $\vdash (\text{init } w \wedge ((f \frown \text{schopstar } (\text{init } w \wedge f)) \wedge \text{sfin } (\neg (\text{init } w)))) =$
 $((\text{init } w \wedge (f \frown \text{schopstar } (\text{init } w \wedge f))) \wedge \text{sfin } (\neg (\text{init } w)))$
by (metis Initprop(2) int-simps(4) inteq-reflection)
have 5: $\vdash ((f \frown (\text{schopstar } (\text{init } w \wedge f))) \wedge (\text{sfin } (\neg (\text{init } w))))$
 $= (f \frown (\text{schopstar } (\text{init } w \wedge f) \wedge (\text{sfin } (\neg (\text{init } w)))))$
by (rule SCHopAndSFin)
have 51: $\vdash (f \frown (\text{schopstar } (\text{init } w \wedge f) \wedge (\text{sfin } (\neg (\text{init } w))))) =$
 $(f \frown (\text{schopstar } (\text{init } w \wedge f) \wedge (\text{sfin } (\neg (\text{init } w)))))$
by (metis Initprop(2) int-simps(1) inteq-reflection)
have 52: $\vdash (\text{init } w \wedge (f \frown \text{schopstar } (\text{init } w \wedge f)) \wedge \text{sfin } (\neg (\text{init } w))) =$
 $(\text{init } w \wedge (f \frown (\text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w)))))$
using SFinAndSCHop by fastforce
have 6: $\vdash (f \frown (\text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w)))) = f \frown \text{swhile } (\text{init } w) \text{ do } f$
by (simp add: swwhile-d-def)
have 61: $\vdash (\text{init } w \wedge ((f \frown (\text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w))))) =$
 $(\text{init } w \wedge (f \frown \text{swhile } (\text{init } w) \text{ do } f))$ **using 6**
by auto
have 611: $\vdash ((\text{init } w \wedge f) \frown \text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg \text{init } w)) =$
 $(\text{init } w \wedge (f \frown \text{swhile } (\text{init } w) \text{ do } f))$
by (metis 41 52 61 inteq-reflection)
have 62: $\vdash ((\text{empty} \wedge \text{sfin } (\neg (\text{init } w))) \vee$
 $((\text{init } w \wedge f) \frown \text{schopstar } (\text{init } w \wedge f)) \wedge \text{sfin } (\neg (\text{init } w)))$
 $= ((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f \frown \text{swhile } (\text{init } w) \text{ do } f)))$
by (metis 3 611 int-simps(1) inteq-reflection)
have 7: $\vdash \text{swhile } (\text{init } w) \text{ do } f$
 $= ((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f \frown \text{swhile } (\text{init } w) \text{ do } f)))$
proof –
have $\vdash \text{swhile } (\text{init } w) \text{ do } f = ((\text{empty} \vee (\text{init } w \wedge f) \frown \text{schopstar } (\text{init } w \wedge f)) \wedge \text{sfin } (\neg \text{init } w))$
by (simp add: 21 swwhile-d-def)
then show ?thesis
by (meson 22 62 Prop04 int-simps(20))
qed
have 71: $\vdash \text{if}_i (\text{init } w) \text{ then } (f \frown (\text{swhile } (\text{init } w) \text{ do } f)) \text{ else } \text{empty} =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f \frown \text{swhile } (\text{init } w) \text{ do } f)))$
by (auto simp: ifthenelse-d-def)
from 7 71 show ?thesis by fastforce
qed

lemma SWhileSCHopEqvIf:

$\vdash (\text{swwhile } (\text{init } w) \text{ do } f) \frown g = \text{if}_i (\text{init } w) \text{ then } (f \frown ((\text{swwhile } (\text{init } w) \text{ do } f) \frown g)) \text{ else } g$
proof –
have 1: $\vdash \text{swwhile } (\text{init } w) \text{ do } f =$
 $\text{if}_i (\text{init } w) \text{ then } (f \frown (\text{swwhile } (\text{init } w) \text{ do } f)) \text{ else } \text{empty}$
by (rule *SWhileEqvIf*)
hence 2: $\vdash (\text{swwhile } (\text{init } w) \text{ do } f) \frown g =$
 $\text{if}_i (\text{init } w) \text{ then } ((f \frown \text{swwhile } (\text{init } w) \text{ do } f) \frown g) \text{ else } (\text{empty} \frown g)$
by (rule *IfSCHopEqvRule*)
have 3: $\vdash \text{empty} \frown g = g$
by (rule *EmptySCHop*)
have 4: $\vdash \text{if}_i (\text{init } w) \text{ then } ((f \frown \text{swwhile } (\text{init } w) \text{ do } f) \frown g) \text{ else } (\text{empty} \frown g) =$
 $\text{if}_i (\text{init } w) \text{ then } ((f \frown \text{swwhile } (\text{init } w) \text{ do } f) \frown g) \text{ else } g$
using 3 **using** *inteq-reflection* **by** *fastforce*
have 5: $\vdash ((f \frown \text{swwhile } (\text{init } w) \text{ do } f) \frown g) = (f \frown (\text{swwhile } (\text{init } w) \text{ do } f \frown g))$
by (rule *SCHopAssocB*)
have 6: $\vdash \text{if}_i (\text{init } w) \text{ then } ((f \frown \text{swwhile } (\text{init } w) \text{ do } f) \frown g) \text{ else } g =$
 $\text{if}_i (\text{init } w) \text{ then } (f \frown ((\text{swwhile } (\text{init } w) \text{ do } f) \frown g)) \text{ else } g$
using 5 **using** *inteq-reflection* **by** *fastforce*
from 1 2 4 6 **show** ?thesis **by** *fastforce*
qed

lemma *SWhileSCHopEqvIfRule*:

assumes $\vdash f = (\text{swwhile } (\text{init } w) \text{ do } g) \frown h$
shows $\vdash f = \text{if}_i (\text{init } w) \text{ then } (g \frown f) \text{ else } h$
proof –
have 1: $\vdash f = (\text{swwhile } (\text{init } w) \text{ do } g) \frown h$
using *assms* **by** *auto*
have 2: $\vdash (\text{swwhile } (\text{init } w) \text{ do } g) \frown h =$
 $\text{if}_i (\text{init } w) \text{ then } (g \frown ((\text{swwhile } (\text{init } w) \text{ do } g) \frown h)) \text{ else } h$
by (rule *SWhileSCHopEqvIf*)
have 3: $\vdash (g \frown f) = (g \frown ((\text{swwhile } (\text{init } w) \text{ do } g) \frown h))$
using 1 **by** (rule *RightSCHopEqvSCHop*)
have 4: $\vdash (g \frown ((\text{swwhile } (\text{init } w) \text{ do } g) \frown h)) = (g \frown f)$
using 3 **by** *auto*
have 5: $\vdash \text{if}_i (\text{init } w) \text{ then } (g \frown ((\text{swwhile } (\text{init } w) \text{ do } g) \frown h)) \text{ else } h =$
 $\text{if}_i (\text{init } w) \text{ then } (g \frown f) \text{ else } h$
using 4 **using** *inteq-reflection* **by** *fastforce*
from 1 2 5 **show** ?thesis **by** *fastforce*
qed

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 *SWhileEqvEmptyOrSCHopSWhile*:

$\vdash \text{swwhile } (\text{init } w) \text{ do } f = ((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f \wedge \text{more}) \frown \text{swwhile } (\text{init } w) \text{ do } f))$
proof –

have 1: $\vdash \text{schopstar } (\text{init } w \wedge f) = (\text{empty} \vee ((\text{init } w \wedge f) \wedge \text{more}) \multimap \text{schopstar } (\text{init } w \wedge f))$
by (rule *SChopstarEqv*)
have 2: $\vdash ((\text{init } w \wedge f) \wedge \text{more}) = (\text{init } w \wedge (f \wedge \text{more}))$
by *auto*
hence 3: $\vdash ((\text{init } w \wedge f) \wedge \text{more}) \multimap \text{schopstar } (\text{init } w \wedge f) =$
 $(\text{init } w \wedge f \wedge \text{more}) \multimap \text{schopstar } (\text{init } w \wedge f)$
by (rule *LeftSChopEqvSChop*)
have 4: $\vdash \text{schopstar } (\text{init } w \wedge f) = (\text{empty} \vee (\text{init } w \wedge f \wedge \text{more}) \multimap \text{schopstar } (\text{init } w \wedge f))$
using 1 3 **by** *fastforce*
have 5: $\vdash (\text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w))) =$
 $((\text{empty} \wedge \text{sfin } (\neg (\text{init } w))) \vee$
 $((\text{init } w \wedge f \wedge \text{more}) \multimap \text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w))))$
using 1 4 **by** *fastforce*
have 6: $\vdash (\text{empty} \wedge \text{sfin } (\neg (\text{init } w))) = (\neg (\text{init } w) \wedge \text{empty})$
by (meson *Prop04 SFinAndEmpty lift-and-com*)
have 7: $\vdash (\text{init } w \wedge f \wedge \text{more}) \multimap \text{schopstar } (\text{init } w \wedge f) =$
 $(\text{init } w \wedge (f \wedge \text{more}) \multimap \text{schopstar } (\text{init } w \wedge f))$
by (rule *StateAndSChop*)
have 8: $\vdash (((f \wedge \text{more}) \multimap \text{schopstar } (\text{init } w \wedge f)) \wedge \text{sfin } (\text{init } (\neg w))) =$
 $((f \wedge \text{more}) \multimap (\text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\text{init } (\neg w))))$
by (rule *SChopAndSFin*)
have 81: $\vdash \text{sfin } (\text{init } (\neg w)) = \text{sfin } (\neg (\text{init } w))$
by (metis *Initprop(2) SFinStateEqvStateAndEmptyOrNextSFinState inteq-reflection*)
have 9: $\vdash (\text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w))) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge (f \wedge \text{more}) \multimap (\text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w)))))$
proof –
have f2: $\vdash (\text{empty} \wedge \text{sfin } (\neg \text{init } w)) = (\neg \text{init } w \wedge \text{empty})$
using 6 **by** *fastforce*
have $\vdash (\text{init } w \wedge f \wedge \text{more}) \multimap (\text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg \text{init } w)) =$
 $(\text{init } w \wedge (f \wedge \text{more}) \multimap (\text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg \text{init } w)))$
by (meson *StateAndSChop*)
then have $\vdash (((\text{init } w \wedge f \wedge \text{more}) \multimap \text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg \text{init } w)) =$
 $(\text{init } w \wedge (f \wedge \text{more}) \multimap (\text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg \text{init } w)))) =$
 $((\text{init } w \wedge f \wedge \text{more}) \multimap \text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\text{init } (\neg w))) =$
 $(\text{init } w \wedge f \wedge \text{more}) \multimap (\text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\text{init } (\neg w)))$
by (metis 81 *int-simps(1) inteq-reflection*)
then have $\vdash ((\text{init } w \wedge f \wedge \text{more}) \multimap \text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg \text{init } w)) =$
 $(\text{init } w \wedge (f \wedge \text{more}) \multimap (\text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg \text{init } w)))$
by (meson *MP SChopAndSFin int-iffD2*)
then have $\vdash (\neg \text{init } w \wedge \text{empty} \vee \text{init } w \wedge (f \wedge \text{more}) \multimap (\text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg \text{init } w))) =$
 $(\text{empty} \wedge \text{sfin } (\neg \text{init } w) \vee (\text{init } w \wedge f \wedge \text{more}) \multimap \text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg \text{init } w))$
using f2 **by** (metis *int-simps(1) inteq-reflection*)
then have $\vdash (\text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg \text{init } w)) =$
 $(\neg \text{init } w \wedge \text{empty} \vee \text{init } w \wedge (f \wedge \text{more}) \multimap (\text{schopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg \text{init } w)))$
by (metis 5 *inteq-reflection*)
then show ?thesis **by** meson
qed
from 9 **show** ?thesis **by** (simp add: *swhile-d-def*)
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: $\vdash \neg (init\ w) \wedge f \longrightarrow empty$

using *assms* **by** *blast*

have 2: $\vdash init\ w \wedge f \longrightarrow (g \wedge more) \frown f$

using *assms* **by** *blast*

have 3: $\vdash swhile\ (init\ w)\ do\ g =$

$((\neg (init\ w) \wedge empty) \vee (init\ w \wedge (g \wedge more) \frown swhile\ (init\ w)\ do\ g))$

by (*rule SWhileEqvEmptyOrSChopSWhile*)

hence 31: $\vdash \neg (swhile\ (init\ w)\ do\ g) =$

$(\neg (\neg (init\ w) \wedge empty) \vee (init\ w \wedge (g \wedge more) \frown swhile\ (init\ w)\ do\ g))$

by *fastforce*

hence 32: $\vdash (f \wedge \neg (swhile\ (init\ w)\ do\ g)) =$

$(f \wedge \neg (\neg (init\ w) \wedge empty) \vee (init\ w \wedge (g \wedge more) \frown swhile\ (init\ w)\ do\ g))$

by *fastforce*

have 33: $\vdash (f \wedge \neg (\neg (init\ w) \wedge empty) \vee (init\ w \wedge (g \wedge more) \frown swhile\ (init\ w)\ do\ g)) =$

$(f \wedge \neg (\neg (init\ w) \wedge empty) \wedge \neg (init\ w \wedge (g \wedge more) \frown swhile\ (init\ w)\ do\ g))$

by *auto*

have 34: $\vdash (f \wedge \neg (\neg (init\ w) \wedge empty) \wedge \neg ((init\ w) \wedge ((g \wedge more) \frown swhile\ (init\ w)\ do\ g))) =$

$(f \wedge ((init\ w) \vee more) \wedge (\neg (init\ w) \vee \neg ((g \wedge more) \frown swhile\ (init\ w)\ do\ g)))$

by (*auto simp: empty-d-def*)

have 35: $\vdash (f \wedge ((init\ w) \vee more) \wedge (\neg (init\ w) \vee \neg ((g \wedge more) \frown swhile\ (init\ w)\ do\ g))) =$

$((f \wedge (init\ w) \wedge \neg ((g \wedge more) \frown swhile\ (init\ w)\ do\ g)) \vee$

$(f \wedge (init\ w) \wedge \neg (init\ w)) \vee$

$(f \wedge more \wedge \neg ((g \wedge more) \frown swhile\ (init\ w)\ do\ g)) \vee$

$(f \wedge more \wedge \neg (init\ w))$

by *auto*

have 36: $\vdash (f \wedge \neg (swhile\ (init\ w)\ do\ g)) =$

$((f \wedge (init\ w) \wedge \neg ((g \wedge more) \frown swhile\ (init\ w)\ do\ g)) \vee$

$(f \wedge (init\ w) \wedge \neg (init\ w)) \vee$

$(f \wedge more \wedge \neg ((g \wedge more) \frown swhile\ (init\ w)\ do\ g)) \vee$

$(f \wedge more \wedge \neg (init\ w))$

using 32 33 34 35 **by** (*metis int-eq*)

have 37: $\vdash \neg (f \wedge more \wedge \neg (init\ w))$

using 1 **by** (*auto simp: empty-d-def*)

have 38: $\vdash (f \wedge more \wedge \neg ((g \wedge more) \frown swhile\ (init\ w)\ do\ g)) \longrightarrow$

$((g \wedge more) \frown f \wedge \neg ((g \wedge more) \frown swhile\ (init\ w)\ do\ g))$

using 1 2 **by** (*auto simp: empty-d-def Valid-def*)

have 39: $\vdash (f \wedge (init\ w) \wedge \neg ((g \wedge more) \frown swhile\ (init\ w)\ do\ g)) \longrightarrow$

$((g \wedge more) \frown f \wedge \neg ((g \wedge more) \frown swhile\ (init\ w)\ do\ g))$

using 2 **by** *auto*

have 40: $\vdash ((f \wedge (init\ w) \wedge \neg ((g \wedge more) \frown swhile\ (init\ w)\ do\ g)) \vee$

$(f \wedge (init\ w) \wedge \neg (init\ w)) \vee$

$(f \wedge more \wedge \neg ((g \wedge more) \frown swhile\ (init\ w)\ do\ g)) \vee$

$(f \wedge more \wedge \neg (init\ w)) \longrightarrow$

$(g \wedge more) \frown f \wedge \neg ((g \wedge more) \frown swhile\ (init\ w)\ do\ g)$

using 39 38 37 38 by fastforce
 have 4: $\vdash f \wedge \neg (swhile (init\ w) \ do\ g) \longrightarrow$
 $(g \wedge more) \frown f \wedge \neg ((g \wedge more) \frown while (init\ w) \ do\ g)$
 by (meson 36 40 Prop11 lift-imp-trans)
 have 5: $\vdash g \wedge more \longrightarrow more$
 by auto
 from 4 5 show ?thesis using SChopContraB by blast
 qed

lemma SWhileElim:
 assumes $\vdash \neg (init\ w) \wedge empty \longrightarrow g$
 $\vdash init\ w \wedge (f \wedge more) \frown g \longrightarrow g$
 shows $\vdash while (init\ w) \ do\ f \longrightarrow g$
proof –
 have 1: $\vdash while (init\ w) \ do\ f =$
 $((\neg (init\ w) \wedge empty) \vee (init\ w \wedge (f \wedge more) \frown while (init\ w) \ do\ f))$
 by (rule SWhileEqvEmptyOrSChopSWhile)
 hence 11: $\vdash ((while (init\ w) \ do\ f) \wedge \neg g) =$
 $((\neg (init\ w) \wedge empty) \vee (init\ w \wedge (f \wedge more) \frown while (init\ w) \ do\ f)) \wedge \neg g$
 by auto
 have 2: $\vdash \neg (init\ w) \wedge empty \longrightarrow g$
 using assms by blast
 hence 21: $\vdash \neg g \longrightarrow \neg (\neg (init\ w) \wedge empty)$
 by auto
 have 22: $\vdash ((\neg (init\ w) \wedge empty) \vee (init\ w \wedge (f \wedge more) \frown while (init\ w) \ do\ f)) \wedge \neg g \longrightarrow$
 $(init\ w \wedge (f \wedge more) \frown while (init\ w) \ do\ f)$
 using 21 by auto
 have 23: $\vdash (while (init\ w) \ do\ f) \wedge \neg g \longrightarrow$
 $(init\ w \wedge (f \wedge more) \frown while (init\ w) \ do\ f) \wedge \neg g$
 using 11 21 by fastforce
 have 3: $\vdash (init\ w) \wedge ((f \wedge more) \frown g) \longrightarrow g$
 using assms by blast
 hence 31: $\vdash \neg g \longrightarrow \neg ((init\ w) \wedge ((f \wedge more) \frown g))$
 by fastforce
 have 32: $\vdash (init\ w \wedge (f \wedge more) \frown while (init\ w) \ do\ f) \wedge \neg g \longrightarrow$
 $((f \wedge more) \frown (while (init\ w) \ do\ f)) \wedge \neg ((f \wedge more) \frown g) \wedge \neg g$
 using 31 by auto
 have 4: $\vdash (while (init\ w) \ do\ f) \wedge \neg g \longrightarrow$
 $((f \wedge more) \frown (while (init\ w) \ do\ f)) \wedge \neg ((f \wedge more) \frown g)$
 using 23 32 by fastforce
 have 5: $\vdash f \wedge more \longrightarrow more$
 by auto
 have 6: $\vdash (while (init\ w) \ do\ f) \wedge finite \longrightarrow g$
 using ChopContraB 4 5 using SChopContraB by blast
 have 7: $\vdash ((while (init\ w) \ do\ f) \wedge finite) = (while (init\ w) \ do\ f)$
 using while-d-def
 by (metis (no-types, lifting) 1 DiamondEmptyEqvFinite Prop10 Prop11 Prop12 SChopAndB
 SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond int-eq)
 from 6 7 show ?thesis by fastforce
 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 *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

7.10 Properties of Halt

lemma *HaltSCHopEqv*:

$\vdash ((\text{ halt } (\text{ init } w)) \frown f) = (\text{ if}_i (\text{ init } w) \text{ then } (f) \text{ else } (\bigcirc (\text{ halt } (\text{ init } w)) \frown f))$

proof –

have 1: $\vdash \text{ halt } (\text{ init } w) =$
 $(\text{ if}_i (\text{ init } w) \text{ then } \text{ empty } \text{ else } (\bigcirc (\text{ halt } (\text{ init } w))))$

by (*rule HaltStateEqvIfStateThenEmptyElseNext*)

hence 2: $\vdash ((\text{ halt } (\text{ init } w)) \frown f) =$
 $(\text{ if}_i (\text{ init } w) \text{ then } (\text{ empty } \frown f) \text{ else } (\bigcirc (\text{ halt } (\text{ init } w)) \frown f))$

by (*rule IfSCHopEqvRule*)

have 3: $\vdash \text{ empty } \frown f = f$

by (*rule EmptySCHop*)

have 4: $\vdash (\bigcirc (\text{ halt } (\text{ init } w))) \frown f = \bigcirc (\text{ halt } (\text{ init } w) \frown f)$

by (*rule NextSCHop*)

from 2 3 4 **show** *?thesis* **by** (*metis integ-reflection*)

qed

lemma *AndHaltSCHopImp*:

$\vdash \text{ init } w \wedge (\text{ halt } (\text{ init } w) \frown f) \longrightarrow f$

proof –
have 1: $\vdash \text{halt } (\text{init } w) \frown f = \text{if}_i \text{ } (\text{init } w) \text{ then } f \text{ else } (\bigcirc (\text{halt } (\text{init } w) \frown f))$
by (rule *HaltSCHopEqv*)
have 2: $\vdash \text{init } w \wedge \text{if}_i \text{ } (\text{init } w) \text{ then } f \text{ else } (\bigcirc (\text{halt } (\text{init } w) \frown f)) \longrightarrow f$
by (auto simp: *ifthenelse-d-def*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *NotAndHaltSCHopImpNext*:
 $\vdash \neg (\text{init } w) \wedge (\text{halt } (\text{init } w) \frown f) \longrightarrow \bigcirc (\text{halt } (\text{init } w) \frown f)$

proof –
have 1: $\vdash \text{halt } (\text{init } w) \frown f = \text{if}_i \text{ } (\text{init } w) \text{ then } f \text{ else } (\bigcirc (\text{halt } (\text{init } w) \frown f))$
by (rule *HaltSCHopEqv*)
have 2: $\vdash \neg (\text{init } w) \wedge \text{if}_i \text{ } (\text{init } w) \text{ then } f \text{ else } (\bigcirc (\text{halt } (\text{init } w) \frown f)) \longrightarrow$
 $\bigcirc (\text{halt } (\text{init } w) \frown f)$
by (auto simp: *ifthenelse-d-def*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *NotAndHaltSCHopImpSkipSYields*:
 $\vdash \neg (\text{init } w) \wedge (\text{halt } (\text{init } w) \frown f) \longrightarrow \text{skip syields } (\text{halt } (\text{init } w) \frown f)$

proof –
have 1: $\vdash \neg (\text{init } w) \wedge (\text{halt } (\text{init } w) \frown f) \longrightarrow \bigcirc (\text{halt } (\text{init } w) \frown f)$
by (rule *NotAndHaltSCHopImpNext*)
have 2: $\vdash \bigcirc (\text{halt } (\text{init } w) \frown f) \longrightarrow \text{skip syields } (\text{halt } (\text{init } w) \frown f)$
by (rule *NextImpSkipSYields*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *SChopAndEmptyEqvSCHopAndEmpty*:

$\vdash ((\# \text{True} \frown (f \wedge \text{empty})) \wedge g) = (g \frown (f \wedge \text{empty}))$
proof –
have 1: $\vdash (\# \text{True} \frown (f \wedge \text{empty})) \wedge g \longrightarrow g \frown (f \wedge \text{empty})$
by (simp add: *FiniteChopAndEmptyEqvChopAndEmpty int-iffD1 schop-d-def*)
have 2: $\vdash g \frown (f \wedge \text{empty}) \longrightarrow (\# \text{True} \frown (f \wedge \text{empty})) \wedge g$
by (metis *AndSFinEqvSCHopAndEmpty Prop12 SFinEqvTrueSCHopAndEmpty int-iffD1 inteq-reflection*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *NotSCHopSkipEqvFmoreAndNotSCHopSkip*:

$\vdash (\neg f) \frown \text{skip} = (f \text{ more} \wedge \neg (f \frown \text{skip}))$
proof –
have 1: $\vdash (\neg f) \frown \text{skip} = ((\neg f \wedge \text{finite}); \text{skip})$
by (simp add: *s chop-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: $\vdash (more \wedge \neg((f \vee inf);skip)) = (more \wedge \neg(f;skip \vee inf))$
 using 4 by auto
 have 6: $\vdash (more \wedge \neg(f;skip \vee inf)) = (more \wedge \neg(f;skip) \wedge finite)$
 unfolding finite-d-def by fastforce
 have 7: $\vdash (more \wedge \neg(f;skip) \wedge finite) = (more \wedge \neg(f \frown skip \vee (f \wedge inf))) \wedge finite$
 by (metis ChopEmpty ChopSChopdef inteq-reflection)
 have 8: $\vdash (more \wedge \neg(f \frown skip \vee (f \wedge inf))) \wedge finite =$
 $(more \wedge \neg(f \frown skip) \wedge \neg(f \wedge inf) \wedge finite)$
 by auto
 have 9: $\vdash (\neg(f \wedge inf) \wedge finite) = finite$
 unfolding finite-d-def by force
 have 10: $\vdash (more \wedge \neg(f \frown skip) \wedge \neg(f \wedge inf) \wedge finite) =$
 $(more \wedge \neg(f \frown skip) \wedge finite)$
 using 9 by fastforce
 have 11: $\vdash (more \wedge \neg(f \frown skip) \wedge finite) = (fmore \wedge \neg(f \frown skip))$
 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:

$\vdash \text{halt } (init\ w) \frown f \wedge finite \longrightarrow \neg (\text{halt } (init\ w) \frown (\neg\ f))$

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: $\vdash \text{if}_i\ (init\ w)\ \text{then } f\ \text{else } (\bigcirc(\text{halt } (init\ w) \frown f)) \longrightarrow$
 $((init\ w) \longrightarrow f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(\text{halt } (init\ w) \frown f))))$
 by (rule IfThenElseImp)
 have 3: $\vdash \text{halt } (init\ w) \frown (\neg f) =$
 $\text{if}_i\ (init\ w)\ \text{then } (\neg f)\ \text{else } (\bigcirc(\text{halt } (init\ w) \frown (\neg f)))$
 by (rule HaltSChopEqv)
 have 4: $\vdash \text{if}_i\ (init\ w)\ \text{then } (\neg f)\ \text{else } (\bigcirc(\text{halt } (init\ w) \frown (\neg f))) \longrightarrow$
 $((init\ w) \longrightarrow \neg f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(\text{halt } (init\ w) \frown (\neg f))))$
 by (rule IfThenElseImp)
 have 5: $\vdash \text{halt } (init\ w) \frown f \wedge \text{halt } (init\ w) \frown (\neg f) \longrightarrow$
 $((init\ w) \longrightarrow f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(\text{halt } (init\ w) \frown f))) \wedge$
 $((init\ w) \longrightarrow \neg f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(\text{halt } (init\ w) \frown (\neg f))))$
 using 1 2 3 4 by fastforce
 have 6: $\vdash ((init\ w) \longrightarrow f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(\text{halt } (init\ w) \frown f))) \wedge$
 $((init\ w) \longrightarrow \neg f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(\text{halt } (init\ w) \frown (\neg f)))) \longrightarrow$
 $(\bigcirc(\text{halt } (init\ w) \frown f)) \wedge (\bigcirc(\text{halt } (init\ w) \frown (\neg f)))$
 by auto
 have 7: $\vdash \text{halt } (init\ w) \frown f \wedge \text{halt } (init\ w) \frown (\neg f) \longrightarrow$
 $(\bigcirc(\text{halt } (init\ w) \frown f)) \wedge (\bigcirc(\text{halt } (init\ w) \frown (\neg f)))$
 using 5 6 lift-imp-trans by blast
 have 8: $\vdash ((\bigcirc(\text{halt } (init\ w) \frown f)) \wedge (\bigcirc(\text{halt } (init\ w) \frown (\neg f)))) =$
 $\bigcirc(\text{halt } (init\ w) \frown f \wedge \text{halt } (init\ w) \frown (\neg f))$
 using NextAndEqvNextAndNext by fastforce
 have 9: $\vdash \text{halt } (init\ w) \frown f \wedge \text{halt } (init\ w) \frown (\neg f) \longrightarrow$
 $\bigcirc(\text{halt } (init\ w) \frown f \wedge \text{halt } (init\ w) \frown (\neg f))$

using 7 8 by fastforce
 hence 10: $\vdash \text{finite} \longrightarrow \neg(\text{halt } (\text{init } w) \frown f \wedge \text{halt } (\text{init } w) \frown (\neg f))$
 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: $\vdash (\text{halt } (\text{init } w) \wedge f) \frown f1 \longrightarrow$
 $(\text{halt } (\text{init } w) \wedge f) \frown (\neg g) \vee ((\text{halt } (\text{init } w) \wedge f) \frown (f1 \wedge g))$
 by (rule SCHopOrImpRule)
 have 3: $\vdash (\text{halt } (\text{init } w) \wedge f) \frown (\neg g) \longrightarrow \text{halt } (\text{init } w) \frown (\neg g)$
 by (rule AndSCHopA)
 have 31: $\vdash (\text{halt } (\text{init } w) \wedge f) \frown f1 \longrightarrow$
 $\text{halt } (\text{init } w) \frown (\neg g) \vee ((\text{halt } (\text{init } w) \wedge f) \frown (f1 \wedge g))$
 using 23 by fastforce
 have 4: $\vdash \text{halt } (\text{init } w) \frown g \wedge \text{finite} \longrightarrow \neg(\text{halt } (\text{init } w) \frown (\neg g))$
 by (rule HaltSCHopImpNotHaltSCHopNot)
 hence 41: $\vdash (\text{halt } (\text{init } w) \frown (\neg g)) \wedge \text{finite} \longrightarrow \neg(\text{halt } (\text{init } w) \frown g)$
 by auto
 have 42: $\vdash (\text{halt } (\text{init } w) \wedge f) \frown f1 \wedge \text{finite} \longrightarrow$
 $\neg(\text{halt } (\text{init } w) \frown g) \vee ((\text{halt } (\text{init } w) \wedge f) \frown (f1 \wedge g))$
 using 31 41 by fastforce
 from 42 show ?thesis by auto

qed

lemma *HaltImpBoxSYields*:

$\vdash (\text{halt } (\text{init } w)) \frown f \wedge \text{finite} \longrightarrow (\Box(\neg (\text{init } w))) \text{ syields } ((\text{halt } (\text{init } w)) \frown f)$

proof –

have 1: $\vdash (\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)) \longrightarrow \text{df } (\Box(\neg (\text{init } w)))$
by (rule *SChopImpDf*)

have 2: $\vdash \Box(\neg (\text{init } w)) \longrightarrow \neg (\text{init } w)$
by (rule *BoxElim*)

hence 3: $\vdash \text{df } (\Box(\neg (\text{init } w))) \longrightarrow \text{df } (\neg (\text{init } w))$
by (rule *DfImpDf*)

have 4: $\vdash \text{df } (\text{init } (\neg w)) = (\text{init } (\neg w))$
by (rule *DfState*)

have 41: $\vdash (\text{init } (\neg w)) = (\neg (\text{init } w))$
using *Initprop*(2) **by** *fastforce*

have 42: $\vdash \text{df } (\neg (\text{init } w)) = (\neg (\text{init } w))$
using 4 41 **by** (*metis integ-reflection*)

have 5: $\vdash ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))) \longrightarrow \neg (\text{init } w)$
using 1 2 42 **using** 3 **by** *fastforce*

hence 51: $\vdash (\text{halt } (\text{init } w) \frown f) \wedge ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))) \longrightarrow$
 $(\text{halt } (\text{init } w) \frown f) \wedge \neg (\text{init } w)$
by *fastforce*

have 6: $\vdash \text{halt } (\text{init } w) \frown f = \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt } (\text{init } w) \frown f))$
by (rule *HaltSChopEqv*)

hence 61: $\vdash (\text{halt } (\text{init } w) \frown f \wedge \neg (\text{init } w)) =$
 $((\text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt } (\text{init } w) \frown f)))) \wedge \neg (\text{init } w)$
using 6 **by** *auto*

have 62: $\vdash (\text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt } (\text{init } w) \frown f))) \wedge$
 $\neg (\text{init } w) \longrightarrow (\bigcirc(\text{halt } (\text{init } w) \frown f))$
by (*auto simp: ifthenelse-d-def*)

have 63: $\vdash \text{halt } (\text{init } w) \frown f \wedge \neg (\text{init } w) \longrightarrow (\bigcirc(\text{halt } (\text{init } w) \frown f))$
using 61 62 **by** *fastforce*

have 7: $\vdash (\text{halt } (\text{init } w) \frown f) \wedge (\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)) \longrightarrow$
 $\bigcirc((\text{halt } (\text{init } w) \frown f))$
using 51 63 **using** *lift-imp-trans* **by** *blast*

have 8: $\vdash \Box(\neg (\text{init } w)) \longrightarrow \text{empty} \vee \bigcirc(\Box(\neg (\text{init } w)))$
by (*metis BoxImpYields WeakNextBoxImpMoreYields WnextEqvEmptyOrNext fmore-d-def int-eq*)

hence 9: $\vdash ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))) \longrightarrow$
 $\neg (\text{halt } (\text{init } w) \frown f) \vee \bigcirc((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)))$
by (rule *EmptyOrNextSChopImpRule*)

hence 10: $\vdash ((\text{halt } (\text{init } w) \frown f) \wedge (\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))) \longrightarrow$
 $\bigcirc((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)))$
by *fastforce*

have 11: $\vdash (\text{halt } (\text{init } w) \frown f) \wedge (\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)) \longrightarrow$
 $\bigcirc((\text{halt } (\text{init } w) \frown f) \wedge ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))))$
using 7 10 **by** *fastforce*

have 12: $\vdash \bigcirc((\text{halt } (\text{init } w) \frown f) \wedge ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))))$
 $\longrightarrow \bigcirc(((\text{halt } (\text{init } w) \frown f) \wedge ((\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f))))$
using *NextAndEqvNextAndNext* **by** *fastforce*

have 13: $\vdash (\text{halt } (\text{init } w) \frown f) \wedge (\Box(\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)) \longrightarrow$

```

      ○((( halt (init w))⊃ f) ∧ ((□(¬ (init w)))⊃ (¬ ( halt (init w)⊃ f))))
    using 11 12 by fastforce
  hence 14: ⊢ finite ⟶ ¬ (( halt (init w))⊃ f ∧ (□ (¬ (init w)))⊃ (¬(halt (init w)⊃ f)))
    using NextLoop by blast
  hence 15: ⊢ ( halt (init w))⊃ f ∧ finite ⟶ ¬ ((□ (¬ (init w)))⊃ (¬ ( halt (init w)⊃ f)))
    by auto
  from 15 show ?thesis by (simp add: syields-d-def)
qed

```

7.11 Properties of Groups of strong chops

lemma *NestedSChopImpSChop*:

```

assumes ⊢ init w ∧ f ⟶ g ⊃ (init w1 ∧ f1)
          ⊢ init w1 ∧ f1 ⟶ g1 ⊃ (init w2 ∧ f2)
shows   ⊢ init w ∧ f ⟶ g ⊃ (g1 ⊃ (init w2 ∧ f2))
proof −
  have 1: ⊢ init w ∧ f ⟶ g ⊃ (init w1 ∧ f1) using assms(1) by auto
  have 2: ⊢ init w1 ∧ f1 ⟶ g1 ⊃ (init w2 ∧ f2) using assms(2) by auto
  hence 3: ⊢ g ⊃ (init w1 ∧ f1) ⟶ g ⊃ (g1 ⊃ (init w2 ∧ f2)) by (rule RightSChopImpSChop)
  from 1 3 show ?thesis by fastforce
qed

```

end

8 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
begin

```

8.1 Definition of Set of intervals and Operations on them

type-synonym 'a *iintervals* = 'a *nellist set*

```

definition lan:: ('a:: world) formula ⇒ 'a iintervals
where lan f = { σ . (σ ⊨ f) }

```

```

definition fusion :: 'a iintervals ⇒ 'a iintervals ⇒ 'a iintervals (infixl · 70)
where X · Y = { σ .

```

```

  (∃ σ1 σ2. σ = nfuse σ1 σ2 ∧ nfinite σ1 ∧
    (σ1 ∈ X) ∧ (σ2 ∈ Y) ∧ (nlast σ1 = nfirst σ2))
  ∨ (¬ nfinite σ ∧ σ ∈ X) }

```

definition *semply* :: 'a iintervals (*SEmpty*)

where

$$SEmpty \equiv \{ \sigma \mid nlength \sigma = 0 \}$$

definition *smore* :: 'a iintervals (*SMore*)

where

$$SMore \equiv -\ SEmpty$$

definition *sskip* :: 'a iintervals (*SSkip*)

where

$$SSkip \equiv -(SEmpty \cup (SMore \cdot SMore))$$

definition *sfalse* :: 'a iintervals (*SFalse*)

where

$$SFalse \equiv \{\}$$

definition *strue* :: 'a iintervals (*STrue*)

where

$$STrue \equiv -\{\}$$

definition *sinit* :: 'a iintervals \Rightarrow 'a iintervals ((*SInit* -) [85] 85)

where

$$SInit\ X \equiv (X \cap SEmpty) \cdot STrue$$

definition *sinf* :: 'a iintervals (*SInf*)

where *SInf* \equiv *STrue* \cdot *SFalse*

definition *sfinite* :: 'a iintervals (*SFinite*)

where *SFinite* \equiv $-$ *SInf*

definition *sfmore* :: 'a iintervals (*SFMore*)

where *SFMore* \equiv *SFinite* \cap *SMore*

definition *sfin* :: 'a iintervals \Rightarrow 'a iintervals ((*SFin* -) [85] 85)

where

$$SFin\ X \equiv SFinite \cdot (X \cap SEmpty)$$

definition *ssometime* :: 'a iintervals \Rightarrow 'a iintervals ((*SSometime* -) [85] 85)

where

$$SSometime\ X \equiv SFinite \cdot X$$

definition *salways* :: 'a iintervals \Rightarrow 'a iintervals ((*SAlways* -) [85] 85)

where

$$SAlways\ X \equiv -(SSometime\ (-X))$$

definition *sdi* :: 'a iintervals \Rightarrow 'a iintervals ((*SDi* -) [85] 85)

where

$$SDi\ X \equiv X \cdot STrue$$

definition $sbi :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((Sbi\ -)\ [85]\ 85)$

where

$$Sbi\ X \equiv -(SDi\ (-X))$$

definition $sda :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SDa\ -)\ [85]\ 85)$

where

$$SDa\ X \equiv SFinite.X.STrue$$

definition $sba :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SBa\ -)\ [85]\ 85)$

where

$$SBa\ X \equiv -(SDa\ (-X))$$

definition $snext :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SNext\ -)\ [85]\ 85)$

where

$$SNext\ X \equiv SSkip.X$$

definition $swnext :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SWnext\ -)\ [85]\ 85)$

where

$$SWnext\ X \equiv -(SSkip.(-X))$$

definition $sprev :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SPrev\ -)\ [85]\ 85)$

where

$$SPrev\ X \equiv X.SSkip$$

definition $swprev :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SWprev\ -)\ [85]\ 85)$

where

$$SWprev\ X \equiv -((-X).SSkip)$$

primrec $spower :: 'a\ iintervals \Rightarrow nat \Rightarrow 'a\ iintervals\ ((SPower\ -)\ [88,88]\ 87)$

where

$$\begin{aligned} pwr-0 & : SPower\ X\ 0 = SEmpty \\ | pwr-Suc & : SPower\ X\ (Suc\ n) = ((X \cap SFinite).(SPower\ X\ n)) \end{aligned}$$

definition $sfpowerstar :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SFPowerstar\ -)\ [85]\ 85)$

where

$$SFPowerstar\ X \equiv (\bigcup n. SPower\ X\ n)$$

definition $spowerstar :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SPowerstar\ -)\ [85]\ 85)$

where

$$SPowerstar\ X \equiv (SFPowerstar\ X).(SEmpty \cup (X \cap SInf))$$

definition $sstar :: 'a\ iintervals \Rightarrow 'a\ iintervals\ ((SStar\ -)\ [85]\ 85)$

where

$$SStar\ X \equiv SPowerstar(X \cap SMore)$$

8.2 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 ≠ 0 ⇒
    nlength σ - enat n ≠ 0 ⇒ enat na ≤ nlength σ ⇒ enat na ≠ 0 ⇒
    nlength σ ≠ 0 ⇒ nlength σ - enat na ≠ 0 ⇒
    enat (Suc 0) < nlength σ
  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 ∧n. σ ∈ SMore · SMore ⇒
  enat n ≤ nlength σ ⇒
  enat n ≠ 0 ⇒ nlength σ - enat n ≠ 0 ⇒
  ¬ nfinite σ ⇒
  nlength σ ≠ 0 ⇒
  enat (Suc 0) < nlength σ
  by (metis Suc-ile-eq i0-less nlength-eq-enat-nfiniteD order.not-eq-order-implies-strict
    zero-enat-def)
show ∧n. σ ∈ SMore · SMore ⇒
  ¬ nfinite σ ⇒
  enat n ≤ nlength σ ⇒
  enat n ≠ 0 ⇒
  nlength σ ≠ 0 ⇒
  nlength σ - enat n ≠ 0 ⇒
  enat (Suc 0) < nlength σ
  by (metis Suc-ile-eq i0-less nlength-eq-enat-nfiniteD order.not-eq-order-implies-strict zero-enat-def)
show σ ∈ SMore · SMore ⇒ ¬ nfinite σ ⇒ nlength σ ≠ 0 ⇒ enat (Suc 0) < nlength σ
  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 σ ∉ SMore · SMore ⇒
  enat (Suc 0) < nlength σ ⇒
  ∀n. enat n ≤ nlength σ → enat n = 0 ∨ nlength σ = 0 ∨ nlength σ - enat n = 0 ⇒
  nfinite σ ⇒
  False
proof -
  assume a1: enat (Suc 0) < nlength σ
  assume nfinite σ
  assume a2: ∀n. enat n ≤ nlength σ → enat n = 0 ∨ nlength σ = 0 ∨ nlength σ - enat n = 0
  have enat (Suc 0) = 1
    using One-nat-def one-enat-def by presburger
  then have f3: enat 1 < nlength σ
    using a1 one-enat-def by presburger
  have f4: enat 1 ≤ enat 1
    by blast
  have enat 1 ≠ ∞
    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 skip-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 *sinfinit-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 sinfinit-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 σ $X \cap SFinite$ 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* :

$\sigma \in (X \cap SFinite) \cdot (SPower\ X\ n) \longleftrightarrow$
 $(\exists\ \sigma 1\ \sigma 2. \sigma = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge$
 $\sigma 1 \in X \wedge \sigma 2 \in (SPower\ X\ n) \wedge$

$nlast \sigma 1 = nfirst \sigma 2$)
)
 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*:

$\sigma \in \text{SFPowerstar } X \longleftrightarrow (\sigma \in \text{SEmpty} \vee \sigma \in (X \cap \text{SFinite}) \cdot (\text{SFPowerstar } X))$

by (*simp add: sfpowerstar-elim sfpowerstar-elim-1 sfpowerstar-suc-1*)

lemma *sfpowerstar-equiv*:

$\text{SFPowerstar } X = \text{SEmpty} \cup (X \cap \text{SFinite}) \cdot (\text{SFPowerstar } X)$

using *sfpowerstar-equiv-sem* **by** *blast*

lemma *sfpowerstar-equiv-1*:

$(\bigcup n. \text{SPower } X \ n) = \text{SEmpty} \cup (X \cap \text{SFinite}) \cdot (\bigcup n. \text{SPower } X \ n)$

using *sfpowerstar-equiv* **by** (*simp add: sfpowerstar-def*)

8.3 Algebraic Laws

8.3.1 Commutative Additive Monoid

lemma *UnionCommute*:

$(X::'a \text{ iintervals}) \cup Y = Y \cup X$

by (*simp add: Un-commute*)

lemma *UnionSFalse*:

$X \cup \text{SFalse} = X$

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*)

8.3.2 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*:

$\text{STrue} = X \cup -X$

by (*simp add: strue-def*)

lemma *SFalseBottom*:

$\text{SFalse} = X \cap -X$

by (*simp add: sfalse-def*)

8.3.3 multiplicative monoid

lemma *FusionSEmptyLsem*:

$$\sigma \in SEmpty \cdot X \longleftrightarrow \sigma \in X$$

using *fusion-iff-1*[of σ *SEmpty* X]

by (*auto simp add: fusion-iff-1 empty-elim min.absorb1 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 σ X *SEmpty*]

by (*auto simp add: fusion-iff-1 empty-elim*)

(*metis is-NNil-ndropn le-numeral-extra(3) ndropn-0 ndropn-nlength ntaken-all zero-enat-def,*
metis add.right-neutral le-iff-add ndropn-all ndropn-nlength nfinite-nlength-enat nlength-NNil
ntaken-all)

lemma *FusionSEmptyR* :

$$X \cdot SEmpty = X$$

using *set-eqI*[of $X \cdot SEmpty$ 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 (nfuse σ 1 σ 3) X Y]
  using 10 5 nfirst-nfuse by fastforce
show ?thesis using fusion-iff[of x X·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·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 σ 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 nlast-nfuse by fastforce
  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*

8.3.4 Subsumption order

lemma *Subsumption*:
 $((X :: 'a \text{ intervals}) \subseteq Y) = (X \cup Y = Y)$
by *auto*

8.3.5 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 *power-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  $Y \cup (X \cap SFinite) \cdot Z \subseteq Z$ 
  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 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:  $\bigwedge n. \text{SPower } X \ n \cdot (X \cap \text{SInf}) \cdot Y \subseteq Z$ 
  by (meson assms fusion-inductl-inf)
have  $X \cap \text{SMore} \cap \text{SInf} \subseteq X \cap \text{SInf}$ 
  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  $(\text{SPower } (X \cap \text{SMore}) \ n) \cdot Y \subseteq Z$ 
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  $Y \cdot (\text{SPower } X \ n) \subseteq Z$ 
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  $\forall S. Y \cdot (\text{SPower } X \ n \cdot S) \subseteq Z \cdot S$ 
  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 $Y \cup Z \cdot X \subseteq Z$

$$\text{shows } Y \cdot (SPower X n) \cdot (X \cap SInf) \subseteq Z$$

proof –

$$\text{have } 1: Y \cdot (SPower X n) \subseteq Z$$

using *assms fusion-inductr* **by** *blast*

show *?thesis*

proof –

$$\text{have } f4: Y \cdot SPower X n = Y \cdot SPower X n \cap Z$$

using *1* **by** *blast*

$$\text{have } Y \cdot SPower X n \cup Z = Z$$

by (*metis 1 subset-Un-eq*)

$$\text{then have } f5: Z \cdot (X \cap SInf) = Z \cdot (X \cap SInf) \cup Y \cdot SPower X n \cap Z \cdot (X \cap SInf)$$

using *f4* **by** (*metis (no-types) FusionUnionDistL sup.commute*)

$$\text{have } Y \cdot SPower X n = Y \cdot SPower X n \cap Z$$

using *f4* **by** *auto*

$$\text{then have } Y \cdot (SPower X n \cdot (X \cap SInf)) = Y \cdot SPower X n \cap Z \cdot (X \cap SInf)$$

by (*simp add: FusionAssoc1*)

then show *?thesis*

proof –

$$\text{have } f1: Y \cup Z \cdot X \cup Z = Z$$

by (*meson Subsumption assms*)

$$\text{have } f2: Y \cdot SPower X n \cap Z \cdot X \cup Z \cdot X = Z \cdot X$$

by (*metis (no-types) FusionUnionDistL (Y · SPower X n ∪ Z = Z) f4*)

$$\text{have } Y \cdot SPower X n \cap Z \cdot X \cup Y \cdot SPower X n \cap Z \cdot (X \cap SInf) = Y \cdot SPower X n \cap Z \cdot X$$

by (*metis (no-types) FusionUnionDistR sup-inf-absorb*)

then show *?thesis*

$$\text{using } f2 f1 \text{ (} Y \cdot (SPower X n \cdot (X \cap SInf)) = Y \cdot SPower X n \cap Z \cdot (X \cap SInf) \text{) by auto}$$

qed

qed

qed

lemma *fusion-inductr-more*:

assumes $Y \cup Z \cdot X \subseteq Z$

$$\text{shows } Y \cdot (SPower (X \cap SMore) n) \subseteq Z$$

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 *powercommute*:

$$(X \cap SFinite) \cdot (\bigcup n. (SPower\ X\ n)) = (\bigcup n. (SPower\ X\ n) \cdot (X \cap SFinite))$$

by (*metis (no-types, lifting) FusionUnionSPower1 Sup.SUP-cong power-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 –

$$\textbf{have } 1: Y \cdot (SStar\ X) = Y \cdot ((\bigcup n. (SPower\ (X \cap SMore)\ n))) \cdot (SEmpty \cup X \cap SMore \cap SInf))$$

by (*simp add: sstar-def spowerstar-def sfpowerstar-def*)

$$\textbf{have } 2: Y \cdot ((\bigcup n. (SPower\ (X \cap SMore)\ n))) \cdot (SEmpty \cup X \cap SMore \cap SInf) =$$

$$(Y \cdot (\bigcup n. (SPower\ (X \cap SMore)\ n))) \cdot (SEmpty \cup X \cap SMore \cap SInf)$$

using *FusionAssoc* **by** *blast*

$$\textbf{have } 3: (Y \cdot (\bigcup n. (SPower\ (X \cap SMore)\ n))) = (\bigcup n. Y \cdot (SPower\ (X \cap SMore)\ n))$$

using *FusionUnionSPower1* **by** *blast*

$$\textbf{have } 4: (Y \cdot (\bigcup n. (SPower\ (X \cap SMore)\ n))) \cdot (SEmpty \cup X \cap SMore \cap SInf) =$$

$$(\bigcup n. Y \cdot (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup X \cap SMore \cap SInf)$$

by (*simp add: 3*)

$$\textbf{have } 5: (\bigcup n. Y \cdot (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup X \cap SMore \cap SInf) =$$

$$(\bigcup n. (Y \cdot (SPower\ (X \cap SMore)\ n)) \cdot (SEmpty \cup X \cap SMore \cap SInf))$$

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 –

$$\text{have } 1: (SStar\ X) \cdot Y = ((\bigcup n. ((SPower\ (X \cap SMore)\ n))) \cdot (SEmpty \cup X \cap SMore \cap SInf)) \cdot Y$$

by (*simp add: sstar-def spowerstar-def sfpowerstar-def*)

$$\text{have } 2: ((\bigcup n. ((SPower\ (X \cap SMore)\ n))) \cdot (SEmpty \cup X \cap SMore \cap SInf)) \cdot Y = (\bigcup n. ((SPower\ (X \cap SMore)\ n))) \cdot ((SEmpty \cup X \cap SMore \cap SInf) \cdot Y)$$

using *FusionAssoc* by *blast*

$$\text{have } 3: (\bigcup n. ((SPower\ (X \cap SMore)\ n))) \cdot ((SEmpty \cup X \cap SMore \cap SInf) \cdot Y) = (\bigcup n. ((SPower\ (X \cap SMore)\ n))) \cdot ((SEmpty \cup X \cap SMore \cap SInf) \cdot Y)$$

using *UnionFusionSPower1*[of $X \cap SMore$ ($(SEmpty \cup X \cap SMore \cap SInf) \cdot Y$)]

by *auto*

$$\text{have } 4: (\bigcup n. ((SPower\ (X \cap SMore)\ n))) \cdot ((SEmpty \cup X \cap SMore \cap SInf) \cdot Y) = (\bigcup n. ((SPower\ (X \cap SMore)\ n) \cdot (SEmpty \cup X \cap SMore \cap SInf))) \cdot Y$$

using *FusionAssoc* by *blast*

show *?thesis*

by (*simp add: 1 2 3 4*)

qed

lemma *FPowerstarInductL*:

$$\text{assumes } Y \cup (X \cap SFinite) \cdot Z \subseteq Z$$

$$\text{shows } (SFPowerstar\ X) \cdot Y \subseteq Z$$

proof –

$$\text{have } 1: (SFPowerstar\ X) \cdot Y = (\bigcup n. (SPower\ X\ n)) \cdot Y$$

by (*simp add: sfpowerstar-def*)

$$\text{have } 2: (\bigcup n. (SPower\ X\ n)) \cdot Y = (\bigcup n. (SPower\ X\ n) \cdot Y)$$

using *UnionFusionSPower1* by *fastforce*

$$\text{have } 3: \bigwedge n. (SPower\ X\ n) \cdot Y \subseteq Z$$

using *assms fusion-inductl-finite* by *blast*

$$\text{have } 4: (\bigcup n. (SPower\ X\ n) \cdot Y) \subseteq Z$$

using 3 by *blast*

show *?thesis*

using 1 2 4 by *blast*

qed

lemma *FPowerstarInductMoreL*:

$$\text{assumes } Y \cup ((X \cap SMore) \cap SFinite) \cdot Z \subseteq Z$$

$$\text{shows } (SFPowerstar\ X) \cdot Y \subseteq Z$$

proof –

$$\text{have } 1: (SFPowerstar\ X) \cdot Y = (\bigcup n. (SPower\ X\ n)) \cdot Y$$

by (*simp add: sfpowerstar-def*)

$$\text{have } 2: (\bigcup n. (SPower\ X\ n)) \cdot Y = (\bigcup n. (SPower\ X\ n) \cdot Y)$$

using *UnionFusionSPower1* by *fastforce*

$$\text{have } 3: \bigwedge n. (SPower\ X\ n) \cdot Y \subseteq Z$$

by (*metis Int-assoc Int-commute assms fusion-inductl-fmore sfmore-def*)

$$\text{have } 4: (\bigcup n. (SPower\ X\ n) \cdot Y) \subseteq Z$$

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: spowerstar-def spowerstar-def*)

have 2: $(\bigcup n. (SPower X n)) \cdot (SEmpty \cup (X \cap SInf)) =$
 $(SEmpty \cup (X \cap SFinite)) \cdot (\bigcup n. (SPower X n)) \cdot (SEmpty \cup (X \cap SInf))$

by (*metis sfpowerstar-equiv-1*)

have 3: $(SEmpty \cup (X \cap SFinite)) \cdot (\bigcup n. (SPower X n)) \cdot (SEmpty \cup (X \cap SInf)) =$
 $SEmpty \cdot (SEmpty \cup (X \cap SInf)) \cup (X \cap SFinite) \cdot (\bigcup n. (SPower X n)) \cdot (SEmpty \cup (X \cap SInf))$

by (*simp add: FusionUnionDistL*)

have 4: $SEmpty \cdot (SEmpty \cup (X \cap SInf)) \cup (X \cap SFinite) \cdot (\bigcup n. (SPower X n)) \cdot (SEmpty \cup (X \cap SInf))$

=

$SEmpty \cup (X \cap SInf) \cup (X \cap SFinite) \cdot (\bigcup n. (SPower X n)) \cdot (SEmpty \cup (X \cap SInf))$

by (*simp add: FusionSEmptyL*)

have 5: $SEmpty \cup (X \cap SInf) \cup (X \cap SFinite) \cdot (\bigcup n. (SPower X n)) \cdot (SEmpty \cup (X \cap SInf)) =$
 $SEmpty \cup (X \cap SInf) \cdot (\bigcup n. (SPower X n)) \cdot (SEmpty \cup (X \cap SInf)) \cup$
 $(X \cap SFinite) \cdot (\bigcup n. (SPower X n)) \cdot (SEmpty \cup (X \cap SInf))$
by (metis Powerstarhelp2)
have 6: $SEmpty \cup (X \cap SInf) \cdot (\bigcup n. (SPower X n)) \cdot (SEmpty \cup (X \cap SInf)) \cup$
 $(X \cap SFinite) \cdot (\bigcup n. (SPower X n)) \cdot (SEmpty \cup (X \cap SInf)) =$
 $SEmpty \cup X \cdot ((\bigcup n. (SPower X n)) \cdot (SEmpty \cup (X \cap SInf)))$
by (metis FusionAssoc Powerstarhelp3 UnionAssoc)
show ?thesis
using 1 2 3 4 5 6 **by** presburger
qed

lemma SStareqv:
 $SStar X = SEmpty \cup (X \cap SMore) \cdot (SStar X)$
by (metis Powerstareqv sstar-def)

lemma SSkipSFinite:
 $SSkip \cap SFinite = SSkip$
proof –
have 1: $SSkip \cap SFinite \subseteq SSkip$
by simp
have 2: $SInf \subseteq SEmpty \cup (- SEmpty \cdot - SEmpty)$
by (metis Diff-Compl Diff-subset-conv Powerstarhelp2 Powerstarhelp3 double-compl inf-commute sup-ge1)
have 3: $SSkip \subseteq SSkip \cap SFinite$
using 2 **by** (simp add: sskip-def smore-def sfinite-def) **blast**
show ?thesis
using 3 **by** blast
qed

lemma spower-skip-elim :
 $(\sigma \in SPower SSkip n) \longleftrightarrow$
 $(nlength \sigma = n)$
proof (induct n arbitrary: σ)
case 0
then show ?case
by (simp add: 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 $\sigma SSkip SPower SSkip n$] **by** (simp add: inf-commute)
have 4: $(\exists \sigma 1 \sigma 2. \sigma = nfuse \sigma 1 \sigma 2 \wedge nfinite \sigma 1 \wedge \sigma 1 \in SSkip \wedge \sigma 2 \in SPower SSkip n \wedge nlast \sigma 1 =$

$\text{first } \sigma 2) \longleftrightarrow$
 $(\text{nlength } \sigma = \text{enat } (\text{Suc } n))$
by (*auto simp add: Suc.hyps nfuse-nlength one-enat-def sskip-elim*)
 $(\text{metis One-nat-def add.commute eSuc-enat enat.simps}(3) \text{ enat-add-sub-same enat-le-plus-same}(2)$
 $\text{min.orderE ndropn-nfirst ndropn-nlength nfinite-ntaken nfuse-ntaken-ndropn ntaken-nlast}$
 $\text{ntaken-nlength one-enat-def plus-1-eSuc}(2))$
show *?thesis*
using 1 3 4 **by** *presburger*
qed
qed

8.3.6 Kleene Algebra

— left unfold

lemma *UnfoldL*:

$$SEmpty \cup X \cdot (SStar X) = (SStar X)$$

proof —

$$\text{have } 1: (SStar X) = SEmpty \cup (X \cap SMore) \cdot (SStar X)$$

by (*meson Un-iff set-eqI SStareqv*)

$$\text{have } 2: (X \cap SMore) \cdot (SStar X) \subseteq X \cdot (SStar X)$$

by (*simp add: FusionRuleL*)

$$\text{have } 3: (SStar X) \subseteq SEmpty \cup X \cdot (SStar X)$$

using 1 2 **by** *blast*

$$\text{have } 4: SEmpty \subseteq (SStar X)$$

using 1 **by** *auto*

$$\text{have } 5: X \subseteq SEmpty \cup (X \cap SMore)$$

by (*simp add: Un-Int-distrib smore-def*)

$$\text{have } 6: X \cdot (SStar X) \subseteq (SStar X) \cup (X \cap SMore) \cdot (SStar X)$$

using 5 **by** (*metis FusionRuleL FusionUnionDistL FusionSEmptyL*)

$$\text{have } 7: (SStar X) \subseteq SEmpty \cup (X \cap SMore) \cdot (SStar X)$$

using 1 **by** *auto*

$$\text{have } 8: X \cdot (SStar X) \subseteq SEmpty \cup (X \cap SMore) \cdot (SStar X)$$

using 6 7 **by** *blast*

$$\text{hence } 9: X \cdot (SStar X) \subseteq (SStar X)$$

using 1 **by** *auto*

$$\text{have } 10: SEmpty \cup X \cdot (SStar X) \subseteq (SStar X)$$

using 9 4 **by** *simp*

from 3 10 **show** *?thesis* **by** *auto*

qed

— Left induction

lemma *SStarInductL*:

assumes $Y \cup X \cdot Z \subseteq Z$

shows $(SStar X) \cdot Y \subseteq Z$

proof —

$$\text{have } 1: (SStar X) \cdot Y = ((\bigcup n. (SPower (X \cap SMore) n)) \cdot (SEmpty \cup (X \cap SMore) \cap SInf)) \cdot Y$$

by (*simp add: sstar-def spowerstar-def sfpowerstar-def*)

$$\text{have } 2: ((\bigcup n. (SPower (X \cap SMore) n)) \cdot (SEmpty \cup (X \cap SMore) \cap SInf)) \cdot Y =$$

$(\bigcup n. ((SPower (X \cap SMore) n))) \cdot ((SEmpty \cup (X \cap SMore) \cap SInf) \cdot Y)$
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

8.3.7 Omega Algebra

lemma *SFMoresem* [*simp*, *mono*]:

$x \in (SInf \cap ((F \cap SFMore) \cdot G)) \longleftrightarrow$
 $\neg nfinite\ x \wedge$
 $((\exists \sigma 1\ \sigma 2.\ x = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge \sigma 1 \in F \wedge 0 < (nlength\ \sigma 1) \wedge \sigma 2 \in G \wedge nlast\ \sigma 1 = nfirst\ \sigma 2))$

by (*simp add: fusion-iff sinfinite-elim sfmore-elim*) *blast*

lemma *monoomega*[*simp*, *mono*]:

mono ($\lambda p\ x.\ x \in SInf \cap (F \cap SFMore \cdot p)$)

by (*auto simp add: mono-def sinfinite-elim fusion-iff sfmore-elim*)

coinductive-set *somega* :: 'a *iintervals* \Rightarrow 'a *iintervals*

for *F* where

$\neg nfinite\ x \wedge$
 $((\exists \sigma 1\ \sigma 2.\ x = nfuse\ \sigma 1\ \sigma 2 \wedge nfinite\ \sigma 1 \wedge \sigma 1 \in F \wedge 0 < (nlength\ \sigma 1) \wedge$
 $\sigma 2 \in (somega\ F) \wedge nlast\ \sigma 1 = nfirst\ \sigma 2))$
 $\implies x \in (somega\ F)$

lemma *SOmegaIntros*:

$SInf \cap ((F \cap SFMore) \cdot (somega\ F)) \subseteq (somega\ F)$

using *somega.intros*[*of - F*] *SFMoresem*[*of - F (somega F)*]

by (*meson subsetI*)

lemma *SOmegaCases*:

assumes $x \in somega\ F$

$x \in (SInf \cap ((F \cap SFMore) \cdot (somega\ F))) \implies P$

shows *P*

using *assms somega.cases*[*of x F P*] *SFMoresem* **by** *blast*

lemma *SOmegaUnrollsem*:

$x \in (somega\ F) \longleftrightarrow x \in ((F \cap SFMore) \cdot (somega\ F))$

proof *auto*

show $x \in somega\ F \implies x \in F \cap SFMore \cdot somega\ F$

using *SOmegaCases* **by** *blast*

show $x \in F \cap SFMore \cdot somega\ F \implies x \in somega\ F$

by (*metis (no-types, lifting) FusionAssoc1 FusionSFalse SOmegaCases SOmegaIntros*
inf-commute inf-left-idem subset-antisym subset-eq)

qed

lemma *SOmegaUnroll*:

$(somega\ F) = ((F \cap SFMore) \cdot (somega\ F))$

using *SOmegaUnrollsem* **by** *blast*

lemma *SOmegaCoinductsem*:

assumes $\bigwedge x.\ x \in X \implies x \in (SInf \cap ((F \cap SFMore) \cdot (X \cup (somega\ F))))$

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 $X \subseteq (SInf \cap ((F \cap SFMore) \cdot (X \cup (\text{somega } F))))$
shows $X \subseteq (\text{somega } F)$
using *assms SOmegaCoinductsem[of $X \ F$] by blast*

lemma *SOmegaWeakCoinductsem*:
assumes $\bigwedge x. x \in X \implies x \in (SInf \cap ((F \cap SFMore) \cdot X))$
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 $X \subseteq (SInf \cap ((F \cap SFMore) \cdot X))$
shows $X \subseteq (\text{somega } F)$
using *assms SOmegaWeakCoinductsem[of $X \ F$] by blast*

8.3.8 ITL specific Laws

lemma *PwrFusionInterLsem*:
 $x \in (((SPower \ SSkip \ n) \cap X) \cdot V) \cap (((SPower \ SSkip \ n) \cap Y) \cdot W) \longleftrightarrow$
 $x \in (((SPower \ SSkip \ n) \cap X \cap Y) \cdot (V \cap W))$
by *(auto simp add: spower-skip-elim fusion-iff-1 min-absorb1 nlength-eq-enat-nfiniteD)*

lemma *PwrFusionInterL*:
 $((((SPower \ SSkip \ n) \cap X) \cdot V) \cap (((SPower \ SSkip \ n) \cap Y) \cdot W)) =$
 $((SPower \ SSkip \ n) \cap X \cap Y) \cdot (V \cap W)$
using *PwrFusionInterLsem by blast*

lemma *PwrFusionInterRsem* :
 $x \in ((V \cdot ((SPower \ SSkip \ n) \cap X)) \cap ((W \cdot ((SPower \ SSkip \ n) \cap Y)))) \longleftrightarrow$
 $x \in ((V \cap W) \cdot ((SPower \ SSkip \ n) \cap X \cap Y))$
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* :
 $((V \cdot ((SPower \ SSkip \ n) \cap X)) \cap ((W \cdot ((SPower \ SSkip \ n) \cap Y)))) =$
 $((V \cap W) \cdot ((SPower \ SSkip \ n) \cap X \cap Y))$
using *PwrFusionInterRsem by blast*

lemma *SSkipFusionImpSMore*:
 $SSkip \cdot STrue \subseteq SMore$
using *subsetI[of $SSkip \cdot STrue \ SMore$]*
by *(auto simp add: min.absorb1 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: min.absorb1 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-skip-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)

8.4 Derived Laws

8.4.1 Helper Lemmas

lemma *B01*:

assumes $(X:: 'a \text{ iintervals}) \subseteq Y$

shows $-Y \subseteq -X$

using *assms* **by** *auto*

lemma *B04*:

$((X:: 'a \text{ iintervals}) = Y) \longleftrightarrow (X \subseteq Y) \wedge (Y \subseteq X)$

by *auto*

lemma *B09*:

assumes $-X \cup Y = STrue$

shows $(X:: 'a \text{ iintervals}) \subseteq Y$

using *assms* **using** *strue-def* **by** *auto*

lemma *B20*:

$(X:: 'a \text{ iintervals}) \subseteq Y \cup Z \longleftrightarrow X \cap -Y \subseteq Z$

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, hide-lams) 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 -

have 1: (SAlways ((- Y) \cup Y1)) = -((SFinite \cap (X1 \cup -X1)) \cdot (-((- Y) \cup Y1)))

by (simp add: salways-def ssometime-def)

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)

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)

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

have 5: (X1 \cdot Y) = ((SFinite \cap X1) \cdot Y) \cup (SInf \cap X1)

by (metis Powerstarhelp2 Powerstarhelp3 UnionCommute inf-commute)

have 6: -((SFinite \cap X1) \cdot (-((- Y) \cup Y1))) =
 -((SFinite \cap X1) \cdot (Y \cap -Y1)))

by simp

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)

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)

have 9: (SFinite \cap X1) \cdot Y1 \subseteq X1 \cdot Y1

by (simp add: FusionRuleL)

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

have 11: ((SFinite \cap X1) \cdot Y1) \cup (SInf \cap X1) \subseteq X1 \cdot Y1

by (metis B04 Powerstarhelp2 Powerstarhelp3 UnionCommute inf-commute)

have 12: -((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)

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

have 14: (-((SFinite \cap (X1 \cup -X1)) \cdot (-((- Y) \cup Y1))) \cap (SInf \cap X1)) \subseteq X1 \cdot Y1

using 10 11 by blast

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

8.4.2 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

8.5 Extra Laws

8.5.1 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*

8.5.2 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

8.5.3 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 UnionCommute 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, hide-lams) 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*)

8.5.4 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:

$((X \cap Y) \cap \text{SEmpty}) \cdot \text{SEmpty} = ((X \cap Y) \cap \text{SEmpty})$
by (*simp add: FusionSEmptyR*)

lemma ST16:

$(X \cap \text{SEmpty}) \cap (Y \cap \text{SEmpty}) \subseteq \text{SEmpty}$
by (*simp add: le-infI2*)

lemma ST17:

$(X \cap \text{SEmpty}) \cdot (Y \cap \text{SEmpty}) \subseteq \text{SEmpty}$
using ST10 **by** *auto*

lemma ST18:

$\neg((X \cap \text{SEmpty}) \cup (Y \cap \text{SEmpty})) = \neg(X \cap \text{SEmpty}) \cap \neg(Y \cap \text{SEmpty})$
by *auto*

lemma ST19:

$(X \cap \text{SEmpty}) \cdot (\neg X \cap \text{SEmpty}) \subseteq (X \cap \text{SEmpty})$
using ST10 **by** *blast*

lemma ST20:

$(X \cap \text{SEmpty}) \cdot (\neg X \cap \text{SEmpty}) \subseteq (\neg X \cap \text{SEmpty})$
using ST01 **by** *auto*

lemma ST22:

$((X \cap \text{SEmpty}) \cdot \text{SSkip}) \cdot (Y \cap \text{SEmpty}) \subseteq (X \cap \text{SEmpty}) \cdot \text{SSkip}$
using *FusionRuleR FusionSEmptyR* **by** *blast*

lemma ST23:

$((X \cap \text{SEmpty}) \cdot \text{SSkip}) \cdot (Y \cap \text{SEmpty}) \subseteq \text{SSkip} \cdot (Y \cap \text{SEmpty})$
by (*simp add: ST01 FusionRuleL*)

lemma ST27:

$(\text{SInit } X) \cap (Y \cdot Z) \subseteq ((\text{SInit } X) \cap Y) \cdot Z$
by (*metis B04 compl-bot-eq FusionAssoc FusionSEmptyL inf-commute inf-top.left-neutral CH09 sinit-def strue-def*)

lemma ST29:

$(\text{SInit } X) \cdot Y \subseteq (\text{SInit } X)$
using ST02 *FusionAssoc sinit-def* **by** *fastforce*

lemma ST30:

$(\text{SInit } X) \cap (\text{SDi } Y) = (\text{SDi } ((\text{SInit } X) \cap Y))$
unfolding *sdi-def sinit-def strue-def*
by (*metis CH09 FusionAssoc FusionSEmptyL compl-bot-eq inf-top.left-neutral*)

lemma ST31:

$(X \cdot (\text{STrue} \cap \text{SEmpty})) \cap (\text{STrue} \cdot (Y \cap \text{SEmpty})) = X \cdot (Y \cap \text{SEmpty})$
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*)

8.5.5 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*

8.5.6 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 $(\text{metis (no-types, hide-lams) SFalseBottom compl-sup disjoint-eq-subset-Compl double-compl inf-commute inf-le2 sfinite-def})$
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 $(\text{simp add: 1 BD38})$
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

8.5.7 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 $F \neg G$]
 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 $F \cap -G \subseteq (H \cdot F) \cap -(H \cdot G)$
 $H \subseteq SMore$
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: 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*)

8.5.8 Omega

lemma *OA01*:
 $(somega SSkip) = SSkip \cdot (somega SSkip)$
by (*metis SOmegaUnroll SSkipSFinite SSkipSMore inf-assoc sfmore-def*)

lemma *OA02*:
 $(somega SEmpty) = SFalse$
by (*metis Powerstarhelp2 SFalseBottom SOmegaUnroll SStar15 SStarSkip disjoint-eq-subset-Compl inf-bot-right inf-left-commute sfalse-def sfinite-def sfmore-def smore-def*)

lemma *OA03*:

(somega SFalse) = SFalse
by (metis Compl-disjoint2 Powerstarhelp2 SOmegaUnroll inf-assoc inf-compl-bot-right sfalse-def)

lemma OA04:

(somega SInf) = SFalse
by (metis FusionSFalse N02 Powerstarhelp2 SOmegaUnroll inf-compl-bot-left2 sfalse-def sfinite-def
 sfmore-def snext-def)

lemma BD59:

$SInf \cap G \cap SAlways (-G \cup ((F \cap SFMore) \cdot G)) \subseteq$
 $(F \cap SFMore) \cdot (SInf \cap G \cap SAlways (-G \cup ((F \cap SFMore) \cdot G)))$
proof –
have 1: $SInf \cap SAlways (-G \cup ((F \cap SFMore) \cdot G)) \subseteq (-G \cup ((F \cap SFMore) \cdot G))$
using BD12 **by** blast
have 2: $SInf \cap G \cap SAlways (-G \cup ((F \cap SFMore) \cdot G)) \subseteq$
 $SInf \cap ((F \cap SFMore) \cdot G) \cap SAlways (-G \cup ((F \cap SFMore) \cdot G))$
using 1 **by** blast
have 3: $SInf \cap ((F \cap SFMore) \cdot G) \cap SAlways (-G \cup ((F \cap SFMore) \cdot G)) \subseteq$
 $(F \cap SFMore) \cdot (SInf \cap G \cap SAlways (-G \cup ((F \cap SFMore) \cdot G)))$
by (metis BD58 FI23 inf-commute)
show ?thesis **using** 2 3 **by** blast
qed

lemma OA06:

$SInf \cap G \cap SAlways (-G \cup ((F \cap SFMore) \cdot G)) \subseteq (somega F)$
by (metis (no-types, lifting) BD59 SOmegaWeakCoinduct inf.cobounded1 inf-left-idem inf-mono)

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:

$(somega ((SPower SSkip (Suc n)))) \subseteq SInf$
by (meson IntE SOmegaCases subsetI)

lemma OA08:

$SInf \subseteq (somega ((SPower SSkip (Suc n))))$
proof –
have 1: $SInf \cap STrue \cap SAlways (-STrue \cup (((SPower SSkip (Suc n)) \cap SFMore) \cdot STrue)) \subseteq$
 $(somega (SPower SSkip (Suc n)))$
by (simp add: OA06)

```

have 2:  $- STrue \cup SPower SSkip (Suc\ n) \cap SFMore \cdot STrue =$ 
   $SPower SSkip (Suc\ n) \cap SFMore \cdot STrue$ 
  by (simp add: strue-def)
have 21:  $((SPower SSkip (Suc\ n)) \cap SFMore) \cdot STrue \subseteq SMore$ 
  by (metis (no-types, lifting) FI21 N09 N13 SMoreImpSSkipFusion SSkipFusionImpSMore SStar25
    SStarInductR SStarSkip Subsumption Un-mono inf-le2 subset-antisym)
have 3:  $SAways ((SPower SSkip (Suc\ n)) \cap SFMore) \cdot STrue \subseteq SAways (SMore)$ 
  using BD38 21 by auto
have 4:  $SAways (SMore) \subseteq SInf$ 
  using FI05 by blast
have 5:  $SAways (-STrue \cup ((SPower SSkip (Suc\ n)) \cap SFMore) \cdot STrue) \subseteq SInf$ 
  by (metis 2 3 FI05 subset-trans)
have 6:  $SInf \subseteq SAways (SMore)$ 
  by (simp add: BD01 FI15 salways-def smore-def)
have 7:  $(SPower SSkip (Suc\ n)) \cap SFMore \subseteq SFMore$ 
  using FI07 FI21 by blast
have 8:  $SMore \subseteq SFMore \cdot STrue$ 
  by (metis FI19 FI21 FusionAssoc1 SMoreImpSSkipFusion SSkipFusionImpSMore SStar19 SStarSkip
    subset-antisym)
have 9:  $SInf \cap ((SPower SSkip (Suc\ n)) \cap SFMore) \cdot STrue = ((SPower SSkip (Suc\ n)) \cap SFMore) \cdot SInf$ 
  by (metis FI23 boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def)
have 10:  $SInf \cap SAways ((SPower SSkip (Suc\ n)) \cap SFMore) \cdot STrue =$ 
   $SAways ((SPower SSkip (Suc\ n)) \cap SFMore) \cdot SInf$ 
  by (metis 9 BD29 FI53)
have 11:  $((SPower SSkip (Suc\ n)) \cap SFMore) \cdot SInf = SInf$ 
  proof (induct n)
  case 0
  then show ?case
  proof –
  have f1:  $(SFinite) \cap SSkip = SSkip$ 
    using SSkipSFinite by blast
  have f2:  $\forall S. S \cdot SSkip \subseteq SFMore \vee \neg S \subseteq SFinite$ 
    using FI10 FI21 FusionRuleL by blast
  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:  $(SSkip) \cap SFMore = SSkip$ 
    using f4 f3 f2 by auto
  have  $(SSkip) \cdot SInf = SInf$ 
    using f6 f5 f4 f3 f2 by (metis FI21 FusionAssoc FusionRuleL FusionSFalse inf.orderE inf-commute)
  then show ?thesis
    by (simp add: f7 FusionSEmptyR SSkipSFinite)
  qed

```

```

next
case (Suc n)
then show ?case
proof -
have f1:  $SSkip \cdot STrue = SMore$ 
  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:  $\forall S. SFMore \cap (SSkip \cdot S) = SFinite \cap (SSkip \cdot S)$ 
  by (simp add: f3 Int-assoc sfmore-def)
then show ?thesis
  using f3 f2 f1
proof -
have f4:  $SPower (SSkip) (Suc (Suc n)) \cap SFMore = SFinite \cap (SSkip \cap SFinite \cdot SPower SSkip (Suc n))$ 
  by (metis  $SSkipSFinite$   $\langle \forall S. SFMore \cap (SSkip \cdot S) = SFinite \cap (SSkip \cdot S) \rangle$  inf-commute pwr-Suc)
then have f5:  $SPower (SSkip) (Suc (Suc n)) \cap SFMore = SSkip \cap SFinite \cdot SPower SSkip (Suc n)$ 
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 FI23 FI25 SChopAssoc  $SSkipSFinite$   $Suc.hyps$  f0 f1 f2 inf.orderE inf-commute pwr-Suc spower-finite)
qed
qed
qed
have 12:  $SInf \subseteq SAlways ( (((SPower SSkip (Suc n)) \cap SFMore) \cdot SInf))$ 
  by (metis 11 B04 FI53)
show ?thesis
  using 11  $S\Omega WeakCoinduct$  by fastforce
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 Int-assoc $S\Omega Cases$ $S\Omega Unroll$ inf.absorb-iff2 subsetI)

lemma *OA12*:

$SInf \subseteq (\text{somega } STrue)$

proof –

have 1: $(SAlways (SFalse \cup (STrue \cap SMore) \cdot STrue)) \subseteq SInf$

by (*metis BD01 BD19 Compl-empty-eq FI15 FI21 SChopAssoc SMoreImpSSkipFusion SSkipFusion-ImpSMore*

double-complement equalityI inf-top.left-neutral order-eq-refl salways-def sda-def sfalse-def smore-def ssometime-def strue-def sup-bot.left-neutral)

have 2: $SInf \subseteq (SAlways (SFalse \cup (STrue \cap SMore) \cdot STrue))$

by (*metis BD01 BD19 Compl-empty-eq FI15 FI21 SChopAssoc SMoreImpSSkipFusion SSkipFusion-ImpSMore*

double-complement equalityI inf-top.left-neutral order-eq-refl salways-def sda-def sfalse-def smore-def ssometime-def strue-def sup-bot.left-neutral)

have 3: $SInf \cap STrue \cap (SAlways (SFalse \cup (STrue \cap SMore) \cdot STrue)) \subseteq (\text{somega } STrue)$

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*:

$(\text{somega } SMore) \subseteq SInf$

using *SOmegaCases* **by** *auto*

lemma *OA15*:

$SInf \subseteq (\text{somega } SMore)$

proof –

have 1: $(SAlways (SFalse \cup (SMore \cap SMore) \cdot STrue)) \subseteq SInf$

by (*metis FI05 FI19 FI21 FusionSEmptyR N09 N10 SChopAssoc SMoreImpSSkipFusion SSkipFusion-ImpSMore*

SStar19 SStarSkip ST33 compl-bot-eq compl-inf compl-top-eq inf.absorb2 sfalse-def smore-def strue-def subset-antisym)

have 2: $SInf \subseteq (SAlways (SFalse \cup (SMore \cap SMore) \cdot STrue))$

by (*metis B01 BD38 FI15 FI19 FI21 FI53 FusionSEmptyR N09 N10 SChopAssoc SMoreImpSSkipFusion*

SSkipFusionImpSMore SStar15 SStar19 SStarSkip ST33 compl-bot-eq compl-inf compl-top-eq inf.absorb2 sfalse-def smore-def strue-def subset-antisym)

have 3: $SInf \cap STrue \cap (SAlways (SFalse \cup (SMore \cap SMore) \cdot STrue)) \subseteq (\text{somega } STrue)$

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:

$(\text{somega } S\text{Finite}) \subseteq S\text{Inf}$

by (*metis FI15 FI42 SOmegaUnroll compl-le-swap1 inf-left-idem order-refl sfmore-def*)

lemma OA18:

$S\text{Inf} \subseteq (\text{somega } S\text{Finite})$

proof –

have 1: $(S\text{Always } (S\text{False} \cup (S\text{Finite} \cap S\text{FMore}) \cdot S\text{True})) \subseteq S\text{Inf}$

by (*metis FI05 FI07 FI19 FI21 FusionSEmptyR SChopAssoc SMoreImpSSkipFusion SSkipFusion-ImpSMore*

SStar19 SStarSkip ST33 compl-bot-eq compl-inf compl-top-eq inf.absorb2 sfalse-def smore-def strue-def subset-antisym)

have 2: $S\text{Inf} \subseteq (S\text{Always } (S\text{False} \cup (S\text{Finite} \cap S\text{FMore}) \cdot S\text{True}))$

by (*metis BD38 FI07 FI19 FI21 FI53 FusionSEmptyR SChopAssoc SMoreImpSSkipFusion SSkipFusion-ImpSMore*

SStar15 SStar19 SStarSkip ST33 compl-bot-eq compl-inf compl-le-swap1 compl-top-eq inf.absorb2 sfalse-def sfinite-def smore-def strue-def subset-antisym)

have 3: $S\text{Inf} \cap S\text{True} \cap (S\text{Always } (S\text{False} \cup (S\text{Finite} \cap S\text{FMore}) \cdot S\text{True})) \subseteq (\text{somega } S\text{True})$

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:

$(\text{somega } S\text{Finite}) = S\text{Inf}$

by (*simp add: B04 OA17 OA18*)

lemma OA20:

assumes $H \subseteq (F \cap S\text{FMore}) \cdot H$

shows $H \cap S\text{Inf} \subseteq (\text{somega } F)$

using *assms*

by (*metis B08 BD08 OA06 boolean-algebra-cancel.inf0 compl-bot-eq inf-commute strue-def*)

lemma OA21:

assumes $F \subseteq G$

shows $(\text{somega } F) \cap S\text{Inf} \subseteq (\text{somega } G)$

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:

$(\text{somega } (F \cap G)) \cap S\text{Inf} \subseteq (\text{somega } F)$

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: $SBa (-F \cup G) \cap ((F \cap SFMore) \cdot (somega F)) \subseteq$
 $((-F \cup G) \cap (F \cap SFMore)) \cdot ((-F \cup G) \cap (somega F))$
by *(simp add: BD62)*
have 2: $(-F \cup G) \cap (F \cap SFMore) \subseteq (G \cap SFMore)$
by *auto*
have 3: $(-F \cup G) \cap (somega F) \subseteq (somega F)$
by *auto*
have 4: $SBa (-F \cup G) \cap ((F \cap SFMore) \cdot (somega F)) \subseteq (G \cap SFMore) \cdot (somega F)$
using 1 2 3 *CH06 by blast*
have 5: $SBa (-F \cup G) \cap ((F \cap SFMore) \cdot (somega F)) \subseteq$
 $((-F \cup G) \cap (F \cap SFMore)) \cdot (SBa (-F \cup G) \cap (somega F))$
using *BD63 by blast*
have 6: $((-F \cup G) \cap (F \cap SFMore)) \cdot (SBa (-F \cup G) \cap (somega F)) \subseteq$
 $(G \cap SFMore) \cdot (SBa (-F \cup G) \cap (somega F))$
using 2 *FusionRuleL by blast*
have 7: $(SBa (-F \cup G) \cap (somega F)) \subseteq (G \cap SFMore) \cdot (SBa (-F \cup G) \cap (somega F))$
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:

$$SInf \cap ((F \cap SFMore) \cdot G) = (F \cap SFMore) \cdot (G \cap SInf)$$

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)

8.6 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*:

$$(\sigma \in (\neg(\text{lan } f) \cup (\text{lan } g))) \longleftrightarrow (\sigma \models f \longrightarrow g)$$

by *simp*

lemma *subset-impl-1*:

$$(\neg(\text{lan } f) \cup (\text{lan } g)) = (\text{lan } (\text{LIFT}(f \longrightarrow g)))$$

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*:

$$\sigma \in S\text{More} \longleftrightarrow (\sigma \models \text{more})$$

by (*simp add: itl-defs smore-elim smore-def zero-enat-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 *power-chop-power*:

$(SPower(\text{lan } f) \ n) = (\text{lan}(LIFT(\text{power } f \ n)))$

proof (*induct n*)

case 0

then show ?*case* **by** (*simp add: empty-empty-1*)

next

case (*Suc n*)

then show ?*case* **by** (*simp add: and-inter-finite-1 fusion-chop-1*)

qed

lemma *powerstar*:

$\sigma \in SPowerstar(\text{lan } f) \longleftrightarrow \sigma \in SFPowerstar(\text{lan}(f)) \cdot (SEmpty \cup ((\text{lan } f) \cap SInf))$

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*:

$\sigma \in (\bigcup n. SPower\ (lan\ f)\ n) \longleftrightarrow \sigma \in lan(LIFT(\exists n. power\ f\ n))$

by (*simp add: spower-chop-power*)

lemma *union-exists-1*:

$(\bigcup n. SPower\ (lan\ f)\ n) = lan(LIFT(\exists n. power\ f\ n))$

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: $\bigwedge n. SPower\ (lan(LIFT(f \wedge more)))\ n = lan(LIFT(power\ (f \wedge more)\ n))$

using *power-chop-power* **by** *blast*

have 32: $SFPowerstar(lan(LIFT(f \wedge more))) = lan(LIFT(fpowerstar\ (f \wedge more)))$

using *union-exists-1* **by** (*auto simp add: sfpowerstar-def fpowerstar-d-def*)

have 4: $(SEmpty \cup (((lan\ f) \cap SMore) \cap SInf)) =$

$lan(LIFT(empty \vee ((f \wedge more) \wedge inf)))$

by (*metis Morgan and-inter-more-2 finite-d-def inter-and-1 not-negation-1 empty-empty-1 sfinite-1 sfinite-def union-or-1*)

have 5: $SFPowerstar\ (lan\ (f) \cap SMore) \cdot (SEmpty \cup (((lan\ f) \cap SMore) \cap SInf)) =$

$lan(LIFT(powerstar\ (f \wedge more)))$

by (*simp add: powerstar-d-def 3 32 4 fpowerstar-d-def fusion-chop-1*)

have 6: $lan(LIFT(powerstar\ (f \wedge more))) =$

$lan(LIFT(chopstar\ f))$

by (*simp add: chopstar-d-def*)

show *?thesis*

by (*simp add: 1 2 5 6*)

qed

lemma *chopstar-sstar-1*:

$(SStar\ (lan\ f)) = (lan\ (LIFT(f^*)))$

using *sstar-chopstar lan-def* **by** *blast*

lemma *chopstar-seqv*:

$\sigma \in (lan\ (LIFT(f^*))) \longleftrightarrow \sigma \in (lan\ (LIFT(empty \vee (f \wedge more); f^*)))$

by (*metis (no-types, lifting) SChopstarEqv chopstar-sstar-1 fusion-chop-1 inter-and-1*)

empty-empty-1 smore-more-1 union-or-1)

lemma *chopstar-seqv-1:*

$(\text{lan } (LIFT(f^*)) = (\text{lan } (LIFT(\text{empty} \vee (f \wedge \text{more}); f^*)))$

using *chopstar-seqv lan-def* **by** *blast*

lemma *sinf:*

$\sigma \in SInf \iff (\sigma \models \text{inf})$

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:*

$\sigma \in SFMore \iff (\sigma \models \text{fmore})$

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:*

$x \in - (\text{lan } g) \cup (SInf \cap ((\text{lan } f) \cap SFMore) \cdot (\text{lan } g)) \iff$

$(x \models g \longrightarrow \text{inf} \wedge (f \wedge \text{fmore}); g)$

by (*simp add: fmore-1 fusion-chop-1 inter-and-1 sinf-1*)

lemma *omega-induct:*

$- (\text{lan } g) \cup (SInf \cap ((\text{lan } f) \cap SFMore) \cdot (\text{lan } g)) =$

$\text{lan}(LIFT(g \longrightarrow \text{inf} \wedge (f \wedge \text{fmore}); g))$

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: $x \models f^\omega \longrightarrow \text{inf} \wedge (f \wedge \text{fmore}); f^\omega$

using *OmegaCases* **by** *auto*

have 2: $x \in - (\text{lan } (LIFT(f^\omega))) \cup ((\text{lan } f) \cap SFMore) \cdot (\text{lan } (LIFT(f^\omega)))$

by (*simp add: OmegaUnrollSem fmore-1 fusion-chop-1 inter-and-1*)

have 3: $\bigwedge x . x \in (\text{lan } (LIFT(f^\omega))) \implies x \in (SInf \cap ((\text{lan } f) \cap SFMore) \cdot (\text{lan } (LIFT(f^\omega))))$

by (*metis (no-types, lifting) OmegaCases fmore-1 fusion-chop-1 inter-and-1 interval-lan sinf-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: $x \in - (\text{somega } (\text{lan } f)) \cup (SInf \cap ((\text{lan } f) \cap SFMore) \cdot (\text{somega } (\text{lan } f)))$

```

using SOmegaCases by blast
have 2:  $\bigwedge x. (\lambda x. x \in \text{somega } (\text{lan } f)) x \implies$ 
 $x \in (SInf \cap ((\text{lan } f) \cap SFMore) \cdot (\text{somega } (\text{lan } f))))$ 
using SOmegaCases by blast
have 3:  $\bigwedge x. (\lambda x. x \in \text{somega } (\text{lan } f)) x \implies$ 
 $(\neg \text{nfinite } x \wedge (\exists n. f ( \text{ntaken } n \ x)) \wedge n > 0 \wedge$ 
 $(\lambda x. x \in \text{somega } (\text{lan } f)) ( \text{ndropn } n \ ( x))))$ 

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:  $\bigwedge x. (\lambda x. x \in \text{somega } (\text{lan } f)) x \implies$ 
 $x \models \text{inf} \wedge (f \wedge \text{fmore}); (\lambda x. x \in \text{somega } (\text{lan } f))$ 

using 3 FMoreSem by blast
have 5:  $(\lambda x. x \in \text{somega } (\text{lan } f)) x$ 
by (simp add: assms)
show ?thesis using 4 5 OmegaWeakCoinductSem[of (lambda x. x in somega (lan f)) f]
by auto
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:
 $\text{somega } (\text{lan } f) = \text{lan}(LIFT( f^\omega))$ 
using somega-omega by fastforce

```

end

9 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 [10] are proven in a mostly deductive style.

9.1 Definitions

```

definition until-d :: ('a :: world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula
where until-d  $F \ G \equiv \lambda s. ( (\exists k. k \leq \text{nlength } s \wedge ( \text{ndropn } k \ s ) \models G) \wedge$ 
 $(\forall j. j < k \longrightarrow ( \text{ndropn } j \ s ) \models F) ) )$ 

```

```

syntax
-until-d :: [lift, lift]  $\Rightarrow$  lift          ((-  $\mathcal{U}$  -) [84,84] 83)

```

syntax (*ASCII*)

$-until-d \quad :: [lift, lift] \Rightarrow lift \quad ((- \text{ until } -) [84, 84] \ 83)$

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

$-srelease-d \quad \Rightarrow CONST \text{ srelease-}d$

9.2 Axioms

9.2.1 NextUntil

lemma *NextUntilsema*:

assumes $(\sigma \models \bigcirc(f \ \mathcal{U} \ g))$

shows $(\sigma \models (\bigcirc \ f) \ \mathcal{U} \ (\bigcirc \ g))$

proof –

have $0: 0 < nlength \ \sigma \wedge$

$(\exists k. k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \sigma) \wedge g ((\text{ndropn } (\text{Suc } k) \sigma)) \wedge$
 $(\forall j < k. f ((\text{ndropn } (\text{Suc } j) \sigma)))$
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: $0 < \text{nlength } \sigma$
using 2 **by** *auto*
have 4: $k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \sigma)$
by *auto*
 $(\text{metis } 2 \ 3 \ \text{add.commute } \text{add.right-neutral } \text{enat-min } \text{iless-Suc-eq } \text{ndropn-0}$
 $\text{ndropn-Suc-conv-ndropn } \text{ndropn-nlength } \text{nlength-NCons } \text{zero-enat-def})$
have 5: $g \ (\ (\text{ndropn } (\text{Suc } k) \sigma))$
using 2 **by** *auto*
have 6: $(\forall j < k. j < \text{nlength } \sigma \wedge f \ (\ (\text{ndropn } (\text{Suc } j) \sigma)))$
using 2 **by** *blast*
have 7: $0 < \text{nlength } \sigma \wedge$
 $(\exists k. k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \sigma) \wedge g \ (\ (\text{ndropn } (\text{Suc } k) \sigma)) \wedge$
 $(\forall j < k. f \ (\ (\text{ndropn } (\text{Suc } j) \sigma))))$
using 2 3 4 **by** *blast*
from 7 **show** ?thesis **by** (*auto simp: next-defs until-d-def zero-enat-def ndropn-ndropn*)
qed

lemma *NextUntilsem*:

$\sigma \models \circ(f \ \mathcal{U} \ g) = (\circ f) \ \mathcal{U} \ (\circ g)$
using *NextUntilsema NextUntilsemb* **using** *unl-lift2* **by** *blast*

lemma *NextUntil*:

$\vdash \circ(f \ \mathcal{U} \ g) = (\circ f) \ \mathcal{U} \ (\circ g)$
using *NextUntilsem Valid-def* **by** *blast*

9.2.2 UntilNextUntil

lemma *UntilNextUntilsema*:

assumes $0 < \text{nlength } \sigma \wedge$
 $(\exists k. 0 < k \wedge k \leq \text{nlength } \sigma \wedge g \ ((\text{ndropn } k \sigma)) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f \ ((\text{ndropn } j \sigma))))$
shows $(\sigma) \models \circ \ (\ f \ \mathcal{U} \ g)$
proof –
have 1: $0 < \text{nlength } \sigma \wedge$
 $(\exists k. 0 < k \wedge k \leq \text{nlength } \sigma \wedge g \ ((\text{ndropn } k \sigma)) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f \ ((\text{ndropn } j \sigma))))$
using *assms* **by** *auto*
have 3: $(\exists k. 0 < k \wedge k \leq \text{nlength } \sigma \wedge g \ ((\ (\text{ndropn } k \sigma))) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f \ ((\text{ndropn } j \sigma))))$
using 1 **by** *auto*
obtain k **where** 4: $0 < (\text{Suc } k) \wedge (\text{Suc } k) \leq \text{nlength } \sigma \wedge g \ (\ (\text{ndropn } (\text{Suc } k) \sigma)) \wedge$
 $(\forall j. 0 < j \wedge j < (\text{Suc } k) \longrightarrow f \ (\ (\text{ndropn } j \sigma)))$
using 3 **by** (*metis Suc-pred*)
have 5: $k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \sigma)$
by (*metis* 4 *One-nat-def add.commute co.enat.sel(2) eSuc-enat epred-conv-minus le-cases min-absorb1*
 $\text{ndropn-nlength ntaken-all ntaken-ndropn-swap-nlength ntaken-nlength one-enat-def plus-1-eSuc(2)$
 plus-1-eq-Suc)
have 6: $g \ (\ (\text{ndropn } (\text{Suc } k) \sigma))$
using 4 **by** *auto*
have 7: $(\forall j < k. f \ (\ (\text{ndropn } (\text{Suc } j) \sigma)))$
using 4 **by** *blast*
have 8: $(\exists k. k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \sigma) \wedge g \ ((\text{ndropn } (\text{Suc } k) \sigma)) \wedge (\forall j < k. f \ ((\text{ndropn } (\text{Suc } j) \sigma))))$
using 4 5 **by** *blast*

have 9: $0 < \text{nlength } \sigma \wedge$
 $(\exists k. k \leq \text{nlength } (\text{ndropn } (\text{Suc } 0) \sigma) \wedge g ((\text{ndropn } (\text{Suc } k) \sigma)) \wedge (\forall j < k. f ((\text{ndropn } (\text{Suc } j) \sigma))))$
using 1 8 **by** *blast*
from 9 **show** *?thesis* **by** *(auto simp add: next-defs until-d-def zero-enat-def ndropn-ndropn)*
qed

lemma *UntilNextUntilsemb:*

assumes $\sigma \models \bigcirc (f \mathcal{U} g)$

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: ( $\sigma \models g \vee$ 
   $f \wedge$ 
   $(\lambda\sigma. 0 < nlength\ \sigma \wedge$ 
     $(\exists k. 0 < k \wedge k \leq nlength\ \sigma \wedge g((ndropn\ k\ \sigma)) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f((ndropn\ j\ \sigma))))))$ 
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)

```

9.2.3 NotUntilFalse

lemma *NotUntilFalse*:

```

 $\vdash \neg (f\ \mathcal{U}\ \#False)$ 
by (simp add: intI until-d-def)

```

9.2.4 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

```

9.2.5 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)

```

9.2.6 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)

```

9.2.7 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

9.2.8 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*)

9.2.9 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

```

9.2.10 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

```

9.2.11 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: $(\forall j < k.$
 $\quad \exists ka. ka \leq nlength (ndropn\ j\ \sigma) \wedge g ((ndropn\ (ka + j)\ \sigma)) \wedge (\forall ja < ka. f ((ndropn\ (ja + j)\ \sigma))))$
proof *auto*
fix j
assume $a0: j < k$
show $\exists ka. enat\ ka \leq nlength\ \sigma - enat\ j \wedge g (ndropn\ (ka + j)\ \sigma) \wedge (\forall ja < ka. f (ndropn\ (ja + j)\ \sigma))$
proof –
have 51: $k - j \leq nlength (ndropn\ j\ \sigma)$
using 4 $a0$
by (*metis enat-minus-mono1 idiff-enat-enat ndropn-nlength*)
have 52: $g ((ndropn\ ((k - j) + j)\ \sigma))$
by (*simp add: 2 a0 less-imp-le-nat*)
have 53: $(\forall ja < (k - j). f ((ndropn\ (ja + j)\ \sigma)))$
using 2 *less-diff-conv* **by** *blast*
show *?thesis*
using 51 52 53 **by** *auto*
qed
qed
have 6: $\exists k. k \leq nlength\ \sigma \wedge$
 $h ((ndropn\ k\ \sigma)) \wedge$
 $(\forall j < k. \exists k' \leq nlength (ndropn\ j\ \sigma). g ((ndropn\ (k + j)\ \sigma)) \wedge (\forall ja < k. f ((ndropn\ (ja + j)\ \sigma))))$
using 2 5 **by** *blast*
from 6 **show** *?thesis* **by** (*simp add: until-d-def ndropn-ndropn add commute*)
qed

lemma *UntilRightAnd*:
 $\vdash f\ \mathcal{U}\ (g \wedge h) \longrightarrow (f\ \mathcal{U}\ g)\ \mathcal{U}\ h$
using *UntilRightAndsem Valid-def* **by** *auto*

9.2.12 DiamondEqvTrueUntil

lemma *DiamondEqvTrueUntil*:
 $\vdash \Diamond f = \#True\ \mathcal{U}\ f$
by (*simp add: Valid-def sometimes-defs until-d-def*)

9.2.13 TrueUntillImpNotUntil

lemma *nellist-ndropn-first-upto*:
assumes $(\exists i \leq k. f ((ndropn\ i\ xs)))$
shows $(\exists i \leq k. f ((ndropn\ i\ xs)) \wedge (\forall j < i. \neg (f ((ndropn\ j\ xs)))))$
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)

```

9.2.14 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:  $n \leq \text{nlength } \sigma \wedge \neg f \ ( \text{ndropn } n \ \sigma) \wedge$ 

```



```

      (∀ k < n . f ( (ndropn k σ)))
  using 3 using NotSuffixFirstfinite by blast
have 16: n ≤ nlength σ
  by (simp add: 4)
have 17: ¬ g ( (ndropn n σ))
  using 1 4 by blast
have 18: (∀ j < n. ¬ g ( (ndropn j σ)))
  by (metis 2 4 dual-order.strict-iff-order dual-order.strict-trans1 enat-ord-simps(2))
have 19: ∃ k ≤ nlength σ. ¬ f ( (ndropn k σ)) ∧ ¬ g ( (ndropn k σ)) ∧ (∀ j < k. ¬ g ( (ndropn j σ)))
  using 16 17 18 4 by blast
have 20: ( σ ⊨ ((¬ g) U (¬ f ∧ ¬ g)))
  using 19 by (simp add: until-d-def)
show ?thesis using 20 by auto
qed

```

```

lemma WaitNotDistUntilsem2:
assumes (σ ⊨ ((¬ g) U ((¬ f) ∧ (¬ g))))
shows   ( σ ⊨ ¬(f W g))
using assms not-less-iff-gr-or-eq by (auto simp add: always-defs wait-d-def until-d-def)

```

```

lemma WaitNotDistUntilsem:
(σ ⊨ (¬(f W g)) = ((¬ g) U ((¬ f) ∧ (¬ g))))
using WaitNotDistUntilsem1 WaitNotDistUntilsem2
unl-lift2 by blast

```

```

lemma WaitNotDistUntil:
⊢ (¬(f W g)) = ((¬ g) U (¬ f ∧ ¬ g))
using WaitNotDistUntilsem Valid-def by (metis )

```

9.2.15 UntilInduction

```

lemma LFPUntilsem1:
assumes ∀ n ≤ nlength σ.
      (g ( (ndropn n σ)) → h ( (ndropn n σ))) ∧
      (f ( (ndropn n σ)) ∧ n < nlength σ ∧ h ( (ndropn (Suc n) σ)) →
        h ( (ndropn n σ)))
      k ≤ nlength σ
      g ( (ndropn k σ))
      ∀ j < k. f ( (ndropn j σ))
shows   h ( σ)
using assms
proof (induct k arbitrary: σ )
case 0
then show ?case by auto
next
case (Suc k)
then show ?case
  proof -
    have 1: g (ndropn (Suc k) σ) → h (ndropn (Suc k) σ)
      using Suc.prem1(1) Suc.prem1(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 LFPUntilsemI[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.simpis(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: ⊢ (□ f ∨ f U g) = f 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

```

9.3 Theorems

lemma *NextFalseSUntil*:

$\vdash \bigcirc g = \#False \mathcal{U}^s g$

proof –

have 1: $\vdash \#False \mathcal{U} g = g$

using UntilNextUntil[of LIFT($\#False$) g] by auto

show ?thesis unfolding suntill-d-def using 1 inteq-reflection by force

qed

lemma *WNextUntil*:

$\vdash wnext(f \mathcal{U} g) = (empty \vee (\bigcirc f) \mathcal{U} (\bigcirc g))$

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 ∧ g) LIFT(f ∧ 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(¬ 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, hide-lams) 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 *WaitAbsorptionBox*:
 $\vdash f \mathcal{W} (\Box f) = \Box f$
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) ChopAssocB FiniteChopSkipEqvSkipChopFinite inteq-reflection next-d-def sometimes-d-def*)

have 22: $\vdash (\neg(\bigcirc f)) = (\text{empty} \vee \bigcirc(\neg f))$

using *WnextEqvEmptyOrNext*[of *LIFT*($\neg f$)] **unfolding** *wnext-d-def* **by** *simp*

have 23: $\vdash \text{inf} \longrightarrow \bigcirc(\neg f) = (\neg(\bigcirc f))$

using 22 *MoreAndInfEqvInf WnextAndMoreEqvNext WnextEqvEmptyOrNext* **by** *fastforce*

have 24: $\vdash \text{inf} \longrightarrow \Diamond(\bigcirc(\neg f)) = \Diamond(\neg(\bigcirc f))$

unfolding *sometimes-d-def* **using** 23

InfRightChopEqvChop[of *LIFT*($\bigcirc(\neg f)$) *LIFT*($(\neg(\bigcirc f))$) *LIFT*(*finite*)]

by *simp*

have 25: $\vdash \text{inf} \longrightarrow \bigcirc(\Diamond(\neg f)) = \Diamond(\neg(\bigcirc f))$

using 21 24 **by** *fastforce*

have 26: $\vdash \text{inf} \longrightarrow (\text{empty} \vee \bigcirc(\Diamond(\neg f))) = \bigcirc(\Diamond(\neg f))$

using *MoreAndInfEqvInf WnextAndMoreEqvNext WnextEqvEmptyOrNext* **by** *fastforce*

have 27: $\vdash \text{inf} \longrightarrow \text{wnext}(\Diamond(\neg f)) = \bigcirc(\Diamond(\neg f))$

by (*metis* 26 *WnextEqvEmptyOrNext inteq-reflection*)

have 28: $\vdash \text{inf} \longrightarrow (\neg(\bigcirc(\neg \Diamond(\neg f)))) = \Diamond(\neg \bigcirc f)$

using 21 24 27 **unfolding** *wnext-d-def* **by** *fastforce*

have 3: $\vdash \text{inf} \longrightarrow \bigcirc(\Box f) = \Box(\bigcirc f)$

unfolding *always-d-def* **using** 28 **by** *fastforce*

have 4: $\vdash \bigcirc(f \mathcal{U} g) = \bigcirc f \mathcal{U} \bigcirc g$

by (*simp add: NextUntil*)

have 5: $\vdash \bigcirc(\Box f \vee f \mathcal{U} g) = (\bigcirc(\Box f) \vee \bigcirc(f \mathcal{U} g))$

using 1 2 *int-iffI* **by** *blast*

have 6: $\vdash \text{inf} \longrightarrow \bigcirc(f \mathcal{W} g) = \bigcirc(\Box f \vee f \mathcal{U} g)$

by (*simp add: wait-d-def*)

have 7: $\vdash \text{inf} \longrightarrow \bigcirc(\Box f \vee f \mathcal{U} g) = (\bigcirc(\Box f) \vee \bigcirc(f \mathcal{U} g))$

using 5 **by** *auto*

have 8: $\vdash \text{inf} \longrightarrow (\bigcirc(\Box f) \vee \bigcirc(f \mathcal{U} g)) = (\Box(\bigcirc f) \vee \bigcirc f \mathcal{U} \bigcirc g)$

using 4 3 **by** *fastforce*

show *?thesis*

by (*metis* 5 8 *inteq-reflection wait-d-def*)

qed

lemma *WnextAlwaysEqvAlwaysWnext*:

$\vdash \text{finite} \longrightarrow \text{wnext}(\Box f) = \Box(\text{wnext } f)$

by (metis (mono-tags, lifting) BoxEqvFiniteYields FiniteChopSkipEqvSkipChopFinite
SkipYieldsEqvWeakNext YieldsYieldsEqvChopYields intI inteq-reflection unl-lift2)

lemma WaitExpand:

$\vdash f \mathcal{W} g = (g \vee (f \wedge \bigcirc(f \mathcal{W} g)))$

— nitpick finds counterexample, does not hold because of finite intervals

oops

lemma WaitExpand:

$\vdash f \mathcal{W} g = (g \vee (f \wedge \text{wnext}(f \mathcal{W} g)))$

proof —

have 1: $\vdash f \mathcal{W} g = (\Box f \vee f \mathcal{U} g)$

by (simp add: wait-d-def)

have 2: $\vdash \Box f = (f \wedge \text{wnext}(\Box f))$

by (simp add: BoxEqvAndWnextBox)

have 3: $\vdash f \mathcal{U} g = (g \vee (f \wedge \bigcirc(f \mathcal{U} g)))$

using UntilNextUntil **by** blast

have 4: $\vdash (f \wedge \text{wnext}(\Box f)) = (f \wedge (\text{empty} \vee \bigcirc(\Box f)))$

using 2 BoxEqvAndEmptyOrNextBox[of f] **by** fastforce

have 5: $\vdash \text{wnext}(f \mathcal{W} g) = (\text{empty} \vee \bigcirc(f \mathcal{W} g))$

using WnextEqvEmptyOrNext **by** blast

have 6: $\vdash (f \wedge (\text{empty} \vee \bigcirc(\Box f))) = ((f \wedge \text{empty}) \vee (f \wedge \bigcirc(\Box f)))$

by auto

have 7: $\vdash ((f \wedge \text{empty}) \vee (f \wedge \bigcirc(\Box f))) \vee$

$(g \vee (f \wedge \bigcirc(f \mathcal{U} g))) =$

$(g \vee (f \wedge (\text{empty} \vee \bigcirc(\Box f) \vee \bigcirc(f \mathcal{U} g))))$

by auto

have 8: $\vdash (\bigcirc(\Box f) \vee \bigcirc(f \mathcal{U} g)) = \bigcirc(\Box f \vee f \mathcal{U} g)$

by (metis ChopOrEqv Prop11 next-d-def)

show ?thesis

by (metis 1 2 3 4 5 6 7 8 inteq-reflection)

qed

lemma InfImpWnextEqnNext:

$\vdash \text{inf} \longrightarrow \text{wnext } f = \bigcirc f$

proof —

have 1: $\vdash \text{wnext } f = (\text{empty} \vee \bigcirc f)$

by (simp add: WnextEqvEmptyOrNext)

have 2: $\vdash \text{inf} \longrightarrow \text{more}$

using MoreAndInfEqvInf **by** auto

have 3: $\vdash \text{inf} \longrightarrow (\text{empty} \vee \bigcirc f) = \bigcirc f$

using 2 NotEmptyEqvMore **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*:

$\vdash f \mathcal{W} \#True$

by (*meson* *MP TrueW UntilImpWait UntilIntro*)

lemma *WaitLeftIdentity*:

$\vdash \#False \mathcal{W} g = g$

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 *WaitAbsorptiona*:

$\vdash (f \vee f \mathcal{W} g) = (f \vee g)$

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 *WaitAbsorptionb*:
 $\vdash (f \mathcal{W} g \vee g) = f \mathcal{W} g$
using *WaitInsertion*[*of g f*] **by** *auto*

lemma *WaitAbsorptionc*:
 $\vdash (f \mathcal{W} g \wedge g) = g$
using *WaitInsertion* **by** *fastforce*

lemma *WaitAbsorptiond*:
 $\vdash (f \mathcal{W} g \wedge (f \vee g)) = f \mathcal{W} g$
by (*meson Prop10 Prop11 WaitImpOr*)

lemma *WaitAbsorptione*:
 $\vdash (f \mathcal{W} g \vee (f \wedge g)) = f \mathcal{W} g$
unfolding *wait-d-def*
using *UntilAbsorp-d* **by** *fastforce*

lemma *WaitLeftAbsorption*:
 $\vdash f \mathcal{W} (f \mathcal{W} g) = f \mathcal{W} g$
by (*metis (no-types, lifting) BoxEqvBoxBox UntilUntil WaitAbsorptionBox WaitAbsorptiona WaitLeftDistOr inteq-reflection wait-d-def*)

lemma *WaitRightAbsorption*:
 $\vdash (f \mathcal{W} g) \mathcal{W} g = f \mathcal{W} g$
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 WaitAbsorptionb WaitImpOr WaitRightAbsorption int-eq wait-d-def*)

lemma *WaitRightUntilAbsorption*:
 $\vdash f \mathcal{W} (f \mathcal{U} g) = f \mathcal{W} g$
by (*metis UntilUntil WaitOrRule inteq-reflection wait-d-def*)

lemma *WaitLeftUntilAbsorption*:

$\vdash (f \mathcal{U} g) \mathcal{W} g = f \mathcal{U} g$
by (*metis Prop11 RightUntilAbsorp UntilAbsorp-b UntilImpWait WaitImpOr inteq-reflection*)

lemma *UntilRightWaitAbsorption*:

$\vdash f \mathcal{U} (f \mathcal{W} g) = f \mathcal{W} g$

using *UntilImpWait UntilIntro WaitLeftAbsorption* **by** *fastforce*

lemma *UntilLeftWaitAbsorption*:

$\vdash (f \mathcal{W} g) \mathcal{U} g = f \mathcal{U} g$

by (*metis UntilWait WaitRightAbsorption inteq-reflection*)

lemma *WaitDiamondAbsorption*:

$\vdash (\Diamond g) \mathcal{W} g = \Diamond g$

by (*metis DiamondEqvTrueUntil WaitLeftUntilAbsorption inteq-reflection*)

lemma *WaitAndBoxAbsorption*:

$\vdash (\Box f \wedge f \mathcal{W} g) = \Box f$

by (*meson BoxImpWait NotDiamondNotEqvBox Prop04 Prop10*)

lemma *WaitOrBoxAbsorption*:

$\vdash (\Box f \vee f \mathcal{W} g) = f \mathcal{W} g$

by (*metis UntilRightWaitAbsorption WaitLeftAbsorption inteq-reflection wait-d-def*)

lemma *WaitAndBoxImpBox*:

$\vdash f \mathcal{W} g \wedge \Box (\neg g) \longrightarrow \Box f$

by (*metis (no-types, hide-lams) 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 WaitLeftAbsorption 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 WaitAbsorptiond 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 WaitAbsorptione 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: $\vdash \Diamond(f \vee g) = \# \text{True} \mathcal{U} (f \vee g)$

by (*metis DiamondEqvTrueUntil*)

have 3: $\vdash \# \text{True} \mathcal{U} (f \vee g) = (\neg (f \vee g)) \mathcal{U} (f \vee g)$

using 2 *UntilRule* **by** *fastforce*

have 4: $\vdash (\neg (f \vee g)) \mathcal{U} (f \vee g) = (\neg f \wedge \neg g) \mathcal{U} (f \vee g)$

by (*metis UntilAbsorp-c int-eq int-simps(14) int-simps(33)*)

have 5: $\vdash (\neg f \wedge \neg g) \mathcal{U} (f \vee g) \longrightarrow (\neg f \wedge \neg g) \mathcal{U} f \vee (\neg f \wedge \neg g) \mathcal{U} g$

by (*simp add: UntilOrDist int-iffD1*)

have 6: $\vdash (\neg f \wedge \neg g) \mathcal{U} f \longrightarrow (\neg g) \mathcal{U} f$

by (*metis UntilAndRule int-iffD2 inteq-reflection lift-and-com*)

have 7: $\vdash (\neg f \wedge \neg g) \mathcal{U} g \longrightarrow (\neg f) \mathcal{U} g$

by (*metis UntilAndRule int-iffD2*)

have 8: $\vdash (\neg f \wedge \neg g) \mathcal{U} f \vee (\neg f \wedge \neg g) \mathcal{U} g \longrightarrow (\neg f) \mathcal{U} g \vee (\neg g) \mathcal{U} f$

using 6 7 **by** *fastforce*

have 9: $\vdash \Diamond(f \vee g) \longrightarrow (\neg f) \mathcal{U} g \vee (\neg g) \mathcal{U} f$

using 2 3 4 5 8 **by** *fastforce*

show *?thesis* **using** 1 9 **by** *fastforce*

qed

lemma *WaitOrder*:

$\vdash (\neg f) \mathcal{W} g \vee (\neg g) \mathcal{W} f$

proof –

have 1: $\vdash (\neg f) \mathcal{W} g = (\Box (\neg f) \vee (\neg f) \mathcal{U} g)$

by (*simp add: wait-d-def*)

have 2: $\vdash (\neg g) \mathcal{W} f = (\Box (\neg g) \vee (\neg g) \mathcal{U} f)$

by (*simp add: wait-d-def*)

have 3: $\vdash ((\Box (\neg f) \vee (\neg f) \mathcal{U} g) \vee (\Box (\neg g) \vee (\neg g) \mathcal{U} f)) =$
 $(\Box (\neg f) \vee \Box (\neg g)) \vee ((\neg f) \mathcal{U} g \vee (\neg g) \mathcal{U} f))$

by *auto*

have 4: $\vdash ((\neg f) \mathcal{U} g \vee (\neg g) \mathcal{U} f) = \Diamond(f \vee g)$

using *UntilOrder* **by** *blast*

have 5: $\vdash (\Box (\neg f) \vee \Box (\neg g)) = (\neg (\Diamond f) \vee \neg (\Diamond g))$

by (*simp add: always-d-def*)

have 6: $\vdash \Diamond(f \vee g) = (\Diamond f \vee \Diamond g)$

by (*simp add: ChopOrEqv sometimes-d-def*)

have 7: $\vdash (\Box (\neg f) \vee \Box (\neg g)) \vee \Diamond(f \vee g)$

using 5 6 **by** *fastforce*

show *?thesis*

using 1 2 4 7 **by** *fastforce*

qed

lemma *WaitImpOrder*:

$\vdash f \mathcal{W} g \wedge (\neg g) \mathcal{W} h \longrightarrow f \mathcal{W} h$

proof –

have 1: $\vdash f \mathcal{W} g = (\Box f \vee f \mathcal{U} g)$

```

  by (simp add: wait-d-def)
have 2:  $\vdash f \mathcal{W} h = (\Box f \vee f \mathcal{U} h)$ 
  by (simp add: wait-d-def)
have 3:  $\vdash (\neg g) \mathcal{W} h = (\Box (\neg g) \vee (\neg g) \mathcal{U} h)$ 
  by (simp add: wait-d-def)
have 4:  $\vdash \Box f \wedge (\Box (\neg g) \vee (\neg g) \mathcal{U} h) \longrightarrow (\Box f \vee f \mathcal{U} h)$ 
  by auto
have 5:  $\vdash (\Box f \vee f \mathcal{U} g) \wedge \Box (\neg g) \longrightarrow (\Box f \vee f \mathcal{U} h)$ 
  using 1 WaitAndBoxImpBox by fastforce
have 6:  $\vdash f \mathcal{U} g \wedge (\neg g) \mathcal{U} h \longrightarrow (\Box f \vee f \mathcal{U} h)$ 
  by (simp add: Prop05 UntilNotImp)
have 7:  $\vdash \Box f \wedge (\neg g) \mathcal{U} h \longrightarrow (\Box f \vee f \mathcal{U} h)$ 
  by auto
have 8:  $\vdash (\Box f \vee f \mathcal{U} g) \wedge (\neg g) \mathcal{U} h \longrightarrow (\Box f \vee f \mathcal{U} h)$ 
  using 6 7 by fastforce
have 9:  $\vdash (\Box f \vee f \mathcal{U} g) \wedge (\Box (\neg g) \vee (\neg g) \mathcal{U} h) \longrightarrow (\Box f \vee f \mathcal{U} h)$ 
  using 5 8 by fastforce
show ?thesis by (simp add: 9 wait-d-def)
qed

```

end

10 Pi operator for infinite and finite ITL

```

theory Pi
imports Until NELListKFilter
begin

```

This theory introduces the Pi operator [9, 6]. 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 [6].

10.1 Definitions

```

definition sfxfilt :: 'a nellist  $\Rightarrow$  ('a:: world) formula  $\Rightarrow$  'a nellist nellist
where sfxfilt xs f = (nfilter ( $\lambda$  ys. ys  $\models$  f) (ndropns xs))

```

```

definition pifilt :: 'a nellist  $\Rightarrow$  ('a:: world) formula  $\Rightarrow$  'a nellist
where pifilt xs f = (nmap ( $\lambda$ s. nnth s 0) (sfxfilt xs f))

```

```

definition pi-d :: ('a:: world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula
where pi-d F G  $\equiv$   $\lambda$ s. ( ( $\exists$  i  $\leq$  nlength s. (ndropn i s)  $\models$  F)  $\wedge$  ( (pifilt s F)  $\models$  G ) )

```

```

syntax
  -pi-d    :: [lift, lift]  $\Rightarrow$  lift          ((-  $\Pi$  -) [84,84] 83)

```

```

syntax (ASCII)
  -pi-d    :: [lift, lift]  $\Rightarrow$  lift          ((-  $\Pi$  -) [84,84] 83)

```

translations

$-pi-d \Rightarrow CONST\ pi-d$

definition $upi-d :: ('a :: world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula$

where $upi-d\ F\ G \equiv LIFT(\neg(F \Pi (\neg\ G)))$

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

$-upi-d \Rightarrow CONST\ upi-d$

10.2 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*:

assumes $(\exists\ i \leq nlength\ \sigma . (ndropn\ i\ \sigma) \models f)$

shows $nlength\ (pfilt\ \sigma\ f) = nlength\ (nmap\ (\lambda s . nnth\ s\ 0)\ (sfxfilt\ \sigma\ f))$

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

lemma *sfxfilt-pfilt-nnth*:

assumes $(\exists\ i \leq nlength\ \sigma . (ndropn\ i\ \sigma) \models f)$

$j \leq nlength\ (pfilt\ \sigma\ f)$

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*:

assumes $(\exists i \leq \text{nlength } \sigma. (\text{ndropn } i \ \sigma) \models f)$

shows $(\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{sfxfilt } \sigma \ f)) = \text{pifilt } \sigma \ f$

using *assms* **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 sfxfilter-help*)

lemma *sfxfilt-pifilt-nnth-ndropn*:

assumes $(\exists i \leq \text{nlength } \sigma. (\text{ndropn } i \ \sigma) \models f)$

$j \leq \text{nlength } (\text{pifilt } \sigma \ f)$

shows $\text{nnth } (\text{sfxfilt } \sigma \ f) \ j = \text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ j) \ \sigma$

proof –

have *0*: $\exists x \in \text{nset } (\text{ndropns } \sigma). f \ x$

by (*simp add: assms(1) sfxfilter-help*)

have *1*: $\text{nnth } (\text{sfxfilt } \sigma \ f) \ j = \text{nnth } (\text{nfilter } f (\text{ndropns } \sigma)) \ j$

by (*simp add: sfxfilt-def*)

have *2*: $\text{nnth } (\text{nfilter } f (\text{ndropns } \sigma)) \ j = \text{nnth } (\text{ndropns } \sigma) (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ j)$

using *nkfilter-nfilter[of ndropns 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 le-cases nnth-beyond*)

lemma *sfxfilt-nlength-imp*:
assumes $(\exists i \leq nlength \ \sigma. f \ (ndropn \ i \ \sigma) \wedge g \ (ndropn \ i \ \sigma))$
shows $nlength \ (sfxfilt \ \sigma \ (LIFT(f \wedge g))) \leq nlength \ (sfxfilt \ \sigma \ f)$
proof –
have 1: $nlength \ (sfxfilt \ \sigma \ (LIFT(f \wedge g))) =$
 $nlength \ (nfilter \ (\lambda ys. f \ ys \wedge g \ ys) \ (ndropns \ \sigma))$
by (*simp add: sfxfilt-def*)
have 2: $nlength \ (nfilter \ f \ (ndropns \ \sigma)) = nlength \ (sfxfilt \ \sigma \ f)$
by (*simp add: sfxfilt-def*)
have 3: $\exists x \in nset \ (ndropns \ \sigma). f \ x \wedge g \ x$
using *assms in-nset-ndropns* **by** *blast*
have 4: $nlength \ (nfilter \ (\lambda x. f \ x \wedge g \ x) \ (ndropns \ \sigma)) \leq nlength \ (nfilter \ f \ (ndropns \ \sigma))$
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 $(\exists i \leq nlength \ \sigma. f \ (ndropn \ i \ \sigma) \wedge g \ (ndropn \ i \ \sigma))$
shows $nlength \ (pfilt \ \sigma \ (LIFT(f \wedge g))) \leq nlength \ (pfilt \ \sigma \ f)$
using *assms*
by (*simp add: sfxfilt-nlength-imp pfilt-def*)

lemma *nellist-nset-sfxfilt [simp]*:
assumes $(\exists i \leq nlength \ xs. f \ (ndropn \ i \ xs))$
shows $(nset \ (sfxfilt \ xs \ f)) = \{ys. ys \in nset \ (ndropns \ xs) \wedge f \ (ys)\}$
using *assms*
proof –
have 1: $nset \ (sfxfilt \ xs \ f) = nset \ (nfilter \ f \ (ndropns \ xs))$
by (*simp add: sfxfilt-def*)
have 2: $\exists x \in nset \ (ndropns \ xs). f \ x$
using *assms using in-nset-ndropns* **by** *blast*
have 3: $nset \ (nfilter \ f \ (ndropns \ xs)) = \{ys. ys \in nset \ (ndropns \ xs) \wedge f \ ys\}$
using 2 *nset-nfilter[of ndropns xs f]* **by** *auto*
show ?thesis **by** (*simp add: 1 3*)
qed

lemma *subset-nfilter*:
assumes $\exists x \in nset \ xs. P \ x$
shows $nset(nfilter \ P \ xs) \leq nset(nfilter \ (\lambda x. P \ x \vee Q \ x) \ xs)$
proof –
have 1: $\exists x \in nset \ xs. P \ x \vee Q \ x$
using *assms* **by** *blast*
have 2: $nset(nfilter \ (\lambda x. P \ x \vee Q \ x) \ xs) = \{x \in nset \ xs. P \ x \vee Q \ x\}$
using *assms nset-nfilter[of xs $(\lambda x. P \ x \vee Q \ x)$]* **by** *blast*
have 3: $nset(nfilter \ P \ xs) = \{x \in nset \ xs. P \ x\}$
using *assms nset-nfilter[of xs P]* **by** (*simp add: Collect-conj-eq*)
have 4: $\{x \in nset \ xs. P \ x\} \leq \{x \in nset \ xs. P \ x \vee Q \ x\}$
by *auto*
show ?thesis **by** (*simp add: 2 3 4*)

qed

lemma *nellist-subset-sxfilt* [simp]:

assumes $(\exists i \leq \text{nlength } xs. f (\text{ndropn } i \text{ } xs))$

shows $(\text{nset } (\text{sxfilt } xs \ f)) \leq (\text{nset } (\text{sxfilt } xs \ (\text{LIFT}(f \vee g))))$

proof –

have 1: $\exists x \in \text{nset } (\text{ndropns } xs). f \ x$

using *assms* **using** *in-nset-ndropns* **by** *blast*

have 2: $\text{nset } (\text{sxfilt } xs \ f) = \text{nset } (\text{nfilter } f \ (\text{ndropns } xs))$

by (*simp add: sxfilt-def*)

have 3: $\text{nset } (\text{sxfilt } xs \ (\text{LIFT}(f \vee g))) = \text{nset } (\text{nfilter } (\lambda x. f \ x \vee g \ x) \ (\text{ndropns } xs))$

by (*simp add: sxfilt-def*)

have 4: $\text{nset } (\text{nfilter } f \ (\text{ndropns } xs)) \leq \text{nset } (\text{nfilter } (\lambda x. f \ x \vee g \ x) \ (\text{ndropns } xs))$

using 1 *subset-nfilter*[*of ndropns xs f g*] **by** *auto*

show ?thesis **using** 2 3 4 **by** *blast*

qed

lemma *nellist-nset-nmap*:

$(\text{nset } (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ xs)) = \{(\text{nnth } ys \ 0) \mid ys. ys \in \text{nset } xs\}$

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 $(\text{nset } (\text{pifilt } xs \ f)) = \{(\text{nnth } ys \ 0) \mid ys. ys \in \text{nset } (\text{ndropns } xs) \wedge f \ (ys)\}$

proof –

have 1: $(\text{nset } (\text{pifilt } xs \ f)) = (\text{nset } (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{sxfilt } xs \ f)))$

by (*simp add: pifilt-def*)

have 2: $(\text{nset } (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{sxfilt } xs \ f))) =$

$\{(\text{nnth } ys \ 0) \mid ys. ys \in \text{nset } (\text{sxfilt } xs \ f)\}$

using *nellist-nset-nmap*[*of sxfilt xs f*] **by** *auto*

have 3: $\{(\text{nnth } ys \ 0) \mid ys. ys \in \text{nset } (\text{sxfilt } xs \ f)\} =$

$\{(\text{nnth } ys \ 0) \mid ys. ys \in \text{nset } (\text{ndropns } xs) \wedge f \ (ys)\}$

using *assms* **by** *auto*

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

qed

lemma *nellist-nnth-sxfilt-in-nset*:

$x \in \text{nset } (\text{sxfilt } \sigma \ (\text{LIFT}(f \vee g))) =$

$(\exists i \leq \text{nlength } (\text{sxfilt } \sigma \ (\text{LIFT}(f \vee g))). x = (\text{nnth } (\text{sxfilt } \sigma \ (\text{LIFT}(f \vee g))) \ i))$

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

lemma *nfilter-nnth-or*:

assumes $\exists x \in \text{nset } xs. P \ x$

shows $\exists x \in \text{nset } (\text{nfilter } (\lambda x. P \ x \vee Q \ x) \ xs). P \ x$

proof –

have 0: $\exists x \in \text{nset } xs. P \ x \vee Q \ x$

using *assms* **by** *auto*

have 1: $\exists i \leq \text{nlength } (\text{nfilter } (\lambda x. P \ x \vee Q \ x) \ xs). P \ (\text{nnth } (\text{nfilter } P \ xs) \ i)$

using *assms nkfilter-nnth-aa*[*of xs* ($\lambda x. P \ x$)] *nkfilter-nnth-aa*

by (metis (mono-tags, lifting) le-cases le-zero-eq zero-enat-def)
obtain i **where** $2: i \leq \text{nlength}(\text{nfilter} (\lambda x. P x \vee Q x) xs) \wedge P (\text{nnth} (\text{nfilter} P xs) i) \wedge$
 $(\text{nnth} (\text{nfilter} P xs) i) \in \text{nset} (\text{nfilter} P xs)$
 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*:

$\sigma \models \neg ((\# \text{False}) \Pi f)$

by (*simp add: pi-d-def*)

lemma *pifilt-true*:

$\text{pifilt } \sigma (\text{LIFT}(\# \text{True})) = \sigma$

by (*simp add: pifilt-def sfxfilt-def nmap-first-ndropns*)

lemma *PiStatesem*:

$(\sigma \models (\text{init } w) \Pi f) =$

$((\exists i \leq \text{nlength } \sigma. w (\text{NNil} (\text{nnth } \sigma i))) \wedge f (\text{pifilt } \sigma (\text{LIFT}(\text{init } w))))$

by (*simp add: pi-d-def init-defs ndropn-nfirst*)

10.3 Soundness of Axioms

10.3.1 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*

10.3.2 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*

10.3.3 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*)

10.3.4 PiTrueEqvDiamond

lemma *PiTrueEqvDiamond*:

$\vdash f \Pi \#True = \Diamond f$

by (*simp add: Valid-def pi-d-def itl-defs*)

10.3.5 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*

10.3.6 UPiFalseEqvBoxNot:

lemma *UPiFalseEqvBoxNot*:

$\vdash f \Pi^u \#False = \Box (\neg f)$

by (*simp add: Valid-def upi-d-def pi-d-def itl-defs*)

10.3.7 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: $\forall n \leq \text{nlength } \sigma. f1 (\text{ndropn } n \sigma) = f2 (\text{ndropn } n \sigma)$

using *assms* **by** (*simp add: itl-defs*)

have 2: $(\sigma \models (f1 \Pi g)) = ((\exists i \leq \text{nlength } \sigma. f1 (\text{ndropn } i \sigma)) \wedge g (\text{pifilt } \sigma f1))$

by (*simp add: pi-d-def*)

have 3: $(\exists i \leq \text{nlength } \sigma. f1 (\text{ndropn } i \sigma)) = (\exists i \leq \text{nlength } \sigma. f2 (\text{ndropn } i \sigma))$

using 1 **by** *blast*

have 6: $(\exists i \leq \text{nlength } \sigma. f1 (\text{ndropn } i \sigma)) \implies$

$\{ (\text{ndropn } n \sigma) \mid n. n \leq \text{nlength } \sigma \wedge f1 (\text{ndropn } n \sigma) \} =$

$\{ (\text{ndropn } n \sigma) \mid n. n \leq \text{nlength } \sigma \wedge f2 (\text{ndropn } n \sigma) \}$

```

using 1 by blast
have 7: ( $\exists i \leq \text{nlength } \sigma. f1 (\text{ndropn } i \ \sigma) \implies (\text{sfxfilt } \sigma \ f1) = (\text{sfxfilt } \sigma \ f2)$ )
  by (metis 1 in-nset-ndropns nfilter-cong sfxfilt-def)
have 8: ( $(\exists i \leq \text{nlength } \sigma. f1 (\text{ndropn } i \ \sigma)) \wedge g (\text{pifilt } \sigma \ f1) =$ 
  ( $(\exists i \leq \text{nlength } \sigma. f2 (\text{ndropn } i \ \sigma)) \wedge g (\text{pifilt } \sigma \ f2)$ )
  by (metis 3 7 pifilt-def)
show ?thesis
  by (simp add: 8 pi-d-def)
qed
qed

```

```

lemma BoxEqvImpPiEqv:
 $\vdash \Box (f1 = f2) \longrightarrow (f1 \ \Pi \ g = f2 \ \Pi \ g)$ 
using BoxEqvImpPiEqvsem by (simp add: Valid-def, auto)

```

10.3.8 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

```

10.3.9 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:  $(\text{nnth } (\text{pifilt } \sigma \ f) \ ia) = (\text{nnth } (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{sfxfilt } \sigma \ f)) \ ia)$ 
    using assms(1) assms(2) assms(3) sfxfilt-pifilt-nnth by blast
  have 2:  $(\text{nnth } (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{sfxfilt } \sigma \ f)) \ ia) =$ 
     $(\lambda s. \text{nnth } s \ 0) (\text{nnth } (\text{sfxfilt } \sigma \ f) \ ia)$ 
    by (metis assms(3) nlength-nmap nnth-nmap pifilt-def)
  have 3:  $f (\text{nnth } (\text{sfxfilt } \sigma \ f) \ ia)$ 
    using sfxfilt-nnth
    by (metis assms(1) assms(2) assms(3) nlength-nmap pifilt-def)
  have 4:  $\text{nnth } (\text{sfxfilt } \sigma \ f) \ ia = \text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 (\text{ndropns } \sigma)) \ ia) \ \sigma$ 
    using sfxfilt-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 $i \leq \text{nlength } \sigma$
 $f \text{ (ndropn } i \text{ } \sigma)$
 $w \text{ (NNil (nnth } \sigma \text{ } i))$
shows $\exists j \leq \text{nlength } (\text{pifilt } \sigma \text{ } f). w \text{ (NNil (nnth } (\text{pifilt } \sigma \text{ } f) \text{ } j))$
proof –
have 1: $\exists j \leq \text{nlength } (\text{sfxfilt } \sigma \text{ } f). f \text{ (nnth } (\text{sfxfilt } \sigma \text{ } f) \text{ } j)$
using *assms pifilt-exists* **by** *blast*
have 2: $(\text{LIFT } (\text{init } w)) \text{ (ndropn } i \text{ } \sigma)$
by *(simp add: assms init-defs ndropn-nfirst)*
have 3: $\exists j \leq \text{nlength } (\text{sfxfilt } \sigma \text{ } (\text{LIFT}(\text{init } w))). (\text{LIFT}(\text{init } w)) \text{ (nnth } (\text{sfxfilt } \sigma \text{ } (\text{LIFT}(\text{init } w))) \text{ } j)$
using *pifilt-exists 2 assms* **by** *blast*
have 4: $(\text{LIFT } (f \wedge \text{init } w)) \text{ (ndropn } i \text{ } \sigma)$
by *(simp add: 2 assms)*
have 5: $\exists j \leq \text{nlength } (\text{sfxfilt } \sigma \text{ } (\text{LIFT}(f \wedge \text{init } w))).$
 $(\text{LIFT}(f \wedge \text{init } w)) \text{ (nnth } (\text{sfxfilt } \sigma \text{ } (\text{LIFT}(f \wedge \text{init } w))) \text{ } j)$
using *pifilt-exists 4 assms* **by** *blast*
have 6: $\exists i \leq \text{nlength } \sigma. \text{ndropn } i \text{ } \sigma \models f \wedge \text{init } w$
using 4 *assms* **by** *blast*
have 7: $\exists j \leq \text{nlength } (\text{sfxfilt } \sigma \text{ } (\text{LIFT}((f \wedge \text{init } w) \vee (f \wedge \neg (\text{init } w))))).$
 $(\text{LIFT}(f \wedge \text{init } w)) \text{ (nnth } (\text{sfxfilt } \sigma \text{ } (\text{LIFT}((f \wedge \text{init } w) \vee (f \wedge \neg (\text{init } w)))) \text{ } j)$
using 6 *sfxfilt-nnth-or[of* σ *LIFT(f ∧ init w) LIFT(f ∧ ¬(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: $(\text{sfxfilt } \sigma \text{ } (\text{LIFT}((f \wedge \text{init } w) \vee (f \wedge \neg (\text{init } w)))) =$
 $(\text{sfxfilt } \sigma \text{ } f)$
using 8 **by** *(simp add: sfxfilt-def)*
have 10: $\exists j \leq \text{nlength } (\text{sfxfilt } \sigma \text{ } f).$
 $(\text{LIFT}(f \wedge \text{init } w)) \text{ (nnth } (\text{sfxfilt } \sigma \text{ } f) \text{ } j)$
using 7 9 **by** *auto*
have 11: $\text{nlength } (\text{sfxfilt } \sigma \text{ } f) = \text{nlength } (\text{pifilt } \sigma \text{ } f)$
by *(simp add: pifilt-def)*
have 12: $\exists j \leq \text{nlength } (\text{pifilt } \sigma \text{ } f).$
 $(\text{LIFT}(\text{init } w)) \text{ (nnth } (\text{sfxfilt } \sigma \text{ } f) \text{ } j)$
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:*

$((\exists i \leq \text{nlength } \sigma. f \text{ (ndropn } i \text{ } \sigma)) \wedge$
 $(\exists i \leq \text{nlength } (\text{pifilt } \sigma \text{ } f). w \text{ (NNil (nnth } (\text{ndropn } i \text{ } (\text{pifilt } \sigma \text{ } f)) \text{ } 0)) \text{ })) =$
 $(\exists i \leq \text{nlength } \sigma. f \text{ (ndropn } i \text{ } \sigma) \wedge w \text{ (NNil (nnth } (\text{ndropn } i \text{ } \sigma) \text{ } 0))))$
using *PiAssocsem1[of - σ f - w] PiAssocsem2[of - σ f w]* **by** *fastforce*

lemma *PiAssocsemb:*

$((\exists i \leq \text{nlength } \sigma. f \text{ (ndropn } i \text{ } \sigma)) \wedge$
 $(\exists i \leq \text{nlength } (\text{pifilt } \sigma \text{ } f). (\text{LIFT}(\text{init } w)) \text{ (ndropn } i \text{ } (\text{pifilt } \sigma \text{ } f)) \text{ })) =$
 $(\exists i \leq \text{nlength } \sigma. (\text{LIFT}(f \wedge \text{init } w)) \text{ (ndropn } i \text{ } \sigma))$
using *PiAssocsem1[of - σ f - w] PiAssocsem2[of - σ f w]*

by (simp add: init-defs ndropn-nfirst) blast

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*:

assumes ($\exists i \leq \text{nlength } xs . (\lambda s . s \models \text{init } w) (\text{ndropn } i \text{ } xs)$)
shows ($\text{pfilt } xs (\lambda s . s \models \text{init } w) = \text{nfilter } (\lambda y . w ((\text{NNil } y))) xs$)
using *assms*
proof (simp add: init-defs pfilt-def sxfilt-def)
assume *a0*: $\exists i . \text{enat } i \leq \text{nlength } xs \wedge w (\text{NNil } (\text{nfirst } (\text{ndropn } i \text{ } xs)))$
show $\text{nmap } (\lambda s . \text{nnth } s \text{ } 0) (\text{nfilter } (\lambda s . w (\text{NNil } (\text{nfirst } s)))) (\text{ndropns } xs) =$
 $\text{nfilter } (\lambda y . w (\text{NNil } y)) xs$
proof –
have 1: $\bigwedge x . ((\lambda y . w ((\text{NNil } y))) \circ (\lambda s . \text{nnth } s \text{ } 0)) x = (\lambda s . w (\text{NNil } (\text{nfirst } s))) x$
by *simp* (metis *ndropn-0 ndropn-nfirst*)
have 2: $((\lambda y . w ((\text{NNil } y))) \circ (\lambda s . \text{nnth } s \text{ } 0)) = (\lambda s . w (\text{NNil } (\text{nfirst } s)))$
using 1 **by** *blast*
have 3: $\exists x \in \text{nset } (\text{nmap } (\lambda s . \text{nnth } s \text{ } 0) (\text{ndropns } xs)) . (\lambda y . w ((\text{NNil } y))) x$
by (metis *a0 ndropn-nfirst nmap-first-ndropns pfiltinit-help*)
have 4: $\text{nmap } (\lambda s . \text{nnth } s \text{ } 0) (\text{nfilter } (\lambda s . w (\text{NNil } (\text{nfirst } s)))) (\text{ndropns } xs) =$
 $\text{nfilter } (\lambda y . w ((\text{NNil } y))) (\text{nmap } (\lambda s . \text{nnth } s \text{ } 0) (\text{ndropns } xs))$
by (metis (*mono-tags, lifting*) 1 3 *nfilter-cong nfilter-nmap*)
have 5: $(\text{nmap } (\lambda s . \text{nnth } s \text{ } 0) (\text{ndropns } xs)) = xs$
by (simp add: *nmap-first-ndropns*)
show *?thesis*
by (simp add: 4 5)
qed
qed

lemma *pfilt-init-a*:

assumes ($\exists i . (\text{enat } i) \leq \text{nlength } xs \wedge w (\text{NNil } (\text{nnth } xs \text{ } i))$)
shows ($\text{pfilt } xs (\lambda s . w (\text{NNil } (\text{nnth } s \text{ } 0))) = \text{nfilter } (\lambda y . w (\text{NNil } y)) xs$)
proof –
have 1: $(\exists i \leq \text{nlength } xs . (\lambda s . s \models \text{init } w) (\text{ndropn } i \text{ } xs))$
by (simp add: *assms init-defs ndropn-nfirst*)
have 2: $(\text{pfilt } xs (\lambda s . s \models \text{init } w)) = (\text{pfilt } xs (\lambda s . w (\text{NNil } (\text{nnth } s \text{ } 0))))$
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* 1 2
by *fastforce*
qed

lemma *pfilt-pfilt* :

```

assumes ( $\exists i \leq \text{nlength } xs. f \text{ (ndropn } i \text{ xs)}$ )
  ( $\exists i \leq \text{nlength } (\text{pfilt } xs \text{ f}). w \text{ (NNil (nnth (ndropn } i \text{ (pfilt } xs \text{ f)) } 0))}$ )
shows ( $\text{pfilt } (\text{pfilt } xs \text{ f}) \text{ (LIFT(init } w)) = \text{pfilt } xs \text{ (LIFT(f } \wedge \text{ init } w))}$ )
proof –
  have 1:  $\exists i \leq \text{nlength } (\text{pfilt } xs \text{ f}). (\text{LIFT(init } w)) \text{ (ndropn } i \text{ (pfilt } xs \text{ f)})$ 
    using assms by (simp add: init-defs ndropn-nfirst)
  have 2:  $(\text{pfilt } (\text{pfilt } xs \text{ f}) \text{ (LIFT(init } w))) =$ 
     $\text{nfilter } (\lambda y. w \text{ (NNil } y)) (\text{pfilt } xs \text{ f})$ 

    using 1 pfilt-init[of (pfilt xs f) w] by auto
  have 3:  $(\text{pfilt } xs \text{ f}) =$ 
     $\text{nmap } (\lambda s. \text{nnth } s \text{ } 0) \text{ (sfxfilt } xs \text{ f)}$ 
    by (simp add: assms sfxfilt-pfilt)
  have 4:  $(\text{sfxfilt } xs \text{ f}) = \text{nfilter } (\lambda ys. f \text{ ys}) \text{ (ndropns } xs)$ 
    using sfxfilt-def by blast
  have 5:  $(\text{pfilt } xs \text{ f}) = \text{nmap } (\lambda s. \text{nnth } s \text{ } 0) \text{ (nfilter } (\lambda ys. f \text{ ys}) \text{ (ndropns } xs))$ 
    by (simp add: 3 4)
  have 6:  $\text{nfilter } (\lambda y. w \text{ (NNil } y)) (\text{pfilt } xs \text{ f}) =$ 
     $\text{nfilter } (\lambda y. w \text{ (NNil } y)) (\text{nmap } (\lambda s. \text{nnth } s \text{ } 0) \text{ (nfilter } (\lambda ys. f \text{ ys}) \text{ (ndropns } xs)))$ 
    using 5 by simp
  have 7:  $\text{nfilter } (\lambda y. w \text{ (NNil } y)) (\text{nmap } (\lambda s. \text{nnth } s \text{ } 0) \text{ (nfilter } (\lambda ys. f \text{ ys}) \text{ (ndropns } xs))) =$ 
     $\text{nmap } (\lambda s. \text{nnth } s \text{ } 0) \text{ (nfilter } ((\lambda y. w \text{ (NNil } y)) \circ (\lambda s. \text{nnth } s \text{ } 0)) \text{ (nfilter } (\lambda ys. f \text{ ys}) \text{ (ndropns } xs)))$ 
    using assms 3 4 by (simp add: nfilter-nmap pfiltinit-help)
  have 8:  $\exists x \in \text{nset } (\text{nfilter } (\lambda ys. f \text{ ys}) \text{ (ndropns } xs)). ((\lambda y. w \text{ (NNil } y)) \circ (\lambda s. \text{nnth } s \text{ } 0)) \text{ } x$ 
    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:  $\exists x \in \text{nset } (\text{ndropns } xs). (\lambda ys. f \text{ ys}) \text{ } x$ 
    using assms by (simp add: sfxfilter-help)
  have 10:  $\exists x \in \text{nset } (\text{ndropns } xs). ((\lambda y. w \text{ (NNil } y)) \circ (\lambda s. \text{nnth } s \text{ } 0)) \text{ } x \wedge (\lambda ys. f \text{ ys}) \text{ } x$ 
    using 8 9
    using exist-conj[of f (ndropns xs) ((\lambda y. w (NNil y)) \circ (\lambda s. nnth s 0))] by fastforce
  have 11:  $\text{nfilter } ((\lambda y. w \text{ (NNil } y)) \circ (\lambda s. \text{nnth } s \text{ } 0)) \text{ (nfilter } (\lambda ys. f \text{ ys}) \text{ (ndropns } xs)) =$ 
     $\text{nfilter } (\lambda zs. ((\lambda y. w \text{ (NNil } y)) \circ (\lambda s. \text{nnth } s \text{ } 0)) \text{ } zs \wedge (\lambda ys. f \text{ ys}) \text{ } zs) \text{ (ndropns } xs)$ 
    using nfilter-nfilter[of (\lambda ys. f ys) (ndropns xs) ((\lambda y. w (NNil y)) \circ (\lambda s. nnth s 0))] ]
    8 10 by blast
  have 12:  $\exists i \leq \text{nlength } xs. (\lambda zs. ((\lambda y. w \text{ (NNil } y)) \circ (\lambda s. \text{nnth } s \text{ } 0)) \text{ } zs \wedge (\lambda ys. f \text{ ys}) \text{ } zs) \text{ (ndropn } i \text{ xs)}$ 
    by (metis 10 in-nset-ndropns)
  have 13:  $(\text{nfilter } (\lambda zs. ((\lambda y. w \text{ (NNil } y)) \circ (\lambda s. \text{nnth } s \text{ } 0)) \text{ } zs \wedge (\lambda ys. f \text{ ys}) \text{ } zs) \text{ (ndropns } xs))$ 
     $= (\text{sfxfilt } xs \text{ } (\lambda zs. ((\lambda y. w \text{ (NNil } y)) \circ (\lambda s. \text{nnth } s \text{ } 0)) \text{ } zs \wedge (\lambda ys. f \text{ ys}) \text{ } zs))$ 
    by (simp add: sfxfilt-def)
  have 14:  $\exists i \leq \text{nlength } xs. (\lambda zs. ((\lambda y. w \text{ (NNil } y)) \circ (\lambda s. \text{nnth } s \text{ } 0)) \text{ } zs \wedge (\lambda ys. f \text{ ys}) \text{ } zs) \text{ (ndropn } i \text{ xs)}$ 
    using 12 by blast
  have 15:  $\text{nmap } (\lambda s. \text{nnth } s \text{ } 0)$ 
     $((\text{sfxfilt } xs \text{ } (\lambda zs. ((\lambda y. w \text{ (NNil } y)) \circ (\lambda s. \text{nnth } s \text{ } 0)) \text{ } zs \wedge (\lambda ys. f \text{ ys}) \text{ } zs)) \text{ }) =$ 
     $\text{pfilt } xs \text{ } (\lambda zs. ((\lambda y. w \text{ (NNil } y)) \circ (\lambda s. \text{nnth } s \text{ } 0)) \text{ } zs \wedge (\lambda ys. f \text{ ys}) \text{ } zs)$ 
    using 14 by (simp add: sfxfilt-pfilt)
  have 16:  $\bigwedge xs. (\lambda zs. ((\lambda y. w \text{ (NNil } y)) \circ (\lambda s. \text{nnth } s \text{ } 0)) \text{ } zs \wedge (\lambda ys. f \text{ ys}) \text{ } zs) \text{ } xs =$ 
     $(\text{LIFT}(f \wedge \text{init } w)) \text{ } xs$ 
    by (simp add: init-defs (metis ndropn-0 ndropn-nfirst))

```

```

have 17: pifilt xs (λ zs. ((λ y. w (NNil y)) ∘ (λ s. nnth s 0)) zs ∧ (λ ys. f ys) zs) =
  pifilt xs (LIFT(f ∧ init w))
  using 16 by presburger
show ?thesis
  using 11 13 15 17 2 3 4 7 by auto
qed

```

lemma *PiAssocsem*:

```

σ ⊨ f Π ((init w) Π g) = (f ∧ (init w)) Π g
proof (auto simp add: pi-d-def init-defs)
  fix i
  fix ia
  assume a0: g (pifilt (pifilt σ f) (LIFT(init w)))
  assume a1: (enat i) ≤ nlength σ
  assume a2: f (ndropn i σ)
  assume a3: (enat ia) ≤ nlength (pifilt σ f)
  assume a4: w (NNil (nfirst (ndropn ia (pifilt σ f))))
  show ∃ i. enat i ≤ nlength σ ∧ f (ndropn i σ) ∧ w (NNil (nfirst (ndropn i σ)))
    using a0 a1 a2 a3 a4 PiAssocsem1
    by (metis ndropn-nfirst nnth-zero-ndropn)
next
  fix i
  fix ia
  assume a0: g (pifilt (pifilt σ f) (LIFT(init w)))
  assume a1: (enat i) ≤ nlength σ
  assume a2: f (ndropn i σ)
  assume a3: (enat ia) ≤ nlength (pifilt σ f)
  assume a4: w (NNil (nfirst (ndropn ia (pifilt σ f))))
  show g (pifilt σ (LIFT(f ∧ init w)))
    using a0 a1 a2 a3 a4 by (metis ndropn-nfirst nnth-zero-ndropn pifilt-pifilt)
next
  fix i
  assume a0: g (pifilt σ (LIFT(f ∧ init w)))
  assume a1: (enat i) ≤ nlength σ
  assume a2: f (ndropn i σ)
  assume a3: w (NNil (nfirst (ndropn i σ)))
  show ∃ i. enat i ≤ nlength (pifilt σ f) ∧ w (NNil (nfirst (ndropn i (pifilt σ f))))
    using a0 a1 a2 a3 by (metis PiAssocsem2 ndropn-nfirst)
next
  fix i
  assume a0: g (pifilt σ (LIFT(f ∧ init w)))
  assume a1: (enat i) ≤ nlength σ
  assume a2: f (ndropn i σ)
  assume a3: w (NNil (nfirst (ndropn i σ)))
  show g (pifilt (pifilt σ f) (LIFT(init w)))
    using a0 a1 a2 a3 by (metis PiAssocsem2 ndropn-nfirst nnth-zero-ndropn pifilt-pifilt)
qed

```

lemma *PiAssoc*:

```

⊢ f Π ((init w) Π g) = (f ∧ (init w)) Π g

```


using *PiAssocsem Valid-def* by *blast*

10.3.10 PiNotEqvDiamondAndNotPi

lemma *PiNotEqvDiamondAndNotPisem*:

$\sigma \models f \sqcap (\neg g) = (\Diamond f \wedge \neg(f \sqcap g))$

by (*simp add: pi-d-def itl-defs*) *blast*

lemma *PiNotEqvDiamondAndNotPi*:

$\vdash f \sqcap (\neg g) = (\Diamond f \wedge \neg(f \sqcap g))$

using *PiNotEqvDiamondAndNotPisem Valid-def* by *blast*

10.3.11 PiSChopDist

lemma *PiSChopDistsema*:

assumes $(\sigma \models (\text{init } w) \sqcap (g \smallfrown h))$

shows $(\sigma \models ((\text{init } w) \sqcap g) \smallfrown ((\text{init } w) \wedge ((\text{init } w) \sqcap h)))$

proof –

have 1: $(\sigma \models (\text{init } w) \sqcap (g \smallfrown h))$

using *assms* by *auto*

have 2: $((\exists i \leq \text{nlength } \sigma. (\text{LIFT}(\text{init } w)) (\text{ndropn } i \ \sigma)) \wedge ((\text{pifilt } \sigma (\text{LIFT}(\text{init } w))) \models g \smallfrown h))$

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: $((\text{pifilt } \sigma (\text{LIFT}(\text{init } w))) \models g \smallfrown h)$

using 2 by *auto*

have 5: $\text{nfilter } (\lambda y. w ((\text{NNil } y))) \ \sigma \models g \smallfrown h$

using *pifilt-init*

using 2 by *fastforce*

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: $(\text{ntaken } n (\text{nfilter } (\lambda y. w (\text{NNil } y)) \ \sigma)) = (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{ntaken } ((\text{nnth } (\text{nkfilter } (\lambda y. w (\text{NNil } y)) \ 0 \ \sigma) \ n)) \ \sigma))$

by (*simp add: 7 9 nfilter-nkfilter-ntaken-1 nkfilter-nlength*)

have 11: $(\text{ndropn } n (\text{nfilter } (\lambda y. w (\text{NNil } y)) \ \sigma)) = (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{ndropn } ((\text{nnth } (\text{nkfilter } (\lambda y. w (\text{NNil } y)) \ 0 \ \sigma) \ n)) \ \sigma))$

by (*simp add: 7 9 nfilter-nkfilter-ndropn-1 nkfilter-nlength*)

have 12: $g (\text{nfilter } (\lambda y. w (\text{NNil } y)) (\text{ntaken } ((\text{nnth } (\text{nkfilter } (\lambda y. w (\text{NNil } y)) \ 0 \ \sigma) \ n)) \ \sigma))$

using 10 7 by *auto*

have 13: $h \text{ (nfilter } (\lambda y. w \text{ (NNil } y)) \text{ (ndropn ((nnth (nkfilter } (\lambda y. w \text{ (NNil } y)) 0 \sigma) n)) \sigma))}$
using 11 7 by auto
have 14: $((\text{nnth (nkfilter } (\lambda y. w \text{ (NNil } y)) 0 \sigma) n)) \leq \text{nlength } \sigma$
using 7 9 nkfilter-upperbound[of $\sigma \text{ (}\lambda y. w \text{ (NNil } y)) - 0]$
by (metis gen-nlength-def nkfilter-nlength nlength-code)
have 15: $w \text{ (NNil (nnth (ndropn ((nnth (nkfilter } (\lambda y. w \text{ (NNil } y)) 0 \sigma) n)) \sigma) 0))}$
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
 $(\text{metis (mono-tags, lifting) 7 nkfilter-nlength nnth-nmap})$
have 16: $(\exists i. (\text{enat } i) \leq \text{nlength (ntaken ((nnth (nkfilter } (\lambda y. w \text{ (NNil } y)) 0 \sigma) n)) \sigma) \wedge$
 $w \text{ (NNil (nnth (ndropn i (ntaken ((nnth (nkfilter } (\lambda y. w \text{ (NNil } y)) 0 \sigma) n)) \sigma)) 0)))$
using 14 15
by (metis le-cases min.orderE ndropn-0 ndropn-nfirst ntaken-nlength ntaken-nnth)
have 17: $(\exists i. (\text{enat } i) \leq \text{nlength (ndropn ((nnth (nkfilter } (\lambda y. w \text{ (NNil } y)) 0 \sigma) n)) \sigma) \wedge$
 $w \text{ (NNil (nnth (ndropn (i + ((nnth (nkfilter } (\lambda y. w \text{ (NNil } y)) 0 \sigma) n))) \sigma) 0)))$
using 15 by (metis add.left-neutral zero-enat-def zero-le)
have 18: $(\exists n. (\text{enat } n) \leq \text{nlength } \sigma \wedge$
 $(\exists i. (\text{enat } i) \leq \text{nlength (ntaken } n \sigma) \wedge w \text{ (NNil (nnth (ndropn i (ntaken } n \sigma)) 0)) \wedge$
 $g \text{ (nfilter } (\lambda y. w \text{ (NNil } y)) \text{ (ntaken } n \sigma)) \wedge$
 $w \text{ (NNil (nnth (ndropn } n \sigma) 0)) \wedge$
 $(\exists i. (\text{enat } i) \leq \text{nlength (ndropn } n \sigma) \wedge w \text{ (NNil (nnth (ndropn (i + n) \sigma) 0))) \wedge$
 $h \text{ (nfilter } (\lambda y. w \text{ (NNil } y)) \text{ (ndropn } n \sigma)))$
using 12 13 14 15 16 17 by blast
have 181: $(\exists n. (\text{enat } n) \leq \text{nlength } \sigma \wedge (\exists i. (\text{enat } i) \leq \text{nlength (ntaken } n \sigma) \wedge w \text{ (NNil (nnth (ntaken } n \sigma)$
 $i))))$
using 18 by auto
have 182: $(\exists n. (\text{enat } n) \leq \text{nlength } \sigma \wedge (\exists i. (\text{enat } i) \leq \text{nlength (ndropn } n \sigma) \wedge w \text{ (NNil (nnth } \sigma (i + n)))$
 $))$
using 18 by auto
have 19: $(\exists n. (\text{enat } n) \leq \text{nlength } \sigma \wedge$
 $(\exists i. (\text{enat } i) \leq \text{nlength (ntaken } n \sigma) \wedge w \text{ (NNil (nnth (ndropn i (ntaken } n \sigma)) 0)) \wedge$
 $g \text{ (pifilt (ntaken } n \sigma) \text{ (LIFT(init } w))) \wedge$
 $w \text{ (NNil (nnth (ndropn } n \sigma) 0)) \wedge$
 $(\exists i. (\text{enat } i) \leq \text{nlength (ndropn } n \sigma) \wedge w \text{ (NNil (nnth (ndropn (i + n) \sigma) 0)) \wedge$
 $h \text{ (pifilt (ndropn } n \sigma) \text{ (LIFT(init } w)))))$
using 18 pifilt-init[of - w]
proof (simp add: init-defs ndropn-nfirst)
obtain nn :: nat and nna :: nat and nnb :: nat where
 $f1: \text{enat } nn \leq \text{nlength } \sigma \wedge$
 $\text{enat } nna \leq \text{nlength (ntaken } nn \sigma) \wedge$
 $w \text{ (NNil (nnth (ndropn nna (ntaken } nn \sigma)) 0)) \wedge$
 $g \text{ (nfilter } (\lambda a. w \text{ (NNil } a)) \text{ (ntaken } nn \sigma)) \wedge$
 $w \text{ (NNil (nnth (ndropn } nn \sigma) 0)) \wedge$
 $(\text{enat } nnb \leq \text{nlength (ndropn } nn \sigma) \wedge$
 $w \text{ (NNil (nnth (ndropn (nnb + nn) \sigma) 0))) \wedge$
 $h \text{ (nfilter } (\lambda a. w \text{ (NNil } a)) \text{ (ndropn } nn \sigma))$
by (metis 18)
have $\forall as. \text{pifilt } as \text{ (LIFT(init } w)) = \text{nfilter } (\lambda a. w \text{ (NNil } a)) as \vee \neg (\text{LIFT(init } w)) as$
by (metis $\bigwedge xs. \exists i. \text{enat } i \leq \text{nlength } xs \wedge (\text{LIFT(init } w)) \text{ (ndropn } i xs) \implies$
 $\text{pifilt } xs \text{ (LIFT(init } w)) = \text{nfilter } (\lambda y. w \text{ (NNil } y)) xs \text{ i0-lb ndropn-0 zero-enat-def}$

then have $f2$: $\text{pifilt } (\text{ndropn } nn \ \sigma) \ (LIFT(\text{init } w)) = \text{nfilter } (\lambda a. w \ (NNil \ a)) \ (\text{ndropn } nn \ \sigma)$
using $f1$ **by** $(\text{metis } (\text{no-types}) \ \text{init-defs } \text{ndropn-0} \ \text{ndropn-nfirst} \ \text{ntaken-0})$
have $\text{pifilt } (\text{ntaken } nn \ \sigma) \ (LIFT(\text{init } w)) = \text{nfilter } (\lambda a. w \ (NNil \ a)) \ (\text{ntaken } nn \ \sigma) \vee$
 $\neg (LIFT(\text{init } w)) \ (\text{ndropn } nna \ (\text{ntaken } nn \ \sigma))$
using $f1$ **by** $(\text{metis } \langle \wedge xs. \exists i. \text{enat } i \leq \text{nlength } xs \wedge (LIFT(\text{init } w)) \ (\text{ndropn } i \ xs) \implies$
 $\text{pifilt } xs \ (LIFT(\text{init } w)) = \text{nfilter } (\lambda y. w \ (NNil \ y)) \ xs \rangle)$
then have $f3$: $\text{pifilt } (\text{ntaken } nn \ \sigma) \ (LIFT(\text{init } w)) = \text{nfilter } (\lambda a. w \ (NNil \ a)) \ (\text{ntaken } nn \ \sigma)$
using $f1$ **by** $(\text{simp add: init-defs ndropn-nfirst})$
have $\text{ntaken } 0 \ (\text{ndropn } nn \ \sigma) = \text{ndropn } nn \ (\text{ntaken } nn \ \sigma)$
using $f1$ **by** $(\text{metis add.left-neutral ntaken-ndropn-swap zero-enat-def})$
then show $\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$
 $(\exists na \leq n. \text{enat } na \leq \text{nlength } \sigma \wedge w \ (NNil \ (\text{nnth } (\text{ntaken } n \ \sigma) \ na)) \wedge$
 $g \ (\text{pifilt } (\text{ntaken } n \ \sigma) \ (LIFT(\text{init } w))) \wedge w \ (NNil \ (\text{nnth } \sigma \ n)) \wedge$
 $(\exists na. \text{enat } na \leq \text{nlength } \sigma - \text{enat } n \wedge w \ (NNil \ (\text{nnth } \sigma \ (na + n))) \wedge$
 $h \ (\text{pifilt } (\text{ndropn } n \ \sigma) \ (LIFT(\text{init } w))))$
using $f3 \ f2 \ f1$
by $(\text{metis } (\text{no-types}) \ \text{add.left-neutral } i0\text{-lb } \text{le-add2} \ \text{ndropn-0} \ \text{ndropn-nfirst}$
 $\text{ntaken-0} \ \text{ntaken-NNil} \ \text{zero-enat-def})$

qed

have 20 : $((\exists n. (\text{enat } n) \leq \text{nlength } \sigma \wedge$
 $(\exists i. (\text{enat } i) \leq \text{nlength } (\text{ntaken } n \ \sigma) \wedge (LIFT(\text{init } w)) \ (\text{ndropn } i \ (\text{ntaken } n \ \sigma))) \wedge$
 $g \ (\text{pifilt } (\text{ntaken } n \ \sigma) \ (LIFT(\text{init } w))) \wedge$
 $(LIFT(\text{init } w)) \ (\text{ndropn } n \ \sigma) \wedge$
 $(\exists i. (\text{enat } i) \leq \text{nlength } (\text{ndropn } n \ \sigma) \wedge (LIFT(\text{init } w)) \ (\text{ndropn } (i + n) \ \sigma))$
 $\wedge h \ (\text{pifilt } (\text{ndropn } n \ \sigma) \ (LIFT(\text{init } w))))$

using 19

by $(\text{metis } \text{init-defs } \text{ndropn-nfirst} \ \text{nnth-zero-ndropn} \ \text{ntaken-0})$

have 21 : $(\sigma \models ((\text{init } w) \ \Pi \ g) \neg ((\text{init } w) \wedge ((\text{init } w) \ \Pi \ h)))$

using 20 **by** $(\text{simp add: add.commute itl-defs ndropn-ndropn pi-d-def})$

show $?thesis$ **by** $(\text{simp add: } 21)$

qed

lemma $PiSChopDistsemb$:

assumes $(\sigma \models ((\text{init } w) \ \Pi \ g) \neg ((\text{init } w) \wedge ((\text{init } w) \ \Pi \ h)))$

shows $(\sigma \models (\text{init } w) \ \Pi \ (g \neg h))$

proof –

have 1 : $(\sigma \models ((\text{init } w) \ \Pi \ g) \neg ((\text{init } w) \wedge ((\text{init } w) \ \Pi \ h)))$

using assms **by** auto

have 2 : $(\exists n. (\text{enat } n) \leq \text{nlength } \sigma \wedge$
 $((\text{ntaken } n \ \sigma) \models ((\text{init } w) \ \Pi \ g)) \wedge$
 $((\text{ndropn } n \ \sigma) \models ((\text{init } w) \wedge ((\text{init } w) \ \Pi \ h))))$

using assms schop-defs **by** blast

obtain n **where** 3 : $(\text{enat } n) \leq \text{nlength } \sigma \wedge ((\text{ntaken } n \ \sigma) \models ((\text{init } w) \ \Pi \ g)) \wedge$
 $((\text{ndropn } n \ \sigma) \models ((\text{init } w) \wedge ((\text{init } w) \ \Pi \ h)))$

using 2 **by** auto

have 4 : $((\exists i. (\text{enat } i) \leq \text{nlength } (\text{ntaken } n \ \sigma) \wedge (LIFT(\text{init } w)) \ (\text{ndropn } i \ (\text{ntaken } n \ \sigma))) \wedge$
 $((\text{pifilt } (\text{ntaken } n \ \sigma) \ (LIFT(\text{init } w))) \models g)$
 $)$

by $(\text{meson } 3 \ \text{pi-d-def})$

have 5 : $(\exists i. (\text{enat } i) \leq \text{nlength } (\text{ntaken } n \ \sigma) \wedge (LIFT(\text{init } w)) \ (\text{ndropn } i \ (\text{ntaken } n \ \sigma)))$

```

using 4 by auto
have 6:  $g \text{ (pifilt (ntaken } n \text{ } \sigma) \text{ (LIFT(init } w)))}$ 
using 4 by auto
have 7:  $g \text{ (nfilter } (\lambda y. w \text{ ((NNil } y))) \text{ (ntaken } n \text{ } \sigma))}$ 
using 5 6 pifilt-init by metis
have 8:  $((\exists i. \text{ (enat } i) \leq \text{nlength (ndropn } n \text{ } \sigma) \wedge \text{ (LIFT(init } w)) (ndropn } i \text{ (ndropn } n \text{ } \sigma))) \wedge$ 
 $((\text{pifilt (ndropn } n \text{ } \sigma) \text{ (LIFT(init } w))) \models h)$ 
)
by (metis 3 intensional-rews(3) pi-d-def)
have 9:  $(\exists i. \text{ (enat } i) \leq \text{nlength (ndropn } n \text{ } \sigma) \wedge \text{ (LIFT(init } w)) (ndropn } i \text{ (ndropn } n \text{ } \sigma)))$ 
using 8 by auto
have 10:  $h \text{ (pifilt (ndropn } n \text{ } \sigma) \text{ (LIFT(init } w)))}$ 
using 8 by auto
have 11:  $h \text{ (nfilter } (\lambda y. w \text{ ((NNil } y))) \text{ (ndropn } n \text{ } \sigma))}$ 
using 10 9 pifilt-init by metis
have 12:  $(\lambda y. w \text{ ((NNil } y))) \text{ (nlast (ntaken } n \text{ } \sigma))}$ 
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:  $\exists x \in \text{nset (ntaken } n \text{ } \sigma) . (\lambda y. w \text{ ((NNil } y))) x$ 
using 12 using nfinite-ntaken nset-nlast by blast
have 15:  $\exists x \in \text{nset (ndropn } n \text{ } \sigma) . (\lambda y. w \text{ ((NNil } y))) x$ 
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:  $(\text{nfilter } (\lambda y. w \text{ ((NNil } y))) \text{ (ntaken } n \text{ } \sigma)) =$ 
 $\text{ntaken (the-enat (nlength (nfilter } (\lambda y. w \text{ ((NNil } y))) \text{ (ntaken } n \text{ } \sigma))))}$ 
 $(\text{nfilter } (\lambda y. w \text{ ((NNil } y))) \sigma)$ 
using 12 13 14 15 3 nfilter-chop1-ntaken[of  $n \text{ } \sigma$   $(\lambda y. w \text{ (NNil } y))$ ] by auto
have 17:  $(\text{nfilter } (\lambda y. w \text{ ((NNil } y))) \text{ (ndropn } n \text{ } \sigma)) =$ 
 $\text{ndropn (the-enat (nlength (nfilter } (\lambda y. w \text{ ((NNil } y))) \text{ (ntaken } n \text{ } \sigma))))}$ 
 $(\text{nfilter } (\lambda y. w \text{ ((NNil } y))) \sigma)$ 
using 12 13 14 15 3 nfilter-chop1-ndropn[of  $n \text{ } \sigma$   $(\lambda y. w \text{ (NNil } y))$ ] by auto
have 18:  $\exists n. \text{ (enat } n) \leq \text{nlength (nfilter } (\lambda y. w \text{ (NNil } y)) \sigma) \wedge$ 
 $g \text{ (ntaken } n \text{ (nfilter } (\lambda y. w \text{ (NNil } y)) \sigma)) \wedge$ 
 $h \text{ (ndropn } n \text{ (nfilter } (\lambda y. w \text{ (NNil } y)) \sigma))$ 
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:  $((\text{pifilt } \sigma \text{ (LIFT(init } w))) \models g \smallfrown h)$ 
proof -
obtain  $nn :: \text{nat}$  where
 $\text{enat } nn \leq \text{nlength } \sigma \wedge \text{ (LIFT(init } w \text{ } \Pi g)) \text{ (ntaken } nn \text{ } \sigma) \wedge \text{ (ndropn } nn \text{ } \sigma \models \text{init } w \wedge \text{init } w \text{ } \Pi h)$ 
using  $\langle \bigwedge \text{thesis.}$ 
 $(\bigwedge n. \text{ enat } n \leq \text{nlength } \sigma \wedge \text{ (LIFT(init } w \text{ } \Pi g)) \text{ (ntaken } n \text{ } \sigma) \wedge$ 
 $(\text{ndropn } n \text{ } \sigma \models \text{init } w \wedge \text{init } w \text{ } \Pi h) \implies \text{thesis}) \implies \text{thesis}$  by moura
then show ?thesis
using 19 pifilt-init by fastforce
qed
have 21:  $(\exists i. \text{ (enat } i) \leq \text{nlength } \sigma \wedge \text{ (LIFT(init } w)) \text{ (ndropn } i \text{ } \sigma))$ 
using 3 by auto

```

show *?thesis* **by** (*simp add: 20 21 pi-d-def*)
qed

lemma *PiSchopDistsem*:

$\sigma \models (init\ w) \sqcap (g \multimap h) = ((init\ w) \sqcap g) \multimap ((init\ w) \wedge ((init\ w) \sqcap h))$
using *PiSchopDistsema PiSchopDistsemb unl-lift2* **by** *blast*

lemma *PiSchopDist*:

$\vdash (init\ w) \sqcap (g \multimap h) = ((init\ w) \sqcap g) \multimap ((init\ w) \wedge ((init\ w) \sqcap h))$
using *PiSchopDistsem Valid-def* **by** *blast*

10.3.12 PiProp

lemma *Pistate*:

$(\sigma \models (init\ w) \sqcap f) =$
 $((\exists x \in nset\ \sigma. w\ (NNil\ x)) \wedge ((nfilter\ (\lambda y. w\ (NNil\ y))\ \sigma) \models f))$

proof –

have 1: $(\sigma \models (init\ w) \sqcap f) =$
 $((\exists i \leq nlength\ \sigma. w\ (NNil\ (nnth\ \sigma\ i))) \wedge ((pifilt\ \sigma\ (LIFT\ (init\ w))) \models f))$
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*:

$(\sigma \models f \sqcap \$p) =$
 $((\exists i. (enat\ i) \leq nlength\ \sigma \wedge f\ (ndropn\ i\ \sigma)) \wedge p\ (nnth\ \sigma\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ 0)))$
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*:

$(\sigma \models (\neg f) \sqcup (f \wedge \$p)) =$
 $(\exists k \leq nlength\ \sigma. f\ (ndropn\ k\ \sigma) \wedge p\ (nnth\ \sigma\ k) \wedge (\forall j < k. \neg f\ (ndropn\ j\ \sigma)))$
by (*simp add: until-d-def current-val-d-def ndropn-nfirst*)

lemma *PiPropsem3a*:

assumes $(\sigma \models f \sqcap \$p)$
shows $(\sigma \models (\neg f) \sqcup (f \wedge \$p))$
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:  $\exists x \in \text{nset}(\text{ndropns } \sigma). f x$ 
using 1 by (simp add: sxfilt-filter-help)
have 3:  $\forall x \in \text{nset}(\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)). f \ (\text{nnth } (\text{ndropns } \sigma) \ x)$ 
using 2 nkfilter-holds by fastforce
have 31:  $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0) \leq \text{nlength } \sigma$ 
by (metis 1 2 dual-order.strict-trans2 enat-ord-simps(2) leI ndropns-nnth nkfilter-not-before)
have 4:  $f \ (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0) \ \sigma)$ 
by (metis 3 31 in-nset-ndropns in-nset-ndropns-nhd ndropn-0 ndropns-nnth zero-enat-def zero-le)
have 5:  $(\forall j < (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0). \neg f \ (\text{ndropn } j \ \sigma))$ 
using nkfilter-not-before[of ndropns  $\sigma$  f ] 2
by (simp add: 31 less-imp-le ndropns-nnth order-less-le-subst2)
have 6:  $(\exists k \leq \text{nlength } \sigma. f \ (\text{ndropn } k \ \sigma) \wedge p \ (\text{nnth } \sigma \ k) \wedge (\forall j < k. \neg f \ (\text{ndropn } j \ \sigma)))$ 
using 1 31 4 5 by blast
show ?thesis using 6 PiPropsem2a by metis
qed

```

lemma *PiPropsem3b*:

assumes $(\sigma \models (\neg f) \ \mathcal{U} \ (f \wedge \$p))$

shows $(\sigma \models f \ \Pi \ \$p)$

proof –

have 1: $(\exists k \leq \text{nlength } \sigma. f \ (\text{ndropn } k \ \sigma) \wedge p \ (\text{nnth } \sigma \ k) \wedge (\forall j < k. \neg f \ (\text{ndropn } j \ \sigma)))$

using *assms PiPropsem2a* **by** *auto*

obtain *k* **where** 2: $k \leq \text{nlength } \sigma \wedge f \ (\text{ndropn } k \ \sigma) \wedge p \ (\text{nnth } \sigma \ k) \wedge (\forall j < k. \neg f \ (\text{ndropn } j \ \sigma))$

using 1 **by** *auto*

have 3: $(\exists i. (\text{enat } i) \leq \text{nlength } \sigma \wedge f \ (\text{ndropn } i \ \sigma))$

using 2 **by** *blast*

have 31: $\exists x \in \text{nset}(\text{ndropns } \sigma). f x$

using 2 *in-nset-ndropns* **by** *auto*

have 331: $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0) \leq \text{nlength } \sigma$

by (*metis 31 gen-nlength-def ndropns-nlength nkfilter-upperbound nlength-code zero-enat-def zero-le*)

have 32: $f \ (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0) \ \sigma)$

using *nkfilter-holds[of ndropns σ f 0] nkfilter-not-before[of ndropns σ f]*

by (*metis 2 31 ndropns-nfilter-nnth sxfilt-def sxfilt-pifilt-nnth-ndropn zero-enat-def zero-le*)

have 4: $p \ (\text{nnth } \sigma \ (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0))$

by (*metis 2 31 32 linorder-neqE-nat ndropns-nnth nkfilter-not-before*)

show ?thesis **using** 4 3 **by** (*simp add: PiPropsem1a*)

qed

lemma *PiPropsema*:

$\sigma \models f \ \Pi \ \$p = (\neg f) \ \mathcal{U} \ (f \wedge \$p)$

using *PiPropsem3a PiPropsem3b unl-lift2* **by** *blast*

lemma *PiProp*:

$\vdash f \ \Pi \ \$p = (\neg f) \ \mathcal{U} \ (f \wedge \$p)$

using *PiPropsema Valid-def* **by** *blast*

10.3.13 PiNext

lemma *PiNextsem1*:

$(\sigma \models f \Pi (\odot g)) =$
 $((\exists i \leq \text{nlength } \sigma. f (\text{ndropn } i \sigma)) \wedge$
 $0 < \text{nlength } (\text{nfilter } f (\text{ndropns } \sigma)) \wedge$
 $g (\text{ndropn } (\text{Suc } 0) (\text{nmap } (\lambda s. \text{nnth } s 0) (\text{nfilter } f (\text{ndropns } \sigma))))))$
using *zero-enat-def* **by** (*simp add: pi-d-def itl-defs pifilt-def sxfilt-def*)

lemma *PiNextsem2*:

$(\sigma \models (\neg f) \mathcal{U} (f \wedge \odot(f \Pi g))) =$
 $(\exists k \leq \text{nlength } \sigma.$
 $f (\text{ndropn } k \sigma) \wedge$
 $k < \text{nlength } \sigma \wedge$
 $(\exists i \leq \text{nlength } \sigma - \text{Suc } k. f (\text{ndropn } (\text{Suc } (i + k)) \sigma)) \wedge$
 $g (\text{nmap } (\lambda s. \text{nnth } s 0) (\text{nfilter } f (\text{ndropns } (\text{ndropn } (\text{Suc } k) \sigma)))) \wedge (\forall j < k. \neg f (\text{ndropn } j \sigma)))$
by (*simp add: until-d-def itl-defs pi-d-def pifilt-def sxfilt-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 $(\sigma \models f \Pi (\odot g))$
shows $(\sigma \models (\neg f) \mathcal{U} (f \wedge \odot(f \Pi g)))$
proof –
have 1: $((\exists i \leq \text{nlength } \sigma. f (\text{ndropn } i \sigma)) \wedge 0 < \text{nlength } (\text{nfilter } f (\text{ndropns } \sigma)) \wedge$
 $g (\text{ndropn } (\text{Suc } 0) (\text{nmap } (\lambda s. \text{nnth } s 0) (\text{nfilter } f (\text{ndropns } \sigma))))$
using *assms* **by** (*simp add: PiNextsem1*)
have 2: $\exists x \in \text{nset}(\text{ndropns } \sigma). f x$
using 1 **by** (*simp add: sxfilter-help*)
have 3: $\forall x \in \text{nset}(\text{nkfilter } f 0 (\text{ndropns } \sigma)). f (\text{nnth } (\text{ndropns } \sigma) x)$
using 2 *nkfilter-holds* **by** *fastforce*
have 31: $(\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 0) \leq \text{nlength } \sigma$
by (*metis 2 gen-nlength-def i0-lb ndropns-nlength nkfilter-upperbound nlength-code zero-enat-def*)
have 4: $f (\text{ndropn } (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 0) \sigma)$
by (*metis 3 31 i0-lb in-nset-conv-nnth ndropns-nnth zero-enat-def*)
have 41: $(\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 1) \in \text{nset}(\text{nkfilter } f 0 (\text{ndropns } \sigma))$
by (*metis 1 2 One-nat-def eSuc-enat ileI1 in-nset-conv-nnth nkfilter-nlength zero-enat-def*)
have 42: $(\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 1) \leq \text{nlength } (\text{ndropns } \sigma)$
by (*metis 2 41 gen-nlength-def in-nset-conv-nnth nkfilter-upperbound nlength-code*)
have 5: $f (\text{ndropn } (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 1) \sigma)$
using 3 41 42 **by** (*metis ndropns-nlength ndropns-nnth*)
have 6: $(\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 0) < (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 1)$
by (*metis 1 2 One-nat-def ileI1 nidx-nkfilter-expand nkfilter-nlength one-eSuc one-enat-def*)
have 7: $\text{Suc } (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 0) \leq \text{nlength } \sigma$
by (*metis 42 6 Suc-leI dual-order.trans enat-ord-simps(1) ndropns-nlength*)
have 8: $0 < \text{nlength } (\text{nkfilter } f 0 (\text{ndropns } \sigma))$
using 1 2 *nkfilter-nlength* **by** *fastforce*
have 9: $(\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 1) \leq \text{nlength } \sigma$
by (*metis 42 ndropns-nlength*)
have 10: $(\text{Suc } (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 0)) \leq (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 1)$
using 6 *Suc-leI* **by** *blast*
have 11: $(\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 1) =$
 $(\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 1) - (\text{Suc } (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 0)) +$
 $(\text{Suc } (\text{nnth } (\text{nkfilter } f 0 (\text{ndropns } \sigma)) 0))$

```

using 10 by auto
have 12:  $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 1) - (\text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0)) \leq$ 
 $\text{nlength } \sigma - \text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0)$ 
using 9 diff-le-mono by (metis enat-minus-mono1 idiff-enat-enat)
have 13:  $\exists \ i \leq \text{nlength } \sigma - \text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0).$ 
 $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 1) = (\text{Suc } (i + (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0)))$ 
using 11 12 by auto
have 14:  $(\exists i \leq \text{nlength } \sigma - \text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0).$ 
 $f \ (\text{ndropn } (\text{Suc } (i + (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0))) \ \sigma))$ 
using 13 5 by auto
have 15:  $(\text{ndropn } (\text{Suc } 0) \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfilter } f \ (\text{ndropns } \sigma)))) =$ 
 $(\text{nmap } (\lambda s. \text{nnth } s \ 0)$ 
 $(\text{nfilter } f \ (\text{ndropns } (\text{ndropn } (\text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0)) \ \sigma))))$ 
using 2 8
by (metis One-nat-def ileI1 nfilter-ndropns-nmap nkfilter-nlength one-eSuc one-enat-def)
have 16:  $g \ (\text{nmap } (\lambda s. \text{nnth } s \ 0)$ 
 $(\text{nfilter } f \ (\text{ndropns } (\text{ndropn } (\text{Suc } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0)) \ \sigma))))$ 
using 1 15 by auto
have 17:  $\forall \ j < (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0). \neg f \ (\text{ndropn } j \ \sigma)$ 
using 2 31 nkfilter-not-before[of ( $\text{ndropns } \sigma$ )  $f$ ]
by (metis enat-ord-simps(1) less-imp-le ndropns-nnth order.trans)
have 18:  $(\exists k \leq \text{nlength } \sigma.$ 
 $f \ (\text{ndropn } k \ \sigma) \wedge$ 
 $k < \text{nlength } \sigma \wedge$ 
 $(\exists i \leq \text{nlength } \sigma - \text{Suc } k. f \ (\text{ndropn } (\text{Suc } (i + k)) \ \sigma)) \wedge$ 
 $g \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfilter } f \ (\text{ndropns } (\text{ndropn } (\text{Suc } k) \ \sigma)))) \wedge$ 
 $(\forall j < k. \neg f \ (\text{ndropn } j \ \sigma)))$ 
using 14 16 17 4 7 Suc-ile-eq less-imp-le by blast
show ?thesis using 18 by (simp add: PiNextsem2)
qed

```

lemma *PiNextsem4*:

assumes $(\sigma \models (\neg f) \ \mathcal{U} \ (f \wedge \bigcirc(f \ \Pi \ g)))$

shows $(\sigma \models f \ \Pi \ (\bigcirc \ g))$

proof –

```

have 1:  $(\exists k \leq \text{nlength } \sigma.$ 
 $f \ (\text{ndropn } k \ \sigma) \wedge$ 
 $k < \text{nlength } \sigma \wedge$ 
 $(\exists i \leq \text{nlength } \sigma - \text{Suc } k. f \ (\text{ndropn } (\text{Suc } (i + k)) \ \sigma)) \wedge$ 
 $g \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfilter } f \ (\text{ndropns } (\text{ndropn } (\text{Suc } k) \ \sigma)))) \wedge$ 
 $(\forall j < k. \neg f \ (\text{ndropn } j \ \sigma)))$ 
using assms by (simp add: PiNextsem2)
obtain  $k$  where 2:  $k \leq \text{nlength } \sigma \wedge f \ (\text{ndropn } k \ \sigma) \wedge k < \text{nlength } \sigma \wedge (\exists i \leq \text{nlength } \sigma - \text{Suc } k.$ 
 $f \ (\text{ndropn } (\text{Suc } (i + k)) \ \sigma)) \wedge$ 
 $g \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) \ (\text{nfilter } f \ (\text{ndropns } (\text{ndropn } (\text{Suc } k) \ \sigma)))) \wedge (\forall j < k. \neg f \ (\text{ndropn } j \ \sigma))$ 
using 1 by auto
have 3:  $\exists \ x \in \text{nset } (\text{ndropns } \sigma). f \ x$ 
using 1 using in-nset-ndropns by auto
have 4:  $\forall \ x \in \text{nset } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)). f \ (\text{nnth } (\text{ndropns } \sigma) \ x)$ 
using 3 nkfilter-holds by fastforce

```


have 41: $(\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0) \leq \text{nlength } \sigma$
by (metis 3 gen-nlength-def le-cases le-zero-eq ndropns-nlength nkfilter-upperbound nlength-code zero-enat-def)
have 5: $f \ (\text{ndropn } (\text{nnth } (\text{nkfilter } f \ 0 \ (\text{ndropns } \sigma)) \ 0) \ \sigma)$
by (metis 4 41 i0-lb in-nset-conv-nnth ndropns-nnth zero-enat-def)
have 6: $0 < \text{nlength } (\text{nfilter } f \ (\text{ndropns } \sigma))$
using 2 3 nfilter-nlength-zero-conv-a[of $(\text{ndropns } \sigma) \ f$]
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 $(\text{ndropns } \sigma) \ f$] nfilter-ndropns-nmap[of $\sigma \ f \ 0$]
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

10.3.14 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 sfxfilt-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 sxfilt-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.prem1 less-SucE* **by** *blast*
have 2: $(\forall i < k. \text{ff}(i) < \text{ff}(\text{Suc } i))$
using *Suc.prem2* **by** *auto*
have 3: $\text{ff } k < \text{ff } (\text{Suc } k)$
by (*simp add: Suc.prem2*)
have 4: $(\text{ff}(0) < j \wedge j \leq \text{ff}(k)) \vee (\text{ff } (k) < j \wedge j \leq \text{ff}(\text{Suc } k))$
using 1 2 *Suc.hyps* **by** *blast*
have 41: $\text{ff}(0) < j$
using 2 4 *Suc.hyps order-refl* **by** *force*
have 42: $j \leq \text{ff}(\text{Suc } k)$
using 3 4 **by** *linarith*
have 5: $\text{ff}(0) < j \wedge j \leq \text{ff}(\text{Suc } k)$
by (*simp add: 41 42*)
show ?*thesis*
by (*simp add: 5*)
qed
qed

lemma *cover-a*:

assumes $(\forall j.$
 $(j \leq \text{ff } 0) \vee (\exists i < k. \text{ff}(i) < (j::\text{nat}) \wedge j \leq \text{ff}(\text{Suc } i))$
 $\longrightarrow \text{gg } j)$
 $(\forall i < k. \text{ff}(i) < \text{ff}(\text{Suc } i))$
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: $(\forall j. \text{ff } 0 < j \wedge j \leq \text{ff } k \longrightarrow \text{gg } j)$
proof

```

fix j
show  $\text{ff } 0 < j \wedge j \leq \text{ff } k \longrightarrow \text{gg } j$ 
using assms
proof (induct k arbitrary: j)
case 0
then show ?case by simp
next
case (Suc k)
then show ?case
proof -
have 21:  $(\forall j. (j \leq \text{ff } 0) \vee (\exists i < k. \text{ff}(i) < (j::\text{nat}) \wedge j \leq \text{ff}(Suc\ i))$ 
 $\vee (\text{ff } k < j \wedge j \leq \text{ff}(Suc\ k)) \longrightarrow \text{gg } j)$ 
using Suc.prem1 less-SucI by blast
have 22:  $(\forall i < k. \text{ff}(i) < \text{ff}(Suc\ i))$ 
using Suc.prem2 by auto
have 23:  $\text{ff } k < \text{ff}(Suc\ k)$ 
by (simp add: Suc.prem2)
have 24:  $(\forall j. (j \leq \text{ff } 0) \vee (\text{ff } 0 < j \wedge j \leq \text{ff } k)$ 
 $\vee (\text{ff } k < j \wedge j \leq \text{ff}(Suc\ k)) \longrightarrow \text{gg } j)$ 
using 21 22 Suc.hyps by blast
have 25:  $\text{ff } 0 < j \wedge j \leq \text{ff}(Suc\ k) \longrightarrow \text{gg } j$ 
using 24 not-less by blast
show ?thesis using 25 by blast
qed
qed
qed
show ?thesis
by (metis 2 assms1 less-or-eq-imp-le linorder-neqE-nat)
qed

```

lemma *PiUntilDistsem3*:

```

assumes  $(\sigma \models f \Pi (g \mathcal{U} h))$ 
shows  $(\sigma \models (f \Pi g) \mathcal{U} (f \Pi h))$ 
proof -
have 1:  $(\exists i \leq \text{nlength } \sigma. f(\text{ndropn } i \ \sigma)) \wedge$ 
 $(\exists k \leq \text{nlength } (\text{nfilter } f \ (\text{ndropns } \sigma)).$ 
 $h(\text{ndropn } k \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f \ (\text{ndropns } \sigma)))) \wedge$ 
 $(\forall j < k. g(\text{ndropn } j \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f \ (\text{ndropns } \sigma))))))$ 
using assms PiUntilDistsem1 by blast
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 4:  $(\exists k \leq \text{nlength } (\text{nfilter } f \ (\text{ndropns } \sigma)).$ 
 $h(\text{ndropn } k \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f \ (\text{ndropns } \sigma)))) \wedge$ 
 $(\forall j < k. g(\text{ndropn } j \ (\text{nmap } (\lambda s. \text{nnth } s \ 0) (\text{nfilter } f \ (\text{ndropns } \sigma))))))$ 
using 1 by auto
obtain k where 5:  $k \leq \text{nlength } (\text{nfilter } f \ (\text{ndropns } \sigma)) \wedge$ 

```

$$h \text{ (ndropn } k \text{ (nmap } (\lambda s. \text{nnth } s \text{ } 0) \text{ (nfilter } f \text{ (ndropns } \sigma)))) \wedge$$

$$(\forall j < k. g \text{ (ndropn } j \text{ (nmap } (\lambda s. \text{nnth } s \text{ } 0) \text{ (nfilter } f \text{ (ndropns } \sigma))))))$$

using 4 **by** *auto*
have 51: $(\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } k) \leq \text{nlength } \sigma$
by (*metis* (*mono-tags, lifting*) 2 5 *gen-nlength-def ndropns-nlength nkfilter-nlength nkfilter-upperbound nlength-code*)
have 6: $f \text{ (ndropn } (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } k) \text{ } \sigma)$
using 2 3 5
by (*metis* 1 *ndropns-nfilter-nnth nlength-nmap pifilt-def sfxfilt-def sfxfilt-pifilt-nnth-ndropn*)
have 7: $k=0 \longrightarrow$

$$(\exists i. (\text{enat } i) \leq \text{nlength } \sigma - k \wedge f \text{ (ndropn } (i + k) \text{ } \sigma)) \wedge$$

$$h \text{ (nmap } (\lambda s. \text{nnth } s \text{ } 0) \text{ (nfilter } f \text{ (ndropns (ndropn } k \text{ } \sigma)))) \wedge$$

$$(\forall j < k. (\exists i \leq \text{nlength } \sigma - j. f \text{ (ndropn } (i + j) \text{ } \sigma)) \wedge$$

$$g \text{ (nmap } (\lambda s. \text{nnth } s \text{ } 0) \text{ (nfilter } f \text{ (ndropns (ndropn } j \text{ } \sigma))))))$$
using 1 5 **by** *auto*
have 71: $k > 0 \longrightarrow (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } (k - 1)) \in \text{nset}(\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma))$
by (*metis* 2 5 *One-nat-def Suc-ile-eq Suc-pred in-nset-conv-nnth less-imp-le nkfilter-nlength*)
have 8: $k > 0 \longrightarrow f \text{ (ndropn } (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } (k - 1)) \text{ } \sigma)$
using 3 71
by (*metis* *enat-ile le-cases ndropn-all ndropns-nlength ndropns-nnth nnth-beyond*)
have 9: $k > 0 \longrightarrow (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } (k - 1)) < (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } k)$
by (*metis* 2 5 *One-nat-def Suc-pred nkfilter-mono nkfilter-nlength*)
have 10: $k > 0 \longrightarrow (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } (k - 1)) \leq \text{nlength } \sigma$
by (*metis* 2 71 *gen-nlength-def in-nset-conv-nnth ndropns-nlength nkfilter-upperbound nlength-code*)
have 11: $k > 0 \longrightarrow (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } k) \leq \text{nlength } \sigma$
using *nkfilter-upperbound[of ndropns σ f k 0]*
by (*metis* 2 9 *One-nat-def Suc-pred eSuc-enat gen-nlength-def ileI1 leI le-cases less-not-refl3 ndropns-nlength nlength-code nnth-beyond*)
have 12: $k > 0 \longrightarrow$

$$h \text{ (nmap } (\lambda s. \text{nnth } s \text{ } 0)$$

$$(\text{nfilter } f \text{ (ndropns (ndropn } (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } k) \text{ } \sigma))))$$
using *nfilter-nkfilter-ndropn[of f (ndropns σ) k]*

$$\text{ndropns-nfilter-ndropn-a[of } f \text{ } k \text{ } \sigma]$$
by (*metis* 2 5 *ndropn-nmap*)
have 121: $k > 0 \longrightarrow$

$$h \text{ (nmap } (\lambda s. \text{nnth } s \text{ } 0)$$

$$(\text{nfilter } f \text{ (ndropns (ndropn } (\text{Suc } (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } (k - 1))) \text{ } \sigma))))$$
by (*metis* 2 5 *One-nat-def Suc-pred nfilter-ndropns-nmap*)
have 13: $k > 0 \longrightarrow (\exists i \leq \text{nlength } \sigma - (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } k).$

$$f \text{ (ndropn } (i + (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } k)) \text{ } \sigma))$$
using 6 **by** (*metis* *add.left-neutral zero-enat-def zero-le*)
have 130: $k > 0 \longrightarrow (\exists i. i + (\text{Suc } (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } (k - 1))) \leq \text{nlength } \sigma \wedge$

$$(\text{ndropn } (i + (\text{Suc } (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } (k - 1)))) \text{ } \sigma) =$$

$$(\text{ndropn } (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } k) \text{ } \sigma))$$
using 9 **by** (*metis* 11 *Suc-leI diff-add*)
have 131: $k > 0 \longrightarrow (\exists i. i + (\text{Suc } (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } (k - 1))) \leq \text{nlength } \sigma \wedge$

$$f \text{ (ndropn } (i + (\text{Suc } (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } (k - 1)))) \text{ } \sigma))$$
using 10 6 9 11 130 **by** *auto*
have 132: $k > 0 \longrightarrow (\exists i \leq \text{nlength } \sigma - (\text{Suc } (\text{nnth } (\text{nkfilter } f \text{ } 0 \text{ (ndropns } \sigma)) \text{ } (k - 1))).$

$f \text{ (ndropn (i + (Suc (nnth (nkfilter f 0 (ndropns \sigma)) (k-1)))) \sigma))$
using 131
by (metis add-diff-cancel-right' enat-minus-mono1 idiff-enat-enat)
have 14: $k > 0 \longrightarrow (\forall j < (\text{nnth (nkfilter f 0 (ndropns \sigma)) } k). \\ (\exists i. i+j \leq \text{nlength } \sigma \wedge f \text{ (ndropn (i + j) } \sigma)))$
using 131 **by** (metis 11 6 diff-add less-imp-le plus-enat-simps(1))
have 141: $k > 0 \longrightarrow (\forall j < (\text{Suc (nnth (nkfilter f 0 (ndropns \sigma)) (k-1)))) \\ (\exists i. i+j \leq \text{nlength } \sigma \wedge f \text{ (ndropn (i + j) } \sigma)))$
using 14 9 **by** auto
have 142: $k > 0 \longrightarrow (\forall j < (\text{Suc (nnth (nkfilter f 0 (ndropns \sigma)) (k-1)))) \\ (\exists i \leq \text{nlength } \sigma - j. f \text{ (ndropn (i + j) } \sigma)))$
using 141 **by** (metis add.commute enat.simps(3) enat-add-sub-same enat-minus-mono1)
have 15: $(\forall j < k. g \text{ (ndropn j (nmap (\lambda s. \text{nnth s 0) (nfilter f (ndropns \sigma)))))})$
using 5 **by** blast
have 151: $(\forall j < k. g \text{ (nmap (\lambda s. \text{nnth s 0) (ndropn j (nfilter f (ndropns \sigma)))))})$
by (metis 15 ndropn-nmap)
have 152: $(\forall j < k. (\text{ndropn j (nfilter f (ndropns \sigma))}) = \\ (\text{nfilter f (ndropn (nnth (nkfilter f 0 (ndropns \sigma)) j) (ndropns \sigma))}))$
using nfilter-nkfilter-ndropn-1[of ndropns \sigma f]
by (simp add: 2 5 less-imp-le nkfilter-nlength order-less-le-subst2)
have 16: $(\forall j < k. g \text{ (nmap (\lambda s. \text{nnth s 0}) \\ (\text{nfilter f (ndropn (nnth (nkfilter f 0 (ndropns \sigma)) j) (ndropns \sigma)))))})$
using 151 152 nfilter-nkfilter-ndropn **by** simp
have 1610: $(\text{nnth (nkfilter f 0 (ndropns \sigma)) } k) \leq \text{nlength(ndropns \sigma)}$
by (simp add: 51 ndropns-nlength)
have 1611: $(\forall j < k. (\text{nnth (nkfilter f 0 (ndropns \sigma)) } j) < (\text{nnth (nkfilter f 0 (ndropns \sigma)) } k))$
by (simp add: 2 5 nidx-nkfilter-gr nkfilter-nlength)
have 1612: $(\forall j < k. (\text{nnth (nkfilter f 0 (ndropns \sigma)) } j) \leq \text{nlength(ndropns \sigma)})$
using 1610 1611 **by** (meson enat-ord-simps(2) less-imp-le order-less-le-subst2)
have 161: $(\forall j < k. ((\text{ndropn (nnth (nkfilter f 0 (ndropns \sigma)) j) (ndropns \sigma))} = \\ (\text{ndropns (ndropn (nnth (nkfilter f 0 (ndropns \sigma)) j) \sigma))))$
using 1612 **by** (simp add: ndropn-ndropns)
have 17: $(\forall j < k. g \text{ (nmap (\lambda s. \text{nnth s 0}) \\ (\text{nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns \sigma)) j) \sigma)))))})$
using 16 161 **by** auto
have 18: $k > 0 \longrightarrow \\ g \text{ (nmap (\lambda s. \text{nnth s 0}) \\ (\text{nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns \sigma)) (k-1)) \sigma))}))$
using 17 **by** simp
have 19: $k > 0 \longrightarrow \\ g \text{ (nmap (\lambda s. \text{nnth s 0}) \\ (\text{nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns \sigma)) 0) \sigma))}))$
using 17 **by** blast
have 20: $k > 0 \longrightarrow (\text{nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns \sigma)) 0) \sigma))} = \\ (\text{nfilter f (ndropn (nnth (nkfilter f 0 (ndropns \sigma)) 0) (ndropns \sigma))})$
using 161 **by** auto
have 200: $\text{nnth (nkfilter f 0 (ndropns \sigma)) } 0 \leq \text{nlength (ndropns \sigma)}$
using 1610 1612 **by** blast
have 21: $k > 0 \longrightarrow (\forall j \leq (\text{nnth (nkfilter f 0 (ndropns \sigma)) } 0). \\ ((\text{ndropns (ndropn j \sigma)})) =$

```

      ( (ndropn j (ndropns σ))))
using 200 ndropn-ndropns by (simp add: ndropn-ndropns order-subst2)
have 22: k>0 ⟶ (∀ j ≤ (nnth (nkfilter f 0 (ndropns σ)) 0).
  (nfilter f (ndropns (ndropn j σ))) =
  (nfilter f (ndropn j (ndropns σ))))
using 21 by auto
have 23: k>0 ⟶
  (∀ j ≤ (nnth (nkfilter f 0 (ndropns σ)) 0).
    (nmap (λs. nnth s 0)
      (nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) 0) σ)))) =
    (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ)))))
  )
by (simp add: 2 22 nfilter-ndropns-nmap-help-0)
have 24: k>0 ⟶
  (∀ j ≤ (nnth (nkfilter f 0 (ndropns σ)) 0).
    g (nmap (λs. nnth s 0)
      (nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) 0) σ)))) =
    g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ)))))
  )
using 23 by auto
have 241: k>0 ⟶ (∀ j ≤ (nnth (nkfilter f 0 (ndropns σ)) 0).
  g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ))))) )
using 19 24 by blast
have 25: k>0 ⟶ (∀ i < k - 1.
  (∀ l. l ≤ (nnth (nkfilter f 0 (ndropns σ)) (Suc i)) ∧
    (nnth (nkfilter f 0 (ndropns σ)) i) < l ⟶
    (nfilter f (ndropn (nnth (nkfilter f 0 (ndropns σ)) (Suc i)) (ndropns σ)) =
    (nfilter f (ndropn l (ndropns σ)) ) ) )
proof auto
  fix i
  fix l
  assume a0: 0 < k
  assume a1: i < k - Suc 0
  assume a2: l ≤ nnth (nkfilter f 0 (ndropns σ)) (Suc i)
  assume a3: nnth (nkfilter f 0 (ndropns σ)) i < l
  show nfilter f (ndropn (nnth (nkfilter f 0 (ndropns σ)) (Suc i)) (ndropns σ)) =
    nfilter f (ndropn l (ndropns σ))
proof -
  have 251: k=1 ⟹
    nfilter f (ndropn (nnth (nkfilter f 0 (ndropns σ)) (Suc i)) (ndropns σ)) =
    nfilter f (ndropn l (ndropns σ))
  using a1 by force
  have 2511: 1 < k ⟹ enat (Suc i) ≤ nlength (nfilter f (ndropns σ))
  by (metis 2 Suc-ile-eq a2 a3 leD le-cases nkfilter-nlength nnth-beyond not-le-imp-less)
  then have 252: 1 < k ⟹
    nfilter f (ndropn (nnth (nkfilter f 0 (ndropns σ)) (Suc i)) (ndropns σ)) =
    nfilter f (ndropn l (ndropns σ))
  using 2 5 a1 nfilter-ndropns-nmap-help-j[of ndropns σ f l i] a2 a3 by fastforce
  show ?thesis
using 252 a1 by fastforce

```

qed
qed
have 261: $k > 0 \longrightarrow (\forall i < k - 1.$
 $(\forall l. l \leq (nnth (nkfilter f 0 (ndropns \sigma)) (Suc i)) \wedge$
 $(nnth (nkfilter f 0 (ndropns \sigma)) i) < l \longrightarrow$
 $l \leq nlength (ndropns \sigma))$
using 1612 by (*meson diff-less enat-ord-simps(1) less-one less-trans-Suc order-subst2*)
have 262: $k > 0 \longrightarrow (\forall i < k - 1.$
 $(\forall l. l \leq (nnth (nkfilter f 0 (ndropns \sigma)) (Suc i)) \wedge$
 $(nnth (nkfilter f 0 (ndropns \sigma)) i) < l \longrightarrow$
 $(ndropn l (ndropns \sigma)) = (ndropns (ndropn l \sigma)))$
using 261 by (*simp add: ndropn-ndropns*)
have 26: $k > 0 \longrightarrow (\forall i < k - 1.$
 $(\forall l. l \leq (nnth (nkfilter f 0 (ndropns \sigma)) (Suc i)) \wedge$
 $(nnth (nkfilter f 0 (ndropns \sigma)) i) < l \longrightarrow$
 $(nmap (\lambda s. nnth s 0)$
 $(nfilter f (ndropn (nnth (nkfilter f 0 (ndropns \sigma)) (Suc i)) (ndropns \sigma)))) =$
 $(nmap (\lambda s. nnth s 0) (nfilter f (ndropns (ndropn l \sigma)))$
 $))$
using 25 262 by auto
have 27: $k > 0 \longrightarrow (\forall i < k - 1.$
 $(\forall l. l \leq (nnth (nkfilter f 0 (ndropns \sigma)) (Suc i)) \wedge$
 $(nnth (nkfilter f 0 (ndropns \sigma)) i) < l \longrightarrow$
 $g (nmap (\lambda s. nnth s 0)$
 $(nfilter f$
 $(ndropn (nnth (nkfilter f 0 (ndropns \sigma)) (Suc i)) (ndropns \sigma))))$
by (*simp add: 16*)
have 28: $k > 0 \longrightarrow (\forall i < k - 1.$
 $(\forall l. l \leq (nnth (nkfilter f 0 (ndropns \sigma)) (Suc i)) \wedge$
 $(nnth (nkfilter f 0 (ndropns \sigma)) i) < l \longrightarrow$
 $g (nmap (\lambda s. nnth s 0) (nfilter f (ndropns (ndropn l \sigma)))$
 $))$
using 25 262 27 by auto
have 281: $k > 0 \longrightarrow$
 $(\forall j.$
 $(j \leq (nnth (nkfilter f 0 (ndropns \sigma)) 0) \vee$
 $(\exists i. i < k - 1 \wedge$
 $j \leq (nnth (nkfilter f 0 (ndropns \sigma)) (Suc i)) \wedge$
 $(nnth (nkfilter f 0 (ndropns \sigma)) i) < j) \longrightarrow$
 $g (nmap (\lambda s. nnth s 0) (nfilter f (ndropns (ndropn j \sigma)))$
 $))$
using 241 28 by blast
have 282: $k > 0 \longrightarrow (\forall i < k - 1.$
 $nnth (nkfilter f 0 (ndropns \sigma)) i < nnth (nkfilter f 0 (ndropns \sigma)) (Suc i))$
using *nidx-nkfilter-expand[of ndropns \sigma f - 0]*
by (*metis 1611 2 One-nat-def Suc-mono Suc-pred le-cases less-SucI ntaken-all ntaken-nnth-a*)
have 29: $k > 0 \longrightarrow (\forall j < (Suc (nnth (nkfilter f 0 (ndropns \sigma)) (k - 1))).$
 $g (nmap (\lambda s. nnth s 0) (nfilter f (ndropns (ndropn j \sigma))))$
using 281 282 cover-a[of $\lambda x. nnth (nkfilter f 0 (ndropns \sigma)) x$ $k - 1$
 $(\lambda j. g (nmap (\lambda s. nnth s 0) (nfilter f (ndropns (ndropn j \sigma))))]$
using 18 less-antisym by blast
have 30: $k > 0 \longrightarrow (\exists k \leq nlength \sigma.$
 $(\exists i \leq nlength \sigma - k. f (ndropn (i + k) \sigma)) \wedge$

$h \text{ (nmap } (\lambda s. \text{nnth } s \ 0) \text{ (nfilter } f \text{ (ndropns (ndropn } k \ \sigma)))) \wedge$
 $(\forall j < k. (\exists i \leq \text{nlength } \sigma - j. f \text{ (ndropn } (i + j) \ \sigma)) \wedge$
 $g \text{ (nmap } (\lambda s. \text{nnth } s \ 0) \text{ (nfilter } f \text{ (ndropns (ndropn } j \ \sigma))))))$
using 10 11 121 132 142 29 9 **by** *simp*
 $(\text{metis add-Suc-right antisym-conv2 eSuc-enat enat-ord-simps}(1) \text{ ileI1 leD})$
have 31: $(\exists k \leq \text{nlength } \sigma.$
 $(\exists i \leq \text{nlength } \sigma - k. f \text{ (ndropn } (i + k) \ \sigma)) \wedge$
 $h \text{ (nmap } (\lambda s. \text{nnth } s \ 0) \text{ (nfilter } f \text{ (ndropns (ndropn } k \ \sigma)))) \wedge$
 $(\forall j < k. (\exists i \leq \text{nlength } \sigma - j. f \text{ (ndropn } (i + j) \ \sigma)) \wedge$
 $g \text{ (nmap } (\lambda s. \text{nnth } s \ 0) \text{ (nfilter } f \text{ (ndropns (ndropn } j \ \sigma))))))$
using 30 7 **by** $(\text{metis not-gr-zero zero-enat-def zero-le})$
show ?thesis **using** 31 **by** $(\text{simp add: PiUntilDistsem2})$
qed

lemma *PiUntilDistsem4*:

assumes $(\sigma \models (f \ \Pi \ g) \ \mathcal{U} \ (f \ \Pi \ h))$

shows $(\sigma \models f \ \Pi \ (g \ \mathcal{U} \ h))$

proof –

have 1: $(\exists k \leq \text{nlength } \sigma.$
 $(\exists i \leq \text{nlength } \sigma - k. f \text{ (ndropn } (i + k) \ \sigma)) \wedge$
 $h \text{ (nmap } (\lambda s. \text{nnth } s \ 0) \text{ (nfilter } f \text{ (ndropns (ndropn } k \ \sigma)))) \wedge$
 $(\forall j < k.$
 $(\exists i \leq \text{nlength } \sigma - j. f \text{ (ndropn } (i + j) \ \sigma)) \wedge$
 $g \text{ (nmap } (\lambda s. \text{nnth } s \ 0) \text{ (nfilter } f \text{ (ndropns (ndropn } j \ \sigma))))))$
using *assms* **by** $(\text{simp add: PiUntilDistsem2})$
obtain k **where** 2: $k \leq \text{nlength } \sigma \wedge$
 $(\exists i \leq \text{nlength } \sigma - k. f \text{ (ndropn } (i + k) \ \sigma)) \wedge$
 $h \text{ (nmap } (\lambda s. \text{nnth } s \ 0) \text{ (nfilter } f \text{ (ndropns (ndropn } k \ \sigma)))) \wedge$
 $(\forall j < k.$
 $(\exists i \leq \text{nlength } \sigma - j. f \text{ (ndropn } (i + j) \ \sigma)) \wedge$
 $g \text{ (nmap } (\lambda s. \text{nnth } s \ 0) \text{ (nfilter } f \text{ (ndropns (ndropn } j \ \sigma))))))$
using 1 **by** *auto*
have 3: $(\exists i \leq \text{nlength } \sigma - k. f \text{ (ndropn } (i + k) \ \sigma))$
using 2 **by** *auto*
have 31: $(\exists i \leq \text{nlength } (\text{ndropns (ndropn } k \ \sigma)). f \text{ (nnth (ndropns (ndropn } k \ \sigma)) } i))$
by $(\text{metis 3 add.commute ndropn-ndropn ndropn-nlength ndropns-nlength ndropns-nnth})$
have 4: $k \leq \text{nlength } \sigma$
using 2 **by** *auto*
obtain i **where** 5: $(\text{enat } i) \leq \text{nlength } (\text{ndropns (ndropn } k \ \sigma)) \wedge f \text{ (nnth (ndropns (ndropn } k \ \sigma)) } i) \wedge$
 $i = (\text{LEAST } na. \text{enat } na \leq \text{nlength } (\text{ndropns (ndropn } k \ \sigma)) \wedge$
 $f \text{ (nnth (ndropns (ndropn } k \ \sigma)) } na))$
by $(\text{metis (no-types, lifting) 31 LeastI-ex})$
have 51: $i = (\text{LEAST } na. \text{enat } na \leq \text{nlength } (\text{ndropns (ndropn } k \ \sigma)) \wedge f \text{ (nnth (ndropns (ndropn } k \ \sigma)) } na))$
using 5 **by** *auto*
have 60: $(\text{enat } i) \leq \text{nlength } \sigma - k$
by $(\text{metis 5 ndropn-nlength ndropns-nlength})$
have 601: $k + i \leq \text{nlength } \sigma$
by $(\text{metis 4 60 dual-order.eq-iff enat.simps}(3) \text{ enat-add-sub-same enat-less-enat-plusI2}$
 $\text{less-eqE less-imp-le order.not-eq-order-implies-strict plus-enat-simps}(1))$


```

have 6:  $i+k \leq \text{nlength } \sigma$ 
  by (metis 601 add.commute)
have 61:  $\exists x \in \text{nset } (\text{ndropns } (\text{ndropn } k \ \sigma)). f \ x$ 
  using 31 exists-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 $?nkf = (the-enat ((nlength(nkfilter\ f\ 0\ (ntaken\ ((nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma))))\ 0)+k)$
 $(ndropns\ \sigma))))))$
let $?pkf = (the-enat ((nlength\ (nkfilter\ f\ 0\ (ntaken\ k\ (ndropns\ \sigma))))))$
have 93: $?kf = ?nkf$
by (metis 90 nfinite-ntaken nkfilter-nlength nset-nlast)
have 94: $((nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma)))\ 0)+k) \leq nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ ?nkf$
by (metis 74 90 91 add.left-neutral eq-iff ndropn-nfirst ndropns-nlength ndropns-nnth
nkfilter-chop1-ndropn nkfilter-nfirst)
have 95: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma)))\ 0) = 0 \implies k = nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ ?nkf$
by (metis 10 90 91 add-cancel-left-left ndropn-nfirst nkfilter-chop1-ndropn)
have 98: $0 < ?nkf \wedge 0 < (nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma)))\ 0) \implies$
 $nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (?nkf - 1) < k$
proof –
assume $b: 0 < ?nkf \wedge 0 < (nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma)))\ 0)$
have 980: $(nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (?nkf - 1)) \in nset\ (nkfilter\ f\ 0\ (ndropns\ \sigma))$
by (metis in-nset-conv-nnth le-cases nfinite-ntaken nset-nlast ntaken-all ntaken-nlast)
have 981: $f\ (nnth\ (ndropns\ \sigma)\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (?nkf - 1)))$
using nkfilter-holds[of (ndropns σ) $f\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (?nkf - 1))\ 0]$
using 71 980 **by** auto
have 982: $\neg f\ (ndropn\ k\ \sigma)$
by (metis 10 4 add-cancel-left-left b gr-implies-not0 ndropn-nfirst ndropns-nnth
nkfilter-nfirst)
have 984: $f\ (ndropn\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ \sigma))\ (?nkf - 1))\ \sigma)$
using 71 980 981 ndropns-nnth[of (nnth (nkfilter f 0 (ndropns σ)) (?nkf - 1)) $\sigma]$
by (metis gen-nlength-def in-nset-conv-nnth ndropns-nlength nkfilter-upperbound nlength-code)
have 985: $\bigwedge j. k \leq j \implies j < k+i \implies \neg f\ (ndropn\ j\ \sigma)$
proof –
fix j
assume $a0: k \leq j$
assume $a1: j < k+i$
show $\neg f\ (ndropn\ j\ \sigma)$
proof –
have 9851: $\exists x \in nset\ (ndropn\ k\ (ndropns\ \sigma)).\ f\ x$
by (simp add: 4 61 ndropn-ndropns ndropns-nlength)
have 9852: $enat\ k \leq nlength\ (ndropns\ \sigma)$
by (simp add: 4 ndropns-nlength)
have 9853: $k \leq j$
by (simp add: a0)
have 9854: $j < nnth\ (nkfilter\ f\ k\ (ndropn\ k\ (ndropns\ \sigma)))\ 0$
using 10 101 92 9852 a1 ndropn-ndropns **by** fastforce
have 9855: $\neg f\ (nnth\ (ndropns\ \sigma)\ j)$
using 9851 9852 9853 9854 nkfilter-n-not-before[of $k\ ndropns\ \sigma\ f\ j]$ **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: $ntaken\ ?nkf\ (nkfilter\ f\ 0\ (ndropns\ \sigma)) =$
 $nkfilter\ f\ 0\ (ntaken\ (nnth\ (nkfilter\ f\ 0\ (ndropns\ (ndropn\ k\ \sigma)))\ 0 + k)\ (ndropns\ \sigma))$

```

    using nkfilter-chop1-ntaken[of (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0 + k) (ndropns σ) f 0]
    using 90 91 by blast
  have 987: ¬ enat (?nkf) ≤ nlength (nkfilter f 0 (ndropns σ)) ∨
    nnth (nkfilter f 0 (ndropns σ)) (?nkf - 1) < nnth (nkfilter f 0 (ndropns σ)) (?nkf)
    using 71 by (meson b diff-less less-one nidx-nkfilter-gr)
  have 988: nlast (nkfilter f 0 (ntaken (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0 + k) (ndropns σ)))
=
    0 + (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0 + k)
    by (simp add: 90 91 nkfilter-nlast)
  have 989: nnth (nkfilter f 0 (ndropns σ)) (?nkf) = nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0 + k
    using 986
    by (metis 988 add.left-neutral ntaken-nlast)
  have 990: nnth (nkfilter f 0 (ndropns σ)) (?nkf) = nfirst (nkfilter f k (ndropn k (ndropns σ)))
    using 10 989 by fastforce
  have 991: nnth (nkfilter f 0 (ndropns σ)) (?nkf) = k + i
    using 10 101 989 by presburger
  show ?thesis
    by (metis 984 985 986 987 991 enat-the-enat infinity-ileE le-cases not-le-imp-less ntaken-all)
qed
have 100: h (nmap (λs. nnth s 0) (ndropn ?kf (nfilter f (ndropns σ))))
  using ndropns-nfilter-ndropn-a[of - f ndropn k σ ]
    nfilter-chop1-ndropn[of ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k) (ndropns σ) f ]
    101 61 76 78 8 90 91
  by simp
  (metis 10 ndropn-0 ndropn-ndropn ndropn-ndropns zero-enat-def zero-le)
have 11: h (ndropn ?kf (nmap (λs. nnth s 0) (nfilter f (ndropns σ))))
  by (simp add: 100 ndropn-nmap)
have 111: ?kf ≤ nlength (nfilter f (ndropns σ))
  by (metis 90 enat-ile le-cases nfilter-chop1-ntaken ntaken-all the-enat.simps)
have 112: nfinite (nfilter f (ntaken ((nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0)+k) (ndropns σ)))
  by (metis 90 91 nfilter-chop1-ntaken nfinite-ntaken)
have 113: (∀ j < k.
  (∃ x ∈ nset(ndropns (ndropn j σ)). f x) ∧
  g (nmap (λs. nnth s 0) (nfilter f (ndropns (ndropn j σ)))))
  by (metis 2 add commute in-nset-ndropns ndropn-ndropn ndropn-nlength)
have 151: (∀ jj < ?kf.
  (ndropn jj (nfilter f (ndropns σ))) =
  nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) jj) σ))
  ) using nfilter-nkfilter-ndropn
  by (simp add: 111 71 less-imp-le ndropns-nfilter-ndropn-a order-less-le-subst2)
have 152: (∧ jj. (enat jj) < ?kf ⟹ (nnth (nkfilter f 0 (ndropns σ)) jj) < k )
  proof -
    fix jj
    assume a: (enat jj) < ?kf
    show nnth (nkfilter f 0 (ndropns σ)) jj < k
    proof -
      have 1522: (enat jj) ≤ ?nkf - 1
        using 93 a by auto
      have 1523: nnth (nkfilter f 0 (ndropns σ)) (?nkf - 1) < nnth (nkfilter f 0 (ndropns σ)) ?nkf
        by (metis 111 71 93 a diff-less gr-implies-not-zero less-numeral-extra(1))

```

```

    nidx-nkfilter-gr nkfilter-nlength not-gr-zero zero-enat-def)
have 15230: enat (?nkf - 1) ≤ nlength (nkfilter f 0 (ndropns σ))
  by (metis 111 71 93 One-nat-def Suc-ile-eq Suc-pred a less-imp-le nkfilter-nlength
    not-gr-zero not-less-zero zero-enat-def)
have 1524: nnth (nkfilter f 0 (ndropns σ)) jj ≤ nnth (nkfilter f 0 (ndropns σ)) (?nkf - 1)
  using 1522 1523 71 93 15230
    nidx-less-eq[of (nkfilter f 0 (ndropns σ)) (jj) (?nkf - 1)]
    nidx-nkfilter[of (ndropns σ) f 0]
  using enat-ord-simps(1) by blast
have 1525: k ≤ nnth (nkfilter f 0 (ndropns σ)) (?nkf)
  using 94 add-leE by blast
have 1526: (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0) = 0 ⇒
  nnth (nkfilter f 0 (ndropns σ)) jj < k
  using 1523 1524 95 by linarith
have 1527: 0 < (nnth (nkfilter f 0 (ndropns (ndropn k σ))) 0) ⇒
  nnth (nkfilter f 0 (ndropns σ)) (?nkf - 1) < k
  using 93 98 a by auto
show ?thesis using 1524 1526 1527 dual-order.strict-trans2 by blast
qed
qed
have 153: (∀ jj < ?kf.
  g (nmap (λs. nnth s 0)
    (nfilter f (ndropns (ndropn (nnth (nkfilter f 0 (ndropns σ)) jj) σ)))
  ))
  using 112 13 152 nfinite-nlength-enat by force
have 1530: (∀ jj < ?kf.
  g (nmap (λs. nnth s 0) (ndropn jj (nfilter f (ndropns σ))))))
  by (simp add: 151 153)
have 1531: (∀ jj < ?kf.
  g (ndropn jj (nmap (λs. nnth s 0) (nfilter f (ndropns σ))))))
  by (simp add: 1530 ndropn-nmap)
have 154: ( (∃ i. (enat i) ≤ nlength σ ∧ f (ndropn i σ)) ∧
  (∃ kk. (enat kk) ≤ nlength (nfilter f (ndropns σ)) ∧
    h (ndropn kk (nmap (λs. nnth s 0) (nfilter f (ndropns σ)))))) ∧
  (∀ jj < kk. g (ndropn jj (nmap (λs. nnth s 0) (nfilter f (ndropns σ))))))
  using 11 111 1531 7 by blast
show ?thesis by (simp add: 154 PiUntilDistsem1)
qed

lemma PiUntilDistsem:
  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

```

10.3.15 PiChopstar

lemma *wnextboxnotstatesem*:

assumes $k \leq \text{nlength } \sigma$
shows $(\forall j \leq \text{nlength } \sigma. k < j \longrightarrow \neg (\lambda y. w (NNil y)) (\text{nnth } \sigma j)) =$
 $(LIFT(wnext (\Box (\text{init } (\neg w)))) (\text{ndropn } k \sigma))$

using *assms*
proof (*auto simp add: itl-defs ndropn-nfirst*)
show $\bigwedge n. \text{enat } k \leq \text{nlength } \sigma \implies$
 $\forall j. \text{enat } j \leq \text{nlength } \sigma \longrightarrow k < j \longrightarrow \neg w (NNil (\text{nnth } \sigma j)) \implies$
 $\text{nlength } \sigma - \text{enat } k \neq \text{enat } 0 \implies$
 $\text{enat } n \leq \text{nlength } \sigma - \text{enat } k - \text{enat } (Suc\ 0) \implies w (NNil (\text{nnth } \sigma (Suc\ (k + n)))) \implies False$
by (*metis enat-ile enat-ord-simps(2) le-add1 le-cases le-imp-less-Suc ndropn-nlast ndropn-nlength*
 $\text{nlength-NNil nlength-eq-enat-nfiniteD nnth-beyond order.not-eq-order-implies-strict}$
 $\text{the-enat.simps zero-enat-def}$)
show $\bigwedge j. \text{enat } k \leq \text{nlength } \sigma \implies \text{enat } j \leq \text{nlength } \sigma \implies k < j \implies w (NNil (\text{nnth } \sigma j)) \implies$
 $\text{nlength } \sigma - \text{enat } k = \text{enat } 0 \implies False$
by (*metis add.right-neutral enat.inject enat-add-sub-same enat-ord-code(4) leD less-eqE*
 $\text{min.strict-order-iff min-enat-simps(1) zero-enat-def}$)
show $\bigwedge j. \text{enat } k \leq \text{nlength } \sigma \implies$
 $\text{enat } j \leq \text{nlength } \sigma \implies$
 $k < j \implies w (NNil (\text{nnth } \sigma j)) \implies$
 $\forall n. \text{enat } n \leq \text{nlength } \sigma - \text{enat } k - \text{enat } (Suc\ 0) \longrightarrow \neg w (NNil (\text{nnth } \sigma (Suc\ (k + n)))) \implies False$
proof –
fix j
assume $a0: \text{enat } k \leq \text{nlength } \sigma$
assume $a1: \text{enat } j \leq \text{nlength } \sigma$
assume $a2: k < j$
assume $a3: w (NNil (\text{nnth } \sigma j))$
assume $a4: \forall n. \text{enat } n \leq \text{nlength } \sigma - \text{enat } k - \text{enat } (Suc\ 0) \longrightarrow \neg w (NNil (\text{nnth } \sigma (Suc\ (k + n))))$
show *False*
proof –
have 1: $j = (Suc\ (k + (j - (Suc\ k))))$
using $a2$ **by** *auto*
have 2: $\forall n. \text{enat } n + 1 + k \leq \text{nlength } \sigma \longrightarrow \neg w (NNil (\text{nnth } \sigma (Suc\ (k + n))))$
by *auto*
 $(metis One-nat-def a4 add.commute enat.simps(3) enat-add-sub-same enat-minus-mono1$
 $\text{one-enat-def})$
have 3: $(j - (Suc\ k)) + 1 + k \leq \text{nlength } \sigma$
using 1 $a1$ **by** *simp*
have 4: $\neg w (NNil (\text{nnth } \sigma (Suc\ (k + (j - (Suc\ k))))))$
using 2 3 $eSuc\text{-enat plus-1-eSuc(2)}$ **by** *auto*
from 1 4 $a3$ **show** *?thesis* **by** *auto*
qed
qed
qed

lemma *NotStateUntilStateAndsem:*

$(\sigma \models (\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge f)) =$
 $(\exists k. (\text{enat } k) \leq \text{nlength } \sigma \wedge w (NNil (\text{nnth } \sigma k)) \wedge f (\text{ndropn } k \sigma) \wedge (\forall j < k. \neg w (NNil (\text{nnth } \sigma j))))$

by (*auto simp add: until-d-def init-defs ndropn-nfirst*)

lemma *StateUntilEqvWPrevChopsem:*

$\sigma \models (\text{init } w) \mathcal{U} f = (\text{wprev } (\Box (\text{init } w))) \frown f$

proof (*auto simp add: until-d-def itl-defs ntaken-nnth ndropn-nfirst min-absorb1*)

show $\bigwedge k. \text{enat } k \leq \text{nlength } \sigma \implies$

$f (\text{ndropn } k \sigma) \implies$

$\forall j < k. w (\text{NNil } (\text{nnth } \sigma j)) \implies$

$\exists n. \text{enat } n \leq \text{nlength } \sigma \wedge$

$(\min (\text{enat } n) (\text{nlength } \sigma) = \text{enat } 0 \vee$

$(\forall na. na \leq \text{the-enat } (\min (\text{enat } (n - \text{Suc } 0)) (\text{epred } (\text{nlength } \sigma))) \wedge na \leq n \wedge \text{enat } na \leq$

$\text{nlength } \sigma \longrightarrow$

$w (\text{NNil } (\text{nnth } \sigma na)))) \wedge$

$f (\text{ndropn } n \sigma)$

by (*metis One-nat-def Suc-pred epred-enat epred-min le-imp-less-Suc min.orderE not-gr-zero the-enat.simps*)

next

fix n

assume $a0: \text{enat } n \leq \text{nlength } \sigma$

assume $a1: f (\text{ndropn } n \sigma)$

assume $a2: \forall na. na \leq \text{the-enat } (\min (\text{enat } (n - \text{Suc } 0)) (\text{epred } (\text{nlength } \sigma))) \wedge na \leq n \wedge$
 $\text{enat } na \leq \text{nlength } \sigma \longrightarrow w (\text{NNil } (\text{nnth } \sigma na))$

show $\exists k. \text{enat } k \leq \text{nlength } \sigma \wedge f (\text{ndropn } k \sigma) \wedge (\forall j < k. w (\text{NNil } (\text{nnth } \sigma j)))$

proof –

have $1: \text{enat } n \leq \text{nlength } \sigma \wedge f (\text{ndropn } n \sigma)$

using $a0 a1$ **by** *auto*

have $2: (\forall na. na \leq \text{the-enat } (\min (\text{enat } (n - \text{Suc } 0)) (\text{epred } (\text{nlength } \sigma))) \wedge na \leq n \wedge$
 $\text{enat } na \leq \text{nlength } \sigma \longrightarrow w (\text{NNil } (\text{nnth } \sigma na))) \longrightarrow$

$(\forall j < n. w (\text{NNil } (\text{nnth } \sigma j)))$

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 $3: (\forall j < n. w (\text{NNil } (\text{nnth } \sigma j)))$

using $2 a2$ **by** *blast*

show *?thesis*

using $1 3$ **by** *blast*

qed

qed

lemma *StateUntilEqvWPrevChop:*

$\vdash (\text{init } w) \mathcal{U} f = (\text{wprev } (\Box (\text{init } w))) \frown f$

using *StateUntilEqvWPrevChopsem Valid-def* **by** *blast*

lemma *UntilChopDist:*

$\vdash (\text{init } w) \mathcal{U} (g \frown h) = ((\text{init } w) \mathcal{U} g) \frown h$

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: $(\exists k \leq \text{nlength } \sigma. (\lambda y. w (\text{NNil } y)) (\text{nnth } \sigma k) \wedge$
 $(\forall j. j < k \longrightarrow \neg (\lambda y. w (\text{NNil } y)) (\text{nnth } \sigma j)) \wedge$
 $(\forall j \leq \text{nlength } \sigma. k < j \longrightarrow \neg (\lambda y. w (\text{NNil } y)) (\text{nnth } \sigma j))) =$
 $(\exists k \leq \text{nlength } \sigma.$
 $w (\text{NNil } (\text{nnth } \sigma k)) \wedge$
 $(\text{LIFT}(\text{wnext } (\Box (\text{init } (\neg w)))) (\text{ndropn } k \sigma) \wedge$
 $(\forall j < k. \neg w (\text{NNil } (\text{nnth } \sigma j)))) \text{ (is ?lhs = ?rhs)}$
proof
assume a: ?lhs
show ?rhs **using** a *wnextboznstatesem* **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 *wnextboznstatesem*[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* $\langle \text{enat } k \leq \text{nlength } \sigma \implies (\forall j. \text{enat } j \leq \text{nlength } \sigma \longrightarrow k < j \longrightarrow \neg w (\text{NNil } (\text{nnth } \sigma j))) \rangle$)
 $=$
 $(\text{LIFT}(\text{wnext } (\Box (\text{init } (\neg w)))) (\text{ndropn } k \sigma))$
then show ?thesis **using** f1 **by** *presburger*
qed
show ?thesis **using** 2 **by** *blast*

qed
 qed
 have 4: $(\exists k \leq \text{nlength } \sigma. w \text{ (NNil (nnth } \sigma \ k))} \wedge$
 $(\text{LIFT}(w\text{next } (\Box (\text{init } (\neg w)))) (\text{ndropn } k \ \sigma) \wedge$
 $(\forall j < k. \neg w \text{ (NNil (nnth } \sigma \ j)))) =$
 $(\sigma \models (\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge w\text{next } (\Box (\text{init } (\neg w)))))$
 by (simp add: NotStateUntilStateAndsem)
 from 1 2 3 4 show ?thesis by auto
 qed

lemma PiEmpty:
 $\vdash (\text{init } w) \Pi \text{ empty} = (\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge w\text{next } (\Box (\text{init } (\neg w))))$
 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: $((\text{nfilter } (\lambda y. w \text{ (NNil } y)) \sigma) \models f \wedge \Box (\text{init } w))$
 $= (f \text{ (nfilter } (\lambda y. w \text{ (NNil } y)) \sigma) \wedge$
 $(\forall n \leq \text{nlength } (\text{nfilter } (\lambda y. w \text{ (NNil } y)) \sigma). w \text{ (NNil (nnth (nfilter } (\lambda y. w \text{ (NNil } y)) \sigma) \ n))))$
 by (simp add: itl-defs ndropn-nfirst)
 have 4: $((\exists x \in \text{nset } \sigma. w \text{ (NNil } x)) \wedge (f \text{ (nfilter } (\lambda y. w \text{ (NNil } y)) \sigma) \wedge$
 $(\forall n \leq \text{nlength } (\text{nfilter } (\lambda y. w \text{ (NNil } y)) \sigma). w \text{ (NNil (nnth (nfilter } (\lambda y. w \text{ (NNil } y)) \sigma) \ n)))) =$
 $((\exists x \in \text{nset } \sigma. w \text{ (NNil } x)) \wedge f \text{ (nfilter } (\lambda y. w \text{ (NNil } y)) \sigma))$
 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:
 $\vdash ((\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge (\text{init } w) \Pi f)) =$
 $(w\text{prev } (\Box (\text{init } (\neg w)))) \frown ((\text{init } w) \wedge (\text{init } w) \Pi f)$
 using StateUntilEqWPrevChop by blast

lemma StatePiUntilsem2:
 $(\sigma \models (w\text{prev } (\Box (\text{init } (\neg w)))) \frown ((\text{init } w) \wedge (\text{init } w) \Pi f)) =$
 $(\sigma \models ((w\text{prev } (\Box (\text{init } (\neg w)))) \frown ((\text{init } w) \wedge \text{empty})) \frown ((\text{init } w) \wedge (\text{init } w) \Pi f))$
 by (auto simp add: schop-defs init-defs empty-defs min.absorb1 ndropn-nfirst)
 (metis diff-self-eq-0 idiff-enat-enat min.absorb1 ntaken-nnth order-refl)

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: min.absorb1 Valid-def itl-defs ndropn-nfirst*)
 $(metis ndropn-0 ndropn-nfirst ,$
 $metis add.right-neutral add-diff-inverse-nat enat.simps(3) enat-add-sub-same enat-ord-simps(2)$
 $iless-Suc-eq less-eqE min.absorb2 min-def ntaken-all one-eSuc one-enat-def order-refl$
 $plus-1-eq-Suc zero-enat-def zero-le)$
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 min.absorb1 ndropn-nfirst*)
 $(metis diff-self-eq-0 idiff-enat-enat min.orderE ntaken-nnth order-refl)$
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*:
 $\vdash (init\ w) \Pi f = (init\ (\neg\ w)) \mathcal{U} ((init\ w) \wedge (init\ w) \Pi f)$
using *StatePiUntilsem* **by** *blast*

lemma *StateAndPiEmpty*:
 $\vdash ((init\ w) \wedge (init\ w) \Pi empty) = (w \wedge empty) \frown (wnext\ (\Box (init\ (\neg\ w))))$
proof –
have 1: $\vdash ((init\ w) \wedge (init\ w) \Pi empty) =$
 $((init\ w) \wedge (init\ (\neg\ w)) \mathcal{U} ((init\ w) \wedge (wnext\ (\Box (init\ (\neg\ w)))))$
using *PiEmpty* **by** *fastforce*
have 2: $\vdash ((init\ w) \wedge (init\ (\neg\ w)) \mathcal{U} ((init\ w) \wedge (wnext\ (\Box (init\ (\neg\ w)))))$
 $= ((init\ w) \wedge (wnext\ (\Box (init\ (\neg\ w))))$
by (*auto simp add: until-d-def Valid-def init-defs ndropn-nfirst*)
(metis ndropn-0 ndropn-nfirst neq0-conv,
metis gr-implies-not-zero ndropn-0 ndropn-nfirst zero-enat-def zero-le)
have 3: $\vdash ((init\ w) \wedge (wnext\ (\Box (init\ (\neg\ w)))) = (w \wedge empty) \frown (wnext\ (\Box (init\ (\neg\ w))))$
proof –
have $\bigwedge p\ pa. \vdash ((p::'a\ nellist \Rightarrow bool) \wedge empty) \frown pa = (init\ p \wedge pa)$
by (*metis InitAndEmptyEqvAndEmpty StateAndEmptySChop inteq-reflection*)
then show *?thesis*
by (*simp add: Prop11*)
qed
show *?thesis*
by (*metis 1 2 3 inteq-reflection*)
qed

lemma *PiPowerExpandsem*:
 $(\sigma \models (\exists k. (init\ w) \Pi (spower\ f\ k))) =$
 $(\sigma \models (init\ w) \Pi empty \vee (\exists k. (init\ w) \Pi (f \frown (spower\ f\ (k)))))$
proof –
have 1: $(\sigma \models (\exists k. (init\ w) \Pi (spower\ f\ k))) =$
 $(\exists k. (\sigma \models (init\ w) \Pi (spower\ f\ k)))$
by *simp*
have 2: $(\exists k. (\sigma \models (init\ w) \Pi (spower\ f\ k))) =$
 $((\sigma \models (init\ w) \Pi (spower\ f\ 0)) \vee (\exists k. 1 \leq k \wedge (\sigma \models (init\ w) \Pi (spower\ f\ (k)))))$
by (*metis One-nat-def diff-Suc-1 le-SucE le-add1 plus-1-eq-Suc*)
have 3: $(\sigma \models (init\ w) \Pi (spower\ f\ 0)) = (\sigma \models (init\ w) \Pi empty)$
by *simp*
have 4: $(\exists k. 1 \leq k \wedge (\sigma \models (init\ w) \Pi (spower\ f\ (k)))) =$
 $(\exists k. (\sigma \models (init\ w) \Pi (spower\ f\ (Suc\ k))))$
by (*metis le-add1 ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc*)
have 5: $(\exists k. (\sigma \models (init\ w) \Pi (spower\ f\ (Suc\ k)))) =$
 $(\sigma \models (\exists k. (init\ w) \Pi (spower\ f\ (Suc\ k))))$
by *simp*
have 6: $((\sigma \models (init\ w) \Pi empty) \vee (\sigma \models (\exists k. (init\ w) \Pi (spower\ f\ (Suc\ k))))) =$
 $(\sigma \models (init\ w) \Pi empty \vee (\exists k. (init\ w) \Pi (f \frown spower\ f\ (k))))$

by auto
 show ?thesis using 1 2 3 4 5 6 by blast
 qed

lemma *PiPowerExpandsem1*:

$\forall \sigma. \sigma \models (\exists k. (init\ w) \Pi (spower\ f\ k)) =$
 $((init\ w) \Pi empty \vee (\exists k. (init\ w) \Pi (spower\ f\ (Suc\ k))))$

proof

fix σ

show $\sigma \models (\exists k. (init\ w) \Pi (spower\ f\ k)) =$
 $((init\ w) \Pi empty \vee (\exists k. (init\ w) \Pi (spower\ f\ (Suc\ k))))$

proof –

have 1: $(\sigma \models (\exists k. (init\ w) \Pi (spower\ f\ k))) =$
 $((init\ w) \Pi empty \vee (\exists k. (init\ w) \Pi (spower\ f\ (Suc\ k))))$
 $= ((\sigma \models (\exists k. (init\ w) \Pi (spower\ f\ k))) =$
 $(\sigma \models (init\ w) \Pi empty \vee (\exists k. (init\ w) \Pi (spower\ f\ (Suc\ k)))))$

by auto

have 2: $((\sigma \models (\exists k. (init\ w) \Pi (spower\ f\ k))) =$
 $(\sigma \models (init\ w) \Pi empty \vee (\exists k. (init\ w) \Pi (spower\ f\ (Suc\ k)))))$

using *PiPowerExpandsem*[of $w\ f\ \sigma$] by simp

show ?thesis using 1 2 by blast

qed

qed

lemma *PiPowerExpand*:

$\vdash (\exists k. (init\ w) \Pi (spower\ f\ k)) =$
 $((init\ w) \Pi empty \vee (\exists k. (init\ w) \Pi (spower\ f\ (Suc\ k))))$

using *PiPowerExpandsem1*[of $w\ f$] by (auto simp add: Valid-def *PiPowerExpandsem1*)

lemma *exists-expand-sem*:

$(\sigma \models (\exists k. (spower\ ((init\ w) \Pi f \wedge fin\ w)\ k))) =$
 $((\sigma \models (spower\ ((init\ w) \Pi f \wedge fin\ w)\ 0)) \vee$
 $(\sigma \models (\exists k. (spower\ ((init\ w) \Pi f \wedge fin\ w)\ (Suc\ k))))$

by (metis (no-types, lifting) not0-implies-Suc unl-Rex)

lemma *exists-expand*:

$\vdash (\exists k. (spower\ ((init\ w) \Pi f \wedge fin\ w)\ k)) =$
 $((spower\ ((init\ w) \Pi f \wedge fin\ w)\ 0) \vee (\exists k. (spower\ ((init\ w) \Pi f \wedge fin\ w)\ (Suc\ k))))$

using *exists-expand-sem* Valid-def by fastforce

10.3.16 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)

10.3.17 BoxImpEqvPi

lemma *BoxImpEqvPi*:

$\vdash \Box f \longrightarrow g = f \Pi g$

proof (*simp add: Valid-def itl-defs pi-d-def pifilt-def sxfilt-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

10.3.18 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*)

10.3.19 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*)

10.3.20 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*)

10.4 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 $\vdash f \amalg g1 = f \amalg g2$
proof –
have 1: $\vdash g1 \longrightarrow g2$
using *assms* **by** (*simp add: int-iffD1*)
have 2: $\vdash f \amalg g1 \longrightarrow f \amalg g2$
using 1 *PiImpRule* **by** *blast*
have 3: $\vdash g2 \longrightarrow g1$
using *assms* **by** (*simp add: int-iffD2*)
have 4: $\vdash f \amalg g2 \longrightarrow f \amalg g1$
using 3 *PiImpRule* **by** *blast*
from 2 4 **show** ?thesis **by** (*simp add: int-iffI*)
qed

lemma *UPiAndPiImpPiAnd*:
 $\vdash f1 \amalg^u f \wedge f1 \amalg (\neg g) \longrightarrow f1 \amalg (f \wedge \neg g)$
proof –
have 1: $\vdash (\neg(f \longrightarrow g)) = (f \wedge \neg g)$
by *fastforce*
have 2: $\vdash (\neg(f1 \amalg^u (f \longrightarrow g))) = f1 \amalg (\neg(f \longrightarrow g))$
by (*simp add: NotPiEqvNotUPi int-iffD1 int-iffD2 int-iffI*)
have 3: $\vdash \neg(f1 \amalg^u f \longrightarrow f1 \amalg^u g) \longrightarrow \neg(f1 \amalg^u (f \longrightarrow g))$
by (*simp add: PiK*)
have 4: $\vdash (\neg(f1 \amalg^u f \longrightarrow f1 \amalg^u g)) = (f1 \amalg^u f \wedge f1 \amalg (\neg g))$
using *NotPiEqvNotUPi*[*of f1 g*] **by** *fastforce*
have 5: $\vdash f1 \amalg (\neg(f \longrightarrow g)) = f1 \amalg (f \wedge \neg g)$
using 1 **by** (*simp add: PiEqvRule*)
from 1 2 3 4 5 **show** ?thesis **by** *fastforce*
qed

lemma *UPiAndPiImpPiAndA*:
 $\vdash f1 \amalg^u f \wedge f1 \amalg g \longrightarrow f1 \amalg (f \wedge g)$
using *UPiAndPiImpPiAnd*[*of f1 f LIFT(¬g)*] **by** *fastforce*

lemma *PiAndPiImpPiAnd*:
 $\vdash f1 \amalg f \wedge f1 \amalg g \longrightarrow f1 \amalg (f \wedge g)$
proof –
have 1: $\vdash f1 \amalg^u f \wedge f1 \amalg g \longrightarrow f1 \amalg (f \wedge g)$
using *UPiAndPiImpPiAndA* **by** *fastforce*
have 2: $\vdash f1 \amalg f \longrightarrow f1 \amalg^u f$
using *PiDc* **by** *blast*
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *PiAnd*:
 $\vdash f \amalg (g1 \wedge g2) = (f \amalg g1 \wedge f \amalg g2)$
proof –
have 1: $\vdash f \amalg (g1 \wedge g2) \longrightarrow f \amalg g1$
by (*meson PiImpRule Prop12 int-iffD1 lift-and-com*)
have 2: $\vdash f \amalg (g1 \wedge g2) \longrightarrow f \amalg g2$
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 *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, hide-lams) InitAndEmptyEqvAndEmpty Prop11 Prop12 StateAndEmptySChop int-simps(1) inteq-reflection*)

show *?thesis*

by (*simp add: 1 2 3 Prop12 int-iffI*)

qed

lemma *PiChopstarhelp2a*:

$\vdash (w \wedge empty) \frown (spower\ ((f \frown (w \wedge empty)) \wedge more)\ k) =$
 $(spower\ (((w \wedge empty) \frown f) \wedge more)\ k) \frown (w \wedge empty)$

proof (*induction k*)

case 0

then show *?case*

proof –

have $\bigwedge p. \vdash (empty \wedge finite);p = p$

by (*metis EmptySChop schop-d-def*)

then show *?thesis*

by (*metis (no-types) PowerSucAndEmptyEqvAndEmpty Prop04 pow-0 pow-Suc schop-d-def spow-0*)

qed

next

case (*Suc k*)

then show *?case*

proof –

have 1: $\vdash (w \wedge \text{empty}) \multimap \text{spower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) (\text{Suc } k) =$
 $(w \wedge \text{empty}) \multimap ((f \multimap (w \wedge \text{empty}) \wedge \text{more}) \multimap \text{spower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k)$
by *simp*
have 11: $\vdash (f \wedge \text{more}) \multimap (w \wedge \text{empty}) = (\text{sfin } w \wedge (f \wedge \text{more}))$
by (*metis* *SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty int-eq*)
have 12: $\vdash (f \multimap (w \wedge \text{empty})) = (\text{sfin } w \wedge f)$
by (*metis* *SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection*)
have 13: $\vdash (f \multimap (w \wedge \text{empty}) \wedge \text{more}) = ((\text{sfin } w \wedge f) \wedge \text{more})$
using 12 **by** *auto*
have 14: $\vdash ((\text{sfin } w \wedge f) \wedge \text{more}) = (\text{sfin } w \wedge (f \wedge \text{more}))$
by *fastforce*
have 2: $\vdash (f \multimap (w \wedge \text{empty}) \wedge \text{more}) = (f \wedge \text{more}) \multimap (w \wedge \text{empty})$
using 11 13 **by** *fastforce*
have 20: $\vdash ((f \multimap (w \wedge \text{empty}) \wedge \text{more}) \multimap \text{spower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k) =$
 $((f \wedge \text{more}) \multimap (w \wedge \text{empty})) \multimap \text{spower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k$
by (*simp* *add: 2 LeftSChopEqvSChop*)
have 21: $\vdash ((f \wedge \text{more}) \multimap (w \wedge \text{empty})) \multimap \text{spower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k =$
 $(f \wedge \text{more}) \multimap ((w \wedge \text{empty}) \multimap \text{spower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k)$
using *SChopAssocB* **by** *blast*
have 22: $\vdash (f \wedge \text{more}) \multimap ((w \wedge \text{empty}) \multimap \text{spower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k) =$
 $((f \wedge \text{more}) \multimap (\text{spower } ((w \wedge \text{empty}) \multimap f \wedge \text{more}) k \multimap (w \wedge \text{empty}))))$
by (*simp* *add: RightSChopEqvSChop Suc.IH*)
have 23: $\vdash (w \wedge \text{empty}) \multimap ((f \multimap (w \wedge \text{empty}) \wedge \text{more}) \multimap \text{spower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k) =$
 $(w \wedge \text{empty}) \multimap (((f \wedge \text{more}) \multimap (w \wedge \text{empty})) \multimap \text{spower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k)$
using 20 *RightSChopEqvSChop* **by** *blast*
have 24: $\vdash (w \wedge \text{empty}) \multimap (((f \wedge \text{more}) \multimap (w \wedge \text{empty})) \multimap \text{spower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k) =$
 $(w \wedge \text{empty}) \multimap ((f \wedge \text{more}) \multimap ((w \wedge \text{empty}) \multimap \text{spower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k))$
using 21 *RightSChopEqvSChop* **by** *blast*
have 25: $\vdash (w \wedge \text{empty}) \multimap ((f \wedge \text{more}) \multimap ((w \wedge \text{empty}) \multimap \text{spower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k)) =$
 $(w \wedge \text{empty}) \multimap ((f \wedge \text{more}) \multimap (\text{spower } ((w \wedge \text{empty}) \multimap f \wedge \text{more}) k \multimap (w \wedge \text{empty}))))$
using 23 22 *RightSChopEqvSChop* **by** *blast*
have 3: $\vdash (w \wedge \text{empty}) \multimap ((f \multimap (w \wedge \text{empty}) \wedge \text{more}) \multimap \text{spower } (f \multimap (w \wedge \text{empty}) \wedge \text{more}) k) =$
 $(w \wedge \text{empty}) \multimap ((f \wedge \text{more}) \multimap (\text{spower } ((w \wedge \text{empty}) \multimap f \wedge \text{more}) k \multimap (w \wedge \text{empty}))))$
using 23 24 25 **by** *fastforce*
have 4: $\vdash (w \wedge \text{empty}) \multimap ((f \wedge \text{more}) \multimap (\text{spower } ((w \wedge \text{empty}) \multimap f \wedge \text{more}) k \multimap (w \wedge \text{empty})))) =$
 $((w \wedge \text{empty}) \multimap (f \wedge \text{more})) \multimap ((\text{spower } ((w \wedge \text{empty}) \multimap f \wedge \text{more}) k \multimap (w \wedge \text{empty}))))$
using *SChopAssoc* **by** *blast*
have 6: $\vdash ((w \wedge \text{empty}) \multimap (f \wedge \text{more})) \multimap ((\text{spower } ((w \wedge \text{empty}) \multimap f \wedge \text{more}) k \multimap (w \wedge \text{empty})))) =$
 $((w \wedge \text{empty}) \multimap f \wedge \text{more}) \multimap ((\text{spower } ((w \wedge \text{empty}) \multimap f \wedge \text{more}) k \multimap (w \wedge \text{empty}))))$
by (*meson* *EmptyAndSChopAndMoreEqvAndSChop LeftSChopEqvSChop*)
have 7: $\vdash ((w \wedge \text{empty}) \multimap f \wedge \text{more}) \multimap ((\text{spower } ((w \wedge \text{empty}) \multimap f \wedge \text{more}) k \multimap (w \wedge \text{empty})))) =$
 $((\text{spower } ((w \wedge \text{empty}) \multimap f \wedge \text{more}) (\text{Suc } k) \multimap (w \wedge \text{empty}))))$
by (*simp* *add: SChopAssoc*)
show ?thesis **using** 1 3 4 6 7
by (*metis* *inteq-reflection*)
qed
qed

lemma *PiChopstarhelp2*:

$\vdash (w \wedge \text{empty}) \neg (\text{schopstar}(f \neg (w \wedge \text{empty}))) = (\text{schopstar}((w \wedge \text{empty}) \neg f)) \neg (w \wedge \text{empty})$

proof –

have 1: $\vdash (w \wedge \text{empty}) \neg (\text{schopstar}(f \neg (w \wedge \text{empty}))) =$
 $(w \wedge \text{empty}) \neg (\exists k. \text{spower } (f \neg (w \wedge \text{empty})) \wedge \text{more}) k)$
by (*simp add: schopstar-d-def spowerstar-d-def*)
have 2: $\vdash (w \wedge \text{empty}) \neg (\exists k. \text{spower } (f \neg (w \wedge \text{empty})) \wedge \text{more}) k) =$
 $(\exists k. (w \wedge \text{empty}) \neg (\text{spower } (f \neg (w \wedge \text{empty})) \wedge \text{more}) k))$
using *SChopExist* **by** *fastforce*
have 3: $\vdash (\exists k. (w \wedge \text{empty}) \neg (\text{spower } (f \neg (w \wedge \text{empty})) \wedge \text{more}) k) =$
 $(\exists k. (\text{spower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k) \neg (w \wedge \text{empty}))$
by (*simp add: ExEqvRule PiChopstarhelp2a*)
have 4: $\vdash (\exists k. (\text{spower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k) \neg (w \wedge \text{empty})) =$
 $(\exists k. (\text{spower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k)) \neg (w \wedge \text{empty})$
using *ExistSChop* **by** *fastforce*
have 5: $\vdash (\exists k. (\text{spower } ((w \wedge \text{empty}) \neg f \wedge \text{more}) k)) \neg (w \wedge \text{empty}) =$
 $(\text{schopstar}((w \wedge \text{empty}) \neg f)) \neg (w \wedge \text{empty})$
by (*simp add: schopstar-d-def spowerstar-d-def*)
show *?thesis*
by (*metis 1 2 3 4 5 int-eq*)
qed

lemma *PiPowerSuca*:

$\vdash (\text{init } w) \Pi (\text{spower } f (\text{Suc } k)) = ((\text{init } w) \Pi f) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{spower } f k))$
by (*simp add: PiSChopDist*)

lemma *PiPowerSuch*:

$\vdash (\text{init } w) \Pi (\text{spower } f (\text{Suc } k)) = ((\text{init } w) \Pi f \wedge \text{sfin } w) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{spower } f k))$

proof –

have 0: $\vdash (\text{init } w) \Pi (\text{spower } f (\text{Suc } k)) = ((\text{init } w) \Pi f) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{spower } f k))$
by (*meson PiPowerSuca*)
have 1: $\vdash ((\text{init } w) \wedge (\text{init } w) \Pi (\text{spower } f k)) = (w \wedge \text{empty}) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{spower } f k))$
by (*metis (no-types, lifting) AndEmptySChopAndEmptyEqvAndEmpty InitAndEmptyEqvAndEmpty SChopAssoc*
StateAndEmptySChop int-eq)
have 2: $\vdash ((\text{init } w) \Pi f) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{spower } f k)) =$
 $((\text{init } w) \Pi f) \neg ((w \wedge \text{empty}) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{spower } f k)))$
using 1 *RightSChopEqvSChop* **by** *blast*
have 3: $\vdash ((\text{init } w) \Pi f) \neg ((w \wedge \text{empty}) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{spower } f k))) =$
 $((\text{init } w) \Pi f) \neg ((w \wedge \text{empty}) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{spower } f k)))$
by (*simp add: SChopAssoc*)
have 4: $\vdash (((\text{init } w) \Pi f) \neg (w \wedge \text{empty})) = ((\text{init } w) \Pi f \wedge \text{sfin } w)$
by (*metis SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection lift-and-com*)
have 5: $\vdash (((\text{init } w) \Pi f) \neg (w \wedge \text{empty})) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{spower } f k)) =$
 $((\text{init } w) \Pi f \wedge \text{sfin } w) \neg ((\text{init } w) \wedge (\text{init } w) \Pi (\text{spower } f k))$
by (*simp add: 4 LeftSChopEqvSChop*)
show *?thesis*
using 0 2 3 5 **by** *fastforce*
qed

lemma *PiPowerSucc*:

$\vdash (init\ w) \Pi (spower\ f\ (Suc\ k)) =$
 $(spower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w) \wedge (init\ w) \Pi empty)$
proof (*induction k*)
case 0
then show ?case
using *PiPowerSucb[of w f 0]*
unfolding *schop-d-def spow-0 spow-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: $\vdash (init\ w \Pi f) \frown$
 $(init\ w \wedge spower\ (init\ w \Pi f \wedge sfin\ w)\ (Suc\ k) \frown (init\ w \wedge init\ w \Pi empty)) =$
 $((init\ w \Pi f \wedge sfin\ w) \frown spower\ (init\ w \Pi f \wedge sfin\ w)\ (Suc\ k)) \frown$
 $(init\ w \wedge init\ w \Pi empty)$
by (*metis (no-types, lifting) AndSFinSChopEqvStateAndSChop InitAndEmptyEqvAndEmpty SChopAssoc*
SFinEqvTrueSChopAndEmpty inteq-reflection)
have 4: $\vdash (init\ w \Pi f) \frown (init\ w \wedge init\ w \Pi (f \frown spower\ f\ k)) =$
 $((init\ w \Pi f \wedge sfin\ w) \frown$
 $((init\ w \Pi f \wedge sfin\ w) \frown spower\ (init\ w \Pi f \wedge sfin\ w)\ k)) \frown$
 $(init\ w \wedge init\ w \Pi empty)$
using 3 *Suc.IH* **unfolding** *spow-Suc* **by** (*metis int-eq*)
show ?thesis **using** *PiPowerSuca[of w f k]* 3 **unfolding** *spow-Suc*
by (*metis 4 PiSChopDist inteq-reflection*)
qed
qed

lemma *PiPowerSuccd*:

$\vdash (init\ w) \Pi (spower\ f\ (Suc\ k)) = (spower\ ((init\ w) \Pi f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w) \Pi empty)$

proof (*induction k*)

case 0

then show ?case **unfolding** *spow-0 spow-Suc*

by (*metis (no-types, lifting) AndSFinSChopEqvStateAndSChop EmptySChop InitAndEmptyEqvAndEmpty*
PiSChopDist SChopAssocB SFinEqvTrueSChopAndEmpty int-eq)

next

case (*Suc k*)

then show ?case

by (*metis (no-types, lifting) PiPowerSucc SChopAssoc inteq-reflection spow-Suc*)

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 SFinEqvTrueSChopAndEmpty*
TrueSChopEqvDiamond int-eq)
show ?thesis
using 1 2 3 **by** (*metis* *inteq-reflection*)
qed

lemma *PiChopstar*:

$\vdash (\text{init}\ w) \Pi (\text{scho}pstar\ f) =$
 $(\text{init}\ (\neg w)) \mathcal{U} ((\text{init}\ w) \wedge (\text{scho}pstar((\text{init}\ w) \Pi f) \wedge \text{sf}in\ w) \frown \text{wnext}(\Box (\text{init}\ (\neg w))))$

proof –

have 1: $\vdash (\text{init}\ w) \Pi (\text{scho}pstar\ f) = (\text{init}\ w) \Pi (\exists k. \text{spower}\ f\ k)$
by (*metis* *SChopstarEqvSPowerstar PiEqvRule spowerstar-d-def*)
have 2: $\vdash (\text{init}\ w) \Pi (\exists k. \text{spower}\ f\ k) = (\exists k. (\text{init}\ w) \Pi (\text{spower}\ f\ k))$
by (*simp* *add: Valid-def pi-d-def*)
have 3: $\vdash (\exists k. (\text{init}\ w) \Pi (\text{spower}\ f\ k)) =$
 $((\text{init}\ w) \Pi \text{empty} \vee (\exists k. (\text{init}\ w) \Pi (\text{spower}\ f\ (\text{Suc}\ k))))$
using *PiPowerExpand* **by** *auto*
have 4: $\vdash (\exists k. (\text{init}\ w) \Pi (\text{spower}\ f\ (\text{Suc}\ k))) =$
 $(\exists k. (\text{spower}\ ((\text{init}\ w) \Pi f \wedge \text{sf}in\ w) (\text{Suc}\ k)) \frown ((\text{init}\ w) \Pi \text{empty}))$
by (*meson* *ExEqvRule PiPowerSuccd*)
have 5: $\vdash (\text{init}\ w) \Pi \text{empty} = \text{empty} \frown ((\text{init}\ w) \Pi \text{empty})$
by (*simp* *add: EmptySChop int-iffD1 int-iffD2 int-iffI*)
have 6: $\vdash (\exists k. (\text{spower}\ ((\text{init}\ w) \Pi f \wedge \text{sf}in\ w) (\text{Suc}\ k)) \frown ((\text{init}\ w) \Pi \text{empty})) =$
 $(\exists k. (\text{spower}\ ((\text{init}\ w) \Pi f \wedge \text{sf}in\ w) (\text{Suc}\ k)) \frown ((\text{init}\ w) \Pi \text{empty}))$
by (*simp* *add: Semantics.ExistSChop*)
have 7: $\vdash ((\text{init}\ w) \Pi \text{empty} \vee (\exists k. (\text{spower}\ ((\text{init}\ w) \Pi f \wedge \text{sf}in\ w) (\text{Suc}\ k)) \frown ((\text{init}\ w) \Pi \text{empty}))) =$
 $(\text{empty} \frown ((\text{init}\ w) \Pi \text{empty}) \vee$
 $(\exists k. (\text{spower}\ ((\text{init}\ w) \Pi f \wedge \text{sf}in\ w) (\text{Suc}\ k)) \frown ((\text{init}\ w) \Pi \text{empty}))$
using 5 6 **by** *fastforce*
have 8: $\vdash (\text{empty} \frown ((\text{init}\ w) \Pi \text{empty}) \vee$
 $(\exists k. (\text{spower}\ ((\text{init}\ w) \Pi f \wedge \text{sf}in\ w) (\text{Suc}\ k)) \frown ((\text{init}\ w) \Pi \text{empty})) =$
 $(\text{empty} \vee (\exists k. (\text{spower}\ ((\text{init}\ w) \Pi f \wedge \text{sf}in\ w) (\text{Suc}\ k)))) \frown ((\text{init}\ w) \Pi \text{empty})$
by (*meson* *OrSChopEqv Prop11*)
have 9: $\vdash \text{empty} = (\text{spower}\ ((\text{init}\ w) \Pi f \wedge \text{sf}in\ w) 0)$
by *simp*
have 10: $\vdash (\text{empty} \vee (\exists k. (\text{spower}\ ((\text{init}\ w) \Pi f \wedge \text{sf}in\ w) (\text{Suc}\ k)))) =$
 $((\text{spower}\ ((\text{init}\ w) \Pi f \wedge \text{sf}in\ w) 0) \vee (\exists k. (\text{spower}\ ((\text{init}\ w) \Pi f \wedge \text{sf}in\ w) (\text{Suc}\ k))))$
by *simp*
have 11: $\vdash ((\text{spower}\ ((\text{init}\ w) \Pi f \wedge \text{sf}in\ w) 0) \vee (\exists k. (\text{spower}\ ((\text{init}\ w) \Pi f \wedge \text{sf}in\ w) (\text{Suc}\ k)))) =$
 $(\exists k. (\text{spower}\ ((\text{init}\ w) \Pi f \wedge \text{sf}in\ w) k))$

using *exists-expand*[*of w f*]
by (*metis* (*mono-tags*) *Valid-def inteq-reflection spow-0 spowersem1*)
have 12: $\vdash ((init\ w) \Pi\ empty \vee$
 $(\exists k. (spower\ ((init\ w) \Pi\ f \wedge sfin\ w)\ (Suc\ k)) \neg ((init\ w) \Pi\ empty))) =$
 $(\exists k. (spower\ ((init\ w) \Pi\ f \wedge sfin\ w)\ (k)) \neg ((init\ w) \Pi\ empty))$
by (*metis* 11 7 8 9 *inteq-reflection*)
have 13: $\vdash (\exists k. (spower\ ((init\ w) \Pi\ f \wedge sfin\ w)\ (k)) \neg ((init\ w) \Pi\ empty) =$
 $(schopstar\ ((init\ w) \Pi\ f \wedge sfin\ w)) \neg ((init\ w) \Pi\ empty)$
by (*metis* *SChopstarEqvSPowerstar LeftSChopEqvSChop inteq-reflection spowerstar-d-def*)
have 14: $\vdash (schopstar\ ((init\ w) \Pi\ f \wedge sfin\ w)) \neg ((init\ w) \Pi\ empty) =$
 $((init\ w) \Pi\ empty) \vee$
 $((init\ w) \Pi\ f \wedge sfin\ w) \neg (schopstar\ ((init\ w) \Pi\ f \wedge sfin\ w)) \neg ((init\ w) \Pi\ empty))$
using *SCSSChopEqvOrSChopPlusSChop* **by** *blast*
have 15: $\vdash ((init\ w) \Pi\ empty) = (init\ (\neg w)) \mathcal{U} ((init\ w) \wedge wnext\ (\Box (init\ (\neg w))))$
by (*simp* *add: PiEmpty*)
have 16: $\vdash ((init\ w) \Pi\ f \wedge sfin\ w) \neg (schopstar\ ((init\ w) \Pi\ f \wedge sfin\ w)) \neg ((init\ w) \Pi\ empty) =$
 $((init\ w) \Pi\ f \wedge sfin\ w) \neg (schopstar\ ((init\ w) \Pi\ f \wedge sfin\ w)) \neg wnext\ (\Box (init\ (\neg w)))$
proof –
have 161: $\vdash ((init\ w) \Pi\ f \wedge sfin\ w) \neg (schopstar\ ((init\ w) \Pi\ f \wedge sfin\ w)) \neg ((init\ w) \Pi\ empty) =$
 $((init\ w) \Pi\ f \wedge sfin\ w) \neg ((schopstar\ ((init\ w) \Pi\ f \wedge sfin\ w)) \neg ((init\ w) \Pi\ empty))$
by (*simp* *add: SChopAssocB*)
have 162: $\vdash ((init\ w) \Pi\ f \wedge sfin\ w) = ((init\ w) \Pi\ f) \neg (w \wedge empty)$
by (*metis* *SChopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty inteq-reflection lift-and-com*)
have 163: $\vdash (schopstar\ ((init\ w) \Pi\ f \wedge sfin\ w)) = (schopstar\ (((init\ w) \Pi\ f) \neg (w \wedge empty)))$
by (*simp* *add: 162 SCSEqvSCS*)
have 164: $\vdash ((init\ w) \Pi\ f \wedge sfin\ w) \neg (schopstar\ ((init\ w) \Pi\ f \wedge sfin\ w)) =$
 $((init\ w) \Pi\ f) \neg ((w \wedge empty) \neg (schopstar\ ((init\ w) \Pi\ f) \neg (w \wedge empty)))$
using 162 *SChopAssocB int-eq* **by** *metis*
have 165: $\vdash ((init\ w) \Pi\ f) \neg ((w \wedge empty) \neg (schopstar\ ((init\ w) \Pi\ f) \neg (w \wedge empty))) =$
 $((init\ w) \Pi\ f) \neg ((schopstar\ ((w \wedge empty) \neg ((init\ w) \Pi\ f))) \neg (w \wedge empty))$
by (*simp* *add: PiChopstarhelp2 RightSChopEqvSChop*)
have 166: $\vdash (((init\ w) \Pi\ f) \neg ((schopstar\ ((w \wedge empty) \neg ((init\ w) \Pi\ f))) \neg (w \wedge empty))) \neg ((init\ w) \Pi\ empty) =$
 $((init\ w) \Pi\ f) \neg (schopstar\ ((w \wedge empty) \neg ((init\ w) \Pi\ f))) \neg ((init\ w) \wedge ((init\ w) \Pi\ empty))$
by (*metis* (*no-types, lifting*) *InitAndEmptyEqvAndEmpty SChopAssocB StateAndEmptySChop int-eq*)
have 167: $\vdash (((init\ w) \Pi\ f) \neg (schopstar\ ((w \wedge empty) \neg ((init\ w) \Pi\ f))) \neg ((init\ w) \wedge ((init\ w) \Pi\ empty))) =$
 $((init\ w) \Pi\ f) \neg (schopstar\ ((w \wedge empty) \neg ((init\ w) \Pi\ f))) \neg ((w \wedge empty) \neg wnext\ (\Box (init\ (\neg w))))$
by (*simp* *add: RightSChopEqvSChop StateAndPiEmpty*)
have 168: $\vdash (((init\ w) \Pi\ f) \neg (schopstar\ ((w \wedge empty) \neg ((init\ w) \Pi\ f))) \neg ((w \wedge empty) \neg wnext\ (\Box (init\ (\neg w)))) =$
 $((init\ w) \Pi\ f) \neg (schopstar\ ((w \wedge empty) \neg ((init\ w) \Pi\ f))) \neg (w \wedge empty) \neg wnext\ (\Box (init\ (\neg w)))$
by (*simp* *add: SChopAssoc*)
have 169: $\vdash (((init\ w) \Pi\ f) \neg (schopstar\ ((w \wedge empty) \neg ((init\ w) \Pi\ f))) \neg (w \wedge empty)) =$
 $((init\ w) \Pi\ f \wedge sfin\ w) \neg (schopstar\ ((init\ w) \Pi\ f \wedge sfin\ w))$
using 164 165 *SChopAssocB* **by** *fastforce*

```

show ?thesis by (metis 164 165 166 167 168 169 int-eq)
qed
have 17:  $\vdash ((init\ w) \amalg f \wedge sfin\ w) = ((init\ w) \amalg f) \frown ((init\ w) \wedge empty)$ 
by (metis InitAndEmptyEqvAndEmpty SchopAndEmptyEqvSChopAndEmpty SFinEqvTrueSChopAndEmpty
inteq-reflection lift-and-com)
have 18:  $\vdash ((init\ w) \amalg f) = wprev(\Box (init\ (\neg w))) \frown ((init\ w) \wedge (init\ w) \amalg f)$ 
by (simp add: WPrevPi)
have 19:  $\vdash wprev(\Box (init\ (\neg w))) \frown ((init\ w) \wedge (init\ w) \amalg f) =$ 
 $wprev(\Box (init\ (\neg w))) \frown (((init\ w) \wedge empty) \frown ((init\ w) \amalg f))$ 
by (meson RightSChopImpSChop StateAndEmptySChop int-iffD1 int-iffD2 int-iffI)
have 20:  $\vdash wprev(\Box (init\ (\neg w))) \frown (((init\ w) \wedge empty) \frown ((init\ w) \amalg f)) =$ 
 $(init\ (\neg w)) \mathcal{U} ((init\ w) \wedge ((init\ w) \amalg f))$ 
using 18 19 StatePiUntil by fastforce
have 21:  $\vdash (((init\ w) \amalg f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \amalg f \wedge sfin\ w))) \frown wnext\ (\Box (init\ (\neg w))) =$ 
 $(init\ (\neg w)) \mathcal{U} ((init\ w) \wedge$ 
 $((((init\ w) \amalg f) \wedge sfin\ w) \frown (schopstar\ ((init\ w) \amalg f \wedge sfin\ w))) \frown wnext\ (\Box (init\ (\neg w))))$ 
proof –
have  $\vdash (init\ w \amalg f) \frown (init\ w \wedge empty) = (init\ w \amalg f \wedge sfin\ w)$ 
by (metis 17 inteq-reflection)
then show ?thesis using StatePiUntil[of w f]
UntilChopDist[of w LIFT((init w  $\wedge$  empty)) LIFT((schopstar ((init w)  $\amalg$  f  $\wedge$  sfin w)))  $\frown$  wnext ( $\Box$  (init
 $\neg$  w)))]
by (metis (no-types, lifting) AndSFinSChopEqvStateAndSChop SChopAssocB StateUntilEqvWPrevChop
int-eq)
qed
have 22:  $\vdash ((init\ (\neg w)) \mathcal{U} ((init\ w) \wedge wnext\ (\Box (init\ (\neg w))))) \vee$ 
 $(init\ (\neg w)) \mathcal{U} ((init\ w) \wedge$ 
 $((((init\ w) \amalg f) \wedge sfin\ w) \frown (schopstar\ ((init\ w) \amalg f \wedge sfin\ w))) \frown wnext\ (\Box (init\ (\neg w))))$ 
 $) =$ 
 $(init\ (\neg w)) \mathcal{U} ((init\ w) \wedge wnext\ (\Box (init\ (\neg w)))) \vee$ 
 $((init\ w) \wedge (((init\ w) \amalg f) \wedge sfin\ w) \frown (schopstar\ ((init\ w) \amalg f \wedge sfin\ w))) \frown wnext\ (\Box (init\ (\neg w))))$ 
using UntilOrDist by fastforce
have 23:  $\vdash ((init\ w) \wedge wnext\ (\Box (init\ (\neg w)))) \vee$ 
 $((init\ w) \wedge (((init\ w) \amalg f) \wedge sfin\ w) \frown (schopstar\ ((init\ w) \amalg f \wedge sfin\ w))) \frown wnext\ (\Box (init\ (\neg w))))$ 
 $=$ 
 $((init\ w) \wedge (schopstar\ (((init\ w) \amalg f) \wedge sfin\ w)) \frown wnext\ (\Box (init\ (\neg w))))$ 
by (metis (no-types, lifting) SCSSChopEqvOrSChopPlusSChop SChopOrEqv StateAndEmptySChop inteq-reflection)
have 24:  $\vdash (init\ w) \amalg (schopstar\ f) = ((init\ w) \amalg empty \vee (\exists k. (init\ w) \amalg (spower\ f\ (Suc\ k))))$ 
using 1 2 3 by fastforce
have 25:  $\vdash ((init\ w) \amalg empty \vee (\exists k. (init\ w) \amalg (spower\ f\ (Suc\ k)))) =$ 
 $((init\ w) \amalg empty \vee (\exists k. (spower\ ((init\ w) \amalg f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w) \amalg empty)))$ 
using 4 by fastforce

```

```

have 26:  $\vdash ((init\ w) \sqcap empty) \vee$ 
       $(\exists k. (spower\ ((init\ w) \sqcap f \wedge sfin\ w)\ (Suc\ k)) \frown ((init\ w) \sqcap empty))) =$ 
       $(schopstar\ ((init\ w) \sqcap f \wedge sfin\ w)) \frown ((init\ w) \sqcap empty)$ 
using 12 13 by fastforce
have 27:  $\vdash (schopstar\ ((init\ w) \sqcap f \wedge sfin\ w)) \frown ((init\ w) \sqcap empty) =$ 
       $((init\ (\neg w)) \mathcal{U} ((init\ w) \wedge wnext\ (\Box (init\ (\neg w)))) \vee$ 
       $((init\ w) \sqcap f \wedge sfin\ w) \frown (schopstar\ ((init\ w) \sqcap f \wedge sfin\ w)) \frown wnext\ (\Box (init\ (\neg w))))$ 
using 14 15 16 by fastforce
have 28:  $\vdash (schopstar\ ((init\ w) \sqcap f \wedge sfin\ w)) \frown ((init\ w) \sqcap empty) =$ 
       $(init\ (\neg w)) \mathcal{U} ((init\ w) \wedge (schopstar\ (((init\ w) \sqcap f) \wedge sfin\ w)) \frown wnext\ (\Box (init\ (\neg w))))$ 
using 27 21 22 23
by (metis integ-reflection)
show ?thesis
by (metis 24 25 26 28 integ-reflection)
qed

```

end

11 First Order Finite and Infinite ITL theorems

```

theory FOTheorems
imports
  SChopTheorems
begin

```

We give the proofs of a list of first order (in)finite ITL theorems.

```

lemma EExI-unl:
   $w \models f\ x \implies w \models (\exists \exists\ x. f\ x)$ 
by (meson exist-state-d-def)

```

```

lemma EExNoDep:
   $\vdash (\exists \exists\ x. g) = g$ 
proof –
  have 1:  $\vdash g \longrightarrow (\exists \exists\ x. g)$  by (meson EExI)
  have 2:  $\bigwedge x. \vdash g \longrightarrow g$  by simp
  have 3:  $\vdash (\exists \exists\ x. g) \longrightarrow g$  using 2 by (meson EExE)
  from 1 3 show ?thesis using int-iffI by blast
qed

```

```

lemma AAxNoDep:
   $\vdash (\forall \forall\ x. g) = g$ 
using EExNoDep[of LIFT( $\neg$   $g$ )] AAxDef EExE EExI
by (simp add: exist-state-d-def forall-state-d-def intI)

```

```

lemma EExEqvRule:
  assumes  $\bigwedge x. \vdash f\ x = g\ x$ 

```

shows $\vdash (\exists \exists x. f x) = (\exists \exists x. g x)$
by (*metis EExE EExI assms int-iffD1 int-iffD2 int-iffI lift-imp-trans*)

lemma *AAxImpEEx*:
 $\vdash (\forall \forall x. f x) \longrightarrow (\exists \exists x. f x)$
by (*simp add: exist-state-d-def forall-state-d-def intI*)

lemma *EExImpRule*:
assumes $\vdash f x \longrightarrow g x$
shows $\vdash (\exists \exists x. f x \longrightarrow g x)$
using *assms* **by** (*meson MP EExI*)

lemma *EExImpRuleDist*:
assumes $\vdash f x \longrightarrow g x$
shows $\vdash (\forall \forall x. f x) \longrightarrow (\exists \exists x. g x)$
proof –
have 1: $\vdash (f x) \longrightarrow (\exists \exists x. g x)$ **using** *EExI assms lift-imp-trans* **by** *blast*
have 2: $\vdash \neg(f x) \vee (\exists \exists x. g x)$ **using** 1 **by** *auto*
have 3: $\vdash \neg(f x) \longrightarrow (\exists \exists x. \neg(f x))$ **by** (*meson EExI*)
have 4: $\vdash (\exists \exists x. \neg(f x)) = (\neg(\forall \forall x. f x))$ **using** *AAxDef* **by** *fastforce*
from 2 3 4 **show** *?thesis* **by** *fastforce*
qed

lemma *EExImpNoDepDist*:
assumes $\vdash f \longrightarrow g x$
shows $\vdash f \longrightarrow (\exists \exists x. g x)$
using *assms* **by** (*metis EExI lift-imp-trans*)

lemma *EExOrDist-1*:
 $\vdash (\exists \exists x. h x) \longrightarrow (\exists \exists x. (f x) \vee (h x))$
proof –
have 1: $\bigwedge x. \vdash h x \longrightarrow f x \vee h x$ **by** (*simp add: Valid-def*)
have 2: $\bigwedge x. \vdash f x \vee h x \longrightarrow (\exists \exists x. (f x) \vee (h x))$ **by** (*meson EExI*)
have 3: $\bigwedge x. \vdash h x \longrightarrow (\exists \exists x. (f x) \vee (h x))$ **using** 1 2 **by** (*meson lift-imp-trans*)
from 3 **show** *?thesis* **using** *EExE* **by** *blast*
qed

lemma *EExOrDist-2*:
 $\vdash (\exists \exists x. f x) \longrightarrow (\exists \exists x. (f x) \vee (h x))$
proof –
have 1: $\bigwedge x. \vdash f x \longrightarrow f x \vee h x$ **by** (*simp add: Valid-def*)
have 2: $\bigwedge x. \vdash f x \vee h x \longrightarrow (\exists \exists x. (f x) \vee (h x))$ **by** (*meson EExI*)
have 3: $\bigwedge x. \vdash f x \longrightarrow (\exists \exists x. (f x) \vee (h x))$ **using** 1 2 **by** (*meson lift-imp-trans*)
from 3 **show** *?thesis* **using** *EExE* **by** *blast*
qed

lemma *EExOrDist-3*:
 $\vdash (\exists \exists x. f x) \vee (\exists \exists x. h x) \longrightarrow (\exists \exists x. (f x) \vee (h x))$
using *EExOrDist-2 EExOrDist-1* **by** *fastforce*

lemma *EEExOrDist-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: EEExI-unl intI*)

from 1 **show** *?thesis* **by** (*simp add: EEExE*)

qed

lemma *EEExOrDist*:

$\vdash ((\exists \exists x. f x) \vee (\exists \exists x. h x)) = (\exists \exists x. (f x) \vee (h x))$

using *EEExOrDist-3 EEExOrDist-4* **by** *fastforce*

lemma *EEExOrImport-1*:

$\vdash g \longrightarrow (\exists \exists x. g \vee (f x))$

by (*simp add: EEExI-unl Valid-def*)

lemma *EEExOrImport-2*:

$\vdash (\exists \exists x. f x) \longrightarrow (\exists \exists x. g \vee (f x))$

by (*simp add: EEExOrDist-1*)

lemma *EEExOrImport-3*:

$\vdash (g \vee (\exists \exists x. f x)) \longrightarrow (\exists \exists x. g \vee (f x))$

using *EEExOrImport-1 EEExOrImport-2* **by** *fastforce*

lemma *EEExOrImport-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 EEExI int-iffD2 int-simps(27) Prop04 Prop08*)

from 1 **show** *?thesis* **by** (*simp add: EEExE*)

qed

lemma *EEExOrImport*:

$\vdash (g \vee (\exists \exists x. f x)) = (\exists \exists x. g \vee f x)$

by (*metis EEExOrImport-3 EEExOrImport-4 int-iffI*)

lemma *EEExAndImport-1*:

$\vdash g \wedge (\exists \exists x. f x) \longrightarrow (\exists \exists x. g \wedge f x)$

proof –

have 1: $\vdash (g \wedge (\exists \exists x. f x) \longrightarrow (\exists \exists x. g \wedge f x)) = ((\exists \exists x. f x) \longrightarrow (g \longrightarrow (\exists \exists x. g \wedge f x)))$

by *fastforce*

have 2: $\bigwedge x. \vdash f x \longrightarrow (g \longrightarrow (\exists \exists x. g \wedge f x))$

using *EEExI[of $\lambda x. LIFT(g \wedge f x)$]* **by** *fastforce*

hence 3: $\vdash (\exists \exists x. f x) \longrightarrow (g \longrightarrow (\exists \exists x. g \wedge f x))$

by (*simp add: EEExE*)

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

qed

lemma *EEExAndImport-2*:

$\vdash (\exists \exists x. g \wedge f x) \longrightarrow g \wedge (\exists \exists x. f x)$

proof –

have 1: $\bigwedge x. \vdash g \wedge f x \longrightarrow g \wedge (\exists \exists x. f x)$
by (*metis EExI int-iffD2 lift-and-com lift-imp-trans Prop12*)
from 1 **show** ?thesis **by** (*simp add: EExE*)
qed

lemma EExAndImport:
 $\vdash (g \wedge (\exists \exists x. f x)) = (\exists \exists x. g \wedge f x)$
by (*simp add: EExAndImport-1 EExAndImport-2 int-iffI*)

lemma EExAndDist:
assumes $\vdash f x \wedge g x$
shows $\vdash (\exists \exists x. f x) \wedge (\exists \exists x. g x)$
proof –
have 1: $\vdash f x$ **using** *assms* **by** *fastforce*
have 2: $\vdash g x$ **using** *assms* **by** *fastforce*
have 3: $\vdash (\exists \exists x. f x)$ **using** 1 **by** (*meson EExI MP*)
have 4: $\vdash (\exists \exists x. g x)$ **using** 2 **by** (*meson EExI MP*)
from 3 4 **show** ?thesis **by** *fastforce*
qed

lemma EExAndNoDepDist:
assumes $\vdash f \wedge g x$
shows $\vdash f \wedge (\exists \exists x. g x)$
proof –
have 1: $\vdash f$ **using** *assms* **by** *fastforce*
have 2: $\vdash g x$ **using** *assms* **by** *fastforce*
have 3: $\vdash (\exists \exists x. g x)$ **using** 2 **by** (*meson EExI MP*)
from 1 3 **show** ?thesis **by** *fastforce*
qed

lemma Spec:
 $\vdash (\forall \forall x. f x) \longrightarrow f x$
proof –
have 1: $\vdash \neg(f x) \longrightarrow (\exists \exists x. \neg(f x))$ **by** (*meson EExI*)
have 2: $\vdash \neg(\exists \exists x. \neg(f x)) \longrightarrow f x$ **using** 1 **by** *auto*
from 2 **show** ?thesis **using** *AAxDef* **by** *fastforce*
qed

lemma AAxE:
assumes $\vdash (\forall \forall x. f x)$
 $\vdash f x \longrightarrow g$
shows $\vdash g$
using *MP Spec assms(1) assms(2)* **by** *blast*

lemma AAxI:
assumes $\bigwedge x. \vdash f x$
shows $\vdash (\forall \forall x. f x)$
using *assms* **by** (*simp add: Valid-def exist-state-d-def forall-state-d-def*)

lemma AAxEqvRule:

assumes $\bigwedge x. \vdash f x = g x$
shows $\vdash (\forall \forall x. f x) = (\forall \forall x. g x)$
unfolding *forall-state-d-def* **using** *assms EExEqvRule*[*of* $\lambda x. (LIFT(\neg f x)) \lambda x. (LIFT(\neg g x))$]
by *fastforce*

lemma *AAxAndDist*:

$\vdash (\forall \forall x. (f x) \wedge (g x)) = ((\forall \forall x. f x) \wedge (\forall \forall x. g x))$

proof –

have 1: $\bigwedge x. \vdash (\neg(f x) \vee \neg(g x)) = (\neg((f x) \wedge (g x)))$

by *auto*

have 2: $\vdash (\exists \exists x. \neg(f x) \vee \neg(g x)) = (\exists \exists x. \neg((f x) \wedge (g x)))$

using 1 **by** (*simp add: EExEqvRule*)

have 3: $\vdash (\exists \exists fa. \neg(f fa \wedge g fa)) = (\exists \exists fa. \neg f fa \vee \neg g fa)$

using 2 **by** *auto*

have 4: $\vdash (\neg(\neg(\exists \exists fa. \neg f fa) \wedge \neg(\exists \exists f. \neg g f))) = ((\exists \exists fa. \neg f fa) \vee (\exists \exists f. \neg g f))$

by *auto*

have $\vdash ((\exists \exists fa. \neg f fa) \vee (\exists \exists f. \neg g f)) = (\exists \exists fa. \neg f fa \vee \neg g fa)$

by (*simp add: EExOrDist inteq-reflection*)

then have 5: $\vdash (\neg(\exists \exists fa. \neg f fa) \wedge \neg(\exists \exists f. \neg g f)) = (\neg(\exists \exists fa. \neg f fa \vee \neg g fa))$

by *auto*

show *?thesis* **using** 3 5 **unfolding** *forall-state-d-def* **by** *fastforce*

qed

lemma *AAxAndImport*:

$\vdash (g \wedge (\forall \forall x. f x)) = (\forall \forall x. g \wedge f x)$

proof –

have 1: $\vdash (\neg g \vee (\exists \exists x. \neg(f x))) = (\exists \exists x. \neg g \vee \neg(f x))$

by (*simp add: EExOrImport*)

have 2: $\vdash (\neg(\exists \exists x. \neg(f x))) = (\neg(\forall \forall x. f x))$

using *AAxDef* **by** *fastforce*

have 3: $\vdash (\neg g \vee (\exists \exists x. \neg(f x))) = (\neg(g \wedge (\forall \forall x. f x)))$

using 2 **by** *fastforce*

have 4: $\bigwedge x. \vdash (\neg g \vee \neg(f x)) = (\neg(g \wedge f x))$

by *auto*

have 5: $\vdash (\exists \exists x. \neg g \vee \neg(f x)) = (\exists \exists x. \neg(g \wedge f x))$

using 4 **by** (*simp add: EExEqvRule*)

have 6: $\vdash (\exists \exists x. \neg(g \wedge f x)) = (\neg(\forall \forall x. g \wedge f x))$

using *AAxDef* **by** *fastforce*

have 7: $\vdash (\neg(g \wedge (\forall \forall x. f x))) = (\neg(\forall \forall x. g \wedge f x))$

by (*metis 1 3 5 6 inteq-reflection*)

from 7 **show** *?thesis* **by** *fastforce*

qed

lemma *AAxOrImport*:

$\vdash (g \vee (\forall \forall x. f x)) = (\forall \forall x. g \vee f x)$

proof –

have 1: $\vdash (\neg g \wedge (\exists \exists x. \neg(f x))) = (\exists \exists x. \neg g \wedge \neg(f x))$ **by** (*simp add: EExAndImport*)

have 2: $\vdash (\exists \exists x. \neg(f x)) = (\neg(\forall \forall x. f x))$ **using** *AAxDef* **by** *fastforce*

have 3: $\vdash (\neg g \wedge (\exists \exists x. \neg(f x))) = (\neg(g \vee (\forall \forall x. f x)))$ **using** 2 **by** *fastforce*

have 4: $\bigwedge x. \vdash (\neg g \wedge \neg(f x)) = (\neg(g \vee f x))$ **by** *auto*
have 5: $\vdash (\exists \exists x. \neg g \wedge \neg(f x)) = (\exists \exists x. \neg(g \vee f x))$ **using** 4 **by** (*simp add: EExEqvRule*)
have 6: $\vdash (\exists \exists x. \neg(g \vee f x)) = (\neg(\forall \forall x. g \vee f x))$ **using** *AAxDef* **by** *fastforce*
have 7: $\vdash (\neg(g \vee (\forall \forall x. f x))) = (\neg(\forall \forall x. g \vee f x))$ **by** (*metis 1 3 5 6 inteq-reflection*)
from 7 **show** *?thesis* **by** *auto*
qed

lemma *EExImpChopRule*:

assumes $\vdash f x \longrightarrow g x$
shows $\vdash (\exists \exists x. h;(f x) \longrightarrow h;(g x))$
using *RightChopImpChop[of f x g x h]*
EExImpRule[of $\lambda x. LIFT(h;(f x))$ 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 itl-defs*)

lemma *CSPowerChop*:
 $\vdash (f^*) = (\exists n. \text{power } (f \wedge \text{more}) n);(\text{empty} \vee (f \wedge \text{more}) \wedge \text{inf})$
by (*simp add: chopstar-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

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. A Hierarchical Completeness Proof for Propositional Interval Temporal Logic with Finite Time. *Journal of Applied Non-Classical Logics*, 14(1–2):55–104, 2004.
- [6] B. Moszkowski and D. Guelev. An application of temporal projection to interleaving concurrency. *Formal Aspects of Computing*, 29(4):705–750, July 2017.
- [7] B. C. Moszkowski. Imperative reasoning in interval temporal logic. Technical report, University of Newcastle upon Tyne, 1996.
- [8] B. C. Moszkowski. A Complete Axiom System for Propositional Interval Temporal Logic with Infinite Time. *Logical Methods in Computer Science Journal*, 8(3), 2012.

- [9] 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.
- [10] 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.