

# An encoding of Interval Temporal Logic in Isabelle/HOL

Antonio Cau

Ben Moszkowski

David Smallwood

February 28, 2021

## Abstract

These Isabelle theories introduce the semantics and syntax of Finite Interval Temporal Logic (ITL). The ITL proof system, as introduced in [5, 9], has been encoded and its soundness has been checked. The encoding is shallow using the Intensional Logic technique of [3]. An extensive library of Finite ITL theorems, taken from [8], has been checked.

We also present a theory of first occurrence and use it to derive an algebra of Runtime verification (RV) monitors. Furthermore we provide examples of using quantification over both static (rigid) and state (flexible) variables and several RV examples.

Furthermore the notion of executability of ITL formulae has been formalised.

## Contents

<b>1</b>	<b>Finite Intervals</b>	<b>6</b>
1.1	Definitions . . . . .	6
1.2	Lemmas . . . . .	8
1.2.1	Shifting indexes to zero . . . . .	8
1.2.2	Interval Length . . . . .	8
1.2.3	inth . . . . .	9
1.2.4	prefix, suffix and sub . . . . .	10
1.2.5	iapp . . . . .	22
1.2.6	Reverse . . . . .	28
1.2.7	Induction rule . . . . .	31
1.2.8	Imap . . . . .	32
1.2.9	index sequence . . . . .	36
1.2.10	upt . . . . .	44
1.2.11	Set . . . . .	46
<b>2</b>	<b>Finite ITL Semantics</b>	<b>55</b>
2.1	Types of Formulas . . . . .	55
2.2	Semantics of ITL . . . . .	56
2.3	Abbreviations . . . . .	57
2.4	Properties of Operators . . . . .	63
2.5	Soundness of Finite ITL Axioms . . . . .	67
2.5.1	ChopAssoc . . . . .	67
2.5.2	OrChopImp . . . . .	69
2.5.3	ChopOrImp . . . . .	69

2.5.4	EmptyChop	69
2.5.5	ChopEmpty	70
2.5.6	StateImpBi	70
2.5.7	NextImpNotNextNot	70
2.5.8	BiBoxChopImpChop	70
2.5.9	BoxInduct	70
2.5.10	ChopStarEqv	71
2.6	Quantification over State (Flexible) Variables	72
2.7	Temporal Quantifiers	72
<b>3</b>	<b>Fuse operator</b>	<b>73</b>
3.1	Definitions	73
3.2	Lemmas	73
<b>4</b>	<b>Finite ITL: Axioms and Rules</b>	<b>87</b>
4.1	Rules	88
4.2	Axioms	88
4.3	Additional Lemmas	89
4.4	Quantification	90
4.5	Lemmas about <i>current-val</i>	90
4.6	Lemmas about <i>next-val</i>	91
4.7	Lemmas about <i>fin-val</i>	91
4.8	Lemmas about <i>penult-val</i>	92
4.9	Basic temporal variables properties	92
<b>5</b>	<b>Finite ITL theorems</b>	<b>93</b>
5.1	Propositional reasoning	93
5.2	State formulas	94
5.3	Basic Theorems	95
5.4	Further Properties Di and Bi	107
5.5	Properties of Da and Ba	113
5.6	Properties of Fin	121
5.7	Properties of Chopstar and Chopplus	142
5.8	Properties of While	165
5.9	Properties of Halt	170
5.10	Properties of Groups of chops	175
<b>6</b>	<b>First Order Finite ITL theorems</b>	<b>175</b>
<b>7</b>	<b>Time Reversal</b>	<b>182</b>
7.1	Definition	182
7.2	Time reversal Rules	182
7.3	Properties of Time Reversal	185
<b>8</b>	<b>Projection operator</b>	<b>194</b>
8.1	Definitions	194
8.2	Lemmas	195
8.2.1	filt Lemmas	195

8.2.2	powerinterval lemmas	202
8.2.3	cpl lemmas	207
8.2.4	lcpl lemmas	211
8.2.5	lsum lemmas	219
8.3	Soundness of Projection Axioms	241
8.3.1	PJ1	241
8.3.2	PJ2	241
8.3.3	PJ3	241
8.3.4	PJ4	242
8.3.5	PJ5	247
8.3.6	PJ6	247
8.3.7	PJ7	250
8.3.8	PJ8	266
8.3.9	PJ9	267
8.4	Axioms	267
8.5	Time Reversal	268
8.6	Theorems	273
8.6.1	Projection	273
8.6.2	dp and bp	277
<b>9</b>	<b>The First Occurrence Operator in finite ITL</b>	<b>283</b>
9.1	Definitions	284
9.1.1	Definitions Strict Initial and Final	284
9.1.2	Definition First and Last Operators	285
9.2	First and Time Reversal	285
9.3	Semantic Theorems	288
9.3.1	Semantics First and Last Operators	288
9.3.2	Various Semantic Lemmas	290
9.4	Theorems	292
9.4.1	Fixed length intervals	292
9.4.2	Additional ITL theorems	297
9.4.3	Strict initial intervals	306
9.4.4	First occurrence	315
<b>10</b>	<b>Monitors</b>	<b>341</b>
10.1	Syntax	341
10.2	Derived Monitors	343
10.3	Monitor Laws	346
<b>11</b>	<b>Finite ITL Examples</b>	<b>367</b>
11.1	Example 1	367
11.2	Example 2	367
11.3	Example 3	369
11.4	Example 4	370
11.5	Example 5	371
<b>12</b>	<b>Monitor Example</b>	<b>371</b>

<b>13 Filter on Intervals</b>	<b>377</b>
13.1 Definitions . . . . .	377
13.2 Lemmas . . . . .	378
13.2.1 opfx and sopfx . . . . .	378
13.2.2 idistinct and iremdups . . . . .	387
13.2.3 prefixes and suffixes . . . . .	391
13.2.4 ifilter and nfilter . . . . .	397
<b>14 Until and Since operator</b>	<b>458</b>
14.1 Definitions . . . . .	458
14.2 Axioms . . . . .	460
14.2.1 NextUntil . . . . .	460
14.2.2 UntilNextUntil . . . . .	461
14.2.3 NotUntilFalse . . . . .	463
14.2.4 UntilOrDist . . . . .	463
14.2.5 UntilRightDistOr . . . . .	463
14.2.6 UntilLeftDistAnd . . . . .	463
14.2.7 UntilAndDist . . . . .	463
14.2.8 untilNotImp . . . . .	464
14.2.9 UntilUntil . . . . .	464
14.2.10 UntilRightor . . . . .	465
14.2.11 UntilRightAnd . . . . .	465
14.2.12 SincePrevSince . . . . .	466
14.2.13 RevUntil . . . . .	467
14.2.14 DiamondEqvTrueUntil . . . . .	469
14.2.15 TrueUntillImpNotUntil . . . . .	469
14.2.16 WaitNotDistUntil . . . . .	469
14.2.17 UntillInduction . . . . .	470
14.3 Theorems . . . . .	472
<b>15 Pi operator</b>	<b>498</b>
15.1 Definitions . . . . .	498
15.2 Time reversal . . . . .	499
15.3 Semantic Lemmas . . . . .	500
15.4 Soundness of Axioms . . . . .	506
15.4.1 PiK . . . . .	506
15.4.2 PiDc . . . . .	507
15.4.3 PiN . . . . .	507
15.4.4 PiTrueEqvDiamond . . . . .	507
15.4.5 PiOr . . . . .	507
15.4.6 UPiFalseEqvBoxNot: . . . . .	507
15.4.7 BoxEqvImpPiEqv . . . . .	507
15.4.8 PiDiamondImpDiamond . . . . .	509
15.4.9 PiAssoc . . . . .	509
15.4.10 PiNotEqvDiamondAndNotPi . . . . .	514
15.4.11 PiChopDist . . . . .	514
15.4.12 PiProp . . . . .	520

15.4.13	PiNext . . . . .	522
15.4.14	PiUntil . . . . .	525
15.4.15	PiChopstar . . . . .	534
15.4.16	TruePiEqv . . . . .	540
15.4.17	BoxImpEqvPi . . . . .	540
15.4.18	PiEqvDiamondUPi . . . . .	540
15.4.19	PiEqvUntilPi . . . . .	540
15.4.20	UPiEqvBoxOrPi . . . . .	540
15.5	Theorems . . . . .	540
<b>16</b>	<b>Interval Temporal Algebra</b>	<b>548</b>
16.1	Definition of Set of intervals and Operations on them . . . . .	549
16.2	Simplification Lemmas . . . . .	550
16.3	Algebraic Laws . . . . .	553
16.3.1	Commutative Additive Monoid . . . . .	553
16.3.2	Boolean algebra . . . . .	553
16.3.3	multiplicative monoid . . . . .	553
16.3.4	Subsumption order . . . . .	555
16.3.5	Helper lemmas . . . . .	555
16.3.6	Kleene Algebra . . . . .	558
16.3.7	ITL specific Laws . . . . .	559
16.4	Derived Laws . . . . .	560
16.4.1	Helper Lemmas . . . . .	560
16.4.2	ITL Axioms derived . . . . .	565
16.5	Extra Laws . . . . .	567
16.5.1	Boolean Laws . . . . .	567
16.5.2	Chop . . . . .	569
16.5.3	Next . . . . .	570
16.5.4	SInit . . . . .	574
16.5.5	SStar . . . . .	577
16.5.6	Box and Diamond . . . . .	580
16.6	Time Reversal . . . . .	584
16.6.1	Time Reversal Axioms . . . . .	584
16.6.2	Time Reversal Laws . . . . .	586
16.7	Link between Set of Intervals and ITL . . . . .	587
<b>17</b>	<b>Executability of ITL formulae</b>	<b>591</b>
17.1	Forward Executability . . . . .	592
17.1.1	Common Prefix Value Trace Definitions . . . . .	592
17.1.2	Semantic lemmas . . . . .	592
17.1.3	Common prefix value trace theorems . . . . .	595
17.1.4	Common prefix value trace and len and assignment . . . . .	597
17.1.5	Tempura . . . . .	598
17.1.6	Basic theorems . . . . .	598
17.1.7	Forward Executability Theorems . . . . .	606
17.2	Backward Executability . . . . .	610
17.2.1	Common Suffix Value Trace Definitions . . . . .	610

17.2.2	Common suffix value trace theorems	610
17.2.3	Time reversal theorems	612
17.2.4	Common suffix value trace and len and assignment	612
17.2.5	Basic theorems	616
17.2.6	Backward Executability theorems	618
17.3	Reversing bad computations	622

# 1 Finite Intervals

**theory** *Interval*

**imports**

*Main*

**begin**

An interval is a finite sequence of elements of a particular type. Intervals are similar to list in Isabelle/HOL, the difference is that intervals have instead of nil a single element at the end. So an interval of length zero is a single element whereas for Isabelle's list we have that an empty list is nil (no element present).

The usual operations on intervals are defined: *ilen*, *prefix*, *suffix*, *sub*, *inth*, *ifirst*, *ilast*, *iapp* and *irev*.

We also introduce *index-sequence* which is a sequence of chop (fuse) points. This sequence is used in the old definition of chopstar which is an existential quantification over this sequence. The type *index-sequence* is again of type interval but the elements are natural numbers. Two functions *shift* and *shiftn* are introduced that are used to add (shift) and subtract a natural number of each element in the sequence of chop (fuse) points. The operation *upt* to produce a sequence of consecutive chop points between two natural numbers.

## 1.1 Definitions

**datatype** (*iset*: 'a) *interval* =  
     *INil* 'a ([*-*])  
   | *ICons* 'a 'a *interval* (**infixr**  $\odot$  65)

**for**

*map*: *imap*

*rel*: *interval-all2*

*pred*: *interval-all*

**type-synonym** *index* = *nat interval*

**syntax**

— interval Enumeration

*-interval* :: *args* => 'a *interval*    (*(((-))*)

**translations**

$\langle x, xs \rangle == x \odot \langle xs \rangle$

$\langle x \rangle == [x]$

**primrec** (*nonexhaustive*) *ilen* :: 'a interval  $\Rightarrow$  nat **where**

*ilen*  $\langle x \rangle = 0$   
| *ilen* ( $x \odot xs$ ) = 1 + (*ilen* *xs*)

**primrec** (*nonexhaustive*) *inth* :: 'a interval  $\Rightarrow$  nat  $\Rightarrow$  'a **where**

*inth*  $\langle x \rangle$  *n* = *x*  
| *inth* ( $x \odot xs$ ) *n* = (case *n* of 0  $\Rightarrow$  *x* | Suc *k*  $\Rightarrow$  *inth* *xs* *k*)

**primrec** *prefix*:: nat  $\Rightarrow$  'a interval  $\Rightarrow$  'a interval **where**

*prefix* *n*  $\langle x \rangle = \langle x \rangle$   
| *prefix* *n* ( $x \odot xs$ ) = (case *n* of 0  $\Rightarrow$   $\langle x \rangle$  | Suc *m*  $\Rightarrow$   $x \odot (\text{prefix } m \text{ } xs)$ )

**primrec** *suffix*:: nat  $\Rightarrow$  'a interval  $\Rightarrow$  'a interval **where**

*suffix* *n*  $\langle x \rangle = \langle x \rangle$   
| *suffix* *n* ( $x \odot xs$ ) = (case *n* of 0  $\Rightarrow$  ( $x \odot xs$ ) | Suc *m*  $\Rightarrow$  *suffix* *m* *xs*)

**definition** *sub*:: nat  $\Rightarrow$  nat  $\Rightarrow$  'a interval  $\Rightarrow$  'a interval

**where**

*sub* *n* *k* *xs* = *prefix* (*k* − *n*) (*suffix* *n* *xs*)

**abbreviation** *ifirst* :: 'a interval  $\Rightarrow$  'a **where**

*ifirst* *xs*  $\equiv$  (*inth* *xs* 0)

**abbreviation** *ilast* :: 'a interval  $\Rightarrow$  'a **where**

*ilast* *xs*  $\equiv$  (*inth* *xs* (*ilen* *xs*))

**primrec** *iapp* :: 'a interval  $\Rightarrow$  'a interval  $\Rightarrow$  'a interval (**infixr**  $\ominus$  65) **where**

*iapp-INil*:  $\langle x \rangle \ominus ys = x \odot ys$  |

*iapp-ICons*: ( $x \odot xs$ )  $\ominus ys = x \odot (xs \ominus ys)$

**primrec** *irev* :: 'a interval  $\Rightarrow$  'a interval **where**

*irev*  $\langle x \rangle = \langle x \rangle$

| *irev* ( $x \odot xs$ ) = (*irev* *xs*)  $\ominus \langle x \rangle$

**definition** *index-sequence* :: nat  $\Rightarrow$  index  $\Rightarrow$  bool **where**

*index-sequence* *x* *idx*  $\equiv$  (*inth* *idx* 0 = *x*)  $\wedge$  ( $\forall$  *n*. *n* < *ilen* *idx*  $\longrightarrow$  *inth* *idx* *n* < *inth* *idx* (Suc *n*))

**definition** *shift* :: nat  $\Rightarrow$  nat  $\Rightarrow$  nat **where**

*shift* *k* = ( $\lambda$  *x*. *x* + *k*)

**definition** *shiftn* :: nat  $\Rightarrow$  nat  $\Rightarrow$  nat **where**

*shiftn* *k* = ( $\lambda$  *x*. *x* − *k*)

**primrec** *upt* :: nat  $\Rightarrow$  nat  $\Rightarrow$  nat interval ((1[.. $\leq$ ]/- $\rfloor$ ))

**where**

*upt-0* : [*i*.. $\leq$ 0] =  $\langle 0 \rangle$

| *upt-Suc*: [*i*.. $\leq$ (Suc *j*)] = (if *i*  $\leq$  *j* then [*i*.. $\leq$ *j*]  $\ominus \langle$ (Suc *j*) $\rangle$  else  $\langle$ (Suc *j*) $\rangle$  )

## 1.2 Lemmas

Basic lemmas are introduced for each of the above operations on intervals.

### 1.2.1 Shifting indexes to zero

**lemma** *interval-shift-index-to-zero-a*:

**shows**  $(\forall (i::nat). a \leq i \wedge i < a+b \longrightarrow f (g1(i)) (g2(Suc i))) =$   
 $(\forall i. 0 \leq i \wedge i < b \longrightarrow f (g1(i+a)) (g2((Suc i)+a)))$

**by** *simp (metis add.commute le-Suc-ex le-add2 nat-add-left-cancel-less)*

**lemma** *interval-shift-index-to-zero-b*:

**shows**  $(\forall (i::nat). a \leq i \wedge i < a+b \longrightarrow f (g(i-a)) (g2((Suc i)-a))) =$   
 $(\forall i. 0 \leq i \wedge i < b \longrightarrow f (g(i)) (g2(Suc i)))$  (**is**  $?L=?R$ )

**proof** –

**have** 1:  $?L \Longrightarrow ?R$

**by** (*metis add-Suc add-diff-cancel-left' add-diff-cancel-right' le-add2 less-diff-conv*)

**have** 2:  $?R \Longrightarrow ?L$

**by** (*metis Nat.add-diff-assoc add-diff-cancel-left' le0 le-add-diff-inverse2 less-diff-conv plus-1-eq-Suc*)

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

**qed**

### 1.2.2 Interval Length

**lemma** *ilen-gr-zero* [*simp*]:

$ilen\ xs \geq 0$

**by** *auto*

**lemma** *ilen-ICons* [*simp*]:

$(ilen\ (x \odot xs)) = (ilen\ xs) + 1$

**by** *simp*

**lemma** *ilen-ICons-1* :

$ilen\ l > 0 = (\exists\ x\ ls. l = x \odot ls)$

**by** (*induct l*) *simp-all*

**lemma** *ilen-imap* [*simp*]:

$ilen\ (imap\ f\ xs) = ilen\ xs$

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

**lemma** *ilen-iapp* [*simp*]:

$ilen\ (xs \ominus ys) = (ilen\ xs) + (ilen\ ys) + 1$

**by** (*induct xs arbitrary: ys*) *simp-all*

**lemma** *irev-ilen* [*simp*]:

$ilen\ (irev\ xs) = ilen\ xs$

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



### 1.2.3 inth

**lemma** *inth-zero* [simp]:

$$\text{inth } (x \odot xs) \ 0 = x$$

**by** *simp*

**lemma** *inth-Suc* [simp]:

$$\text{inth } (x \odot xs) \ (\text{Suc } n) = \text{inth } xs \ n$$

**by** *auto*

**lemma** *inth-last*:

$$\text{inth } (x \odot xs) \ (\text{ilen } (x \odot xs)) = \text{inth } xs \ (\text{ilen } xs)$$

**by** *simp*

**lemma** *inth-last-stutter*:

$$\text{inth } xs \ (\text{ilen } xs + i) = \text{inth } xs \ (\text{ilen } xs)$$

**proof** (*induction xs arbitrary:i*)

**case** (*INil x*)

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

**next**

**case** (*ICons x1a xs*)

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

**qed**

**lemma** *inth-ICons-a*:

**assumes**  $0 < i$

**shows**  $\text{inth}(x \odot xs) \ i = \text{inth } xs \ (i-1)$

**using** *assms* **by** (*metis Suc-diff-1 inth-Suc*)

**lemma** *inth-ICons-b*:

**shows**  $\text{inth}(x \odot xs) \ (i+1) = \text{inth } xs \ i$

**by** *simp*

**lemma** *inth-ICons*:

**assumes**  $0 < i$

**shows**  $\text{inth}(x \odot xs) \ i = \text{inth } xs \ (i-1) \wedge$   
 $\text{inth}(x \odot xs) \ (i+1) = \text{inth } xs \ i$

**by** (*meson assms inth-ICons-a inth-ICons-b*)

**lemma** *inth-zero-ifirst*:

$$\text{ifirst } xs = \text{inth } xs \ 0$$

**by** *simp*

**lemma** *inth-ilen-ilast*:

$$\text{ilast } xs = \text{inth } xs \ (\text{ilen } xs)$$

**by** *simp*

**lemma** *INil-ilen* :

$$(xs = \langle x \rangle) \longleftrightarrow \text{ilen } xs = 0 \wedge \text{inth } xs \ 0 = x$$

**by** (*cases xs*) *auto*

**lemma** *interval-eq-inth-eq* :  
 $(xs = ys) = (ilen\ xs = ilen\ ys \wedge (\forall\ i \leq ilen\ xs.\ inth\ xs\ i = inth\ ys\ i))$   
**proof**  
*(induct xs arbitrary: ys)*  
**case** (*INil* *x*)  
**then show** ?*case* **by** (*metis INil-ilen le-numeral-extra*(3))  
**next**  
**case** (*ICons* *x1a xs*)  
**then show** ?*case*  
**proof** (*cases ys*)  
**case** (*INil* *x1*)  
**then show** ?*thesis* **by** *simp*  
**next**  
**case** (*ICons* *x21 x22*)  
**then show** ?*thesis*  
**using** *ICons.hyps* **by** *fastforce*  
**qed**  
**qed**

**lemma** *inth-imap* :  
 $inth\ (imap\ f\ xs)\ i = f\ (inth\ xs\ i)$   
**proof**  
*(induct xs arbitrary: i)*  
**case** (*INil* *x*)  
**then show** ?*case* **by** *simp*  
**next**  
**case** (*ICons* *x1a xs*)  
**then show** ?*case*  
**proof** (*cases i*)  
**case** 0  
**then show** ?*thesis* **by** *simp*  
**next**  
**case** (*Suc* *nat*)  
**then show** ?*thesis* **using** *ICons.hyps* **by** *simp*  
**qed**  
**qed**

#### 1.2.4 prefix, suffix and sub

**lemma** *prefix-INil* [*simp*]:  
 $prefix\ m\ \langle x \rangle = \langle x \rangle$   
**by** *simp*

**lemma** *prefix-suc* [*simp*]:  
 $prefix\ (Suc\ m)\ (x \odot xs) = x \odot (prefix\ m\ xs)$   
**by** *auto*

**lemma** *prefix-zero* [*simp*]:  
 $prefix\ 0\ (x \odot xs) = \langle x \rangle$

**by** *auto*

**lemma** *prefix-zero-ifirst* [*simp*]:

$\text{prefix } 0 \text{ } xs = \langle \text{inth } xs \ 0 \rangle$

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

**lemma** *ifirst-prefix* [*simp*]:

**shows**  $\text{ifirst } (\text{prefix } i \text{ } xs) = \text{ifirst } xs$

**proof**

(*induct xs arbitrary: i*)

**case** (*INil x*)

**then show** *?case* **by** *auto*

**next**

**case** (*ICons x1a xs*)

**then show** *?case*

**proof** (*cases i*)

**case** *0*

**then show** *?thesis* **by** *auto*

**next**

**case** (*Suc nat*)

**then show** *?thesis* **using** *ICons.hyps* **by** *auto*

**qed**

**qed**

**lemma** *ilast-suffix* [*simp*]:

**shows**  $\text{ilast } (\text{suffix } i \text{ } xs) = \text{ilast } xs$

**proof**

(*induct xs arbitrary: i*)

**case** (*INil x*)

**then show** *?case* **by** *auto*

**next**

**case** (*ICons x1a xs*)

**then show** *?case*

**proof** (*cases i*)

**case** *0*

**then show** *?thesis* **by** *auto*

**next**

**case** (*Suc nat*)

**then show** *?thesis* **using** *ICons.hyps* **by** *auto*

**qed**

**qed**

**lemma** *prefix-ilen* [*simp*]:

$(\text{prefix } (\text{ilen } xs) \text{ } xs) = xs$

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

**lemma** *prefix-ilen-gr-1* [*simp*]:

$(\text{prefix } ((\text{ilen } xs) + i) \text{ } xs) = xs$

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

**lemma** *ilen-prefix-ICons* [*simp*]:

$ilen(\text{prefix } (Suc\ i) (x \odot xs)) = 1 + ilen(\text{prefix } i\ xs)$

**using** *ilen-ICons* **by** *auto*

**lemma** *prefix-ilen-code* [*code*]:

$ilen(\text{prefix } i\ xs) = (\text{if } i \leq ilen\ xs \text{ then } i \text{ else } ilen\ xs)$

**proof**

(*induct xs arbitrary: i*)

**case** (*INil x*)

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

**next**

**case** (*ICons x1a xs*)

**then show** ?*case*

**proof** (*cases i*)

**case** 0

**then show** ?*thesis* **by** *auto*

**next**

**case** (*Suc nat*)

**then show** ?*thesis* **using** *ICons.hyps* **by** *auto*

**qed**

**qed**

**lemma** *prefix-ilen-min* [*simp*]:

$ilen(\text{prefix } i\ xs) = \min i (ilen\ xs)$

**by** (*simp add: prefix-ilen-code min-def*)

**lemma** *prefix-ilen-good* [*simp*]:

**assumes**  $i \leq ilen\ xs$

**shows**  $(ilen(\text{prefix } i\ xs)) = i$

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

**lemma** *prefix-ilen-bad* :

**assumes**  $i > ilen\ xs$

**shows**  $ilen(\text{prefix } i\ xs) = ilen\ xs$

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

**lemma** *prefix-ilen-bound* :

**shows**  $ilen(\text{prefix } i\ xs) \leq ilen\ xs$

**by** *simp*

**lemma** *suffix-ilen-code* [*code*]:

$ilen(\text{suffix } i\ xs) = (\text{if } i \leq ilen\ xs \text{ then } (ilen\ xs) - i \text{ else } 0)$

**proof**

(*induct xs arbitrary: i*)

**case** (*INil x*)

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

**next**

**case** (*ICons x1a xs*)

```

then show ?case
proof (cases i)
case 0
then show ?thesis by auto
next
case (Suc nat)
then show ?thesis using ICons.hyps by auto
qed
qed

```

```

lemma suffix-ilen [simp]:
  ilen (suffix i xs) = (ilen xs) - i
by (simp add: suffix-ilen-code)

```

```

lemma suffix-ilen-good [simp]:
  assumes i ≤ ilen xs
  shows ilen (suffix i xs) = (ilen xs) - i
using assms by simp

```

```

lemma suffix-ilen-bad:
  assumes i > ilen xs
  shows ilen (suffix i xs) = 0
using assms by simp

```

```

lemma suffix-ilen-bound:
  ilen (suffix i xs) ≤ ilen xs
by simp

```

```

lemma inth-prefix [simp]:
  assumes k ≤ i
  shows inth (prefix i xs) k = inth xs k
using assms
proof
  (induct i arbitrary: xs k)
  case 0
  then show ?case
  proof (cases xs)
  case (INil x1)
  then show ?thesis by auto
  next
  case (ICons x21 x22)
  then show ?thesis using 0.prem by auto
  qed
  next
  case (Suc i)
  then show ?case
  proof (cases xs)
  case (INil x1)
  then show ?thesis by auto
  next

```

```

case (ICons x21 x22)
then show ?thesis
proof (cases k)
case 0
then show ?thesis by (simp add: ICons)
next
case (Suc nat)
then show ?thesis
using Suc.hyps Suc.prem1s ICons by auto
qed
qed
qed

```

```

lemma inth-suffix [simp]:
assumes  $k \leq \text{ilen } xs - i$ 
shows  $\text{inth } (\text{suffix } i \text{ } xs) \text{ } k = \text{inth } xs \text{ } (i+k)$ 
using assms
proof (induct xs arbitrary: i k)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
proof (cases i)
case 0
then show ?thesis by auto
next
case (Suc nat)
then show ?thesis
proof auto
assume a0:  $i = \text{Suc } nat$ 
show  $\text{inth } (\text{suffix } nat \text{ } xs) \text{ } k = \text{inth } xs \text{ } (nat + k)$ 
using a0 ICons.hyps ICons.prem1s by auto
qed
qed
qed

```

```

lemma suffix-prefix-help-1:
assumes  $ia+i \leq \text{ilen } xs$ 
 $k \leq ia$ 
shows  $\text{inth } (\text{prefix } ia \text{ } (\text{suffix } i \text{ } xs)) \text{ } k = \text{inth } (\text{suffix } i \text{ } (\text{prefix } (ia+i) \text{ } xs)) \text{ } k$ 
proof -
have 1:  $\text{inth } (\text{prefix } ia \text{ } (\text{suffix } i \text{ } xs)) \text{ } k = \text{inth } (\text{suffix } i \text{ } xs) \text{ } k$ 
using inth-prefix assms by metis
have 2:  $\text{inth } (\text{suffix } i \text{ } xs) \text{ } k = \text{inth } xs \text{ } (i+k)$ 
using inth-suffix assms by (simp add: add-le-imp-le-diff)
have 3:  $\text{inth } xs \text{ } (i+k) = \text{inth } (\text{prefix } (ia+i) \text{ } xs) \text{ } (i+k)$ 
using inth-prefix assms by simp
have 4:  $\text{inth } (\text{prefix } (ia+i) \text{ } xs) \text{ } (i+k) = \text{inth } (\text{suffix } i \text{ } (\text{prefix } (ia+i) \text{ } xs)) \text{ } k$ 
using inth-suffix assms by simp

```

from 1 2 3 4 show ?thesis by auto  
qed

lemma suffix-prefix-help-2:

assumes  $ia+i \leq \text{ilen } xs$

shows  $(\forall k \leq ia . \text{inth } (\text{prefix } ia \ (\text{suffix } i \ xs)) \ k = \text{inth } (\text{suffix } i \ (\text{prefix } (ia+i) \ xs)) \ k)$

using suffix-prefix-help-1 using assms by fastforce

lemma suffix-prefix-help-3:

assumes  $ia+i \leq \text{ilen } xs$

shows  $\text{ilen } (\text{prefix } ia \ (\text{suffix } i \ xs)) = \text{ilen } (\text{suffix } i \ (\text{prefix } (ia+i) \ xs))$

using assms prefix-ilen-good suffix-ilen-good by auto

lemma suffix-prefix-swap:

assumes  $ia+i \leq \text{ilen } xs$

shows  $\text{prefix } ia \ (\text{suffix } i \ xs) = \text{suffix } i \ (\text{prefix } (ia+i) \ xs)$

using assms using interval-eq-inth-eq by fastforce

lemma prefix-prefix-zero [simp]:

$\text{prefix } 0 \ (\text{prefix } 0 \ xs) = \text{prefix } 0 \ xs$

by (induct xs) simp-all

lemma pref-pref [simp]:

$(\text{prefix } i \ (\text{prefix } i \ xs)) = \text{prefix } i \ xs$

by (metis prefix-ilen prefix-ilen-gr-1 prefix-ilen-good less-imp-add-positive not-less)

lemma pref-pref-3 [simp]:

$(\text{prefix } i \ (\text{prefix } (i+k) \ xs)) = \text{prefix } i \ xs$

proof

(induct xs arbitrary: i k)

case (INil x)

then show ?case by simp

next

case (ICons x1a xs)

then show ?case

proof (cases i)

case 0

then show ?thesis

by (auto simp add: Nitpick.case-nat-unfold)

next

case (Suc nat)

then show ?thesis

by (simp add: ICons.hyps)

qed

qed

lemma pref-help:

assumes  $i \leq \text{ilen } (\text{prefix } (\text{ilen } xs - \text{Suc } 0) \ xs)$

shows  $(\text{prefix } i \ (\text{prefix } (\text{ilen } xs - \text{Suc } 0) \ xs)) = (\text{prefix } i \ xs)$

```

using assms
by (metis diff-le-self pref-pref-3 prefix-ilen-good
      ordered-cancel-comm-monoid-diff-class.add-diff-inverse)

lemma pref-pref-help:
  assumes ilen xs > 0
           ia < ilen xs
  shows   (prefix ia (prefix (ilen xs - Suc 0) xs)) = (prefix ia xs)
using assms
by (metis Suc-leI Suc-le-mono Suc-pred diff-le-self pref-help prefix-ilen-good)

lemma interval-pref-pref-help-1:
  assumes i > 0
           i ≤ ilen xs
  shows   (prefix (ilen (prefix i xs) - Suc 0) (prefix i xs)) =
            (prefix (ilen (prefix i xs) - Suc 0) xs)
using assms pref-pref-3 by (metis diff-le-self prefix-ilen-good le-iff-add)

lemma suffix-suc [simp]:
  suffix (Suc m) (x ⊙ xs) = suffix m xs
by auto

lemma suffix-zero [simp]:
  suffix 0 xs = xs
by (induct xs) simp-all

lemma interval-hd-tail:
  assumes ilen xs > 0
  shows   xs = (ifirst xs) ⊙ (suffix 1 xs)
by (metis One-nat-def assms ilen-ICons-1 inth-zero suffix-suc suffix-zero)

lemma suffix-ilen-last [simp]:
  suffix (ilen xs) xs = ⟨(inth xs (ilen xs))⟩
by (induct xs) simp-all

lemma suffix-ilast:
  suffix (ilen xs) xs = ⟨ilast xs⟩
by simp

lemma suffix-suffix [simp]:
  suffix i (suffix j xs) = suffix (i+j) xs
proof
  (induct xs arbitrary: i j)
  case (INil x)
  then show ?case by simp
  next
  case (ICons x1a xs)
  then show ?case
  proof (cases i)
  case 0

```



```

then show ?thesis
  by auto
next
case (Suc nat)
then show ?thesis
  by (simp add: Nitpick.case-nat-unfold add.commute ICons)
qed
qed

lemma prefix-suffix-ilen-code [code]:
  ilen (prefix ia (suffix i xs)) =
    (if i ≤ ilen xs then
      (if ia ≤ ilen xs - i then ia else (ilen xs) - i)
      else 0)
using suffix-ilen-code by auto

lemma prefix-suffix-ilen [simp]:
  ilen (prefix ia (suffix i xs)) = min ia (ilen xs - i)
by auto

lemma prefix-suffix-ilen-good [simp]:
  assumes ia ≤ ilen xs - i
    i ≤ ilen xs
  shows ilen (prefix ia (suffix i xs)) = ia
using assms by auto

lemma prefix-suffix-ilen-bad-0:
  assumes i > ilen xs
  shows ilen (prefix ia (suffix i xs)) = 0
using assms by simp

lemma prefix-suffix-ilen-bad-1 :
  assumes i ≤ ilen xs
    ia > ilen xs - i
  shows ilen (prefix ia (suffix i xs)) = (ilen xs) - i
using assms by simp

lemma suffix-suffix-3:
  assumes i > 0
    ia < i
    i ≤ ilen xs
  shows (suffix (i - ia) (suffix ((ilen xs) - i) xs)) = (suffix (((ilen xs) - ia)) xs)
using assms by simp

lemma sub-zero-prefix :
  sub 0 k xs = prefix k xs
by (simp add: sub-def)

lemma sub-suffix :
  assumes i < j

```

$j \leq (\text{ilen } xs) - k$   
**shows**  $(\text{sub } (i+k) (j+k) xs) = (\text{sub } i j (\text{suffix } k xs))$   
**using** *assms* **by** (*simp add: sub-def*)

**lemma** *sub-prefix-suffix-0*:  
**assumes**  $0 \leq i$   
 $ia + i \leq \text{ilen } xs$   
**shows**  $(\text{sub } i (i+ia) xs) = (\text{prefix } (ia) (\text{suffix } i xs))$   
**using** *assms* **by** (*simp add: sub-def*)

**lemma** *sub-prefix-suffix*:  
**assumes**  $0 \leq i$   
 $i \leq j$   
 $j \leq \text{ilen } xs$   
**shows**  $(\text{sub } i j xs) = (\text{prefix } (j-i) (\text{suffix } i xs))$   
**using** *assms* **by** (*simp add: sub-def*)

**lemma** *ilast-prefix*:  
**assumes**  $k \leq \text{ilen } xs$   
**shows**  $\text{ilast}(\text{prefix } k xs) = (\text{inth } xs k)$   
**using** *assms prefix-ilen-good* **by** *fastforce*

**lemma** *ifirst-suffix*:  
**assumes**  $k \leq \text{ilen } xs$   
**shows**  $\text{ifirst}(\text{suffix } k xs) = (\text{inth } xs k)$   
**by** (*simp add: assms*)

**lemma** *suffix-gr*:  
**assumes**  $i > \text{ilen } xs$   
**shows**  $\text{suffix } i xs = \langle \text{ilast}(xs) \rangle$   
**by** (*metis add.commute assms suffix-ilast suffix-suffix less-imp-add-positive suffix.simps(1)*)

**lemma** *ilast-ifirst*:  
 $(\text{ilast } (\text{prefix } i xs)) = (\text{ifirst } (\text{suffix } i xs))$   
**proof** –  
**have** 1:  $(\text{ilast } (\text{prefix } i xs)) = (\text{inth } (\text{prefix } i xs) (\text{ilen } (\text{prefix } i xs)))$   
**by** *simp*  
**have** 2:  $i \leq \text{ilen } xs \longrightarrow \text{ilen } (\text{prefix } i xs) = i$   
**using** *prefix-ilen-good* **by** *blast*  
**have** 3:  $i > \text{ilen } xs \longrightarrow \text{ilen } (\text{prefix } i xs) = \text{ilen } xs$   
**using** *prefix-ilen-bad* **by** *blast*  
**have** 4:  $i \leq \text{ilen } xs \longrightarrow$   
 $(\text{inth } (\text{prefix } i xs) (\text{ilen } (\text{prefix } i xs))) = (\text{inth } xs i)$   
**using** *ilast-prefix* **by** *auto*  
**have** 5:  $i > \text{ilen } xs \longrightarrow$   
 $(\text{inth } (\text{prefix } i xs) (\text{ilen } (\text{prefix } i xs))) = (\text{inth } xs (\text{ilen } xs))$   
**using** 3 **by** *auto*  
**have** 6:  $(\text{ifirst } (\text{suffix } i xs)) = (\text{inth } (\text{suffix } i xs) 0)$   
**by** *simp*

**have** 7:  $i \leq \text{ilen } xs \longrightarrow$   
 $(\text{inth } (\text{suffix } i \text{ } xs) \ 0) = (\text{inth } xs \ i)$   
**by** *simp*  
**have** 8:  $i > \text{ilen } xs \longrightarrow$   
 $(\text{inth } (\text{suffix } i \text{ } xs) \ 0) = (\text{inth } xs \ (\text{ilen } xs))$   
**by** (*simp add: suffix-gr*)  
**show** ?thesis **using** 4 5 8 **by** *auto*  
**qed**

**lemma** *ilen-sub [simp]*:  
**assumes**  $k \leq n$   
 $n \leq \text{ilen } xs$   
**shows**  $\text{ilen}(\text{sub } k \ n \ xs) = (n - k)$   
**using** *assms*  
**by** (*metis sub-def prefix-ilen-good suffix-ilen suffix-prefix-swap le-add-diff-inverse2*)

**lemma** *inth-sub [simp]*:  
**assumes**  $k \leq n$   
 $n \leq \text{ilen } xs$   
 $j \leq n - k$   
**shows**  $\text{inth}(\text{sub } k \ n \ xs) \ j = (\text{inth } xs \ (k + j))$   
**proof** –  
**have** 1:  $\text{inth}(\text{sub } k \ n \ xs) \ j = \text{inth } (\text{prefix } (n - k) \ (\text{suffix } k \ xs)) \ j$   
**by** (*simp add: sub-def*)  
**have** 2:  $n - k \leq \text{ilen } (\text{suffix } k \ xs)$   
**using** *sub-def assms* **by** *auto*  
**have** 3:  $j \leq (n - k)$   
**using** *assms* **by** *auto*  
**have** 4:  $\text{inth } (\text{prefix } (n - k) \ (\text{suffix } k \ xs)) \ j =$   
 $\text{inth } (\text{suffix } k \ xs) \ j$   
**using** 2 *assms inth-prefix* **by** *blast*  
**have** 5:  $\text{inth } (\text{suffix } k \ xs) \ j = \text{inth } xs \ (k + j)$   
**using** *assms* **by** *auto*  
**show** ?thesis  
**by** (*simp add: 1 4 5*)  
**qed**

**lemma** *ilast-sub*:  
**assumes**  $k \leq n$   
 $n \leq \text{ilen } xs$   
**shows**  $\text{ilast } (\text{sub } k \ n \ xs) = (\text{inth } xs \ n)$   
**by** (*simp add: assms*)

**lemma** *ifirst-sub*:  
**assumes**  $k \leq n$   
 $n \leq \text{ilen } xs$   
**shows**  $\text{ifirst } (\text{sub } k \ n \ xs) = (\text{inth } xs \ k)$   
**by** (*simp add: assms*)

**lemma** *sub-sub*:

**assumes**  $n1 \leq n2$

$n0 \leq n4$

$n2 \leq n4 - n0$

$n4 \leq n3$

$n3 \leq \text{ilen } xs$

**shows**  $(\text{sub } n1 \ n2 (\text{sub } n0 \ n3 \ xs)) = (\text{sub } n1 \ n2 (\text{sub } n0 \ n4 \ xs))$

**proof** –

**have** 1:  $\text{ilen}(\text{sub } n0 \ n3 \ xs) = n3 - n0$

**by** (*meson assms ilen-sub le-trans*)

**have** 2:  $\text{ilen}(\text{sub } n1 \ n2 (\text{sub } n0 \ n3 \ xs)) = n2 - n1$

**using** *ilen-sub assms* **by** *auto*

**have** 3:  $\text{ilen}(\text{sub } n0 \ n4 \ xs) = n4 - n0$

**using** *assms ilen-sub le-trans* **by** *blast*

**have** 4:  $\text{ilen}(\text{sub } n1 \ n2 (\text{sub } n0 \ n4 \ xs)) = n2 - n1$

**by** (*simp add: 3 assms*)

**have** 5:  $\bigwedge i. i \leq (n3 - n0) \longrightarrow (\text{inth}(\text{sub } n0 \ n3 \ xs) \ i) = (\text{inth } xs \ (n0 + i))$

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

**have** 6:  $\bigwedge i. i \leq (n2 - n1) \longrightarrow (\text{inth}(\text{sub } n1 \ n2 (\text{sub } n0 \ n3 \ xs)) \ i) = (\text{inth}(\text{sub } n0 \ n3 \ xs) \ (n1 + i))$

**using** *inth-sub assms*

**by** (*metis 1 Nat.le-diff-conv2 le-trans*)

**have** 7:  $\bigwedge i. i \leq (n2 - n1) \longrightarrow (\text{inth}(\text{sub } n1 \ n2 (\text{sub } n0 \ n3 \ xs)) \ i) = (\text{inth } xs \ (n0 + (n1 + i)))$

**using** 5 6 *assms* **by** *auto*

**have** 8:  $n0 \leq n4 \wedge n4 \leq \text{ilen } xs$

**using** *assms le-trans* **by** *blast*

**have** 9:  $\bigwedge i. i \leq (n4 - n0) \longrightarrow (\text{inth}(\text{sub } n0 \ n4 \ xs) \ i) = (\text{inth } xs \ (n0 + i))$

**using** 8 *inth-sub* **by** *blast*

**have** 10:  $\bigwedge i. i \leq (n2 - n1) \longrightarrow (\text{inth}(\text{sub } n1 \ n2 (\text{sub } n0 \ n4 \ xs)) \ i) = (\text{inth}(\text{sub } n0 \ n4 \ xs) \ (n1 + i))$

**by** (*simp add: 3 assms*)

**have** 11:  $\bigwedge i. i \leq (n2 - n1) \longrightarrow (\text{inth}(\text{sub } n1 \ n2 (\text{sub } n0 \ n4 \ xs)) \ i) = (\text{inth } xs \ (n0 + (n1 + i)))$

**by** (*metis 3 Nat.le-diff-conv2 add commute assms inth-sub le-trans*)

**have** 12:  $\bigwedge i. i \leq (n2 - n1) \longrightarrow$

$(\text{inth}(\text{sub } n1 \ n2 (\text{sub } n0 \ n3 \ xs)) \ i) = (\text{inth}(\text{sub } n1 \ n2 (\text{sub } n0 \ n4 \ xs)) \ i)$

**by** (*simp add: 11 7*)

**from** 12 2 4 **show** *?thesis* **by** (*simp add: interval-eq-inth-eq*)

**qed**

**lemma** *sub-sub-1*:

**assumes**  $n1 \leq n2$

$n0 \leq n3$

$n2 \leq n3 - n0$

$n3 \leq \text{ilen } xs$

**shows**  $(\text{sub } n1 \ n2 (\text{sub } n0 \ n3 \ xs)) = (\text{sub } (n0 + n1) \ (n0 + n2) \ xs)$

**proof** –

**have** 1:  $\text{ilen}(\text{sub } n1 \ n2 (\text{sub } n0 \ n3 \ xs)) = \text{ilen}(\text{sub } (n1 + n0) \ (n2 + n0) \ xs)$

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

**have** 2:  $\bigwedge i. i \leq (n2 - n1) \longrightarrow (\text{inth}(\text{sub } n1 \ n2 (\text{sub } n0 \ n3 \ xs)) \ i) = (\text{inth } xs \ (n0 + (n1 + i)))$

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

**have** 3:  $\bigwedge i. i \leq (n2 - n1) \longrightarrow (\text{inth}(\text{sub } (n0 + n1) \ (n0 + n2) \ xs) \ i) = (\text{inth } xs \ (n0 + (n1 + i)))$

**using** *assms* **by** (*metis* *ab-semigroup-add-class.add-ac(1)* *diff-add-inverse* *inth-sub* *le-Suc-ex* *le-add1*)  
**have** 4:  $\bigwedge i. i \leq (n2 - n1) \longrightarrow (inth \ (sub \ n1 \ n2 \ (sub \ n0 \ n3 \ xs)) \ i) = (inth \ (sub \ (n0 + n1) \ (n0 + n2) \ xs) \ i)$   
**by** (*simp* *add: 2 3*)  
**show** *?thesis*  
**by** (*metis* 1 4 *add.commute* *assms* *interval-eq-inth-eq* *ilen-sub*)  
**qed**

**lemma** *suf-first-upto*:

**assumes**  $(\exists i < k. f \ (suffix \ i \ xs))$   
 $k \leq ilen \ xs + 1$

**shows**  $(\exists i < k. f \ (suffix \ i \ xs) \wedge$   
 $(\forall j < i. \neg (f \ (suffix \ j \ xs))))$

**using** *assms*

**proof** (*induct* *xs* *arbitrary:k*)

**case** (*INil* *x*)

**then show** *?case* **by** *auto*

**next**

**case** (*ICons* *x1a* *xs*)

**then show** *?case*

**proof** –

**have** 0:  $k = 0 \longrightarrow (\exists i < k. f \ (suffix \ i \ (x1a \odot xs)) \wedge (\forall j < i. \neg f \ (suffix \ j \ (x1a \odot xs))))$

**using** *ICons.prem(1)* **by** *blast*

**have** 1:  $\bigwedge n. k = (Suc \ n) \longrightarrow (\exists i < (Suc \ n). f \ (suffix \ i \ (x1a \odot xs)) \wedge$   
 $(\forall j < i. \neg f \ (suffix \ j \ (x1a \odot xs)))) =$   
 $(f \ (x1a \odot xs) \vee$   
 $(\exists i. 1 \leq i \wedge i < (Suc \ n) \wedge f \ (suffix \ i \ (x1a \odot xs)) \wedge$   
 $(\forall j < i. \neg f \ (suffix \ j \ (x1a \odot xs))))$

**by** *auto*

(*metis* *Nitpick.case-nat-unfold* *less-antisym* *not-le-imp-less* *not-less0*)

**have** 2:  $\bigwedge n. k = (Suc \ n) \longrightarrow (\exists i. 1 \leq i \wedge i < (Suc \ n) \wedge f \ (suffix \ i \ (x1a \odot xs)) \wedge$   
 $(\forall j < i. \neg f \ (suffix \ j \ (x1a \odot xs)))) =$   
 $(\exists i. i < n \wedge f \ (suffix \ (Suc \ i) \ (x1a \odot xs)) \wedge$   
 $(\forall j < (Suc \ i). \neg f \ (suffix \ j \ (x1a \odot xs))))$

**by** *auto*

(*metis* *Suc-le-D* *nat.case(2)* *not-less* *not-less-eq-eq*)

**have** 3:  $\bigwedge n. k = (Suc \ n) \longrightarrow (\exists i. i < n \wedge f \ (suffix \ (Suc \ i) \ (x1a \odot xs)) \wedge$   
 $(\forall j < (Suc \ i). \neg f \ (suffix \ j \ (x1a \odot xs)))) =$   
 $(\exists i. i < n \wedge f \ (suffix \ i \ xs) \wedge$   
 $\neg(f \ (suffix \ 0 \ (x1a \odot xs))) \wedge$   
 $(\forall j. 1 \leq j \wedge j < (Suc \ i) \longrightarrow \neg f \ (suffix \ j \ (x1a \odot xs))))$

**by** (*metis* *suffix-suc* *le-add1* *less-Suc-eq-0-disj* *plus-1-eq-Suc*)

**have** 4:  $\bigwedge n. k = (Suc \ n) \longrightarrow (\exists i. i < n \wedge f \ (suffix \ i \ xs) \wedge$   
 $\neg(f \ (suffix \ 0 \ (x1a \odot xs))) \wedge$   
 $(\forall j. 1 \leq j \wedge j < (Suc \ i) \longrightarrow \neg f \ (suffix \ j \ (x1a \odot xs)))) =$   
 $(\neg(f \ (x1a \odot xs)) \wedge$   
 $(\exists i. i < n \wedge f \ (suffix \ i \ xs) \wedge$   
 $(\forall j. j < i \longrightarrow \neg f \ (suffix \ (Suc \ j) \ (x1a \odot xs))))$   
**using** *ICons.hyps* *ICons.prem(1)* **by** (*auto*, *auto* *simp* *add: less-Suc-eq-0-disj*)

**have 5:**  $\bigwedge n. k = (Suc\ n) \longrightarrow ( \neg(f\ (x1a \odot xs)) \wedge$   
 $(\exists i. i < n \wedge f\ (suffix\ i\ xs) \wedge$   
 $(\forall j. j < i \longrightarrow \neg f\ (suffix\ (Suc\ j)\ (x1a \odot xs)))) ) =$   
 $( \neg(f\ (x1a \odot xs)) \wedge$   
 $(\exists i < n. f\ (suffix\ i\ xs) \wedge$   
 $(\forall j < i. \neg f\ (suffix\ j\ xs)))) )$   
**by auto**  
**have 6:**  $\bigwedge n. k = (Suc\ n) \longrightarrow (\exists i < (Suc\ n). f\ (suffix\ i\ (x1a \odot xs)) \wedge$   
 $(\forall j < i. \neg f\ (suffix\ j\ (x1a \odot xs)))) =$   
 $( f\ (x1a \odot xs) \vee ( \neg(f\ (x1a \odot xs)) \wedge$   
 $(\exists i < n. f\ (suffix\ i\ xs) \wedge$   
 $(\forall j < i. \neg f\ (suffix\ j\ xs)))) ) )$   
**using 1 2 3 4 by auto**  
**have 7:**  $\bigwedge n. k = (Suc\ n) \longrightarrow ( f\ (x1a \odot xs) \vee ( \neg(f\ (x1a \odot xs)) \wedge$   
 $(\exists i < n. f\ (suffix\ i\ xs) \wedge$   
 $(\forall j < i. \neg f\ (suffix\ j\ xs)))) ) =$   
 $( f\ (x1a \odot xs) \vee (\exists i < n. f\ (suffix\ i\ xs) \wedge$   
 $(\forall j < i. \neg f\ (suffix\ j\ xs)))) )$   
**by auto**  
**have 8:**  $\bigwedge n. k = (Suc\ n) \longrightarrow n \leq ilen\ xs + 1$   
**using ICons.premis(2) by auto**  
**have 9:**  $\bigwedge n. k = (Suc\ n) \longrightarrow ( f\ (x1a \odot xs) \vee (\exists i < n. f\ (suffix\ i\ xs) \wedge$   
 $(\forall j < i. \neg f\ (suffix\ j\ xs)))) =$   
 $( f\ (x1a \odot xs) \vee (\exists i < n. f\ (suffix\ i\ xs)))$   
**using 7 8 ICons.hyps by fastforce**  
**have 10:**  $\bigwedge n. k = (Suc\ n) \longrightarrow (\exists i < k. f\ (suffix\ i\ (x1a \odot xs))) =$   
 $( f\ (x1a \odot xs) \vee (\exists i < n. f\ (suffix\ i\ (xs))))$   
**using Nitpick.case-nat-unfold less-Suc-eq-0-disj by auto**  
**have 11:**  $\bigwedge n. k = (Suc\ n) \longrightarrow$   
 $( \exists i < k. f\ (suffix\ i\ (x1a \odot xs)) \wedge (\forall j < i. \neg f\ (suffix\ j\ (x1a \odot xs))))$   
**using 9 10 6 ICons.premis(1) by force**  
**show ?thesis**  
**using 0 11 less-imp-Suc-add by blast**  
**qed**  
**qed**

### 1.2.5 iapp

**lemma** *ilen-snoc-1:*

*ilen l > 0 = ( $\exists x\ ls. l = ls \odot \langle x \rangle$ )*

**proof** (*induct l*)

**case** (*INil x*)

**then show** *?case by fastforce*

**next**

**case** (*ICons x1a l*)

**then show** *?case*

**by** (*metis Suc-eq-plus1 iapp-ICons iapp-INil interval.exhaust ilen-iapp nat.simps(3) neq0-conv*)

**qed**

```

lemma prefix-iapp [simp]:
  prefix (ilen xs - k) (xs ⊖ ys) = prefix (ilen xs - k) xs
proof
  (induct xs arbitrary: k)
  case (INil x)
  then show ?case by simp
  next
  case (ICons x1a xs)
  then show ?case
    proof (cases k)
    case 0
    then show ?thesis
      by (metis ICons.hyps diff-zero iapp-ICons prefix-suc ilen.simps(2) plus-1-eq-Suc)
    next
    case (Suc nat)
    then show ?thesis
      by (auto simp add: ICons.hyps Nitpick.case-nat-unfold)
    qed
  qed

```

```

lemma prefix-iapp2 [simp]:
  prefix (ilen xs + k + 1) (xs ⊖ ys) = xs ⊖ prefix k ys
proof
  (induct xs arbitrary: k)
  case (INil x)
  then show ?case by simp
  next
  case (ICons x1a xs)
  then show ?case
    proof (cases k)
    case 0
    then show ?thesis
      by auto
      (metis ICons.hyps Suc-eq-plus1 add.right-neutral prefix-zero-ifirst)
    next
    case (Suc nat)
    then show ?thesis
      by auto
      (metis ICons.hyps Suc-eq-plus1 add-Suc-right)
    qed
  qed

```

```

lemma suffix-iapp [simp]:
  suffix (ilen xs + m + 1) (xs ⊖ ys) = suffix (m) ys
proof
  (induct xs arbitrary: m)

```

```

case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
  proof (cases m)
    case 0
    then show ?thesis
      by auto
      (metis ICons.hyps One-nat-def Suc-eq-plus1 suffix-zero plus-1-eq-Suc
        semiring-normalization-rules(23))
    next
    case (Suc nat)
    then show ?thesis
      by auto
      (metis ICons.hyps Suc-eq-plus1 suffix-suffix)
  qed
qed

```

```

lemma suffix-iapp2 [simp]:
  (suffix (ilen xs - k) xs)  $\ominus$  ys = suffix (ilen xs - k) (xs  $\ominus$  ys)
proof
  (induct xs)
  case (INil x)
  then show ?case by simp
  next
  case (ICons x1a xs)
  then show ?case
    by (auto simp add: Nitpick.case-nat-unfold)
  qed

```

```

lemma iapp-assoc [simp]:
  (xs  $\ominus$  ys)  $\ominus$  zs = xs  $\ominus$  (ys  $\ominus$  zs)
by (induct xs) simp-all

```

```

lemma iapp-inth:
  inth (xs  $\ominus$  ys) k = (if k  $\leq$  ilen xs
    then (inth xs k)
    else (inth ys (k - (ilen xs) - 1)) )
proof
  (induct xs arbitrary: k)
  case (INil x)
  then show ?case
    proof (cases k)
      case 0
      then show ?thesis by simp
    next
    case (Suc nat)
    then show ?thesis by simp
  qed

```



```

next
case (ICons x1a xs)
then show ?case
proof (cases k)
case 0
then show ?thesis by simp
next
case (Suc nat)
then show ?thesis by (simp add: ICons.hyps)
qed
qed

```

```

lemma irev-iapp [simp]:
  irev (xs  $\ominus$  ys) = (irev ys)  $\ominus$  (irev xs)
by (induct xs) simp-all

```

```

lemma ilast-iapp [simp]:
  ilast(xs  $\ominus$   $\langle x \rangle$ ) = x
by (induct xs) simp-all

```

```

lemma ilast-iapp2 [simp]:
  ilast (xs  $\ominus$  ys) = ilast ys
by (induct xs arbitrary: ys) simp-all

```

```

lemma ifirst-iapp [simp]:
  ifirst ( $\langle x \rangle$   $\ominus$  xs) = x
by (induct xs) simp-all

```

```

lemma ifirst-iapp2 [simp]:
  ifirst(xs  $\ominus$  ys) = ifirst xs
by (induct xs arbitrary: ys) simp-all

```

```

lemma iapp-not-INil [simp]:
  xs  $\ominus$  ys  $\neq$   $\langle x \rangle$ 
by (induct xs arbitrary: ys) simp-all

```

```

lemma iapp-eq-iapp-conv [simp]:
  assumes ilen xs = ilen ys  $\vee$  ilen us = ilen vs
  shows (xs  $\ominus$  us = ys  $\ominus$  vs) = (xs = ys  $\wedge$  us = vs)
using assms
proof
  (induct xs arbitrary: ys)
  case (INil x)
  then show ?case
  proof (cases ys)
  case (INil x1)
  then show ?thesis by simp
  next
  case (ICons x21 x22)

```

```

    then show ?thesis using INil.prem by auto
  qed
next
case (ICons x1a xs)
then show ?case
  proof (cases ys)
    case (INil x1)
    then show ?thesis using ICons.prem by auto
  next
    case (ICons x21 x22)
    then show ?thesis using ICons.hyps ICons.prem by auto
  qed
qed

```

**lemma** *iapp-eq-iapp-conv2*:

$$\begin{aligned}
 (xs \ominus ys = zs \ominus ts) = \\
 (\exists us. xs = zs \ominus us \wedge us \ominus ys = ts \vee \\
 xs = zs \wedge ys = ts \vee \\
 xs \ominus us = zs \wedge ys = us \ominus ts)
 \end{aligned}$$

**proof**

(*induct xs arbitrary: ys zs ts*)

**case** (INil x)

**then show** ?case

**proof** (cases zs)

**case** (INil x1)

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

**next**

**case** (ICons x21 x22)

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

**qed**

**next**

**case** (ICons x1a xs)

**then show** ?case

**proof** (cases zs)

**case** (INil x1)

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

**next**

**case** (ICons x21 x22)

**then show** ?thesis **by** (*auto simp add: ICons.hyps*)

**qed**

**qed**

**lemma** *same-iapp-eq[iff, induct-simp]*:

$$(xs \ominus ys = xs \ominus zs) = (ys = zs)$$

**using** *suffix-iapp* **by** (*metis suffix-zero*)

**lemma** *iapp-eq-conv[iff]*:

$$(xs \ominus \langle x \rangle = ys \ominus \langle y \rangle) = (xs = ys \wedge x = y)$$

**by** *auto*

**lemma** *iapp-same-eq*[*iff*, *induct-simp*]:

$(ys \ominus xs = zs \ominus xs) = (ys = zs)$

**by** *auto*

**lemma** *suffix1-iapp*:

$\text{suffix } 1 \ (xs \ominus ys) = (\text{case } xs \text{ of } \langle x \rangle \Rightarrow ys \mid x \odot zs \Rightarrow zs \ominus ys)$

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

**lemma** *ICons-eq-iapp-conv*:

$(x \odot xs = ys \ominus zs) =$

$((\langle x \rangle = ys \wedge xs = zs) \vee (\exists \ ys'. \ x \odot ys' = ys \wedge xs = ys' \ominus zs))$

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

**lemma** *iapp-eq-ICons-conv*:

$(ys \ominus zs = x \odot xs) =$

$((\langle x \rangle = ys \wedge zs = xs) \vee (\exists \ ys'. \ ys = x \odot ys' \wedge ys' \ominus zs = xs))$

**by** (*cases ys*) *auto*

**lemma** *ICons-eq-iappI*:

**assumes**  $x \odot xs1 = ys$

$xs = xs1 \ominus zs$

**shows**  $x \odot xs = ys \ominus zs$

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

**lemma** *iapp-eq-iappI*:

**assumes**  $xs \ominus xs1 = zs$

$ys = xs1 \ominus us$

**shows**  $xs \ominus ys = zs \ominus us$

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

**lemma** *iapp-gr-zero*:

$\text{ilen } (xs \ominus ys) > 0$

**by** *auto*

**lemma** *iapp-prefix-suffix*:

**assumes**  $i+1 \leq \text{ilen } xs$

$\text{ilen } xs > 0$

**shows**  $xs = (\text{prefix } i \ xs) \ominus (\text{suffix } (i+1) \ xs)$

**using** *assms*

**proof** (*induct xs arbitrary:i*)

**case** (*INil x*)

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

**next**

**case** (*ICons x1a xs*)

**then show** *?case*

**proof** (*cases i*)

**case** *0*

```

then show ?thesis by auto
next
case (Suc nat)
then show ?thesis using ICons.hyps ICons.prem1 by auto
qed
qed

```

### 1.2.6 Reverse

**lemma** *irev-irev-ident* [simp]:

$irev (irev xs) = xs$

**by** (induct xs) auto

**lemma** *irev-swap* :

$((irev xs) = ys) = (xs = irev ys)$

**by** auto

**lemma** *irev-singleton-conv* [simp]:

$(irev xs = \langle x \rangle) = (xs = \langle x \rangle)$

**by** (metis irev-irev-ident irev.simps(1))

**lemma** *single-irev-conv* [simp]:

$(\langle x \rangle = irev xs) = (\langle x \rangle = xs)$

**by** (metis irev-irev-ident irev.simps(1))

**lemma** *irev-is-irev-conv* [iff]:

$(irev xs = irev ys) = (xs = ys)$

**proof**

(induct xs arbitrary: ys)

**case** (INil x)

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

**next**

**case** (ICons x1a xs)

**then show** ?case

**using** irev-swap **by** force

**qed**

**lemma** *irev-induct* [case-names INil snoc]:

**assumes**  $\bigwedge y. P \langle y \rangle$

$\bigwedge x xs. P xs \implies P(xs \ominus \langle x \rangle)$

**shows**  $P xs$

**using** assms

**using** interval.induct[of  $\lambda xs. P (irev xs) irev xs$ ]

**by** simp

**lemma** *irev-exhaust* [case-names INil snoc]:

**assumes**  $\bigwedge x. xs = \langle x \rangle \implies P$

$\bigwedge ys y. xs = ys \ominus \langle y \rangle \implies P$

**shows**  $P$

**using** *assms*  
**by** (*induct xs rule:irev-induct*) *auto*

**lemmas** *irev-cases* = *irev-exhaust*

**lemma** *interval-irev-eq-ICons-iff* [*iff*]:  
 $(\text{irev } xs = y \odot ys) = (xs = (\text{irev } ys) \ominus \langle y \rangle)$   
**by** (*metis irev-irev-ident irev.simps(2)*)

**lemma** *irev-iapp-ICons*:  
 $\text{irev } (xs \ominus \langle x \rangle) = x \odot \text{irev } xs$   
**by** (*cases xs*) *simp-all*

**lemma** *ilast-irev*:  
 $\text{ilast } (\text{irev } xs) = \text{ifirst } xs$   
**proof** (*cases xs*)  
**case** (*INil x1*)  
**then show** *?thesis*  
**by** *simp*  
**next**  
**case** (*ICons x21 x22*)  
**then show** *?thesis*  
**by** (*metis ilast-iapp inth-zero irev.simps(2)*)  
**qed**

**lemma** *ifirst-irev*:  
 $\text{ifirst } (\text{irev } xs) = \text{ilast } xs$   
**proof**  
 $(\text{induct } xs)$   
**case** (*INil x*)  
**then show** *?case* **by** *simp*  
**next**  
**case** (*ICons x1a xs*)  
**then show** *?case*  
**by** (*metis ifirst-iapp2 inth-last irev.simps(2)*)  
**qed**

**lemma** *irev-inth*:  
**assumes**  $k \leq \text{ilen } (\text{irev } xs)$   
**shows**  $(\text{inth } (\text{irev } xs) k) = (\text{inth } xs ((\text{ilen } xs) - k))$   
**using** *assms*  
**proof**  
 $(\text{induct } xs)$   
**case** (*INil x*)  
**then show** *?case* **by** *simp*  
**next**  
**case** (*ICons x1a xs*)  
**then show** *?case*

```

proof (cases k)
case 0
then show ?thesis
  by auto
    (metis ICons.hyps diff-zero ilen-gr-zero)
next
case (Suc nat)
then show ?thesis
  using ICons.hyps Suc-diff-le by (auto simp add: iapp-inth) fastforce
qed
qed

```

```

lemma irev-prefix:
assumes  $k \leq \text{ilen } xs$ 
shows  $\text{irev}(\text{prefix } k \text{ } xs) = \text{suffix } ((\text{ilen } xs) - k) (\text{irev } xs)$ 
proof
  (induct xs arbitrary: k)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
  proof (cases k)
    case 0
    then show ?thesis
      by auto
        (metis Suc-eq-plus1 add.right-neutral ilast-iapp ilen-iapp
          irev-ilen suffix-ilast ilen.simps(1))
    next
    case (Suc nat)
    then show ?thesis
      by auto
        (metis ICons.hyps irev-ilen suffix-iapp2)
  qed
qed

```

```

lemma irev-suffix:
assumes  $k \leq \text{ilen } xs$ 
shows  $\text{irev}(\text{suffix } k \text{ } xs) = \text{prefix } ((\text{ilen } xs) - k) (\text{irev } xs)$ 
using assms
proof
  (induct xs arbitrary: k)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case by (simp add: irev-prefix irev-swap)
qed

```

**lemma** *irev-sub*:  
**assumes**  $0 \leq i$   
 $i \leq j$   
 $j \leq \text{ilen } xs$   
**shows**  $\text{irev } (\text{sub } i \ j \ xs) = \text{sub } ((\text{ilen } xs) - j) ((\text{ilen } xs) - i) (\text{irev } xs)$   
**using** *assms*  
**proof** –  
**have** 1:  $\text{irev } (\text{sub } i \ j \ xs) = \text{irev } (\text{prefix } (j-i) (\text{suffix } i \ xs))$   
**using** *assms sub-prefix-suffix* **by** (*simp add: sub-prefix-suffix*)  
**have** 2:  $\text{irev } (\text{prefix } (j-i) (\text{suffix } i \ xs)) = \text{suffix } ((\text{ilen } xs) - j) (\text{irev } (\text{suffix } i \ xs))$   
**using** *assms irev-prefix[of j-i suffix i xs]* **by** *auto*  
**have** 3:  $\text{suffix } ((\text{ilen } xs) - j) (\text{irev } (\text{suffix } i \ xs)) =$   
 $\text{suffix } ((\text{ilen } xs) - j) (\text{prefix } ((\text{ilen } xs) - i) (\text{irev } xs))$   
**using** *assms irev-suffix[of i xs]* **by** *auto*  
**have** 4:  $\text{suffix } ((\text{ilen } xs) - j) (\text{prefix } ((\text{ilen } xs) - i) (\text{irev } xs)) =$   
 $\text{sub } ((\text{ilen } xs) - j) ((\text{ilen } xs) - i) (\text{irev } xs)$   
**using** *assms* **by** (*simp add: diff-le-mono2 sub-prefix-suffix suffix-prefix-swap*)  
**from** 1 2 3 4 **show** ?thesis **by** *auto*  
**qed**

### 1.2.7 Induction rule

**lemma** *interval-ilen-induct*:  
**assumes**  $(\bigwedge xs. \forall ys. \text{ilen } ys < \text{ilen } xs \longrightarrow P \ ys \Longrightarrow P \ xs)$   
**shows**  $P \ xs$   
**using** *assms* **by** (*fact measure-induct*)

**lemma** *interval-induct-12*:  
**assumes**  $\bigwedge x. P \ \langle x \rangle$   
 $\bigwedge x \ y. P \ \langle x, y \rangle$   
 $\bigwedge x \ y \ zs. P \ (y \odot zs) \Longrightarrow P \ (x \odot y \odot zs)$   
**shows**  $P \ xs$   
**using** *assms*  
**proof** (*induction xs*)  
**case** (*INil x*)  
**then show** ?case **by** *simp*  
**next**  
**case** (*ICons x1a xs*)  
**then show** ?case  
**by** *auto*  
 $(\text{metis add-cancel-right-right ilen-ICons-1 INil-ilen le-add1 le-eq-less-or-eq less-add-same-cancel1})$   
**qed**

**lemma** *interval-induct2* [*consumes 1, case-names INil ICons*]:  
**assumes**  $\text{ilen } xs = \text{ilen } ys$   
 $(\bigwedge x \ y. P \ \langle x \rangle \ \langle y \rangle)$   
 $(\bigwedge x \ xs \ y \ ys. \text{ilen } xs = \text{ilen } ys \Longrightarrow P \ xs \ ys \Longrightarrow P \ (x \odot xs) \ (y \odot ys))$   
**shows**  $P \ xs \ ys$

```

using assms
proof (induction xs arbitrary: ys)
case (INil x)
then show ?case
by (metis INil-ilen)
next
case (ICons x1a xs)
then show ?case
by (metis ilen-ICons-1 ilen.simps(2) nat.simps(1) plus-1-eq-Suc)
qed

```

### 1.2.8 Imap

```

lemma imap-ext:
  assumes  $(\forall x. x \in \text{iset } xs \longrightarrow f\ x = g\ x)$ 
  shows  $\text{imap } f\ xs = \text{imap } g\ xs$ 
using assms
by (induct xs simp-all)

```

```

lemma imap-ident [simp]:
   $\text{imap } (\lambda x. x) = (\lambda xs. xs)$ 
proof (rule ext)
  show  $\bigwedge xs. \text{imap } (\lambda x. x)\ xs = xs$ 
  by (simp add: interval.map-ident)
qed

```

```

lemma imap-iapp [simp]:
   $\text{imap } f\ (xs \ominus ys) = \text{imap } f\ xs \ominus \text{imap } f\ ys$ 
by (induct xs auto)

```

```

lemma imap-imap [simp]:
   $\text{imap } f\ (\text{imap } g\ xs) = \text{imap } (f \circ g)\ xs$ 
by (simp add: interval.map-comp)

```

```

lemma imap-comp-imap [simp]:
   $((\text{imap } f) \circ (\text{imap } g)) = \text{imap}(f \circ g)$ 
by (rule ext simp)

```

```

lemma irev-imap:
   $\text{irev } (\text{imap } f\ xs) = \text{imap } f\ (\text{irev } xs)$ 
by (induct xs auto)

```

```

lemma imap-eq-conv [simp]:
   $(\text{imap } f\ xs = \text{imap } g\ xs) = (\forall x \in \text{iset } xs. (f\ x) = (g\ x))$ 
by (induct xs auto)

```

```

lemma imap-cong [fundef-cong]:
  assumes  $xs = ys$ 

```



$(\forall x. x \in \text{iset } ys \longrightarrow f x = g x)$   
**shows**  $\text{imap } f xs = \text{imap } g ys$   
**using** *assms* **by** *simp*

**lemma** *imap-injective*:  
**assumes**  $\text{imap } f xs = \text{imap } f ys$   
 $\text{inj } f$   
**shows**  $xs = ys$   
**using** *assms* **by** (*meson injD interval.inj-map*)

**lemma** *inj-imap-eq-imap [simp]*:  
**assumes**  $\text{inj } f$   
**shows**  $(\text{imap } f xs = \text{imap } f ys) = (xs = ys)$   
**using** *assms* **by** (*blast dest: imap-injective*)

**lemma** *inj-imapI*:  
**assumes**  $\text{inj } f$   
**shows**  $\text{inj } (\text{imap } f)$   
**using** *assms interval.inj-map* **by** *blast*

**lemma** *inj-imapD*:  
**assumes**  $\text{inj } (\text{imap } f)$   
**shows**  $\text{inj } f$   
**using** *assms* **by** (*metis inj-def interval.map(1) interval.simps(1)*)

**lemma** *inj-imap[iff]*:  
 $\text{inj } (\text{imap } f) = \text{inj } f$   
**by** (*blast dest: inj-imapD intro: inj-imapI*)

**lemma** *imap-idI*:  
**assumes**  $(\forall x. x \in \text{iset } xs \longrightarrow f x = x)$   
**shows**  $\text{imap } f xs = xs$   
**using** *assms* **by** (*induct xs*) *auto*

**lemma** *imap-is-INil-conv[iff]*:  
 $(\text{imap } f xs = \langle x \rangle) = (\exists y. xs = \langle y \rangle \wedge f y = x)$   
**proof** (*cases xs*)  
**case** (*INil x1*)  
**then show** *?thesis* **by** *simp*  
**next**  
**case** (*ICons x21 x22*)  
**then show** *?thesis* **by** *simp*  
**qed**

**lemma** *INil-is-imap-conv [iff]*:  
 $(\langle x \rangle = \text{imap } f xs) = (\exists y. \langle y \rangle = xs \wedge f y = x)$   
**proof** (*cases xs*)  
**case** (*INil x1*)  
**then show** *?thesis* **by** *auto*  
**next**

**case** (*ICons* *x21* *x22*)  
**then show** *?thesis* **by** *auto*  
**qed**

**lemma** *imap-eq-ICons-conv*:  
 $(\text{imap } f \text{ } xs = y \odot ys) = (\exists z \text{ } zs. xs = z \odot zs \wedge f \text{ } z = y \wedge \text{imap } f \text{ } zs = ys)$   
**by** (*cases* *xs*) *auto*

**lemma** *ICons-eq-imap-conv*:  
 $(y \odot ys = \text{imap } f \text{ } xs) = (\exists z \text{ } zs. z \odot zs = xs \wedge f \text{ } z = y \wedge ys = \text{imap } f \text{ } zs)$   
**by** (*cases* *xs*) *auto*

**lemma** *ex-imap-conv*:  
 $(\exists xs. ys = \text{imap } f \text{ } xs) = (\forall y \in \text{iset } ys. \exists x. y = f \text{ } x)$   
**by** (*induct* *ys*) (*auto simp add: ICons-eq-imap-conv*)

**functor** *imap*: *imap*  
**by** (*simp-all add: id-def*)

**declare** *imap.id* [*simp*]

**lemma** *ifirst-imap*:  
 $\text{ifirst } (\text{imap } f \text{ } xs) = f \text{ } (\text{ifirst } xs)$   
**by** (*cases* *xs*) *simp-all*

**lemma** *ilast-imap*:  
 $\text{ilast } (\text{imap } f \text{ } xs) = f \text{ } (\text{ilast } xs)$   
**proof** (*cases* *xs* *rule: irev-cases*)  
**case** (*INil* *x*)  
**then show** *?thesis* **by** *simp*  
**next**  
**case** (*snoc* *ys* *y*)  
**then show** *?thesis* **by** (*simp add: iapp-inth*)  
**qed**

**lemma** *imap-tail*:  
**shows**  $\text{imap } f \text{ } (\text{suffix } 1 \text{ } xs) = (\text{suffix } 1 \text{ } (\text{imap } f \text{ } xs))$   
**by** (*cases* *xs*) *simp-all*

**lemma** *imap-eq-imp-ilen-eq*:  
**assumes**  $\text{imap } f \text{ } xs = \text{imap } g \text{ } ys$   
**shows**  $\text{ilen } xs = \text{ilen } ys$   
**using** *assms*  
**proof** (*induct* *ys* *arbitrary: xs*)  
**case** (*INil* *x*)  
**then show** *?case* **by** *auto*

```

next
case (ICons x1a ys)
then show ?case by (metis ilen-imap)
qed

lemma iset-imap [simp]:
  iset (imap f xs) = f'(iset xs)
by (induct xs) auto

lemma imap-inj-on:
  assumes imap:  $\text{imap } f \text{ } xs = \text{imap } f \text{ } ys$  and
    inj:  $\text{inj-on } f \text{ } (\text{iset } xs \cup \text{iset } ys)$ 
  shows  $xs = ys$ 
using imap-eq-imp-ilen-eq[OF imap] assms
proof (induction rule: interval-induct2)
case (INil x y)
then show ?case by auto
next
case (ICons x xs y ys)
then show ?case
by (metis (mono-tags, hide-lams) UnI1 UnI2 inj-on-def interval.inj-map-strong)
qed

lemma inj-on-imap-eq-imap:
  assumes inj-on  $f \text{ } (\text{iset } xs \cup \text{iset } ys)$ 
  shows  $(\text{imap } f \text{ } xs = \text{imap } f \text{ } ys) = (xs = ys)$ 
using assms by (blast dest:imap-inj-on)

lemma inj-on-imapI:
  assumes inj-on  $f \text{ } (\bigcup (\text{iset } 'A))$ 
  shows  $\text{inj-on } (\text{imap } f) \text{ } A$ 
using assms by (blast intro:inj-onI dest:inj-onD imap-inj-on)

lemma imap-prefix:
  assumes  $k \leq \text{ilen } xs$ 
  shows  $\text{imap } f \text{ } (\text{prefix } k \text{ } xs) = \text{prefix } k \text{ } (\text{imap } f \text{ } xs)$ 
using assms
proof (induction xs arbitrary: k)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
  proof (cases k)
  case 0
  then show ?thesis by auto
  next
  case (Suc nat)
  then show ?thesis using ICons.IH ICons.prem by auto
  qed
qed

```

qed

```
lemma imap-suffix:
  assumes  $k \leq \text{ilen } xs$ 
  shows  $\text{imap } f (\text{suffix } k \text{ } xs) = \text{suffix } k (\text{imap } f \text{ } xs)$ 
using assms
proof (induction xs arbitrary: k)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
proof (cases k)
case 0
then show ?thesis by auto
next
case (Suc nat)
then show ?thesis using ICons.IH ICons.prem1 by auto
qed
qed
```

### 1.2.9 index sequence

```
lemma idx-less:
  assumes  $\text{index-sequence } x \text{ } idx$ 
            $n+k < \text{ilen } idx$ 
  shows  $\text{inth } idx \text{ } n < \text{inth } idx (\text{Suc}(n+k))$ 
using index-sequence-def assms by (induct k) auto
```

```
lemma idx-less-eq:
  assumes  $\text{index-sequence } x \text{ } l$ 
            $k \leq j$ 
            $j \leq \text{ilen } l$ 
  shows  $\text{inth } l \text{ } k \leq \text{inth } l \text{ } j$ 
using assms
proof (cases k=j)
show  $\text{index-sequence } x \text{ } l \implies k \leq j \implies j \leq \text{ilen } l \implies k = j \implies \text{inth } l \text{ } k \leq \text{inth } l \text{ } j$ 
  by blast
show  $\text{index-sequence } x \text{ } l \implies k \leq j \implies j \leq \text{ilen } l \implies k \neq j \implies \text{inth } l \text{ } k \leq \text{inth } l \text{ } j$ 
  by (metis Suc-le-lessD idx-less le-SucE le-eq-less-or-eq le-zero-eq
    ordered-cancel-comm-monoid-diff-class.add-diff-inverse zero-induct)
qed
```

```
lemma idx-mono:
  assumes  $\text{index-sequence } x \text{ } l$ 
  shows  $\text{mono } (\lambda x. \text{inth } l \text{ } x)$ 
proof -
  have 1:  $\forall x \ y. x \leq y \longrightarrow \text{inth } l \text{ } x \leq \text{inth } l \text{ } y$ 
```

```

proof
  fix x
  show  $\forall y \geq x. \text{inth } l \ x \leq \text{inth } l \ y$ 
  proof
    fix y
    show  $x \leq y \longrightarrow \text{inth } l \ x \leq \text{inth } l \ y$ 
    proof -
      have 2:  $x \leq y \wedge y \leq \text{ilen } l \implies \text{inth } l \ x \leq \text{inth } l \ y$ 
        using assms idx-less-eq by blast
      have 3:  $x \leq y \wedge x > \text{ilen } l \implies \text{inth } l \ x \leq \text{inth } l \ y$ 
        using assms inth-last-stutter
        by (metis 2 le-cases less-imp-add-positive ordered-cancel-comm-monoid-diff-class.add-diff-inverse)
      have 4:  $x \leq y \wedge x \leq \text{ilen } l \wedge y > \text{ilen } l \implies \text{inth } l \ x \leq \text{inth } l \ y$ 
        by (metis assms idx-less-eq inth-last-stutter less-imp-add-positive order-refl)
      show ?thesis
        using 2 3 4 not-less by blast
    qed
  qed
  qed
  show ?thesis
    by (simp add: 1 monoI)
qed

```

```

lemma idx-less-last :
  assumes index-sequence x idx
    i < ilen idx
    i + (ilen idx - (i + 1)) < ilen idx
  shows inth idx i < inth idx (Suc(i + (ilen idx - (i + 1))))
  using assms idx-less by blast

```

```

lemma idx-less-last-1:
  assumes index-sequence x idx
    i < ilen idx
  shows inth idx i < inth idx (ilen idx)
  using assms idx-less-last by auto

```

```

lemma idx-greater-first:
  assumes index-sequence x idx
    0 < i
    i ≤ ilen idx
  shows x < inth idx i
  using assms
  proof -
    have 1: inth idx 0 = x
      using assms by (simp add: index-sequence-def)
    have 2:  $\forall i. 0 < i \wedge i \leq \text{ilen } idx \longrightarrow (\text{inth } idx \ 0) < (\text{inth } idx \ i)$ 
      proof
        fix i
        show  $0 < i \wedge i \leq \text{ilen } idx \longrightarrow \text{inth } idx \ 0 < \text{inth } idx \ i$ 

```

by (meson Suc-leI assms(1) dual-order.strict-trans1 index-sequence-def idx-less-eq)  
 qed  
 show ?thesis  
 using 1 2 assms(2) assms(3) by blast  
 qed

**lemma** *idx-ICons*:  
 $index\text{-}sequence\ y\ (x \odot ls) =$   
 $(x=y \wedge x < inth\ ls\ 0 \wedge index\text{-}sequence\ (inth\ ls\ 0)\ ls)$   
 using less-Suc-eq-0-disj by (simp add: index-sequence-def) auto

**lemma** *idx-shift-mono*:  
 $mono\ (shift\ k)$   
 by (simp add: shift-def mono-def)

**lemma** *idx-expand*:  
**assumes**  $index\text{-}sequence\ 0\ l$   
 $(inth\ l\ (ilen\ l)) = (ilen\ xs)$   
 $i < (ilen\ l)$   
**shows**  $(inth\ l\ i) \leq (inth\ l\ (i+1)) \wedge (inth\ l\ (i+1)) \leq (ilen\ xs)$   
 using assms  
 by (metis Suc-eq-plus1 Suc-lessI add.right-neutral idx-less idx-less-last-1  
 less-imp-le-nat order-refl)

**lemma** *idx-shift-idx* [simp]:  
 $(index\text{-}sequence\ (x+k)\ (imap\ (shift\ k)\ idx)) = (index\text{-}sequence\ x\ idx)$   
 by (simp add: shift-def index-sequence-def inth-imap)

**lemma** *idx-shiftn* :  
**assumes**  $index\text{-}sequence\ k\ lsk$   
**shows**  $index\text{-}sequence\ 0\ (imap\ (shiftn\ k)\ lsk) \wedge k \leq (inth\ lsk\ 0)$   
 using assms  
**proof** (auto simp add: index-sequence-def shiftn-def inth-imap )  
**show**  $\bigwedge n. \forall n < ilen\ lsk. inth\ lsk\ n < inth\ lsk\ (Suc\ n) \implies$   
 $k = inth\ lsk\ 0 \implies$   
 $n < ilen\ lsk \implies inth\ lsk\ n - inth\ lsk\ 0 < inth\ lsk\ (Suc\ n) - inth\ lsk\ 0$   
 by (metis assms diff-less-mono idx-greater-first le-eq-less-or-eq neq0-conv)  
 qed

**lemma** *idx-shiftn-a* :  
**assumes**  $index\text{-}sequence\ 0\ (imap\ (shiftn\ k)\ lsk)$   
 $k \leq (inth\ lsk\ 0)$   
**shows**  $index\text{-}sequence\ k\ lsk$   
 using assms  
 by (auto simp add: index-sequence-def shiftn-def inth-imap)

**lemma** *idx-shiftn-b* :  
 $index\text{-}sequence\ k\ lsk = (index\text{-}sequence\ 0\ (imap\ (shiftn\ k)\ lsk) \wedge k \leq (inth\ lsk\ 0))$   
 using idx-shiftn idx-shiftn-a by blast

**lemma** *idx-shiftm-c* :

*index-sequence (inth lsk 0) lsk = index-sequence 0 (imap (shiftm (inth lsk 0)) lsk)*

**using** *idx-shiftm-b* **by** *blast*

**lemma** *lsk-ls* :

*(index-sequence k (lsk)  $\wedge$  lsk = imap (shift k) ls  $\wedge$  index-sequence 0 (ls) ) =*  
*(index-sequence k (lsk)  $\wedge$  ls = imap (shiftm k) lsk  $\wedge$  index-sequence 0 (ls) )*

**proof** (*simp add: interval-eq-inth-eq index-sequence-def shift-def shiftm-def inth-imap*)

**show** (*inth lsk 0 = k  $\wedge$*

*( $\forall n < \text{ilen lsk}. \text{inth lsk } n < \text{inth lsk (Suc n)}$ )  $\wedge$*

*ilen lsk = ilen ls  $\wedge$*

*( $\forall i \leq \text{ilen lsk}. \text{inth lsk } i = \text{inth ls } i + k$ )  $\wedge$*

*inth ls 0 = 0  $\wedge$  ( $\forall n < \text{ilen ls}. \text{inth ls } n < \text{inth ls (Suc n)}$ )) =*

*(inth lsk 0 = k  $\wedge$*

*( $\forall n < \text{ilen lsk}. \text{inth lsk } n < \text{inth lsk (Suc n)}$ )  $\wedge$*

*ilen ls = ilen lsk  $\wedge$*

*( $\forall i \leq \text{ilen ls}. \text{inth ls } i = \text{inth lsk } i - k$ )  $\wedge$*

*inth ls 0 = 0  $\wedge$  ( $\forall n < \text{ilen ls}. \text{inth ls } n < \text{inth ls (Suc n)}$ )) (is ?L=?R)*

**proof** *rule*

**show** *?L  $\implies$  ?R*

**by** (*metis (no-types, lifting) add-diff-cancel-right'*)

**show** *?R  $\implies$  ?L*

**by** (*metis Suc-le-lessD add.commute diff-is-0-eq nat-le-linear not0-implies-Suc*  
*ordered-cancel-comm-monoid-diff-class.add-diff-inverse zero-order(3)*)

**qed**

**qed**

**lemma** *idx-link-shiftm*:

*(index-sequence k (lsk)  $\wedge$  ls = imap (shiftm k) lsk ) =*  
*(index-sequence k (lsk)  $\wedge$  ls = imap (shiftm k) lsk  $\wedge$*   
*index-sequence 0 (ls)  $\wedge$  (ilen ls) =(ilen lsk))*

**using** *idx-shiftm* **using** *ilen-imap* **by** *blast*

**lemma** *idx-link*:

*(lsk = imap (shift k) ls  $\wedge$  index-sequence 0 (ls) ) =*  
*(lsk = imap (shift k) ls  $\wedge$  index-sequence k (lsk)  $\wedge$  index-sequence 0 (ls)  $\wedge$*   
*(ilen ls) =(ilen lsk))*

**by** (*metis add.left-neutral idx-shift-idx ilen-imap*)

**lemma** *idx-bound-0* :

**assumes** *index-sequence 0 ls*

*inth ls (ilen ls) = ilen (suffix k xs)*

*i  $\leq$  ilen ls*

**shows** *inth ls i  $\leq$  ilen (suffix k xs)*

**using** *assms*

**by** (*metis eq-iff idx-less-last-1 le-neq-implies-less less-imp-le-nat*)

**lemma** *idx-bound-1*:

*(index-sequence 0 (ls)  $\wedge$  (inth (ls) (ilen (ls)))) = (ilen (suffix k xs))) =*  
*(index-sequence 0 (ls)  $\wedge$  (inth (ls) (ilen (ls)))) = (ilen (suffix k xs))  $\wedge$*

$(\forall i. (i \leq \text{ilen } ls) \longrightarrow ((\text{inth } ls \ i) \leq (\text{ilen } (\text{suffix } k \ xs))))$  )  
**using** *idx-bound-0* **by** *blast*

**lemma** *idx-less-equal*:

**assumes** *index-sequence 0 l*  
 $(\text{inth } l \ (\text{ilen } l)) = \text{ilen } xs$   
 $i \leq \text{ilen } l$   
 $n \leq \text{ilen } l$

**shows**  $\forall j. j \leq n \longrightarrow \text{inth } l \ j \leq \text{inth } l \ n$

**using** *assms*

**using** *idx-less-eq* **by** *blast*

**lemma** *idx-less-than*:

**assumes** *index-sequence 0 l*  
 $(\text{inth } l \ (\text{ilen } l)) = \text{ilen } xs$   
 $i \leq \text{ilen } l$   
 $n \leq \text{ilen } l$

**shows**  $\forall j. j > n \wedge j \leq \text{ilen } l \longrightarrow \text{inth } l \ j > \text{inth } l \ n$

**by** (*meson Suc-leI assms idx-less-equal index-sequence-def less-le-trans*)

**lemma** *idx-sub*:

**assumes**  $k \leq n$   
 $n \leq \text{ilen } l$   
*index-sequence 0 l*

**shows** *index-sequence* (*inth l k*) (*sub k n l*)

**proof** –

**have** 1: *index-sequence* (*inth l k*) (*sub k n l*) =  
 $((\text{inth } (\text{sub } k \ n \ l) \ 0) = (\text{inth } l \ k) \wedge$   
 $(\forall i. i < \text{ilen}(\text{sub } k \ n \ l) \longrightarrow (\text{inth } (\text{sub } k \ n \ l) \ i) < (\text{inth } (\text{sub } k \ n \ l) \ (\text{Suc } i))))$

**using** *index-sequence-def* **by** *auto*

**have** 2:  $(\text{inth } (\text{sub } k \ n \ l) \ 0) = (\text{inth } l \ k)$

**using** *assms ifirst-sub* **by** *auto*

**have** 3:  $\text{ilen}(\text{sub } k \ n \ l) = (n - k)$

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

**have** 4:  $(\forall i. i < \text{ilen}(\text{sub } k \ n \ l) \longrightarrow (\text{inth } (\text{sub } k \ n \ l) \ i) < (\text{inth } (\text{sub } k \ n \ l) \ (\text{Suc } i)))$

**proof**

**fix** *i*

**show**  $i < \text{ilen } (\text{sub } k \ n \ l) \longrightarrow \text{inth } (\text{sub } k \ n \ l) \ i < \text{inth } (\text{sub } k \ n \ l) \ (\text{Suc } i)$

**proof** –

**have** 41:  $i < (n - k) \longrightarrow \text{inth } (\text{sub } k \ n \ l) \ i = \text{inth } l \ (k + i)$

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

**have** 42:  $i < (n - k) \longrightarrow \text{inth } (\text{sub } k \ n \ l) \ (\text{Suc } i) = \text{inth } l \ (k + (\text{Suc } i))$

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

**have** 43:  $i < (n - k) \longrightarrow \text{inth } l \ (k + i) < \text{inth } l \ (k + (\text{Suc } i))$

**using** *assms*

**using** *index-sequence-def* **by** *auto*

**show** *?thesis* **using** 3 41 42 43 **by** *auto*



```

qed
qed
show ?thesis by (simp add: 1 2 4)
qed

```

lemma *idx-split*:

```

assumes  $n \leq \text{ilen } l$ 
shows  $\text{index-sequence } 0 \ l =$ 
       $(\text{index-sequence } 0 \ (\text{prefix } n \ l) \wedge \text{index-sequence } (\text{inth } l \ n) \ (\text{suffix } n \ l))$ 
proof -
  have 1:  $\text{index-sequence } 0 \ l \longrightarrow$ 
       $\text{index-sequence } 0 \ (\text{prefix } n \ l) \wedge \text{index-sequence } (\text{inth } l \ n) \ (\text{suffix } n \ l)$ 
    using idx-sub using assms index-sequence-def by auto
  have 2:  $\text{index-sequence } 0 \ (\text{prefix } n \ l) =$ 
       $( (\text{inth } (\text{prefix } n \ l) \ 0) = 0 \wedge$ 
         $(\forall i. i < \text{ilen}(\text{prefix } n \ l) \longrightarrow (\text{inth } (\text{prefix } n \ l) \ i) < (\text{inth } (\text{prefix } n \ l) \ (\text{Suc } i))))$ 
    using index-sequence-def by blast
  have 3:  $( (\text{inth } (\text{prefix } n \ l) \ 0) = 0 \wedge$ 
       $(\forall i. i < \text{ilen}(\text{prefix } n \ l) \longrightarrow (\text{inth } (\text{prefix } n \ l) \ i) < (\text{inth } (\text{prefix } n \ l) \ (\text{Suc } i)))) =$ 
       $( (\text{inth } l \ 0) = 0 \wedge$ 
         $(\forall i. i < n \longrightarrow (\text{inth } l \ i) < (\text{inth } l \ (\text{Suc } i))))$ 
    using assms by auto
  have 4:  $\text{index-sequence } (\text{inth } l \ n) \ (\text{suffix } n \ l) =$ 
       $( (\text{inth } (\text{suffix } n \ l) \ 0) = (\text{inth } l \ n) \wedge$ 
         $(\forall i. i < \text{ilen}(\text{suffix } n \ l) \longrightarrow (\text{inth } (\text{suffix } n \ l) \ i) < (\text{inth } (\text{suffix } n \ l) \ (\text{Suc } i))))$ 
    using index-sequence-def by blast
  have 5:  $( (\text{inth } (\text{suffix } n \ l) \ 0) = (\text{inth } l \ n) \wedge$ 
       $(\forall i. i < \text{ilen}(\text{suffix } n \ l) \longrightarrow (\text{inth } (\text{suffix } n \ l) \ i) < (\text{inth } (\text{suffix } n \ l) \ (\text{Suc } i)))) =$ 
       $( (\text{inth } l \ n) = (\text{inth } l \ n) \wedge$ 
         $(\forall i. i < \text{ilen } l - n \longrightarrow (\text{inth } l \ (i+n)) < (\text{inth } l \ ((\text{Suc } i)+n))))$ 
    by (simp add: add commute)
  have 6:  $(\forall i. i < \text{ilen } l - n \longrightarrow (\text{inth } l \ (i+n)) < (\text{inth } l \ ((\text{Suc } i)+n))) =$ 
       $(\forall i. n \leq (i+n) \wedge (i+n) < \text{ilen } l \longrightarrow (\text{inth } l \ (i+n)) < (\text{inth } l \ ((\text{Suc } (i+n)))))$ 
    by auto
  have 61:  $(\forall i. n \leq (i+n) \wedge (i+n) < \text{ilen } l \longrightarrow (\text{inth } l \ (i+n)) < (\text{inth } l \ ((\text{Suc } (i+n))))) =$ 
       $(\forall j. n \leq j \wedge j < \text{ilen } l \longrightarrow (\text{inth } l \ j) < (\text{inth } l \ ((\text{Suc } j))))$ 
    by (metis diff-add)
  have 7:  $(\text{index-sequence } 0 \ (\text{prefix } n \ l) \wedge \text{index-sequence } (\text{inth } l \ n) \ (\text{suffix } n \ l)) \longrightarrow$ 
       $(\text{inth } l \ 0) = 0 \wedge (\forall i. i < n \longrightarrow (\text{inth } l \ i) < (\text{inth } l \ (\text{Suc } i)))$ 
    using 2 3 by blast
  have 8:  $(\text{index-sequence } 0 \ (\text{prefix } n \ l) \wedge \text{index-sequence } (\text{inth } l \ n) \ (\text{suffix } n \ l)) \longrightarrow$ 
       $(\forall i. n \leq i \wedge i < \text{ilen } l \longrightarrow (\text{inth } l \ i) < (\text{inth } l \ ((\text{Suc } i))))$ 
    using 4 5 6 61 by blast
  have 9:  $(\text{index-sequence } 0 \ (\text{prefix } n \ l) \wedge \text{index-sequence } (\text{inth } l \ n) \ (\text{suffix } n \ l)) \longrightarrow$ 
       $(\text{inth } l \ 0) = 0 \wedge (\forall i. i < \text{ilen } l \longrightarrow (\text{inth } l \ i) < (\text{inth } l \ (\text{Suc } i)))$ 
    using 7 8 not-le by blast
  have 10:  $(\text{index-sequence } 0 \ (\text{prefix } n \ l) \wedge \text{index-sequence } (\text{inth } l \ n) \ (\text{suffix } n \ l)) \longrightarrow$ 
       $\text{index-sequence } 0 \ l$ 
    using 9 index-sequence-def by blast
  from 10 1 show ?thesis by blast

```

qed

**lemma** *idx-suffixa*:

**assumes**  $n \leq \text{ilen } l$

$\text{index-sequence } (\text{inth } l \ n) \ (\text{suffix } n \ l)$

**shows**  $\text{index-sequence } 0 \ ((\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l)))$

**using** *assms idx-shiftn* **by** *blast*

**lemma** *idx-greater*:

**assumes**  $\text{index-sequence } k \ l$

**shows**  $(\forall i. i \leq \text{ilen } l \longrightarrow k \leq (\text{inth } l \ i))$

**by** (*metis assms eq-iff index-sequence-def idx-greater-first less-imp-le neq0-conv*)

**lemma** *idx-suffixb*:

**assumes**  $n \leq \text{ilen } l$

$\text{index-sequence } 0 \ ((\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l)))$

**shows**  $\text{index-sequence } (\text{inth } l \ n) \ (\text{suffix } n \ l)$

**by** (*metis assms idx-shiftn-c ilast-ifirst ilast-prefix*)

**lemma** *idx-suffix*:

**assumes**  $n \leq \text{ilen } l$

**shows**  $\text{index-sequence } (\text{inth } l \ n) \ (\text{suffix } n \ l) =$

$\text{index-sequence } 0 \ ((\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l)))$

**using** *assms idx-shiftn idx-suffixb* **by** *blast*

**lemma** *idx-ifirst*:

**assumes**  $\text{index-sequence } 0 \ (x1a \odot l)$

**shows**  $x1a < \text{ifirst}(l)$

**by** (*metis assms idx-ICons*)

**lemma** *idx-expand1*:

$(\text{index-sequence } x1a \ (x1a \odot l)) = (x1a < \text{ifirst } l \wedge \text{index-sequence } (\text{ifirst } l) \ l)$

**using** *inth-zero-ifirst less-Suc-eq-0-disj* **by** (*auto simp add: index-sequence-def*)

**lemma** *idx-ilen-leq-ilast-ifirst*:

**assumes**  $\text{index-sequence } (\text{ifirst } l) \ l$

**shows**  $\text{ilen } (l) \leq (\text{ilast } l - \text{ifirst } l)$

**using** *assms*

**proof**

(*induct*  $l$ )

**case** (*INil*  $x$ )

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

**next**

**case** (*ICons*  $x1a \ l$ )

**then show** *?case*

**proof** –

**have** 1:  $\text{ilen } (x1a \odot l) = \text{ilen } l + 1$

**by** *simp*

**have** 2:  $\text{index-sequence } (\text{ifirst } l) \ l$

```

  using ICons.premis idx-expand1 by auto
  have 3: ilast (x1a  $\odot$  l) = ilast l
  by simp
  have 4: ifirst (x1a  $\odot$  l) = x1a
  by simp
  have 5: x1a < ifirst l
  using ICons.premis idx-expand1 by auto
  have 6: ilen l  $\leq$  ilast l - ifirst l
  using 2 ICons.hyps by blast
  have 7: ilen l + 1  $\leq$  (ilast l - ifirst l) + 1
  using 6 add-le-cancel-right by blast
  have 8: (ilast l - ifirst l) + 1  $\leq$  ilast l - x1a
  using 5 6 le-less-linear by fastforce
  show ?thesis using 7 8 by auto
qed
qed

```

**lemma** *idx-ilen-leq*:

**assumes** *index-sequence* (ifirst l) l

$ilast(l) \leq ilen\ xs$

**shows**  $ilen\ (l) \leq ilen\ (sub\ (ifirst\ l)\ (ilast\ l)\ xs)$

**proof** –

have 1: ifirst l  $\leq$  ilast l

using *assms* by (metis *eq-iff* *gr0I* *index-sequence-def* *idx-less-last-1* *less-imp-le-nat*)

have 2:  $ilen\ (sub\ (ifirst\ l)\ (ilast\ l)\ xs) = (ilast\ l - ifirst\ l)$

using 1 *assms* *ilen-sub* by blast

have 3:  $ilen\ (l) \leq (ilast\ l - ifirst\ l)$

using *assms* *idx-ilen-leq-ilast-ifirst* by blast

**show** ?thesis using 2 3 by *linarith*

qed

**lemma** *idx-shiftm-sub-inth*:

**assumes** *index-sequence* 0 l

$(inth\ l\ (ilen\ l)) = ilen\ xs$

$k \leq n$

$n \leq ilen\ l$

**shows**  $\forall j. j \leq n-k \longrightarrow$

$inth\ (imap\ (shiftm\ (inth\ l\ k))\ (sub\ k\ n\ l))\ j = inth\ l\ (k+j) - (inth\ l\ k)$

**by** (*simp* *add*: *Nat.le-diff-conv2* *assms* *inth-imap* *shiftm-def*)

**lemma** *idx-shiftm-suffix-inth*:

**assumes** *index-sequence* 0 l

$(inth\ l\ (ilen\ l)) = ilen\ xs$

$n \leq ilen\ l$

**shows**  $\forall j. j \leq ilen\ l - n \longrightarrow$

$inth\ (imap\ (shiftm\ (inth\ l\ n))\ (suffix\ n\ l))\ j = inth\ l\ (n+j) - (inth\ l\ n)$

**using** *assms* **by** (*metis* *inth-imap* *inth-suffix* *shiftm-def*)

### 1.2.10 upt

**lemma** *upt-rec*[*code*]:

$[i.. \leq j] = (\text{if } i < j \text{ then } i \odot [Suc\ i.. \leq j] \text{ else } \langle j \rangle)$

**by** (*induct j*) *auto*

**lemma** *upt-conv-INil* [*simp*]:

**assumes**  $j < i$

**shows**  $[i.. \leq j] = \langle j \rangle$

**using** *assms*

**by** (*metis upt.simps(1) upt.simps(2) Suc-leD less-Suc-eq-0-disj not-le*)

**lemma** *upt-same*:

$[i.. \leq i] = \langle i \rangle$

**by** (*metis upt.simps(1) upt.simps(2) less-Suc-eq less-Suc-eq-0-disj not-le*)

**lemma** *upt-eq-INil-conv*[*simp*]:

$([i.. \leq j] = \langle j \rangle) = (j \leq i)$

**by** (*simp add: upt-rec*)

**lemma** *upt-eq-ICons-conv*:

$([i.. \leq j] = x \odot xs) = (i < j \wedge i = x \wedge [Suc\ i.. \leq j] = xs)$

**using** *upt-rec* **by** (*induct j arbitrary: x xs*) *auto*

**lemma** *upt-suc-iapp*:

**assumes**  $i \leq j$

**shows**  $[i.. \leq (Suc\ j)] = [i.. \leq j] \oplus \langle (Suc\ j) \rangle$

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

**lemma** *upt-conv-ICons*:

**assumes**  $i < j$

**shows**  $[i.. \leq j] = i \odot [(Suc\ i).. \leq j]$

**using** *assms* **by** (*simp add: upt-rec*)

**lemma** *upt-conv-ICons-ICons*:

$(m \odot n \odot ns = [m.. \leq q]) = (n \odot ns = [(Suc\ m).. \leq q])$

**proof** (*cases m ≤ q*)

**case** *True*

**then show** *?thesis* **by** (*simp add: upt-rec*)

**next**

**case** *False*

**then show** *?thesis* **by** *auto*

**qed**

**lemma** *upt-add-eq-iapp*:

**assumes**  $i \leq j$

$k > 0$

**shows**  $[i.. \leq j+k] = [i.. \leq j] \oplus [Suc\ j.. \leq j+k]$

**using** *assms*

**proof**

(*induct k*)

```

case 0
then show ?case by blast
next
case (Suc k)
then show ?case using Suc-less-eq le-simps(2) by auto
qed

```

```

lemma upt-ilen:
  ilen [i.. $\leq$ j] = j - i
by (induct j) (auto simp add: Suc-diff-le)

```

```

lemma upt-inth-help:
  inth [i.. $\leq$ i + k] k = i + k
proof
  (induct k arbitrary: i)
  case 0
  then show ?case by (simp add: upt-same)
  next
  case (Suc k)
  then show ?case
  by (metis upt-rec add-Suc-shift inth-Suc less-add-same-cancel1 zero-less-Suc)
qed

```

```

lemma upt-inth:
  assumes i + k  $\leq$  j
  shows (inth [i.. $\leq$ j] k) = i + k
  using assms
proof
  (induct j arbitrary: k i)
  case 0
  then show ?case by simp
  next
  case (Suc j)
  then show ?case
  by (metis upt.upt-Suc Nat.le-diff-conv2 add commute add-leD1
    iapp-inth le-SucE upt-ilen upt-inth-help)
qed

```

```

lemma upt-ifirst:
  assumes i  $\leq$  j
  shows ifirst [i.. $\leq$ j] = i
  using assms by (simp add: upt-rec)

```

```

lemma upt-ilast:
  ilast [i.. $\leq$ j] = j
by (metis add-diff-inverse-nat INil-ilen order-refl upt-conv-INil upt-ilen upt-inth)

```

```

lemma prefix-upt:
  assumes i + m  $\leq$  n
  shows prefix m [i.. $\leq$ n] = [i.. $\leq$ i + m]

```

```

using assms
proof
  (induct m arbitrary: i)
  case 0
  then show ?case by (simp add: upt-inth upt-same)
  next
  case (Suc m)
  then show ?case using upt-rec by auto
qed

lemma suffix-upt:
   $\text{suffix } m \ [i..j] = [i+m..j]$ 
proof
  (induct m arbitrary: i j)
  case 0
  then show ?case by simp
  next
  case (Suc j)
  then show ?case using upt-rec
  by (metis add-Suc-shift suffix-suc not-less-eq not-less-iff-gr-or-eq suffix.simps(1))
qed

```

```

lemma imap-suc-upt:
   $\text{imap } \text{Suc} \ [m..n] = [\text{Suc } m.. \text{Suc } n]$ 
proof
  (induct n arbitrary: m)
  case 0
  then show ?case by simp
  next
  case (Suc n)
  then show ?case by simp
qed

```

```

lemma imap-add-upt:
   $\text{imap } (\lambda i. i + n) \ [0..m] = [n..m+n]$ 
proof
  (induct m)
  case 0
  then show ?case by (simp add: upt-same)
  next
  case (Suc m)
  then show ?case by simp
qed

```

### 1.2.11 Set

```

lemma iset-iapp [simp]:
   $\text{iset } (xs \ominus ys) = (\text{iset } xs \cup \text{iset } ys)$ 
by (induct xs) auto

```

**lemma** *interval-finite-iset* [iff]:  
   *finite (iset (xs:: 'a interval) )*  
**by** (*induct xs*) *auto*

**lemma** *interval-hd-in-iset* [simp]:  
    $x \in \text{iset } (x \odot xs)$   
**by** *simp*

**lemma** *iset-subset-ICons*:  
    $\text{iset } xs \subseteq \text{iset } (x \odot xs)$   
**by** *auto*

**lemma** *iset-IConsD*:  
**assumes**  $y \in \text{iset } (x \odot xs)$   
**shows**  $y = x \vee y \in \text{iset } xs$   
**using** *assms* **by** *auto*

**lemma** *exists-ICons*:  
 $(\exists \text{ } ys \in \text{iset } (x \odot xs)). P \text{ } ys) \longleftrightarrow$   
 $(P \text{ } x \wedge (\exists \text{ } ys \in \text{iset } xs. P \text{ } ys)) \vee (\neg P \text{ } x \wedge (\exists \text{ } ys \in \text{iset } xs. P \text{ } ys)) \vee$   
 $(P \text{ } x \wedge (\forall \text{ } ys \in \text{iset } xs. \neg P \text{ } ys))$   
**by** *auto*

**lemma** *interval-iset-nonempty*:  
    $\text{iset } xs \neq \{\}$   
**by** (*induct xs*) *auto*

**lemma** *iset-irev* [simp]:  
    $\text{iset } (\text{irev } xs) = \text{iset } xs$   
**by** (*induct xs*) *auto*

**lemma** *split-interval*:  
**assumes**  $x \in \text{iset } xs$   
**shows**  $\exists \text{ } ys \text{ } zs. xs = ys \ominus (x \odot zs) \vee xs = \langle x \rangle \vee xs = x \odot zs \vee xs = ys \ominus \langle x \rangle$   
**using** *assms*  
**proof** (*induct xs*)  
**case** (*INil*  $x$ )  
**then show** ?*case* **by** *simp*  
**next**  
**case** (*ICons*  $x1a \text{ } xs$ )  
**then show** ?*case*  
**by** (*metis iapp-INil interval.inject(2) interval.set-cases iapp-assoc*)  
**qed**

**lemma** *interval-in-iset-conv-decomp*:  
    $x \in \text{iset } xs =$   
 $(\exists \text{ } ys \text{ } zs. xs = ys \ominus (x \odot zs) \vee xs = \langle x \rangle \vee xs = x \odot zs \vee xs = ys \ominus \langle x \rangle)$   
**by** (*auto elim: split-interval*)

**lemma** *split-interval-first*:

**assumes**  $x \in \text{iset } xs$

**shows**  $\exists \text{ys zs. } xs = \text{ys} \ominus (x \odot zs) \wedge x \notin \text{iset } \text{ys} \vee xs = \langle x \rangle \vee xs = x \odot zs \vee$   
 $xs = \text{ys} \ominus \langle x \rangle \wedge x \notin \text{iset } \text{ys}$

**using** *assms*

**proof** (*induct xs*)

**case** (*INil x*)

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

**next**

**case** (*ICons x1a xs*)

**then show** *?case*

**proof** (*cases x = x1a*)

**case** *True*

**then show** *?thesis* **by** *blast*

**next**

**case** *False*

**then show** *?thesis*

**using** *ICons.premss ICons.hyps*

**proof** *auto*

**show**  $\bigwedge \text{ys zs.}$

$x \neq x1a \implies$

$x \notin \text{iset } \text{ys} \implies$

$xs = \text{ys} \ominus x \odot zs \implies$

$\exists \text{ysa. } (\exists \text{zsa. } x1a \odot \text{ys} \ominus x \odot zs = \text{ysa} \ominus x \odot \text{zsa}) \wedge x \notin \text{iset } \text{ysa} \vee$   
 $x1a \odot \text{ys} \ominus x \odot zs = \text{ysa} \ominus \langle x \rangle \wedge x \notin \text{iset } \text{ysa}$

**by** (*meson ICons-eq-iappI iset-IConsD*)

**show**  $x \neq x1a \implies$

$xs = \langle x \rangle \implies$

$\exists \text{ys. } (\exists \text{zs. } \langle x1a, x \rangle = \text{ys} \ominus x \odot zs) \wedge x \notin \text{iset } \text{ys} \vee \langle x1a, x \rangle = \text{ys} \ominus \langle x \rangle \wedge x \notin \text{iset } \text{ys}$

**by** (*metis False Set.set-insert iapp-INil interval.simps(15) singleton-insert-inj-eq'*)

**show**  $\bigwedge \text{zs. } x \neq x1a \implies$

$xs = x \odot zs \implies$

$\exists \text{ys. } (\exists \text{zsa. } x1a \odot x \odot zs = \text{ys} \ominus x \odot \text{zsa}) \wedge x \notin \text{iset } \text{ys} \vee$

$x1a \odot x \odot zs = \text{ys} \ominus \langle x \rangle \wedge x \notin \text{iset } \text{ys}$

**by** (*metis iapp-INil interval.simps(15) singleton-iff*)

**show**  $\bigwedge \text{ys. } x \neq x1a \implies$

$x \notin \text{iset } \text{ys} \implies$

$xs = \text{ys} \ominus \langle x \rangle \implies$

$\exists \text{ysa. } (\exists \text{zs. } x1a \odot \text{ys} \ominus \langle x \rangle = \text{ysa} \ominus x \odot zs) \wedge x \notin \text{iset } \text{ysa} \vee$

$x1a \odot \text{ys} \ominus \langle x \rangle = \text{ysa} \ominus \langle x \rangle \wedge x \notin \text{iset } \text{ysa}$

**by** (*meson ICons-eq-iappI iset-IConsD*)

**qed**

**qed**

**qed**

**lemma** *in-iset-conv-decomp-first*:

$x \in \text{iset } xs =$

$(\exists \text{ys zs. } xs = \text{ys} \ominus (x \odot zs) \wedge x \notin \text{iset } \text{ys} \vee xs = \langle x \rangle \vee xs = x \odot zs \vee$   
 $xs = \text{ys} \ominus \langle x \rangle \wedge x \notin \text{iset } \text{ys})$



by (auto dest!: split-interval-first)

**lemma** *split-interval-last*:

**assumes**  $x \in \text{iset } xs$

**shows**  $\exists ys\ zs.\ xs = ys \ominus (x \odot zs) \wedge x \notin \text{iset } zs \vee xs = \langle x \rangle \vee xs = x \odot zs \wedge x \notin \text{iset } zs \vee$   
 $xs = ys \ominus \langle x \rangle$

**using** *assms*

**proof** (*induct xs rule: irev-induct*)

**case** (*INil y*)

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

**next**

**case** (*snoc x1a xs*)

**then show** ?case **proof** (*cases x = x1a*)

**case** *True*

**then show** ?thesis **by** *blast*

**next**

**case** *False*

**then show** ?thesis **using** *snoc* **by** *fastforce*

**qed**

**qed**

**lemma** *interval-in-iset-conv-decomp-last*:

$x \in \text{iset } xs =$

$(\exists ys\ zs.\ xs = ys \ominus (x \odot zs) \wedge x \notin \text{iset } zs \vee xs = \langle x \rangle \vee xs = x \odot zs \wedge x \notin \text{iset } zs \vee$   
 $xs = ys \ominus \langle x \rangle)$

**by** (auto dest!: split-interval-last)

**lemma** *interval-list-prop*:

**assumes**  $\exists x \in \text{iset } xs.\ P\ x$

**shows**  $(\exists ys\ x\ zs.\ (xs = ys \ominus (x \odot zs) \vee xs = \langle x \rangle \vee xs = x \odot zs \vee$   
 $xs = ys \ominus \langle x \rangle) \wedge P\ x)$

**using** *assms*

**proof** (*induct xs*)

**case** (*INil x*)

**then show** ?case **by** *auto*

**next**

**case** (*ICons x1a xs*)

**then show** ?case

**by** (*meson split-interval*)

**qed**

**lemma** *split-interval-propE*:

**assumes**  $\exists x \in \text{iset } xs.\ P\ x$

**obtains**  $ys\ x\ zs$  **where**  $xs = ys \ominus (x \odot zs) \vee xs = \langle x \rangle \vee xs = x \odot zs \vee$   
 $xs = ys \ominus \langle x \rangle$  **and**  $P\ x$

**using** *interval-list-prop* [*OF assms*] **by** *blast*

**lemma** *split-interval-first-prop*:

**assumes**  $\exists x \in \text{iset } xs.\ P\ x$

**shows**  $(\exists ys\ x\ zs.\ ((xs = ys \ominus (x \odot zs) \wedge (\forall y \in \text{iset } ys.\ \neg P\ y)) \vee$

```


$$xs = \langle x \rangle \vee xs = x \odot zs \vee$$


$$(xs = ys \ominus \langle x \rangle \wedge (\forall y \in \text{iset } ys. \neg P y))) \wedge P x$$

)
using assms
proof (induct xs)
case (INil x)
then show ?case by auto
next
  case (ICons x1a xs)
  then show ?case proof (cases P x1a)
  case True
  then show ?thesis by blast
  next
  case False
  then show ?thesis
    using ICons
    proof auto
    show  $\bigwedge ys \ x a \ zs.$ 
       $\neg P \ x1a \implies$ 
       $P \ x a \implies$ 
       $\forall y \in \text{iset } ys. \neg P \ y \implies$ 
       $xs = ys \ominus xa \odot zs \implies$ 
       $\exists ysa \ x.$ 
       $(\exists zsa. \ x1a \odot ys \ominus xa \odot zs = ysa \ominus x \odot zsa \wedge (\forall y \in \text{iset } ysa. \neg P \ y) \vee$ 
       $\ x1a = x \wedge ys \ominus xa \odot zs = zsa \vee x1a \odot ys \ominus xa \odot zs = ysa \ominus \langle x \rangle \wedge (\forall y \in \text{iset } ysa. \neg P \ y)) \wedge$ 
       $P \ x$ 
      by (metis insertE iapp-ICons interval.simps(16))
    show  $\bigwedge x \ ys \ xa \ zs.$ 
       $\neg P \ x1a \implies$ 
       $P \ x \implies$ 
       $P \ xa \implies$ 
       $\forall y \in \text{iset } ys. \neg P \ y \implies$ 
       $xs = ys \ominus xa \odot zs \implies$ 
       $x \in \text{iset } zs \implies$ 
       $\exists ysa \ x.$ 
       $(\exists zsa. \ x1a \odot ys \ominus xa \odot zs = ysa \ominus x \odot zsa \wedge (\forall y \in \text{iset } ysa. \neg P \ y) \vee$ 
       $\ x1a = x \wedge ys \ominus xa \odot zs = zsa \vee x1a \odot ys \ominus xa \odot zs = ysa \ominus \langle x \rangle \wedge (\forall y \in \text{iset } ysa. \neg P \ y)) \wedge$ 
       $P \ x$ 
      by (metis ICons-eq-iappI iset-IConsD)
    show  $\bigwedge xa. \neg P \ x1a \implies$ 
       $P \ xa \implies$ 
       $xs = \langle xa \rangle \implies$ 
       $\exists ys \ x.$ 
       $(\exists zs. \ \langle x1a, xa \rangle = ys \ominus x \odot zs \wedge (\forall y \in \text{iset } ys. \neg P \ y) \vee$ 
       $\ x1a = x \wedge \langle xa \rangle = zs \vee \langle x1a, xa \rangle = ys \ominus \langle x \rangle \wedge (\forall y \in \text{iset } ys. \neg P \ y)) \wedge P \ x$ 
      by (metis iapp-INil interval.simps(15) singleton-iff)
    show  $\bigwedge xa \ zs.$ 
       $\neg P \ x1a \implies$ 
       $P \ xa \implies$ 
       $xs = xa \odot zs \implies$ 

```

```

 $\exists ys\ x.$ 
  ( $\exists zsa.\ x1a \odot xa \odot zs = ys \ominus x \odot zsa \wedge (\forall y \in iset\ ys.\ \neg P\ y) \vee$ 
     $x1a = x \wedge xa \odot zs = zsa \vee x1a \odot xa \odot zs = ys \ominus \langle x \rangle \wedge (\forall y \in iset\ ys.\ \neg P\ y)) \wedge$ 
     $P\ x$ 
  by (metis iapp-INil interval.simps(15) singleton-iff)
show  $\bigwedge x\ xa\ zs.$ 
   $\neg P\ x1a \implies$ 
   $P\ x \implies$ 
   $P\ xa \implies$ 
   $xs = xa \odot zs \implies$ 
   $x \in iset\ zs \implies$ 
 $\exists ys\ x.$ 
  ( $\exists zsa.\ x1a \odot xa \odot zs = ys \ominus x \odot zsa \wedge (\forall y \in iset\ ys.\ \neg P\ y) \vee$ 
     $x1a = x \wedge xa \odot zs = zsa \vee x1a \odot xa \odot zs = ys \ominus \langle x \rangle \wedge (\forall y \in iset\ ys.\ \neg P\ y)) \wedge$ 
     $P\ x$ 
  by (metis iapp-INil interval.simps(15) singleton-iff)
show  $\bigwedge ys\ xa.$ 
   $\neg P\ x1a \implies$ 
   $P\ xa \implies$ 
 $\forall y \in iset\ ys.\ \neg P\ y \implies$ 
   $xs = ys \ominus \langle xa \rangle \implies$ 
 $\exists ysa\ x.$ 
  ( $\exists zs.\ x1a \odot ys \ominus \langle xa \rangle = ysa \ominus x \odot zs \wedge (\forall y \in iset\ ysa.\ \neg P\ y) \vee$ 
     $x1a = x \wedge ys \ominus \langle xa \rangle = zs \vee x1a \odot ys \ominus \langle xa \rangle = ysa \ominus \langle x \rangle \wedge (\forall y \in iset\ ysa.\ \neg P\ y)) \wedge$ 
     $P\ x$ 
  by (metis insertE iapp.simps(2) interval.set(2))
qed
qed
qed

```

**lemma** *split-interval-first-propE*:

**assumes**  $\exists x \in iset\ xs.\ P\ x$

**obtains**  $ys\ x\ zs$  **where**  $((xs = ys \ominus (x \odot zs) \wedge (\forall y \in iset\ ys.\ \neg P\ y)) \vee$

$xs = \langle x \rangle \vee xs = x \odot zs \vee$

$(xs = ys \ominus \langle x \rangle \wedge (\forall y \in iset\ ys.\ \neg P\ y)))$  **and**  $P\ x$

**using** *split-interval-first-prop* [*OF assms*] **by** *blast*

**lemma** *split-first-prop-iff*:

$(\exists x \in iset\ xs.\ P\ x) \longleftrightarrow$

$(\exists ys\ x\ zs.\ ((xs = ys \ominus (x \odot zs) \wedge (\forall y \in iset\ ys.\ \neg P\ y)) \vee$

$xs = \langle x \rangle \vee xs = x \odot zs \vee$

$(xs = ys \ominus \langle x \rangle \wedge (\forall y \in iset\ ys.\ \neg P\ y))) \wedge P\ x$

)

**by** (*rule, erule split-interval-first-prop*) *auto*

**lemma** *split-interval-last-prop*:

**assumes**  $\exists x \in iset\ xs.\ P\ x$

**shows**  $(\exists ys\ x\ zs.\ ((xs = ys \ominus (x \odot zs) \wedge (\forall y \in iset\ zs.\ \neg P\ y)) \vee$

$xs = \langle x \rangle \vee xs = x \odot zs \wedge (\forall y \in iset\ zs.\ \neg P\ y) \vee$

$(xs = ys \ominus \langle x \rangle)) \wedge P\ x$

```

)
using assms
proof (induct xs rule: irev-induct)
case (INil y)
then show ?case by auto
next
case (snoc x1a xs)
  then show ?case proof (cases P x1a)
    case True
    then show ?thesis by blast
  next
  case False
  then show ?thesis
    using snoc by fastforce
  qed
qed

lemma split-interval-last-propE:
  assumes  $\exists x \in \text{iset } xs. P\ x$ 
  obtains ys x zs where  $((xs = ys \ominus (x \odot zs) \wedge (\forall y \in \text{iset } zs. \neg P\ y)) \vee$ 
 $xs = \langle x \rangle \vee xs = x \odot zs \wedge (\forall y \in \text{iset } zs. \neg P\ y) \vee$ 
 $(xs = ys \ominus \langle x \rangle)) \text{ and } P\ x$ 
  using split-interval-last-prop [OF assms] by blast

lemma split-interval-last-prop-iff:
  ( $\exists x \in \text{iset } xs. P\ x$ )  $\longleftrightarrow$ 
  ( $\exists\ ys\ x\ zs. ((xs = ys \ominus (x \odot zs) \wedge (\forall y \in \text{iset } zs. \neg P\ y)) \vee$ 
 $xs = \langle x \rangle \vee xs = x \odot zs \wedge (\forall y \in \text{iset } zs. \neg P\ y) \vee$ 
 $(xs = ys \ominus \langle x \rangle)) \wedge P\ x$ 
  )
by (rule, erule split-interval-last-prop, auto)

lemma inth-and-iset:
   $x \in \text{iset } xs = (\exists\ i \leq \text{ilen } xs. (\text{inth } xs\ i) = x)$ 
proof (induct xs)
case (INil x)
then show ?case by auto
next
case (ICons x1a xs)
then show ?case
by (metis Suc-le-mono insert-iff interval.simps(16) ilen-ICons ilen-gr-zero
  inth-Suc inth-ICons inth-zero ilen.simps(2) le-diff-conv neq0-conv plus-1-eq-Suc)
qed

lemma card-ilen:
   $\text{card } (\text{iset } xs) \leq \text{ilen } xs + 1$ 
proof (induct xs)
case (INil x)
then show ?case by simp
next

```

```

case (ICons x1a xs)
then show ?case by (simp add: card-insert-le-m1)
qed

```

**lemma** *iset-inth*:

```

iset xs = { (inth xs k) | k. k ≤ ilen xs }
proof (induct xs)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
proof auto
show iset xs = { inth xs k | k. k ≤ ilen xs } ⇒
  ∃ k. x1a = (case k of 0 ⇒ x1a | Suc x ⇒ inth xs x) ∧ k ≤ Suc (ilen xs)
by force
show ∧k. iset xs = { inth xs k | k. k ≤ ilen xs } ⇒
  k ≤ ilen xs ⇒
  ∃ ka. inth xs k = (case ka of 0 ⇒ x1a | Suc x ⇒ inth xs x) ∧ ka ≤ Suc (ilen xs)
by force
show ∧k. iset xs = { inth xs k | k. k ≤ ilen xs } ⇒
  (case k of 0 ⇒ x1a | Suc x ⇒ inth xs x) ≠ x1a ⇒
  k ≤ Suc (ilen xs) ⇒
  ∃ ka. (case k of 0 ⇒ x1a | Suc x ⇒ inth xs x) = inth xs ka ∧ ka ≤ ilen xs
by (metis (mono-tags, lifting) Nitpick.case-nat-unfold Suc-eq-plus1 le-diff-conv)
qed
qed

```

**lemma** *prefix-iset*:

```

assumes k ≤ ilen xs
shows iset (prefix k xs) = { (inth xs i) | i. i ≤ k }
proof -
have 1: iset (prefix k xs) = { (inth (prefix k xs) i) | i. i ≤ ilen (prefix k xs) }
by (simp add: iset-inth)
have 2: k ≤ ilen xs
using assms by auto
have 3: { (inth (prefix k xs) i) | i. i ≤ ilen (prefix k xs) } =
  { (inth xs i) | i. i ≤ k }
using 2 by (metis inth-prefix prefix-ilen-good)
show ?thesis using 1 3 by auto
qed

```

**lemma** *suffix-iset*:

```

assumes k ≤ ilen xs
shows iset (suffix k xs) = { (inth xs (k+i)) | i. i ≤ ilen xs - k }
proof -
have 1: iset (suffix k xs) = { (inth (suffix k xs) i) | i. i ≤ ilen (suffix k xs) }
by (simp add: iset-inth)
have 2: k ≤ ilen xs
using assms by auto

```

**have**  $\beta$ :  $\{ (inth\ (suffix\ k\ xs)\ i) \mid i.\ i \leq ilen\ (suffix\ k\ xs) \} =$   
 $\{ (inth\ xs\ (k+i)) \mid i.\ i \leq ilen\ xs - k \}$   
**by** *force*  
**show** *?thesis* **using** 1  $\beta$  **by** *auto*  
**qed**

**lemma** *suffix-iset-a*:  
**assumes**  $k \leq ilen\ xs$   
**shows**  $iset\ (suffix\ k\ xs) = \{ (inth\ xs\ i) \mid i.\ k \leq i \wedge i \leq ilen\ xs \}$   
**proof** –  
**have** 1:  $iset\ (suffix\ k\ xs) = \{ (inth\ xs\ (k+i)) \mid i.\ i \leq ilen\ xs - k \}$   
**using** *assms suffix-iset* **by** *blast*  
**have** 2:  $\forall x \in \{ (inth\ xs\ (k+i)) \mid i.\ i \leq ilen\ xs - k \}.$   
 $x \in \{ (inth\ xs\ i) \mid i.\ k \leq i \wedge i \leq ilen\ xs \}$   
**using** *assms* **by** *auto*  
**have** 3:  $\forall x \in \{ (inth\ xs\ i) \mid i.\ k \leq i \wedge i \leq ilen\ xs \}.$   
 $x \in \{ (inth\ xs\ (k+i)) \mid i.\ i \leq ilen\ xs - k \}$   
**using** *nat-le-iff-add* **by** *force*  
**have** 4:  $\{ (inth\ xs\ (k+i)) \mid i.\ i \leq ilen\ xs - k \} =$   
 $\{ (inth\ xs\ i) \mid i.\ k \leq i \wedge i \leq ilen\ xs \}$   
**using** 2 3 **by** *blast*  
**show** *?thesis* **by** (*simp add: 1 4*)  
**qed**

**lemma** *sub-interval-iset*:  
**assumes**  $k \leq n$   
 $n \leq ilen\ xs$   
**shows**  $iset\ (sub\ k\ n\ xs) = \{ (inth\ xs\ (k+i)) \mid i.\ i \leq n-k \}$   
**proof** –  
**have** 1:  $iset\ (sub\ k\ n\ xs) = \{ (inth\ (sub\ k\ n\ xs)\ i) \mid i.\ i \leq ilen\ (sub\ k\ n\ xs) \}$   
**by** (*simp add: iset-inth*)  
**have** 2:  $k \leq n$   
**using** *assms* **by** *auto*  
**have** 3:  $n \leq ilen\ xs$   
**using** *assms* **by** *auto*  
**have** 4:  $\{ (inth\ (sub\ k\ n\ xs)\ i) \mid i.\ i \leq ilen\ (sub\ k\ n\ xs) \} =$   
 $\{ (inth\ xs\ (k+i)) \mid i.\ i \leq n-k \}$   
**using** 2 3 **by** *force*  
**show** *?thesis* **by** (*simp add: 1 4*)  
**qed**

**lemma** *sub-iset-a*:  
**assumes**  $k \leq n$   
 $n \leq ilen\ xs$   
**shows**  $iset\ (sub\ k\ n\ xs) = \{ (inth\ xs\ i) \mid i.\ k \leq i \wedge i \leq n \}$   
**proof** –  
**have** 1:  $iset\ (sub\ k\ n\ xs) = \{ (inth\ xs\ (k+i)) \mid i.\ i \leq n-k \}$   
**using** *assms sub-interval-iset* **by** *blast*  
**have** 2:  $\forall x \in \{ (inth\ xs\ (k+i)) \mid i.\ i \leq n-k \}.$

```

      x ∈ { (inth xs i) | i. k ≤ i ∧ i ≤ n }
    using assms by auto
  have 3: ∀ x ∈ { (inth xs i) | i. k ≤ i ∧ i ≤ n }.
      x ∈ { (inth xs (k+i)) | i. i ≤ n-k }
    using assms using le-Suc-ex by auto fastforce
  have 4: { (inth xs (k+i)) | i. i ≤ n-k } = { (inth xs i) | i. k ≤ i ∧ i ≤ n }
    using 2 3 by blast
  show ?thesis by (simp add: 1 4)
qed

```

```

lemma inth-iset:
  assumes k ≤ ilen xs
  shows (inth xs k) ∈ iset xs
  using assms
  by (meson inth-and-iset)

```

end

## 2 Finite ITL Semantics

```

theory Semantics
  imports Interval HOL-TLA.Intensional HOL-TLA.Stfun
begin

```

This theory mechanises a *shallow* embedding of finite ITL using the *Interval* and *Intensional* theories. A shallow embedding represents ITL using Isabelle/HOL predicates, while a *deep* embedding [1] would represent ITL formulas as mutually inductive datatypes. See, e.g., [12] for a discussion about deep vs. shallow embeddings in Isabelle/HOL. The choice of a shallow over a deep embedding is motivated [3, 2] by the following factors: a shallow embedding is usually less involved, and existing Isabelle theories and tools can be applied more directly to enhance automation; due to the lifting in the *Intensional* theory, a shallow embedding can reuse standard logical operators, whilst a deep embedding requires a different set of operators for formulas. Finally, since our target is system verification rather than proving meta-properties of the logic, which requires a deep embedding, a shallow embedding is more fit for purpose.

### 2.1 Types of Formulas

To mechanise the ITL semantics, the following type abbreviations are used:

```

type-synonym ('a,'b) formfun = 'a interval ⇒ 'b
type-synonym 'a formula      = ('a,bool) formfun
type-synonym ('a,'b) stfun   = 'a ⇒ 'b
type-synonym 'a stpred       = ('a,bool) stfun

```

```

instance
  fun :: (type,type) world ..

```

**instance**

*prod* :: (type,type) world ..

**instance**

*interval* :: (type) world ..

Pair, function, and interval are instantiated to be of type class world. This allows use of the lifted Intensional logic for formulas, and standard logical connectives can therefore be used.

## 2.2 Semantics of ITL

The semantics of ITL is defined. Note chopstar is a derived operator, i.e., it is defined recursively in terms of chop.

**definition** *skip-d* :: ('a :: world) formula

**where** *skip-d*  $\equiv \lambda s. \text{ilen } s = 1$

**definition** *chop-d* :: ('a :: world) formula  $\Rightarrow$  ('a :: world) formula  $\Rightarrow$  ('a :: world) formula

**where** *chop-d* *F1 F2*  $\equiv \lambda s. \exists n. 0 \leq n \wedge n \leq \text{ilen } s \wedge ((\text{prefix } n \ s) \models F1) \wedge ((\text{suffix } n \ s) \models F2)$

**definition** *current-val-d* :: ('a :: world, 'b) stfun  $\Rightarrow$  ('a, 'b) formfun

**where** *current-val-d* *f*  $\equiv \lambda s. (\text{inth } s \ 0) \models f$

**definition** *next-val-d* :: ('a :: world, 'b) stfun  $\Rightarrow$  ('a, 'b) formfun

**where** *next-val-d* *f*  $\equiv \lambda s. \text{if } \text{ilen } s > 0 \text{ then } ((\text{inth } s \ 1) \models f) \text{ else } (\epsilon \ (x :: 'b) . x = x)$

**definition** *fin-val-d* :: ('a :: world, 'b) stfun  $\Rightarrow$  ('a, 'b) formfun

**where** *fin-val-d* *f*  $\equiv \lambda s. (\text{inth } s \ (\text{ilen } s)) \models f$

**definition** *penult-val-d* :: ('a :: world, 'b) stfun  $\Rightarrow$  ('a, 'b) formfun

**where** *penult-val-d* *f*  $\equiv \lambda s. \text{if } \text{ilen } s > 0 \text{ then } (\text{inth } s \ ((\text{ilen } s) - 1)) \models f \text{ else } (\epsilon \ (x :: 'b) . x = x)$

This is the concrete syntax for the (abstract) operators above.

**syntax**

*-skip-d* :: lift ((skip))  
*-chop-d* :: [lift, lift]  $\Rightarrow$  lift ((;-) [84,84] 83)  
*-current-val-d* :: lift  $\Rightarrow$  lift ((\$-) [100] 99)  
*-next-val-d* :: lift  $\Rightarrow$  lift ((-\$) [100] 99)  
*-fin-val-d* :: lift  $\Rightarrow$  lift ((!-) [100] 99)  
*-penult-val-d* :: lift  $\Rightarrow$  lift ((-!) [100] 99)  
*TEMP* :: lift  $\Rightarrow$  'b ((TEMP -))

**syntax (ASCII)**

*-skip-d* :: lift ((skip))  
*-chop-d* :: [lift, lift]  $\Rightarrow$  lift ((;-) [84,84] 83)  
*-current-val-d* :: lift  $\Rightarrow$  lift ((\$-) [100] 99)  
*-next-val-d* :: lift  $\Rightarrow$  lift ((-\$) [100] 99)  
*-fin-val-d* :: lift  $\Rightarrow$  lift ((!-) [100] 99)  
*-penult-val-d* :: lift  $\Rightarrow$  lift ((-!) [100] 99)

**translations**



$-skip-d \quad \Rightarrow \text{CONST } skip-d$   
 $-chop-d \quad \Rightarrow \text{CONST } chop-d$   
 $-current-val-d \Rightarrow \text{CONST } current-val-d$   
 $-next-val-d \quad \Rightarrow \text{CONST } next-val-d$   
 $-fin-val-d \quad \Rightarrow \text{CONST } fin-val-d$   
 $-penult-val-d \Rightarrow \text{CONST } penult-val-d$   
 $TEMP F \quad \rightarrow (F :: (- \text{ interval}) \Rightarrow -)$

## 2.3 Abbreviations

Some standard temporal abbreviations, with their concrete syntax.

**definition**  $sometimes-d :: ('a::world) \text{ formula} \Rightarrow 'a \text{ formula}$   
**where**  $sometimes-d F \equiv LIFT(\#True; F)$

**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**  $da-d F \equiv LIFT(\#True; (F; \#True))$

**definition**  $next-d :: ('a::world) \text{ formula} \Rightarrow 'a \text{ formula}$   
**where**  $next-d F \equiv LIFT(skip; F)$

**definition**  $prev-d :: ('a::world) \text{ formula} \Rightarrow 'a \text{ formula}$   
**where**  $prev-d F \equiv LIFT(F; skip)$

### syntax

$-sometimes-d :: lift \Rightarrow lift ((\Diamond-) [88] 87)$   
 $-di-d \quad \quad \quad :: lift \Rightarrow lift ((di -) [88] 87)$   
 $-da-d \quad \quad \quad :: lift \Rightarrow lift ((da -) [88] 87)$   
 $-next-d \quad \quad \quad :: lift \Rightarrow lift ((\bigcirc -) [88] 87)$   
 $-prev-d \quad \quad \quad :: lift \Rightarrow lift ((prev -) [88] 87)$

### syntax (ASCII)

$-sometimes-d :: lift \Rightarrow lift ((<>-) [88] 87)$   
 $-di-d \quad \quad \quad :: lift \Rightarrow lift ((di -) [88] 87)$   
 $-da-d \quad \quad \quad :: lift \Rightarrow lift ((da -) [88] 87)$   
 $-next-d \quad \quad \quad :: lift \Rightarrow lift ((next -) [88] 87)$   
 $-prev-d \quad \quad \quad :: lift \Rightarrow lift ((prev -) [88] 87)$

### translations

$-sometimes-d \Rightarrow \text{CONST } sometimes-d$   
 $-di-d \quad \quad \quad \Rightarrow \text{CONST } di-d$   
 $-da-d \quad \quad \quad \Rightarrow \text{CONST } da-d$   
 $-next-d \quad \quad \quad \Rightarrow \text{CONST } next-d$   
 $-prev-d \quad \quad \quad \Rightarrow \text{CONST } prev-d$

**definition** *always-d* :: ('a::world) formula  $\Rightarrow$  'a formula  
**where** *always-d*  $F \equiv LIFT(\neg(\Diamond(\neg F)))$

**definition** *bi-d* :: ('a::world) formula  $\Rightarrow$  'a formula  
**where** *bi-d*  $F \equiv LIFT(\neg(di(\neg F)))$

**definition** *ba-d* :: ('a::world) formula  $\Rightarrow$  'a formula  
**where** *ba-d*  $F \equiv LIFT(\neg(da(\neg F)))$

**definition** *wnext-d* :: ('a::world) formula  $\Rightarrow$  'a formula  
**where** *wnext-d*  $F \equiv LIFT(\neg(\bigcirc(\neg F)))$

**definition** *wprev-d* :: ('a::world) formula  $\Rightarrow$  'a formula  
**where** *wprev-d*  $F \equiv LIFT(\neg(prev(\neg F)))$

**definition** *more-d* :: ('a::world) formula  
**where** *more-d*  $\equiv LIFT(\bigcirc(\#True))$

#### **syntax**

-*always-d* :: lift  $\Rightarrow$  lift (( $\Box$ -) [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)**

-*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**

-*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** *empty-d* :: ('a::world) formula  
**where** *empty-d*  $\equiv LIFT(\neg(\text{more}))$

**definition** *dm-d* :: ('a::world) formula  $\Rightarrow$  'a formula  
**where** *dm-d*  $F \equiv LIFT(\#True;(\text{more} \wedge F))$

**syntax**

-empty-d :: lift ((empty))  
 -dm-d :: lift  $\Rightarrow$  lift ((dm -) [88] 87)

**syntax (ASCII)**

-empty-d :: lift ((empty))  
 -dm-d :: lift  $\Rightarrow$  lift ((dm -) [88] 87)

**translations**

-empty-d  $\Rightarrow$  CONST empty-d  
 -dm-d  $\Rightarrow$  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** ifthenelse-d :: ('a::world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula

**where** ifthenelse-d F G H  $\equiv$  LIFT((F  $\wedge$  G)  $\vee$  ( $\neg$ F  $\wedge$  H) )

**primrec** power-d :: ('a::world) formula  $\Rightarrow$  nat  $\Rightarrow$  'a formula

**where** pow-0 : (power-d F 0) = LIFT(empty)

| pow-Suc: (power-d F (Suc n)) = LIFT((F);(power-d F n))

**syntax**

-bm-d :: lift  $\Rightarrow$  lift ((bm -) [88] 87)  
 -init-d :: lift  $\Rightarrow$  lift ((init -) [88] 87)  
 -fin-d :: lift  $\Rightarrow$  lift ((fin -) [88] 87)  
 -halt-d :: lift  $\Rightarrow$  lift ((halt -) [88] 87)  
 -initonly-d :: lift  $\Rightarrow$  lift ((initonly -) [88] 87)  
 -keep-d :: lift  $\Rightarrow$  lift ((keep -) [88] 87)

$\text{-yields-d} \quad :: [\text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((- \text{ yields } -) [88, 88] \ 87)$   
 $\text{-ifthenelse-d} \quad :: [\text{lift}, \text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((\text{if}_i - \text{ then } - \text{ else } -) [88, 88, 88] \ 87)$   
 $\text{-power-d} \quad :: [\text{lift}, \text{nat}] \Rightarrow \text{lift} \quad ((\text{power } -) [88, 88] \ 87)$

#### **syntax** (*ASCII*)

$\text{-bm-d} \quad :: \text{lift} \Rightarrow \text{lift} \quad ((\text{bm } -) [88] \ 87)$   
 $\text{-init-d} \quad :: \text{lift} \Rightarrow \text{lift} \quad ((\text{init } -) [88] \ 87)$   
 $\text{-fin-d} \quad :: \text{lift} \Rightarrow \text{lift} \quad ((\text{fin } -) [88] \ 87)$   
 $\text{-halt-d} \quad :: \text{lift} \Rightarrow \text{lift} \quad ((\text{halt } -) [88] \ 87)$   
 $\text{-initonly-d} \quad :: \text{lift} \Rightarrow \text{lift} \quad ((\text{initonly } -) [88] \ 87)$   
 $\text{-keep-d} \quad :: \text{lift} \Rightarrow \text{lift} \quad ((\text{keep } -) [88] \ 87)$   
 $\text{-yields-d} \quad :: [\text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((- \text{ yields } -) [88, 88] \ 87)$   
 $\text{-ifthenelse-d} \quad :: [\text{lift}, \text{lift}, \text{lift}] \Rightarrow \text{lift} \quad ((\text{if}_i - \text{ then } - \text{ else } -) [88, 88, 88] \ 87)$   
 $\text{-power-d} \quad :: [\text{lift}, \text{nat}] \Rightarrow \text{lift} \quad ((\text{power } -) [88, 88] \ 87)$

#### **translations**

$\text{-bm-d} \quad \Rightarrow \text{CONST bm-d}$   
 $\text{-init-d} \quad \Rightarrow \text{CONST init-d}$   
 $\text{-fin-d} \quad \Rightarrow \text{CONST fin-d}$   
 $\text{-halt-d} \quad \Rightarrow \text{CONST halt-d}$   
 $\text{-initonly-d} \quad \Rightarrow \text{CONST initonly-d}$   
 $\text{-keep-d} \quad \Rightarrow \text{CONST keep-d}$   
 $\text{-yields-d} \quad \Rightarrow \text{CONST yields-d}$   
 $\text{-ifthenelse-d} \quad \Rightarrow \text{CONST ifthenelse-d}$   
 $\text{-power-d} \quad \Rightarrow \text{CONST power-d}$

**definition**  $\text{len-d} :: \text{nat} \Rightarrow ('a::\text{world}) \text{ formula}$

**where**  $\text{len-d } n \equiv \text{LIFT}(\text{power skip } n)$

**definition**  $\text{powerstar-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$

**where**  $\text{powerstar-d } F \equiv \text{LIFT}(\exists k. \text{power } F k)$

#### **syntax**

$\text{-len-d} \quad :: \text{nat} \Rightarrow \text{lift} \quad ((\text{len } -) [88] \ 87)$   
 $\text{-powerstar-d} \quad :: \text{lift} \Rightarrow \text{lift} \quad ((\text{powerstar } -) [85] \ 85)$

#### **syntax** (*ASCII*)

$\text{-len-d} \quad :: \text{nat} \Rightarrow \text{lift} \quad ((\text{len } -) [88] \ 87)$   
 $\text{-powerstar-d} \quad :: \text{lift} \Rightarrow \text{lift} \quad ((\text{powerstar } -) [85] \ 85)$

#### **translations**

$\text{-len-d} \quad \Rightarrow \text{CONST len-d}$   
 $\text{-powerstar-d} \quad \Rightarrow \text{CONST powerstar-d}$

**definition**  $\text{chopstar-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$

**where**  $\text{chopstar-d } F \equiv \text{LIFT}(\text{powerstar } (F \wedge \text{more}))$

#### **syntax**

$\text{-chopstar-d} \quad :: \text{lift} \Rightarrow \text{lift} \quad ((-^*) [85] \ 85)$

**syntax** (*ASCII*)

*-chopstar-d* :: *lift*  $\Rightarrow$  *lift* ((*chopstar -*) [85] 85)

**translations**

*-chopstar-d*  $\Rightarrow$  *CONST chopstar-d*

**definition** *ifthen-d* :: (*'a::world*) *formula*  $\Rightarrow$  *'a formula*  $\Rightarrow$  *'a formula*

**where** *ifthen-d* *F G*  $\equiv$  *LIFT*(*if<sub>i</sub>* *F* then *G* else #*True* )

**definition** *while-d* :: (*'a::world*) *formula*  $\Rightarrow$  *'a formula*  $\Rightarrow$  *'a formula*

**where** *while-d* *F G*  $\equiv$  *LIFT*( ( *F*  $\wedge$  *G* )<sup>\*</sup>  $\wedge$  (*fin* ( $\neg$ *F*))) )

**syntax**

*-ifthen-d* :: [*lift, lift*]  $\Rightarrow$  *lift* ((*if<sub>i</sub>* - then - ) [88,88] 87)

*-while-d* :: [*lift, lift*]  $\Rightarrow$  *lift* ((*while* - do - ) [88,88] 87)

**syntax** (*ASCII*)

*-ifthen-d* :: [*lift, lift*]  $\Rightarrow$  *lift* ((*if<sub>i</sub>* - then - ) [88,88] 87)

*-while-d* :: [*lift, lift*]  $\Rightarrow$  *lift* ((*while* - do - ) [88,88] 87)

**translations**

*-ifthen-d*  $\Rightarrow$  *CONST ifthen-d*

*-while-d*  $\Rightarrow$  *CONST while-d*

**definition** *repeat-d* :: (*'a::world*) *formula*  $\Rightarrow$  *'a formula*  $\Rightarrow$  *'a formula*

**where** *repeat-d* *F G*  $\equiv$  *LIFT*(*F*; *while* ( $\neg$  *G*) do *F* )

**syntax**

*-repeat-d* :: [*lift, lift*]  $\Rightarrow$  *lift* ((*repeat* - until - ) [88,88] 87)

**syntax** (*ASCII*)

*-repeat-d* :: [*lift, lift*]  $\Rightarrow$  *lift* ((*repeat* - until - ) [88,88] 87)

**translations**

*-repeat-d*  $\Rightarrow$  *CONST repeat-d*

**definition** *next-assign-d* :: (*'a::world, 'b*) *stfun*  $\Rightarrow$  (*'a, 'b*) *formfun*  $\Rightarrow$  *'a formula*

**where** *next-assign-d* *v e*  $\equiv$  *LIFT*( *v*\$ = *e* )

**definition** *prev-assign-d* :: (*'a::world, 'b*) *stfun*  $\Rightarrow$  (*'a, 'b*) *formfun*  $\Rightarrow$  *'a formula*

**where** *prev-assign-d* *v e*  $\equiv$  *LIFT*( *v*! = *e* )

**definition** *always-eq-d* :: (*'a::world, 'b*) *stfun*  $\Rightarrow$  (*'a, 'b*) *formfun*  $\Rightarrow$  *'a formula*

**where** *always-eq-d* *v e*  $\equiv$   $\lambda s. s \models \Box(\$v = e)$

**definition** *temporal-assign-d* :: (*'a::world, 'b*) *stfun*  $\Rightarrow$  (*'a, 'b*) *formfun*  $\Rightarrow$  *'a formula*

**where** *temporal-assign-d* *v e*  $\equiv$   $\lambda s. s \models !v = e$

**definition** *gets-d* :: ('a::world,'b) stfun  $\Rightarrow$  ('a,'b) formfun  $\Rightarrow$  'a formula  
**where** *gets-d* v e  $\equiv \lambda s. s \models \text{keep}(\text{temporal-assign-d } v \text{ } e)$

**definition** *stable-d* :: ('a::world,'b) stfun  $\Rightarrow$  'a formula  
**where** *stable-d* v  $\equiv \lambda s. s \models \text{gets-d } v \text{ } (\text{current-val-d } v)$

**definition** *padded-d* :: ('a::world,'b) stfun  $\Rightarrow$  'a formula  
**where** *padded-d* v  $\equiv \lambda s. s \models (\text{stable-d } v); \text{skip} \vee \text{empty}$

**definition** *padded-temp-assign-d* :: ('a::world,'b) stfun  $\Rightarrow$  ('a,'b) formfun  $\Rightarrow$  'a formula  
**where** *padded-temp-assign-d* v e  $\equiv \lambda s. s \models (\text{temporal-assign-d } v \text{ } e) \wedge (\text{padded-d } v)$

### syntax

-next-assign-d :: [lift, lift]  $\Rightarrow$  lift ((- := -) [50,51] 50)  
-prev-assign-d :: [lift, lift]  $\Rightarrow$  lift ((- =: -) [50,51] 50)  
-always-eq-d :: [lift, lift]  $\Rightarrow$  lift ((-  $\approx$  -) [50,51] 50)  
-temporal-assign-d :: [lift, lift]  $\Rightarrow$  lift ((-  $\leftarrow$  -) [50,51] 50)  
-gets-d :: [lift, lift]  $\Rightarrow$  lift ((- gets -) [50,51] 50)  
-stable-d :: lift  $\Rightarrow$  lift ((stable -) [51] 50)  
-padded-d :: lift  $\Rightarrow$  lift ((padded -) [51] 50)  
-padded-temp-assign-d :: [lift, lift]  $\Rightarrow$  lift ((-  $<\sim$  -) [50,51] 50)

### syntax (ASCII)

-next-assign-d :: [lift, lift]  $\Rightarrow$  lift ((- := -) [50,51] 50)  
-prev-assign-d :: [lift, lift]  $\Rightarrow$  lift ((- =: -) [50,51] 50)  
-always-eq-d :: [lift, lift]  $\Rightarrow$  lift ((- alweqv -) [50,51] 50)  
-temporal-assign-d :: [lift, lift]  $\Rightarrow$  lift ((- <-- -) [50,51] 50)  
-gets-d :: [lift, lift]  $\Rightarrow$  lift ((- gets -) [50,51] 50)  
-stable-d :: lift  $\Rightarrow$  lift ((stable -) [51] 50)  
-padded-d :: lift  $\Rightarrow$  lift ((padded -) [51] 50)  
-padded-temp-assign-d :: [lift, lift]  $\Rightarrow$  lift ((-  $<\sim$  -) [50,51] 50)

### translations

-next-assign-d  $\Rightarrow$  CONST next-assign-d  
-prev-assign-d  $\Rightarrow$  CONST prev-assign-d  
-always-eq-d  $\Rightarrow$  CONST always-eq-d  
-temporal-assign-d  $\Rightarrow$  CONST temporal-assign-d  
-gets-d  $\Rightarrow$  CONST gets-d  
-stable-d  $\Rightarrow$  CONST stable-d  
-padded-d  $\Rightarrow$  CONST padded-d  
-padded-temp-assign-d  $\Rightarrow$  CONST padded-temp-assign-d

**lemmas** *itl-def* = skip-d-def chop-d-def current-val-d-def next-val-d-def fin-val-d-def penult-val-d-def  
sometimes-d-def di-d-def da-d-def next-d-def prev-d-def always-d-def bi-d-def ba-d-def wnext-d-def  
wprev-d-def more-d-def empty-d-def dm-d-def bm-d-def init-d-def fin-d-def halt-d-def initempty-d-def  
keep-d-def yields-d-def ifthenelse-d-def power-d-def len-d-def powerstar-d-def chopstar-d-def  
ifthen-d-def while-d-def repeat-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

## 2.4 Properties of Operators

The following lemmas show that above operators have the expected semantics.

**lemma** *skip-defs* :

$$(w \models \text{skip}) = (\text{ilen } w = 1)$$

**by** (*simp add: itl-def*)

**lemma** *chop-defs* :

$$(w \models F1 ; F2) = (\exists n . n \leq \text{ilen } w \wedge ((\text{prefix } n \ w) \models F1) \wedge ((\text{suffix } n \ w) \models F2))$$

**by** (*simp add: itl-def*)

**lemma** *sometimes-defs* :

$$(w \models \Diamond F) = (\exists n . n \leq \text{ilen } w \wedge ((\text{suffix } n \ w) \models F))$$

**by** (*simp add: itl-def chop-defs*)

**lemma** *always-defs* :

$$(w \models \Box F) = (\forall n . n \leq \text{ilen } w \longrightarrow ((\text{suffix } n \ w) \models F))$$

**by** (*simp add: itl-def sometimes-defs*)

**lemma** *di-defs* :

$$(w \models \text{di } F) = (\exists n . n \leq \text{ilen } w \wedge ((\text{prefix } n \ w) \models F))$$

**by** (*simp add: itl-def chop-defs*)

**lemma** *bi-defs* :

$$(w \models \text{bi } F) = (\forall n . n \leq \text{ilen } w \longrightarrow ((\text{prefix } n \ w) \models F))$$

**by** (*simp add: itl-def di-defs*)

**lemma** *da-defs* :

$$(w \models \text{da } F) = (\exists n \text{ na} . n + \text{na} \leq \text{ilen } w \wedge ((\text{sub } n \ (n + \text{na}) \ w) \models F))$$

**proof** (*auto simp add: itl-def chop-defs*)

**show**  $\bigwedge n \text{ na} .$

$$n \leq \text{ilen } w \implies$$

$$\text{na} \leq \text{ilen } w - n \implies F (\text{prefix } \text{na} \ (\text{suffix } n \ w)) \implies$$

$$\exists n \text{ na} . n + \text{na} \leq \text{ilen } w \wedge F (\text{sub } n \ (n + \text{na}) \ w)$$

**by** (*metis sub-def Nat.le-diff-conv2 add.commute add-diff-cancel-left*)

**show**  $\bigwedge n \text{ na} .$

$$n + \text{na} \leq \text{ilen } w \implies F (\text{sub } n \ (n + \text{na}) \ w) \implies$$

$$\exists n \leq \text{ilen } w . \exists \text{na} \leq \text{ilen } w - n . F (\text{prefix } \text{na} \ (\text{suffix } n \ w))$$

**by** (*metis sub-def add-leD1 ilen-sub prefix-ilen-bound suffix-ilen le-add1*)

**qed**

**lemma** *ba-defs* :

$$(w \models \text{ba } F) = (\forall n \text{ na} . n + \text{na} \leq \text{ilen } w \longrightarrow ((\text{sub } n \ (n + \text{na}) \ w) \models F))$$

**by** (*auto simp add: ba-d-def da-defs*)

**lemma** *next-defs* :

$$(w \models \bigcirc F) = (\text{ilen } w > 0 \wedge ((\text{suffix } 1 \ w) \models F))$$

**using** *Suc-le-eq min.absorb1* **by** (*simp add: itl-def chop-defs skip-defs*) *force*

**lemma** *wnext-defs* :

$(w \models wnext\ F) = (ilen\ w = 0 \vee ((suffix\ 1\ w) \models F))$   
**by** (*simp add: wnext-d-def next-defs*)

**lemma** *prev-defs* :

$(w \models prev\ F) = (ilen\ w > 0 \wedge ((prefix\ ((ilen\ w) - 1)\ w) \models F))$   
**by** (*simp add: itl-def chop-defs skip-defs*)  
*(metis One-nat-def Suc-leI diff-diff-cancel diff-is-0-eq' diff-le-self neq0-conv zero-neq-one)*

**lemma** *wprev-defs* :

$(w \models wprev\ F) = (ilen\ w = 0 \vee ((prefix\ ((ilen\ w) - 1)\ w) \models F))$   
**by** (*metis (mono-tags, lifting) less-le prev-defs unl-lift wprev-d-def zero-le*)

**lemma** *more-defs* :

$(w \models more) = (ilen\ w > 0)$   
**by** (*simp add: more-d-def next-defs*)

**lemma** *empty-defs* :

$(w \models empty) = (ilen\ w = 0)$   
**by** (*simp add: empty-d-def more-defs*)

**lemma** *init-defs* :

$(w \models init\ F) = ((prefix\ 0\ w) \models F)$   
**using** *min.absorb1* **by** (*simp add: init-d-def empty-defs chop-defs*) *force*

**lemma** *initalt-defs* :

$(w \models bi(empty \longrightarrow F)) = ((prefix\ 0\ w) \models F)$   
**using** *min.absorb1* **by** (*simp add: bi-defs empty-defs*) *force*

**lemma** *fin-defs* :

$(w \models fin\ F) = ((suffix\ (ilen\ w)\ w) \models F)$   
**by** (*simp add: fin-d-def empty-defs always-defs*)

**lemma** *finalt-defs* :

$(w \models \#True;(F \wedge empty)) = ((suffix\ (ilen\ w)\ w) \models F)$   
**by** (*simp add: chop-defs empty-defs*) *fastforce*

**lemma** *halt-defs* :

$(w \models halt(F)) = (\forall n \leq ilen\ w. (ilen\ w = n) = F\ (suffix\ n\ w))$   
**by** (*simp add: halt-d-def empty-defs always-defs*)

**lemma** *initonly-defs* :

$(w \models initonly(F)) = (\forall n \leq ilen\ w. (n = 0) = F\ (prefix\ n\ w))$   
**using** *min.absorb1* **by** (*simp add: initonly-d-def bi-defs empty-defs*) *force*

**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* :  
 $(w \models \text{len } n) = (\text{ilen } w = n)$   
**proof**  
*(induct n arbitrary: w)*  
**case** 0  
**then show** ?case **by** (*simp add: len-d-def empty-defs*)  
**next**  
**case** (Suc n)  
**then show** ?case **by** (*simp add: len-d-def chop-defs skip-defs fastforce*)  
**qed**

**lemma** *currentval-defs* :  
 $(s \models \$v) = (v (\text{inth } s \ 0))$   
**by** (*simp add: current-val-d-def*)

**lemma** *nextval-defs* :  
 $(s \models v\$) = (\text{if } \text{ilen } s > 0 \text{ then } (v (\text{inth } s \ 1)) \text{ else } (\epsilon \ x. x=x))$   
**by** (*simp add: itl-def*)

**lemma** *finval-defs* :  
 $(s \models !v) = (v (\text{inth } s (\text{ilen } s)))$   
**by** (*simp add: itl-def*)

**lemma** *penultval-defs* :  
 $(s \models v!) = (\text{if } \text{ilen } s > 0 \text{ then } (v (\text{inth } s ((\text{ilen } s) - 1))) \text{ else } (\epsilon \ x. x=x))$   
**by** (*simp add: itl-def*)

**lemma** *next-assign-defs* :  
**assumes** *ilen s > 0*  
**shows**  $(s \models v := e) = v (\text{inth } s \ 1) = e \ s$   
**using** *assms* **by** (*auto simp: itl-def*)

**lemma** *prev-assign-defs* :  
**assumes** *ilen s > 0*  
**shows**  $(s \models v :=: e) = v (\text{inth } s ((\text{ilen } s) - 1)) = e \ s$   
**using** *assms* **by** (*auto simp: itl-def*)

**lemma** *always-eqv-defs* :  
 $(s \models v \approx e) = (\forall \ i \leq \text{ilen } s. v (\text{inth } s \ i) = e (\text{suffix } i \ s))$   
**by** (*simp add: always-eq-d-def always-defs current-val-d-def*)

**lemma** *temporal-assign-defs* :  
 $(s \models v \leftarrow e) = (v (\text{inth } s (\text{ilen } s)) = e \ s)$   
**by** (*simp add: itl-def*)

**lemma** *gets-defs* :  
 $(s \models v \text{ gets } e) = (\forall \ i < \text{ilen } s. v (\text{inth } s (\text{Suc } i)) = e (\text{sub } i \ (i+1) \ s))$   
**using** *Suc-le-eq min.absorb1 add-le-imp-le-diff prefix-suffix-ilen-good*  
**by** (*auto simp add: gets-d-def keep-d-def ba-defs skip-defs sub-def temporal-assign-defs*)

*fastforce*

**lemma** *stable-defs-helpa*:

**assumes**  $(\forall i < \text{ilen } s. v \text{ ( } \text{inth } s \text{ ( } \text{Suc } i \text{))} = v \text{ ( } \text{inth } s \text{ } i \text{))}$

$i \leq \text{ilen } s$

**shows**  $(v \text{ ( } \text{inth } s \text{ } i \text{))} = v \text{ ( } \text{inth } s \text{ } 0 \text{))}$

**using** *assms*

**proof** (*induct s arbitrary:i*)

**case** (*INil x*)

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

**next**

**case** (*ICons x1a s*)

**then show** *?case*

**proof** (*cases i*)

**case** *0*

**then show** *?thesis* **by** *blast*

**next**

**case** (*Suc nat*)

**then show** *?thesis*

**by** (*metis ICons.hyps ICons.prem1 ICons.prem2 Suc-le-mono Suc-mono inth-Suc inth-zero ilen.simps(2) plus-1-eq-Suc zero-less-Suc*)

**qed**

**qed**

**lemma** *stable-defs-helpb*:

**assumes**  $(\forall i \leq \text{ilen } s. v \text{ ( } \text{inth } s \text{ } i \text{))} = v \text{ ( } \text{inth } s \text{ } 0 \text{))}$

$i < \text{ilen } s$

**shows**  $v \text{ ( } \text{inth } s \text{ ( } \text{Suc } i \text{))} = v \text{ ( } \text{inth } s \text{ } i \text{))}$

**using** *assms*

**proof** (*induct s arbitrary:i*)

**case** (*INil x*)

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

**next**

**case** (*ICons x1a s*)

**then show** *?case*

**proof** (*cases i*)

**case** *0*

**then show** *?thesis* **using** *Suc-leI ICons.prem1 ICons.prem2* **by** *blast*

**next**

**case** (*Suc nat*)

**then show** *?thesis* **using** *ICons.prem1 ICons.prem2 Suc-leI less-imp-le-nat* **by** *presburger*

**qed**

**qed**

**lemma** *stable-defs-help*:

$(\forall i < \text{ilen } s. v \text{ ( } \text{inth } s \text{ ( } \text{Suc } i \text{))} = v \text{ ( } \text{inth } s \text{ } i \text{))} \longleftrightarrow$

$(\forall i \leq \text{ilen } s. v \text{ ( } \text{inth } s \text{ } i \text{))} = v \text{ ( } \text{inth } s \text{ } 0 \text{))}$

**proof** –

**have** *1*:  $(\forall i < \text{ilen } s. v \text{ ( } \text{inth } s \text{ ( } \text{Suc } i \text{))} = v \text{ ( } \text{inth } s \text{ } i \text{))} \longrightarrow$

```

      (∀ i ≤ ilen s. v ( inth s i) = v ( inth s 0))
using stable-defs-helpa by auto
have 2: (∀ i ≤ ilen s. v ( inth s i) = v ( inth s 0)) →
      (∀ i < ilen s. v ( inth s (Suc i)) = v ( inth s i))
using stable-defs-helpb by blast
show ?thesis using 1 2 by blast
qed

```

**lemma** *stable-defs*:

```

(s ⊨ stable v) = (∀ i ≤ ilen s. (v (inth s i)) = (v (inth s 0)))
by (simp add: stable-d-def gets-defs current-val-d-def sub-def stable-defs-help)

```

**lemma** *padded-defs* :

```

(s ⊨ padded v) = ((∀ i < ilen s. (v (inth s i)) = (v (inth s 0))) ∨ ilen s = 0)
proof (simp add: padded-d-def stable-defs chop-defs skip-defs empty-defs)
show ((∃ n ≤ ilen s.
      (∀ i. i ≤ n ∧ i ≤ ilen s → v (inth s i) = v (inth s 0)) ∧ ilen s - n = Suc 0) ∨
      ilen s = 0) =
      ((∀ i < ilen s. v ( inth s i) = v ( inth s 0)) ∨ ilen s = 0)
proof rule+
show ∧ i. (∃ n ≤ ilen s.
      (∀ i. i ≤ n ∧ i ≤ ilen s → v (inth s i) = v (inth s 0)) ∧ ilen s - n = Suc 0) ∨
      ilen s = 0 ⇒
      i < ilen s ⇒ v (inth s i) = v (inth s 0)
by (metis One-nat-def Suc-leI Suc-le-mono le-add-diff-inverse2 less-imp-le-nat not-less-zero
      plus-1-eq-Suc)
show (∀ i < ilen s. v (inth s i) = v (inth s 0)) ∨ ilen s = 0 ⇒
      (∃ n ≤ ilen s.
      (∀ i. i ≤ n ∧ i ≤ ilen s → v (inth s i) = v (inth s 0)) ∧ ilen s - n = Suc 0) ∨
      ilen s = 0
by (metis Suc-leI Suc-pred diff-diff-cancel diff-le-self gr-zeroI le-imp-less-Suc)
qed
qed

```

**lemma** *padded-temporal-assign-defs* :

```

(s ⊨ v <~ e) =
((s ⊨ padded v) ∧ (v ( inth s (ilen s)) = e s ))
by (auto simp add: padded-temp-assign-d-def padded-defs temporal-assign-defs)

```

**lemmas** *itl-defs* = *skip-defs chop-defs sometimes-defs always-defs di-defs bi-defs da-defs ba-defs next-defs*  
*wnext-defs prev-defs wprev-defs more-defs empty-defs init-defs initalt-defs fin-defs finalt-defs*  
*halt-defs initalonly-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*

## 2.5 Soundness of Finite ITL Axioms

### 2.5.1 ChopAssoc

**lemma** *ChopAssocSemHelpa*:

**assumes**  $(\exists i \text{ ia} . i \leq \text{ilen } \sigma \wedge \text{ia} \leq \text{ilen } \sigma - i \wedge (\text{prefix } i \text{ } \sigma \models f) \wedge$   
 $(\text{prefix } \text{ia} (\text{suffix } i \text{ } \sigma) \models g) \wedge (\text{suffix } (\text{ia} + i) \text{ } \sigma \models h))$   
**shows**  $(\exists j \text{ ja} . j \leq \text{ilen } \sigma \wedge \text{ja} \leq j \wedge (\text{prefix } \text{ja} (\text{prefix } j \text{ } \sigma) \models f) \wedge$   
 $(\text{suffix } \text{ja} (\text{prefix } j \text{ } \sigma) \models g) \wedge (\text{suffix } j \text{ } \sigma \models h))$   
**proof** –  
**have** 1:  $(\exists i \text{ ia} . i \leq \text{ilen } \sigma \wedge \text{ia} \leq \text{ilen } \sigma - i \wedge (\text{prefix } i \text{ } \sigma \models f) \wedge$   
 $(\text{prefix } \text{ia} (\text{suffix } i \text{ } \sigma) \models g) \wedge (\text{suffix } (\text{ia} + i) \text{ } \sigma \models h))$   
**using** *assms* **by** *auto*  
**obtain** *i ia* **where** 2:  $i \leq \text{ilen } \sigma \wedge \text{ia} \leq \text{ilen } \sigma - i \wedge (\text{prefix } i \text{ } \sigma \models f) \wedge$   
 $(\text{prefix } \text{ia} (\text{suffix } i \text{ } \sigma) \models g) \wedge (\text{suffix } (\text{ia} + i) \text{ } \sigma \models h)$   
**using** 1 **by** *auto*  
**have** 3:  $(\text{suffix } (\text{ia} + i) \text{ } \sigma \models h)$   
**using** 2 **by** *auto*  
**have** 4:  $\text{ia} + i \leq \text{ilen } \sigma$   
**using** 2 *Nat.le-diff-conv2* **by** *blast*  
**have** 5:  $i \leq \text{ia} + i$   
**by** *simp*  
**have** 6:  $(\text{suffix } i (\text{prefix } (\text{ia} + i) \text{ } \sigma) \models g)$   
**using** 2 4 *suffix-prefix-swap* **by** *force*  
**have** 7:  $(\text{prefix } i (\text{prefix } (\text{ia} + i) \text{ } \sigma) \models f)$   
**by**  $(\text{simp add: 2 add.commute})$   
**show** *?thesis* **using** 2 4 5 6 7 **by** *blast*  
**qed**

**lemma** *ChopAssocSemHelpb*:

**assumes**  $(\exists j \text{ ja} . j \leq \text{ilen } \sigma \wedge \text{ja} \leq j \wedge (\text{prefix } \text{ja} (\text{prefix } j \text{ } \sigma) \models f) \wedge$   
 $(\text{suffix } \text{ja} (\text{prefix } j \text{ } \sigma) \models g) \wedge (\text{suffix } j \text{ } \sigma \models h))$   
**shows**  $(\exists i \text{ ia} . i \leq \text{ilen } \sigma \wedge \text{ia} \leq \text{ilen } \sigma - i \wedge (\text{prefix } i \text{ } \sigma \models f) \wedge$   
 $(\text{prefix } \text{ia} (\text{suffix } i \text{ } \sigma) \models g) \wedge (\text{suffix } (\text{ia} + i) \text{ } \sigma \models h))$   
**proof** –  
**have** 1:  $(\exists j \text{ ja} . j \leq \text{ilen } \sigma \wedge \text{ja} \leq j \wedge (\text{prefix } \text{ja} (\text{prefix } j \text{ } \sigma) \models f) \wedge$   
 $(\text{suffix } \text{ja} (\text{prefix } j \text{ } \sigma) \models g) \wedge (\text{suffix } j \text{ } \sigma \models h))$   
**using** *assms* **by** *auto*  
**obtain** *j ja* **where** 2:  $j \leq \text{ilen } \sigma \wedge \text{ja} \leq j \wedge (\text{prefix } \text{ja} (\text{prefix } j \text{ } \sigma) \models f) \wedge$   
 $(\text{suffix } \text{ja} (\text{prefix } j \text{ } \sigma) \models g) \wedge (\text{suffix } j \text{ } \sigma \models h)$   
**using** 1 **by** *auto*  
**have** 3:  $\text{ja} \leq \text{ilen } \sigma$   
**using** 2 *le-trans* **by** *blast*  
**have** 4:  $j - \text{ja} \leq \text{ilen } \sigma - \text{ja}$   
**by**  $(\text{simp add: 2 diff-le-mono})$   
**have** 5:  $(\text{prefix } \text{ja} \text{ } \sigma \models f)$   
**by**  $(\text{metis 2 pref-pref-3 le-add-diff-inverse})$   
**have** 6:  $(\text{prefix } (j - \text{ja}) (\text{suffix } \text{ja} \text{ } \sigma) \models g)$   
**by**  $(\text{simp add: 2 suffix-prefix-swap})$   
**have** 7:  $(\text{suffix } ((j - \text{ja}) + \text{ja}) \text{ } \sigma \models h)$   
**by**  $(\text{simp add: 2})$   
**show** *?thesis* **using** 3 4 5 6 7 **by** *blast*  
**qed**

**lemma** *ChopAssocSemHelp*:

$(\exists i \text{ ia} . i \leq \text{ilen } \sigma \wedge \text{ia} \leq \text{ilen } \sigma - i \wedge (\text{prefix } i \sigma \models f) \wedge$   
 $(\text{prefix } \text{ia} (\text{suffix } i \sigma) \models g) \wedge (\text{suffix } (\text{ia} + i) \sigma \models h)) =$   
 $(\exists j \text{ ja} . j \leq \text{ilen } \sigma \wedge \text{ja} \leq j \wedge (\text{prefix } \text{ja} (\text{prefix } j \sigma) \models f) \wedge$   
 $(\text{suffix } \text{ja} (\text{prefix } j \sigma) \models g) \wedge (\text{suffix } j \sigma \models h))$   
**using** *ChopAssocSemHelpa*[of  $\sigma f g h$ ]  
*ChopAssocSemHelpb*[of  $\sigma f g h$ ] **by** *auto*

**lemma** *ChopAssocSemHelp2*:

$(\sigma \models f ; (g ; h)) = (\sigma \models (f;g);h)$

**proof** –

**have**  $(\sigma \models f ; (g ; h)) =$   
 $((\exists i \leq \text{ilen } \sigma . (\text{prefix } i \sigma \models f) \wedge (\exists \text{ia} \leq \text{ilen } (\text{suffix } i \sigma).$   
 $(\text{prefix } \text{ia} (\text{suffix } i \sigma) \models g) \wedge (\text{suffix } (\text{ia} + i) \sigma \models h))))$

**by** (*simp add: chop-defs*)

**also have** ... =

$(\exists i \text{ ia} . i \leq \text{ilen } \sigma \wedge \text{ia} \leq \text{ilen } \sigma - i \wedge (\text{prefix } i \sigma \models f) \wedge$   
 $(\text{prefix } \text{ia} (\text{suffix } i \sigma) \models g) \wedge (\text{suffix } (\text{ia} + i) \sigma \models h))$

**by** *fastforce*

**also have** ... =

$(\exists j \text{ ja} . j \leq \text{ilen } \sigma \wedge \text{ja} \leq j \wedge (\text{prefix } \text{ja} (\text{prefix } j \sigma) \models f) \wedge$   
 $(\text{suffix } \text{ja} (\text{prefix } j \sigma) \models g) \wedge (\text{suffix } j \sigma \models h))$

**using** *ChopAssocSemHelp*[of  $\sigma f g h$ ] **by** *blast*

**also have** ... =

$(\exists i \leq \text{ilen } \sigma . (\exists \text{ia} \leq \text{ilen } (\text{prefix } i \sigma). (\text{prefix } \text{ia} (\text{prefix } i \sigma) \models f) \wedge$   
 $(\text{suffix } \text{ia} (\text{prefix } i \sigma) \models g)) \wedge (\text{suffix } i \sigma \models h))$

**by** *fastforce*

**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** *ChopAssocSemHelp2* **using** *unl-lift2* **by** *blast*

## 2.5.2 OrChopImp

**lemma** *OrChopImpSem*:

$(\sigma \models (f \vee g);h \longrightarrow f;h \vee g;h)$

**by** (*simp add: itl-defs*) *blast*

## 2.5.3 ChopOrImp

**lemma** *ChopOrImpSem*:

$(\sigma \models f;(g \vee h) \longrightarrow f;g \vee f;h)$

**by** (*simp add: itl-defs*) *blast*

## 2.5.4 EmptyChop

**lemma** *EmptyChopSem*:

$(\sigma \models \text{empty} ; f = f)$

using *min.absorb1* by (simp add: *itl-defs*) force

### 2.5.5 ChopEmpty

lemma *ChopEmptySem*:  
 $(\sigma \models f; \text{empty} = f)$   
 by (simp add: *itl-defs*) auto

### 2.5.6 StateImpBi

lemma *StateImpBiSem*:  
 $(\sigma \models \text{init } f \longrightarrow \text{bi } (\text{init } f))$   
 by (simp add: *itl-defs*)

### 2.5.7 NextImpNotNextNot

lemma *NextImpNotNextNotSem*:  
 $(\sigma \models \bigcirc f \longrightarrow \neg (\bigcirc (\neg f)))$   
 by (simp add: *itl-defs*)

### 2.5.8 BiBoxChopImpChop

lemma *BiBoxChopImpChopSem*:  
 $(\sigma \models \text{bi } (f \longrightarrow f1) \wedge \Box (g \longrightarrow g1) \longrightarrow f; g \longrightarrow f1; g1)$   
 by (simp add: *itl-defs*) fastforce

### 2.5.9 BoxInduct

lemma *box-induct-help-1* :  
 assumes  $(f \ \sigma)$   
 $(\forall i. \text{Suc } 0 \leq \text{ilen } \sigma - i \longrightarrow$   
 $i \leq \text{ilen } \sigma \longrightarrow (\text{suffix } i \ \sigma \models f) \longrightarrow f (\text{suffix } (\text{Suc } i) \ \sigma))$   
 shows  $(\forall j. j \leq \text{ilen } \sigma \longrightarrow f (\text{suffix } j \ \sigma))$   
 proof  
 fix  $j$   
 show  $j \leq \text{ilen } \sigma \longrightarrow f (\text{suffix } j \ \sigma)$   
 using *assms*  
 proof  
 (induct  $j$  arbitrary:  $\sigma$ )  
 case 0  
 then show ?case by simp  
 next  
 case (Suc  $j$ )  
 then show ?case  
 by (metis *Nat.le-diff-conv2 One-nat-def Suc-eq-plus1-left Suc-leD*)  
 qed  
 qed

lemma *BoxInductSem*:  
 $(\sigma \models \Box (f \longrightarrow \text{wnext } f) \wedge f \longrightarrow \Box f)$   
 proof –  
 have 1:  $(\sigma \models \Box (f \longrightarrow \text{wnext } f) \wedge f \longrightarrow \Box f) =$

```

      (( $\forall n \leq \text{ilen } \sigma. f (\text{suffix } n \ \sigma) \longrightarrow \text{ilen } \sigma = n \vee f (\text{suffix } (\text{Suc } n) \ \sigma)$ )  $\wedge f \ \sigma \longrightarrow$ 
       ( $\forall n \leq \text{ilen } \sigma. f (\text{suffix } n \ \sigma)$ ))
    by (simp add: always-defs wnext-defs)
  from 1 show ?thesis using box-induct-help-1[of f  $\sigma$ ] by fastforce
qed

```

### 2.5.10 ChopStarEqv

**lemma** *ChopExist*:

```

 $\vdash (\exists k. f;g \ k) = f;(\exists k. g \ k)$ 
by (auto simp add: itl-defs Valid-def)

```

**lemma** *ExistChop*:

```

 $\vdash (\exists k. (g \ k);f) = (\exists k. g \ k);f$ 
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  $\bigwedge x. \sigma \models (\text{power } f \ x) \implies \forall k. \neg(\sigma \models (f;\text{power } f \ k)) \implies \sigma \models \text{empty}$ 
  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  $\bigwedge k. \sigma \models (f;\text{power } f \ k) \implies \exists x. \sigma \models (\text{power } f \ x)$ 
  by (metis pow-Suc)
qed

```

**lemma** *powersem*:

```

 $\vdash (\exists k. \text{power } f \ k) = (\text{empty} \vee (f);(\exists k. (\text{power } f \ k)))$ 
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:  $\vdash (\exists k. \text{power } f (\text{Suc } k)) = (\exists k. (f);\text{power } f \ k)$ 
  by simp
  have 3:  $\vdash (\exists k. (f);(\text{power } f \ k)) = (f);(\exists k. (\text{power } f \ k))$ 
  using ChopExist by blast
  from 1 2 3 show ?thesis by fastforce
qed

```

**lemma** *PowerstarEqvSem*:

```

( $\sigma \models (\text{powerstar } f) = (\text{empty} \vee f;(\text{powerstar } f))$ )
proof -
  have 1:  $(\sigma \models (\text{powerstar } f)) =$ 
    ( $\sigma \models (\exists k. \text{power } f \ k)$ )
  by (simp add: powerstar-d-def)
  have 2:  $(\sigma \models (\exists k. \text{power } f \ k)) =$ 
    ( $\sigma \models (\text{empty} \vee f;(\exists k. (\text{power } f \ k)))$ )
  using powersem by (metis inteq-reflection)
  from 1 2 show ?thesis by (simp add: powerstar-d-def)
qed

```

**lemma** *ChopstarEqvSem*:

$(\sigma \models f^* = (\text{empty} \vee (f \wedge \text{more}); f^*))$

**by** (*metis PowerstarEqvSem chopstar-d-def*)

## 2.6 Quantification over State (Flexible) Variables

The hidden state approach, as used in the embedding of TLA in Isabelle/HOL TLA embedding [3, 2], is used. Here [3, 2], a state space is defined by its projections, and everything else is unknown. Thus, a variable is a projection of the state space, and has the same type as a state function. Moreover, strong typing is achieved, since the projection function may have any result type. To achieve this, the state space is represented by an undefined type, which is an instance of the *world* class to enable use with the *Intensional* theory.

**type-synonym** *'a statefun* = (*state*, *'a*) *stfun*

**type-synonym** *statepred* = *bool statefun*

**type-synonym** *'a tempfun* = (*state*, *'a*) *formfun*

**type-synonym** *temporal* = *state formula*

Similar to [3, 2] we define a state to be an anonymous type whose only purpose is to provide Skolem IConstants. Similarly, we do not define a type of state variables separate from that of arbitrary state functions, again in order to simplify the definition of flexible quantification later on. Nevertheless, we need to distinguish state variables. Note we deviate from [3, 2] in that we do not use axioms but use definitions and lemmas.

## 2.7 Temporal Quantifiers

**definition** *exist-state-d* :: (*'a statefun*  $\Rightarrow$  *temporal*)  $\Rightarrow$  *temporal* (**binder** *Eex* 10)

**where** *exist-state-d* *F*  $\equiv$  ( $\lambda s. (\exists x. s \models F x)$ )

**syntax**

*-Eex* :: [*idts*, *lift*]  $\Rightarrow$  *lift*  $((\exists \exists \exists \text{ -./ -}) [0,10] 10)$

**translations**

*-Eex v A* == *Eex v. A*

**definition** *forall-state-d* :: (*'a statefun*  $\Rightarrow$  *temporal*)  $\Rightarrow$  *temporal* (**binder** *Aall* 10)

**where** *forall-state-d* *F*  $\equiv$  *LIFT*( $\neg(\exists \exists \exists x. \neg(F x))$ )

**syntax**

*-Aall* :: [*idts*, *lift*]  $\Rightarrow$  *lift*  $((\exists \forall \forall \text{ -./ -}) [0,10] 10)$

**translations**

*-Aall v A* == *Aall v. A*

**end**



### 3 Fuse operator

```
theory Fuse
imports Semantics
begin
```

This theory introduces the fuse operator.

#### 3.1 Definitions

```
primrec fuse :: 'a interval  $\Rightarrow$  'a interval  $\Rightarrow$  'a interval
where fuse-INil : fuse  $\langle x \rangle$  ys = ys
| fuse-ICons : fuse (x  $\odot$  xs) ys = x  $\odot$  (fuse xs ys)
```

```
primrec lfuse :: 'a interval interval  $\Rightarrow$  'a interval
where lfuse-INil : lfuse  $\langle xs \rangle$  = xs
| lfuse-ICons : lfuse (x  $\odot$  xs) = fuse x (lfuse xs)
```

```
primrec lastfirst :: 'a interval interval  $\Rightarrow$  bool
where lastfirst  $\langle xs \rangle$  = True
| lastfirst (xs  $\odot$  xxs) =
  ( ((ilast xs) = (ifirst (ifirst xxs)))  $\wedge$  (lastfirst xxs))
```

#### 3.2 Lemmas

```
lemma fuse-ilen :
  assumes ilast xs = ifirst ys
  shows ilen (fuse xs ys) = (ilen xs) + (ilen ys)
using assms by (induct xs) simp-all
```

```
lemma fuse-ilen-a:
  ilen(fuse xs ys) = ilen xs + ilen ys
proof
  (induct xs arbitrary: ys)
  case (INil x)
  then show ?case by simp
next
  case (ICons x1a xs)
  then show ?case by simp
qed
```

```
lemma fuse-inth:
  assumes  $i \leq \text{ilen} (fuse\ xs\ ys)$ 
  ilast xs = ifirst ys
  shows ( $i \leq \text{ilen}\ xs \longrightarrow \text{inth} (fuse\ xs\ ys)\ i = \text{inth}\ xs\ i$ )
   $\wedge$ 
  ( $(\text{ilen}\ xs \leq i \wedge i \leq \text{ilen} (fuse\ xs\ ys)) \longrightarrow \text{inth} (fuse\ xs\ ys)\ i = \text{inth}\ ys\ (i - \text{ilen}\ xs)$ )
```

```
using assms
proof
```

```

(induct xs arbitrary: i)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
using less-Suc-eq-0-disj less-Suc-eq-le by fastforce
qed

```

```

lemma fuse-inth-a:
  assumes  $j \leq \text{ilen } ys$ 
          $\text{ilast } xs = \text{ifirst } ys$ 
  shows  $\text{inth } (\text{fuse } xs \ ys) \ (\text{ilen } xs + j) = (\text{inth } ys \ j)$ 
using assms
by (simp add: fuse-ilen-a fuse-inth)

```

```

lemma fuse-leftneutral :
   $\text{fuse } (\text{INil } (\text{ifirst } xs)) \ xs = xs$ 
by simp

```

```

lemma fuse-rightneutral :
   $\text{fuse } xs \ (\text{INil } (\text{ilast } xs)) = xs$ 
by (induct xs) simp-all

```

```

lemma ifirst-fuse :
  assumes  $\text{ilast } xs = \text{ifirst } ys$ 
  shows  $\text{ifirst } (\text{fuse } xs \ ys) = \text{ifirst } xs$ 
using assms by (induct xs) simp-all

```

```

lemma ilast-fuse :
  assumes  $\text{ilast } xs = \text{ifirst } ys$ 
  shows  $\text{ilast } (\text{fuse } xs \ ys) = \text{ilast } ys$ 
using assms by (induct xs) simp-all

```

```

lemma FusionAssoc :
  assumes  $(\text{ilast } xs) = (\text{ifirst } ys)$ 
          $(\text{ilast } ys) = (\text{ifirst } zs)$ 
  shows  $(\text{fuse } xs \ (\text{fuse } ys \ zs)) = (\text{fuse } (\text{fuse } xs \ ys) \ zs)$ 
using assms by (induct xs) simp-all

```

```

lemma ilast-ifirst:
   $(\text{ilast } (\text{prefix } i \ xs)) = (\text{ifirst } (\text{suffix } i \ xs))$ 
using ilast-ifirst by blast

```

```

lemma prefix-fuse :
  assumes  $\text{ilast } xs = \text{ifirst } ys$ 
  shows  $(\text{prefix } (\text{ilen } xs) \ (\text{fuse } xs \ ys)) = xs$ 
using assms by (induct xs arbitrary: ys) simp-all

```

**lemma** *suffix-fuse* :  
**assumes**  $ilast\ xs = ifirst\ ys$   
**shows**  $(suffix\ (ilen\ xs)\ (fuse\ xs\ ys)) = ys$   
**using** *assms* **by** (*induct xs arbitrary: ys simp-all*)

**lemma** *fuse-prefix-suffix-ilen* :  
**assumes**  $n \leq ilen\ xs$   
**shows**  $ilen\ (fuse\ (prefix\ n\ xs)\ (suffix\ n\ xs)) = ilen\ xs$   
**using** *assms*  
**by** (*metis fuse-ilen-a prefix-ilen-good suffix-ilen-good le-add-diff-inverse*)

**lemma** *fuse-prefix-suffix-inth* :  
**assumes**  $n \leq ilen\ xs$   
 $i \leq ilen\ xs$   
**shows**  $inth\ (fuse\ (prefix\ n\ xs)\ (suffix\ n\ xs))\ i = inth\ xs\ i$   
**using** *assms fuse-inth[of i (prefix n xs) (suffix n xs)]*  
 $ilast-ifirst[of\ n\ xs]$   
 $fuse-prefix-suffix-ilen[of\ n\ xs]$   
**by** (*metis diff-add inth-prefix inth-suffix prefix-ilen-good le-cases*  
 $le-diff-conv\ ordered-cancel-comm-monoid-diff-class.add-diff-inverse$ )

**lemma** *fuse-prefix-suffix*:  
**assumes**  $n \leq ilen\ xs$   
**shows**  $fuse\ (prefix\ n\ xs)\ (suffix\ n\ xs) = xs$   
**using** *assms*  
**by** (*simp add: fuse-prefix-suffix-ilen fuse-prefix-suffix-inth interval-eq-inth-eq*)

**lemma** *chop-fuse-1* :  
 $(\exists\ \sigma 1\ \sigma 2. \sigma = fuse\ \sigma 1\ \sigma 2 \wedge$   
 $(\sigma 1 \models f) \wedge (\sigma 2 \models g) \wedge$   
 $(ilast\ \sigma 1 = ifirst\ \sigma 2)) =$   
 $(\exists\ i. 0 \leq i \wedge i \leq ilen\ \sigma \wedge (prefix\ i\ \sigma \models f) \wedge (suffix\ i\ \sigma \models g))$   
**by** (*metis fuse-ilen fuse-prefix-suffix ilast-ifirst*  
 $ilen-gr-zero\ prefix-fuse\ suffix-fuse\ le-add-same-cancel1$ )

**lemma** *chop-fuse-2* :  
 $(\exists\ \sigma 1\ \sigma 2. \sigma = fuse\ \sigma 1\ \sigma 2 \wedge$   
 $(\sigma 1 \in X) \wedge (\sigma 2 \in Y) \wedge$   
 $(ilast\ \sigma 1 = ifirst\ \sigma 2)) =$   
 $(\exists\ i \leq ilen\ \sigma. (prefix\ i\ \sigma) \in X \wedge (suffix\ i\ \sigma) \in Y)$   
**by** (*metis fuse-ilen fuse-prefix-suffix ilast-ifirst prefix-fuse suffix-fuse le-add1*)

**lemma** *chop-fuse*:  
 $(\exists\ \sigma 1\ \sigma 2. \sigma = fuse\ \sigma 1\ \sigma 2 \wedge$   
 $(\sigma 1 \models f) \wedge (\sigma 2 \models g) \wedge$   
 $(ilast\ \sigma 1 = ifirst\ \sigma 2)) =$   
 $(\sigma \models f;g)$   
**by** (*metis chop-defs fuse-ilen-a fuse-prefix-suffix ilast-ifirst*  
 $prefix-fuse\ suffix-fuse\ le-add1$ )

**lemma** *sub-fuse*:

**assumes**  $k \leq n$

$n \leq m$

$m \leq \text{ilen } xs$

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

**proof** –

**have** 1:  $\text{ilast}(\text{sub } k \ n \ xs) = (\text{inth } xs \ n)$

**using** *assms ilast-sub le-trans less-imp-le-nat* **by** *blast*

**have** 2:  $\text{ifirst}(\text{sub } n \ m \ xs) = (\text{inth } xs \ n)$

**using** *assms ifirst-sub less-imp-le-nat* **by** *blast*

**have** 3:  $\text{ilen}(\text{fuse } (\text{sub } k \ n \ xs) \ (\text{sub } n \ m \ xs)) = \text{ilen}(\text{sub } k \ m \ xs)$

**by** (*metis Nat.add-diff-assoc2 assms(1) assms(2) assms(3) fuse-ilen-a ilen-sub le-trans ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)

**have** 4:  $(\forall i. i \leq \text{ilen}(\text{sub } k \ m \ xs) \longrightarrow (\text{inth } (\text{fuse } (\text{sub } k \ n \ xs) \ (\text{sub } n \ m \ xs)) \ i) = (\text{inth } xs \ (k+i)))$

**proof**

**fix**  $i$

**show**  $i \leq \text{ilen}(\text{sub } k \ m \ xs) \longrightarrow \text{inth } (\text{fuse } (\text{sub } k \ n \ xs) \ (\text{sub } n \ m \ xs)) \ i = \text{inth } xs \ (k + i)$

**proof** –

**have** 41:  $\text{ilen}(\text{sub } k \ m \ xs) = (m-k)$

**using** *assms ilen-sub le-trans less-imp-le-nat* **by** *metis*

**have** 42:  $i \leq \text{ilen}(\text{sub } k \ m \ xs) \longrightarrow \text{inth } (\text{fuse } (\text{sub } k \ n \ xs) \ (\text{sub } n \ m \ xs)) \ i =$   
 $(\text{if } i \leq \text{ilen}(\text{sub } k \ n \ xs) \text{ then } (\text{inth } (\text{sub } k \ n \ xs) \ i)$   
 $\text{else } (\text{inth } (\text{sub } n \ m \ xs) \ (i - \text{ilen}(\text{sub } k \ n \ xs))))$

**by** (*metis 1 2 3 fuse-inth le-cases*)

**have** 43:  $i \leq (m-k) \longrightarrow \text{inth } (\text{fuse } (\text{sub } k \ n \ xs) \ (\text{sub } n \ m \ xs)) \ i =$   
 $(\text{if } i \leq (n-k) \text{ then } (\text{inth } xs \ (k+i)) \text{ else } (\text{inth } xs \ (n+(i-(n-k)))))$

**using** 42 *Nat.le-diff-conv2 assms(1) assms(2) assms(3)* **by** *auto*

**have** 44:  $i \leq (m-k) \longrightarrow \text{inth } (\text{fuse } (\text{sub } k \ n \ xs) \ (\text{sub } n \ m \ xs)) \ i =$   
 $(\text{inth } xs \ (k+i))$

**by** (*simp add: 43 add.commute assms less-imp-le-nat*)

**show** *?thesis*

**by** (*simp add: 41 44*)

**qed**

**qed**

**have** 5:  $(\forall i. i \leq \text{ilen}(\text{sub } k \ m \ xs) \longrightarrow (\text{inth } (\text{sub } k \ m \ xs) \ i) = (\text{inth } xs \ (k+i)))$

**using** *assms(1) assms(2) assms(3)* **by** *auto*

**show** *?thesis*

**by** (*simp add: 3 4 5 interval-eq-inth-eq*)

**qed**

**lemma** *sub-fuse-idx*:

**assumes** *index-sequence 0 l*

$\text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma$

$(\text{Suc } i) < \text{ilen } l$

**shows**  $\text{fuse } (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma) \ (\text{sub } (\text{inth } l \ (\text{Suc } i)) \ (\text{inth } l \ (\text{ilen } l)) \ \sigma) =$   
 $(\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{ilen } l)) \ \sigma)$

**proof** –

**have** 1:  $\text{ilast}(\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma) = (\text{inth } \sigma \ (\text{inth } l \ (\text{Suc } i)))$

**by** (*metis assms idx-less-equal idx-less-last-1 ilast-sub le-add2 less-imp-le-nat plus-1-eq-Suc*)

```

have 2:  $\text{ifirst}(\text{sub } (\text{inth } l \text{ (Suc } i)) (\text{inth } l \text{ (ilen } l)) \sigma) = (\text{inth } \sigma (\text{inth } l \text{ (Suc } i)))$ 
  by (metis assms dual-order.strict-iff-order eq-imp-le idx-less-last-1 ifirst-sub)
have 3:  $(\text{inth } l \text{ } i) < (\text{inth } l \text{ (Suc } i))$ 
  using Suc-lessD assms idx-less-than less-imp-le-nat by blast
have 4:  $(\text{inth } l \text{ (Suc } i)) < (\text{inth } l \text{ (ilen } l))$ 
  using assms idx-less-last-1 by blast
show ?thesis using 3 4 assms by (simp add: sub-fuse)
qed

```

**lemma** *idx-fuse-ifirst-ilast*:

```

assumes index-sequence 0 l1
  index-sequence 0 l2
   $\text{inth } l1 \text{ (ilen } l1) = cp$ 
   $\text{inth } l2 \text{ (ilen } l2) = \text{ilen } \sigma - cp$ 
   $cp \leq \text{ilen } \sigma$ 
   $l = \text{fuse } l1 \text{ (imap (shift } cp) \text{ } l2)$ 
shows  $\text{ilast } l1 = \text{ifirst } (\text{imap (shift } cp) \text{ } l2)$ 
using assms
by (metis shift-def add.left-neutral index-sequence-def inth-imap )

```

**lemma** *idx-fuse-inth-cp*:

```

assumes index-sequence 0 l1
  index-sequence 0 l2
   $\text{inth } l1 \text{ (ilen } l1) = cp$ 
   $\text{inth } l2 \text{ (ilen } l2) = \text{ilen } \sigma - cp$ 
   $cp \leq \text{ilen } \sigma$ 
   $l = \text{fuse } l1 \text{ (imap (shift } cp) \text{ } l2)$ 
   $i \leq \text{ilen } l2$ 
shows  $\text{inth } l \text{ (ilen } l1 + i) = cp + \text{inth } l2 \text{ } i$ 
proof -
have 1:  $\text{ilast } l1 = \text{ifirst } (\text{imap (shift } cp) \text{ } l2)$ 
  using assms idx-fuse-ifirst-ilast by blast
have 2:  $\text{inth } l \text{ (ilen } l1 + i) = \text{inth } (\text{imap (shift } cp) \text{ } l2) \text{ } i$ 
  using assms by (metis 1 fuse-inth-a ilen-imap)
have 3:  $\text{inth } (\text{imap (shift } cp) \text{ } l2) \text{ } i = \text{inth } l2 \text{ } i + cp$ 
  by (simp add: shift-def inth-imap)
show ?thesis using 2 3 by auto
qed

```

**lemma** *idx-fuse-idx*:

```

assumes index-sequence 0 l1
  index-sequence 0 l2
   $\text{inth } l1 \text{ (ilen } l1) = cp$ 
   $\text{inth } l2 \text{ (ilen } l2) = \text{ilen } \sigma - cp$ 
   $cp \leq \text{ilen } \sigma$ 
   $l = \text{fuse } l1 \text{ (imap (shift } cp) \text{ } l2)$ 
   $i \leq \text{ilen } l2$ 
shows index-sequence 0 l

```

**proof** –

**have** 1:  $ilast\ l1 = ifirst\ (imap\ (shift\ cp)\ l2)$   
**using** *assms idx-fuse-ifirst-ilast* **by** *blast*  
**have** 2:  $inth\ (fuse\ l1\ (imap\ (shift\ cp)\ l2))\ 0 = inth\ l1\ 0$   
**using** 1 *fuse-inth* **by** *blast*  
**have** 3:  $ilen\ (fuse\ l1\ (imap\ (shift\ cp)\ l2)) = ilen\ l1 + ilen\ l2$   
**by** (*simp add: fuse-ilen-a*)  
**have** 4:  $\forall i. 0 \leq i \wedge i \leq ilen\ l1 \longrightarrow$   
 $inth\ (fuse\ l1\ (imap\ (shift\ cp)\ l2))\ i =$   
 $inth\ l1\ i$   
**by** (*metis 1 inth-prefix prefix-fuse*)  
**have** 5:  $\forall i. ilen\ l1 \leq i \wedge i \leq ilen\ l1 + ilen\ l2 \longrightarrow$   
 $inth\ (fuse\ l1\ (imap\ (shift\ cp)\ l2))\ i =$   
 $cp + inth\ l2\ (i - ilen\ l1)$   
**by** (*metis (no-types, lifting) 1 3 shift-def add.commute fuse-inth inth-imap*)  
**have** 6:  $\forall i. 0 \leq i \wedge i < ilen\ l1 + ilen\ l2 \longrightarrow$   
 $inth\ (fuse\ l1\ (imap\ (shift\ cp)\ l2))\ i < inth\ (fuse\ l1\ (imap\ (shift\ cp)\ l2))\ (Suc\ i)$   
**using** *assms* **by** (*metis 1 3 4 add.right-neutral index-sequence-def idx-link*  
*idx-split ilen-gr-zero prefix-fuse suffix-fuse le-add1*)  
**have** 7:  $index-sequence\ 0\ l =$   
 $((inth\ l1\ 0) = 0 \wedge$   
 $(\forall i. 0 \leq i \wedge i < ilen\ l1 + ilen\ l2 \longrightarrow$   
 $inth\ (fuse\ l1\ (imap\ (shift\ cp)\ l2))\ i < inth\ (fuse\ l1\ (imap\ (shift\ cp)\ l2))\ (Suc\ i)))$   
**by** (*simp add: 2 3 assms index-sequence-def*)  
**from** 7 6 2 **show** *?thesis*  
**using** *assms index-sequence-def* **by** *auto*  
**qed**

**lemma** *idx-fuse-ilen*:

**assumes** *index-sequence 0 l1*  
*index-sequence 0 l2*  
 $inth\ l1\ (ilen\ l1) = cp$   
 $inth\ l2\ (ilen\ l2) = ilen\ \sigma - cp$   
 $cp \leq ilen\ \sigma$   
 $l = fuse\ l1\ (imap\ (shift\ cp)\ l2)$   
 $i \leq ilen\ l2$   
**shows**  $inth\ l\ (ilen\ l) = ilen\ \sigma$   
**using** *assms idx-fuse-inth-cp[of l1 l2 cp  $\sigma$ ]* **by** (*simp add: fuse-ilen-a*)

**lemma** *ifirst-lfuse-ifirst*:

**assumes** *lastfirst (xs  $\odot$  xxs)*  
**shows**  $ifirst(lfuse\ xxs) = ifirst(ifirst\ xxs)$   
**using** *assms*  
**proof**  
*(induct xxs)*  
**case** (*INil x*)  
**then show** *?case* **by** *simp*  
**next**  
**case** (*ICons x1a xxs*)

```

then show ?case
  proof (cases x1a)
    case (INil x1)
    then show ?thesis using ICons.hyps ICons.prem by auto
  next
    case (ICons x21 x22)
    then show ?thesis by simp
  qed
qed

```

```

lemma ifirst-lfuse:
  assumes lastfirst (xs  $\odot$  xxs)
  shows (ifirst (lfuse (xs  $\odot$  xxs)) ) = (ifirst xs)
  proof -
    have 1: lastfirst (xs  $\odot$  xxs)
    using assms by auto
    have 2: ( (ilast xs) = (ifirst (ifirst xxs)))  $\wedge$  (lastfirst xxs)
    using 1 by simp
    have 3: (ilast xs) = (ifirst (ifirst xxs))
    using 2 by auto
    have 4: ifirst (lfuse (xs  $\odot$  xxs)) = ifirst(fuse xs (lfuse xxs))
    by simp
    have 5: ifirst(lfuse xxs) = ifirst(ifirst xxs)
    using assms ifirst-lfuse-ifirst by blast
    have 6: ifirst(fuse xs (lfuse xxs)) = ifirst xs
    by (metis 3 5 ifirst-fuse)
    show ?thesis using 6 by auto
  qed

```

```

lemma lastfirst-lfuse-ilast:
  assumes lastfirst xxs
  shows ilast(lfuse xxs) = ilast(ilast xxs)
  using assms
  proof
    (induct xxs)
    case (INil x)
    then show ?case by simp
  next
    case (ICons x1a xxs)
    then show ?case
    by (metis fuse-ilen-a fuse-inth-a ifirst-lfuse-ifirst
        inth-last lastfirst.simps(2) lfuse-ICons order.order-iff-strict)
  qed

```

```

lemma lastfirst-lfuse:
  assumes lastfirst xxs
  shows ifirst (lfuse xxs) = ifirst(ifirst(xxs))
  using assms
  proof
    (cases xxs)

```

```

case (INil x1)
then show ?thesis by simp
next
case (ICons x21 x22)
then show ?thesis
using assms ifirst-lfuse by auto
qed

```

**lemma** *lfuse-ilen* :

**assumes** *lastfirst xs*

**shows**  $ilen (lfuse\ xs) = (\sum k::nat = 0..(ilen\ xs). ilen(inth\ xs\ k))$

**using** *assms*

**proof**

(*induct xs*)

**case** (INil *x*)

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

**next**

**case** (ICons *x1a xs*)

**then show** ?*case* **proof** –

**have** 1 :  $ilen (lfuse (x1a \odot xs)) = ilen (fuse\ x1a\ (lfuse\ xs))$

**by** *simp*

**have** 2 : *lastfirst* (*x1a*  $\odot$  *xs*) **using** *ICons.prem*s **by** *auto*

**have** 3 : *ilast* *x1a* = *ifirst*(*lfuse xs*)

**using** *ICons.prem*s *ifirst-lfuse-ifirst* **by** *fastforce*

**have** 4 :  $ilen (fuse\ x1a\ (lfuse\ xs)) = (ilen\ x1a) + ilen(lfuse\ xs)$

**using** 3 *fuse-ilen* **by** *blast*

**have** 5 :  $(ilen\ x1a) + ilen(lfuse\ xs) =$

$(ilen\ x1a) + (\sum k::nat = 0..(ilen\ xs). ilen(inth\ xs\ k))$

**using** *ICons.hyps* *ICons.prem*s **by** *auto*

**have** 6 :  $(\sum k = 0..ilen\ (x1a \odot xs). ilen ( inth (x1a \odot xs) k)) =$

$(ilen(inth (x1a \odot xs) 0) ) +$

$(\sum k = 1..1+ilen\ (xs). ilen ( inth (x1a \odot xs) k))$

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

**have** 7 :  $(ilen(inth (x1a \odot xs) 0) ) = ilen(x1a)$

**by** *simp*

**have** 8 :  $(\sum k = 1..1+ilen\ (xs). ilen ( inth (x1a \odot xs) k)) =$

$(\sum k = 0..ilen\ (xs). ilen ( inth (x1a \odot xs) (k+1)))$

**using** *sum.shift-bounds-cl-nat-ivl*[of  $\lambda k. ilen ( inth (x1a \odot xs) k) \quad 0\ 1\ ilen(xs)$ ]

**by** *simp*

**have** 9 :  $(\sum k = 0..ilen\ (xs). ilen ( inth (x1a \odot xs) (k+1))) =$

$(\sum k = 0..ilen\ (xs). ilen ( inth (xs) (k)))$

**by** *auto*

**have** 10 :  $(ilen\ x1a) + (\sum k::nat = 0..(ilen\ xs). ilen(inth\ xs\ k)) =$

$(\sum k = 0..ilen\ (x1a \odot xs). ilen ( inth (x1a \odot xs) k))$

**using** 6 7 8 9 **by** *linarith*

**show** ?*thesis*

**by** (*simp add: 10 4 5*)

**qed**

**qed**



```

lemma idx-fuse:
  assumes ilast l1 = ifirst l2
  shows
    (index-sequence (ifirst l1) (fuse l1 l2)) =
      ( index-sequence (ifirst l1) l1  $\wedge$  index-sequence (ifirst l2) l2 )
using assms
proof
  (induct l1 arbitrary: l2)
  case (INil x)
  then show ?case by (simp add: index-sequence-def)
  next
  case (ICons x1a l1)
  then show ?case
    using idx-expand1 ifirst-fuse by force
qed

lemma idx-lfuse-help1:
  assumes ( $\forall k. k < \text{ilen } (\text{lfuse } l) \longrightarrow$ 
     $\text{inth } (\text{fuse } x1a \ (\text{lfuse } l)) \ (\text{ilen } x1a + k) <$ 
     $\text{inth } (\text{fuse } x1a \ (\text{lfuse } l)) \ (\text{ilen } x1a + \text{Suc } k)$ )

     $\text{ilen } x1a \leq n$ 
     $n < \text{ilen } x1a + \text{ilen } (\text{lfuse } l)$ 
  shows  $\text{inth } (\text{fuse } x1a \ (\text{lfuse } l)) \ n < \text{inth } (\text{fuse } x1a \ (\text{lfuse } l)) \ (\text{Suc } n)$ 
  using assms
  by (metis add-Suc-right add-less-imp-less-left le-Suc-ex)

lemma idx-lfuse:
  assumes lastfirst l
  shows (index-sequence (ifirst (lfuse l)) (lfuse l)) =
    ( $\forall i \leq \text{ilen } l. \text{index-sequence } (\text{ifirst } (\text{inth } l \ i)) \ (\text{inth } l \ i)$ )
  using assms
proof
  (induct l)
  case (INil x)
  then show ?case by (simp add: index-sequence-def)
  next
  case (ICons x1a l)
  then show ?case
    proof –
      have 0: lastfirst l
      using ICons.prem lastfirst.simps(2) by blast
      have 1:  $\text{index-sequence } (\text{ifirst } (\text{lfuse } (x1a \odot l))) \ (\text{lfuse } (x1a \odot l)) =$ 
        ( $\text{inth } (\text{fuse } x1a \ (\text{lfuse } l)) \ 0 = \text{ifirst } (\text{fuse } x1a \ (\text{lfuse } l)) \ \wedge$ 
        ( $\forall n < \text{ilen } (\text{fuse } x1a \ (\text{lfuse } l)).$ 
         $\text{inth } (\text{fuse } x1a \ (\text{lfuse } l)) \ n < \text{inth } (\text{fuse } x1a \ (\text{lfuse } l)) \ (\text{Suc } n)$ ))
      by (simp add: index-sequence-def)
      have 2: ( $\text{inth } (\text{fuse } x1a \ (\text{lfuse } l)) \ 0 = \text{ifirst } (\text{fuse } x1a \ (\text{lfuse } l))$ )
      by simp
      have 3:  $\text{ilen } (\text{fuse } x1a \ (\text{lfuse } l)) = \text{ilen } x1a + \text{ilen } (\text{lfuse } l)$ 

```

```

    by (simp add: fuse-ilen-a)
  have 4: (∀ n<ilen (fuse x1a (lfuse l)).
    inth (fuse x1a (lfuse l)) n < inth (fuse x1a (lfuse l)) (Suc n)) =
    (∀ n<ilen x1a + ilen (lfuse l).
    inth (fuse x1a (lfuse l)) n < inth (fuse x1a (lfuse l)) (Suc n))
    by (simp add: 3)
  have 5: (∀ n<ilen x1a + ilen (lfuse l).
    inth (fuse x1a (lfuse l)) n < inth (fuse x1a (lfuse l)) (Suc n)) =
    ((∀ n<ilen x1a. inth (fuse x1a (lfuse l)) n < inth (fuse x1a (lfuse l)) (Suc n)) ∧
    (∀ n. 0 ≤ n-ilen x1a ∧ n - ilen x1a<ilen (lfuse l) →
    inth (fuse x1a (lfuse l)) n < inth (fuse x1a (lfuse l)) (Suc n)))
    by auto
    (metis add.commute less-diff-conv2 not-less)
  have 6: (∀ n<ilen x1a. inth (fuse x1a (lfuse l)) n < inth (fuse x1a (lfuse l)) (Suc n)) =
    index-sequence (ifirst x1a) x1a
    proof (simp add: index-sequence-def)
      have ilast x1a = ifirst (lfuse l)
      using ICons.premis ifirst-lfuse-ifirst by force
      then show (∀ n<ilen x1a. inth (fuse x1a (lfuse l)) n < inth (fuse x1a (lfuse l)) (Suc n)) = (∀ n<ilen
        x1a. inth x1a n < inth x1a (Suc n))
        by (metis (no-types) Suc-leI inth-prefix less-or-eq-imp-le prefix-fuse)
      qed
  have 7: (∀ n. ilen x1a ≤ n ∧ n < ilen x1a + ilen (lfuse l) →
    inth (fuse x1a (lfuse l)) n < inth (fuse x1a (lfuse l)) (Suc n)) =
    (∀ k. k<ilen (lfuse l) →
    inth (fuse x1a (lfuse l)) (ilen x1a + k) <
    inth (fuse x1a (lfuse l)) (ilen x1a + Suc k))
    using idx-lfuse-help1 by auto
  have 8: (∀ k. k<ilen (lfuse l) →
    inth (fuse x1a (lfuse l)) (ilen x1a + k) <
    inth (fuse x1a (lfuse l)) (ilen x1a + Suc k)) =
    (∀ k. k<ilen (lfuse l) →
    inth ( (lfuse l)) (k) <
    inth ( (lfuse l)) (Suc k))    (is ?L=?R)
    proof
      show ?L ⇒ ?R
      by (metis ICons.premis Suc-leI fuse-inth-a ifirst-lfuse-ifirst lastfirst.simps(2)
        le-simps(1))
      show ?R ⇒ ?L
      by (metis ICons.premis Suc-leI add-Suc-right fuse-inth-a
        ifirst-lfuse-ifirst lastfirst.simps(2) less-imp-le-nat)
    qed
  have 9: (∀ k. k<ilen (lfuse l) →
    inth ( (lfuse l)) (k) <
    inth ( (lfuse l)) (Suc k)) = index-sequence (ifirst (lfuse l)) (lfuse l)
    by (simp add: index-sequence-def)
  have 91: index-sequence (ifirst (lfuse (x1a ⊙ l))) (lfuse (x1a ⊙ l)) =
    (index-sequence (ifirst x1a) x1a ∧ index-sequence (ifirst (lfuse l)) (lfuse l))
    using 1 2 4 5 6 7 8 9
    by (metis ICons.premis idx-fuse ifirst-lfuse

```

```

      ifirst-lfuse-ifirst lastfirst.simps(2) lfuse-ICons)
have 10: index-sequence (ifirst (lfuse l)) (lfuse l) =
  (∀ i ≤ ilen l. index-sequence (ifirst (inth l i)) (inth l i))
  using 0 ICons.hyps by blast
have 11: (∀ i ≤ ilen(x1a⊙l). index-sequence (ifirst (inth (x1a⊙l) i)) (inth (x1a⊙l) i)) =
  (∀ i ≤ 1 +ilen l. index-sequence (ifirst (inth (x1a⊙l) i)) (inth (x1a⊙l) i))
  by auto
have 12: (∀ i ≤ 1 +ilen l. index-sequence (ifirst (inth (x1a⊙l) i)) (inth (x1a⊙l) i)) =
  ( index-sequence (ifirst (x1a)) (x1a) ∧
    (∀ i. 1 ≤ i ∧ i ≤ 1 +ilen l →
      index-sequence (ifirst (inth (x1a ⊙ l) i)) (inth (x1a ⊙ l) i)))
  by (metis One-nat-def Suc-leI ilen-gr-zero inth-zero
    prefix-ilen-good ilen.simps(2) order.strict-iff-order)
have 13: (∀ i. 1 ≤ i ∧ i ≤ 1 +ilen l →
  index-sequence (ifirst (inth (x1a ⊙ l) i)) (inth (x1a ⊙ l) i)) =
  (∀ j. j ≤ ilen l →
    index-sequence (ifirst (inth (x1a ⊙ l) (1+j))) (inth (x1a ⊙ l) (1+j)))
  by auto
  (simp add: Nitpick.case-nat-unfold)
have 14: (∀ j. j ≤ ilen l →
  index-sequence (ifirst (inth (x1a ⊙ l) (1+j))) (inth (x1a ⊙ l) (1+j))) =
  (∀ j. j ≤ ilen l →
    index-sequence (ifirst (inth (l) (j))) (inth (l) (j)))
  by simp
show ?thesis
using 10 12 13 91 by auto
qed
qed

```

**lemma** *lfuse-ilen-a*:

**assumes** *lastfirst xs*

**shows** ( ∀ i. i ≤ ilen (xs) →  
 (∀ j ≤ ilen (inth (xs) (i)) . j ≤ ilen(lfuse xs)) )

**using** *assms*

**proof** (*induct xs*)

**case** (*INil x*)

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

**next**

**case** (*ICons x1a xs*)

**then show** ?*case*

**proof** –

**have** 0: *ilast x1a* = *ifirst(ifirst xs)*

**using** *ICons.prem*s *lastfirst.simps(2)* **by** *blast*

**have** 1: (∀ i. i ≤ ilen (x1a ⊙ xs) →

(∀ j ≤ ilen (inth (x1a ⊙ xs) i). j ≤ ilen (lfuse (x1a ⊙ xs))))

=

( (∀ j ≤ ilen ( inth (x1a ⊙ xs) 0). j ≤ ilen (lfuse (x1a ⊙ xs))) ∧

(∀ i. 1 ≤ i ∧ i-1 ≤ ilen (xs) →

$(\forall j \leq \text{ilen} (\text{inth} (x1a \odot xs) i). j \leq \text{ilen} (\text{lfuse} (x1a \odot xs)))$  )  
**by** *auto*  
 $(\text{metis One-nat-def add.commute le-diff-conv le-zero-eq not-less-eq-eq old.nat.simps}(4)$   
 $\text{plus-1-eq-Suc})$   
**have** 2:  $(\forall j \leq \text{ilen} (\text{inth} (x1a \odot xs) 0). j \leq \text{ilen} (\text{lfuse} (x1a \odot xs))) =$   
 $(\forall j \leq \text{ilen} (x1a). j \leq \text{ilen} (\text{fuse } x1a (\text{lfuse } xs)))$   
**by** *simp*  
**have** 3:  $(\forall j \leq \text{ilen} (x1a). j \leq \text{ilen} (\text{fuse } x1a (\text{lfuse } xs))) =$   
 $(\forall j \leq \text{ilen} (x1a). j \leq \text{ilen} (x1a) + \text{ilen} (\text{lfuse } xs))$   
**by**  $(\text{simp add: fuse-ilen-a})$   
**have** 4:  $(\forall j \leq \text{ilen} (x1a). j \leq \text{ilen} (x1a) + \text{ilen} (\text{lfuse } xs))$   
**by** *linarith*  
**have** 5:  $(\forall i. 1 \leq i \wedge i-1 \leq \text{ilen} (xs) \longrightarrow$   
 $(\forall j \leq \text{ilen} (\text{inth} (x1a \odot xs) i). j \leq \text{ilen} (\text{lfuse} (x1a \odot xs))))$   
 $=$   
 $(\forall i. 0 \leq i \wedge i \leq \text{ilen} (xs) \longrightarrow$   
 $(\forall j \leq \text{ilen} (\text{inth} (x1a \odot xs) (\text{Suc } i)). j \leq \text{ilen} (\text{lfuse} (x1a \odot xs))))$   
**by**  $(\text{metis add-diff-cancel-left' ilen-gr-zero prefix-ilen-good le-add1}$   
 $\text{ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc})$   
**have** 6:  $(\forall i. 0 \leq i \wedge i \leq \text{ilen} (xs) \longrightarrow$   
 $(\forall j \leq \text{ilen} (\text{inth} (x1a \odot xs) (\text{Suc } i)). j \leq \text{ilen} (\text{lfuse} (x1a \odot xs)))) =$   
 $(\forall i. 0 \leq i \wedge i \leq \text{ilen} (xs) \longrightarrow$   
 $(\forall j \leq \text{ilen} (\text{inth} (xs) (i)). j \leq \text{ilen } x1a + \text{ilen} (\text{lfuse} (xs))))$   
**by**  $(\text{simp add: fuse-ilen-a})$   
**have** 7:  $\forall i. 0 \leq i \wedge i \leq \text{ilen } xs \longrightarrow (\forall j \leq \text{ilen} (\text{inth } xs i). j \leq \text{ilen} (\text{lfuse } xs))$   
**using** *ICons.hyps ICons.premis lastfirst.simps*(2) **by** *blast*  
**have** 8:  $(\forall i. 0 \leq i \wedge i \leq \text{ilen} (xs) \longrightarrow$   
 $(\forall j \leq \text{ilen} (\text{inth} (xs) (i)). j \leq \text{ilen } x1a + \text{ilen} (\text{lfuse} (xs))))$   
**by**  $(\text{simp add: 7 trans-le-add2})$   
**show** *?thesis* **using** 1 3 5 6 8 **by** *auto*  
**qed**  
**qed**

**lemma** *lfuse-split*:

**assumes** *lastfirst xs*  $\wedge (\forall j \leq \text{ilen} (xs). \text{ilen}(\text{inth} (xs) j) > 0)$

**shows**  $(\forall i \leq \text{ilen}(xs).$

$$\begin{aligned}
& (\forall ia < \text{ilen} ((\text{inth} (xs) i)). \\
& \quad f (\text{sub} (\text{inth} (\text{inth} (xs) i) ia) \\
& \quad \quad (\text{inth} (\text{inth} (xs) i) (\text{Suc } ia)) \\
& \quad \quad \sigma))) = \\
& (\forall j < \text{ilen} (\text{lfuse } xs)). \\
& \quad f (\text{sub} (\text{inth} (\text{lfuse } xs)) j) \\
& \quad \quad (\text{inth} (\text{lfuse } xs) (\text{Suc } j)) \\
& \quad \quad \sigma))
\end{aligned}$$

**using** *assms*

**proof**  $(\text{induct } xs)$

**case**  $(\text{INil } x)$

**then show** *?case* **by** *auto*

**next**

```

case (ICons x1a xxs)
then show ?case
proof -
  have 1: (∀ i ≤ ilen (x1a ⊙ xxs).
    ∀ ia < ilen (inth (x1a ⊙ xxs) i).
    f (sub (inth (inth (x1a ⊙ xxs) i) ia) (inth (inth (x1a ⊙ xxs) i) (Suc ia)) σ)) =
    (∀ i ≤ ilen (xxs)+1.
    ∀ ia < ilen (inth (x1a ⊙ xxs) i).
    f (sub (inth (inth (x1a ⊙ xxs) i) ia) (inth (inth (x1a ⊙ xxs) i) (Suc ia)) σ))
  by simp
  have 2: ... =
    ( (∀ ia < ilen (inth (x1a ⊙ xxs) 0).
      f (sub (inth (inth (x1a ⊙ xxs) 0) ia) (inth (inth (x1a ⊙ xxs) 0) (Suc ia)) σ))
      ∧
      (∀ i. 1 ≤ i ∧ i ≤ ilen (xxs)+1 →
        (∀ ia < ilen (inth (x1a ⊙ xxs) i).
          f (sub (inth (inth (x1a ⊙ xxs) i) ia) (inth (inth (x1a ⊙ xxs) i) (Suc ia)) σ))))
  by (metis One-nat-def Suc-leI add-nonneg-nonneg gr-zeroI ilen-gr-zero zero-le-one)
  have 3: (∀ ia < ilen (inth (x1a ⊙ xxs) 0).
    f (sub (inth (inth (x1a ⊙ xxs) 0) ia) (inth (inth (x1a ⊙ xxs) 0) (Suc ia)) σ)) =
    (∀ ia < ilen x1a.
      f (sub (inth (x1a) ia) (inth (x1a) (Suc ia)) σ))
  by simp
  have 4: (∀ i. 1 ≤ i ∧ i ≤ ilen (xxs)+1 →
    (∀ ia < ilen (inth (x1a ⊙ xxs) i).
      f (sub (inth (inth (x1a ⊙ xxs) i) ia) (inth (inth (x1a ⊙ xxs) i) (Suc ia)) σ))) =
    (∀ i. 0 ≤ i-1 ∧ i-1 ≤ ilen (xxs) →
      (∀ ia < ilen (inth (x1a ⊙ xxs) ((i-1)+1)).
        f (sub (inth (inth (x1a ⊙ xxs) ((i-1)+1)) ia)
          (inth (inth (x1a ⊙ xxs) ((i-1)+1)) (Suc ia)) σ)))
  by (auto simp add: Nitpick.case-nat-unfold)
  have 5: ... =
    (∀ i. 0 ≤ i ∧ i ≤ ilen (xxs) →
      (∀ ia < ilen (inth (x1a ⊙ xxs) ((i)+1)).
        f (sub (inth (inth (x1a ⊙ xxs) ((i)+1)) ia)
          (inth (inth (x1a ⊙ xxs) ((i)+1)) (Suc ia)) σ)))
  using 4 by auto
  have 6: ... =
    (∀ i. 0 ≤ i ∧ i ≤ ilen (xxs) →
      (∀ ia < ilen (inth (xxs) ((i))).
        f (sub (inth (inth (xxs) ((i))) ia) (inth (inth (xxs) ((i))) (Suc ia)) σ)))
  by simp
  have 7: lastfirst xxs
    using ICons.prem lastfirst.simps(2) by blast
  have 8: ilast x1a = ifirst(ifirst xxs)
    using ICons.prem lastfirst.simps(2) by blast
  have 9: (∀ j ≤ ilen (x1a ⊙ xxs). ilen (inth (x1a ⊙ xxs) j) > 0)
    using ICons.prem by blast
  have 10: ilen (inth (x1a ⊙ xxs) 0) > 0

```

using *ICons.prem*s by *blast*  
 have 11:  $(\forall j. 1 \leq j \wedge j \leq \text{ilen } (xs)+1 \longrightarrow \text{ilen}(\text{inth } (x1a \odot xs) j) > 0)$   
 using *ICons.prem*s by *auto*  
 have 12:  $(\forall j. 0 \leq j-1 \wedge j-1 \leq \text{ilen } (xs) \longrightarrow \text{ilen}(\text{inth } (x1a \odot xs) ((j-1)+1)) > 0)$   
 using *ICons.prem*s by *auto*  
 have 13:  $(\forall j. j \leq \text{ilen } (xs) \longrightarrow \text{ilen}(\text{inth } (x1a \odot xs) ((j)+1)) > 0)$   
 using *ICons.prem*s by *auto*  
 have 14:  $(\forall j. j \leq \text{ilen } (xs) \longrightarrow \text{ilen}(\text{inth } (xs) ((j))) > 0)$   
 using 13 by *simp*  
 have 15:  $(\forall i. i \leq \text{ilen } (xs) \longrightarrow$   
 $(\forall ia < \text{ilen } (\text{inth } (xs) ((i))).$   
 $f(\text{sub}(\text{inth}(\text{inth } (xs) ((i))) ia) (\text{inth}(\text{inth } (xs) ((i))) (\text{Suc } ia)) \sigma)) =$   
 $(\forall j < \text{ilen } (\text{lfuse } (xs)).$   
 $f(\text{sub}(\text{inth } (\text{lfuse } (xs)) j)$   
 $(\text{inth } (\text{lfuse } (xs)) (\text{Suc } j))$   
 $\sigma))$   
 by (*simp add: 14 7 ICons.hyps*)  
 have 16:  $(\forall j < \text{ilen } (\text{lfuse } (x1a \odot xs)).$   
 $f(\text{sub}(\text{inth } (\text{lfuse } (x1a \odot xs)) j) (\text{inth } (\text{lfuse } (x1a \odot xs)) (\text{Suc } j)) \sigma) =$   
 $(\forall j < \text{ilen } (\text{fuse } x1a (\text{lfuse } (xs))).$   
 $f(\text{sub}(\text{inth } (\text{lfuse } (x1a \odot xs)) j) (\text{inth } (\text{lfuse } (x1a \odot xs)) (\text{Suc } j)) \sigma))$   
 by *simp*  
 have 17:  $\dots =$   
 $(\forall j < \text{ilen } x1a + \text{ilen } ((\text{lfuse } (xs))).$   
 $f(\text{sub}(\text{inth } (\text{lfuse } (x1a \odot xs)) j) (\text{inth } (\text{lfuse } (x1a \odot xs)) (\text{Suc } j)) \sigma))$   
 by (*simp add: fuse-ilen-a*)  
 have 18:  $\dots =$   
 $((\forall j < \text{ilen } x1a.$   
 $f(\text{sub}(\text{inth } (\text{lfuse } (x1a \odot xs)) j) (\text{inth } (\text{lfuse } (x1a \odot xs)) (\text{Suc } j)) \sigma)) \wedge$   
 $(\forall j. \text{ilen } x1a \leq j \wedge j < \text{ilen } x1a + \text{ilen } ((\text{lfuse } (xs))) \longrightarrow$   
 $f(\text{sub}(\text{inth } (\text{lfuse } (x1a \odot xs)) j) (\text{inth } (\text{lfuse } (x1a \odot xs)) (\text{Suc } j)) \sigma)))$   
 using *le-add1 less-le-trans not-less* by *blast*  
 have 19:  $(\forall j < \text{ilen } x1a.$   
 $f(\text{sub}(\text{inth } (\text{lfuse } (x1a \odot xs)) j) (\text{inth } (\text{lfuse } (x1a \odot xs)) (\text{Suc } j)) \sigma) =$   
 $(\forall j < \text{ilen } x1a.$   
 $f(\text{sub}(\text{inth } (\text{fuse } x1a (\text{lfuse } (xs))) j) (\text{inth } (\text{fuse } x1a (\text{lfuse } (xs))) (\text{Suc } j)) \sigma))$   
 by *simp*  
 have 20:  $\dots =$   
 $(\forall j < \text{ilen } x1a.$   
 $f(\text{sub}(\text{inth } (x1a) j) (\text{inth } (x1a) (\text{Suc } j)) \sigma))$   
 proof –  
 have *prefix*  $(\text{ilen } x1a) (\text{fuse } x1a (\text{lfuse } xs)) = x1a$   
 by (*metis (no-types) ICons.prem*s *ifirst-lfuse-ifirst lastfirst.simps(2) prefix-fuse*)  
 then show *?thesis*  
 by (*metis (no-types) Suc-leI inth-prefix le-simps(1)*)  
 qed  
 have 21:  $(\forall j. \text{ilen } x1a \leq j \wedge j < \text{ilen } x1a + \text{ilen } ((\text{lfuse } (xs))) \longrightarrow$   
 $f(\text{sub}(\text{inth } (\text{lfuse } (x1a \odot xs)) j) (\text{inth } (\text{lfuse } (x1a \odot xs)) (\text{Suc } j)) \sigma) =$   
 $(\forall j. \text{ilen } x1a \leq j \wedge j < \text{ilen } x1a + \text{ilen } ((\text{lfuse } (xs))) \longrightarrow$   
 $f(\text{sub}(\text{inth } (\text{fuse } x1a (\text{lfuse } (xs))) j) (\text{inth } (\text{fuse } x1a (\text{lfuse } (xs))) (\text{Suc } j)) \sigma))$

```

    by simp
  have 22: ... =
    (∀ j. ilen x1a ≤ j ∧ j < ilen x1a + ilen ((lfuse (xs))) →
      f (sub (inth ((lfuse (xs))) (j - ilen x1a))
        (inth (fuse x1a (lfuse (xs))) ((j)+1)) σ)) (is ?L=?R)
  proof rule
    show ?L⇒?R
      by auto
      (metis 8 ICons.prem fuse-ilen-a fuse-inth
        ifirst-lfuse-ifirst less-imp-le-nat)
    show ?R ⇒ ?L
      by auto
      (metis 8 ICons.prem fuse-ilen-a fuse-inth ifirst-lfuse-ifirst less-imp-le-nat)
  qed
  have 23: ... =
    (∀ j. ilen x1a ≤ j ∧ j < ilen x1a + ilen ((lfuse (xs))) →
      f (sub (inth ((lfuse (xs))) (j - ilen x1a))
        (inth ( (lfuse (xs))) (((Suc j) - ilen x1a))) σ)) (is ?L=?R)
  proof
    show ?L⇒?R
      by auto
      (metis 8 ICons.prem Suc-leI fuse-ilen-a fuse-inth ifirst-lfuse-ifirst le-SucI)
    show ?R ⇒ ?L
      by auto
      (metis 8 ICons.prem Suc-leI fuse-ilen-a fuse-inth ifirst-lfuse-ifirst le-SucI)
  qed
  have 24: ... =
    (∀ j. 0 ≤ j ∧ j < ilen ((lfuse (xs))) →
      f (sub (inth ((lfuse (xs))) (j) ) (inth ( (lfuse (xs))) (((Suc j) ))) σ))
    by (rule interval-shift-index-to-zero-b)
  show ?thesis
  using 15 17 18 2 20 22 23 24 4 5 by auto
  qed
  qed
end

```

## 4 Finite ITL: Axioms and Rules

```

theory ITL
imports
  Semantics
begin

```

The Finite ITL axiom and proof rules are introduced (taken from [5]). The soundness of the rules and axioms are checked using the lemmas of Semantics.thy.

## 4.1 Rules

**lemma** *MP* :  
  **assumes**  $\vdash f \longrightarrow g$   
     $\vdash f$   
  **shows**  $\vdash g$   
**using** *assms(1) assms(2) by fastforce*

**lemma** *BoxGen* :  
  **assumes**  $\vdash f$   
  **shows**  $\vdash \Box f$   
**using** *assms by (auto simp: itl-defs)*

**lemma** *BiGen*:  
  **assumes**  $\vdash f$   
  **shows**  $\vdash bi\ f$   
**using** *assms by (auto simp: itl-defs)*

## 4.2 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 :
   $\vdash \Box (f \longrightarrow \text{wnext } f) \wedge f \longrightarrow \Box f$ 
using BoxInductSem Valid-def by blast

lemma ChopstarEqv :
   $\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$ 
using ChopstarEqvSem Valid-def by blast

```

### 4.3 Additional Lemmas

The following is again from [3, 2] but adapted for our need.

```

lemma int-eq-true:
assumes  $\vdash P$ 
shows  $\vdash P = \# \text{True}$ 
using assms by auto

lemma int-eq:
assumes  $\vdash X = Y$ 
shows  $X = Y$ 
using assms by (auto simp: inteq-reflection)

lemma int-iffI:
assumes  $\vdash F \longrightarrow G$ 
            $\vdash G \longrightarrow F$ 
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  $\vdash A \longrightarrow B$ 
            $\vdash B \longrightarrow C$ 
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*

## 4.4 Quantification

**lemma** *EExI* :  
 $\vdash F\ y \longrightarrow (\exists\exists\ x.\ F\ x)$   
 by (*auto simp add: exist-state-d-def Valid-def*)

**lemma** *EExE*:  
 assumes  $\bigwedge x. \vdash F\ x \longrightarrow G$   
 shows  $\vdash (\exists\exists\ x.\ F\ x) \longrightarrow G$   
 using *assms* by (*metis (mono-tags, lifting) Valid-def exist-state-d-def unl-lift2*)

**lemma** *EExVal*:  
 $(w \models (\exists\exists\ x.\ F\ x)) =$   
 $(\exists\ x\ (val :: 'a\ interval). (\ (val = (imap\ x\ w) \wedge (w \models F\ x))))$   
 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*

## 4.5 Lemmas about *current-val*

**lemma** *current-const*:  $\vdash \$(\#c) = \#c$   
 by (*auto simp: itl-def*)

**lemma** *current-fun1*:  $\vdash \$ (f\<x>) = f\<\$x>$   
 by (*auto simp: itl-def*)

**lemma** *current-fun2*:  $\vdash \$ (f\<x,y>) = f\<\$x,\$y>$   
 by (*auto simp: itl-def*)

**lemma** *current-fun3*:  $\vdash \$ (f\<x,y,z>) = f\<\$x,\$y,\$z>$   
 by (*auto simp: itl-def*)

**lemma** *current-forall*:  $\vdash \$ (\forall\ x.\ P\ x) = (\forall\ x.\ \$ (P\ x))$   
 by (*auto simp: itl-def*)

**lemma** *current-exists*:  $\vdash \$ (\exists\ x.\ P\ x) = (\exists\ x.\ \$ (P\ x))$   
 by (*auto simp: itl-def*)

**lemma** *current-exists1*:  $\vdash \$ (\exists!\ x.\ P\ x) = (\exists!\ x.\ \$ (P\ x))$   
 by (*auto simp: itl-def*)

**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*]

## 4.6 Lemmas about *next-val*

**lemma** *next-const*:  $\vdash \text{more} \longrightarrow (\#c)\$ = \#c$   
**by** (*auto simp: itl-defs*)

**lemma** *next-fun1*:  $\vdash \text{more} \longrightarrow f\langle x \rangle\$ = f\langle x\$ \rangle$   
**by** (*auto simp: itl-defs*)

**lemma** *next-fun2*:  $\vdash \text{more} \longrightarrow f\langle x, y \rangle\$ = f\langle x\$, y\$ \rangle$   
**by** (*auto simp: itl-defs*)

**lemma** *next-fun3*:  $\vdash \text{more} \longrightarrow f\langle x, y, z \rangle\$ = f\langle x\$, y\$, z\$ \rangle$   
**by** (*auto simp: itl-defs*)

**lemma** *next-forall*:  $\vdash \text{more} \longrightarrow (\forall x. P\ x)\$ = (\forall x. (P\ x)\$)$   
**by** (*auto simp: itl-defs*)

**lemma** *next-exists*:  $\vdash \text{more} \longrightarrow (\exists x. P\ x)\$ = (\exists x. (P\ x)\$)$   
**by** (*auto simp: itl-defs*)

**lemma** *next-exists1*:  $\vdash \text{more} \longrightarrow (\exists! x. P\ x)\$ = (\exists! x. (P\ x)\$)$   
**by** (*auto simp: itl-defs*)

**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*]

## 4.7 Lemmas about *fin-val*

**lemma** *fin-const*:  $\vdash !(\#c) = \#c$   
**by** (*auto simp: itl-def*)

**lemma** *fin-fun1*:  $\vdash !(f\langle x \rangle) = f\langle !x \rangle$   
**by** (*auto simp: itl-def*)

**lemma** *fin-fun2*:  $\vdash !(f\langle x, y \rangle) = f\langle !x, !y \rangle$   
**by** (*auto simp: itl-def*)

**lemma** *fin-fun3*:  $\vdash !(f\langle x, y, z \rangle) = f\langle !x, !y, !z \rangle$   
**by** (*auto simp: itl-def*)

**lemma** *fin-forall*:  $\vdash !(\forall x. P\ x) = (\forall x. !(P\ x))$

by (auto simp: itl-def)

**lemma** *fin-exists*:  $\vdash !(\exists x. P x) = (\exists x. !(P x))$   
 by (auto simp: itl-def)

**lemma** *fin-exists1*:  $\vdash !(\exists! x. P x) = (\exists! x. !(P x))$   
 by (auto simp: itl-def)

**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]*  
**lemmas** *all-fin-eq* = *all-fin[THEN inteq-reflection]*

## 4.8 Lemmas about *penult-val*

**lemma** *penult-const*:  $\vdash \text{more} \longrightarrow (\#c)! = \#c$   
 by (auto simp: itl-defs)

**lemma** *penult-fun1*:  $\vdash \text{more} \longrightarrow f\langle x \rangle! = f\langle x! \rangle$   
 by (auto simp: itl-defs)

**lemma** *penult-fun2*:  $\vdash \text{more} \longrightarrow f\langle x, y \rangle! = f\langle x!, y! \rangle$   
 by (auto simp: itl-defs)

**lemma** *penult-fun3*:  $\vdash \text{more} \longrightarrow f\langle x, y, z \rangle! = f\langle x!, y!, z! \rangle$   
 by (auto simp: itl-defs)

**lemma** *penult-forall*:  $\vdash \text{more} \longrightarrow (\forall x. P x)! = (\forall x. (P x)!)$   
 by (auto simp: itl-def)

**lemma** *penult-exists*:  $\vdash \text{more} \longrightarrow (\exists x. P x)! = (\exists x. (P x)!)$   
 by (auto simp: itl-def)

**lemma** *penult-exists1*:  $\vdash \text{more} \longrightarrow (\exists! x. P x)! = (\exists! x. (P x)!)$   
 by (auto simp: itl-defs)

**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]*

## 4.9 Basic temporal variables properties

**lemma** *empty-imp-fin-equiv-curr*:  
 $\vdash \text{empty} \longrightarrow !v = \$v$   
 by (simp add: Valid-def itl-defs)

**lemma** *skip-imp-fin-equiv-next*:  
 $\vdash \text{skip} \longrightarrow !v = v\$$

by (simp add: Valid-def itl-defs)

**lemma** *skip-imp-penult-equiv-curr*:

$\vdash \text{skip} \longrightarrow v! = \$v$

by (simp add: Valid-def itl-defs)

end

## 5 Finite ITL theorems

**theory** *Theorems*

**imports**

*ITL*

**begin**

We give the proofs of a list of Finite ITL theorems. These proofs and theorems were from [8].

### 5.1 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-defs 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*(1) *assms*(2) **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*(1) *assms*(2) **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*(1) *assms*(2) **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*(1) *assms*(2) **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*)

## 5.2 State formulas

The *init* operator denotes state formulas, i.e., ITL formula that only constrain the first state of an interval.

**lemma** *Initprop* :

$\vdash ((\text{init } f) \wedge (\text{init } g)) = \text{init}(f \wedge g)$   
 $\vdash (\neg (\text{init } f)) = \text{init } (\neg f)$   
 $\vdash ((\text{init } f) \vee (\text{init } g)) = \text{init } (f \vee g)$   
 $\vdash \text{init } \# \text{True}$

**by** (*auto simp: itl-defs*)

**lemma** *Finprop* :

$\vdash ((\#True;(f \wedge empty)) \wedge (\#True;(g \wedge empty))) = (\#True;((f \wedge g) \wedge empty))$

$\vdash ((\#True;(f \wedge empty)) \vee (\#True;(g \wedge empty))) = (\#True;((f \vee g) \wedge empty))$

$\vdash (\#True;((\#True) \wedge empty))$

$\vdash (\neg (\#True;(f \wedge empty))) = (\#True;(\neg f \wedge empty))$

**by** (*auto simp: finalt-defs*) (*simp add: itl-defs, fastforce*)

### 5.3 Basic Theorems

**lemma** *BiChopImpChop* :

$\vdash bi(f \longrightarrow f1) \longrightarrow f;g \longrightarrow f1;g$

**proof** –

**have** 1:  $\vdash g \longrightarrow g$  **by** *auto*

**hence** 2:  $\vdash \Box (g \longrightarrow g)$  **by** (*rule BoxGen*)

**have** 3:  $\vdash bi(f \longrightarrow f1) \wedge \Box(g \longrightarrow g) \longrightarrow f;g \longrightarrow f1;g$  **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** *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:  $\vdash \text{skip};(\neg (f \longrightarrow g)) = \text{skip};(f \wedge \neg g)$  **by** (rule *RightChopEqvChop*)  
**have** 3:  $\vdash f \longrightarrow g \vee (f \wedge \neg g)$  **by** auto  
**hence** 4:  $\vdash \text{skip};f \longrightarrow (\text{skip};g) \vee (\text{skip};(f \wedge \neg g))$  **by** (rule *ChopOrImpRule*)  
**hence** 5:  $\vdash \neg (\text{skip};(f \wedge \neg g)) \longrightarrow (\text{skip};f) \longrightarrow (\text{skip};g)$  **by** auto  
**have** 6:  $\vdash \neg (\text{skip};(\neg (f \longrightarrow g))) \longrightarrow (\text{skip};f) \longrightarrow (\text{skip};g)$  **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** *ChopImpDiamond*:

$\vdash f;g \longrightarrow \Diamond g$

**proof** –

**have** 1:  $\vdash f \longrightarrow \#True$  **by** auto  
**hence** 2:  $\vdash f;g \longrightarrow \#True;g$  **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:  $\vdash \text{empty} \longrightarrow \#True$  **by** auto  
**hence** 3:  $\vdash \text{empty};f \longrightarrow \#True;f$  **by** (rule *LeftChopImpChop*)  
**have** 4:  $\vdash f \longrightarrow \#True;f$  **using** 1 3 **by** fastforce  
**from** 4 **show** ?thesis **by** (simp add: sometimes-d-def)  
**qed**

**lemma** *BoxElim*:

$\vdash \Box f \longrightarrow f$

**proof** –

**have** 1:  $\vdash \neg f \longrightarrow \Diamond (\neg f)$  **by** (rule *NowImpDiamond*)  
**hence** 2:  $\vdash \neg (\Diamond (\neg f)) \longrightarrow f$  **by** auto  
**from** 2 **show** ?thesis **by** (metis always-d-def)  
**qed**

**lemma** *NextDiamondImpDiamond*:

$\vdash \bigcirc (\Diamond f) \longrightarrow \Diamond f$

**proof** –

**have** 1:  $\vdash \text{skip};(\#True;f) = ((\text{skip};\#True);f)$  **by** (rule *ChopAssoc*)

**hence** 2:  $\vdash (\text{skip}; \# \text{True}); f = \text{skip}; (\# \text{True}; f)$  **by** *auto*  
**hence** 3:  $\vdash (\text{skip}; \# \text{True}); f = \bigcirc(\Diamond f)$  **by** (*simp add: next-d-def sometimes-d-def*)  
**have** 4:  $\vdash (\text{skip}; \# \text{True}); f \longrightarrow \Diamond f$  **by** (*rule ChopImpDiamond*)  
**from** 3 4 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *BoxImpNowAndWeakNext*:

$\vdash \Box f \longrightarrow (f \wedge \text{wnext} (\Box f))$

**proof** –

**have** 1:  $\vdash \neg f \longrightarrow \Diamond (\neg f)$  **by** (*rule NowImpDiamond*)  
**hence** 2:  $\vdash \neg (\Diamond (\neg f)) \longrightarrow f$  **by** *auto*  
**hence** 3:  $\vdash \Box f \longrightarrow f$  **by** (*metis always-d-def*)  
**have** 4:  $\vdash \bigcirc (\Diamond (\neg f)) \longrightarrow \Diamond (\neg f)$  **by** (*rule NextDiamondImpDiamond*)  
**have** 5:  $\vdash \neg \neg (\Diamond (\neg f)) \longrightarrow \Diamond (\neg f)$  **by** *auto*  
**hence** 6:  $\vdash \bigcirc (\neg \neg (\Diamond (\neg f))) \longrightarrow \bigcirc (\Diamond (\neg f))$  **by** (*rule NextImpNext*)  
**have** 7:  $\vdash \bigcirc (\neg \neg (\Diamond (\neg f))) \longrightarrow \Diamond (\neg f)$  **using** 4 6 **by** *auto*  
**hence** 8:  $\vdash \bigcirc (\neg (\Box f)) \longrightarrow \Diamond (\neg f)$  **by** (*simp add: always-d-def*)  
**hence** 9:  $\vdash \neg (\Diamond (\neg f)) \longrightarrow \neg (\bigcirc (\neg (\Box f)))$  **by** *auto*  
**hence** 10:  $\vdash \Box f \longrightarrow \text{wnext} (\Box f)$  **by** (*simp add: always-d-def wnext-d-def*)  
**from** 3 10 **show** ?thesis **by** *fastforce*

**qed**

**lemma** *BoxImpBoxRule*:

**assumes**  $\vdash f \longrightarrow g$

**shows**  $\vdash \Box f \longrightarrow \Box g$

**proof** –

**have** 1:  $\vdash f \longrightarrow g$  **using** *assms* **by** *auto*  
**hence** 2:  $\vdash \neg g \longrightarrow \neg f$  **by** *auto*  
**hence** 3:  $\vdash \Box (\neg g \longrightarrow \neg f)$  **by** (*rule BoxGen*)  
**have** 4:  $\vdash \Box (\neg g \longrightarrow \neg f) \longrightarrow (\# \text{True}; (\neg g)) \longrightarrow (\# \text{True}; (\neg f))$  **by** (*rule BoxChopImpChop*)  
**have** 5:  $\vdash (\# \text{True}; (\neg g)) \longrightarrow (\# \text{True}; (\neg f))$  **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:  $\vdash \Box ((\neg g) \longrightarrow \neg f) \longrightarrow (\# \text{True}; (\neg g)) \longrightarrow (\# \text{True}; (\neg f))$  **by** (*rule BoxChopImpChop*)  
**have** 4:  $\vdash \Box (f \longrightarrow g) \longrightarrow (\# \text{True}; (\neg g)) \longrightarrow (\# \text{True}; (\neg f))$   
**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** *DiamondEmpty*:

```

  ⊢ ◇ empty
proof -
  have 1: ⊢ #True by auto
  have 2: ⊢ #True; empty = #True by (rule ChopEmpty)
  have 3: ⊢ #True; empty using 1 2 by auto
  from 3 show ?thesis by (simp add: sometimes-d-def)
qed

```

```

lemma NextEqvNext:
  assumes ⊢ f = g
  shows ⊢ ○ f = ○ g
proof -
  have 1: ⊢ f = g using assms by auto
  hence 2: ⊢ skip;f = skip;g by (rule RightChopEqvChop)
  from 1 show ?thesis by (metis 2 next-d-def)
qed

```

```

lemma NextAndNextImpNextRule:
  assumes ⊢ (f ∧ g) ⟶ h
  shows ⊢ (○ f ∧ ○ g) ⟶ ○ h
using assms by (auto simp: itl-defs)

```

```

lemma NextAndNextEqvNextRule:
  assumes ⊢ (f ∧ g) = h
  shows ⊢ (○ f ∧ ○ g) = ○ h
using assms by (metis NextAndNextImpNextRule Prop11 Prop12 int-eq int-simps(20))

```

```

lemma WeakNextEqvWeakNext:
  assumes ⊢ f = g
  shows ⊢ wnext f = wnext g
using assms using inteq-reflection by force

```

```

lemma DiamondImpDiamond:
  assumes ⊢ f ⟶ g
  shows ⊢ ◇ f ⟶ ◇ g
using assms by (simp add: RightChopImpChop sometimes-d-def)

```

```

lemma DiamondEqvDiamond:
  assumes ⊢ f = g
  shows ⊢ ◇ f = ◇ g
using assms using int-eq by force

```

```

lemma BoxEqvBox:
  assumes ⊢ f = g
  shows ⊢ □ f = □ g
using assms using inteq-reflection by force

```

```

lemma BoxAndBoxImpBoxRule:
  assumes ⊢ f ∧ g ⟶ h
  shows ⊢ □ f ∧ □ g ⟶ □ 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** *BoxIntro*:

**assumes**  $\vdash f \longrightarrow g$

$\vdash \text{more} \wedge f \longrightarrow \Box f$

**shows**  $\vdash f \longrightarrow \Box g$

**proof** –

**have** 1:  $\vdash \text{more} \wedge f \longrightarrow \Box f$  **using** *assms* **by** *auto*

**hence** 2:  $\vdash f \longrightarrow (\text{empty} \vee \Box f)$  **by** (*auto simp: itl-defs*)

**hence** 3:  $\vdash f \longrightarrow \text{wnext } f$  **by** (*auto simp: itl-defs*)

**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**  $\vdash f \longrightarrow \Box f$

**shows**  $\vdash \neg f$

**proof** –

**have** 1:  $\vdash f \longrightarrow \Box f$  **using** *assms* **by** *auto*

**hence** 2:  $\vdash f \longrightarrow (\text{more} \wedge \text{wnext } f)$  **by** (*auto simp: more-defs wnext-defs next-defs*)

**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:  $\vdash \neg(\Box \text{more})$  **by** (*auto simp: itl-defs*)

**from** 7 9 11 12 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *WnextEqvEmptyOrNext*:

$\vdash \text{wnext } f = (\text{empty} \vee \bigcirc f)$

**by** (*auto simp: itl-defs*)

**lemma** *NotEmptyAndNext*:

$\vdash \neg(\text{empty} \wedge \bigcirc f)$

**by** (*auto simp: 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** (*auto simp: itl-defs*)

**lemma** *DiamondIntro*:

**assumes**  $\vdash (f \wedge \neg g) \longrightarrow \bigcirc f$

**shows**  $\vdash f \longrightarrow \Diamond g$

**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: 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: 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:  $\vdash \neg(\Box \text{ more})$   
     **by** (*auto simp: itl-defs*)  
 have 21:  $\vdash \neg(f \wedge (\Box (\neg g)))$   
     **using** 17 19 20 **by** *fastforce*  
 hence 22:  $\vdash \neg f \vee \neg(\Box (\neg g))$   
     **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**  $\vdash f \longrightarrow \Diamond g$

**proof** –

**have** 1:  $\vdash (f \wedge \neg g) \longrightarrow \bigcirc (f \wedge \neg g)$  **using** *assms* **by** *auto*

**hence** 2:  $\vdash \neg(f \wedge \neg g)$  **by** (*rule NextLoop*)

hence 3:  $\vdash f \longrightarrow g$  **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  $\vdash f \longrightarrow g$

**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:  $\vdash \neg\neg(f \longrightarrow g)$  **by** (rule *NextLoop*)  
 from 3 **show** *?thesis* **by** *auto*

**qed**

**lemma** *DiamondDiamondEqvDiamond*:

$\vdash \Diamond(\Diamond f) = \Diamond f$

**proof** –

have 1:  $\vdash \#True; \#True = \#True$  **by** (*auto simp: itl-defs*)  
 hence 2:  $\vdash (\#True; \#True); f = \#True; f$  **using** *LeftChopEqvChop* **by** *blast*  
 have 3:  $\vdash (\#True; \#True); f = \#True; (\#True; f)$  **using** *ChopAssoc* **by** *fastforce*  
 from 2 3 **show** *?thesis* **by** (*metis inteq-reflection sometimes-d-def*)

**qed**

**lemma** *WeakNextDiamondInduct*:

assumes  $\vdash wnext(\Diamond f) \longrightarrow f$   
 shows  $\vdash f$

**proof** –

have 1:  $\vdash wnext(\Diamond f) \longrightarrow f$  **using** *assms* **by** *blast*  
 hence 2:  $\vdash \neg f \longrightarrow \neg(wnext(\Diamond f))$  **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:  $\vdash \neg\neg f$  **by** (rule *NextLoop*)  
 from 7 **show** *?thesis* **by** *auto*

**qed**

**lemma** *EmptyNextInducta*:

assumes  $\vdash empty \longrightarrow f$   
 $\vdash \bigcirc f \longrightarrow f$

shows  $\vdash f$

**proof** –

have 1:  $\vdash empty \longrightarrow f$  **using** *assms* **by** *auto*  
 have 2:  $\vdash \bigcirc f \longrightarrow f$  **using** *assms* **by** *blast*  
 have 3:  $\vdash (empty \vee \bigcirc f) \longrightarrow f$  **using** 1 2 **by** *fastforce*  
 have 4:  $\vdash wnext f = (empty \vee \bigcirc f)$  **by** (rule *WnextEqvEmptyOrNext*)  
 hence 5:  $\vdash wnext f \longrightarrow f$  **using** 3 **by** *fastforce*  
 hence 6:  $\vdash \neg f \longrightarrow \neg(wnext f)$  **by** *auto*  
 hence 7:  $\vdash \neg f \longrightarrow \bigcirc(\neg f)$  **by** (*auto simp: wnext-d-def*)



hence 8:  $\vdash \neg \neg f$  **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**  $\vdash f \longrightarrow g$

**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:  $\vdash \neg \neg (f \longrightarrow g)$  **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**  $\vdash \text{fin } f \wedge \text{fin } g \longrightarrow \text{fin } h$

**proof** –

**have**  $\vdash f \wedge g \longrightarrow h$  **using** assms **by** auto  
**then show** ?thesis **by** (simp add: fin-defs Valid-def)

**qed**

**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*:

$\vdash bi (f \longrightarrow g) \longrightarrow di f \longrightarrow di g$   
**proof** –  
**have** 1:  $\vdash bi (f \longrightarrow g) \longrightarrow (f; \#True) \longrightarrow (g; \#True)$  **by** (*rule BiChopImpChop*)  
**from** 1 **show** *?thesis* **by** (*simp add: di-d-def*)  
**qed**

**lemma** *DiImpDi*:

**assumes**  $\vdash f \longrightarrow g$   
**shows**  $\vdash di f \longrightarrow di g$   
**proof** –  
**have** 1:  $\vdash f \longrightarrow g$  **using** *assms* **by** *auto*  
**hence** 2:  $\vdash f; \#True \longrightarrow g; \#True$  **by** (*rule LeftChopImpChop*)  
**from** 2 **show** *?thesis* **by** (*simp add: di-d-def*)  
**qed**

**lemma** *BiImpBiRule*:

**assumes**  $\vdash f \longrightarrow g$   
**shows**  $\vdash bi f \longrightarrow bi g$   
**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*:

$\vdash \text{bi } f \wedge (f1; g) \longrightarrow (f \wedge f1); g$

**proof** –

**have** 1:  $\vdash f \longrightarrow (f1 \longrightarrow f \wedge f1)$  **by** *auto*

**hence** 2:  $\vdash \text{bi } f \longrightarrow \text{bi } (f1 \longrightarrow f \wedge f1)$  **by** (rule *BiImpBiRule*)

**have** 3:  $\vdash \text{bi } (f1 \longrightarrow (f \wedge f1)) \longrightarrow f1; g \longrightarrow (f \wedge f1); g$  **by** (rule *BiChopImpChop*)

**from** 2 3 **show** ?thesis **using** *MP* **by** *fastforce*

qed

**lemma** *StateAndChopImport*:

$\vdash (\text{init } w) \wedge (f; g) \longrightarrow ((\text{init } w) \wedge f); g$

**proof** –

**have** 1:  $\vdash (\text{init } w) \longrightarrow \text{bi } (\text{init } w)$  **by** (rule *StateImpBi*)

**hence** 2:  $\vdash (\text{init } w) \wedge (f; g) \longrightarrow \text{bi } (\text{init } w) \wedge (f; g)$  **by** *auto*

**have** 3:  $\vdash \text{bi } (\text{init } w) \wedge (f; g) \longrightarrow ((\text{init } w) \wedge f); g$  **by** (rule *BiAndChopImport*)

**from** 2 3 **show** ?thesis **using** *MP* **by** *fastforce*

qed

## 5.4 Further Properties Di and Bi

**lemma** *ImpDi*:

$\vdash f \longrightarrow \text{di } f$

**proof** –

**have** 1:  $\vdash f; \text{empty} = f$  **by** (rule *ChopEmpty*)

**have** 2:  $\vdash \text{empty} \longrightarrow \#True$  **by** *auto*

**hence** 3:  $\vdash f; \text{empty} \longrightarrow f; \#True$  **by** (rule *RightChopImpChop*)

**have** 4:  $\vdash f \longrightarrow f; \#True$  **using** 1 3 **by** *fastforce*

**from** 4 **show** ?thesis **by** (simp add: di-d-def)

qed

**lemma DiState:**

$\vdash di (init w) = (init w)$

**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:**

$\vdash (init w); f \longrightarrow (init w)$

**using** *DiState* **by** (*auto simp: itl-defs*)

**lemma StateChopExportA:**

$\vdash ((init w) \wedge f); g \longrightarrow (init w)$

**using** *DiState* **by** (*auto simp: itl-defs*)

**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**  $\vdash f = (\text{init } w) \wedge f1$   
**shows**  $\vdash (f; g) = ((\text{init } w) \wedge (f1; g))$   
**proof** –  
**have** 1:  $\vdash f = ((\text{init } w) \wedge f1)$  **using** *assms* **by** *auto*  
**hence** 2:  $\vdash f; g = (((\text{init } w) \wedge f1); g)$  **by** (*rule LeftChopEqvChop*)  
**have** 3:  $\vdash ((\text{init } w) \wedge f1); g = ((\text{init } w) \wedge (f1; g))$  **by** (*rule StateAndChop*)  
**from** 2 3 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *DiIntro*:

$\vdash f \longrightarrow \text{di } f$   
**proof** –  
**have** 1:  $\vdash f; \text{empty} = f$  **by** (*rule ChopEmpty*)  
**have** 2:  $\vdash \text{empty} \longrightarrow \# \text{True}$  **by** *auto*  
**hence** 3:  $\vdash \Box(\text{empty} \longrightarrow \# \text{True})$  **by** (*rule BoxGen*)  
**have** 4:  $\vdash \Box(\text{empty} \longrightarrow \# \text{True}) \longrightarrow (f; \text{empty} \longrightarrow f; \# \text{True})$  **by** (*rule BoxChopImpChop*)  
**have** 5:  $\vdash f; \text{empty} \longrightarrow f; \# \text{True}$  **using** 3 4 *MP* **by** *fastforce*  
**hence** 6:  $\vdash f; \text{empty} \longrightarrow \text{di } f$  **by** (*simp add: di-d-def*)  
**from** 1 6 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *BiElim*:

$\vdash \text{bi } f \longrightarrow f$   
**proof** –  
**have** 1:  $\vdash \neg f \longrightarrow \text{di } (\neg f)$  **by** (*rule DiIntro*)  
**have** 2:  $\vdash (\neg f \longrightarrow \text{di } (\neg f)) \longrightarrow (\neg (\text{di } (\neg f)) \longrightarrow f)$  **by** *auto*  
**have** 3:  $\vdash \neg (\text{di } (\neg f)) \longrightarrow f$  **using** 1 2 *MP* **by** *blast*  
**from** 3 **show** *?thesis* **by** (*metis bi-d-def*)  
**qed**

**lemma** *BiContraPosImpDist*:

$\vdash \text{bi } (\neg g \longrightarrow \neg f) \longrightarrow (\text{bi } f) \longrightarrow (\text{bi } g)$   
**proof** –  
**have** 1:  $\vdash \text{bi } (\neg g \longrightarrow \neg f) \longrightarrow (\text{di } (\neg g)) \longrightarrow (\text{di } (\neg f))$  **by** (*rule BiImpDiImpDi*)  
**hence** 2:  $\vdash \text{bi } (\neg g \longrightarrow \neg f) \longrightarrow (\neg (\text{di } (\neg f))) \longrightarrow (\neg (\text{di } (\neg g)))$  **by** *auto*  
**from** 2 **show** *?thesis* **by** (*metis bi-d-def*)  
**qed**

**lemma** *BiImpDist*:

$\vdash \text{bi } (f \longrightarrow g) \longrightarrow (\text{bi } f) \longrightarrow (\text{bi } g)$   
**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:  $\vdash \text{bi } (\neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g))$  **by** (*rule BiGen*)  
**have** 4:  $\vdash \text{bi } (\neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g))$   
 $\longrightarrow$   
 $\text{bi } (f \longrightarrow g) \longrightarrow \text{bi } (\neg g \longrightarrow \neg f)$  **by** (*rule BiContraPosImpDist*)  
**have** 5:  $\vdash \text{bi } (f \longrightarrow g) \longrightarrow \text{bi } (\neg g \longrightarrow \neg f)$  **using** 3 4 *MP* **by** *blast*  
**have** 6:  $\vdash \text{bi } (\neg g \longrightarrow \neg f) \longrightarrow (\text{bi } f) \longrightarrow (\text{bi } g)$  **by** (*rule BiContraPosImpDist*)  
**from** 5 6 **show** *?thesis* **using** *lift-imp-trans* **by** *blast*

qed

**lemma** *IfChopEqvRule*:

**assumes**  $\vdash f = \text{if}_i \text{ (init } w) \text{ then } f1 \text{ else } f2$

**shows**  $\vdash f; g = \text{if}_i \text{ (init } w) \text{ then } (f1; g) \text{ else } (f2; g)$

**proof** –

**have** 1:  $\vdash f = \text{if}_i \text{ (init } w) \text{ then } f1 \text{ else } f2$

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

**hence** 2:  $\vdash f = (((\text{init } w) \wedge f1) \vee ((\text{init } (\neg w)) \wedge f2))$

**by** (*simp add: itl-defs Valid-def*)

**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** (*simp add: itl-defs Valid-def*)

qed

**lemma** *ChopOrEqvRule*:

**assumes**  $\vdash g = (g1 \vee g2)$

**shows**  $\vdash f; g = (f; g1) \vee (f; g2)$

**proof** –

**have** 1:  $\vdash g = (g1 \vee g2)$  **using** *assms* **by** *auto*

**hence** 2:  $\vdash f; g = (f; (g1 \vee g2))$  **by** (*rule RightChopEqvChop*)

**have** 3:  $\vdash f; (g1 \vee g2) = (f; g1 \vee f; g2)$  **by** (*rule ChopOrEqv*)

**from** 2 3 **show** *?thesis* **by** *fastforce*

qed

**lemma** *EmptyOrChopEqv*:

$\vdash (\text{empty} \vee f); g = (g \vee (f; g))$

**proof** –

**have** 1:  $\vdash (\text{empty} \vee f); g = ((\text{empty}; g) \vee (f; g))$  **by** (*rule OrChopEqv*)

**have** 2:  $\vdash \text{empty}; g = g$  **by** (*rule EmptyChop*)

**from** 1 2 **show** *?thesis* **by** *fastforce*

qed

**lemma** *EmptyOrNextChopEqv*:

$\vdash (\text{empty} \vee \circ f); g = (g \vee \circ(f; g))$

**proof** –

**have** 1:  $\vdash (\text{empty} \vee \circ f); g = (g \vee ((\circ f); g))$  **by** (*rule EmptyOrChopEqv*)

**have** 2:  $\vdash (\circ f); g = \circ(f; g)$  **by** (*rule NextChop*)

**from** 1 2 **show** *?thesis* **by** *fastforce*

qed

**lemma** *EmptyOrChopImpRule*:

**assumes**  $\vdash f \longrightarrow \text{empty} \vee f1$

**shows**  $\vdash f; g \longrightarrow g \vee (f1; g)$

**proof** –  
**have** 1:  $\vdash f \longrightarrow \text{empty} \vee f1$  **using** *assms* **by** *auto*  
**hence** 2:  $\vdash f; g \longrightarrow (\text{empty} \vee f1); g$  **by** (rule *LeftChopImpChop*)  
**have** 3:  $\vdash (\text{empty} \vee f1); g = (g \vee (f1; g))$  **by** (rule *EmptyOrChopEqv*)  
**from** 2 3 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *EmptyOrChopEqvRule*:  
**assumes**  $\vdash f = (\text{empty} \vee f1)$   
**shows**  $\vdash f; g = (g \vee (f1; g))$   
**proof** –  
**have** 1:  $\vdash f = (\text{empty} \vee f1)$  **using** *assms* **by** *auto*  
**hence** 2:  $\vdash f; g = ((\text{empty} \vee f1); g)$  **by** (rule *LeftChopEqvChop*)  
**have** 3:  $\vdash (\text{empty} \vee f1); g = (g \vee (f1; g))$  **by** (rule *EmptyOrChopEqv*)  
**from** 2 3 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *EmptyOrNextChopImpRule*:  
**assumes**  $\vdash f \longrightarrow \text{empty} \vee \circ f1$   
**shows**  $\vdash f; g \longrightarrow g \vee \circ(f1; g)$   
**proof** –  
**have** 1:  $\vdash f \longrightarrow \text{empty} \vee \circ f1$  **using** *assms* **by** *auto*  
**hence** 2:  $\vdash f; g \longrightarrow (\text{empty} \vee \circ f1); g$  **by** (rule *LeftChopImpChop*)  
**have** 3:  $\vdash (\text{empty} \vee \circ f1); g = (g \vee \circ(f1; g))$  **by** (rule *EmptyOrNextChopEqv*)  
**from** 2 3 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *EmptyOrNextChopEqvRule*:  
**assumes**  $\vdash f = (\text{empty} \vee \circ f1)$   
**shows**  $\vdash f; g = (g \vee \circ(f1; g))$   
**proof** –  
**have** 1:  $\vdash f = (\text{empty} \vee \circ f1)$  **using** *assms* **by** *auto*  
**hence** 2:  $\vdash f; g = ((\text{empty} \vee \circ f1); g)$  **by** (rule *LeftChopEqvChop*)  
**have** 3:  $\vdash (\text{empty} \vee \circ f1); g = (g \vee \circ(f1; g))$  **by** (rule *EmptyOrNextChopEqv*)  
**from** 2 3 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *ChopEmptyOrImpRule*:  
**assumes**  $\vdash g \longrightarrow \text{empty} \vee g1$   
**shows**  $\vdash f; g \longrightarrow f \vee (f; g1)$   
**proof** –  
**have** 1:  $\vdash g \longrightarrow \text{empty} \vee g1$  **using** *assms* **by** *auto*  
**hence** 2:  $\vdash f; g \longrightarrow (f; \text{empty}) \vee (f; g1)$  **by** (rule *ChopOrImpRule*)  
**have** 3:  $\vdash f; \text{empty} = f$  **by** (rule *ChopEmpty*)  
**from** 2 3 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *StateAndEmptyImpBoxState*:  
 $\vdash (\text{init } w) \wedge \text{empty} \longrightarrow \Box (\text{init } w)$   
**by** (*simp add: itl-defs Valid-def*)

**lemma** *BoxEqvAndBox*:

$\vdash \Box f = (f \wedge \Box f)$

**by** (*simp add: itl-defs Valid-def*) *fastforce*

**lemma** *NotBoxImpNotOrNotNextBox*:

$\vdash \neg(\Box f) \longrightarrow \neg f \vee \neg(\bigcirc(\Box f))$

**proof** –

**have** 1:  $\vdash f \wedge (\bigcirc(\Box f)) \longrightarrow \Box f$

**using** *BoxEqvAndEmptyOrNextBox* **by** *fastforce*

**hence** 2:  $\vdash \neg(\Box f) \longrightarrow \neg(f \wedge (\bigcirc(\Box f)))$  **by** *fastforce*

**have** 3:  $\vdash (\neg(f \wedge (\bigcirc(\Box f)))) = (\neg f \vee \neg(\bigcirc(\Box f)))$  **by** *auto*

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

**qed**

**lemma** *BoxStateChopBoxEqvBox*:

$\vdash \Box (init\ w); \Box (init\ w) = \Box (init\ w)$

**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:  $\vdash \Box (init\ w); \Box (init\ w) \longrightarrow \Box (init\ w)$   
**by** (*rule NextContra*)

**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 7 17 show ?thesis by fastforce  
qed

lemma NotBoxStateImpBoxYieldsNotBox:

$\vdash \neg(\Box(\text{init } w)) \longrightarrow (\Box(\text{init } w) \text{ yields } (\neg(\Box(\text{init } w))))$

proof –

have 1:  $\vdash \Box(\text{init } w); \Box(\text{init } w) = \Box(\text{init } w)$  by (rule BoxStateChopBoxEqvBox)

have 2:  $\vdash \Box(\text{init } w) = (\neg \neg(\Box(\text{init } w)))$  by auto

hence 3:  $\vdash \Box(\text{init } w); \Box(\text{init } w) = \Box(\text{init } w); (\neg \neg(\Box(\text{init } w)))$  by (rule RightChopEqvChop)

have 4:  $\vdash \neg(\Box(\text{init } w)) \longrightarrow \neg(\Box(\text{init } w); (\neg \neg(\Box(\text{init } w))))$  using 1 3 by auto

from 4 show ?thesis by (simp add: yields-d-def)

qed

lemma StateEqvBi:

$\vdash (\text{init } w) = \text{bi } (\text{init } w)$

proof –

have 1:  $\vdash (\text{init } w) \longrightarrow \text{bi } (\text{init } w)$  by (rule StateImpBi)

have 2:  $\vdash \text{bi } (\text{init } w) \longrightarrow (\text{init } w)$  by (rule BiElim)

from 1 2 show ?thesis by fastforce

qed

lemma TrueChopEqvDiamond:

$\vdash \# \text{True}; f = \Diamond f$

by (simp add: sometimes-d-def)

## 5.5 Properties of Da and Ba

lemma DaEqvDtDi:

$\vdash \text{da } f = \Diamond(\text{di } f)$

proof –

have 1:  $\vdash \# \text{True}; (f; \# \text{True}) = \# \text{True}; (f; \# \text{True})$  by auto

hence 2:  $\vdash \# \text{True}; (f; \# \text{True}) = \# \text{True}; \text{di } f$  by (simp add: di-d-def)

have 3:  $\vdash \# \text{True}; \text{di } f = \Diamond(\text{di } f)$  by (rule TrueChopEqvDiamond)

have 4:  $\vdash \# \text{True}; (f; \# \text{True}) = \Diamond(\text{di } f)$  using 2 3 by fastforce

from 4 show ?thesis by (simp add: da-d-def)

qed

lemma DaEqvDiDt:

$\vdash \text{da } f = \text{di } (\Diamond f)$

proof –

have 1:  $\vdash \# \text{True}; f = \Diamond f$  by (rule TrueChopEqvDiamond)

hence 2:  $\vdash (\# \text{True}; f); \# \text{True} = (\Diamond f); \# \text{True}$  by (rule LeftChopEqvChop)

hence 3:  $\vdash (\# \text{True}; f); \# \text{True} = \text{di } (\Diamond f)$  by (simp add: di-d-def)

have 4:  $\vdash \# \text{True}; (f; \# \text{True}) = (\# \text{True}; f); \# \text{True}$  by (rule ChopAssoc)

have 5:  $\vdash \# \text{True}; (f; \# \text{True}) = \text{di } (\Diamond f)$  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*:

$\vdash \text{di } f = (\neg(\text{bi } (\neg f)))$   
**proof** –  
**have** 1:  $\vdash \text{bi } (\neg f) = (\neg(\text{di } (\neg \neg f)))$  **by** (*simp add: bi-d-def*)  
**hence** 2:  $\vdash \text{di } (\neg \neg f) = (\neg(\text{bi } (\neg f)))$  **by** *auto*  
**have** 3:  $\vdash f = (\neg \neg f)$  **by** *auto*  
**hence** 4:  $\vdash \text{di } f = \text{di } (\neg \neg f)$  **by** (*rule DiEqvDi*)  
**from** 2 4 **show** *?thesis* **by** *auto*  
**qed**

**lemma** *ChopAndBoxImport*:  
 $\vdash f; g \wedge \Box h \longrightarrow f; (g \wedge h)$   
**proof** –  
**have** 1:  $\vdash \Box h \wedge f; g \longrightarrow f; (h \wedge g)$  **by** (*rule BoxAndChopImport*)  
**have** 2:  $\vdash f; (h \wedge g) = f; (g \wedge h)$  **by** (*rule ChopAndCommute*)  
**from** 1 2 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *AndChopAndCommute*:  
 $\vdash (f \wedge g); (f1 \wedge g1) = (g \wedge f); (g1 \wedge f1)$   
**proof** –  
**have** 1:  $\vdash (f \wedge g); (f1 \wedge g1) = (g \wedge f); (f1 \wedge g1)$  **by** (*rule AndChopCommute*)  
**have** 2:  $\vdash (g \wedge f); (f1 \wedge g1) = (g \wedge f); (g1 \wedge f1)$  **by** (*rule ChopAndCommute*)  
**from** 1 2 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *ChopImpChop*:  
**assumes**  $\vdash f \longrightarrow f1 \vdash g \longrightarrow g1$   
**shows**  $\vdash f; g \longrightarrow f1; g1$   
**proof** –  
**have** 1:  $\vdash f \longrightarrow f1$  **using** *assms* **by** *auto*  
**hence** 2:  $\vdash f; g \longrightarrow f1; g$  **by** (*rule LeftChopImpChop*)  
**have** 3:  $\vdash g \longrightarrow g1$  **using** *assms* **by** *auto*  
**hence** 4:  $\vdash f1; g \longrightarrow f1; g1$  **by** (*rule RightChopImpChop*)  
**from** 2 4 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *ChopEqvChop*:  
**assumes**  $\vdash f = f1 \vdash g = g1$   
**shows**  $\vdash f; g = f1; g1$   
**proof** –  
**have** 1:  $\vdash f = f1$  **using** *assms* **by** *auto*  
**hence** 2:  $\vdash f; g = f1; g$  **by** (*rule LeftChopEqvChop*)  
**have** 3:  $\vdash g = g1$  **using** *assms* **by** *auto*  
**hence** 4:  $\vdash f1; g = f1; g1$  **by** (*rule RightChopEqvChop*)  
**from** 2 4 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *BoxImpBoxImpBox*:  
 $\vdash \Box h \longrightarrow \Box(g \longrightarrow \Box h \wedge g)$   
**proof** –

**have** 1:  $\vdash \Box h \longrightarrow (g \longrightarrow \Box h \wedge g)$  **by** *auto*  
**hence** 2:  $\vdash \Box(\Box h) \longrightarrow \Box(g \longrightarrow \Box h \wedge g)$  **by** (*rule ImpBoxRule*)  
**have** 3:  $\vdash \Box h = \Box(\Box h)$  **by** (*rule BoxEqvBoxBox*)  
**from** 2 3 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *BoxChopImpChopBox*:  
 $\vdash \Box h \longrightarrow f; g \longrightarrow f; (\Box h \wedge g)$   
**proof** –  
**have** 1:  $\vdash \Box h \longrightarrow \Box(g \longrightarrow \Box h \wedge g)$  **by** (*rule BoxImpBoxImpBox*)  
**have** 2:  $\vdash \Box(g \longrightarrow \Box h \wedge g) \longrightarrow f; g \longrightarrow f; (\Box h \wedge g)$  **by** (*rule BoxChopImpChop*)  
**from** 1 2 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *NotChopEqvYieldsNot*:  
 $\vdash (\neg(f; g)) = f \text{ yields } (\neg g)$   
**proof** –  
**have** 1:  $\vdash g = (\neg \neg g)$  **by** *auto*  
**hence** 2:  $\vdash f; g = f; (\neg \neg g)$  **by** (*rule RightChopEqvChop*)  
**hence** 3:  $\vdash (\neg(f; g)) = (\neg(f; (\neg \neg g)))$  **by** *auto*  
**from** 3 **show** *?thesis* **by** (*simp add: yields-d-def*)  
**qed**

**lemma** *NotDiFalse*:  
 $\vdash \neg(di \# False)$   
**proof** –  
**have** 1:  $\vdash (init \# True) \longrightarrow bi (init \# True)$  **by** (*rule StateImpBi*)  
**hence** 2:  $\vdash \#True \longrightarrow bi \#True$  **by** (*auto simp: itl-defs*)  
**have** 3:  $\vdash \#True$  **by** *auto*  
**have** 4:  $\vdash bi \#True$  **using** 2 3 *MP* **by** *auto*  
**hence** 5:  $\vdash \neg(di \neg \#True)$  **by** (*simp add: bi-d-def*)  
**have** 6:  $\vdash (\neg \#True) = \#False$  **by** *auto*  
**hence** 7:  $\vdash di (\neg \#True) = di \#False$  **by** (*rule DiEqvDi*)  
**from** 5 7 **show** *?thesis* **by** *auto*  
**qed**

**lemma** *StateAndEmptyChop*:  
 $\vdash ((init\ w) \wedge empty); f = ((init\ w) \wedge f)$   
**proof** –  
**have** 1:  $\vdash ((init\ w) \wedge empty); f = ((init\ w) \wedge empty; f)$  **by** (*rule StateAndChop*)  
**have** 2:  $\vdash empty; f = f$  **by** (*rule EmptyChop*)  
**from** 1 2 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *StateAndNextChop*:  
 $\vdash ((init\ w) \wedge \bigcirc f); g = ((init\ w) \wedge \bigcirc(f; g))$   
**proof** –  
**have** 1:  $\vdash ((init\ w) \wedge \bigcirc f); g = ((init\ w) \wedge (\bigcirc f); g)$  **by** (*rule StateAndChop*)  
**have** 2:  $\vdash (\bigcirc f); g = \bigcirc(f; g)$  **by** (*rule NextChop*)  
**from** 1 2 **show** *?thesis* **by** *fastforce*

qed

**lemma** *NextAndEqvNextAndNext*:

$\vdash \bigcirc (f \wedge g) = (\bigcirc f \wedge \bigcirc g)$

**by** (*auto simp: itl-defs*)

**lemma** *NextStateAndChop*:

$\vdash \bigcirc(((init\ w) \wedge f); g) = (\bigcirc (init\ w) \wedge \bigcirc(f; g))$

**proof** –

**have** 1:  $\vdash ((init\ w) \wedge f); g = ((init\ w) \wedge f; g)$  **by** (*rule StateAndChop*)

**hence** 2:  $\vdash \bigcirc(((init\ w) \wedge f); g) = \bigcirc((init\ w) \wedge f; g)$  **by** (*rule NextEqvNext*)

**have** 3:  $\vdash \bigcirc((init\ w) \wedge f; g) = (\bigcirc (init\ w) \wedge \bigcirc(f; g))$  **by** (*rule NextAndEqvNextAndNext*)

**from** 2 3 **show** ?thesis **by** *fastforce*

qed

**lemma** *StateYieldsEqv*:

$\vdash ((init\ w) \longrightarrow (f\ yields\ g)) = ((init\ w) \wedge f)\ yields\ g$

**proof** –

**have** 1:  $\vdash ((init\ w) \wedge f); (\neg g) = ((init\ w) \wedge f; (\neg g))$  **by** (*rule StateAndChop*)

**hence** 2:  $\vdash ((init\ w) \longrightarrow \neg(f; (\neg g))) = (\neg(((init\ w) \wedge f); (\neg g)))$  **by** *auto*

**from** 2 **show** ?thesis **by** (*simp add: yields-d-def*)

qed

**lemma** *StateAndDi*:

$\vdash ((init\ w) \wedge di\ f) = di\ ((init\ w) \wedge f)$

**proof** –

**have** 1:  $\vdash ((init\ w) \wedge f); \#True = ((init\ w) \wedge f; \#True)$  **by** (*rule StateAndChop*)

**from** 1 **show** ?thesis **by** (*metis di-d-def inteq-reflection*)

qed

**lemma** *DiNext*:

$\vdash di(\bigcirc f) = \bigcirc(di\ f)$

**proof** –

**have** 1:  $\vdash (\bigcirc f); \#True = \bigcirc(f; \#True)$  **by** (*rule NextChop*)

**from** 1 **show** ?thesis **by** (*simp add: di-d-def*)

qed

**lemma** *DiNextState*:

$\vdash di(\bigcirc (init\ w)) = \bigcirc (init\ w)$

**proof** –

**have** 1:  $\vdash di(\bigcirc (init\ w)) = \bigcirc(di\ (init\ w))$  **by** (*rule DiNext*)

**have** 2:  $\vdash di\ (init\ w) = (init\ w)$  **by** (*rule DiState*)

**hence** 3:  $\vdash \bigcirc(di\ (init\ w)) = \bigcirc (init\ w)$  **by** (*rule NextEqvNext*)

**from** 1 3 **show** ?thesis **by** *fastforce*

qed

**lemma** *StateImpBiGen*:

**assumes**  $\vdash (init\ w) \longrightarrow f$

**shows**  $\vdash (init\ w) \longrightarrow bi\ f$

**proof** –

**have** 1:  $\vdash (\text{init } w) \longrightarrow f$  **using** *assms* **by** *auto*  
**hence** 2:  $\vdash \neg f \longrightarrow \neg (\text{init } w)$  **by** *auto*  
**hence** 3:  $\vdash \text{di } (\neg f) \longrightarrow \text{di } (\neg (\text{init } w))$  **by** (*rule DiImpDi*)  
**hence** 4:  $\vdash \text{di } (\neg f) \longrightarrow \text{di } (\text{init } (\neg w))$  **by** (*metis Initprop(2) inteq-reflection*)  
**have** 5:  $\vdash \text{di } (\text{init } (\neg w)) = (\text{init } (\neg w))$  **by** (*rule DiState*)  
**have** 6:  $\vdash \text{di } (\neg f) \longrightarrow \neg (\text{init } w)$  **using** 4 5 **using** *Initprop(2)* **by** *fastforce*  
**hence** 7:  $\vdash (\text{init } w) \longrightarrow \neg (\text{di } (\neg f))$  **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**

## 5.6 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** 6:  $\vdash (\neg(\Diamond(\neg f \wedge \text{empty}))) = \# \text{True}; (f \wedge \text{empty})$   
**using** *Finprop(4) sometimes-d-def* **by** (*metis int-eq int-simps(4)*)  
**from** 1 2 5 6 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *DiamondFin*:  
 $\vdash \Diamond(\text{fin } w) = \text{fin } w$   
**by** (*metis DiamondDiamondEqvDiamond FinEqvTrueChopAndEmpty TrueChopEqvDiamond inteq-reflection*)

**lemma** *ChopFinExportA*:  
 $\vdash f; (g \wedge \text{fin } w) \longrightarrow \text{fin } w$   
**using** *DiamondFin*  
**by** (*metis ChopAndB ChopImpDiamond inteq-reflection lift-imp-trans*)

**lemma** *FinImpBox*:  
 $\vdash \text{fin } w \longrightarrow \Box(\text{fin } w)$

**by** (*simp add: BoxImpBoxBox fin-d-def*)

**lemma** *FinAndChopImport*:

$\vdash (fin\ w) \wedge (f;g) \longrightarrow f;((fin\ w) \wedge g)$

**proof** –

**have** 1:  $\vdash fin\ w \longrightarrow \Box(fin\ w)$  **by** (*rule FinImpBox*)

**hence** 2:  $\vdash fin\ w \wedge f;g \longrightarrow \Box(fin\ w) \wedge (f;g)$  **by** *auto*

**have** 3:  $\vdash \Box(fin\ w) \wedge (f;g) \longrightarrow f;((fin\ w) \wedge g)$  **using** *BoxAndChopImport* **by** *blast*

**from** 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*

**qed**

**lemma** *FinAndChop*:

$\vdash (f;(g \wedge fin\ w)) = (fin\ w \wedge f;g)$

**using** *FinAndChopImport ChopFinExportA ChopAndA ChopAndCommute* **by** *fastforce*

**lemma** *ChopAndEmptyEqvEmptyChopEmpty*:

$\vdash ((f;g) \wedge empty) = (f \wedge empty);(g \wedge empty)$

**by** (*auto simp: itl-defs*)

**lemma** *FinAndEmpty*:

$\vdash ((fin\ w) \wedge empty) = (w \wedge empty)$

**proof** –

**have** 1:  $\vdash ((fin\ w) \wedge empty) = (\#True;(w \wedge empty) \wedge empty)$

**using** *FinEqvTrueChopAndEmpty* **by** *fastforce*

**have** 2:  $\vdash (\#True;(w \wedge empty) \wedge empty) = ((\#True \wedge empty);(w \wedge empty))$

**using** *ChopAndEmptyEqvEmptyChopEmpty* [of *LIFT*( $\#True$ ) *LIFT*( $w \wedge empty$ )]

**by** (*metis AndChopA ChopAndA ChopAndEmptyEqvEmptyChopEmpty Prop11 Prop12 int-eq*)

**have** 3:  $\vdash (\#True \wedge empty);(w \wedge empty) = (empty;(w \wedge empty))$

**using** *LeftChopEqvChop* **by** *fastforce*

**have** 4:  $\vdash (empty;(w \wedge empty)) = (w \wedge empty)$

**using** *EmptyChop* **by** *blast*

**from** 1 2 3 4 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *AndFinEqvChopAndEmpty*:

$\vdash (f \wedge fin\ g) = f;(g \wedge empty)$

**proof** –

**have** 1:  $\vdash (f \wedge fin\ g) = (f;empty \wedge fin\ g)$

**using** *ChopEmpty* **by** (*metis int-eq*)

**have** 2:  $\vdash (fin\ g \wedge f;empty) = (f;(empty \wedge fin\ g))$

**using** *FinAndChop* **by** *fastforce*

**have** 3:  $\vdash (empty \wedge fin\ g) = (fin\ g \wedge empty)$

**by** *auto*

**have** 4:  $\vdash (fin\ g \wedge empty) = (g \wedge empty)$

**using** *FinAndEmpty* **by** *metis*

**have** 5:  $\vdash (empty \wedge fin\ g) = (g \wedge empty)$

**using** 3 4 **by** *auto*

**hence** 6:  $\vdash f;(empty \wedge fin\ g) = f;(g \wedge empty)$

**using** *RightChopEqvChop* **by** *blast*

**from 1 2 5 show ?thesis by (metis integ-reflection lift-and-com)**  
**qed**

**lemma AndFinEqvChopStateAndEmpty:**

$\vdash (f \wedge \text{fin } (\text{init } w)) = f; ((\text{init } w) \wedge \text{empty})$

**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 1 BoxElim DiamondFin NextDiamondImpDiamond int-eq lift-and-com lift-imp-trans Prop10)**

**have 9:**  $\vdash (((\text{empty} \longrightarrow \text{init } w) \wedge \text{empty}) \vee ((\text{empty} \longrightarrow \text{init } w) \wedge \bigcirc (\text{fin } (\text{init } w)))) =$

$((\text{init } w) \wedge \text{empty}) \vee \bigcirc (\text{fin } (\text{init } w))$

**using 7 8 by auto**

**from 5 6 8 9 show ?thesis by fastforce**

**qed**

**lemma FinChopEqvOr:**

$\vdash (\text{fin } (\text{init } w)); f = (((\text{init } w) \wedge f) \vee \bigcirc ((\text{fin } (\text{init } w)); f))$

**proof –**

**have 1:**  $\vdash \text{fin } (\text{init } w) = (((\text{init } w) \wedge \text{empty}) \vee \bigcirc (\text{fin } (\text{init } w)))$

**by (rule FinStateEqvStateAndEmptyOrNextFinState)**

**hence 2:**  $\vdash (\text{fin } (\text{init } w)); f = (((\text{init } w) \wedge \text{empty}) \vee \bigcirc (\text{fin } (\text{init } w))); f$

**by (rule LeftChopEqvChop)**

**have 3:**  $\vdash (((\text{init } w) \wedge \text{empty}) \vee \bigcirc (\text{fin } (\text{init } w))); f$

$= (((\text{init } w) \wedge \text{empty}); f \vee \bigcirc (\text{fin } (\text{init } w))); f$

**by (rule OrChopEqv)**

**have 4:**  $\vdash ((\text{init } w) \wedge \text{empty}); f = ((\text{init } w) \wedge f)$

**by (rule StateAndEmptyChop)**

**have 5:**  $\vdash \bigcirc (\text{fin } (\text{init } w)); f = \bigcirc ((\text{fin } (\text{init } w)); f)$

**by (rule NextChop)**

**from 2 3 4 5 show ?thesis by fastforce**

qed

**lemma** *FinChopEqvDiamond*:

$\vdash ( \text{fin } (\text{init } w)); f = \Diamond ((\text{init } w) \wedge f)$

**proof** –

**have** 1:  $\vdash ( \text{fin } (\text{init } w)) = (\# \text{True}; ((\text{init } w) \wedge \text{empty}))$

**by** (*rule FinEqvTrueChopAndEmpty*)

**hence** 2:  $\vdash ( \text{fin } (\text{init } w)); f = (\# \text{True}; ((\text{init } w) \wedge \text{empty}); f)$

**by** (*rule LeftChopEqvChop*)

**have** 3:  $\vdash \# \text{True}; ((\text{init } w) \wedge \text{empty}); f = (\# \text{True}; ((\text{init } w) \wedge \text{empty}); f)$

**by** (*rule ChopAssoc*)

**have** 4:  $\vdash \# \text{True}; ((\text{init } w) \wedge \text{empty}); f = \Diamond ( ( (\text{init } w) \wedge \text{empty}); f)$

**by** (*simp add: sometimes-d-def*)

**have** 5:  $\vdash ( (\text{init } w) \wedge \text{empty}); f = ((\text{init } w) \wedge f)$

**using** *StateAndEmptyChop* **by** *blast*

**hence** 6:  $\vdash \Diamond ( ( (\text{init } w) \wedge \text{empty}); f) = \Diamond ( (\text{init } w) \wedge f)$

**by** (*rule DiamondEqvDiamond*)

**from** 2 3 4 6 **show** *?thesis* **by** *fastforce*

qed

**lemma** *NotDiamondAndNot*:

$\vdash \neg( \Diamond ( f \wedge \neg f))$

**proof** –

**have** 1:  $\vdash (\neg( \Diamond ( f \wedge \neg f))) = \Box(\neg(f \wedge \neg f))$  **using** *NotDiamondNotEqvBox* **by** *fastforce*

**have** 2:  $\vdash \neg(f \wedge \neg f)$  **by** *simp*

**have** 3:  $\vdash \Box(\neg(f \wedge \neg f))$  **using** 2 **by** (*simp add: BoxGen*)

**from** 1 3 **show** *?thesis* **by** *fastforce*

qed

**lemma** *FinYields*:

$\vdash ( \text{fin } (\text{init } w)) \text{ yields } (\text{init } w)$

**proof** –

**have** 1:  $\vdash ( \text{fin } (\text{init } w)); (\neg(\text{init } w)) = \Diamond((\text{init } w) \wedge \neg(\text{init } w))$  **by** (*rule FinChopEqvDiamond*)

**have** 2:  $\vdash \neg( \Diamond((\text{init } w) \wedge \neg(\text{init } w)))$  **by** (*rule NotDiamondAndNot*)

**have** 3:  $\vdash \neg( ( \text{fin } (\text{init } w)); (\neg(\text{init } w)))$  **using** 1 2 **by** *fastforce*

**from** 3 **show** *?thesis* **by** (*simp add: yields-d-def*)

qed

**lemma** *ImpAndFinStateOrFinNotState*:

$\vdash f \longrightarrow (f \wedge \text{fin } (\text{init } w)) \vee \text{fin } (\neg(\text{init } w))$

**by** (*simp add: itl-defs Valid-def*)

**lemma** *AndFinChopEqvStateAndChop*:

$\vdash (f \wedge \text{fin } (\text{init } w)); g = f; ((\text{init } w) \wedge g)$

**proof** –

**have** 1:  $\vdash ( \text{fin } (\text{init } w)) \text{ yields } (\text{init } w)$

**by** (*rule FinYields*)

**have** 2:  $\vdash f \wedge \text{fin } (\text{init } w) \longrightarrow \text{fin } (\text{init } w)$

**by** *auto*

**hence** 3:  $\vdash ( \text{fin } (\text{init } w)) \text{ yields } (\text{init } w) \longrightarrow (f \wedge \text{fin } (\text{init } w)) \text{ yields } (\text{init } w)$

by (rule *LeftYieldsImpYields*)  
 have 4:  $\vdash (f \wedge \text{fin } (\text{init } w)) \text{ yields } (\text{init } w)$   
 using 1 3 MP by fastforce  
 have 5:  $\vdash (f \wedge \text{fin } (\text{init } w)); g \wedge (f \wedge \text{fin } (\text{init } w)) \text{ yields } (\text{init } w)$   
 $\longrightarrow (f \wedge \text{fin } (\text{init } w)); (g \wedge (\text{init } w))$   
 by (rule *ChopAndYieldsImp*)  
 have 6:  $\vdash (f \wedge \text{fin } (\text{init } w)); g \longrightarrow (f \wedge \text{fin } (\text{init } w)); (g \wedge (\text{init } w))$   
 using 4 5 by fastforce  
 have 7:  $\vdash (f \wedge \text{fin } (\text{init } w)); (g \wedge (\text{init } w)) \longrightarrow f; (g \wedge (\text{init } w))$   
 by (rule *AndChopA*)  
 have 8:  $\vdash g \wedge (\text{init } w) \longrightarrow (\text{init } w) \wedge g$   
 by auto  
 hence 9:  $\vdash f; (g \wedge (\text{init } w)) \longrightarrow f; ((\text{init } w) \wedge g)$   
 by (rule *RightChopImpChop*)  
 have 10:  $\vdash (f \wedge \text{fin } (\text{init } w)); g \longrightarrow f; ((\text{init } w) \wedge g)$   
 using 6 7 9 by fastforce  
 have 11:  $\vdash f \longrightarrow (f \wedge \text{fin } (\text{init } w)) \vee \text{fin } (\neg (\text{init } w))$   
 by (rule *ImpAndFinStateOrFinNotState*)  
 hence 12:  $\vdash f; ((\text{init } w) \wedge g) \longrightarrow$   
 $((f \wedge \text{fin } (\text{init } w)) \vee \text{fin } (\neg (\text{init } w))); ((\text{init } w) \wedge g)$   
 by (rule *LeftChopImpChop*)  
 have 13:  $\vdash ((f \wedge \text{fin } (\text{init } w)) \vee \text{fin } (\neg (\text{init } w))); ((\text{init } w) \wedge g)$   
 $=$   
 $((f \wedge \text{fin } (\text{init } w)); ((\text{init } w) \wedge g) \vee (\text{fin } (\neg (\text{init } w))); ((\text{init } w) \wedge g))$   
 by (rule *OrChopEqv*)  
 have 14:  $\vdash (\text{fin } (\text{init } (\neg w))); ((\text{init } w) \wedge g) \longrightarrow \Diamond (\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g)$   
 using *FinChopEqvDiamond* by fastforce  
 have 141:  $\vdash \neg (\Diamond (\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g)) \longrightarrow$   
 $\neg (\text{fin } (\text{init } (\neg w))); ((\text{init } w) \wedge g)$   
 using 14 by fastforce  
 have 15:  $\vdash \neg (\Diamond (\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g))$   
 using *NotDiamondAndNot Initprop(2)* by (auto simp: *itl-defs*)  
 have 151:  $\vdash \neg (\text{fin } (\text{init } (\neg w))); ((\text{init } w) \wedge g)$   
 using 15 141 by fastforce  
 have 1511:  $\vdash (\text{fin } (\neg (\text{init } w))); ((\text{init } w) \wedge g) \longrightarrow \#False$   
 using 151 by (metis *Initprop(2)* *int-simps(14)* *inteq-reflection*)  
 have 152:  $\vdash (f \wedge \text{fin } (\text{init } w)); ((\text{init } w) \wedge g) \vee (\text{fin } (\neg (\text{init } w))); ((\text{init } w) \wedge g) \longrightarrow$   
 $(f \wedge \text{fin } (\text{init } w)); ((\text{init } w) \wedge g)$   
 using 1511 by fastforce  
 have 16:  $\vdash f; ((\text{init } w) \wedge g) \longrightarrow (f \wedge \text{fin } (\text{init } w)); ((\text{init } w) \wedge g)$   
 using 12 13 152 by fastforce  
 have 17:  $\vdash (f \wedge \text{fin } (\text{init } w)); ((\text{init } w) \wedge g) \longrightarrow (f \wedge \text{fin } (\text{init } w)); g$   
 by (rule *ChopAndB*)  
 have 18:  $\vdash f; ((\text{init } w) \wedge g) \longrightarrow (f \wedge \text{fin } (\text{init } w)); g$   
 using 16 17 by fastforce  
 from 10 18 show ?thesis by fastforce  
 qed

lemma *DiAndFinEqvChopState*:

$\vdash \text{di } (f \wedge \text{fin } (\text{init } w)) = f; (\text{init } w)$

**proof** –

**have** 1:  $\vdash (f \wedge \text{fin}(\text{init } w)); \# \text{True} = f; ((\text{init } w) \wedge \# \text{True})$  **by** (rule *AndFinChopEqvStateAndChop*)  
**have** 2:  $\vdash ((\text{init } w) \wedge \# \text{True}) = (\text{init } w)$  **by** *auto*  
**hence** 3:  $\vdash (f; ((\text{init } w) \wedge \# \text{True})) = (f; (\text{init } w))$  **by** (rule *RightChopEqvChop*)  
**have** 4:  $\vdash (f \wedge \text{fin}(\text{init } w)); \# \text{True} = f; (\text{init } w)$  **using** 1 3 **by** *auto*  
**from** 4 **show** ?thesis **by** (simp add: di-d-def)  
**qed**

**lemma** *FinNotStateEqvNotFinState*:

$\vdash \text{fin}(\text{init } (\neg w)) = (\neg (\text{fin}(\text{init } w)))$

**using** *FinEqvTrueChopAndEmpty*

**by** (metis (no-types, hide-lams) *Finprop(4) Initprop(2) int-eq int-simps(4) int-simps(7) sometimes-d-def*)

**lemma** *BiImpFinEqvYieldsState*:

$\vdash \text{bi}(f \longrightarrow \text{fin}(\text{init } w)) = f \text{ yields } (\text{init } w)$

**proof** –

**have** 1:  $\vdash \text{di}(f \wedge \text{fin}(\text{init } (\neg w))) = f; (\text{init } (\neg w))$   
**by** (rule *DiAndFinEqvChopState*)  
**have** 2:  $\vdash (f \wedge \text{fin}(\text{init } (\neg w))) = (f \wedge \neg(\text{fin}(\text{init } w)))$   
**using** *FinNotStateEqvNotFinState* **by** *fastforce*  
**have** 3:  $\vdash (f \wedge \neg(\text{fin}(\text{init } w))) = (\neg(f \longrightarrow \text{fin}(\text{init } w)))$   
**by** *auto*  
**have** 4:  $\vdash (f \wedge \text{fin}(\text{init } (\neg w))) = (\neg(f \longrightarrow \text{fin}(\text{init } w)))$   
**using** 2 3 **by** *fastforce*  
**hence** 5:  $\vdash \text{di}(f \wedge \text{fin}(\text{init } (\neg w))) = \text{di}(\neg(f \longrightarrow \text{fin}(\text{init } w)))$   
**by** (rule *DiEqvDi*)  
**have** 6:  $\vdash \text{di}(\neg(f \longrightarrow \text{fin}(\text{init } w))) = (\neg(\text{bi}(f \longrightarrow \text{fin}(\text{init } w))))$   
**by** (rule *DiNotEqvNotBi*)  
**have** 7:  $\vdash \neg(\text{bi}(f \longrightarrow \text{fin}(\text{init } w))) = f; (\text{init } (\neg w))$   
**using** 1 5 6 *Initprop* **by** *fastforce*  
**hence** 8:  $\vdash \text{bi}(f \longrightarrow \text{fin}(\text{init } w)) = (\neg(f; (\neg(\text{init } w))))$   
**by** (metis *Initprop(2) int-eq int-simps(7)*)  
**from** 8 **show** ?thesis **by** (simp add: yields-d-def)  
**qed**

**lemma** *StateImpYields*:

**assumes**  $\vdash (\text{init } w) \wedge f \longrightarrow \text{fin}(\text{init } w1)$

**shows**  $\vdash (\text{init } w) \longrightarrow (f \text{ yields } (\text{init } w1))$

**proof** –

**have** 1:  $\vdash (\text{init } w) \wedge f \longrightarrow \text{fin}(\text{init } w1)$  **using** *assms* **by** *auto*  
**hence** 2:  $\vdash (\text{init } w) \longrightarrow (f \longrightarrow \text{fin}(\text{init } w1))$  **by** *auto*  
**hence** 3:  $\vdash (\text{init } w) \longrightarrow \text{bi}(f \longrightarrow \text{fin}(\text{init } w1))$  **by** (rule *StateImpBiGen*)  
**have** 4:  $\vdash \text{bi}(f \longrightarrow \text{fin}(\text{init } w1)) = f \text{ yields } (\text{init } w1)$  **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*:

$\vdash \Box g \longrightarrow f \text{ yields } g$   
**proof** –  
**have** 1:  $\vdash f; (\neg g) \longrightarrow \Diamond(\neg g)$  **by** (*rule ChopImpDiamond*)  
**hence** 2:  $\vdash \neg (\Diamond(\neg g)) \longrightarrow \neg (f; (\neg g))$  **by** *auto*  
**from** 2 **show** ?thesis **by** (*simp add: yields-d-def always-d-def*)  
**qed**

**lemma** *BoxEqvTrueYields*:

$\vdash \Box f = \#True \text{ yields } f$

**proof** –

**have** 1:  $\vdash \#True; (\neg f) = \Diamond (\neg f)$  **by** (rule *TrueChopEqvDiamond*)

**hence** 2:  $\vdash (\neg (\#True; (\neg f))) = (\neg (\Diamond (\neg f)))$  **by** *auto*

**have** 3:  $\vdash \Box f = (\neg (\Diamond (\neg f)))$  **by** (simp add: *always-d-def*)

**have** 4:  $\vdash \Box f = (\neg (\#True; (\neg f)))$  **using** 2 3 **by** *fastforce*

**from** 4 **show** ?thesis **by** (simp add: *yields-d-def*)

**qed**

**lemma** *YieldsGen*:

**assumes**  $\vdash g$

**shows**  $\vdash f \text{ yields } g$

**proof** –

**have** 1:  $\vdash g$  **using** *assms* **by** *auto*

**hence** 2:  $\vdash \Box g$  **by** (rule *BoxGen*)

**have** 3:  $\vdash \Box g \longrightarrow f \text{ yields } g$  **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** (auto simp: *itl-defs*)

**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** (auto simp: itl-defs)  
**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** *ChopMoreImpMore*:

$\vdash f ; \text{more} \longrightarrow \text{more}$

**proof** –  
**have** 1:  $\vdash f ; \text{more} \longrightarrow \Diamond \text{more}$  **by** (rule *ChopImpDiamond*)  
**have** 2:  $\vdash \Diamond \text{more} \longrightarrow \text{more}$  **by** (auto simp: itl-defs)  
**from** 1 2 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *MoreChopEqvNextDiamond*:

$\vdash \text{more} ; f = \bigcirc (\Diamond f)$

**proof** –  
**have** 1:  $\vdash \text{more} ; f = (\bigcirc \# \text{True}) ; f$  **by** (simp add: more-d-def)  
**have** 2:  $\vdash (\bigcirc \# \text{True}) ; f = \bigcirc (\# \text{True} ; f)$  **by** (rule *NextChop*)  
**have** 3:  $\vdash \text{more} ; f = \bigcirc (\# \text{True} ; f)$  **using** 1 2 **by** *fastforce*  
**from** 3 **show** ?thesis **by** (simp add: sometimes-d-def)  
**qed**

**lemma** *WeakNextBoxImpMoreYields*:

$\vdash \text{more} \text{ yields } f = \text{wnext}(\Box f)$

**proof** –  
**have** 1:  $\vdash \text{more} ; (\neg f) = \bigcirc (\Diamond (\neg f))$  **by** (rule *MoreChopEqvNextDiamond*)  
**have** 2:  $\vdash \bigcirc (\Diamond (\neg f)) = \bigcirc (\neg (\Box f))$  **by** (auto simp: always-d-def)  
**have** 3:  $\vdash \bigcirc (\neg (\Box f)) = (\neg (\text{wnext}(\Box f)))$  **by** (auto simp: wnext-d-def)  
**have** 4:  $\vdash \text{more} ; (\neg f) = (\neg (\text{more yields } f))$  **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** *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*:

$\vdash \text{di } f = \text{di } (\text{di } f)$

**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*:

$\vdash \text{bi } f = \text{bi } (\text{bi } f)$

**proof** –

**have** 1:  $\vdash \text{di } (\neg f) = \text{di } (\text{di } (\neg f))$  **by** (rule *DiEqvDiDi*)

**have** 2:  $\vdash \text{di } (\neg f) = (\neg (\text{bi } f))$  **by** (rule *DiNotEqvNotBi*)

**hence** 3:  $\vdash \text{di } (\text{di } (\neg f)) = \text{di } (\neg (\text{bi } f))$  **by** (rule *DiEqvDi*)

**have** 4:  $\vdash \text{di } (\neg f) = \text{di } (\neg (\text{bi } f))$  **using** 1 3 **by** *fastforce*

**hence** 5:  $\vdash (\neg (\text{di } (\neg f))) = (\neg (\text{di } (\neg (\text{bi } f))))$  **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*:

$\vdash \text{di skip} = \text{more}$

**proof** –

**have** 1:  $\vdash \text{skip} ; \# \text{True} = \circ \# \text{True}$  **by** (rule *SkipChopEqvNext*)

**have** 2:  $\vdash \circ \# \text{True} = \text{more}$  **by** (auto simp: *more-d-def*)

**have** 3:  $\vdash \text{skip} ; \# \text{True} = \text{more}$  **using** 1 2 **by** *fastforce*

**from** 3 **show** ?thesis **by** (simp add: *di-d-def*)

**qed**

**lemma** *DiMoreEqvMore*:

$\vdash \text{di more} = \text{more}$

**proof** –

**have** 1:  $\vdash \text{di} (\circ \# \text{True}) = \circ (\text{di} \# \text{True})$  **by** (rule *DiNext*)

**have** 2:  $\vdash \circ (\text{di} \# \text{True}) \longrightarrow \text{more}$  **by** (auto simp: *itl-defs*)

**have** 3:  $\vdash \text{di} (\circ \# \text{True}) \longrightarrow \text{more}$  **using** 1 2 **by** *fastforce*

**hence** 4:  $\vdash \text{di more} \longrightarrow \text{more}$  **by** (simp add: *more-d-def*)

**have** 5:  $\vdash \text{more} \longrightarrow \text{di more}$  **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*:

$\vdash da\ f = (\neg (ba\ (\neg\ f)))$

**proof** –

**have** 1:  $\vdash ba\ (\neg\ f) = (\neg (da\ (\neg\ \neg\ f)))$  **by** (*simp add: ba-d-def*)

**hence** 2:  $\vdash da\ (\neg\ \neg\ f) = (\neg (ba\ (\neg\ f)))$  **by** *fastforce*

**have** 3:  $\vdash f = (\neg\ \neg\ f)$  **by** *simp*

**hence** 4:  $\vdash da\ f = da\ (\neg\ \neg\ f)$  **by** (*rule DaEqvDa*)

**from** 2 4 **show** *?thesis* **by** *simp*

**qed**

**lemma** *BaElim*:

$\vdash ba\ f \longrightarrow f$

**proof** –

**have** 1:  $\vdash ba\ f = \Box(bi\ f)$  **by** (*rule BaEqvBtBi*)

**have** 2:  $\vdash bi\ f \longrightarrow f$  **by** (*rule BiElim*)

**hence** 3:  $\vdash \Box(bi\ f \longrightarrow f)$  **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** *BaAndEqv*:

$\vdash ba (f \wedge g) = (ba f \wedge ba g)$   
**proof** –  
**have** 1:  $\vdash ba (f \wedge g) = \Box (bi (f \wedge g))$   
**by** (*rule BaEqvBtBi*)  
**have** 2:  $\vdash bi (f \wedge g) = (bi f \wedge bi g)$   
**by** (*auto simp: itl-defs*)  
**hence** 3:  $\vdash \Box (bi (f \wedge g)) = \Box (bi f \wedge bi g)$   
**using** *BoxEqvBox* **by** *blast*  
**have** 4:  $\vdash \Box (bi f \wedge bi g) = (\Box (bi f) \wedge \Box (bi g))$   
**by** (*metis 2 BoxAndBoxEqvBoxRule integ-reflection*)  
**have** 5:  $\vdash ba f = \Box (bi f)$   
**by** (*rule BaEqvBtBi*)  
**have** 6:  $\vdash ba g = \Box (bi g)$   
**by** (*rule BaEqvBtBi*)  
**from** 1 3 4 5 6 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *BaImpBaEqvBa*:

$\vdash ba (f = g) \longrightarrow (ba f = ba g)$   
**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** 3:  $\vdash ba (f = g) = ba ((f \longrightarrow g) \wedge (g \longrightarrow f))$  **by** (*auto simp: itl-defs*)  
**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:  $\vdash \Diamond (\Diamond f) = \#True;(\#True;f)$   
     **by** (*simp add: sometimes-d-def*)  
**have** 2:  $\vdash \#True;(\#True;f) = (\#True;\#True);f$   
     **by** (*rule ChopAssoc*)  
**have** 3:  $\vdash (\#True;\#True);f = \#True;f$   
     **using** *LeftChopEqvChop TrueEqvTrueChopTrue* **by** (*metis int-eq*)  
**have** 4:  $\vdash \#True;f = \Diamond f$   
     **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*:

$\vdash \text{ba } (f \longrightarrow f1) \longrightarrow f; g \longrightarrow f1; g$

**proof** –

**have** 1:  $\vdash \text{ba } (f \longrightarrow f1) \longrightarrow \text{bi } (f \longrightarrow f1)$  **by** (rule *BaImpBi*)

**have** 2:  $\vdash \text{bi } (f \longrightarrow f1) \longrightarrow f; g \longrightarrow f1; g$  **by** (rule *BiChopImpChop*)

**from** 1 2 **show** ?thesis **by** fastforce

qed

**lemma** *BaRightChopImpChop*:

$\vdash \text{ba } (g \longrightarrow g1) \longrightarrow f; g \longrightarrow f; g1$

**proof** –

**have** 1:  $\vdash \text{ba } (g \longrightarrow g1) \longrightarrow \Box(g \longrightarrow g1)$  **by** (rule *BaImpBt*)

**have** 2:  $\vdash \Box(g \longrightarrow g1) \longrightarrow f; g \longrightarrow f; g1$  **by** (rule *BoxChopImpChop*)

**from** 1 2 **show** ?thesis **by** fastforce

qed

**lemma** *ChopAndBaImport*:

$\vdash (f; f1) \wedge \text{ba } g \longrightarrow (f \wedge g); (f1 \wedge g)$

**proof** –

**have** 1:  $\vdash \text{ba } g \wedge (f; f1) \longrightarrow (g \wedge f); (g \wedge f1)$  **by** (rule *BaAndChopImport*)

**have** 2:  $\vdash (g \wedge f); (g \wedge f1) = (f \wedge g); (f1 \wedge g)$  **by** (rule *AndChopAndCommute*)

**from** 1 2 **show** ?thesis **by** fastforce

qed

**lemma** *BaImpBaImpBaAnd*:

$\vdash \text{ba } h \longrightarrow \text{ba}(g \longrightarrow \text{ba } h \wedge g)$

**proof** –

**have** 1:  $\vdash \text{ba } h \longrightarrow (g \longrightarrow \text{ba } h \wedge g)$  **by** fastforce

**hence** 2:  $\vdash \text{ba}(\text{ba } h) \longrightarrow \text{ba}(g \longrightarrow \text{ba } h \wedge g)$  **by** (rule *BaImpBa*)

**have** 3:  $\vdash \text{ba } h = \text{ba}(\text{ba } h)$  **by** (rule *BaEqvBaBa*)

**from** 2 3 **show** ?thesis **by** fastforce

qed

**lemma** *BaChopImpChopBa*:

$\vdash \text{ba } f \longrightarrow g; g1 \longrightarrow g; ((\text{ba } f) \wedge g1)$

**proof** –

**have** 1:  $\vdash \text{ba } f \longrightarrow \text{ba } (g1 \longrightarrow (\text{ba } f) \wedge g1)$  **by** (rule *BaImpBaImpBaAnd*)

**have** 2:  $\vdash \text{ba } (g1 \longrightarrow \text{ba } f \wedge g1) \longrightarrow g; g1 \longrightarrow g; (\text{ba } f \wedge g1)$  **by** (rule *BaRightChopImpChop*)

**from** 1 2 **show** ?thesis **by** fastforce

qed

**lemma** *DiNotBaImpNotBa*:

$\vdash \text{di } (\neg (\text{ba } f)) \longrightarrow \neg (\text{ba } f)$

**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:  $\vdash \Diamond\ (fin\ f) = \#True;(\#True;(f \wedge empty))$   
**by** (metis ChopEqvChop TrueEqvTrueChopTrue inteq-reflection sometimes-d-def)  
**have** 3:  $\vdash \#True;(\#True;(f \wedge empty)) = (\#True;\#True);(f \wedge empty)$   
**by** (rule ChopAssoc)  
**have** 4:  $\vdash (\#True;\#True);(f \wedge empty) = \#True;(f \wedge empty)$   
**using** TrueEqvTrueChopTrue **using** LeftChopEqvChop **by** (metis int-eq)  
**from** 1 2 3 4 **show** ?thesis **by** fastforce  
**qed**

**lemma** ChopFinImpFin:

$\vdash f; fin\ (init\ w) \longrightarrow fin\ (init\ w)$   
**proof** –  
**have** 1:  $\vdash f; fin\ (init\ w) \longrightarrow \Diamond\ (fin\ (init\ w))$  **by** (rule ChopImpDiamond)  
**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** FinImpYieldsFin:

$\vdash fin\ (init\ w) \longrightarrow f\ yields\ (fin\ (init\ w))$   
**proof** –  
**have** 1:  $\vdash f; fin\ (init\ (\neg\ w)) \longrightarrow fin\ (init\ (\neg\ w))$   
**by** (rule ChopFinImpFin)  
**have** 2:  $\vdash fin\ (init\ (\neg\ w)) = (\neg\ (fin\ (init\ w)))$   
**using** FinNotStateEqvNotFinState **by** blast  
**hence** 3:  $\vdash f; fin\ (init\ (\neg\ w)) = f; (\neg\ (fin\ (init\ w)))$   
**by** (rule RightChopEqvChop)  
**have** 4:  $\vdash f; (\neg\ (fin\ (init\ w))) \longrightarrow \neg\ (fin\ (init\ w))$   
**using** 1 2 3 **by** fastforce  
**hence** 5:  $\vdash fin\ (init\ w) \longrightarrow \neg\ (f; (\neg\ (fin\ (init\ w))))$

by fastforce  
 from 5 show ?thesis by (simp add: yields-d-def)  
 qed

lemma ChopAndFin:

$\vdash ((f; g) \wedge \text{fin } (\text{init } w)) = f; (g \wedge \text{fin } (\text{init } w))$   
 proof –  
 have 1:  $\vdash \text{fin } (\text{init } w) \longrightarrow f \text{ yields } (\text{fin } (\text{init } w))$   
 by (rule FinImpYieldsFin)  
 hence 2:  $\vdash (f; g) \wedge \text{fin } (\text{init } w) \longrightarrow (f; g) \wedge f \text{ yields } (\text{fin } (\text{init } w))$   
 by auto  
 have 3:  $\vdash (f; g) \wedge f \text{ yields } (\text{fin } (\text{init } w)) \longrightarrow f; (g \wedge \text{fin } (\text{init } w))$   
 by (rule ChopAndYieldsImp)  
 have 4:  $\vdash (f; g) \wedge \text{fin } (\text{init } w) \longrightarrow f; (g \wedge \text{fin } (\text{init } w))$   
 using 2 3 by fastforce  
 have 11:  $\vdash f; (g \wedge \text{fin } (\text{init } w)) \longrightarrow f; g$   
 by (rule ChopAndA)  
 have 12:  $\vdash f; (g \wedge \text{fin } (\text{init } w)) \longrightarrow f; \text{fin } (\text{init } w)$   
 by (rule ChopAndB)  
 have 13:  $\vdash f; \text{fin } (\text{init } w) \longrightarrow \Diamond (\text{fin } (\text{init } w))$   
 by (rule ChopImpDiamond)  
 have 14:  $\vdash \Diamond (\text{fin } (\text{init } w)) \longrightarrow \text{fin } (\text{init } w)$   
 by (rule DiamondFinImpFin)  
 have 15:  $\vdash f; (g \wedge \text{fin } (\text{init } w)) \longrightarrow (f; g) \wedge \text{fin } (\text{init } w)$   
 using 11 12 13 14 by fastforce  
 from 4 15 show ?thesis by fastforce  
 qed

lemma ChopAndNotFin:

$\vdash (f; g \wedge \neg (\text{fin } (\text{init } w))) = f; (g \wedge \neg (\text{fin } (\text{init } w)))$   
 proof –  
 have 1:  $\vdash (f; g \wedge \text{fin } (\text{init } (\neg w))) = f; (g \wedge \text{fin } (\text{init } (\neg w)))$   
 by (rule ChopAndFin)  
 have 2:  $\vdash \text{fin } (\text{init } (\neg w)) = (\neg (\text{fin } (\text{init } w)))$   
 using FinNotStateEqvNotFinState by blast  
 hence 3:  $\vdash (g \wedge \text{fin } (\text{init } (\neg w))) = (g \wedge \neg (\text{fin } (\text{init } w)))$   
 by auto  
 hence 4:  $\vdash f; (g \wedge \text{fin } (\text{init } (\neg w))) = f; (g \wedge \neg (\text{fin } (\text{init } w)))$   
 by (rule RightChopEqvChop)  
 from 1 2 4 show ?thesis by fastforce  
 qed

lemma FinChopChain:

$\vdash ((\text{init } w) \longrightarrow \text{fin } (\text{init } w1)); ((\text{init } w1) \longrightarrow \text{fin } (\text{init } w2))$   
 $\longrightarrow ((\text{init } w) \longrightarrow \text{fin } (\text{init } w2))$   
 proof –  
 have 1:  $\vdash (\text{init } w) \wedge ((\text{init } w) \longrightarrow \text{fin } (\text{init } w1)); ((\text{init } w1) \longrightarrow \text{fin } (\text{init } w2))$   
 $\longrightarrow$   
 $(\text{init } w) \wedge ((\text{init } w) \longrightarrow \text{fin } (\text{init } w1)); ((\text{init } w1) \longrightarrow \text{fin } (\text{init } w2))$

**by** (*rule StateAndChopImport*)  
**have** 2:  $\vdash (\text{init } w) \wedge ((\text{init } w) \longrightarrow \text{fin } (\text{init } w1)) \longrightarrow \text{fin } (\text{init } w1)$   
**by** *auto*  
**have** 3:  $\vdash ((\text{init } w) \wedge ((\text{init } w) \longrightarrow \text{fin } (\text{init } w1))); ((\text{init } w1) \longrightarrow \text{fin } (\text{init } w2))$   
 $\longrightarrow$   
 $(\text{fin } (\text{init } w1)); ((\text{init } w1) \longrightarrow \text{fin } (\text{init } w2))$   
**using** 2 **by** (*rule LeftChopImpChop*)  
**have** 4:  $\vdash (\text{fin } (\text{init } w1)); ((\text{init } w1) \longrightarrow \text{fin } (\text{init } w2)) =$   
 $\diamond((\text{init } w1) \wedge ((\text{init } w1) \longrightarrow \text{fin } (\text{init } w2)))$   
**by** (*rule FinChopEqvDiamond*)  
**have** 41:  $\vdash ((\text{init } w1) \wedge ((\text{init } w1) \longrightarrow \text{fin } (\text{init } w2))) \longrightarrow \text{fin } (\text{init } w2)$   
**by** *auto*  
**have** 42:  $\vdash \diamond((\text{init } w1) \wedge ((\text{init } w1) \longrightarrow \text{fin } (\text{init } w2))) \longrightarrow \diamond(\text{fin } (\text{init } w2))$   
**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:  $\vdash (\text{init } w) \wedge ((\text{init } w) \longrightarrow \text{fin } (\text{init } w1)); ((\text{init } w1) \longrightarrow \text{fin } (\text{init } w2))$   
 $\longrightarrow \text{fin } (\text{init } w2)$   
**using** 1 3 4 5 42 **by** *fastforce*  
**from** 6 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *ChopRule*:

**assumes**  $\vdash (\text{init } w) \wedge f \longrightarrow \text{fin } (\text{init } w1)$   
 $\vdash (\text{init } w1) \wedge f1 \longrightarrow \text{fin } (\text{init } w2)$   
**shows**  $\vdash (\text{init } w) \wedge (f; f1) \longrightarrow \text{fin } (\text{init } w2)$   
**proof** –  
**have** 1:  $\vdash (\text{init } w) \wedge (f; f1) \longrightarrow ((\text{init } w) \wedge f); f1$  **by** (*rule StateAndChopImport*)  
**have** 2:  $\vdash (\text{init } w) \wedge f \longrightarrow \text{fin } (\text{init } w1)$  **using** *assms* **by** *auto*  
**hence** 3:  $\vdash ((\text{init } w) \wedge f); f1 \longrightarrow (\text{fin } (\text{init } w1)); f1$  **by** (*rule LeftChopImpChop*)  
**have** 4:  $\vdash (\text{fin } (\text{init } w1)); f1 = \diamond((\text{init } w1) \wedge f1)$  **by** (*rule FinChopEqvDiamond*)  
**have** 5:  $\vdash (\text{init } w1) \wedge f1 \longrightarrow \text{fin } (\text{init } w2)$  **using** *assms* **by** *auto*  
**hence** 6:  $\vdash \diamond((\text{init } w1) \wedge f1) \longrightarrow \diamond(\text{fin } (\text{init } w2))$  **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**  $\vdash (\text{init } w) \wedge f \longrightarrow f1 \wedge \text{fin } (\text{init } w1)$   
 $\vdash (\text{init } w1) \wedge g \longrightarrow g1$   
**shows**  $\vdash (\text{init } w) \wedge (f; g) \longrightarrow (f1; g1)$   
**proof** –  
**have** 1:  $\vdash (\text{init } w) \wedge f \longrightarrow f1 \wedge \text{fin } (\text{init } w1)$  **using** *assms* **by** *auto*  
**hence** 2:  $\vdash (\text{init } w) \wedge (f; g) \longrightarrow (f1 \wedge \text{fin } (\text{init } w1)); g$  **by** (*rule StateAndChopImpChopRule*)  
**have** 3:  $\vdash (f1 \wedge \text{fin } (\text{init } w1)); g = f1; ((\text{init } w1) \wedge g)$  **by** (*rule AndFinChopEqvStateAndChop*)  
**have** 4:  $\vdash (\text{init } w1) \wedge g \longrightarrow g1$  **using** *assms* **by** *auto*  
**hence** 5:  $\vdash f1; ((\text{init } w1) \wedge g) \longrightarrow f1; g1$  **by** (*rule RightChopImpChop*)  
**from** 2 3 5 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *ChopRepAndFin*:  
**assumes**  $\vdash (\text{init } w) \wedge f \longrightarrow f1 \wedge \text{fin } (\text{init } w1)$   
 $\vdash (\text{init } w1) \wedge g \longrightarrow g1 \wedge \text{fin } (\text{init } w2)$   
**shows**  $\vdash (\text{init } w) \wedge (f; g) \longrightarrow (f1; g1) \wedge \text{fin } (\text{init } w2)$   
**proof** –  
**have** 1:  $\vdash (\text{init } w) \wedge f \longrightarrow f1 \wedge \text{fin } (\text{init } w1)$  **using** *assms* **by** *auto*  
**have** 2:  $\vdash (\text{init } w1) \wedge g \longrightarrow g1 \wedge \text{fin } (\text{init } w2)$  **using** *assms* **by** *auto*  
**have** 3:  $\vdash (\text{init } w) \wedge (f; g) \longrightarrow f1; (g1 \wedge \text{fin } (\text{init } w2))$  **using** 1 2 **by** (rule *ChopRep*)  
**have** 4:  $\vdash f1; (g1 \wedge \text{fin } (\text{init } w2)) \longrightarrow f1; g1$  **by** (rule *ChopAndA*)  
**have** 5:  $\vdash f1; (g1 \wedge \text{fin } (\text{init } w2)) \longrightarrow f1; \text{fin } (\text{init } w2)$  **by** (rule *ChopAndB*)  
**have** 6:  $\vdash f1; \text{fin } (\text{init } w2) \longrightarrow \text{fin } (\text{init } w2)$  **by** (rule *ChopFinImpFin*)  
**show** ?thesis **by** (meson 1 2 3 4 *ChopRule Prop12 lift-imp-trans*)  
**qed**

**lemma** *TrueChopMoreEqvMore*:  
 $\vdash \# \text{True} ; \text{more} = \text{more}$   
**by** (metis *ChopMoreImpMore NowImpDiamond TrueChopEqvDiamond int-eq int-iffI*)

**lemma** *MoreChopLoop*:  
**assumes**  $\vdash f \longrightarrow \text{more} ; f$   
**shows**  $\vdash \neg f$   
**proof** –  
**have** 1:  $\vdash f \longrightarrow \text{more} ; f$   
**using** *assms* **by** *auto*  
**hence** 11:  $\vdash \Diamond f \longrightarrow \Diamond (\text{more};f)$   
**by** (rule *DiamondImpDiamond*)  
**have** 12:  $\vdash \Diamond (\text{more};f) = \# \text{True};(\text{more};f)$   
**by** (simp add: *sometimes-d-def*)  
**have** 13:  $\vdash \# \text{True};(\text{more};f) = (\# \text{True};\text{more});f$   
**by** (rule *ChopAssoc*)  
**have** 14:  $\vdash \Diamond (\text{more};f) = \text{more};f$   
**using** *TrueChopMoreEqvMore* 12 13 **by** (metis *int-eq*)  
**have** 2:  $\vdash \text{more} ; f = \bigcirc(\Diamond f)$   
**by** (rule *MoreChopEqvNextDiamond*)  
**have** 3:  $\vdash \Diamond f \longrightarrow \bigcirc(\Diamond f)$   
**using** 11 14 2 **by** *fastforce*  
**hence** 4:  $\vdash \neg(\Diamond f)$   
**by** (rule *NextLoop*)  
**have** 5:  $\vdash \neg(\Diamond f) \longrightarrow \neg f$   
**using** *NowImpDiamond* **by** *fastforce*  
**from** 4 5 **show** ?thesis **using** *MP* **by** *blast*  
**qed**

**lemma** *MoreChopContra*:  
**assumes**  $\vdash f \wedge \neg g \longrightarrow (\text{more} ; (f \wedge \neg g))$   
**shows**  $\vdash f \longrightarrow g$   
**proof** –  
**have** 1:  $\vdash f \wedge \neg g \longrightarrow (\text{more} ; (f \wedge \neg g))$  **using** *assms* **by** *auto*

hence 2:  $\vdash \neg (f \wedge \neg g)$  **by** (rule *MoreChopLoop*)  
 from 2 **show** *?thesis* **by** *auto*  
**qed**

**lemma** *ChopLoop*:  
**assumes**  $\vdash f \longrightarrow g;f$   
 $\vdash g \longrightarrow \text{more}$   
**shows**  $\vdash \neg f$   
**proof** –  
**have** 1:  $\vdash f \longrightarrow g; f$  **using** *assms* **by** *auto*  
**have** 2:  $\vdash g \longrightarrow \text{more}$  **using** *assms* **by** *auto*  
**hence** 3:  $\vdash g; f \longrightarrow \text{more} ; f$  **by** (rule *LeftChopImpChop*)  
**have** 4:  $\vdash f \longrightarrow \text{more} ; f$  **using** 1 3 **by** *fastforce*  
**from** 4 **show** *?thesis* **using** *MoreChopLoop* **by** *auto*  
**qed**

**lemma** *ChopContra*:  
**assumes**  $\vdash f \wedge \neg g \longrightarrow h; f \wedge \neg (h; g)$   
 $\vdash h \longrightarrow \text{more}$   
**shows**  $\vdash f \longrightarrow g$   
**proof** –  
**have** 1:  $\vdash f \wedge \neg g \longrightarrow h; f \wedge \neg (h; g)$  **using** *assms* **by** *auto*  
**have** 2:  $\vdash h \longrightarrow \text{more}$  **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** *MoreChopContra* **by** *auto*  
**qed**

## 5.7 Properties of Chopstar and Chopplus

**lemma** *Chopstardef*:  
 $\vdash \text{chopstar } f = \text{powerstar } (f \wedge \text{more})$   
**by** (*simp add: chopstar-d-def*)

**lemma** *AndEmptyChopAndEmptyEqvAndEmpty*:  
 $\vdash (f \wedge \text{empty}); (f \wedge \text{empty}) = (f \wedge \text{empty})$   
**by** (*auto simp add: Valid-def itl-defs*) (*metis INil-ilen*)

**lemma** *PowerCommute*:  
 $\vdash f ; \text{power } f \ n = \text{power } f \ n ; f$   
**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 ChopAndA ChopAssoc Prop03 Prop10 Prop12 inteq-reflection lift-and-com pow-Suc*)

qed

**lemma** *ChopInductMoreL*:

**assumes**  $\vdash g \vee ((f \wedge \text{more}));h \longrightarrow h$

**shows**  $\vdash (\text{power } f \ n);g \longrightarrow h$

**proof**

(*induct n*)

**case** 0

**then show** ?case **using** *assms* **by** (*metis ChopInductL pow-0*)

**next**

**case** (*Suc n*)

**then show** ?case

**proof** –

**have** 1:  $\vdash \text{power } f \ (\text{Suc } n);g = (f;\text{power } f \ n);g$

**by** *simp*

**have** 2:  $\vdash (f;\text{power } f \ n);g = f;((\text{power } f \ n);g)$

**by** (*meson ChopAssoc Prop11*)

**have** 3:  $\vdash f;((\text{power } f \ n);g) \longrightarrow f;h$

**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** 4:  $\vdash f;h = ((f \wedge \text{more}); h \vee ((f \wedge \text{empty}); h))$

**using** 31 **by** (*simp add: OrChopEqvRule*)

**have** 5:  $\vdash ((f \wedge \text{more}); h) \longrightarrow h$  **using** *assms* **by** *auto*

**have** 6:  $\vdash ((f \wedge \text{empty}); h) \longrightarrow h$

**by** (*meson AndChopB EmptyChop Prop11 lift-imp-trans*)

**from** 5 6 4 3 2 1 **show** ?thesis **by** *fastforce*

qed

qed

**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*  
**by** (metis *AndChopA ChopAssoc PowerCommute Prop05 Prop10 inteq-reflection lift-and-com*  
*lift-imp-trans pow-Suc*)  
**qed**

**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*:  
 $\vdash f; (\exists n. \text{power } f \ n) = (\exists n. \text{power } f \ n); f$   
**proof** –  
**have** 1:  $\vdash f; (\exists n. \text{power } f \ n) =$   
 $(\exists n. f; \text{power } f \ n)$   
**using** *ChopExistPower* **by** *blast*  
**have** 2:  $\vdash (\exists n. f; \text{power } f \ n) =$   
 $(\exists n. (\text{power } f \ n); f)$   
**using** *PowerCommute* **by** *fastforce*  
**have** 3:  $\vdash (\exists n. (\text{power } f \ n); f) =$   
 $(\exists n. (\text{power } f \ n)); f$   
**using** *ExistChopPower* **by** *blast*  
**from** 1 2 3 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *PowerSucAndEmptyEqvAndEmpty*:  
 $\vdash (\text{power } (f \wedge \text{empty}) \ (Suc \ n)) = (f \wedge \text{empty})$   
**proof**  
*(induct n)*  
**case** 0  
**then show** ?case **using** *ChopEmpty*  
**by** (metis *pow-0 pow-Suc*)  
**next**  
**case** (Suc n)  
**then show** ?case  
**by** (metis *AndEmptyChopAndEmptyEqvAndEmpty inteq-reflection pow-Suc*)  
**qed**

**lemma** *PowerOr*:  
 $\vdash (\text{power } (f \vee g) \ (Suc \ n)) = ( (f; \text{power } (f \vee g) \ n) \vee$   
 $(g; \text{power } (f \vee g) \ n) )$   
**by** (simp add: *OrChopEqvRule*)



**lemma** *PowerEmptyOrMore*:

$\vdash (\text{power } (f \wedge \text{empty}) \vee (f \wedge \text{more})) (\text{Suc } n) =$   
 $((f \wedge \text{empty}); (\text{power } (f \wedge \text{empty}) \vee (f \wedge \text{more})) n) \vee$   
 $(f \wedge \text{more}); (\text{power } (f \wedge \text{empty}) \vee (f \wedge \text{more})) n)$

**using** *PowerOr* **by** *auto*

**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*:

$\vdash ((f \wedge \text{more}); \text{power } (f \wedge \text{more}) n) = \text{power } (f \wedge \text{more}) n; ((f \wedge \text{more}))$

**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** *PowerstarInductL*:

**assumes**  $\vdash g \vee f; h \longrightarrow h$

**shows**  $\vdash (\text{powerstar } f); g \longrightarrow h$

**proof** –  
**have** 1:  $\vdash (\text{powerstar } f);g = (\exists n. \text{power } f \ n);g$   
**by** (*simp add: powerstar-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** *ChopInductL assms* **by** *blast*  
**have** 4:  $\vdash (\exists n. ((\text{power } f \ n));g) \longrightarrow h$   
**using** 3 **by** (*auto simp add: Valid-def*)  
**from** 1 2 4 **show** ?thesis **by** (*metis inteq-reflection*)  
**qed**

**lemma** *PowerstarInductMoreL*:

**assumes**  $\vdash g \vee ((f \wedge \text{more}));h \longrightarrow h$

**shows**  $\vdash (\text{powerstar } f);g \longrightarrow h$

**proof** –  
**have** 1:  $\vdash (\text{powerstar } f);g = (\exists n. \text{power } f \ n);g$   
**by** (*simp add: powerstar-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** *ChopInductMoreL assms* **by** *blast*  
**have** 4:  $\vdash (\exists n. ((\text{power } f \ n));g) \longrightarrow h$   
**using** 3 **by** (*auto simp add: Valid-def*)  
**from** 1 2 4 **show** ?thesis **by** (*metis inteq-reflection*)  
**qed**

**lemma** *ChopstarInductL*:

**assumes**  $\vdash g \vee f;h \longrightarrow h$

**shows**  $\vdash (\text{chopstar } f);g \longrightarrow h$

**proof** –  
**have** 1:  $\vdash (\text{chopstar } f);g = ((\exists n. \text{power } (f \wedge \text{more}) \ n));g$   
**by** (*simp add: chopstar-d-def powerstar-d-def LeftChopEqvChop*)  
**have** 2:  $\vdash (\exists n. \text{power } (f \wedge \text{more}) \ n);g =$   
 $(\exists n. (\text{power } (f \wedge \text{more}) \ n);g)$   
**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** 4:  $\vdash (\exists n. (\text{power } (f \wedge \text{more}) \ n);g) \longrightarrow h$   
**using** 3 **by** *fastforce*  
**from** 1 2 4 **show** ?thesis **by** (*metis 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:  $\vdash (\text{chopstar } f); g = ((\exists n. \text{power } (f \wedge \text{more}) n)); g$   
**by** (*simp add: chopstar-d-def powerstar-d-def LeftChopEqvChop*)  
**have** 2:  $\vdash (\exists n. \text{power } (f \wedge \text{more}) n); g =$   
 $(\exists n. (\text{power } (f \wedge \text{more}) n); g)$   
**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** 4:  $\vdash (\exists n. (\text{power } (f \wedge \text{more}) n); g) \longrightarrow h$   
**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:  $\vdash g; (\text{powerstar } f) = g; (\exists n. \text{power } f n)$   
**by** (*simp add: powerstar-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** (*auto simp add: Valid-def*)  
**from** 1 2 4 **show** ?thesis **by** (*metis inteq-reflection*)  
**qed**

**lemma** *ChopstarInductR*:  
**assumes**  $\vdash g \vee h; f \longrightarrow h$   
**shows**  $\vdash g; (\text{chopstar } f) \longrightarrow h$   
**proof** –  
**have** 1:  $\vdash g; (\text{chopstar } f) =$   
 $g; ((\exists n. \text{power } (f \wedge \text{more}) n))$   
**by** (*simp add: chopstar-d-def powerstar-d-def*)  
**have** 2:  $\vdash (g; (\exists n. \text{power } (f \wedge \text{more}) n)) =$   
 $((\exists n. g; \text{power } (f \wedge \text{more}) n))$   
**using** *ChopExistPower LeftChopEqvChop* **by** *fastforce*  
**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 LIFT(f  $\wedge$  more)] assms* **by** *auto*  
**have** 4:  $\vdash (\exists n. g; ((\text{power } (f \wedge \text{more}) n))) \longrightarrow h$   
**using** 3 **by** (*auto simp add: Valid-def*)  
**from** 1 2 4 **show** ?thesis **by** *fastforce*  
**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:  $\vdash g;(\text{chopstar } f) = g;(\exists n. \text{power } (f \wedge \text{more}) n)$   
**by** (*simp add: chopstar-d-def powerstar-d-def*)  
**have** 2:  $\vdash (g;(\exists n. \text{power } (f \wedge \text{more}) n)) =$   
 $(\exists n. g;\text{power } (f \wedge \text{more}) n)$   
**using** *ChopExistPower LeftChopEqvChop* **by** *fastforce*  
**have** 3:  $\bigwedge n. \vdash g;(\text{power } (f \wedge \text{more}) n) \longrightarrow h$   
**using** *ChopInductR assms* **by** (*metis*)  
**have** 4:  $\vdash (\exists n. g;(\text{power } (f \wedge \text{more}) n)) \longrightarrow h$   
**using** 3 **by** (*auto simp add: Valid-def*)  
**from** 1 2 4 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *PSAndMoreImpPS*:

$\vdash \text{powerstar } (f \wedge \text{more}) \longrightarrow \text{powerstar } f$

**proof** –

**have** 2:  $\vdash \text{empty} \vee ((f \wedge \text{more}); \text{powerstar } f \longrightarrow \text{powerstar } f)$   
**using** *AndChopA PSEqvEmptyOrChopPS* **by** *fastforce*  
**have** 3:  $\vdash \text{powerstar } (f \wedge \text{more}); \text{empty} \longrightarrow \text{powerstar } f$   
**using** 2 *PowerstarInductL* **by** *blast*  
**from** 2 3 **show** ?thesis **by** (*metis ChopEmpty int-eq*)

**qed**

**lemma** *PSImpAndMorePS*:

$\vdash \text{powerstar } f \longrightarrow \text{powerstar } (f \wedge \text{more})$

**by** (*meson ChopEmpty PSEqvEmptyOrChopPS PowerstarInductMoreL int-iffD2 lift-imp-trans*)

**lemma** *FPSAndMoreEqvFPS*:

$\vdash \text{powerstar } (f \wedge \text{more}) = \text{powerstar } f$

**using** *PSAndMoreImpPS PSImpAndMorePS* **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** *PowerchopAndMore*:

$\vdash ((\text{power } (f \wedge \text{more}) (\text{Suc } n)) \wedge \text{more}) = (\text{power } (f \wedge \text{more}) (\text{Suc } n))$

**proof**

(*induct n*)

**case** 0

**then show** ?case

**by** (*metis (no-types, lifting) AndChopB DiamondEmpty MoreChopEqvNextDiamond Prop10 int-eq-true*)

```

    inteq-reflection more-d-def pow-0 pow-Suc)
  next
  case (Suc n)
  then show ?case
  by (metis Prop10 Prop11 Prop12 RightChopImpMoreRule pow-Suc)
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( $f \wedge \text{more}$ )] by auto

```

```

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 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*:

$\vdash f;f^* \longrightarrow f^*;f$

**by** (*metis ChopstarEqvPowerstar PowerStarCommute int-iffD1 inteq-reflection powerstar-d-def*)

**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*:

$\vdash f^* \wedge \text{more} \longrightarrow f^*;f$

**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)  
 have 5:  $\vdash f^* \wedge \text{more} \wedge \neg f \longrightarrow (f \wedge \text{more}); (f^* \wedge \text{more})$   
 by (rule CSMoreNotImpChopCSAndMore)  
 have 6:  $\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$   
 by (rule ChopstarEqv)  
 hence 7:  $\vdash f^*; f = (f \vee ((f \wedge \text{more}); f^*); f)$   
 by (rule EmptyOrChopEqvRule)  
 have 8:  $\vdash (f \wedge \text{more}); (f^*; f) = ((f \wedge \text{more}); f^*); f$   
 by (rule ChopAssoc)  
 have 9:  $\vdash (f^* \wedge \text{more}) \wedge \neg (f^*; f) \longrightarrow$   
 $(f \wedge \text{more}); (f^* \wedge \text{more}) \wedge \neg ((f \wedge \text{more}); (f^*; f))$   
 using 5 7 8 by fastforce  
 have 10:  $\vdash f \wedge \text{more} \longrightarrow \text{more}$   
 by auto  
 from 9 10 show ?thesis by (rule ChopContra)  
 qed

**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:

$\vdash f^*; f^* = f^*$

by (metis CSEqvOrChopCS ChopEmpty ChopstarInductL EmptyImpCS Prop02 Prop03 Prop11 RightChopImpChop  
inteq-reflection)

**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 *CSChopCSImpCS*:

$\vdash f^*; f^* \longrightarrow f^*$

proof –

have 1:  $\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$

by (rule *ChopstarEqv*)

hence 2:  $\vdash f^*; f^* = (f^* \vee ((f \wedge \text{more}); f^*); f^*)$

by (rule *EmptyOrChopEqvRule*)

have 21:  $\vdash f^*; f^* \wedge \neg (f^*) \longrightarrow ((f \wedge \text{more}); f^*); f^*$

using 2 by *auto*

have 22:  $\vdash \neg (f^*) = (\neg \text{empty} \wedge \neg ((f \wedge \text{more}); f^*))$

using 1 by *fastforce*

have 23:  $\vdash \neg (f^*) \longrightarrow \neg ((f \wedge \text{more}); f^*)$

using 2 22 by *fastforce*

have 24:  $\vdash f^*; f^* \wedge \neg (f^*) \longrightarrow \neg (f^*)$

by *auto*

have 25:  $\vdash f^*; f^* \wedge \neg (f^*) \longrightarrow \neg ((f \wedge \text{more}); f^*)$

using 23 24 *MP* by *auto*

have 3:  $\vdash f^*; f^* \wedge \neg (f^*) \longrightarrow ((f \wedge \text{more}); f^*); f^* \wedge \neg ((f \wedge \text{more}); f^*)$

using 21 25 by *fastforce*

have 4:  $\vdash (f \wedge \text{more}); (f^*; f^*) = ((f \wedge \text{more}); f^*); f^*$

by (rule *ChopAssoc*)

have 5:  $\vdash f^*; f^* \wedge \neg (f^*) \longrightarrow (f \wedge \text{more}); (f^*; f^*) \wedge \neg ((f \wedge \text{more}); f^*)$

using 3 4 by *fastforce*

have 6:  $\vdash f \wedge \text{more} \longrightarrow \text{more}$

by *auto*

from 5 6 show ?thesis using *ChopContra* by *blast*

qed

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*)  
**have** 3:  $\vdash f^*; f^* \longrightarrow f^*$  **by** (rule *CSChopCSImpCS*)  
**from** 2 3 **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:  $\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$   
**by** (rule *ChopstarEqv*)  
**have** 2:  $\vdash \text{empty} \longrightarrow g$   
**using** assms **by** blast  
**have** 3:  $\vdash (f \wedge \text{more}); g \longrightarrow g$   
**using** assms **by** blast  
**have** 31:  $\vdash \neg g \longrightarrow \text{more}$   
**using** 2 **by** (auto simp: empty-d-def)  
**have** 32:  $\vdash \neg g \longrightarrow \neg ((f \wedge \text{more}); g)$   
**using** 3 **by** fastforce  
**have** 33:  $\vdash f^* \wedge \text{more} \longrightarrow (f \wedge \text{more}); f^*$   
**using** 1 **using** CSAndMoreEqvAndMoreChop **by** fastforce  
**have** 34:  $\vdash f^* \wedge \neg g \longrightarrow f^* \wedge \text{more}$   
**using** 31 **by** auto  
**have** 35:  $\vdash f^* \wedge \neg g \longrightarrow (f \wedge \text{more}); f^*$   
**using** 33 34 **by** fastforce  
**have** 36:  $\vdash f^* \wedge \neg g \longrightarrow \neg ((f \wedge \text{more}); g)$   
**using** 32 **by** auto  
**have** 4:  $\vdash f^* \wedge \neg g \longrightarrow (f \wedge \text{more}); f^* \wedge \neg ((f \wedge \text{more}); g)$   
**using** 35 36 **by** fastforce  
**have** 5:  $\vdash f \wedge \text{more} \longrightarrow \text{more}$

```

    by auto
  from 4 5 show ?thesis using ChopContra by blast
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:  $\vdash \text{empty} \longrightarrow f^*$  by (rule EmptyImpCS)
  have 2:  $\vdash (f^* \wedge \text{more}); f^* \longrightarrow f^*; f^*$  by (rule AndChopA)
  have 3:  $\vdash f^*; f^* \longrightarrow f^*$  by (rule CSChopCSImpCS)
  have 4:  $\vdash (f^* \wedge \text{more}); f^* \longrightarrow f^*$  using 2 3 lift-imp-trans by blast
  from 1 4 show ?thesis using CSElim 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 (no-types, lifting) *ChopAndEmptyEqvEmptyChopEmpty ChopEmpty ChopImpDiamond DiamondFin*  
*Finprop(1) NotEmptyEqvMore Prop12 always-d-def empty-d-def fin-d-def int-simps(14) int-simps(2)*  
*int-simps(21) inteq-reflection sometimes-d-def*)

**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 ∧ more); empty ∨ (f ∧ more); (f;f*))
    using ChopOrEqv by blast
  have 7: ⊢ (f ∧ more); empty = (f ∧ more)
    using ChopEmpty by blast
  have 8: ⊢ (empty ∨ f;f*) =
    (empty ∨ (f ∧ more) ∨ (f ∧ more); (f;f*))
    using 5 6 7 by (metis 2 3 inteq-reflection)
  have 9: ⊢ ((empty ∨ f;f*) ∧ more) = (f;f* ∧ more)
    by (auto simp: empty-d-def)
  have 10: ⊢ ((empty ∨ (f ∧ more) ∨ (f ∧ more); (f;f*)) ∧ more) =
    (((f ∧ more) ∨ (f ∧ more); (f;f*)) ∧ more)
    by (auto simp: empty-d-def)
  have 12: ⊢ (f;f* ∧ more) = ((f ∧ more) ∨ (f ∧ more); (f;f*))
    by (metis (no-types, hide-lams) 10 3 4 9 ChopEmpty ChopOrEqv LeftChopImpMoreRule
      Prop11 Prop12 inteq-reflection)
  have 13: ⊢ (f;f* ∧ empty) = (f ∧ empty)
    by (rule ChopCSAndEmptyEqvAndEmpty)
  have 14: ⊢ ((f ∧ more) ∨ (f ∧ more); (f;f*) ∨ (f ∧ empty)) =
    (f ∨ (f ∧ more); (f;f*))
    by (auto simp: empty-d-def)
  have 15: ⊢ f;f* = ((f;f* ∧ empty) ∨ (f;f* ∧ more))
    by (auto simp: empty-d-def)
  from 12 13 14 15 show ?thesis by fastforce
qed

```

**lemma** *ChopPlusImpChopPlus*:

```

  assumes ⊢ f ⟶ g
  shows ⊢ f;f* ⟶ g;g*
proof -
  have 1: ⊢ f ⟶ g
    using assms by auto
  have 2: ⊢ f;f* = (f ∨ (f ∧ more); (f;f*))
    by (rule ChopPlusEqv)
  have 3: ⊢ g;g* = (g ∨ (g ∧ more); (g;g*))
    by (rule ChopPlusEqv)
  have 4: ⊢ f;f* ∧ ¬ (g;g*) ⟶ ((f ∧ more); (f;f*)) ∧ ¬ ((g ∧ more); (g;g*))
    using 1 2 3 by fastforce
  have 5: ⊢ f ∧ more ⟶ g ∧ more using 1
    by auto
  have 6: ⊢ (f ∧ more); (f;f*) ⟶ (g ∧ more); (f;f*)
    using 5 by (rule LeftChopImpChop)
  have 7: ⊢ f;f* ∧ ¬ (g;g*) ⟶
    ((g ∧ more); (f;f*)) ∧ ¬ ((g ∧ more); (g;g*))
    using 4 6 by fastforce
  have 8: ⊢ g ∧ more ⟶ more
    by auto
  from 7 8 show ?thesis using ChopContra by blast
qed

```

**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**  $\vdash f \wedge \neg g \longrightarrow (g \wedge \text{more}); f$

**shows**  $\vdash f \longrightarrow g;g^*$

**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** ChopContra **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:  $\vdash f;f^* = (f \vee (f \wedge \text{more}); (f;f^*))$  **by** (rule ChopPlusEqv)

**have** 2:  $\vdash f \longrightarrow g$  **using** assms **by** blast

**hence** 21:  $\vdash \neg g \longrightarrow \neg f$  **by** auto

**have** 3:  $\vdash (f \wedge \text{more}); g \longrightarrow g$  **using** assms **by** blast

**hence** 31:  $\vdash \neg g \longrightarrow \neg ((f \wedge \text{more}); g)$  **by** fastforce

**hence** 32:  $\vdash f;f^* \wedge \neg g \longrightarrow \neg ((f \wedge \text{more}); g)$  **by** auto

**have** 33:  $\vdash f;f^* \wedge \neg g \longrightarrow (f \wedge \text{more}); (f;f^*)$  **using** 1 21 **by** fastforce

**have** 4:  $\vdash f;f^* \wedge \neg g \longrightarrow$

$(f \wedge \text{more}); (f;f^*) \wedge \neg ((f \wedge \text{more}); g)$  **using** 31 33 **by** fastforce

**have** 5:  $\vdash f \wedge \text{more} \longrightarrow \text{more}$  **by** auto

**from** 4 5 **show** ?thesis **using** ChopContra **by** blast

**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**  $\vdash f \longrightarrow g^*$   
**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** *ChopContra* **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**  $\vdash f \longrightarrow g^*; h$

**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** *ChopContra* **by** *blast*  
**qed**

**lemma** *DiamondAndEmptyEqvAndEmpty*:

$\vdash (\Diamond f \wedge \text{empty}) = (f \wedge \text{empty})$   
**by** (auto simp: *itl-defs*)

**lemma** *InitAndEmptyEqvAndEmpty*:

$\vdash ((\text{init } w) \wedge \text{empty}) = (w \wedge \text{empty})$

**proof** –

**have** 1:  $\vdash ((\text{init } w) \wedge \text{empty}) = ((w \wedge \text{empty}); \# \text{True} \wedge \text{empty})$



by (metis init-d-def int-eq lift-and-com)  
 have 2:  $\vdash ((w \wedge \text{empty}); \# \text{True} \wedge \text{empty}) = (w \wedge \text{empty}); (\# \text{True} \wedge \text{empty})$   
 by (meson AndChopA ChopAndA ChopAndEmptyEqvEmptyChopEmpty lift-imp-trans Prop11 Prop12)  
 have 3:  $\vdash (w \wedge \text{empty}); (\# \text{True} \wedge \text{empty}) = (w \wedge \text{empty}); \text{empty}$   
 using RightChopEqvChop by fastforce  
 have 4:  $\vdash (w \wedge \text{empty}); \text{empty} = (w \wedge \text{empty})$   
 using ChopEmpty by blast  
 from 1 2 3 4 show ?thesis by fastforce  
 qed

**lemma** *InitAndNotBoxInitImpNotEmpty*:

$\vdash \text{init } w \wedge \neg(\Box(\text{init } w)) \longrightarrow \neg \text{empty}$

**proof** –

have 1:  $\vdash ((\text{init } w) \wedge \text{empty}) = (w \wedge \text{empty})$   
 by (rule InitAndEmptyEqvAndEmpty)  
 have 2:  $\vdash (\neg(\Box(\text{init } w)) \wedge \text{empty}) = (\Diamond(\neg(\text{init } w)) \wedge \text{empty})$   
 by (auto simp: always-d-def)  
 have 3:  $\vdash (\Diamond(\neg(\text{init } w)) \wedge \text{empty}) = (\neg(\text{init } w) \wedge \text{empty})$   
 by (simp add: DiamondAndEmptyEqvAndEmpty)  
 have 4:  $\vdash (\neg(\text{init } w)) = (\text{init }(\neg w))$  using Initprop(2) by blast  
 have 5:  $\vdash (\neg(\text{init } w) \wedge \text{empty}) = (\neg w \wedge \text{empty})$   
 using 4 InitAndEmptyEqvAndEmpty by (metis inteq-reflection)  
 have 6:  $\vdash (\neg(\Box(\text{init } w)) \wedge \text{empty}) = (\neg w \wedge \text{empty})$   
 using 2 3 5 by fastforce  
 have 7:  $\vdash \neg(\text{init } w \wedge \neg(\Box(\text{init } w)) \wedge \text{empty})$   
 using 1 6 by fastforce  
 from 7 show ?thesis by auto  
 qed

**lemma** *BoxImpTrueChopAndEmpty*:

$\vdash \Box f \longrightarrow \# \text{True}; (f \wedge \text{empty})$

using BoxAndChopImport Finprop(3) by fastforce

**lemma** *BoxInitAndMoreImpBoxInitAndMoreAndFinInit*:

$\vdash \Box(\text{init } w) \wedge \text{more} \longrightarrow (\Box(\text{init } w) \wedge \text{more}) \wedge \text{fin }(\text{init } w)$

**proof** –

have 1:  $\vdash \text{fin }(\text{init } w) = \# \text{True}; (\text{init } w \wedge \text{empty})$  using FinEqvTrueChopAndEmpty by blast  
 have 2:  $\vdash \Box(\text{init } w) \longrightarrow \# \text{True}; (\text{init } w \wedge \text{empty})$  by (rule BoxImpTrueChopAndEmpty)  
 from 1 2 show ?thesis by fastforce  
 qed

**lemma** *CSImpBox*:

**assumes**  $\vdash f \longrightarrow \text{empty} \vee (\Box(\text{init } w) \wedge \text{more}); f$

**shows**  $\vdash \text{init } w \wedge f \longrightarrow \Box(\text{init } w)$

**proof** –

have 1:  $\vdash f \longrightarrow \text{empty} \vee (\Box(\text{init } w) \wedge \text{more}); f$   
 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:  $\vdash \text{init } w \wedge f \wedge \neg(\Box(\text{init } w)) \longrightarrow (\Box(\text{init } w) \wedge \text{more}); f$

**using** 1 2 **by** *fastforce*  
**have** 4:  $\vdash \Box (init\ w) \wedge more \longrightarrow (\Box (init\ w) \wedge more) \wedge fin\ (init\ w)$   
**by** (*rule BoxInitAndMoreImpBoxInitAndMoreAndFinInit*)  
**hence** 5:  $\vdash (\Box (init\ w) \wedge more); f \longrightarrow ((\Box (init\ w) \wedge more) \wedge fin\ (init\ w)); f$   
**by** (*rule LeftChopImpChop*)  
**have** 6:  $\vdash ((\Box (init\ w) \wedge more) \wedge fin\ (init\ w)); f =$   
 $(\Box (init\ w) \wedge more); (init\ w \wedge f)$   
**by** (*rule AndFinChopEqvStateAndChop*)  
**have** 7:  $\vdash \neg(\Box (init\ w)) \longrightarrow (\Box (init\ w))\ yields\ (\neg(\Box (init\ w)))$   
**by** (*rule NotBoxStateImpBoxYieldsNotBox*)  
**have** 8:  $\vdash (\Box (init\ w))\ yields\ (\neg(\Box (init\ w))) \longrightarrow$   
 $(\Box (init\ w) \wedge more)\ yields\ (\neg(\Box (init\ w)))$   
**by** (*rule AndYieldsA*)  
**have** 9:  $\vdash (\Box (init\ w) \wedge more); (init\ w \wedge f) \wedge (\Box (init\ w) \wedge more)\ yields\ (\neg(\Box (init\ w)))$   
 $\longrightarrow$   
 $(\Box (init\ w) \wedge more); ((init\ w \wedge f) \wedge \neg(\Box (init\ w)))$   
**by** (*rule ChopAndYieldsImp*)  
**have** 10:  $\vdash (init\ w \wedge f) \wedge \neg(\Box (init\ w)) \longrightarrow$   
 $(\Box (init\ w) \wedge more); ((init\ w \wedge f) \wedge \neg(\Box (init\ w)))$   
**using** 3 5 6 7 8 9 **by** *fastforce*  
**have** 11:  $\vdash (\Box (init\ w) \wedge more); ((init\ w \wedge f) \wedge \neg(\Box (init\ w))) \longrightarrow$   
 $more; ((init\ w \wedge f) \wedge \neg(\Box (init\ w)))$   
**by** (*rule AndChopB*)  
**have** 12:  $\vdash (init\ w \wedge f) \wedge \neg(\Box (init\ w)) \longrightarrow$   
 $more; ((init\ w \wedge f) \wedge \neg(\Box (init\ w)))$   
**using** 10 11 **by** *fastforce*  
**from** 12 **show** ?thesis **using** *MoreChopContra* **by** *blast*  
**qed**

**lemma** *BoxCSEqvBox*:

$\vdash (init\ w \wedge (\Box (init\ w))^*) = \Box (init\ w)$

**proof** –

**have** 1:  $\vdash (\Box (init\ w))^* = (empty \vee (\Box (init\ w) \wedge more); (\Box (init\ w))^*)$   
**by** (*rule ChopstarEqv*)  
**hence** 2:  $\vdash (\Box (init\ w))^* \longrightarrow empty \vee (\Box (init\ w) \wedge more); (\Box (init\ w))^*$   
**by** *fastforce*  
**hence** 3:  $\vdash init\ w \wedge (\Box (init\ w))^* \longrightarrow \Box (init\ w)$   
**by** (*rule CSImpBox*)  
**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** 3 13 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *BoxStateAndCSEqvCS*:

$\vdash (\Box (init\ w) \wedge f^*) = (init\ w \wedge (\Box (init\ w) \wedge f))^*$

**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:  $\vdash \Box (init\ w) \wedge f^* \longrightarrow (\Box (init\ w) \wedge f)^*$   
**by** (*rule CSIntro*)  
**have** 71:  $\vdash init\ w \wedge \Box (init\ w) \wedge f^* \longrightarrow init\ w \wedge (\Box (init\ w) \wedge f)^*$   
**using** 7 **by** *fastforce*  
**have** 8:  $\vdash \Box (init\ w) \wedge f^* \longrightarrow init\ w \wedge (\Box (init\ w) \wedge f)^*$   
**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*:

$\vdash ba\ (f \longrightarrow g) \longrightarrow f^* \longrightarrow g^*$

**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:  $\vdash ((\text{empty} \vee (f \wedge \text{more}); f^*) \wedge \text{more}) = ((f \wedge \text{more}); f^* \wedge \text{more})$   
**by** (*auto simp: empty-d-def*)  
**have** 32:  $\vdash f^* \wedge \neg (g^*) \longrightarrow (f \wedge \text{more}); f^* \wedge \neg ((g \wedge \text{more}); g^*)$   
**using** 3 31 **by** *fastforce*  
**have** 4:  $\vdash (f \longrightarrow g) \longrightarrow (f \wedge \text{more} \longrightarrow g \wedge \text{more})$   
**by** *auto*  
**hence** 5:  $\vdash \text{ba } (f \longrightarrow g) \longrightarrow \text{ba } (f \wedge \text{more} \longrightarrow g \wedge \text{more})$   
**by** (*rule BaImpBa*)  
**have** 6:  $\vdash \text{ba } (f \wedge \text{more} \longrightarrow g \wedge \text{more}) \longrightarrow$   
 $(f \wedge \text{more}); f^* \longrightarrow (g \wedge \text{more}); f^*$   
**by** (*rule BaLeftChopImpChop*)  
**have** 7:  $\vdash \text{ba } (f \longrightarrow g) \wedge (f \wedge \text{more}); f^* \longrightarrow (g \wedge \text{more}); f^*$   
**using** 5 6 **by** *fastforce*  
**have** 8:  $\vdash (g \wedge \text{more}); f^* \wedge \neg ((g \wedge \text{more}); g^*)$   
 $\longrightarrow (g \wedge \text{more}); (f^* \wedge \neg (g^*))$   
**by** (*rule ChopAndNotChopImp*)  
**have** 9:  $\vdash (g \wedge \text{more}); (f^* \wedge \neg (g^*)) \longrightarrow \text{more}; (f^* \wedge \neg (g^*))$   
**by** (*rule AndChopB*)  
**have** 10:  $\vdash \text{ba } (f \longrightarrow g) \longrightarrow \text{more}; (f^* \wedge \neg (g^*)) \longrightarrow$   
 $\text{more}; (\text{ba } (f \longrightarrow g) \wedge f^* \wedge \neg (g^*))$   
**by** (*rule BaChopImpChopBa*)  
**have** 11:  $\vdash \text{ba } (f \longrightarrow g) \wedge f^* \wedge \neg (g^*) \longrightarrow$   
 $\text{more}; (\text{ba } (f \longrightarrow g) \wedge f^* \wedge \neg (g^*))$   
**using** 32 7 8 9 10 **by** *fastforce*  
**hence** 12:  $\vdash \neg ((\text{ba } (f \longrightarrow g)) \wedge (f^*) \wedge (\neg (g^*)))$   
**using** *MoreChopLoop* **by** *blast*  
**from** 12 **show** *?thesis* **using** *MP* **by** *fastforce*  
**qed**

**lemma** *BaCSEqvCS*:

$\vdash \text{ba } (f = g) \longrightarrow (f^* = g^*)$

**proof** –

**have** 1:  $\vdash \text{ba } (f = g) = (\text{ba } (f \longrightarrow g) \wedge \text{ba } (g \longrightarrow f))$  **by** (*auto simp: itl-defs*)  
**have** 2:  $\vdash \text{ba } (f \longrightarrow g) \longrightarrow (f^* \longrightarrow g^*)$  **by** (*rule BaCSImpCS*)  
**have** 3:  $\vdash \text{ba } (g \longrightarrow f) \longrightarrow (g^* \longrightarrow f^*)$  **by** (*rule BaCSImpCS*)  
**have** 4:  $\vdash \text{ba } (f = g) \longrightarrow (f^* \longrightarrow g^*) \wedge (g^* \longrightarrow f^*)$  **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*:

$\vdash \text{ba } f \wedge g^* \longrightarrow (f \wedge g)^*$

**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:  $\vdash \text{ba } (g \longrightarrow f \wedge g) \longrightarrow g^* \longrightarrow (f \wedge g)^*$  **by** (*rule BaCSImpCS*)  
**from** 2 3 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *CSSkip*:

$\vdash \text{skip}^*$

by (metis ChopPlusImpCS EmptyImpCS EmptyNextInducta next-d-def)

## 5.8 Properties of While

**lemma** *WhileEqvIf*:

$\vdash \text{while } (\text{init } w) \text{ do } f = \text{if}_i (\text{init } w) \text{ then } (f; (\text{while } (\text{init } w) \text{ do } f)) \text{ else } \text{empty}$

**proof** –

**have** 1:  $\vdash \text{while } (\text{init } w) \text{ do } f = (((\text{init } w) \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)))$

**by** (simp add: while-d-def)

**have** 2:  $\vdash (\text{init } w \wedge f)^* = (\text{empty} \vee ((\text{init } w \wedge f); (\text{init } w \wedge f)^*))$

**by** (rule CSeqvOrChopCS)

**have** 21:  $\vdash (((\text{init } w) \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) =$   
 $((\text{empty} \vee ((\text{init } w \wedge f); (\text{init } w \wedge f)^*)) \wedge \text{fin } (\neg (\text{init } w)))$

**using** 2 **by** fastforce

**have** 22:  $\vdash ((\text{empty} \vee ((\text{init } w \wedge f); (\text{init } w \wedge f)^*)) \wedge \text{fin } (\neg (\text{init } w))) =$   
 $((\text{empty} \wedge \text{fin } (\neg (\text{init } w))) \vee ((\text{init } w \wedge f); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))))$

**by** auto

**have** 3:  $\vdash (\text{empty} \wedge \text{fin } (\neg (\text{init } w))) = (\neg (\text{init } w) \wedge \text{empty})$

**using** AndFinEqvChopAndEmpty EmptyChop **by** (metis int-eq)

**have** 4:  $\vdash (\text{init } w \wedge f); (\text{init } w \wedge f)^* = (\text{init } w \wedge (f; (\text{init } w \wedge f)^*))$

**by** (rule StateAndChop)

**have** 41:  $\vdash (((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w))) =$   
 $(\text{init } w \wedge (f; (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)))$

**using** 4 **by** auto

**have** 42:  $\vdash (\text{init } w \wedge (f; (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w))) =$   
 $(\text{init } w \wedge (f; (\text{init } w \wedge f)^*) \wedge \text{fin } (\text{init } (\neg w)))$

**using** Initprop(2) **by** (metis StateAndEmptyChop int-eq)

**have** 5:  $\vdash ((f; ((\text{init } w \wedge f)^*)) \wedge (\text{fin } (\text{init } (\neg w))))$   
 $= (f; ((\text{init } w \wedge f)^* \wedge (\text{fin } (\text{init } (\neg w)))))$

**by** (rule ChopAndFin)

**have** 51:  $\vdash (f; ((\text{init } w \wedge f)^* \wedge (\text{fin } (\text{init } (\neg w)))) =$   
 $(f; ((\text{init } w \wedge f)^* \wedge (\text{fin } (\neg (\text{init } w))))$

**using** Initprop(2) **by** (metis FinAndChop int-eq)

**have** 52:  $\vdash (\text{init } w \wedge (f; (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w))) =$   
 $(\text{init } w \wedge (f; ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))))$

**using** 42 5 51 **by** fastforce

**have** 6:  $\vdash (f; ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)))) = f; \text{while } (\text{init } w) \text{ do } f$

**by** (simp add: while-d-def)

**have** 61:  $\vdash (\text{init } w \wedge (f; ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)))) =$   
 $(\text{init } w \wedge (f; \text{while } (\text{init } w) \text{ do } f))$  **using** 6

**by** auto

**have** 62:  $\vdash (\text{empty} \wedge \text{fin } (\neg (\text{init } w))) \vee (((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)))$   
 $= (\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f; \text{while } (\text{init } w) \text{ do } f))$

**using** 3 41 52 61 **by** fastforce

**have** 7:  $\vdash \text{while } (\text{init } w) \text{ do } f$   
 $= ((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f; \text{while } (\text{init } w) \text{ do } f)))$

**using** 1 21 22 62

**by** (metis 3 41 42 5 51 inteq-reflection)

**have** 71:  $\vdash \text{if}_i (\text{init } w) \text{ then } (f; (\text{while } (\text{init } w) \text{ do } f)) \text{ else } \text{empty} =$

$((\neg (init\ w) \wedge empty) \vee (init\ w \wedge (f; while\ (init\ w)\ do\ f)))$   
**by** (*auto simp: ifthenelse-d-def*)  
**from** 7 71 **show** ?thesis **by** fastforce  
**qed**

**lemma** *WhileChopEqvIf*:

$\vdash (while\ (init\ w)\ do\ f); g = if_i (init\ w)\ then\ (f; ((while\ (init\ w)\ do\ f); g))\ else\ g$   
**proof** –  
**have** 1:  $\vdash while\ (init\ w)\ do\ f =$   
 $if_i (init\ w)\ then\ (f; (while\ (init\ w)\ do\ f))\ else\ empty$   
**by** (*rule WhileEqvIf*)  
**hence** 2:  $\vdash (while\ (init\ w)\ do\ f); g =$   
 $if_i (init\ w)\ then\ ((f; while\ (init\ w)\ do\ f); g)\ else\ (empty; g)$   
**by** (*rule IfChopEqvRule*)  
**have** 3:  $\vdash empty; g = g$   
**by** (*rule EmptyChop*)  
**have** 4:  $\vdash if_i (init\ w)\ then\ ((f; while\ (init\ w)\ do\ f); g)\ else\ (empty; g) =$   
 $if_i (init\ w)\ then\ ((f; while\ (init\ w)\ do\ f); g)\ else\ g$   
**using** 3 **using** *inteq-reflection* **by** fastforce  
**have** 5:  $\vdash ((f; while\ (init\ w)\ do\ f); g) = (f; (while\ (init\ w)\ do\ f; g))$   
**by** (*rule ChopAssocB*)  
**have** 6:  $\vdash if_i (init\ w)\ then\ ((f; while\ (init\ w)\ do\ f); g)\ else\ g =$   
 $if_i (init\ w)\ then\ (f; ((while\ (init\ w)\ do\ f); g))\ else\ g$   
**using** 5 **using** *inteq-reflection* **by** fastforce  
**from** 1 2 4 6 **show** ?thesis **by** fastforce  
**qed**

**lemma** *WhileChopEqvIfRule*:

**assumes**  $\vdash f = (while\ (init\ w)\ do\ g); h$   
**shows**  $\vdash f = if_i (init\ w)\ then\ (g; f)\ else\ h$   
**proof** –  
**have** 1:  $\vdash f = (while\ (init\ w)\ do\ g); h$   
**using** *assms* **by** auto  
**have** 2:  $\vdash (while\ (init\ w)\ do\ g); h =$   
 $if_i (init\ w)\ then\ (g; ((while\ (init\ w)\ do\ g); h))\ else\ h$   
**by** (*rule WhileChopEqvIf*)  
**have** 3:  $\vdash (g; f) = (g; ((while\ (init\ w)\ do\ g); h))$   
**using** 1 **by** (*rule RightChopEqvChop*)  
**have** 4:  $\vdash (g; ((while\ (init\ w)\ do\ g); h)) = (g; f)$   
**using** 3 **by** auto  
**have** 5:  $\vdash if_i (init\ w)\ then\ (g; ((while\ (init\ w)\ do\ g); h))\ else\ h =$   
 $if_i (init\ w)\ then\ (g; f)\ else\ h$   
**using** 4 **using** *inteq-reflection* **by** fastforce  
**from** 1 2 5 **show** ?thesis **by** fastforce  
**qed**

**lemma** *WhileImpFin*:

$\vdash while\ (init\ w)\ do\ f \longrightarrow fin\ (\neg (init\ w))$   
**proof** –  
**have** 1:  $\vdash (init\ w \wedge f)^* \wedge fin\ (\neg (init\ w)) \longrightarrow fin\ (\neg (init\ w))$  **by** auto

from 1 show ?thesis by (simp add: while-d-def)  
qed

**lemma** *WhileEqvEmptyOrChopWhile*:

$\vdash \text{while } (\text{init } w) \text{ do } f = ((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f \wedge \text{more}); \text{while } (\text{init } w) \text{ do } f))$

**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:  $\vdash ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) =$   
 $((\text{empty} \wedge \text{fin } (\neg (\text{init } w))) \vee$   
 $((\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))))$   
using 1 4 by *fastforce*

**have** 6:  $\vdash (\text{empty} \wedge \text{fin } (\neg (\text{init } w))) = (\neg (\text{init } w) \wedge \text{empty})$   
using *AndFinEqvChopAndEmpty EmptyChop* by (metis *int-eq*)

**have** 7:  $\vdash (\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^* = (\text{init } w \wedge (f \wedge \text{more}); (\text{init } w \wedge f)^*)$   
by (rule *StateAndChop*)

**have** 71:  $\vdash ((\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) =$   
 $((\text{init } w \wedge (f \wedge \text{more}); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)))$   
using 7 by *auto*

**have** 8:  $\vdash (((f \wedge \text{more}); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w))) =$   
 $((f \wedge \text{more}); ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))))$   
by (rule *ChopAndFin*)

**have** 81:  $\vdash \text{fin } (\neg (\text{init } w)) = \text{fin } (\neg (\text{init } w))$   
using *FinEqvFin Initprop(2)* by *fastforce*

**have** 83:  $\vdash ((\text{init } w \wedge (f \wedge \text{more}); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w))) =$   
 $(\text{init } w \wedge (f \wedge \text{more}); ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))))$   
by (metis 81 *ChopAndFin StateAndChop inteq-reflection*)

**have** 9:  $\vdash ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) =$   
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$   
 $(\text{init } w \wedge (f \wedge \text{more}); ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)))))$   
by (metis 5 6 71 83 *inteq-reflection*)

from 9 show ?thesis by (simp add: while-d-def)  
qed

**lemma** *WhileIntro*:

**assumes**  $\vdash \neg (\text{init } w) \wedge f \longrightarrow \text{empty}$

$\vdash \text{init } w \wedge f \longrightarrow (g \wedge \text{more}); f$

**shows**  $\vdash f \longrightarrow \text{while } (\text{init } w) \text{ do } g$

**proof** –

**have** 1:  $\vdash \neg (\text{init } w) \wedge f \longrightarrow \text{empty}$   
using *assms* by *blast*

**have** 2:  $\vdash \text{init } w \wedge f \longrightarrow (g \wedge \text{more}); f$   
using *assms* by *blast*

**have** 3:  $\vdash \text{while } (\text{init } w) \text{ do } g =$

$((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g))$   
**by** (rule WhileEqvEmptyOrChopWhile)  
**hence** 31:  $\vdash \neg (\text{while } (\text{init } w) \text{ do } g) =$   
 $\neg ((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g))$   
**by** fastforce  
**hence** 32:  $\vdash (f \wedge \neg (\text{while } (\text{init } w) \text{ do } g)) =$   
 $(f \wedge \neg ((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g)))$   
**by** fastforce  
**have** 33:  $\vdash (f \wedge \neg ((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g))) =$   
 $(f \wedge \neg (\neg (\text{init } w) \wedge \text{empty}) \wedge \neg (\text{init } w \wedge (g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g))$   
**by** auto  
**have** 34:  $\vdash (f \wedge \neg (\neg (\text{init } w) \wedge \text{empty}) \wedge \neg ((\text{init } w) \wedge ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g))) =$   
 $(f \wedge ((\text{init } w) \vee \text{more}) \wedge (\neg (\text{init } w) \vee \neg ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g)))$   
**by** (auto simp: empty-d-def)  
**have** 36:  $\vdash (f \wedge \neg (\text{while } (\text{init } w) \text{ do } g)) =$   
 $((f \wedge (\text{init } w) \wedge \neg ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g)) \vee$   
 $(f \wedge (\text{init } w) \wedge \neg (\text{init } w)) \vee$   
 $(f \wedge \text{more} \wedge \neg ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g)) \vee$   
 $(f \wedge \text{more} \wedge \neg (\text{init } w)))$   
**using** 32 34 **by** fastforce  
**have** 37:  $\vdash \neg (f \wedge \text{more} \wedge \neg (\text{init } w))$   
**using** 1 **by** (auto simp: empty-d-def)  
**have** 38:  $\vdash (f \wedge \text{more} \wedge \neg ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g)) \longrightarrow$   
 $((g \wedge \text{more}); f \wedge \neg ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g))$   
**using** 1 2 **by** (auto simp: empty-d-def Valid-def)  
**have** 39:  $\vdash (f \wedge (\text{init } w) \wedge \neg ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g)) \longrightarrow$   
 $((g \wedge \text{more}); f \wedge \neg ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g))$   
**using** 2 **by** auto  
**have** 40:  $\vdash ((f \wedge (\text{init } w) \wedge \neg ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g)) \vee$   
 $(f \wedge (\text{init } w) \wedge \neg (\text{init } w)) \vee$   
 $(f \wedge \text{more} \wedge \neg ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g)) \vee$   
 $(f \wedge \text{more} \wedge \neg (\text{init } w))) \longrightarrow$   
 $((g \wedge \text{more}); f \wedge \neg ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g))$   
**using** 39 38 37 38 **by** fastforce  
**have** 4:  $\vdash f \wedge \neg (\text{while } (\text{init } w) \text{ do } g) \longrightarrow$   
 $((g \wedge \text{more}); f \wedge \neg ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g))$   
**by** (meson 36 40 Prop11 lift-imp-trans)  
**have** 5:  $\vdash g \wedge \text{more} \longrightarrow \text{more}$   
**by** auto  
**from** 4 5 **show** ?thesis **using** ChopContra **by** blast  
**qed**

**lemma** WhileElim:

**assumes**  $\vdash \neg (\text{init } w) \wedge \text{empty} \longrightarrow g$   
 $\vdash \text{init } w \wedge (f \wedge \text{more}); g \longrightarrow g$

**shows**  $\vdash \text{while } (\text{init } w) \text{ do } f \longrightarrow g$

**proof** –

**have** 1:  $\vdash \text{while } (\text{init } w) \text{ do } f =$   
 $((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f \wedge \text{more}); \text{while } (\text{init } w) \text{ do } f))$   
**by** (rule WhileEqvEmptyOrChopWhile)



**hence 11:**  $\vdash ((\text{while } (init\ w) \text{ do } f) \wedge \neg g) =$   
 $((\neg(init\ w) \wedge \text{empty}) \vee (init\ w \wedge (f \wedge \text{more}); \text{while } (init\ w) \text{ do } f)) \wedge \neg g)$   
**by auto**  
**have 2:**  $\vdash \neg (init\ w) \wedge \text{empty} \longrightarrow g$   
**using assms by blast**  
**hence 21:**  $\vdash \neg g \longrightarrow \neg(\neg (init\ w) \wedge \text{empty})$   
**by auto**  
**have 22:**  $\vdash ((\neg (init\ w) \wedge \text{empty}) \vee (init\ w \wedge (f \wedge \text{more}); \text{while } (init\ w) \text{ do } f)) \wedge \neg g \longrightarrow$   
 $(init\ w \wedge (f \wedge \text{more}); \text{while } (init\ w) \text{ do } f)$   
**using 21 by auto**  
**have 23:**  $\vdash (\text{while } (init\ w) \text{ do } f) \wedge \neg g \longrightarrow$   
 $(init\ w \wedge (f \wedge \text{more}); \text{while } (init\ w) \text{ do } f) \wedge \neg g$   
**using 11 21 by fastforce**  
**have 3:**  $\vdash (init\ w) \wedge ((f \wedge \text{more}); g) \longrightarrow g$   
**using assms by blast**  
**hence 31:**  $\vdash \neg g \longrightarrow \neg((init\ w) \wedge ((f \wedge \text{more}); g))$   
**by fastforce**  
**have 32:**  $\vdash (init\ w \wedge (f \wedge \text{more}); \text{while } (init\ w) \text{ do } f) \wedge \neg g \longrightarrow$   
 $((f \wedge \text{more}); (\text{while } (init\ w) \text{ do } f)) \wedge \neg ((f \wedge \text{more}); g) \wedge \neg g$   
**using 31 by auto**  
**have 4:**  $\vdash (\text{while } (init\ w) \text{ do } f) \wedge \neg g \longrightarrow$   
 $((f \wedge \text{more}); (\text{while } (init\ w) \text{ do } f)) \wedge \neg ((f \wedge \text{more}); g)$   
**using 23 32 by fastforce**  
**have 5:**  $\vdash f \wedge \text{more} \longrightarrow \text{more}$   
**by auto**  
**from 4 5 show ?thesis using ChopContra by blast**  
**qed**

**lemma BaWhileImpWhile:**

$\vdash ba\ (f \longrightarrow g) \longrightarrow (\text{while } (init\ w) \text{ do } f) \longrightarrow (\text{while } (init\ w) \text{ do } g)$

**proof –**

**have 1:**  $\vdash (f \longrightarrow g) \longrightarrow ((init\ w \wedge f) \longrightarrow (init\ w \wedge g))$   
**by auto**  
**hence 2:**  $\vdash ba\ (f \longrightarrow g) \longrightarrow ba\ ((init\ w \wedge f) \longrightarrow (init\ w \wedge g))$   
**by (rule BaImpBa)**  
**have 3:**  $\vdash ba\ ((init\ w \wedge f) \longrightarrow (init\ w \wedge g)) \longrightarrow ((init\ w \wedge f)^* \longrightarrow (init\ w \wedge g)^*)$   
**by (rule BaCSImpCS)**  
**have 4:**  $\vdash ba\ (f \longrightarrow g) \longrightarrow ((init\ w \wedge f)^* \wedge \text{fin } (\neg (init\ w)) \longrightarrow (init\ w \wedge g)^* \wedge \text{fin } (\neg (init\ w)))$   
**using 2 3 by fastforce**  
**from 4 show ?thesis by (simp add: while-d-def)**  
**qed**

**lemma WhileImpWhile:**

**assumes**  $\vdash f \longrightarrow g$   
**shows**  $\vdash (\text{while } (init\ w) \text{ do } f) \longrightarrow (\text{while } (init\ w) \text{ do } g)$   
**proof –**  
**have 1:**  $\vdash f \longrightarrow g$   
**using assms by auto**  
**hence 2:**  $\vdash ba\ (f \longrightarrow g)$   
**by (rule BaGen)**

**have** 3:  $\vdash ba (f \longrightarrow g) \longrightarrow ( \text{while } (init\ w) \text{ do } f) \longrightarrow ( \text{while } (init\ w) \text{ do } g)$   
**by** (rule BaWhileImpWhile)  
**from** 2 3 **show** ?thesis **using** MP **by** blast  
**qed**

## 5.9 Properties of Halt

**lemma** WnextAndMoreEqvNext:

$\vdash (wnext\ f \wedge more) = \bigcirc f$   
**by** (auto simp: itl-defs)

**lemma** BoxStateAndEmptyEqvStateAndEmpty:

$\vdash (\Box(empty = (init\ w)) \wedge empty) = ((init\ w) \wedge empty)$   
**by** (auto simp: itl-defs)

**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:

$\vdash halt( \text{init } w) = if_i (init\ w) \text{ then } empty \text{ else } (\bigcirc(halt( \text{init } w)))$

**proof** –

**have** 1:  $\vdash halt( \text{init } w) = \Box(empty = (init\ w))$   
**by** (simp add: halt-d-def)  
**have** 2:  $\vdash \Box(empty = (init\ w)) =$

$((empty \wedge init\ w) \vee (\neg(init\ w) \wedge \bigcirc(\Box(empty = (init\ w)))))$   
**by** (rule *BoxEmptyEqvIStateEqvEmptyAndStateOrNotStateNext*)  
**have** 21:  $\vdash ((empty \wedge init\ w) \vee (\neg(init\ w) \wedge \bigcirc(\Box(empty = (init\ w))))) =$   
 $((init\ w \wedge empty) \vee (\neg(init\ w) \wedge \bigcirc(\Box(empty = (init\ w)))))$   
**by** *auto*  
**have** 3:  $\vdash if_i\ (init\ w)\ then\ empty\ else\ (\bigcirc(halt\ (init\ w))) =$   
 $((init\ w \wedge empty) \vee (\neg(init\ w) \wedge \bigcirc(halt\ (init\ w))))$   
**by** (simp add: *ifthenelse-d-def*)  
**show** ?thesis **by** (metis 1 2 21 3 int-eq)  
**qed**

**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:  $\vdash empty ; f = f$   
**by** (rule *EmptyChop*)  
**have** 4:  $\vdash (\bigcirc(halt\ (init\ w))); f = \bigcirc(halt\ (init\ w); f)$   
**by** (rule *NextChop*)  
**from** 2 3 4 **show** ?thesis **by** (metis inteq-reflection)  
**qed**

**lemma** *AndHaltChopImp*:

$\vdash init\ w \wedge (halt\ (init\ w); f) \longrightarrow f$   
**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:  $\vdash init\ w \wedge if_i\ (init\ w)\ then\ f\ else\ (\bigcirc(halt\ (init\ w); f)) \longrightarrow f$   
**by** (auto simp: *ifthenelse-d-def*)  
**from** 1 2 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *NotAndHaltChopImpNext*:

$\vdash \neg (init\ w) \wedge (halt\ (init\ w); f) \longrightarrow \bigcirc(halt\ (init\ w); f)$   
**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:  $\vdash \neg (init\ w) \wedge if_i\ (init\ w)\ then\ f\ else\ (\bigcirc(halt\ (init\ w); f)) \longrightarrow$   
 $\bigcirc(halt\ (init\ w); f)$   
**by** (auto simp: *ifthenelse-d-def*)  
**from** 1 2 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *NotAndHaltChopImpSkipYields*:

$\vdash \neg (init\ w) \wedge (halt\ (init\ w); f) \longrightarrow skip\ yields\ (halt\ (init\ w); f)$

**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** *TrueChopAndEmptyEqvChopAndEmpty*:  
 $\vdash ((\# \text{True}; (f \wedge \text{empty})) \wedge g) = (g; (f \wedge \text{empty}))$   
**using** *AndFinEqvChopAndEmpty FinEqvTrueChopAndEmpty* **by** (metis int-eq lift-and-com)

**lemma** *WprevEqvEmptyOrPrev*:  
 $\vdash wprev f = (\text{empty} \vee prev f)$   
**by** (auto simp: itl-defs)

**lemma** *NotChopSkipEqvMoreAndNotChopSkip*:  
 $\vdash (\neg f); \text{skip} = (\text{more} \wedge \neg(f; \text{skip}))$   
**proof** –  
**have** 1:  $\vdash wprev f = (\text{empty} \vee prev f)$  **using** *WprevEqvEmptyOrPrev* **by** auto  
**hence** 2:  $\vdash (\neg(wprev f)) = (\neg(\text{empty} \vee prev f))$  **by** auto  
**have** 3:  $\vdash \neg(wprev f) = ((\neg f); \text{skip})$  **by** (simp add: wprev-d-def prev-d-def)  
**have** 31:  $\vdash (\text{empty} \vee prev f) = (\text{empty} \vee (f; \text{skip}))$  **by** (simp add: prev-d-def)  
**have** 32:  $\vdash (\text{empty} \vee (f; \text{skip})) = (\neg \text{more} \vee \neg \neg(f; \text{skip}))$  **by** (simp add: empty-d-def)  
**have** 33:  $\vdash (\neg \text{more} \vee \neg \neg(f; \text{skip})) = (\neg(\text{more} \wedge \neg \neg(f; \text{skip})))$  **by** fastforce  
**have** 34:  $\vdash (\text{empty} \vee prev f) = (\neg(\text{more} \wedge \neg \neg(f; \text{skip})))$  **using** 31 32 33 **by** (metis int-eq)  
**have** 4:  $\vdash \neg(\text{empty} \vee prev f) = (\text{more} \wedge \neg \neg(f; \text{skip}))$  **using** 34 **by** fastforce  
**from** 2 3 4 **show** ?thesis **by** fastforce  
**qed**

**lemma** *HaltChopImpNotHaltChopNot*:  
 $\vdash \text{halt } (\text{init } w); f \longrightarrow \neg (\text{halt } (\text{init } w); (\neg f))$   
**proof** –  
**have** 1:  $\vdash \text{halt } (\text{init } w); f = \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc (\text{halt } (\text{init } w); f))$   
**by** (rule *HaltChopEqv*)  
**have** 2:  $\vdash \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc (\text{halt } (\text{init } w); f)) \longrightarrow$   
 $( ((\text{init } w) \longrightarrow f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc (\text{halt } (\text{init } w); f))) )$   
**by** (rule *IfThenElseImp*)  
**have** 3:  $\vdash \text{halt } (\text{init } w); (\neg f) =$   
 $\text{if}_i (\text{init } w) \text{ then } (\neg f) \text{ else } (\bigcirc (\text{halt } (\text{init } w); (\neg f)))$   
**by** (rule *HaltChopEqv*)  
**have** 4:  $\vdash \text{if}_i (\text{init } w) \text{ then } (\neg f) \text{ else } (\bigcirc (\text{halt } (\text{init } w); (\neg f))) \longrightarrow$   
 $( ((\text{init } w) \longrightarrow \neg f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc (\text{halt } (\text{init } w); (\neg f)))) )$   
**by** (rule *IfThenElseImp*)  
**have** 5:  $\vdash \text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); (\neg f) \longrightarrow$   
 $( ((\text{init } w) \longrightarrow f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc (\text{halt } (\text{init } w); f))) ) \wedge$   
 $( ((\text{init } w) \longrightarrow \neg f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc (\text{halt } (\text{init } w); (\neg f)))) )$   
**using** 1 2 3 4 **by** fastforce

**have** 6:  $\vdash ((\text{init } w) \longrightarrow f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt } (\text{init } w); f))) \wedge$   
 $(\text{init } w) \longrightarrow \neg f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt } (\text{init } w); \neg f))) \longrightarrow$   
 $(\bigcirc(\text{halt } (\text{init } w); f)) \wedge (\bigcirc(\text{halt } (\text{init } w); \neg f))$   
**by** *auto*  
**have** 7:  $\vdash \text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); \neg f \longrightarrow$   
 $(\bigcirc(\text{halt } (\text{init } w); f)) \wedge (\bigcirc(\text{halt } (\text{init } w); \neg f))$   
**using** 5 6 *lift-imp-trans* **by** *blast*  
**have** 8:  $\vdash ((\bigcirc(\text{halt } (\text{init } w); f)) \wedge (\bigcirc(\text{halt } (\text{init } w); \neg f))) =$   
 $\bigcirc(\text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); \neg f)$   
**using** *NextAndEqvNextAndNext* **by** *fastforce*  
**have** 9:  $\vdash \text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); \neg f \longrightarrow$   
 $\bigcirc(\text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); \neg f)$   
**using** 7 8 **by** *fastforce*  
**hence** 10:  $\vdash \neg(\text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); \neg f)$   
**using** *NextLoop* **by** *blast*  
**from** 10 **show** *?thesis* **by** *auto*  
**qed**

**lemma** *HaltChopImpHaltYields*:

$\vdash \text{halt } (\text{init } w); f \longrightarrow (\text{halt } (\text{init } w)) \text{ yields } f$

**proof** –

**have** 1:  $\vdash \text{halt } (\text{init } w); f \longrightarrow \neg(\text{halt } (\text{init } w); \neg f)$  **by** (*rule HaltChopImpNotHaltChopNot*)  
**from** 1 **show** *?thesis* **by** (*simp add: yields-d-def*)  
**qed**

**lemma** *HaltChopAnd*:

$\vdash (\text{halt } (\text{init } w)); f \wedge (\text{halt } (\text{init } w)); g \longrightarrow (\text{halt } (\text{init } w)); (f \wedge g)$

**proof** –

**have** 1:  $\vdash (\text{halt } (\text{init } w)); g \longrightarrow (\text{halt } (\text{init } w)) \text{ yields } g$  **by** (*rule HaltChopImpHaltYields*)  
**hence** 2:  $\vdash (\text{halt } (\text{init } w)); f \wedge (\text{halt } (\text{init } w)); g \longrightarrow$   
 $(\text{halt } (\text{init } w)); f \wedge (\text{halt } (\text{init } w)) \text{ yields } g$  **by** *auto*  
**have** 3:  $\vdash (\text{halt } (\text{init } w)); f \wedge (\text{halt } (\text{init } w)) \text{ yields } g \longrightarrow$   
 $(\text{halt } (\text{init } w)); (f \wedge g)$  **by** (*rule ChopAndYieldsImp*)  
**from** 2 3 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *HaltAndChopAndHaltChopImpHaltAndChopAnd*:

$\vdash (\text{halt } (\text{init } w) \wedge f); f1 \wedge (\text{halt } (\text{init } w); g) \longrightarrow (\text{halt } (\text{init } w) \wedge f); (f1 \wedge g)$

**proof** –

**have** 1:  $\vdash f1 \longrightarrow \neg g \vee (f1 \wedge g)$   
**by** *auto*  
**hence** 2:  $\vdash (\text{halt } (\text{init } w) \wedge f); f1 \longrightarrow$   
 $(\text{halt } (\text{init } w) \wedge f); (\neg g) \vee ((\text{halt } (\text{init } w) \wedge f); (f1 \wedge g))$   
**by** (*rule ChopOrImpRule*)  
**have** 3:  $\vdash (\text{halt } (\text{init } w) \wedge f); (\neg g) \longrightarrow \text{halt } (\text{init } w); (\neg g)$   
**by** (*rule AndChopA*)  
**have** 31:  $\vdash (\text{halt } (\text{init } w) \wedge f); f1 \longrightarrow$   
 $\text{halt } (\text{init } w); (\neg g) \vee ((\text{halt } (\text{init } w) \wedge f); (f1 \wedge g))$   
**using** 23 **by** *fastforce*  
**have** 4:  $\vdash \text{halt } (\text{init } w); g \longrightarrow \neg(\text{halt } (\text{init } w); \neg g)$

by (rule HaltChopImpNotHaltChopNot)  
 hence 41:  $\vdash (\text{halt } (\text{init } w); (\neg g)) \longrightarrow \neg(\text{halt } (\text{init } w); g)$   
 by auto  
 have 42:  $\vdash (\text{halt } (\text{init } w) \wedge f); f1 \longrightarrow$   
 $\neg(\text{halt } (\text{init } w); g) \vee ((\text{halt } (\text{init } w) \wedge f); (f1 \wedge g))$   
 using 31 41 by fastforce  
 from 42 show ?thesis by auto  
 qed

**lemma** HaltImpBoxYields:  
 $\vdash (\text{halt } (\text{init } w)); f \longrightarrow (\Box(\neg (\text{init } w))) \text{ yields } ((\text{halt } (\text{init } w)); f)$   
**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:  $\vdash \Box(\neg (\text{init } w)) \longrightarrow \text{empty} \vee \bigcirc(\Box(\neg (\text{init } w)))$   
 using BoxBoxImpBox BoxEqvAndEmptyOrNextBox by fastforce  
 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:  $\vdash (\text{halt } (\text{init } w)); f \wedge (\Box (\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)) \longrightarrow$ 
 $\quad \bigcirc((\text{halt } (\text{init } w)); f) \wedge \bigcirc((\Box (\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)))$ 
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:  $\vdash (\text{halt } (\text{init } w)); f \wedge (\Box (\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)) \longrightarrow$ 
 $\quad \bigcirc(((\text{halt } (\text{init } w)); f) \wedge ((\Box (\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f))))$ 
using 11 12 by fastforce
hence 14:  $\vdash \neg ((\text{halt } (\text{init } w)); f \wedge (\Box (\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)))$ 
using NextLoop by blast
hence 15:  $\vdash (\text{halt } (\text{init } w)); f \longrightarrow \neg ((\Box (\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)))$ 
by auto
from 15 show ?thesis by (simp add: yields-d-def)
qed

```

## 5.10 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**

## 6 First Order Finite ITL theorems

**theory** *FOTheorems*

**imports**

*Theorems*

**begin**

We give the proofs of a list of first order Finite ITL theorems.

**lemma** *EEExI-unl*:

```

 $w \models f x \implies w \models (\exists \exists x. f x)$ 
using EEExVal by auto

```

**lemma** *EEExNoDep*:

```

 $\vdash (\exists \exists x. g) = g$ 

```

**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** *EEExOrDist-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 EEExI*)

**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** *EEExE* **by** *blast*

**qed**

**lemma** *EEExOrDist-3*:

$\vdash (\exists \exists x. f x) \vee (\exists \exists x. h x) \longrightarrow (\exists \exists x. (f x) \vee (h x))$

**using** *EEExOrDist-2 EEExOrDist-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:  $\bigwedge x. \vdash f x \longrightarrow (g \longrightarrow (\exists \exists x. g \wedge f x))$  **by** (*metis EExI int-eq lift-and-com Prop09*)  
**hence** 3:  $\vdash (\exists \exists x. f x) \longrightarrow (g \longrightarrow (\exists \exists x. g \wedge f x))$  **by** (*simp add: EExE*)  
**from** 1 3 **show** *?thesis* **by** *auto*  
**qed**

**lemma EExAndImport-2:**  
 $\vdash (\exists \exists x. g \wedge f x) \longrightarrow g \wedge (\exists \exists x. f x)$   
**proof** –  
**have** 1:  $\bigwedge x. \vdash g \wedge f x \longrightarrow g \wedge (\exists \exists x. f x)$   
**by** (*metis EExI int-iffD2 lift-and-com lift-imp-trans Prop12*)  
**from** 1 **show** *?thesis* **by** (*simp add: EExE*)  
**qed**

**lemma EExAndImport:**  
 $\vdash (g \wedge (\exists \exists x. f x)) = (\exists \exists x. g \wedge f x)$   
**by** (*simp add: EExAndImport-1 EExAndImport-2 int-iffI*)

**lemma EExAndDist:**  
**assumes**  $\vdash f x \wedge g x$   
**shows**  $\vdash (\exists \exists x. f x) \wedge (\exists \exists x. g x)$   
**proof** –  
**have** 1:  $\vdash f x$  **using** *assms* **by** *fastforce*  
**have** 2:  $\vdash g x$  **using** *assms* **by** *fastforce*  
**have** 3:  $\vdash (\exists \exists x. f x)$  **using** 1 **by** (*meson EExI MP*)  
**have** 4:  $\vdash (\exists \exists x. g x)$  **using** 2 **by** (*meson EExI MP*)  
**from** 3 4 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma EExAndNoDepDist:**  
**assumes**  $\vdash f \wedge g x$   
**shows**  $\vdash f \wedge (\exists \exists x. g x)$   
**proof** –  
**have** 1:  $\vdash f$  **using** *assms* **by** *fastforce*  
**have** 2:  $\vdash g x$  **using** *assms* **by** *fastforce*  
**have** 3:  $\vdash (\exists \exists x. g x)$  **using** 2 **by** (*meson EExI MP*)  
**from** 1 3 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma Spec:**  
 $\vdash (\forall \forall x. f x) \longrightarrow f x$   
**proof** –  
**have** 1:  $\vdash \neg(f x) \longrightarrow (\exists \exists x. \neg(f x))$  **by** (*meson EExI*)  
**have** 2:  $\vdash \neg(\exists \exists x. \neg(f x)) \longrightarrow f x$  **using** 1 **by** *auto*  
**from** 2 **show** *?thesis* **using** *AAxDef* **by** *fastforce*

qed

**lemma** *AAxE*:

**assumes**  $\vdash (\forall\forall x. f x)$

$\vdash f x \longrightarrow g$

**shows**  $\vdash g$

**using** *MP Spec assms(1) assms(2) by blast*

**lemma** *AAxI*:

**assumes**  $\bigwedge x. \vdash f x$

**shows**  $\vdash (\forall\forall x. f x)$

**using** *assms by (simp add: Valid-def exist-state-d-def forall-state-d-def)*

**lemma** *AAxEqvRule*:

**assumes**  $\bigwedge x. \vdash f x = g x$

**shows**  $\vdash (\forall\forall x. f x) = (\forall\forall x. g x)$

**by** (*metis (mono-tags, lifting) AAxDef EExEqvRule assms int-iffD1 int-iffI  
inteq-reflection lift-imp-neg*)

**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:  $\vdash ((\exists\exists x. \neg(f x)) \vee (\exists\exists x. \neg(g x))) = (\exists\exists x. \neg(f x) \vee \neg(g x))$

**by** (*simp add: EExOrDist*)

**have** 2:  $\vdash ((\exists\exists x. \neg(f x))) = (\neg(\forall\forall x. f x))$

**using** *AAxDef by fastforce*

**have** 3:  $\vdash ((\exists\exists x. \neg(g x))) = (\neg(\forall\forall x. g x))$

**using** *AAxDef by fastforce*

**have** 4:  $\vdash ((\exists\exists x. \neg(f x)) \vee (\exists\exists x. \neg(g x))) = (\neg(\forall\forall x. f x) \vee \neg(\forall\forall x. g x))$

**using** 2 3 **by** *fastforce*

**have** 5:  $\bigwedge x. \vdash (\neg(f x) \vee \neg(g x)) = (\neg((f x) \wedge (g x)))$

**by** *auto*

**have** 6:  $\vdash (\exists\exists x. \neg(f x) \vee \neg(g x)) = (\exists\exists x. \neg((f x) \wedge (g x)))$

**using** 5 **by** (*simp add: EExEqvRule*)

**have** 7:  $\vdash (\exists\exists x. \neg((f x) \wedge (g x))) = (\neg(\forall\forall x. (f x) \wedge (g x)))$

**using** *AAxDef by fastforce*

**have** 8:  $\vdash (\neg(\forall\forall x. f x) \vee \neg(\forall\forall x. g x)) = (\neg((\forall\forall x. f x) \wedge (\forall\forall x. g x)))$

**by** *fastforce*

**have** 9:  $\vdash (\neg((\forall\forall x. f x) \wedge (\forall\forall x. g x))) = (\neg(\forall\forall x. (f x) \wedge (g x)))$

**using** 1 4 6 7 8 **by** *fastforce*

**from** 9 **show** *?thesis 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:  $\vdash ((\exists\exists x. \neg(f x))) = (\neg(\forall\forall x. f x))$  **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:  $\vdash (\neg(g \wedge (\forall \forall x. f x))) = (\neg(\forall \forall x. g \wedge f x))$  **using** 1 3 5 6 **by** *fastforce*  
**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:  $\vdash (\neg(g \vee (\forall \forall x. f x))) = (\neg(\forall \forall x. g \vee f x))$  **using** 1 3 5 6 **by** *fastforce*  
**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** *EEExEEExChopEqvEEExChopEEExA*:  
 $\vdash (\exists \exists v. (\exists \exists y. (f\ v);(g\ y) )) = (\exists \exists v. (f\ v);(\exists \exists y. (g\ y) ))$   
**by** (*simp add: exist-state-d-def Valid-def itl-defs*) *blast*

**lemma** *EEExEEExChopEqvEEExChopEEExB*:  
 $\vdash (\exists \exists y. (\exists \exists v. (f\ v);(g\ y) )) = (\exists \exists y. (\exists \exists v. (f\ v)); (g\ y))$   
**by** (*simp add: exist-state-d-def Valid-def itl-defs*) *blast*

**lemma** *EEExEEExChopEqvEEExChopEEExC*:  
 $\vdash (\exists \exists v. (\exists \exists y. (f\ v);(g\ y) )) = (\exists \exists v. (f\ v));(\exists \exists y. (g\ y))$   
**using** *EEExChopRightNoDep*[*of f LIFT(( $\exists \exists y. g\ y$ ))*]  
*EEExEEExChopEqvEEExChopEEExA*[*of f g*]  
**by** *fastforce*

**lemma** *ExLen*:  
 $\vdash \exists n. \text{len}(n)$   
**by** (*simp add: Valid-def itl-defs*)

**lemma** *CSPowerChop*:  
 $\vdash (f^*) = (\exists n. \text{power}\ (f \wedge \text{more})\ n)$   
**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*)

**lemma** *TassignEqvExFin*:  
 $\vdash v \leftarrow e = (\exists c. \#c=e \wedge \text{fin}(\$v = \#c))$   
**by** (*simp add: Valid-def itl-defs*)

**lemma** *MoreImpNextassignEqvExNext*:  
 $\vdash \text{more} \longrightarrow (v := e) = (\exists c. \#c=e \wedge \bigcirc(\$v = \#c))$   
**by** (*simp add: Valid-def next-assign-d-def next-val-d-def itl-defs*)

**lemma** *MoreImpPrevassignEqvExPrevFin*:  
 $\vdash \text{more} \longrightarrow (v =: e) = (\exists c. \#c=e \wedge \text{prev}(\text{fin}(\$v = \#c)) )$   
**by** (*auto simp add: min.absorb1 Valid-def prev-assign-d-def itl-defs penult-val-d-def*)

**end**

## 7 Time Reversal

```

theory TimeReversal
imports
  Theorems FOTheorems
begin

```

Time reversal operator is defined in [6].

### 7.1 Definition

```

definition reverse-d :: ('a::world, 'b) formfun  $\Rightarrow$  ('a, 'b) formfun
where reverse-d F  $\equiv \lambda s. irev s \models F$ 

```

```

syntax
  -reverse-d      :: lift  $\Rightarrow$  lift      ((r-) [85] 85)

```

```

syntax (ASCII)
  -reverse-d      :: lift  $\Rightarrow$  lift      ((reverse -) [85] 85)

```

```

translations
  -reverse-d       $\equiv$  CONST reverse-d

```

### 7.2 Time reversal Rules

```

lemma EExRev :
   $\vdash (\exists \exists x. F x)^r = (\exists \exists x. (F x)^r)$ 
by (simp add: Valid-def exist-state-d-def reverse-d-def)

```

```

lemma rev-const :
   $\vdash (\#c)^r = \#c$ 
by (auto simp: reverse-d-def)

```

```

lemma rev-fun1 :
   $\vdash (f\langle x \rangle)^r = f\langle x^r \rangle$ 
by (auto simp: reverse-d-def)

```

```

lemma rev-fun2:
   $\vdash (f\langle x, y \rangle)^r = f\langle x^r, y^r \rangle$ 
by (auto simp: reverse-d-def)

```

```

lemma rev-fun3:
   $\vdash (f\langle x, y, z \rangle)^r = f\langle x^r, y^r, z^r \rangle$ 
by (auto simp: reverse-d-def)

```

```

lemma rev-forall:
   $\vdash (\forall x. P x)^r = (\forall x. (P x)^r)$ 
by (auto simp: reverse-d-def)

```

```

lemma rev-exists:
   $\vdash (\exists x. P x)^r = (\exists x. (P x)^r)$ 

```

**by** (*auto simp: reverse-d-def*)

**lemma** *rev-exists1*:

$\vdash (\exists! x. P x)^r = (\exists! x. (P x)^r)$

**by** (*auto simp: reverse-d-def*)

**lemma** *rev-current*:

$\vdash (\$v)^r = (!v)$

**by** (*auto simp: irev-inth itl-def reverse-d-def*)

**lemma** *rev-next*:

$\vdash (v\$)^r = (v!)$

**by** (*auto simp: irev-inth itl-def reverse-d-def*)

**lemma** *rev-penult*:

$\vdash (v!)^r = (v\$)$

**by** (*auto simp: irev-inth itl-def reverse-d-def*)

**lemma** *rev-fin*:

$\vdash (!v)^r = (\$v)$

**by** (*auto simp: irev-inth itl-def reverse-d-def*)

**lemma** *EqvReverseReverse*:

$\vdash (f^r)^r = f$

**by** (*simp add: Valid-def reverse-d-def*)

**lemma** *ReverseEqv*:

$(\vdash f) \longleftrightarrow (\vdash f^r)$

**by** (*metis Valid-def irev-swap reverse-d-def*)

**lemma** *RevSkip*:

$\vdash \text{skip}^r = \text{skip}$

**by** (*simp add: Valid-def reverse-d-def skip-defs*)

**lemma** *RevChop*:

$\vdash (f;g)^r = (g^r;f^r)$

**proof** (*auto simp add: Valid-def itl-def reverse-d-def*)

**show**  $\bigwedge w n. n \leq \text{ilen } w \implies$

$f (\text{prefix } n (\text{irev } w)) \implies$

$g (\text{suffix } n (\text{irev } w)) \implies$

$\exists n \leq \text{ilen } w. g (\text{irev } (\text{prefix } n w)) \wedge f (\text{irev } (\text{suffix } n w))$

**by** (*metis diff-diff-cancel irev-prefix irev-suffix suffix-ilen-bound suffix-ilen*)

**show**  $\bigwedge w n. n \leq \text{ilen } w \implies$

$g (\text{irev } (\text{prefix } n w)) \implies$

$f (\text{irev } (\text{suffix } n w)) \implies$

$\exists n \leq \text{ilen } w. f (\text{prefix } n (\text{irev } w)) \wedge g (\text{suffix } n (\text{irev } w))$

**by** (*metis irev-prefix irev-suffix suffix-ilen-bound suffix-ilen*)

**qed**

**lemma** *RMoreEqvMore*:

$\vdash \text{more}^r = \text{more}$   
**by** (*simp add: Valid-def itl-def reverse-d-def*)

**lemma** *REmptyEqvEmpty*:  
 $\vdash \text{empty}^r = \text{empty}$   
**by** (*metis RMoreEqvMore empty-d-def int-eq rev-fun1*)

**lemma** *PowerCommute*:  
 $\vdash ((f \wedge \text{more}); (\text{power } (f \wedge \text{more}) \ n)) = (\text{power } (f \wedge \text{more}) \ n); (f \wedge \text{more})$   
**proof**  
 (*induct n*)  
**case** 0  
**then show** ?*case* **by** (*metis ChopEmpty EmptyChop inteq-reflection pow-0*)  
**next**  
**case** (*Suc n*)  
**then show** ?*case* **by** (*metis ChopAssoc inteq-reflection pow-Suc*)  
**qed**

**lemma** *REqvRule*:  
**assumes**  $\vdash f = g$   
**shows**  $\vdash (f^r) = (g^r)$   
**using** *assms*  
**using** *inteq-reflection* **by** *force*

**lemma** *RevPowerChop*:  
 $\vdash (\text{power } (f \wedge \text{more}) \ n)^r = (\text{power } ((f \wedge \text{more})^r) \ n)$   
**proof**  
 (*induct n*)  
**case** 0  
**then show** ?*case* **using** *REmptyEqvEmpty* **by** *auto*  
**next**  
**case** (*Suc n*)  
**then show** ?*case*  
**by** (*metis PowerCommute RevChop inteq-reflection pow-Suc*)  
**qed**

**lemma** *RevChopstar*:  
 $\vdash (f^\star)^r = (f^r)^\star$   
**proof** –  
**have** 1:  $\vdash (f^\star) = (\exists n. \text{power } (f \wedge \text{more}) \ n)$   
**by** (*simp add: chopstar-d-def powerstar-d-def Valid-def*)  
**have** 2:  $\vdash (f^\star)^r = (\exists n. \text{power } (f \wedge \text{more}) \ n)^r$   
**using** *REqvRule 1* **by** *blast*  
**have** 3:  $\vdash (\exists n. \text{power } (f \wedge \text{more}) \ n)^r = (\exists n. (\text{power } (f \wedge \text{more}) \ n)^r)$   
**by** (*simp add: rev-exists*)  
**have** 4:  $\vdash (\exists n. (\text{power } (f \wedge \text{more}) \ n)^r) = (\exists n. (\text{power } ((f \wedge \text{more})^r) \ n))$   
**by** (*simp add: RevPowerChop ExEqvRule*)  
**have** 5:  $\vdash (f \wedge \text{more})^r = (f^r \wedge \text{more})$   
**by** (*metis RMoreEqvMore inteq-reflection rev-fun2*)  
**hence** 6:  $\vdash (\exists n. (\text{power } ((f \wedge \text{more})^r) \ n)) = (\exists n. (\text{power } ((f^r \wedge \text{more}) \ n)))$



by (metis 4 inteq-reflection)  
 have 7:  $\vdash (\exists n. (\text{power } ((f^r \wedge \text{more})) n)) = (f^r)^*$   
 by (simp add: chopstar-d-def powerstar-d-def Valid-def)  
 from 2 3 4 6 7 show ?thesis by fastforce  
 qed

lemmas all-rev = rev-const rev-fun1 rev-fun2 rev-fun3 rev-forall rev-exists  
 rev-exists1 rev-current rev-next rev-penult rev-fin RevSkip RevChop RevChopstar

lemmas all-rev-unl = all-rev[THEN intD]  
 lemmas all-rev-eq = all-rev[THEN inteq-reflection]

### 7.3 Properties of Time Reversal

lemma RNot:  
 $\vdash (\neg f)^r = (\neg f^r)$   
 by (simp add: rev-fun1)

lemma RRNot:  
 $\vdash (\neg(f^r))^r = (\neg f)$   
 by (metis EquReverseReverse int-eq rev-fun1)

lemma RTrue:  
 $\vdash (\# \text{True})^r = \# \text{True}$   
 using rev-const by fastforce

lemma ROr:  
 $\vdash (f \vee g)^r = (f^r \vee g^r)$   
 by (simp add: rev-fun2)

lemma RROr:  
 $\vdash (f^r \vee g^r)^r = (f \vee g)$   
 proof –  
 have 1:  $\vdash (f^r \vee g^r)^r = ((f^r)^r \vee (g^r)^r)$  using ROr by blast  
 have 2:  $\vdash ((f^r)^r \vee (g^r)^r) = (f \vee g)$  using EquReverseReverse by (metis inteq-reflection)  
 from 1 2 show ?thesis by fastforce  
 qed

lemma RAnd:  
 $\vdash (f \wedge g)^r = (f^r \wedge g^r)$   
 by (simp add: rev-fun2)

lemma RRAnd:  
 $\vdash (f^r \wedge g^r)^r = (f \wedge g)$   
 proof –  
 have 1:  $\vdash (f^r \wedge g^r)^r = ((f^r)^r \wedge (g^r)^r)$  using RAnd by blast  
 have 2:  $\vdash ((f^r)^r \wedge (g^r)^r) = (f \wedge g)$  using EquReverseReverse by (metis inteq-reflection)  
 from 1 2 show ?thesis by fastforce  
 qed

**lemma** *RImpRule*:  
**assumes**  $\vdash f \longrightarrow g$   
**shows**  $\vdash f^r \longrightarrow g^r$   
**using** *assms* **by** (*simp add: Valid-def reverse-d-def*)

**lemma** *RAndEmptyEqvAndEmpty*:  
 $\vdash (f \wedge \text{empty})^r = (f \wedge \text{empty})$   
**by** (*simp add: Valid-def itl-defs reverse-d-def, metis INil-ilen irev.simps(1)*)

**lemma** *RNextEqvPrev*:  
 $\vdash (\bigcirc f)^r = \text{prev } (f^r)$   
**by** (*metis RevChop RevSkip inteq-reflection next-d-def prev-d-def*)

**lemma** *RRNextEqvPrev*:  
 $\vdash (\bigcirc (f^r))^r = \text{prev } (f)$   
**proof** –  
**have** 1:  $\vdash (\bigcirc (f^r))^r = \text{prev } ((f^r)^r)$  **using** *RNextEqvPrev* **by** *blast*  
**have** 2:  $\vdash \text{prev } ((f^r)^r) = \text{prev } f$  **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)  
**from** 1 2 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *RWNextEqvWPrev*:  
 $\vdash (\text{wnext } f)^r = \text{wprev } (f^r)$   
**by** (*simp add: all-rev-eq(12) all-rev-eq(13) all-rev-eq(2) next-d-def prev-d-def wnext-d-def wprev-d-def*)

**lemma** *RRWNextEqvWPrev*:  
 $\vdash (\text{wnext } (f^r))^r = \text{wprev } (f)$   
**proof** –  
**have** 1:  $\vdash (\text{wnext } (f^r))^r = \text{wprev } ((f^r)^r)$  **using** *RWNextEqvWPrev* **by** *blast*  
**have** 2:  $\vdash \text{wprev } ((f^r)^r) = \text{wprev } f$  **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)  
**from** 1 2 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *RPrevEqvNext*:  
 $\vdash (\text{prev } f)^r = \bigcirc (f^r)$   
**by** (*metis RevChop RevSkip inteq-reflection next-d-def prev-d-def*)

**lemma** *RRPrevEqvNext*:  
 $\vdash (\text{prev } (f^r))^r = \bigcirc (f)$   
**proof** –  
**have** 1:  $\vdash (\text{prev } (f^r))^r = \bigcirc ((f^r)^r)$  **using** *RPrevEqvNext* **by** *blast*  
**have** 2:  $\vdash \bigcirc ((f^r)^r) = \bigcirc f$  **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)  
**from** 1 2 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *RWPrevEqvWNext*:  
 $\vdash (\text{wprev } f)^r = \text{wnext } (f^r)$

**by** (*metis* *EqvReverseReverse* *RRWNextEqvWPrev* *int-eq*)

**lemma** *RRWPrevEqvWNext*:

$\vdash (wprev (f^r))^r = wnext(f)$

**proof** –

**have** 1:  $\vdash (wprev (f^r))^r = wnext ((f^r)^r)$  **using** *RRWPrevEqvWNext* **by** *blast*

**have** 2:  $\vdash wnext ((f^r)^r) = wnext f$  **using** *EqvReverseReverse* **by** (*metis* *inteq-reflection*)

**from** 1 2 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *RDiamondEqvDi*:

$\vdash (\Diamond f)^r = di (f^r)$

**by** (*simp* *add*: *di-d-def* *sometimes-d-def*, *metis* *RevChop* *RTrue* *inteq-reflection*)

**lemma** *RRDiamondEqvDi*:

$\vdash (\Diamond (f^r))^r = di (f)$

**proof** –

**have** 1:  $\vdash (\Diamond (f^r))^r = di ((f^r)^r)$  **using** *RDiamondEqvDi* **by** *blast*

**have** 2:  $\vdash di ((f^r)^r) = di f$  **using** *EqvReverseReverse* **by** (*metis* *inteq-reflection*)

**from** 1 2 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *RBoxEqvBi*:

$\vdash (\Box f)^r = bi (f^r)$

**by** (*simp* *add*: *always-d-def* *bi-d-def*, *metis* *RDiamondEqvDi* *int-eq* *rev-fun1* )

**lemma** *RRBoxEqvBi*:

$\vdash (\Box (f^r))^r = bi (f)$

**proof** –

**have** 1:  $\vdash (\Box (f^r))^r = bi ((f^r)^r)$  **using** *RBoxEqvBi* **by** *blast*

**have** 2:  $\vdash bi ((f^r)^r) = bi f$  **using** *EqvReverseReverse* **by** (*metis* *inteq-reflection*)

**from** 1 2 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *RDIEqvDiamond*:

$\vdash (di f)^r = \Diamond (f^r)$

**by** (*simp* *add*: *di-d-def* *sometimes-d-def*, *metis* *RevChop* *RTrue* *inteq-reflection*)

**lemma** *RRDiEqvDiamond*:

$\vdash (di (f^r))^r = \Diamond (f)$

**proof** –

**have** 1:  $\vdash (di (f^r))^r = \Diamond ((f^r)^r)$  **using** *RDIEqvDiamond* **by** *blast*

**have** 2:  $\vdash \Diamond ((f^r)^r) = \Diamond f$  **using** *EqvReverseReverse* **by** (*metis* *inteq-reflection*)

**from** 1 2 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *RBIEqvBox*:

$\vdash (bi f)^r = \Box (f^r)$

**by** (*simp* *add*: *always-d-def* *bi-d-def*, *metis* *RDIEqvDiamond* *rev-fun1* *int-eq*)

**lemma** *RRBiEqvBox*:

$\vdash (bi\ (f^r))^r = \square\ (f)$

**proof** –

**have** 1:  $\vdash (bi\ (f^r))^r = \square\ ((f^r)^r)$  **using** *RBiEqvBox* **by** *blast*

**have** 2:  $\vdash \square\ ((f^r)^r) = \square\ f$  **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)

**from** 1 2 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *RDaEqvDa*:

$\vdash (da\ f)^r = da(f^r)$

**proof** –

**have** 1:  $\vdash (\#True;(f;\#True))^r = (f;\#True)^r; \#True^r$  **using** *RevChop* **by** *blast*

**have** 2:  $\vdash (f;\#True)^r; \#True^r = (f;\#True)^r; \#True$  **using** *RTrue RightChopEqvChop* **by** *blast*

**have** 3:  $\vdash (f;\#True)^r; \#True = (\#True^r;f^r);\#True$  **by** (*simp add: RevChop LeftChopEqvChop*)

**have** 4:  $\vdash (\#True^r;f^r);\#True = (\#True;f^r);\#True$  **by** (*metis 3 RTrue int-eq*)

**have** 5:  $\vdash (\#True;f^r);\#True = \#True;(f^r;\#True)$  **using** *ChopAssocB* **by** *blast*

**have** 6:  $\vdash (\#True;(f;\#True))^r = \#True;(f^r;\#True)$  **using** 1 2 3 4 5 **by** *fastforce*

**from** 6 **show** *?thesis* **by** (*simp add: da-d-def*)

**qed**

**lemma** *RRDaEqvDa*:

$\vdash (da\ (f^r))^r = da(f)$

**proof** –

**have** 1:  $\vdash (da\ (f^r))^r = da\ ((f^r)^r)$  **using** *RDaEqvDa* **by** *blast*

**have** 2:  $\vdash da\ ((f^r)^r) = da\ f$  **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)

**from** 1 2 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *RBaEqvBa*:

$\vdash (ba\ f)^r = ba(f^r)$

**by** (*simp add: ba-d-def, metis RDaEqvDa int-eq rev-fun1*)

**lemma** *RRBaEqvBa*:

$\vdash (ba\ (f^r))^r = ba(f)$

**proof** –

**have** 1:  $\vdash (ba\ (f^r))^r = ba\ ((f^r)^r)$  **using** *RBaEqvBa* **by** *blast*

**have** 2:  $\vdash ba\ ((f^r)^r) = ba\ f$  **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)

**from** 1 2 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *ChopCsImpCSChop*:

$\vdash f;f^* \longrightarrow f^*;f$

**by** (*meson CSChopEqvChopOrRule CSChopEqvOrChopPlusChop ChopAssocB ChopPlusElimWithoutMore EmptyYields Prop03 Prop04 Prop06*)

**lemma** *CSChopImpChopCS*:

$\vdash f^*;f \longrightarrow f;f^*$

**proof** –

**have** 1:  $\vdash (f^r);(f^r)^* \longrightarrow (f^r)^*; (f^r)$   
**using** *ChopCsImpCSChop* **by** *blast*  
**hence** 2:  $\vdash ((f^r);(f^r)^* \longrightarrow (f^r)^*; (f^r))^r$   
**using** *ReverseEqv* **by** *blast*  
**have** 3:  $\vdash (((f^r);(f^r)^* \longrightarrow (f^r)^*; (f^r))^r) = ((f^r);(f^r)^*)^r \longrightarrow ((f^r)^*; (f^r))^r$   
**by** (*simp add: rev-fun2*)  
**have** 4:  $\vdash ((f^r);(f^r)^*)^r = ((f^r)^*)^r; (f^r)^r$   
**by** (*simp add: RevChop*)  
**have** 5:  $\vdash ((f^r)^*)^r; (f^r)^r = ((f^r)^r)^*; (f^r)^r$   
**by** (*simp add: LeftChopEqvChop RevChopstar*)  
**have** 6:  $\vdash (f^r)^r = f$   
**using** *EqvReverseReverse* **by** *blast*  
**have** 7:  $\vdash ((f^r)^r)^*; (f^r)^r = f^*; f$   
**using** 6 *CSEqvCS ChopEqvChop* **by** *blast*  
**have** 8:  $\vdash ((f^r);(f^r)^*)^r = f^*; f$   
**using** 7 5 **using** 4 **by** *fastforce*  
**have** 9:  $\vdash ((f^r)^*; (f^r))^r = (f^r)^r; ((f^r)^*)^r$   
**by** (*simp add: RevChop*)  
**have** 10:  $\vdash (f^r)^r; ((f^r)^*)^r = (f^r)^r; ((f^r)^r)^*$   
**by** (*simp add: RevChopstar RightChopEqvChop*)  
**have** 11:  $\vdash (f^r)^r; ((f^r)^r)^* = f; f^*$   
**using** 6 *ChopPlusEqvChopPlus* **by** *blast*  
**have** 12:  $\vdash ((f^r);(f^r)^*)^r = f; f^*$   
**using** 9 10 11 **by** (*metis* 4 5 *ChopCsImpCSChop RImpRule int-eq int-iffI*)  
**from** 2 3 8 12 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *CSChopEqvChopCS*:

$\vdash f; f^* = f^*; f$

**using** *ChopCsImpCSChop CSChopImpChopCS* **by** *fastforce*

**lemma** *TrueChopSkipEqvSkipChopTrue*:

$\vdash \#True; skip = skip; \#True$

**proof** –

**have** 1:  $\vdash skip; skip^* = skip^*; skip$  **using** *CSChopEqvChopCS* **by** *blast*

**have** 2:  $\vdash skip^* = \#True$  **using** *CSSkip* **by** *simp*

**have** 3:  $\vdash skip; skip^* = skip; \#True$  **using** 2 **using** *RightChopEqvChop* **by** *blast*

**have** 4:  $\vdash skip^*; skip = \#True; skip$  **using** 2 **using** *LeftChopEqvChop* **by** *blast*

**from** 1 3 4 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *RInitEqvFin*:

$\vdash (init\ f)^r = fin(f)$

**proof** –

**have** 1:  $\vdash (init\ f)^r = ((f \wedge empty); \#True)^r$

**by** (*metis* *AndChopCommute REqvRule init-d-def*)

**have** 2:  $\vdash ((f \wedge empty); \#True)^r = (\#True; (f \wedge empty))^r$

**using** *RTrue* **by** (*metis* *RevChop int-eq*)

**have** 3:  $\vdash \#True; (f \wedge empty)^r = \#True; (f^r \wedge empty)$

**by** (*metis* *RAnd REmptyEqvEmpty RightChopEqvChop int-eq*)

**have** 4:  $\vdash \#True; (f^r \wedge \text{empty}) = \#True; (f \wedge \text{empty})$   
**using** *RAndEmptyEqvAndEmpty*  
**by** (*metis REmptyEqvEmpty RightChopEqvChop all-rev-eq(3) int-eq*)  
**have** 5:  $\vdash \#True; (f \wedge \text{empty}) = \text{fin}(f)$   
**using** *FinEqvTrueChopAndEmpty* **by** *fastforce*  
**from** 1 2 3 4 5 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *RFinEqvInit*:

$\vdash (\text{fin } f)^r = \text{init } (f)$   
**proof** –  
**have** 1:  $\vdash \text{fin } f = \#True; (f \wedge \text{empty})$   
**using** *FinEqvTrueChopAndEmpty* **by** *auto*  
**have** 2:  $\vdash (\text{fin } f)^r = (\#True; (f \wedge \text{empty}))^r$   
**using** 1 *REqvRule* **by** *blast*  
**have** 3:  $\vdash (\#True; (f \wedge \text{empty}))^r = (f \wedge \text{empty})^r; \#True$   
**using** *RTrue* **by** (*metis RevChop int-eq*)  
**have** 4:  $\vdash (f \wedge \text{empty})^r; \#True = (f^r \wedge \text{empty}); \#True$   
**using** *LeftChopEqvChop RAnd REmptyEqvEmpty* **by** (*metis int-eq*)  
**have** 5:  $\vdash (f \wedge \text{empty})^r; \#True = (f \wedge \text{empty}); \#True$   
**by** (*simp add: RAndEmptyEqvAndEmpty LeftChopEqvChop*)  
**have** 6:  $\vdash (f \wedge \text{empty}); \#True = \text{init}(f)$   
**by** (*simp add: AndChopCommute init-d-def*)  
**from** 1 2 3 4 5 6 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *RHaltEqvInitonly*:

$\vdash (\text{halt } f)^r = \text{initonly } (f^r)$   
**proof** –  
**have** 1:  $\vdash (\text{halt } f)^r = (\Box ( \text{empty} = f ))^r$  **by** (*simp add: halt-d-def*)  
**have** 2:  $\vdash (\Box ( \text{empty} = f ))^r = \text{bi } ( \text{empty} = f )^r$  **by** (*simp add: RBoxEqvBi*)  
**have** 3:  $\vdash (\text{empty} = f)^r = (\text{empty} = f^r)$  **by** (*metis REmptyEqvEmpty inteq-reflection rev-fun2*)  
**hence** 4:  $\vdash \text{bi } ( \text{empty} = f )^r = \text{bi}(\text{empty} = f^r)$  **by** (*simp add: BiEqvBi*)  
**have** 5:  $\vdash \text{bi}(\text{empty} = f^r) = \text{initonly}(f^r)$  **by** (*simp add: initonly-d-def*)  
**from** 1 2 4 5 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *RInitonlyEqvHalt*:

$\vdash (\text{initonly } f)^r = \text{halt}(f^r)$   
**proof** –  
**have** 1:  $\vdash (\text{initonly } f)^r = (\text{bi } (\text{empty} = f))^r$  **by** (*simp add: initonly-d-def*)  
**have** 2:  $\vdash (\text{bi } (\text{empty} = f))^r = \Box((\text{empty} = f)^r)$  **by** (*simp add: RBiEqvBox*)  
**have** 3:  $\vdash (\text{empty} = f)^r = (\text{empty} = f^r)$  **by** (*metis REmptyEqvEmpty inteq-reflection rev-fun2*)  
**hence** 4:  $\vdash \Box((\text{empty} = f)^r) = \Box(\text{empty} = f^r)$  **by** (*simp add: BoxEqvBox*)  
**have** 5:  $\vdash \Box(\text{empty} = f^r) = \text{halt}(f^r)$  **by** (*simp add: halt-d-def*)  
**from** 1 2 4 5 **show** *?thesis* **by** *fastforce*

qed

**lemma** *RRHaltEqvInitonly*:

$\vdash (\text{halt } (f^r))^r = \text{initonly } (f)$

**proof** –

**have** 1:  $\vdash (\text{halt } (f^r))^r = \text{initonly } ((f^r)^r)$  **using** *RRHaltEqvInitonly* **by** *blast*

**have** 2:  $\vdash \text{initonly } ((f^r)^r) = \text{initonly } (f)$  **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)

**from** 1 2 **show** *?thesis* **by** *fastforce*

qed

**lemma** *RRInitonlyEqvHalt* :

$\vdash (\text{initonly } (f^r))^r = \text{halt } (f)$

**proof** –

**have** 1:  $\vdash (\text{initonly } (f^r))^r = \text{halt } ((f^r)^r)$  **using** *RRInitonlyEqvHalt* **by** *blast*

**have** 2:  $\vdash \text{halt } ((f^r)^r) = \text{halt } (f)$  **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)

**from** 1 2 **show** *?thesis* **by** *fastforce*

qed

**lemma** *RKeepEqvKeep* :

$\vdash (\text{keep } f)^r = \text{keep } (f^r)$

**proof** –

**have** 1:  $\vdash (\text{keep } f)^r = (\text{ba } (\text{skip} \longrightarrow f))^r$  **by** (*simp add: keep-d-def*)

**have** 2:  $\vdash (\text{ba } (\text{skip} \longrightarrow f))^r = \text{ba } ((\text{skip} \longrightarrow f)^r)$  **by** (*simp add: RBaEqvBa*)

**have** 3:  $\vdash (\text{skip} \longrightarrow f)^r = (\text{skip} \longrightarrow f^r)$  **by** (*metis all-rev-eq(12) rev-fun2*)

**hence** 4:  $\vdash \text{ba } ((\text{skip} \longrightarrow f)^r) = \text{ba } (\text{skip} \longrightarrow f^r)$  **by** (*simp add: BaEqvBa*)

**have** 5:  $\vdash \text{ba } (\text{skip} \longrightarrow f^r) = \text{keep } (f^r)$  **by** (*simp add: keep-d-def*)

**from** 1 2 4 5 **show** *?thesis* **by** *fastforce*

qed

**lemma** *RRKeepEqvKeep* :

$\vdash (\text{keep } (f^r))^r = \text{keep } (f)$

**proof** –

**have** 1:  $\vdash (\text{keep } (f^r))^r = \text{keep } ((f^r)^r)$  **using** *RKeepEqvKeep* **by** *blast*

**have** 2:  $\vdash \text{keep } ((f^r)^r) = \text{keep } (f)$  **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)

**from** 1 2 **show** *?thesis* **by** *fastforce*

qed

**lemma** *NextDiamondEqvDiamondNext*:

$\vdash \bigcirc(\Diamond f) = \Diamond(\bigcirc f)$

**proof** –

**have** 1:  $\vdash \# \text{True}; \text{skip} = \text{skip}; \# \text{True}$  **by** (*rule TrueChopSkipEqvSkipChopTrue*)

**hence** 2:  $\vdash (\# \text{True}; \text{skip}); f = (\text{skip}; \# \text{True}); f$  **using** *LeftChopEqvChop* **by** *blast*

**have** 3:  $\vdash (\# \text{True}; \text{skip}); f = \# \text{True}; (\text{skip}; f)$  **by** (*simp add: ChopAssocB*)

**have** 4:  $\vdash (\text{skip}; \# \text{True}); f = \text{skip}; (\# \text{True}; f)$  **by** (*simp add: ChopAssocB*)

**from** 2 3 4 **show** *?thesis* **by** (*metis int-eq next-d-def sometimes-d-def*)

qed

**lemma** *WeakNextBoxInduct*:

**assumes**  $\vdash \text{wnext } (\Box f) \longrightarrow f$

**shows**  $\vdash f$

**proof** –

**have** 1:  $\vdash \text{wnext } (\Box f) \longrightarrow f$  **using** *assms* **by** *blast*  
**hence** 2:  $\vdash \neg f \longrightarrow \neg (\text{wnext } (\Box f))$  **by** *fastforce*  
**hence** 3:  $\vdash \neg f \longrightarrow \bigcirc (\neg (\Box f))$  **by** (*simp add: wnext-d-def*)  
**have** 4:  $\vdash (\neg (\Box f)) = (\Diamond (\neg f))$  **by** (*auto simp: always-d-def*)  
**hence** 5:  $\vdash \bigcirc (\neg (\Box f)) = \bigcirc (\Diamond (\neg f))$  **using** *NextEqvNext* **by** *blast*  
**have** 6:  $\vdash \neg f \longrightarrow \bigcirc (\Diamond (\neg f))$  **using** 3 5 **by** *fastforce*  
**have** 7:  $\vdash \bigcirc (\Diamond (\neg f)) = \Diamond (\bigcirc (\neg f))$  **using** *NextDiamondEqvDiamondNext* **by** *blast*  
**have** 8:  $\vdash \neg f \longrightarrow \Diamond (\bigcirc (\neg f))$  **using** 6 7 **by** *fastforce*  
**have** 9:  $\vdash \Diamond (\neg f) \longrightarrow \Diamond (\Diamond (\bigcirc (\neg f)))$  **using** 8 *DiamondImpDiamond* **by** *blast*  
**have** 10:  $\vdash \Diamond (\Diamond (\bigcirc (\neg f))) = \Diamond (\bigcirc (\neg f))$  **using** *DiamondDiamondEqvDiamond* **by** *blast*  
**have** 11:  $\vdash \Diamond (\neg f) \longrightarrow \Diamond (\bigcirc (\neg f))$  **using** 9 10 **by** *fastforce*  
**have** 12:  $\vdash \Diamond (\neg f) \longrightarrow \bigcirc (\Diamond (\neg f))$  **using** 7 11 **by** *fastforce*  
**hence** 13:  $\vdash \neg (\Diamond (\neg f))$  **using** *NextLoop* **by** *blast*  
**hence** 14:  $\vdash \Box f$  **by** (*simp add: always-d-def*)  
**have** 15:  $\vdash \Box f \longrightarrow f$  **using** *BoxElim* **by** *blast*  
**from** 14 15 **show** *?thesis* **using** *MP* **by** *blast*

**qed**

**lemma** *RassignEqvTAssign*:

$\vdash (\$v = e)^r = (v \leftarrow e^r)$

**proof** –

**have** 1:  $\vdash (\$v = e)^r = ((\$v)^r = e^r)$  **by** (*simp add: rev-fun2*)  
**have** 2:  $\vdash ((\$v)^r = e^r) = (!v = e^r)$  **by** (*simp add: all-rev-eq(8)*)  
**have** 3:  $\vdash (!v = e^r) = (v \leftarrow e^r)$  **by** (*simp add: intI temporal-assign-d-def*)  
**from** 1 2 3 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *RTAssignEqvAssign*:

$\vdash (v \leftarrow e)^r = (\$v = e^r)$

**proof** –

**have** 1:  $\vdash (v \leftarrow e)^r = (!v = e)^r$  **by** (*simp add: REqvRule intI temporal-assign-d-def*)  
**have** 2:  $\vdash (!v = e)^r = (\$v = e^r)$  **by** (*metis all-rev-eq(11) rev-fun2*)  
**from** 1 2 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *RNextAssignEqvPrevAssign*:

$\vdash (v := e)^r = (v := e^r)$

**proof** –

**have** 1:  $\vdash (v := e)^r = (v\$ = e)^r$  **by** (*simp add: REqvRule intI next-assign-d-def*)  
**have** 2:  $\vdash (v\$ = e)^r = (v! = e^r)$  **by** (*metis all-rev-eq(9) rev-fun2*)  
**have** 3:  $\vdash (v! = e^r) = (v := e^r)$  **by** (*simp add: prev-assign-d-def*)  
**from** 1 2 3 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *RPrevAssignEqvNextAssign*:

$\vdash (v := e)^r = (v := e^r)$

**proof** –

**have** 1:  $\vdash (v := e)^r = (v! = e)^r$  **by** (*simp add: REqvRule intI prev-assign-d-def*)  
**have** 2:  $\vdash (v! = e)^r = (v\$ = e^r)$  **by** (*metis all-rev-eq(10) rev-fun2*)



**have** 3:  $\vdash (v\$ = e^r) = (v := e^r)$  **by** (*simp add: next-assign-d-def*)  
**from** 1 2 3 **show** ?thesis **by** fastforce  
**qed**

**lemma** *RGetsEqvBaSkipImp*:

$\vdash (v \text{ gets } e)^r = \text{ba}(\text{skip} \longrightarrow (\$v = e^r))$   
**proof** –  
**have** 1:  $\vdash (v \text{ gets } e)^r = (\text{ba}(\text{skip} \longrightarrow (!v = e)))^r$   
**using** gets-d-def temporal-assign-d-def keep-d-def REqvRule  
**by** (metis Prop04 ba-d-def int-simps(15))  
**have** 2:  $\vdash (\text{ba}(\text{skip} \longrightarrow (!v = e)))^r = \text{ba} ( (\text{skip} \longrightarrow (!v = e))^r )$   
**by** (simp add: RBaEqvBa)  
**have** 3:  $\vdash (\text{skip} \longrightarrow (!v = e))^r = (\text{skip} \longrightarrow (\$v = e^r))$   
**by** (simp add: all-rev-eq(11) all-rev-eq(12) all-rev-eq(3))  
**hence** 4:  $\vdash \text{ba} ((\text{skip} \longrightarrow (!v = e))^r) = \text{ba} (\text{skip} \longrightarrow (\$v = e^r))$   
**by** (simp add: BaEqvBa)  
**from** 1 2 4 **show** ?thesis **by** fastforce  
**qed**

**lemma** *RIfThenElse*:

$\vdash (\text{if}_i f0 \text{ then } f1 \text{ else } f2)^r = \text{if}_i (f0^r) \text{ then } (f1^r) \text{ else } (f2^r)$   
**by** (simp add: all-rev-eq(2) all-rev-eq(3) ifthenelse-d-def)

**lemma** *RWhile*:

$\vdash (\text{init } f \wedge \text{while } f0 \text{ do } f1)^r = ( \text{fin}(f) \wedge ((f0^r) \wedge (f1^r))^* \wedge \text{init } (\neg(f0)) )$   
**proof** –  
**have** 1:  $\vdash (\text{init } f \wedge \text{while } f0 \text{ do } f1)^r = ( \text{init } f \wedge (f0 \wedge f1)^* \wedge \text{fin } (\neg f0) )^r$   
**by** (simp add: while-d-def)  
**have** 2:  $\vdash ( \text{init } f \wedge (f0 \wedge f1)^* \wedge \text{fin } (\neg f0) )^r = ((\text{init } f)^r \wedge ((f0 \wedge f1)^*)^r \wedge (\text{fin } (\neg f0))^r)$   
**by** (simp add: all-rev-eq(3))  
**have** 3:  $\vdash (\text{init } f)^r = \text{fin}(f)$   
**by** (simp add: RInitEqvFin)  
**have** 4:  $\vdash ((f0 \wedge f1)^*)^r = ((f0^r) \wedge (f1^r))^*$   
**by** (metis RevChopstar all-rev-eq(3))  
**have** 5:  $\vdash (\text{fin } (\neg f0))^r = \text{init } (\neg(f0))$   
**by** (metis RFinEqvInit)  
**have** 6:  $\vdash ((\text{init } f)^r \wedge ((f0 \wedge f1)^*)^r \wedge (\text{fin } (\neg f0))^r) =$   
 $( \text{fin}(f) \wedge ((f0^r) \wedge (f1^r))^* \wedge \text{init } (\neg(f0)) )$  **using** 3 4 5 **by** fastforce  
**from** 1 2 6 **show** ?thesis **by** fastforce  
**qed**

**lemma** *AAxRev*:

$\vdash (\forall \forall x. F x)^r = (\forall \forall x. (F x)^r)$   
**proof** –  
**have** 1:  $\vdash (\forall \forall x. F x) = (\neg(\exists \exists x. \neg(F x)))$  **using** AAxDef **by** blast  
**have** 2:  $\vdash (\forall \forall x. F x)^r = (\neg(\exists \exists x. \neg(F x)))^r$  **using** REqvRule 1 **by** blast  
**have** 3:  $\vdash (\neg(\exists \exists x. \neg(F x)))^r = (\neg((\exists \exists x. (\neg(F x))^r)))$  **by** (simp add: rev-fun1)  
**have** 4:  $\vdash ((\exists \exists x. (\neg(F x))^r) = ((\exists \exists x. (\neg(F x))^r))$  **by** (simp add: EExRev)  
**hence** 5:  $\vdash (\neg((\exists \exists x. (\neg(F x))^r))) = (\neg(\exists \exists x. (\neg(F x))^r))$  **by** auto  
**have** 51:  $\bigwedge x. \vdash (\neg(F x))^r = (\neg(F x)^r)$  **by** (simp add: rev-fun1)

```

hence 52:  $\vdash (\exists \exists x. (\neg (F x))^r) = (\exists \exists x. \neg (F x)^r)$ 
  using EEqRule[of  $(\lambda x. \text{LIFT}(\neg (F x))^r) (\lambda x. \text{LIFT}(\neg (F x)^r))$ ]
  by fastforce
hence 6:  $\vdash (\neg(\exists \exists x. (\neg (F x))^r)) = (\neg(\exists \exists x. \neg (F x)^r))$  by fastforce
have 7:  $\vdash (\neg(\exists \exists x. \neg (F x)^r)) = (\forall \forall x. (F x)^r)$  using AAxDef by fastforce
from 1 2 3 5 6 7 show ?thesis by fastforce
qed

end

```

## 8 Projection operator

```

theory Projection
imports Fuse TimeReversal
begin

```

This theory introduces the projection operator [4]. The projection operator is defined and we prove the soundness of the rules and axiom system.

### 8.1 Definitions

```

primrec filt :: 'a interval  $\Rightarrow$  index  $\Rightarrow$  'a interval
  where filt  $\sigma \langle l \rangle = \langle \text{inth } \sigma l \rangle$ 
  | filt  $\sigma (x \odot ls) = (\text{inth } \sigma x) \odot \text{filt } \sigma ls$ 

primrec lsum :: 'a interval interval  $\Rightarrow$  nat  $\Rightarrow$  index
  where lsum  $\langle xs \rangle a = \langle a + (\text{ilen } xs) \rangle$ 
  | lsum  $(xs \odot xxs) a = (a + (\text{ilen } xs)) \odot (\text{lsum } xxs (a + (\text{ilen } xs)))$ 

definition addzero :: index  $\Rightarrow$  index
  where addzero  $ls = (\text{if } \text{ilen } ls = 0 \text{ then}$ 
     $(\text{if } \text{ifirst } ls = 0 \text{ then } ls \text{ else } 0 \odot ls) \text{ else } 0 \odot ls)$ 

definition powerinterval :: ('a::world) formula  $\Rightarrow$  'a interval  $\Rightarrow$  index  $\Rightarrow$  bool
  where powerinterval  $F \sigma l = (\forall i. i < \text{ilen } l \longrightarrow ((\text{sub } (\text{inth } l i) (\text{inth } l (\text{Suc } i)) \sigma) \models F))$ 

definition cpl :: ('a::world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a interval  $\Rightarrow$  index
  where cpl  $f g \sigma = (\epsilon l. \text{index-sequence } 0 l \wedge \text{powerinterval } f \sigma l \wedge$ 
     $(\text{inth } l (\text{ilen } l)) = \text{ilen } \sigma \wedge ((\text{filt } \sigma l) \models g))$ 

primrec lcpl :: ('a::world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a interval  $\Rightarrow$  index  $\Rightarrow$  nat interval interval
  where lcpl  $f g \sigma \langle x \rangle = \langle (\text{imap } (\text{shift } x) (\text{cpl } f g (\text{sub } x x \sigma))) \rangle$ 
  | lcpl  $f g \sigma (x \odot xs) =$ 
     $(\text{case } xs \text{ of } \langle y \rangle \Rightarrow \langle (\text{imap } (\text{shift } x) (\text{cpl } f g (\text{sub } x y \sigma))) \rangle$ 
     $| y \odot ys \Rightarrow (\text{imap } (\text{shift } x) (\text{cpl } f g (\text{sub } x y \sigma))) \odot (\text{lcpl } f g \sigma xs))$ 

definition projection-d :: ('a::world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula
  where projection-d  $F G \equiv \lambda s. (\exists l. \text{index-sequence } 0 l \wedge (\text{inth } l (\text{ilen } l)) = (\text{ilen } s) \wedge$ 
     $\text{powerinterval } F s l \wedge ((\text{filt } s l) \models G)$ 

```

)

#### **syntax**

*-projection-d* ::  $[lift, lift] \Rightarrow lift$  ((-  $\Delta$  -) [84,84] 83)

#### **syntax (ASCII)**

*-projection-d* ::  $[lift, lift] \Rightarrow lift$  ((- *proj* -) [84,84] 83)

#### **translations**

*-projection-d*  $\Rightarrow$  *CONST projection-d*

**definition** *upprojection-d* :: ('a:: world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula

**where** *upprojection-d*  $F\ G \equiv LIFT(\neg(F \Delta (\neg G)))$

#### **syntax**

*-upprojection-d* ::  $[lift, lift] \Rightarrow lift$  ((-  $\nabla$  -) [84,84] 83)

#### **syntax (ASCII)**

*-upprojection-d* ::  $[lift, lift] \Rightarrow lift$  ((- *uproj* -) [84,84] 83)

#### **translations**

*-upprojection-d*  $\Rightarrow$  *CONST upprojection-d*

**definition** *dp-d* :: ('a:: world) formula  $\Rightarrow$  'a formula

**where** *dp-d*  $F \equiv LIFT(\#True \Delta F)$

**definition** *bp-d* :: ('a:: world) formula  $\Rightarrow$  'a formula

**where** *bp-d*  $F \equiv LIFT(\#True \nabla F)$

#### **syntax**

*-dp-d* ::  $lift \Rightarrow lift$  ((*dp* -) [88] 87)

*-bp-d* ::  $lift \Rightarrow lift$  ((*bp* -) [88] 87)

#### **syntax (ASCII)**

*-dp-d* ::  $lift \Rightarrow lift$  ((*dp* -) [88] 87)

*-bp-d* ::  $lift \Rightarrow lift$  ((*bp* -) [88] 87)

#### **translations**

*-dp-d*  $\Rightarrow$  *CONST dp-d*

*-bp-d*  $\Rightarrow$  *CONST bp-d*

## **8.2 Lemmas**

### **8.2.1 filt Lemmas**

**lemma** *filt-ilen*:

*ilen(filt s0 l) = ilen l*

**by** (*induct l, simp, simp*)

**lemma** *filt-inth*:

```

assumes  $i \leq \text{ilen } (\text{filt } s0 \ l)$ 
shows  $(\text{inth } (\text{filt } s0 \ l) \ i) = (\text{inth } s0 \ (\text{inth } l \ i))$ 
using assms
proof (induct l arbitrary: i)
case (INil x)
then show ?case by simp
next
case (ICons x1a l)
then show ?case by simp (metis Nitpick.case-nat-unfold Suc-eq-plus1 le-diff-conv)
qed

```

**lemma** *filt-expand:*

```

 $(s1 = (\text{filt } s0 \ l)) =$ 
 $(\text{ilen } s1 = \text{ilen } l \wedge$ 
 $(\forall i \leq \text{ilen } s1. (\text{inth } s1 \ i) = (\text{inth } s0 \ (\text{inth } l \ i))))$ 
by (metis filt-ilen filt-inth interval-eq-inth-eq)

```

**lemma** *filt-fuse:*

```

 $\text{filt } xs \ (\text{fuse } l1 \ l2) = (\text{fuse } (\text{filt } xs \ l1) \ (\text{filt } xs \ l2))$ 
by (induct l1 arbitrary: l2 xs) simp-all

```

**lemma** *fuse-filt-ilen:*

```

assumes index-sequence 0 l
 $(\text{inth } l \ (\text{ilen } l)) = \text{ilen } xs$ 
shows  $\text{ilen } (\text{filt } (\text{fuse } (\text{prefix } (\text{inth } l \ n) \ xs) \ (\text{suffix } (\text{inth } l \ n) \ xs)) \ l) =$ 
 $\text{ilen } (\text{fuse } (\text{filt } (\text{prefix } (\text{inth } l \ n) \ xs) \ (\text{prefix } n \ l))$ 
 $(\text{filt } (\text{suffix } (\text{inth } l \ n) \ xs) \ (\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l) \ )))$ 
proof –
have 1:  $\text{ilen } (\text{filt } (\text{fuse } (\text{prefix } (\text{inth } l \ n) \ xs) \ (\text{suffix } (\text{inth } l \ n) \ xs)) \ l) = \text{ilen } l$ 
by (simp add: filt-ilen)
have 2:  $\text{ilen } (\text{fuse } (\text{filt } (\text{prefix } (\text{inth } l \ n) \ xs) \ (\text{prefix } n \ l))$ 
 $(\text{filt } (\text{suffix } (\text{inth } l \ n) \ xs) \ (\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l) \ ))) =$ 
 $(\text{ilen}((\text{filt } (\text{prefix } (\text{inth } l \ n) \ xs) \ (\text{prefix } n \ l))) +$ 
 $\text{ilen}(\text{filt } (\text{suffix } (\text{inth } l \ n) \ xs) \ (\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l) \ )))$ 
using fuse-ilen-a by blast
have 3:  $\text{ilen}((\text{filt } (\text{prefix } (\text{inth } l \ n) \ xs) \ (\text{prefix } n \ l))) = \text{ilen}(\text{prefix } n \ l)$ 
using filt-ilen by blast
have 4:  $\text{ilen}(\text{filt } (\text{suffix } (\text{inth } l \ n) \ xs) \ (\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l) \ ))) =$ 
 $\text{ilen}(\text{suffix } n \ l)$ 
by (simp add: filt-ilen)
have 5:  $\text{ilen}(\text{prefix } n \ l) + \text{ilen}(\text{suffix } n \ l) = \text{ilen } l$ 
by (simp)
from 1 2 3 4 5 show ?thesis by auto
qed

```

**lemma** *fuse-filt-inth-a:*

**assumes** *index-sequence 0 l*  
 $(\text{inth } l (\text{ilen } l)) = \text{ilen } xs$   
 $i \leq \text{ilen } l$   
 $n \leq \text{ilen } l$   
**shows**  $\text{inth } (\text{filt } (\text{fuse } (\text{prefix } (\text{inth } l n) xs) (\text{suffix } (\text{inth } l n) xs)) l) i =$   
 $\text{inth } xs (\text{inth } l i)$   
**proof** –  
**have** 1:  $\text{filt } (\text{fuse } (\text{prefix } (\text{inth } l n) xs) (\text{suffix } (\text{inth } l n) xs)) l =$   
 $\text{filt } xs l$   
**using** *assms by (metis fuse-prefix-suffix idx-less-last-1 le-neq-implies-less*  
*less-imp-le-nat not-less)*  
**have** 2:  $\text{inth } (\text{filt } xs l) i = \text{inth } xs (\text{inth } l i)$   
**using** *assms by (metis filt-inth filt-ilen )*  
**from** 1 2 **show** *?thesis by auto*  
**qed**

**lemma** *fuse-filt-inth-b:*

**assumes** *index-sequence 0 l*  
 $(\text{inth } l (\text{ilen } l)) = \text{ilen } xs$   
 $i \leq \text{ilen } l$   
 $n \leq \text{ilen } l$   
**shows**  $\text{inth } (\text{fuse } (\text{filt } (\text{prefix } (\text{inth } l n) xs) (\text{prefix } n l))$   
 $(\text{filt } (\text{suffix } (\text{inth } l n) xs) (\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l) ))) i =$   
 $\text{inth } xs (\text{inth } l i)$   
**proof** –  
**have** 1:  $i \leq \text{ilen}(\text{fuse } (\text{filt } (\text{prefix } (\text{inth } l n) xs) (\text{prefix } n l))$   
 $(\text{filt } (\text{suffix } (\text{inth } l n) xs) (\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l) )))$   
**using** *assms*  
**by** *(metis filt-ilen fuse-ilen-a fuse-prefix-suffix-ilen ilen-imap)*  
**have** 2:  $\text{ilast}(\text{filt } (\text{prefix } (\text{inth } l n) xs) (\text{prefix } n l)) = \text{inth } xs (\text{inth } l n)$   
**using** *assms filt-inth[of - (prefix (inth l n) xs) (prefix n l) ]*  
**by** *(metis filt-ilen idx-less-equal ilast-prefix prefix-ilen-good order-refl)*  
**have** 3:  $\text{ifirst}(\text{filt } (\text{suffix } (\text{inth } l n) xs) (\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l) )) =$   
 $\text{inth } (\text{filt } (\text{suffix } (\text{inth } l n) xs) (\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l) )) 0$   
**by** *simp*  
**have** 4:  $\text{inth } (\text{filt } (\text{suffix } (\text{inth } l n) xs) (\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l) )) 0 =$   
 $\text{inth } (\text{suffix } (\text{inth } l n) xs) (\text{inth } (\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l) ) 0)$   
**using** *filt-inth by blast*  
**have** 5:  $\text{inth } (\text{suffix } (\text{inth } l n) xs) (\text{inth } (\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l) ) 0) =$   
 $\text{inth } (\text{suffix } (\text{inth } l n) xs) (\text{inth } l (n+0) - (\text{inth } l n))$   
**by** *(simp add: assms inth-imap shiftn-def)*  
**have** 6:  $\text{inth } (\text{suffix } (\text{inth } l n) xs) (\text{inth } l (n+0) - (\text{inth } l n)) =$   
 $\text{inth } (\text{suffix } (\text{inth } l n) xs) 0$   
**by** *simp*  
**have** 7:  $\text{inth } (\text{suffix } (\text{inth } l n) xs) 0 = \text{inth } xs (\text{inth } l n)$   
**using** *assms by (metis Nat.add-0-right inth-suffix le0 )*  
**have** 8:  $\text{ilast}(\text{filt } (\text{prefix } (\text{inth } l n) xs) (\text{prefix } n l)) =$   
 $\text{ifirst}(\text{filt } (\text{suffix } (\text{inth } l n) xs) (\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l) ))$   
**using** 2 4 5 7 **by** *auto*

**have 10:**  $\text{inth} (\text{fuse} (\text{filt} (\text{prefix} (\text{inth } l \ n) \ xs) (\text{prefix } n \ l))$   
 $(\text{filt} (\text{suffix} (\text{inth } l \ n) \ xs) (\text{imap} (\text{shiftn} (\text{inth } l \ n)) (\text{suffix } n \ l) \ ))) \ i =$   
 $(\text{if } i \leq \text{ilen} (\text{filt} (\text{prefix} (\text{inth } l \ n) \ xs) (\text{prefix } n \ l))$   
 $\text{then } \text{inth} (\text{filt} (\text{prefix} (\text{inth } l \ n) \ xs) (\text{prefix } n \ l)) \ i$   
 $\text{else } \text{inth} (\text{filt} (\text{suffix} (\text{inth } l \ n) \ xs) (\text{imap} (\text{shiftn} (\text{inth } l \ n)) (\text{suffix } n \ l) \ )))$   
 $(i - \text{ilen} (\text{filt} (\text{prefix} (\text{inth } l \ n) \ xs) (\text{prefix } n \ l))))$   
**using 1 8 fuse-inth by auto**  
**have 11:**  $\text{ilen} (\text{filt} (\text{prefix} (\text{inth } l \ n) \ xs) (\text{prefix } n \ l)) = n$   
**using assms by (metis filt-ilen prefix-ilen-good)**  
**have 12:**  $i \leq n \longrightarrow \text{inth} (\text{filt} (\text{prefix} (\text{inth } l \ n) \ xs) (\text{prefix } n \ l)) \ i =$   
 $\text{inth} (\text{prefix} (\text{inth } l \ n) \ xs) (\text{inth} (\text{prefix } n \ l) \ i)$   
**by (simp add: 11 filt-inth)**  
**have 13:**  $i \leq n \longrightarrow \text{inth} (\text{prefix} (\text{inth } l \ n) \ xs) (\text{inth} (\text{prefix } n \ l) \ i) =$   
 $\text{inth} (\text{prefix} (\text{inth } l \ n) \ xs) (\text{inth } l \ i)$   
**by (simp add: assms)**  
**have 15:**  $i \leq n \longrightarrow \text{inth} (\text{prefix} (\text{inth } l \ n) \ xs) (\text{inth } l \ i) = \text{inth } xs (\text{inth } l \ i)$   
**using assms(1) assms(2) assms(4) idx-less-equal inth-prefix by blast**  
**have 16:**  $\text{inth} (\text{filt} (\text{suffix} (\text{inth } l \ n) \ xs) (\text{imap} (\text{shiftn} (\text{inth } l \ n)) (\text{suffix } n \ l) \ )))$   
 $(i - \text{ilen} (\text{filt} (\text{prefix} (\text{inth } l \ n) \ xs) (\text{prefix } n \ l)))) =$   
 $\text{inth} (\text{filt} (\text{suffix} (\text{inth } l \ n) \ xs) (\text{imap} (\text{shiftn} (\text{inth } l \ n)) (\text{suffix } n \ l) \ )))$   
 $(i - n)$   
**by (simp add: 11)**  
**have 17:**  $\text{inth} (\text{filt} (\text{suffix} (\text{inth } l \ n) \ xs) (\text{imap} (\text{shiftn} (\text{inth } l \ n)) (\text{suffix } n \ l) \ ))) (i - n) =$   
 $\text{inth} (\text{suffix} (\text{inth } l \ n) \ xs) (\text{inth} (\text{imap} (\text{shiftn} (\text{inth } l \ n)) (\text{suffix } n \ l) \ )) (i - n))$   
**using assms filt-inth[of  $i - n$  ( $\text{suffix} (\text{inth } l \ n) \ xs$ ) ( $\text{imap} (\text{shiftn} (\text{inth } l \ n)) (\text{suffix } n \ l) \ )$  ]**  
**by (simp add: filt-ilen)**  
**have 18:**  $i > n \longrightarrow \text{inth} (\text{imap} (\text{shiftn} (\text{inth } l \ n)) (\text{suffix } n \ l) \ ) (i - n) =$   
 $\text{inth } l \ (n + (i - n)) - (\text{inth } l \ n)$   
**using assms**  
**by (simp add: idx-shiftn-suffix-inth)**  
**have 19:**  $i > n \longrightarrow \text{inth} (\text{suffix} (\text{inth } l \ n) \ xs) (\text{inth} (\text{imap} (\text{shiftn} (\text{inth } l \ n)) (\text{suffix } n \ l) \ ) (i - n))$   
 $= \text{inth} (\text{suffix} (\text{inth } l \ n) \ xs) ((\text{inth } l \ i) - (\text{inth } l \ n))$   
**by (simp add: 18)**  
**have 20 :**  $i > n \longrightarrow \text{inth} (\text{suffix} (\text{inth } l \ n) \ xs) ((\text{inth } l \ i) - (\text{inth } l \ n)) =$   
 $\text{inth } xs ((\text{inth } l \ n) + ((\text{inth } l \ i) - (\text{inth } l \ n)))$   
**using assms by (metis diff-le-mono idx-less-last-1 inth-suffix**  
 $\text{le-neq-implies-less less-imp-le-nat nat-le-linear})$   
**have 22:**  $i > n \longrightarrow (\text{inth } l \ n) + ((\text{inth } l \ i) - (\text{inth } l \ n)) = (\text{inth } l \ i)$   
**using assms using idx-less-equal by fastforce**  
**have 23:**  $i > n \longrightarrow \text{inth } xs ((\text{inth } l \ n) + ((\text{inth } l \ i) - (\text{inth } l \ n))) = \text{inth } xs (\text{inth } l \ i)$   
**by (simp add: 22)**  
**from 10 show ?thesis by (simp add: 11 12 13 15 17 19 20 23)**  
**qed**

**lemma fuse-filt-inth:**

**assumes**  $\text{index-sequence } 0 \ l$   
 $(\text{inth } l \ (\text{ilen } l)) = \text{ilen } xs$   
 $i \leq \text{ilen } l$   
 $n \leq \text{ilen } l$

**shows**  $\text{inth} (\text{filt} (\text{fuse} (\text{prefix} (\text{inth } l \ n) \ xs) (\text{suffix} (\text{inth } l \ n) \ xs)) \ l) \ i =$

$inth (fuse (filt (prefix (inth l n) xs) (prefix n l))$   
 $(filt (suffix (inth l n) xs) (imap (shiftn (inth l n)) (suffix n l) ))) i$   
**using** *assms fuse-filt-inth-a*[of *l xs i n*]  
 $fuse-filt-inth-b$ [of *l xs i n*] **by** *simp*

**lemma** *fuse-filt*:

**assumes** *index-sequence 0 l*  
 $(inth l (ilen l)) = ilen xs$   
 $n \leq ilen l$   
**shows**  $filt (fuse (prefix (inth l n) xs) (suffix (inth l n) xs)) l =$   
 $fuse (filt (prefix (inth l n) xs) (prefix n l))$   
 $(filt (suffix (inth l n) xs) (imap (shiftn (inth l n)) (suffix n l) ))$   
**using** *assms fuse-filt-ilen fuse-filt-inth interval-eq-inth-eq* **by** (*metis filt-ilen*)

**lemma** *fuse-filt-a*:

**assumes** *index-sequence 0 l1*  
 $index-sequence (ilast l1) l2$   
 $ilast l2 = ilen xs$   
**shows**  $filt (fuse (prefix (ilast l1) xs) (suffix (ifirst l2) xs)) (fuse l1 l2) =$   
 $fuse (filt (prefix (ilast l1) xs) l1)$   
 $(filt (suffix (ifirst l2) xs) (imap (shiftn (ifirst l2)) l2))$   
**proof** –  
**have** 1:  $ilast l1 = ifirst l2$   
**using** *assms* **by** (*metis index-sequence-def*)  
**have** 2: *index-sequence 0 (fuse l1 l2)*  
**using** *assms* **by** (*metis index-sequence-def idx-fuse*)  
**have** 3:  $ilast (fuse l1 l2) = ilen xs$   
**using** *assms* **by** (*metis 1 add.left-neutral fuse-inth-a fuse-ilen-a le-add2*)  
**have** 4:  $ilen l1 \leq ilen (fuse l1 l2)$   
**by** (*simp add: fuse-ilen-a*)  
**have** 5:  $filt (fuse (prefix (inth (fuse l1 l2) (ilen l1)) xs)$   
 $(suffix (inth (fuse l1 l2) (ilen l1)) xs))$   
 $(fuse l1 l2)$   
 $=$   
 $fuse (filt (prefix (inth (fuse l1 l2) (ilen l1)) xs) (prefix (ilen l1) (fuse l1 l2)))$   
 $(filt (suffix (inth (fuse l1 l2) (ilen l1)) xs)$   
 $(imap (shiftn (inth (fuse l1 l2) (ilen l1))) (suffix (ilen l1) (fuse l1 l2))))$   
**using** 2 3 4 *fuse-filt* **by** *auto*  
**have** 6:  $filt (fuse (prefix (ilast l1) xs) (suffix (ifirst l2) xs)) (fuse l1 l2) =$   
 $filt (fuse (prefix (inth (fuse l1 l2) (ilen l1)) xs)$   
 $(suffix (inth (fuse l1 l2) (ilen l1)) xs))$   
 $(fuse l1 l2)$   
**using** 1 4 *ilast-prefix*[of *ilast l1 fuse l1 l2*]  
*prefix-fuse* **by** (*simp add: fuse-inth*)  
**have** 7:  $(filt (prefix (ilast l1) xs) l1) =$   
 $(filt (prefix (inth (fuse l1 l2) (ilen l1)) xs) (prefix (ilen l1) (fuse l1 l2)))$   
**using** 1 4 *ilast-prefix*[of *(ilast l1) fuse l1 l2*]  
*prefix-fuse*  
**by** (*simp add: prefix-fuse fuse-inth*)

**have** 8: (filt (suffix (ifirst l2) xs) (imap (shiftn (ifirst l2)) l2)) =  
 (filt (suffix (inth (fuse l1 l2) (ilen l1)) xs)  
 (imap (shiftn (inth (fuse l1 l2) (ilen l1))) (suffix (ilen l1) (fuse l1 l2)) ))  
**using** 1 4 ilast-prefix prefix-fuse suffix-fuse **by** metis  
**show** ?thesis **using** 5 6 7 8 **by** auto  
**qed**

**lemma** filt-prefix:  
**assumes**  $n \leq \text{ilen } l$   
**shows**  $\text{prefix } n (\text{filt } \sigma \ l) = \text{filt } \sigma (\text{prefix } n \ l)$   
**proof** –  
**have** 1:  $\text{ilen } (\text{prefix } n (\text{filt } \sigma \ l)) = \text{ilen } (\text{filt } \sigma (\text{prefix } n \ l))$   
**by** (simp add: assms filt-ilen)  
**have** 2:  $\forall i \leq \text{ilen}(\text{prefix } n (\text{filt } \sigma \ l)).$   
 $(\text{inth } (\text{prefix } n (\text{filt } \sigma \ l)) \ i) = (\text{inth } (\text{filt } \sigma (\text{prefix } n \ l)) \ i)$   
**by** (simp add: assms filt-inth filt-ilen le-trans)  
**show** ?thesis **using** 1 2 interval-eq-inth-eq **by** blast  
**qed**

**lemma** filt-prefix-idx:  
**assumes**  $n \leq \text{ilen } \sigma$   
 $\text{index-sequence } 0 \ l$   
 $\text{inth } l (\text{ilen } l) = n$   
**shows**  $\text{filt } \sigma \ l = \text{filt } (\text{prefix } n \ \sigma) \ l$   
**proof** –  
**have** 1:  $\text{ilen}(\text{filt } \sigma \ l) = \text{ilen } (\text{filt } (\text{prefix } n \ \sigma) \ l)$   
**by** (simp add: filt-ilen)  
**have** 2:  $\forall i \leq \text{ilen}(\text{filt } \sigma \ l). (\text{inth } (\text{filt } \sigma \ l) \ i) = (\text{inth } (\text{filt } (\text{prefix } n \ \sigma) \ l) \ i)$   
**by** (metis assms filt-inth filt-ilen idx-less-equal inth-prefix prefix-ilen-good order-refl)  
**show** ?thesis  
**by** (simp add: 1 2 interval-eq-inth-eq)  
**qed**

**lemma** filt-suffix:  
**assumes**  $n \leq \text{ilen } l$   
**shows**  $\text{suffix } n (\text{filt } \sigma \ l) = \text{filt } \sigma (\text{suffix } n \ l)$   
**proof** –  
**have** 1:  $\text{ilen } (\text{suffix } n (\text{filt } \sigma \ l)) = \text{ilen } (\text{filt } \sigma (\text{suffix } n \ l))$   
**by** (simp add: assms filt-ilen)  
**have** 2:  $\forall i \leq \text{ilen } (\text{suffix } n (\text{filt } \sigma \ l)).$   
 $(\text{inth } (\text{suffix } n (\text{filt } \sigma \ l)) \ i) = (\text{inth } (\text{filt } \sigma (\text{suffix } n \ l)) \ i)$   
**by** (simp add: assms filt-inth filt-ilen  
 ordered-cancel-comm-monoid-diff-class.le-diff-conv2)  
**show** ?thesis **using** 1 2 interval-eq-inth-eq **by** blast  
**qed**

**lemma** filt-suffix-idx-ilen:



**assumes**  $n \leq \text{ilen } \sigma$   
 $\text{index-sequence } 0 \ l$   
 $\text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma - n$   
**shows**  $\text{filt } \sigma \ (\text{imap } (\text{shift } n) \ l) = \text{ilen } (\text{filt } (\text{suffix } n \ \sigma) \ l)$   
**using** *assms* **by** (*simp add: filt-ilen*)

**lemma** *filt-suffix-idx-inth*:  
**assumes**  $n \leq \text{ilen } \sigma$   
 $\text{index-sequence } 0 \ l$   
 $i \leq \text{ilen } l$   
 $\text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma - n$   
**shows**  $\text{inth } (\text{filt } \sigma \ (\text{imap } (\text{shift } n) \ l)) \ i = \text{inth } (\text{filt } (\text{suffix } n \ \sigma) \ l) \ i$   
**proof** –  
**have** 1:  $\text{inth } (\text{filt } \sigma \ (\text{imap } (\text{shift } n) \ l)) \ i = \text{inth } \sigma \ (\text{inth } (\text{imap } (\text{shift } n) \ l) \ i)$   
**by** (*simp add: assms filt-inth filt-ilen*)  
**have** 2:  $\text{inth } \sigma \ (\text{inth } (\text{imap } (\text{shift } n) \ l) \ i) = (\text{inth } \sigma \ ((\text{inth } l \ i) + n))$   
**by** (*simp add: shift-def inth-imap*)  
**have** 3:  $\text{inth } (\text{filt } (\text{suffix } n \ \sigma) \ l) \ i = \text{inth } (\text{suffix } n \ \sigma) \ (\text{inth } l \ i)$   
**by** (*simp add: assms filt-inth filt-ilen*)  
**have** 4:  $\text{inth } (\text{suffix } n \ \sigma) \ (\text{inth } l \ i) = (\text{inth } \sigma \ ((\text{inth } l \ i) + n))$   
**by** (*metis add.commute assms eq-iff idx-less-equal inth-suffix suffix-ilen-good*)  
**show** ?thesis **by** (*simp add: 1 2 3 4*)  
**qed**

**lemma** *filt-suffix-idx*:  
**assumes**  $n \leq \text{ilen } \sigma$   
 $\text{index-sequence } 0 \ l$   
 $\text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma - n$   
**shows**  $\text{filt } \sigma \ (\text{imap } (\text{shift } n) \ l) = \text{filt } (\text{suffix } n \ \sigma) \ l$   
**using** *assms* *filt-suffix-idx-ilen* *filt-suffix-idx-inth* *interval-eq-inth-eq*  
**by** (*simp add: filt-suffix-idx-inth interval-eq-inth-eq filt-ilen*)

**lemma** *filt-filt*:  
 $(\text{inth } (\text{filt } (\text{filt } \text{xs } l1) \ l2) \ k) =$   
 $(\text{inth } \text{xs } (\text{inth } l1 \ (\text{inth } l2 \ k)))$   
**by** (*metis filt-expand ilen-imap inth-imap*)

**lemma** *filt-filt-imap*:  
 $(\text{filt } (\text{filt } \text{xs } l1) \ l2) = (\text{filt } \text{xs } (\text{imap } (\lambda x. (\text{inth } l1 \ x)) \ l2))$   
**by** (*metis filt-expand ilen-imap inth-imap*)

**lemma** *filt-imap*:  
 $\text{filt } \text{xs } l = \text{imap } (\lambda x. \text{inth } \text{xs } x) \ l$   
**by** (*metis filt-expand ilen-imap inth-imap*)

**lemma** *filt-imap-filt*:  
 $(\text{filt } (\text{filt } \text{xs } l1) \ l2) = \text{filt } \text{xs } (\text{filt } l1 \ l2)$   
**by** (*metis filt-filt-imap filt-imap*)

**lemma** *filt-sub*:  
**assumes**  $k \leq n$   
 $n \leq \text{ilen } l$   
**shows**  $(\text{sub } k \ n \ (\text{filt } \sigma \ l)) = (\text{filt } \sigma \ (\text{sub } k \ n \ l))$   
**using** *assms*  
**by** (*simp add: sub-def filt-prefix filt-suffix*)

**lemma** *filt-lfuse-imap*:  
 $(\text{filt } \sigma \ (\text{lfuse } (x\text{xs}))) =$   
 $(\text{lfuse } (\text{imap } (\lambda \text{xs} . (\text{filt } \sigma \ \text{xs}))) \ x\text{xs}))$   
**proof** (*induct xxs*)  
**case** (*INil x*)  
**then show** ?*case* **by** *simp*  
**next**  
**case** (*ICons x1a xxs*)  
**then show** ?*case* **by** (*simp add: filt-fuse*)  
**qed**

### 8.2.2 powerinterval lemmas

**lemma** *powerinterval-split0*:  
**assumes** *index-sequence 0 l*  
 $n \leq \text{ilen } l$   
 $\text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma$   
 $i < \text{ilen}(\text{prefix } n \ l)$   
**shows**  $(\text{sub } (\text{inth } (\text{prefix } n \ l) \ i) \ (\text{inth } (\text{prefix } n \ l) \ (\text{Suc } i)) \ (\text{prefix } (\text{inth } l \ n) \ \sigma)) =$   
 $(\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)$

**proof** –  
**have** 01:  $(\text{inth } l \ n) \leq \text{ilen } \sigma$   
**by** (*metis assms(1) assms(2) assms(3) idx-less-last-1 le-less*)  
**have** 02:  $(\text{inth } (\text{prefix } n \ l) \ i) \leq (\text{inth } (\text{prefix } n \ l) \ (\text{Suc } i))$   
**using** *assms(1) assms(3) assms(4) idx-expand* **by** *fastforce*  
**have** 03:  $(\text{inth } (\text{prefix } n \ l) \ (\text{Suc } i)) \leq (\text{inth } l \ n)$   
**using** *assms(1) assms(2) assms(3) assms(4) idx-less-equal* **by** *fastforce*  
**have** 1:  $\text{ilen } (\text{sub } (\text{inth } (\text{prefix } n \ l) \ i) \ (\text{inth } (\text{prefix } n \ l) \ (\text{Suc } i)) \ (\text{prefix } (\text{inth } l \ n) \ \sigma)) =$   
 $\text{ilen}(\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ (\sigma))$   
**using** *ilen-sub*  
**using** 01 02 03 *assms(4)* **by** *auto*  
**have** 2:  $(\forall j \leq \text{ilen } (\text{sub } (\text{inth } (\text{prefix } n \ l) \ i) \ (\text{inth } (\text{prefix } n \ l) \ (\text{Suc } i)) \ (\text{prefix } (\text{inth } l \ n) \ \sigma)).$   
 $(\text{inth } (\text{sub } (\text{inth } (\text{prefix } n \ l) \ i) \ (\text{inth } (\text{prefix } n \ l) \ (\text{Suc } i)) \ (\text{prefix } (\text{inth } l \ n) \ \sigma)) \ j) =$   
 $(\text{inth } (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ (\sigma)) \ j))$

**proof**  
**fix** *j*  
**show**  $j \leq \text{ilen } (\text{sub } (\text{inth } (\text{prefix } n \ l) \ i) \ (\text{inth } (\text{prefix } n \ l) \ (\text{Suc } i)) \ (\text{prefix } (\text{inth } l \ n) \ \sigma)) \longrightarrow$   
 $(\text{inth } (\text{sub } (\text{inth } (\text{prefix } n \ l) \ i) \ (\text{inth } (\text{prefix } n \ l) \ (\text{Suc } i)) \ (\text{prefix } (\text{inth } l \ n) \ \sigma)) \ j) =$   
 $(\text{inth } (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ (\sigma)) \ j)$

**proof** –

**have** 21:  $\text{ilen}(\text{sub}(\text{inth}(\text{prefix } n \ l) \ i) (\text{inth}(\text{prefix } n \ l) (\text{Suc } i)) (\text{prefix}(\text{inth } l \ n) \ \sigma)) =$   
 $(\text{inth } l (\text{Suc } i)) - (\text{inth } l \ i)$   
**using** 1 *assms*(1) *assms*(3) *assms*(4) *idx-expand* **by** *fastforce*  
**have** 22:  $(\text{inth}(\text{prefix } n \ l) \ i) = (\text{inth } l \ i)$   
**using** *assms* **by** *auto*  
**have** 23:  $(\text{inth}(\text{prefix } n \ l) (\text{Suc } i)) = (\text{inth } l (\text{Suc } i))$   
**using** *assms* **by** *auto*  
**have** 24:  $j \leq (\text{inth } l (\text{Suc } i)) - (\text{inth } l \ i) \longrightarrow$   
 $\text{inth}(\text{sub}(\text{inth } l \ i) (\text{inth } l (\text{Suc } i)) \ \sigma) \ j = (\text{inth } \sigma ((\text{inth } l \ i) + j))$   
**using** *assms* *idx-expand* *inth-sub* *less-le-trans* **by** *fastforce*  
**have** 25:  $j \leq (\text{inth } l (\text{Suc } i)) - (\text{inth } l \ i) \longrightarrow (\text{inth } l (\text{Suc } i)) \leq \text{ilen}(\text{prefix}(\text{inth } l \ n) \ \sigma)$   
**using** *assms*(1) *assms*(2) *assms*(3) *assms*(4) *idx-expand* *idx-less-equal*  
**by** *fastforce*  
**have** 26:  $j \leq (\text{inth } l (\text{Suc } i)) - (\text{inth } l \ i) \longrightarrow$   
 $(\text{inth}(\text{sub}(\text{inth}(\text{prefix } n \ l) \ i) (\text{inth}(\text{prefix } n \ l) (\text{Suc } i)) (\text{prefix}(\text{inth } l \ n) \ \sigma)) \ j =$   
 $(\text{inth}(\text{prefix}(\text{inth } l \ n) \ \sigma) ((\text{inth } l \ i) + j))$   
**using** 02 22 23 25 **by** *auto*  
**have** 27:  $(\text{inth } l (\text{Suc } i)) \leq (\text{inth } l \ n)$   
**using** *assms*(1) *assms*(2) *assms*(3) *assms*(4) *idx-less-equal* **by** *fastforce*  
**have** 28:  $j \leq (\text{inth } l (\text{Suc } i)) - (\text{inth } l \ i) \longrightarrow (\text{inth } l \ i) + j \leq (\text{inth } l \ n)$   
**using** 27 *assms*(1) *assms*(3) *assms*(4) *idx-expand* **by** *fastforce*  
**have** 29:  $j \leq (\text{inth } l (\text{Suc } i)) - (\text{inth } l \ i) \longrightarrow$   
 $(\text{inth}(\text{prefix}(\text{inth } l \ n) \ \sigma) ((\text{inth } l \ i) + j)) = (\text{inth } \sigma ((\text{inth } l \ i) + j))$   
**using** *assms* *inth-prefix* **using** 28 **by** *blast*  
**show** ?thesis  
**using** 21 22 23 24 26 29 **by** *auto*  
**qed**  
**qed**  
**show** ?thesis  
**using** 1 2 *interval-eq-inth-eq* **by** *blast*  
**qed**

**lemma** *powerinterval-split*:

**assumes** *index-sequence* 0 *l*

$n \leq \text{ilen } l$

$\text{inth } l (\text{ilen } l) = \text{ilen } \sigma$

*powerinterval* *f*  $\sigma$  *l*

**shows** *powerinterval* *f*  $(\text{prefix}(\text{inth } l \ n) \ \sigma) (\text{prefix } n \ l)$

**proof** –

**have** 0:  $(\forall \ i. \ i < \text{ilen } l \longrightarrow ((\text{sub}(\text{inth } l \ i) (\text{inth } l (\text{Suc } i)) \ \sigma) \models f))$

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

**have** 1: *powerinterval* *f*  $(\text{prefix}(\text{inth } l \ n) \ \sigma) (\text{prefix } n \ l) =$

$(\forall \ i. \ i < \text{ilen}(\text{prefix } n \ l) \longrightarrow$

$((\text{sub}(\text{inth}(\text{prefix } n \ l) \ i) (\text{inth}(\text{prefix } n \ l) (\text{Suc } i)) (\text{prefix}(\text{inth } l \ n) \ \sigma)) \models f))$

**using** *powerinterval-def* **by** *blast*

**have** 2:  $(\forall \ i. \ i < \text{ilen}(\text{prefix } n \ l) \longrightarrow$

$((\text{sub}(\text{inth}(\text{prefix } n \ l) \ i) (\text{inth}(\text{prefix } n \ l) (\text{Suc } i)) (\text{prefix}(\text{inth } l \ n) \ \sigma)) \models f))$

**proof**

**fix** *i*

```

show  $i < \text{ilen}(\text{prefix } n \ l) \longrightarrow$ 
   $((\text{sub } (\text{inth } (\text{prefix } n \ l) \ i) \ (\text{inth } (\text{prefix } n \ l) \ (\text{Suc } i)) \ (\text{prefix } (\text{inth } l \ n) \ \sigma)) \models f)$ 
proof –
  have 21:  $i < n \longrightarrow ((\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma) \models f)$ 
    using 0 assms less-le-trans by blast
  have 22:  $i < \text{ilen}(\text{prefix } n \ l) \longrightarrow$ 
     $((\text{sub } (\text{inth } (\text{prefix } n \ l) \ i) \ (\text{inth } (\text{prefix } n \ l) \ (\text{Suc } i)) \ \sigma) \models f)$ 
    by (simp add: 21 assms)
  have 23:  $i < \text{ilen}(\text{prefix } n \ l) \longrightarrow$ 
     $((\text{sub } (\text{inth } (\text{prefix } n \ l) \ i) \ (\text{inth } (\text{prefix } n \ l) \ (\text{Suc } i)) \ (\text{prefix } (\text{inth } l \ n) \ \sigma)) \models f)$ 
    using 22 assms powerinterval-split0 by fastforce
  show ?thesis using 23 by blast
qed
qed
show ?thesis using 1 2 by blast
qed

lemma powerinterval-splitb0:
assumes index-sequence 0  $l$ 
   $n \leq \text{ilen } l$ 
   $\text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma$ 
   $i < (\text{ilen } l - n)$ 
shows  $(\text{sub } (\text{inth } (((\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l)))) \ i)$ 
   $(\text{inth } (((\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l)))) \ (\text{Suc } i))$ 
   $(\text{suffix } (\text{inth } l \ n) \ \sigma) =$ 
   $(\text{sub } (\text{inth } l \ (i+n)) \ (\text{inth } l \ ((\text{Suc } i)+n)) \ \sigma)$ 
proof –
  have 1:  $\text{ilen}(((\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l)))) = (\text{ilen } l) - n$ 
    by (simp add: assms)
  have 2:  $i < (\text{ilen } l - n) \longrightarrow$ 
     $(\text{inth } (((\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l)))) \ i) =$ 
     $(\text{inth } l \ (n + i)) - (\text{inth } l \ n)$ 
    using assms idx-shiftn-suffix-inth by force
  have 3:  $i < (\text{ilen } l - n) \longrightarrow$ 
     $(\text{inth } (((\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l)))) \ (\text{Suc } i)) =$ 
     $(\text{inth } l \ (n + (\text{Suc } i))) - (\text{inth } l \ n)$ 
    using assms idx-shiftn-suffix-inth by fastforce
  have 4:  $i < (\text{ilen } l - n) \longrightarrow$ 
     $(\text{sub } (\text{inth } (((\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l)))) \ i)$ 
     $(\text{inth } (((\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l)))) \ (\text{Suc } i))$ 
     $(\text{suffix } (\text{inth } l \ n) \ \sigma) =$ 
     $(\text{sub } (\text{inth } l \ (n + i)) - (\text{inth } l \ n))$ 
     $(\text{inth } l \ (n + (\text{Suc } i))) - (\text{inth } l \ n)$ 
     $(\text{suffix } (\text{inth } l \ n) \ \sigma)$ 
    by (simp add: 2 3)
  have 5:  $i < (\text{ilen } l - n) \longrightarrow (\text{inth } l \ (n + i)) < (\text{inth } l \ (n + (\text{Suc } i)))$ 
    using assms index-sequence-def by auto
  have 6:  $i < (\text{ilen } l - n) \longrightarrow (\text{inth } l \ n) \leq (\text{inth } l \ (n + i))$ 
    using assms by (metis add.commute idx-less-equal le-add1 less-diff-conv less-imp-le-nat)
  have 7:  $i < (\text{ilen } l - n) \longrightarrow (\text{inth } l \ n) \leq (\text{inth } l \ (n + (\text{Suc } i)))$ 

```

using 5 6 by *linarith*  
**have** 8:  $i < (\text{ilen } l - n) \longrightarrow (\text{inth } l (n + i)) - (\text{inth } l n) < (\text{inth } l (n + (\text{Suc } i))) - (\text{inth } l n)$   
 using 5 6 *diff-less-mono* by *blast*  
**have** 9:  $i < (\text{ilen } l - n) \longrightarrow (\text{inth } l (n + (\text{Suc } i))) - (\text{inth } l n) \leq \text{ilen } \sigma - (\text{inth } l n)$   
 by (*metis Suc-leI add.commute assms diff-le-mono idx-less-last-1 le-eq-less-or-eq less-diff-conv ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)  
**have** 10:  $i < (\text{ilen } l - n) \longrightarrow$   

$$\begin{aligned} & (\text{sub } ( (\text{inth } l (n + i)) - (\text{inth } l n) ) \\ & \quad ( (\text{inth } l (n + (\text{Suc } i))) - (\text{inth } l n) ) \\ & \quad (\text{suffix } (\text{inth } l n) \sigma)) = \\ & (\text{sub } (\text{inth } l (i+n)) (\text{inth } l ((\text{Suc } i)+n)) \sigma) \end{aligned}$$
  
 using *sub-suffix*  
 by (*metis 6 7 8 9 add.commute le-add-diff-inverse2*)  
**show** *?thesis*  
 using 2 3 10 *assms* by *auto*  
**qed**

**lemma** *powerinterval-splitb*:

**assumes** *index-sequence 0 l*

$n \leq \text{ilen } l$

$\text{inth } l (\text{ilen } l) = \text{ilen } \sigma$

*powerinterval f σ l*

**shows** *powerinterval f (suffix (inth l n) σ) ((imap (shiftn (inth l n)) (suffix n l)))*

**proof** –

**have** 0:  $(\forall i. i < \text{ilen } l \longrightarrow ( (\text{sub } (\text{inth } l i) (\text{inth } l (\text{Suc } i)) \sigma) \models f ))$

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

**have** 1: *powerinterval f (suffix (inth l n) σ) ((imap (shiftn (inth l n)) (suffix n l))) =*  

$$\begin{aligned} & (\forall i. i < \text{ilen}(((\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l)))) \longrightarrow \\ & \quad ( (\text{sub } (\text{inth } (((\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l)))) i) \\ & \quad \quad (\text{inth } (((\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l)))) (\text{Suc } i)) \\ & \quad \quad (\text{suffix } (\text{inth } l n) \sigma)) \models f )) \end{aligned}$$

by (*simp add: powerinterval-def*)

**have** 2:  $(\forall i. i < \text{ilen}(((\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l)))) \longrightarrow$   

$$\begin{aligned} & \quad ( (\text{sub } (\text{inth } (((\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l)))) i) \\ & \quad \quad (\text{inth } (((\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l)))) (\text{Suc } i)) \\ & \quad \quad (\text{suffix } (\text{inth } l n) \sigma)) \models f )) \end{aligned}$$

**proof**

**fix** *i*

**show**  $i < \text{ilen}(((\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l)))) \longrightarrow$   

$$\begin{aligned} & \quad ( (\text{sub } (\text{inth } (((\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l)))) i) \\ & \quad \quad (\text{inth } (((\text{imap } (\text{shiftn } (\text{inth } l n)) (\text{suffix } n l)))) (\text{Suc } i)) \\ & \quad \quad (\text{suffix } (\text{inth } l n) \sigma)) \models f \end{aligned}$$

**proof** –

**have** 21:  $i < (\text{ilen } l) - n \longrightarrow$   

$$((\text{sub } (\text{inth } l (i+n)) (\text{inth } l ((\text{Suc } i)+n)) \sigma) \models f)$$

by (*simp add: 0*)

**show** *?thesis*

using 21 *assms powerinterval-splitb0* by *fastforce*

**qed**

**qed**

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

**lemma** *powerinterval-split*:

**assumes** *index-sequence* 0 *l*

$n \leq \text{ilen } l$

$\text{inth } l (\text{ilen } l) = \text{ilen } \sigma$

**shows**  $\text{powerinterval } f \ \sigma \ l =$

$(\text{powerinterval } f (\text{prefix } (\text{inth } l \ n) \ \sigma) (\text{prefix } n \ l) \wedge$   
 $\text{powerinterval } f (\text{suffix } (\text{inth } l \ n) \ \sigma) ((\text{imap } (\text{shiftn } (\text{inth } l \ n)) (\text{suffix } n \ l))))$

**proof** –

**have** 1:  $\text{powerinterval } f \ \sigma \ l \implies$

$\text{powerinterval } f (\text{prefix } (\text{inth } l \ n) \ \sigma) (\text{prefix } n \ l)$

**by** (*simp add: assms powerinterval-splita*)

**have** 2:  $\text{powerinterval } f \ \sigma \ l \implies$

$\text{powerinterval } f (\text{suffix } (\text{inth } l \ n) \ \sigma) ((\text{imap } (\text{shiftn } (\text{inth } l \ n)) (\text{suffix } n \ l)))$

**by** (*simp add: assms powerinterval-splitb*)

**have** 3:  $\text{powerinterval } f (\text{prefix } (\text{inth } l \ n) \ \sigma) (\text{prefix } n \ l) =$

$(\forall i. i < n \longrightarrow ((\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma) \models f))$

**by** (*metis assms prefix-ilen-good powerinterval-def powerinterval-splita0*)

**have** 4:  $\text{powerinterval } f (\text{suffix } (\text{inth } l \ n) \ \sigma) ((\text{imap } (\text{shiftn } (\text{inth } l \ n)) (\text{suffix } n \ l))) =$

$(\forall i. i < (\text{ilen } l) - n \longrightarrow ((\text{sub } (\text{inth } l \ (i+n)) (\text{inth } l \ ((\text{Suc } i)+n)) \ \sigma) \models f))$

**by** (*simp add: assms powerinterval-def powerinterval-splitb0*)

**have** 5:  $(\forall i. i < (\text{ilen } l) - n \longrightarrow ((\text{sub } (\text{inth } l \ (i+n)) (\text{inth } l \ ((\text{Suc } i)+n)) \ \sigma) \models f)) =$

$(\forall i. n \leq i \wedge i < (\text{ilen } l) \longrightarrow ((\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma) \models f))$

**by** (*metis le-add2 le-add-diff-inverse2 less-diff-conv plus-nat.simps(2)*)

**have** 5:  $(\text{powerinterval } f (\text{prefix } (\text{inth } l \ n) \ \sigma) (\text{prefix } n \ l) \wedge$

$\text{powerinterval } f (\text{suffix } (\text{inth } l \ n) \ \sigma) ((\text{imap } (\text{shiftn } (\text{inth } l \ n)) (\text{suffix } n \ l)))) \implies$

$\text{powerinterval } f \ \sigma \ l$

**using** 3 4 5 *assms powerinterval-def*

**by** (*metis not-less-eq not-less-less-Suc-eq order.order-iff-strict*)

**show** *?thesis*

**using** 1 2 5 **by** *blast*

**qed**

**lemma** *powerinterval-fuse*:

**assumes** *index-sequence* 0 *l1*

*index-sequence* 0 *l2*

*ilast* *l1* = *cp*

$cp \leq \text{ilen } \sigma$

$\text{ilast } l2 = \text{ilen } \sigma - cp$

**shows**  $\text{powerinterval } f \ \sigma \ (\text{fuse } l1 \ (\text{imap } (\text{shift } cp) \ l2)) =$

$(\text{powerinterval } f (\text{prefix } cp \ \sigma) \ l1 \wedge$

$\text{powerinterval } f (\text{suffix } cp \ \sigma) \ l2)$

**proof** –

**have** 1: *index-sequence* 0 (*fuse* *l1* (*imap* (*shift* *cp*) *l2*))

**using** *assms idx-fuse*[of *l1* (*imap* (*shift* *cp*) *l2*)]

**by** (*metis index-sequence-def idx-link*)

**have** 2:  $cp = (\text{inth } l1 (\text{ilen } l1))$

**using** *assms inth-ilen-ilast* **by** *blast*

```

have 3:  $ilen\ l1 \leq ilen\ (fuse\ l1\ (imap\ (shift\ cp)\ l2))$ 
  by (simp add: fuse-ilen-a)
have 4:  $inth\ (fuse\ l1\ (imap\ (shift\ cp)\ l2))\ (ilen\ (fuse\ l1\ (imap\ (shift\ cp)\ l2))) = ilen\ \sigma$ 
  by (metis assms idx-fuse-ilen ilen-gr-zero)
have 5:  $cp = inth\ (fuse\ l1\ (imap\ (shift\ cp)\ l2))\ (ilen\ l1)$ 
  by (metis 3 assms fuse-inth idx-fuse-ifirst-ilast order-refl)
have 6:  $(imap\ (shiftn\ cp)\ (imap\ (shift\ cp)\ l2)) = l2$ 
  using assms by (metis idx-link lsk-ls)
have 7:  $ilast\ l1 = ifirst\ (imap\ (shift\ cp)\ l2)$ 
  by (metis assms idx-fuse-ifirst-ilast)
show ?thesis
using 1 3 4 5 6 7 prefix-fuse suffix-fuse powerinterval-split
by fastforce
qed

```

**lemma** *powerinterval-idx:*

```

(powerinterval (LIFT( $f \wedge more$ ))  $\sigma\ l \wedge (inth\ l\ 0) = 0 \wedge (inth\ l\ (ilen\ l)) = ilen\ \sigma =$ 
 $(index-sequence\ 0\ l \wedge (inth\ l\ (ilen\ l)) = ilen\ \sigma \wedge powerinterval\ f\ \sigma\ l)$ )
proof (auto simp add: powerinterval-def index-sequence-def more-defs)
show  $\bigwedge n. \forall i < ilen\ l.$ 
   $f\ (sub\ (inth\ l\ i)\ (inth\ l\ (Suc\ i))\ \sigma) \wedge$ 
   $0 < ilen\ (sub\ (inth\ l\ i)\ (inth\ l\ (Suc\ i))\ \sigma) \implies$ 
   $inth\ l\ 0 = 0 \implies$ 
   $inth\ l\ (ilen\ l) = ilen\ \sigma \implies n < ilen\ l \implies$ 
   $inth\ l\ n < inth\ l\ (Suc\ n)$ 
  by (simp add: sub-def)
show  $\bigwedge i. inth\ l\ 0 = 0 \implies$ 
   $\forall n < ilen\ l. inth\ l\ n < inth\ l\ (Suc\ n) \implies$ 
   $inth\ l\ (ilen\ l) = ilen\ \sigma \implies$ 
   $\forall i < ilen\ l. f\ (sub\ (inth\ l\ i)\ (inth\ l\ (Suc\ i))\ \sigma) \implies$ 
   $i < ilen\ l \implies 0 < ilen\ (sub\ (inth\ l\ i)\ (inth\ l\ (Suc\ i))\ \sigma)$ 
  by (simp add: sub-def)
  (metis index-sequence-def idx-less-last-1)
qed

```

### 8.2.3 cpl lemmas

**lemma** *cpl-expand:*

```

assumes ( $\exists l. index-sequence\ 0\ l \wedge powerinterval\ f\ \sigma\ l \wedge (inth\ l\ (ilen\ l)) = ilen\ \sigma \wedge$ 
 $((filt\ \sigma\ l) \models g)$ )
shows  $index-sequence\ 0\ (cpl\ f\ g\ \sigma) \wedge powerinterval\ f\ \sigma\ (cpl\ f\ g\ \sigma) \wedge$ 
 $(inth\ (cpl\ f\ g\ \sigma)\ (ilen\ (cpl\ f\ g\ \sigma))) = ilen\ \sigma \wedge ((filt\ \sigma\ (cpl\ f\ g\ \sigma)) \models g)$ 
proof –
  have 0:  $cpl\ f\ g\ \sigma = (\epsilon\ l. index-sequence\ 0\ l \wedge powerinterval\ f\ \sigma\ l \wedge$ 
 $(inth\ l\ (ilen\ l)) = ilen\ \sigma \wedge ((filt\ \sigma\ l) \models g))$ 
  using cpl-def by force
  have 1: ( $\exists l. index-sequence\ 0\ l \wedge powerinterval\ f\ \sigma\ l \wedge (inth\ l\ (ilen\ l)) = ilen\ \sigma \wedge$ 
 $((filt\ \sigma\ l) \models g)$ )
  using assms by auto

```

**have** 2: *index-sequence* 0 (*cpl f g*  $\sigma$ )  $\wedge$  *powerinterval* *f*  $\sigma$  (*cpl f g*  $\sigma$ )  $\wedge$   
 (*inth* (*cpl f g*  $\sigma$ ) (*ilen* (*cpl f g*  $\sigma$ ))) = *ilen*  $\sigma \wedge ((\text{filt } \sigma \text{ (cpl f g } \sigma)) \models g)$   
**using** 0 1  
*someI-ex*[of  $\lambda l. \text{index-sequence } 0 \ l \wedge \text{powerinterval } f \ \sigma \ l \wedge (\text{inth } l \ (\text{ilen } l)) = \text{ilen } \sigma \wedge$   
 $((\text{filt } \sigma \ l) \models g)$ ] **by** *simp*  
**show** ?thesis  
**using** 2 **by** *blast*  
**qed**

**lemma** *cpl-projection*:

$(\sigma \models f \triangle g) =$   
 ( *index-sequence* 0 (*cpl f g*  $\sigma$ )  $\wedge$  *powerinterval* *f*  $\sigma$  (*cpl f g*  $\sigma$ )  $\wedge$   
 (*inth* (*cpl f g*  $\sigma$ ) (*ilen* (*cpl f g*  $\sigma$ ))) = *ilen*  $\sigma \wedge g \ (\text{filt } \sigma \text{ (cpl f g } \sigma))$  )  
**using** *cpl-expand* **by** (*simp add: projection-d-def, blast*)

**lemma** *cpl-empty*:

**assumes** *ilen*  $\sigma = 0 \wedge (\sigma \models f \triangle g)$   
**shows** (*cpl f g*  $\sigma$ ) =  $\langle 0 \rangle$   
**using** *assms cpl-projection idx-less-last-1 INil-ilen* **by** *fastforce*

**lemma** *cpl-empty-a*:

**assumes** *ilen*  $\sigma = 0$   
 (*cpl f g*  $\sigma$ ) =  $\langle 0 \rangle$   
 $g(\text{filt } \sigma \ \langle 0 \rangle)$   
**shows** ( $\sigma \models f \triangle g$ )

**proof** –

**have** 1: *index-sequence* 0  $\langle 0 \rangle$   
**by** (*simp add: index-sequence-def*)  
**have** 2: *powerinterval* *f*  $\sigma$   $\langle 0 \rangle$   
**by** (*simp add: powerinterval-def*)  
**have** 3: (*inth*  $\langle 0 \rangle$  (*ilen*  $\langle 0 \rangle$ )) = *ilen*  $\sigma$   
**by** (*simp add: assms*)  
**have** 4:  $g(\text{filt } \sigma \ \langle 0 \rangle)$   
**using** *assms* **by** *blast*  
**from** 1 2 3 4 **show** ?thesis  
**by** (*simp add: assms cpl-projection*)  
**qed**

**lemma** *cpl-more*:

**assumes** *ilen*  $\sigma > 0$   
 $(\sigma \models f \triangle g)$   
**shows** *ilen*(*cpl f g*  $\sigma$ )  $> 0$   
**by** (*metis assms cpl-projection gr0I index-sequence-def*)

**lemma** *cpl-more-than-first*:

**assumes** *ilen*  $\sigma > 0$   
 $(\sigma \models f \triangle g)$   
**shows** (*inth* (*cpl f g*  $\sigma$ ) 0) = 0  
**using** *assms cpl-projection index-sequence-def* **by** *auto*



**lemma** *cpl-more-than-last*:

**assumes**  $ilen\ \sigma > 0$

$(\sigma \models f \triangle g)$

**shows**  $(inth\ (cpl\ f\ g\ \sigma)\ (ilen\ (cpl\ f\ g\ \sigma))) = ilen\ \sigma$

**using** *assms cpl-projection* **by** *blast*

**lemma** *cpl-sub-more*:

**assumes**  $n < k$

$k \leq ilen\ \sigma$

$((sub\ n\ k\ \sigma) \models f \triangle g)$

**shows**  $ilen(cpl\ f\ g\ (sub\ n\ k\ \sigma)) > 0$

**using** *assms*

**by** (*simp add: cpl-more*)

**lemma** *cpl-bounds*:

**assumes**  $n < k$

$k \leq ilen\ \sigma$

$((sub\ n\ k\ \sigma) \models f \triangle g)$

$i < ilen\ (cpl\ f\ g\ (sub\ n\ k\ \sigma))$

**shows**  $0 \leq (inth\ (cpl\ f\ g\ (sub\ n\ k\ \sigma))\ i) \wedge (inth\ (cpl\ f\ g\ (sub\ n\ k\ \sigma))\ i) \leq k - n$

**using** *assms*

**by** (*metis cpl-projection idx-less-last-1 ilen-sub le0 less-imp-le-nat*)

**lemma** *cpl-imap-bounds*:

**assumes**  $n < k$

$k \leq ilen\ \sigma$

$((sub\ n\ k\ \sigma) \models f \triangle g)$

$i < ilen\ (imap\ (shift\ n)\ (cpl\ f\ g\ (sub\ n\ k\ \sigma)))$

**shows**  $n \leq (inth\ (imap\ (shift\ n)\ (cpl\ f\ g\ (sub\ n\ k\ \sigma)))\ i) \wedge$

$(inth\ (imap\ (shift\ n)\ (cpl\ f\ g\ (sub\ n\ k\ \sigma)))\ i) \leq k$

**using** *assms*

**by** (*metis shift-def Nat.le-diff-conv2 cpl-bounds ilen-imap*

*inth-imap le-add2 less-imp-le-nat*)

**lemma** *cpl-ifirst*:

**assumes**  $(sub\ x1a\ (ifirst\ l)\ \sigma) \models f \triangle g$

**shows**  $ifirst((imap\ (shift\ x1a)\ (cpl\ f\ g\ (sub\ x1a\ (ifirst\ l)\ \sigma)))) = x1a$

**proof** –

**have** 1:  $(index\ sequence\ 0\ (cpl\ f\ g\ (sub\ x1a\ (ifirst\ l)\ \sigma)) \wedge$

$powerinterval\ f\ (sub\ x1a\ (ifirst\ l)\ \sigma)\ (cpl\ f\ g\ (sub\ x1a\ (ifirst\ l)\ \sigma)) \wedge$

$(inth\ (cpl\ f\ g\ (sub\ x1a\ (ifirst\ l)\ \sigma))\ (ilen\ (cpl\ f\ g\ (sub\ x1a\ (ifirst\ l)\ \sigma)))) =$

$ilen\ (sub\ x1a\ (ifirst\ l)\ \sigma) \wedge$

$g\ (filt\ (sub\ x1a\ (ifirst\ l)\ \sigma)\ (cpl\ f\ g\ (sub\ x1a\ (ifirst\ l)\ \sigma)))$  )

**using** *cpl-projection assms* **by** *auto*

**have** 2:  $index\ sequence\ 0\ (cpl\ f\ g\ (sub\ x1a\ (ifirst\ l)\ \sigma))$

**using** 1 **by** *auto*

**have** 3:  $(inth\ (cpl\ f\ g\ (sub\ x1a\ (ifirst\ l)\ \sigma))\ 0) = 0$

**using** 2 *index-sequence-def* **by** *blast*

**show** *?thesis*

by (metis 3 shift-def add.left-neutral inth-imap)  
qed

lemma cpl-ifirst-same:

assumes (sub x1a x1a  $\sigma$ )  $\models f \triangle g$   
shows ifirst((imap (shift x1a) (cpl f g (sub x1a x1a  $\sigma$ )))) = x1a  
proof –  
have 1: ifirst ( $\langle x1a \rangle$ ) = x1a  
by auto  
from 1 cpl-ifirst show ?thesis by (metis assms)  
qed

lemma cpl-ilast:

assumes ((sub x (ifirst l)  $\sigma$ )  $\models f \triangle g$ )  $\wedge x < \text{ifirst } l \wedge \text{ifirst } l \leq \text{ilen } \sigma$   
shows ilast((imap (shift x) (cpl f g (sub x (ifirst l)  $\sigma$ )))) = ifirst l  
proof –  
have 01: ( index-sequence 0 (cpl f g (sub x (ifirst l)  $\sigma$ ))  $\wedge$   
powerinterval f (sub x (ifirst l)  $\sigma$ ) (cpl f g (sub x (ifirst l)  $\sigma$ ))  $\wedge$   
(inth (cpl f g (sub x (ifirst l)  $\sigma$ )) (ilen (cpl f g (sub x (ifirst l)  $\sigma$ )))) =  
ilen (sub x (ifirst l)  $\sigma$ )  $\wedge$   
g (filt (sub x (ifirst l)  $\sigma$ ) (cpl f g (sub x (ifirst l)  $\sigma$ ))) )  
using cpl-projection assms by (simp add: cpl-projection)  
have 02: (inth (cpl f g (sub x (ifirst l)  $\sigma$ )) (ilen (cpl f g (sub x (ifirst l)  $\sigma$ )))) =  
ilen (sub x (ifirst l)  $\sigma$ )  
using 01 by auto  
have 03: ilast(cpl f g (sub x (ifirst l)  $\sigma$ )) =  
(inth (cpl f g (sub x (ifirst l)  $\sigma$ )) (ilen (cpl f g (sub x (ifirst l)  $\sigma$ ))))  
by simp  
have 04: x < ifirst l  
using assms by blast  
have 05: ilen (sub x (ifirst l)  $\sigma$ ) = (ifirst l) - x  
using assms ilen-sub less-imp-le-nat by blast  
have 06: ilast((imap (shift x) (cpl f g (sub x (ifirst l)  $\sigma$ )))) =  
((ifirst l) - x) + x  
by (metis 02 05 shift-def ilen-imap inth-imap)  
show ?thesis using 04 06 by auto  
qed

lemma cpl-ilast-i:

assumes ((sub (inth l i) (inth l (Suc i))  $\sigma$ )  $\models f \triangle g$ )  
(inth l i) < (inth l (Suc i))  
(inth l (Suc i))  $\leq$  ilen  $\sigma$   
shows ilast((imap (shift (inth l i)) (cpl f g (sub (inth l i) (inth l (Suc i))  $\sigma$ )))) =  
(inth l (Suc i))  
proof –  
have 01: ( index-sequence 0 (cpl f g (sub (inth l i) (inth l (Suc i))  $\sigma$ ))  $\wedge$   
powerinterval f (sub (inth l i) (inth l (Suc i))  $\sigma$ )  
(cpl f g (sub (inth l i) (inth l (Suc i))  $\sigma$ ))  $\wedge$   
(inth (cpl f g (sub (inth l i) (inth l (Suc i))  $\sigma$ ))  
(ilen (cpl f g (sub (inth l i) (inth l (Suc i))  $\sigma$ )))) =

```

      ilen (sub (inth l i) (inth l (Suc i)) σ) ∧
      g (filt (sub (inth l i) (inth l (Suc i)) σ) (cpl f g (sub (inth l i) (inth l (Suc i)) σ))) )
    using cpl-projection assms by (simp add: cpl-projection)
  have 02: (inth (cpl f g (sub (inth l i) (inth l (Suc i)) σ))
    (ilen (cpl f g (sub (inth l i) (inth l (Suc i)) σ)))) =
    ilen (sub (inth l i) (inth l (Suc i)) σ)
  using 01 by auto
  have 03: ilast(cpl f g (sub (inth l i) (inth l (Suc i)) σ)) =
    (inth (cpl f g (sub (inth l i) (inth l (Suc i)) σ))
      (ilen (cpl f g (sub (inth l i) (inth l (Suc i)) σ))))
  by simp
  have 04: (inth l i) < (inth l (Suc i))
  by (simp add: assms)
  have 05: ilen (sub (inth l i) (inth l (Suc i)) σ) = (inth l (Suc i)) − (inth l i)
  by (simp add: assms less-imp-le-nat)
  have 06: ilast((imap (shift (inth l i)) (cpl f g (sub (inth l i) (inth l (Suc i)) σ)))) =
    ((inth l (Suc i)) − (inth l i)) + (inth l i)
  by (metis 02 05 shift-def ilen-imap inth-imap)
  show ?thesis using 04 06 by auto
qed

```

## 8.2.4 lcpl lemmas

```

lemma lcpl-inth:
  assumes index-sequence (inth l 0) l
    i < ilen l
  shows (inth (lcpl f g σ l) i) =
    (imap (shift (inth l i)) (cpl f g (sub (inth l i) (inth l (Suc i)) σ)))
using assms
proof (induct l arbitrary: i)
case (INil x)
then show ?case by simp
next
case (ICons x1a ls)
then show ?case
  proof (cases i)
  case 0
  then show ?thesis
    proof (cases ls)
    case (INil x1)
    then show ?thesis by (simp add: 0)
    next
    case (ICons x21 x22)
    then show ?thesis by (simp add: 0)
  qed
  next
  case (Suc nat)
  then show ?thesis
    proof (cases ls)
    case (INil x1)

```

```

then show ?thesis
by (metis ICons.prem1(2) One-nat-def Suc ilen.simps(1) ilen.simps(2) leD le-add1
    plus-1-eq-Suc)
next
case (ICons x21 x22)
then show ?thesis
using ICons.hyps ICons.prem1(1) ICons.prem1(2) Suc idx-ICons by auto
qed
qed
qed

```

```

lemma lcpl-ilen:
assumes index-sequence (inth l 0) l
        ilen l > 0
shows   ilen(lcpl f g  $\sigma$  l) = ilen l - 1
using assms
proof
  (induct l)
  case (INil x)
  then show ?case by simp
  next
  case (ICons x1a l)
  then show ?case
  proof (cases l)
  case (INil x1)
  then show ?thesis by simp
  next
  case (ICons x21 x22)
  then show ?thesis using ICons.hyps ICons.prem1(1) idx-ICons by auto
  qed
qed

```

```

lemma lcpl-ilen-zero:
assumes index-sequence (inth l 0) l
        ilen l = 0
shows   ilen(lcpl f g  $\sigma$  l) = 0
using assms
by (metis One-nat-def ilen.simps(1) ilen.simps(2) interval.exhaust lcpl.simps(1) leD le-add1
    le-imp-less-Suc order-refl)

```

```

lemma lcpl-last:
assumes index-sequence (inth l 0) l
        (inth l (ilen l)) = ilen  $\sigma$ 
        ilen l > 0
shows   ilast (lcpl f g  $\sigma$  l) =
        (imap (shift (inth l (ilen l - 1)))
            (cpl f g (sub (inth l (ilen l - 1)) (inth l (ilen l))  $\sigma$ )))
proof -

```

**have 1:**  $ilast (lcpl\ f\ g\ \sigma\ l) = (inth\ (lcpl\ f\ g\ \sigma\ l)\ (ilen\ (lcpl\ f\ g\ \sigma\ l)))$   
**by** *simp*  
**have 2:**  $(inth\ (lcpl\ f\ g\ \sigma\ l)\ (ilen\ (lcpl\ f\ g\ \sigma\ l))) =$   
 $(imap\ (shift\ (inth\ l\ (ilen\ (lcpl\ f\ g\ \sigma\ l))))$   
 $(cpl\ f\ g\ (sub\ (inth\ l\ (ilen\ (lcpl\ f\ g\ \sigma\ l)))$   
 $(inth\ l\ (Suc\ (ilen\ (lcpl\ f\ g\ \sigma\ l))))$   
 $\sigma)))$   
**using** *assms lcpl-inth*  
**by** (*metis One-nat-def Suc-pred diff-le-self lcpl-ilen le-eq-less-or-eq n-not-Suc-n*)  
**have 3:**  $(imap\ (shift\ (inth\ l\ (ilen\ (lcpl\ f\ g\ \sigma\ l))))$   
 $(cpl\ f\ g\ (sub\ (inth\ l\ (ilen\ (lcpl\ f\ g\ \sigma\ l)))$   
 $(inth\ l\ (Suc\ (ilen\ (lcpl\ f\ g\ \sigma\ l))))$   
 $\sigma))) =$   
 $(imap\ (shift\ (inth\ l\ (ilen\ l-1)))$   
 $(cpl\ f\ g\ (sub\ (inth\ l\ (ilen\ l-1))\ (inth\ l\ (Suc\ (ilen\ l-1)))\ \sigma)))$   
**using** *assms by (metis lcpl-ilen)*  
**show** *?thesis* **by** (*simp add: 2 3 assms(3)*)  
**qed**

**lemma** *lcpl-last-last:*

**assumes** *index-sequence*  $(inth\ l\ 0)\ l$   
 $(inth\ l\ (ilen\ l)) = ilen\ \sigma$   
 $((sub\ (inth\ l\ (ilen\ l-1))\ (inth\ l\ (ilen\ l))\ \sigma) \models f\ \triangle\ g)$   
 $ilen\ l > 0$

**shows**  $ilast\ (ilast\ (lcpl\ f\ g\ \sigma\ l)) = ilen\ \sigma$

**using** *assms*

**by** (*metis (no-types, lifting) One-nat-def Suc-pred' cpl-ilast-i diff-Suc-less*  
*dual-order.order-iff-strict index-sequence-def lcpl-last*)

**lemma** *lcpl-zero-zero:*

**assumes** *index-sequence*  $(inth\ l\ 0)\ l$   
 $(inth\ l\ (ilen\ l)) = ilen\ \sigma$   
 $ilen\ l = 0$

**shows**  $(inth\ (lcpl\ f\ g\ \sigma\ l)\ 0) =$   
 $(imap\ (shift\ (inth\ l\ 0))\ (cpl\ f\ g\ (sub\ (inth\ l\ 0)\ (inth\ l\ 0)\ \sigma)))$

**using** *assms*

**by** *simp*

$(metis\ INil-ilen\ lcpl.simps(1))$

**lemma** *lcpl-ifirst:*

**assumes** *index-sequence*  $(inth\ l\ 0)\ l$   
 $(inth\ l\ (ilen\ l)) = ilen\ \sigma$   
 $(\forall\ i < ilen\ l. (sub\ (inth\ l\ i)\ (inth\ l\ (Suc\ i))\ \sigma) \models f\ \triangle\ g)$   
 $ilen\ l > 0$

**shows**  $ifirst(ifirst((lcpl\ f\ g\ \sigma\ l))) = ifirst\ l$

**proof** –

**have 01:**  $(ifirst((lcpl\ f\ g\ \sigma\ l))) =$   
 $(imap\ (shift\ (inth\ l\ 0))\ (cpl\ f\ g\ (sub\ (inth\ l\ 0)\ (inth\ l\ (Suc\ 0))\ \sigma)))$

**using** *assms by (metis lcpl-inth)*

**have 02:**  $ilen\ l > 0 \longrightarrow$

```

    ifirst((imap (shift (inth l 0)) (cpl f g (sub (inth l 0) (inth l (Suc 0))  $\sigma$ )))) =
      (inth l 0)
    using assms by (metis Suc-leI cpl-ifirst ilast-ifirst ilast-prefix)
  show ?thesis
  using 01 02 inth-zero-ifirst by (simp add: assms(4))
qed

```

**lemma** *lcpl-lfuse-lastfirst*:

```

  assumes index-sequence (inth l 0) l
    (inth l (ilen l)) = ilen  $\sigma$ 
    ((ilen l = 0  $\wedge$  ((sub (inth l 0) (inth l 0)  $\sigma$ )  $\models$   $f \triangle g$ ))  $\vee$ 
      (ilen l > 0  $\wedge$  ( $\forall$   $i < \text{ilen } l$ . (sub (inth l i) (inth l (Suc i))  $\sigma$ )  $\models$   $f \triangle g$ )))
  shows lastfirst (lcpl f g  $\sigma$  l)
  using assms
  proof
    (induct l)
    case (INil x)
    then show ?case by simp
    next
    case (ICons x1a ls)
    then show ?case
    proof -
      have 1: ilen ls = 0  $\longrightarrow$  lastfirst (lcpl f g  $\sigma$  (x1a  $\odot$  ls))
        using ICons.prem by (metis One-nat-def add-diff-cancel-left' INil-ilen
          ilen.simps(2) lastfirst.simps(1) lcpl-ilen plus-1-eq-Suc zero-less-one)
      have 2: ilen ls > 0  $\longrightarrow$  lastfirst ( lcpl f g  $\sigma$  (x1a  $\odot$  ls) ) =
        lastfirst((imap (shift x1a) (cpl f g (sub x1a (ifirst ls)  $\sigma$ )))  $\odot$  (lcpl f g  $\sigma$  ls))
      by (metis (no-types, lifting) interval.simps(6) ilen-ICons-1 inth-zero
        lcpl.simps(2))
      have 3: ilen ls > 0  $\longrightarrow$ 
        lastfirst((imap (shift x1a) (cpl f g (sub x1a (ifirst ls)  $\sigma$ )))  $\odot$  (lcpl f g  $\sigma$  ls)) =
        ( ilast( (imap (shift x1a) (cpl f g (sub x1a (ifirst ls)  $\sigma$ ))) ) =
          ifirst(ifirst((lcpl f g  $\sigma$  ls)))  $\wedge$  lastfirst((lcpl f g  $\sigma$  ls)) )
        using lastfirst.simps(2) by blast
      have 4: x1a < (ifirst ls)
        using ICons.prem by (metis index-sequence-def inth-Suc inth-zero
          ilen.simps(2) plus-1-eq-Suc zero-less-Suc)
      have 5: (ifirst ls)  $\leq$  ilen  $\sigma$ 
        using ICons.prem by (metis Suc-lessI eq-iff idx-less-last-1
          inth-Suc ilen.simps(2) less-imp-le-nat plus-1-eq-Suc zero-less-Suc)
      have 6: ilen ls > 0  $\longrightarrow$ 
        ilast( (imap (shift x1a) (cpl f g (sub x1a (ifirst ls)  $\sigma$ ))) ) = ifirst ls
        using ICons.prem by (metis 4 5 cpl-ilast inth-Suc inth-zero less-le-not-le)
      have 7: index-sequence (inth ls 0) ls
        using assms ICons.prem index-sequence-def by auto
      have 8: inth ls (ilen ls) = ilen  $\sigma$ 
        using ICons.prem by auto
      have 9: (  $\forall$   $i < \text{ilen } (ls)$ . (sub (inth (ls) i) (inth (ls) (Suc i))  $\sigma$ )  $\models$   $f \triangle g$  )
        using ICons.prem by auto
    end
  end

```

```

have 10:  $ilen\ ls > 0 \longrightarrow ifirst(ifirst((lcpl\ f\ g\ \sigma\ ls))) = ifirst\ ls$ 
using 7 8 9
by (metis cpl-ifirst ilen-ICons-1 inth-Suc lcpl-inth)
have 11:  $ilen\ ls > 0 \longrightarrow$ 
       $lastfirst((lcpl\ f\ g\ \sigma\ ls))$ 
using 7 8 9 ICons.hyps by blast
show ?thesis using 1 10 11 2 6 by auto
qed
qed

lemma lcpl-lfuse-ilen:
assumes index-sequence (inth l 0) l
      (inth l (ilen l)) = ilen  $\sigma$ 
      ((ilen l = 0  $\wedge$  ((sub (inth l 0) (inth l 0)  $\sigma \models f \triangle g$ )))  $\vee$ 
      (ilen l > 0  $\wedge$  ( $\forall\ i < ilen\ l.$  (sub (inth l i) (inth l (Suc i))  $\sigma \models f \triangle g$ ))))
shows (ilen l = 0  $\longrightarrow$  ilen(lfuse (lcpl f g  $\sigma$  l)) = 0)  $\wedge$ 
      (ilen l > 0  $\longrightarrow$ 
      ilen(lfuse (lcpl f g  $\sigma$  l)) = ( $\sum k=0..ilen\ l-1.$  ilen (inth (lcpl f g  $\sigma$  l) k)))
proof -
have 1: lastfirst (lcpl f g  $\sigma$  l)
using assms lcpl-lfuse-lastfirst by blast
have 2: ilen l = 0  $\longrightarrow$ 
      lfuse (lcpl f g  $\sigma$  l) =
      ((imap (shift (inth l 0)) (cpl f g (sub (inth l 0) (inth l 0)  $\sigma$ ))))
by (metis assms(1) assms(2) INil-ilen lcpl-ilen-zero lcpl-zero-zero lfuse-INil)
have 3: ilen l = 0  $\longrightarrow$ 
      ilen ((imap (shift (inth l 0)) (cpl f g (sub (inth l 0) (inth l 0)  $\sigma$ )))) = 0
using assms cpl-empty ilen-sub by fastforce
have 4: ilen l = 0  $\longrightarrow$  ilen(lfuse (lcpl f g  $\sigma$  l)) = 0
using 2 3 by auto
have 5: ilen l > 0  $\longrightarrow$ 
      ilen(lfuse (lcpl f g  $\sigma$  l)) = ( $\sum k=0..ilen\ l-1.$  ilen (inth (lcpl f g  $\sigma$  l) k))
using lfuse-ilen
by (metis assms lcpl-ilen lcpl-lfuse-lastfirst)
from 4 5 show ?thesis by simp
qed

```

```

lemma lcpl-lfuse-idx:
assumes index-sequence 0 l
      (inth l (ilen l)) = ilen  $\sigma$ 
      ( $\forall\ i < ilen\ l.$  (sub (inth l i) (inth l (Suc i))  $\sigma \models f \triangle g$ )
      ilen l > 0)
shows index-sequence (ifirst (lfuse (lcpl f g  $\sigma$  l))) (lfuse (lcpl f g  $\sigma$  l))
proof -
have 0: ilen  $\sigma$  > 0  $\longrightarrow$  ilen l > 0
using assms gr-zeroI index-sequence-def by fastforce
have 2: ilen  $\sigma$  > 0  $\longrightarrow$  lastfirst (lcpl f g  $\sigma$  l)
using 0 assms index-sequence-def lcpl-lfuse-lastfirst by blast
have 3: ( $\forall\ i < ilen\ l.$  index-sequence 0 (cpl f g (sub (inth l i) (inth l (Suc i))  $\sigma$ )))
using assms cpl-projection by auto

```

**have 4:**  $(\forall i < \text{ilen } l.$   
 $\text{index-sequence } (\text{inth } l \ i)$   
 $(\text{imap } (\text{shift } (\text{inth } l \ i)) (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma))))$   
**using 3 idx-link by blast**  
**have 5:**  $\text{ilen } \sigma > 0 \longrightarrow$   
 $(\forall i < \text{ilen } l. \text{index-sequence } (\text{inth } l \ i) (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i) )$   
**using assms by (metis 4 index-sequence-def lcpl-inth)**  
**have 6:**  $\text{ilen } \sigma > 0 \longrightarrow$   
 $(\forall i < \text{ilen } l. \text{ifirst } (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i) = (\text{inth } l \ i))$   
**by (metis 5 index-sequence-def)**  
**have 7:**  $\text{ilen } \sigma > 0 \longrightarrow$   
 $(\forall i < \text{ilen } l.$   
 $\text{index-sequence } (\text{ifirst } (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i)) (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i) )$   
**using 6 5 by simp**  
**have 8:**  $\text{ilen } \sigma > 0 \longrightarrow$   
 $\text{index-sequence } (\text{ifirst } (\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l))) (\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l)))$   
**using assms**  
**by (metis (mono-tags, lifting) 2 7 Suc-diff-1 index-sequence-def idx-lfuse**  
 $\text{lcpl-ilen le-imp-less-Suc})$   
**from 8 show ?thesis**  
**by (metis assms(1) assms(2) assms(4) index-sequence-def idx-less-last-1)**  
**qed**

**lemma lcpl-ilen-inth-gr-zero:**

**assumes**  $\text{index-sequence } 0 \ l$

$(\text{inth } l \ (\text{ilen } l)) = \text{ilen } \sigma$

$(\forall i < \text{ilen } l. (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma) \models f \ \Delta \ g)$

$\text{ilen } \sigma > 0$

**shows**  $(\forall j \leq \text{ilen } (\text{lcpl } f \ g \ \sigma \ l). \text{ilen}(\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ j) > 0)$

**proof**

**fix**  $j$

**show**  $j \leq \text{ilen } (\text{lcpl } f \ g \ \sigma \ l) \longrightarrow 0 < \text{ilen } (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ j)$

**proof** –

**have 1:**  $\text{ilen } \sigma > 0 \longrightarrow \text{lastfirst } (\text{lcpl } f \ g \ \sigma \ l)$

**by (metis assms index-sequence-def lcpl-lfuse-lastfirst neq0-conv)**

**have 2:**  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen } l > 0$

**using assms gr-zeroI index-sequence-def by fastforce**

**have 3:**  $\text{ilen } \sigma > 0 \longrightarrow j \leq \text{ilen } (\text{lcpl } f \ g \ \sigma \ l) \longrightarrow$

$(\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ j) =$

$(\text{imap } (\text{shift } (\text{inth } l \ j)) (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ j) (\text{inth } l \ (\text{Suc } j)) \ \sigma))))$

**using assms lcpl-inth[of l j f g σ]**

**by (metis 2 One-nat-def Suc-pred index-sequence-def lcpl-ilen less-Suc-eq-le)**

**have 4:**  $\text{ilen } \sigma > 0 \longrightarrow j \leq \text{ilen } (\text{lcpl } f \ g \ \sigma \ l) \longrightarrow$

$\text{ilen } (\text{imap } (\text{shift } (\text{inth } l \ j)) (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ j) (\text{inth } l \ (\text{Suc } j)) \ \sigma)))) > 0$

**by (metis (no-types, lifting) 2 One-nat-def Suc-pred add commute assms cpl-sub-more gr0I**

$\text{index-sequence-def idx-expand ilen-imap lcpl-ilen$

$\text{le-imp-less-Suc plus-1-eq-Suc})$

**show ?thesis**

**using 3 4 by (simp add: assms(4))**

**qed**



qed

**lemma** *lcpl-ilast-inth*:

**assumes** *index-sequence* 0 *l*

$(\text{inth } l \ (\text{ilen } l)) = \text{ilen } \sigma$

$(\forall i < \text{ilen } l. (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma) \models f \ \Delta \ g)$

$\text{ilen } \sigma > 0$

$j \leq \text{ilen } (\text{lcpl } f \ g \ \sigma \ l)$

**shows**  $\text{ilast}(\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ j) = (\text{inth } l \ (\text{Suc } j))$

**proof** –

**have** 0:  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen } l > 0$

**using** *assms* *gr-zeroI* *index-sequence-def* **by** *fastforce*

**have** 1:  $\text{ilen } \sigma > 0 \longrightarrow$

$j \leq \text{ilen } (\text{lcpl } f \ g \ \sigma \ l) \longrightarrow$

$\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ j =$

$(\text{imap } (\text{shift } (\text{inth } l \ j)) \ (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ j) \ (\text{inth } l \ (\text{Suc } j)) \ \sigma)))$

**using** *assms* *lcpl-inth*[*of l j f g σ*]

**proof** –

**show** *?thesis*

**by** (*metis* (*no-types*) 0 *Suc-diff-1*

$\langle \llbracket \text{index-sequence } (\text{inth } l \ 0) \ l; j < \text{ilen } l \rrbracket \Longrightarrow$

$\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ j =$

$\text{imap } (\text{shift } (\text{inth } l \ j)) \ (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ j) \ (\text{inth } l \ (\text{Suc } j)) \ \sigma)) \rangle$

$\langle \text{index-sequence } 0 \ l \rangle \text{index-sequence-def } \text{lcpl-ilen } \text{le-imp-less-Suc}$ )

qed

**have** 2:  $\text{ilen } \sigma > 0 \longrightarrow$

$j \leq \text{ilen } (\text{lcpl } f \ g \ \sigma \ l) \longrightarrow (\text{inth } l \ (\text{Suc } j)) \leq \text{ilen } \sigma$

**using** *assms*

**by** (*metis* 0 *Suc-diff-1* *Suc-eq-plus1* *index-sequence-def* *idx-expand* *lcpl-ilen* *leD* *not-less-eq*)

**have** 2:  $\text{ilen } \sigma > 0 \longrightarrow$

$j \leq \text{ilen } (\text{lcpl } f \ g \ \sigma \ l) \longrightarrow$

$\text{ilast}(\text{imap } (\text{shift } (\text{inth } l \ j)) \ (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ j) \ (\text{inth } l \ (\text{Suc } j)) \ \sigma))) =$   
 $(\text{inth } l \ (\text{Suc } j))$

**using** *assms* *cpl-ilast-i*[*of f g l j σ*]

**by** (*metis* (*no-types*, *hide-lams*) 0 1 2 *One-nat-def* *Suc-le-lessD* *Suc-pred* *index-sequence-def* *lcpl-ilen* *not-less-eq-eq*)

**show** *?thesis* **using** 1 2 **by** (*simp* *add*: *assms*(4) *assms*(5))

qed

**lemma** *lcpl-lfuse-filt-power-help*:

**assumes** *index-sequence* 0 *l*

$(\text{inth } l \ (\text{ilen } l)) = \text{ilen } \sigma$

$(\forall i < \text{ilen } l. (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma) \models f \ \Delta \ g)$

$\text{ilen } \sigma > 0$

**shows**  $(\forall i < \text{ilen } l. g \ (\text{filt } \sigma \ (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i)))$

**proof** –

**have** 1:  $(\forall i < \text{ilen } l. g (\text{filt } (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma) \\ (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ ))$

**using** *assms cpl-projection* **by** *blast*

**have** 2:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $(\forall i < \text{ilen } l.$   
 $(\text{filt } \sigma (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i)) =$   
 $(\text{filt } \sigma (\text{imap } (\text{shift } (\text{inth } l \ i)) (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma))))))$

**using** *assms* **by** *(metis index-sequence-def lcpl-inth)*

**have** 3:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $(\forall i < \text{ilen } l.$   
 $\text{ilen } (\text{filt } \sigma (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i)) =$   
 $\text{ilen } (\text{filt } (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma)$   
 $(\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ ))$   
 $)$

**by** *(simp add: 2 filt-ilen)*

**have** 4:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $(\forall i < \text{ilen } l.$   
 $(\forall j \leq \text{ilen } (\text{filt } \sigma (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i)).$   
 $(\text{inth } (\text{filt } \sigma (\text{imap } (\text{shift } (\text{inth } l \ i)) (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma)))) \ j)$   
 $=$   
 $(\text{inth } \sigma ((\text{inth } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ j) + (\text{inth } l \ i)) \ ))$   
 $)$   
 $)$

**using** *inth-imap shift-def filt-imap* **by** *metis*

**have** 5:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $(\forall i < \text{ilen } l.$   
 $(\forall j \leq \text{ilen } (\text{filt } \sigma (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i)).$   
 $(\text{inth } (\text{filt } (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma)$   
 $(\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ ) \ j) =$   
 $(\text{inth } (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma)$   
 $(\text{inth } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ j))$   
 $))$

**by** *(simp add: filt-imap inth-imap)*

**have** 6:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $(\forall i < \text{ilen } l.$   
 $(\text{inth } l \ (\text{Suc } i)) \leq \text{ilen } \sigma$   
 $)$

**using** *assms idx-expand* **by** *auto*

**have** 7:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $(\forall i < \text{ilen } l.$   
 $(\text{inth } l \ i) \leq (\text{inth } l \ (\text{Suc } i))$   
 $)$

**using** *assms index-sequence-def less-imp-le* **by** *blast*

**have** 8:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $(\forall i < \text{ilen } l.$   
 $(\forall j \leq \text{ilen } (\text{filt } \sigma (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i)).$   
 $(\text{inth } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ j) \leq (\text{inth } l \ (\text{Suc } i)) - (\text{inth } l \ i)$   
 $)$

```

))
by (metis 3 6 7 assms(3) cpl-projection filt-ilen idx-less-last-1 ilen-sub
    le-eq-less-or-eq)
have 9: ilen  $\sigma > 0 \longrightarrow$ 
  ( $\forall i < \text{ilen } l.$ 
    ( $\forall j \leq \text{ilen } (\text{filt } \sigma (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i)).$ 
      ( $\text{inth } (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma)$ 
        ( $\text{inth } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ j$ )) =
      ( $\text{inth } \sigma ((\text{inth } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ j) + (\text{inth } l \ i))$ 
        ))
    )
  using 6
  by (simp add: 7 8 add.commute)
have 10: ilen  $\sigma > 0 \longrightarrow$ 
  ( $\forall i < \text{ilen } l.$ 
    ( $\forall j \leq \text{ilen } (\text{filt } \sigma (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i)).$ 
      ( $\text{filt } (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma)$ 
        ( $\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ ) =$ 
      ( $\text{filt } \sigma (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i)$ 
        ))
    )
  by (simp add: filt-expand)
  (metis 2 3 4 9 filt-ilen filt-imap inth-imap)
show ?thesis using 1 10 assms(4) by fastforce
qed

```

### 8.2.5 lsum lemmas

**lemma** *lsum-INil*:

$\text{lsum } \langle xs \rangle \ a = \langle a + \text{ilen } xs \rangle$

**by** *simp*

**lemma** *lsum-addzero-INil*:

$\text{addzero } (\text{lsum } \langle xs \rangle \ 0) = (\text{if } \text{ilen } xs = 0 \text{ then } \langle 0 \rangle \text{ else } \langle 0, \text{ilen } xs \rangle)$

**by** (simp add: addzero-def)

**lemma** *lsum-addzero-ICons*:

$\text{addzero } (\text{lsum } (xs \odot xxs) \ 0) = 0 \odot (\text{lsum } (xs \odot xxs) \ 0)$

**by** (simp add: addzero-def)

**lemma** *lsum-ifirst*:

$\text{ifirst } (\text{lsum } xxs \ a) = a + \text{ilen}(\text{ifirst } xxs)$

**by** (case-tac xxs) simp-all

**lemma** *lsum-ilen*:

$\text{ilen } (\text{lsum } xxs \ a) = \text{ilen } xxs$

**by** (induct xxs arbitrary: a) simp-all

**lemma** *lsum-addzero-ifirst*:

$\text{ifirst } (\text{addzero } (\text{lsum } xxs \ 0)) = 0$

**by** (simp add: addzero-def lsum-ifirst)

**lemma** *lsum-addzero-ilen*:

$(ilen\ xxs = 0 \wedge ilen(ifirst\ xxs) = 0 \longrightarrow$   
 $ilen\ (addzero\ (lsum\ xxs\ 0)) = 0)$   
 $\wedge$   
 $(ilen\ xxs = 0 \wedge ilen(ifirst\ xxs) > 0 \longrightarrow$   
 $ilen\ (addzero\ (lsum\ xxs\ 0)) = 1)$   
 $\wedge$   
 $(ilen\ xxs > 0 \longrightarrow$   
 $ilen\ (addzero\ (lsum\ xxs\ 0)) = (ilen\ xxs) + 1)$

**by** (*simp add: addzero-def*)  
*(metis add.left-neutral lsum-ifirst lsum-ilen)*

**lemma** *lsum-inth-help*:

**assumes**  $i > 0$

$i \leq ilen\ xxs + 1$

**shows**  $(\sum k = 0..(i-1). ilen\ (inth\ (xxs)\ k)) =$   
 $(\sum k = 1..i. ilen\ (inth\ (xxs)\ (k-1)))$

**using** *assms*

**proof**

*(induct i)*

**case** *0*

**then show** *?case* **by** *blast*

**next**

**case** *(Suc i)*

**then show** *?case*

**proof** *simp-all*

**assume** *a1*:  $0 < i \implies (\sum k = 0..i - Suc\ 0. ilen\ (inth\ xxs\ k)) =$   
 $(\sum k = Suc\ 0..i. ilen\ (inth\ xxs\ (k - Suc\ 0)))$

**have** *f2*:  $\forall f. sum\ f\ \{1::nat..0\} = (0::nat)$

**by** *auto*

**have** *f3*:  $\forall n\ f. sum\ f\ \{n::nat..n\} = (f\ n::nat)$

**by** *simp*

**have** *f4*:  $\forall f\ n. (sum\ f\ \{0::nat..n - 1\} + (f\ n::nat) = sum\ f\ \{0..n\} \vee 0 = n) \vee \neg\ 0 \leq n$

**by** (*metis (no-types) One-nat-def Suc-pred le-eq-less-or-eq sum.atLeast0-atMost-Suc*)

**have** *f5*:  $\forall n. (0::nat) \leq n$

**by** *blast*

**have**  $\forall n. (0::nat) + n = n$

**by** *linarith*

**then show**  $(\sum n = 0..i. ilen\ (inth\ xxs\ n)) =$   
 $(\sum n = Suc\ 0..i. ilen\ (inth\ xxs\ (n - Suc\ 0))) + ilen\ (inth\ xxs\ i)$

**using** *f5 f4 f3 f2 a1* **by** (*metis One-nat-def le-eq-less-or-eq*)

**qed**

**qed**

**lemma** *lsum-inth*:

**assumes**  $i \leq ilen\ xxs$

**shows**  $inth\ (lsum\ xxs\ a)\ i = a + (\sum k::nat = 0..i. ilen(inth\ xxs\ k))$

```

using assms
proof
  (induct xs arbitrary: a i)
  case (INil x)
  then show ?case by simp
  next
  case (ICons x1a xs)
  then show ?case
  proof -
    have 1: inth (lsum (x1a ⊙ xs) a) i = inth ((a+ilen x1a)⊙ (lsum xs (a+ilen x1a))) i
      by simp
    have 2: i ≤ ilen xs + 1
      using ICons.prem by auto
    have 3: i = 0 ⟶
      inth ((a+ilen x1a)⊙ (lsum xs (a+ilen x1a))) i = (a+ilen x1a)
      by simp
    have 4: a+ilen x1a = a + (∑ k = 0..0. ilen ( inth (x1a ⊙ xs) k))
      by simp
    have 5: i > 0 ∧ i ≤ ilen xs + 1 ⟶
      inth ((a+ilen x1a)⊙ (lsum xs (a+ilen x1a))) i =
      inth (lsum xs (a+ilen x1a)) (i-1)
      by (metis One-nat-def Suc-pred inth-Suc)
    have 6: i > 0 ∧ i ≤ ilen xs + 1 ⟶
      inth (lsum xs (a+ilen x1a)) (i-1) =
      (a+ilen x1a) + (∑ k = 0..(i-1). ilen ( inth (xs) k))
      using ICons.hyps le-diff-conv by blast
    have 7: i > 0 ∧ i ≤ ilen xs + 1 ⟶
      (∑ k = 0..(i-1). ilen ( inth (xs) k)) =
      (∑ k = 1..i. ilen ( inth (xs) (k-1)))
      using lsum-inth-help by blast
    have 8: i > 0 ∧ i ≤ ilen xs + 1 ⟶
      (∑ k = 1..i. ilen ( inth (xs) (k-1))) =
      (∑ k = 1..i. ilen ( inth (x1a ⊙ xs) (k)))
      by (metis (no-types, lifting) atLeastAtMost-iff inth-Suc
        ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc sum.cong)
    have 9: i > 0 ∧ i ≤ ilen xs + 1 ⟶
      (a+ilen x1a) + (∑ k = 1..i. ilen ( inth (x1a ⊙ xs) (k))) =
      a + (∑ k = 0..i. ilen ( inth (x1a ⊙ xs) (k)))
      by (simp add: sum.atLeast-Suc-atMost)
    show ?thesis
    by (metis 1 2 3 4 5 6 7 8 9 not-gr-zero)
  qed
qed

```

```

lemma lsum-addzero-inth:
  assumes i ≤ ilen (addzero (lsum xs 0))
  shows (ilen xs = 0 ∧ ilen(ifirst xs) = 0 ⟶
    inth (addzero (lsum xs 0)) i = (inth (lsum xs 0) i) )

```

$$\begin{aligned} & \wedge \\ & (\text{ilen } xs = 0 \wedge \text{ilen}(\text{ifirst } xs) > 0 \longrightarrow \\ & \quad \text{inth} (\text{addzero } (\text{lsum } xs \ 0)) \ i = (\text{inth } (0 \odot (\text{lsum } xs \ 0)) \ i) ) \\ & \wedge \\ & (\text{ilen } xs > 0 \longrightarrow \\ & \quad \text{inth} (\text{addzero } (\text{lsum } xs \ 0)) \ i = (\text{inth } (0 \odot (\text{lsum } xs \ 0)) \ i) ) \end{aligned}$$

**using** *assms*

**by** (*metis add.left-neutral addzero-def less-numeral-extra*(3) *lsum-ifirst lsum-ilen*)

**lemma** *lsum-ilast*:

*ilast* (*lsum* *xs* *a*) = *a* + ( $\sum k::\text{nat} = 0..(\text{ilen } xs).$  *ilen*(*inth* *xs* *k*))

**by** (*metis le-refl lsum-ilen lsum-inth*)

**lemma** *lsum-addzero-ilast*:

*ilast* (*addzero* (*lsum* *xs* 0)) = *ilast*(*lsum* *xs* 0)

**by** (*simp add: addzero-def*)

**lemma** *lsum-inth-leq-Suc*:

**assumes** *i* < *ilen* *xs*

( $\forall j \leq \text{ilen } xs. \text{ilen}(\text{inth } xs \ j) > 0$ )

**shows** *inth* (*lsum* *xs* *a*) *i* < *inth* (*lsum* *xs* *a*) (*Suc* *i*)

**proof** –

**have** 1: *inth* (*lsum* *xs* *a*) *i* = *a* + ( $\sum k::\text{nat} = 0..i. \text{ilen}(\text{inth } xs \ k)$ )

**using** *assms less-imp-le-nat lsum-inth* **by** *blast*

**have** 2: *inth* (*lsum* *xs* *a*) (*Suc* *i*) = *a* + ( $\sum k::\text{nat} = 0..(\text{Suc } i). \text{ilen}(\text{inth } xs \ k)$ )

**using** *assms* **by** (*simp add: lsum-inth Suc-leI*)

**have** 3: ( $\sum k::\text{nat} = 0..(\text{Suc } i). \text{ilen}(\text{inth } xs \ k)$ ) =

( $\sum k::\text{nat} = 0..i. \text{ilen}(\text{inth } xs \ k)$ ) + *ilen*(*inth* *xs* (*Suc* *i*))

**using** *sum.atLeast0-atMost-Suc* **by** *blast*

**have** 4: *ilen*(*inth* *xs* (*Suc* *i*)) > 0

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

**show** ?thesis **using** 1 2 3 4 **by** *linarith*

**qed**

**lemma** *lsum-addzero-inth-leq-Suc*:

**assumes** *i* < *ilen*(*addzero* (*lsum* *xs* 0))

( $\forall j \leq \text{ilen } xs. \text{ilen}(\text{inth } xs \ j) > 0$ )

**shows** *inth* (*addzero* (*lsum* *xs* 0)) *i* < *inth* (*addzero* (*lsum* *xs* 0)) (*Suc* *i*)

**proof** (*cases* *i*)

**case** 0

**then show** ?thesis

**by** (*metis add.left-neutral addzero-def assms*(2) *ilen-gr-zero inth-Suc less-numeral-extra*(3) *lsum-addzero-ifirst lsum-ifirst*)

**next**

**case** (*Suc* *nat*)

**then show** ?thesis

by (metis (no-types, lifting) Suc-less-eq2 add-diff-cancel-left' add-diff-cancel-right' addzero-def  
 assms(1) assms(2) inth-Suc lsum-addzero-ilen lsum-inth-leq-Suc neq0-conv not-add-less1  
 plus-1-eq-Suc)

qed

**lemma** *lsum-id<sub>x</sub>*:

**assumes**  $(\forall j \leq \text{ilen } xs. \text{ilen}(\text{inth } xs \ j) > 0)$

**shows** *index-sequence* (inth (lsum xs a) 0) (lsum xs a)

**by** (simp add: assms index-sequence-def lsum-ilen lsum-inth-leq-Suc)

**lemma** *lsum-addzero-id<sub>x</sub>*:

**assumes**  $(\forall j \leq \text{ilen } xs. \text{ilen}(\text{inth } xs \ j) > 0)$

**shows** *index-sequence* 0 (addzero (lsum xs 0))

**by** (metis idx-expand1 add.left-neutral addzero-def assms le0  
 lsum-id<sub>x</sub> lsum-ifirst)

**lemma** *filt-lfuse-lsum-a*:

**assumes** *lastfirst* (xs  $\odot$  xs)

$(\forall j \leq \text{ilen } xs. \text{ilen}(\text{inth } xs \ j) > 0)$

*ilen* xs > 0

**shows** (filt (fuse xs (lfuse xs)) (lsum xs (ilen xs))) =  
 (filt (lfuse xs) (lsum xs 0))

**using** assms

**proof**

(induct xs arbitrary: xs)

**case** (INil x)

**then show** ?case

**proof** –

**have** 1: filt (fuse xs (lfuse  $\langle x \rangle$ )) (lsum  $\langle x \rangle$  (ilen xs)) =  
 filt (fuse xs x) (lsum  $\langle x \rangle$  (ilen xs))

**by** simp

**have** 2: filt (fuse xs x) (lsum  $\langle x \rangle$  (ilen xs)) =  
 filt (fuse xs x)  $\langle \text{ilen } xs + \text{ilen } x \rangle$

**by** simp

**have** 3: filt (fuse xs x) ( $\langle \text{ilen } xs + \text{ilen } x \rangle$ ) =  
 $\langle (\text{inth } (\text{fuse } xs \ x) (\text{ilen } xs + \text{ilen } x)) \rangle$

**by** simp

**have** 4:  $\langle (\text{inth } (\text{fuse } xs \ x) (\text{ilen } xs + \text{ilen } x)) \rangle$  =  
 $\langle (\text{inth } x (\text{ilen } x)) \rangle$

**using** INil.prem<sub>s</sub> fuse-inth-a **by** auto

**have** 5: filt (lfuse  $\langle x \rangle$ ) (lsum  $\langle x \rangle$  0) =  
 filt (x) ( $\langle \text{ilen } x \rangle$ )

**by** simp

**have** 6: filt (x) ( $\langle \text{ilen } x \rangle$ ) =  $\langle (\text{inth } x (\text{ilen } x)) \rangle$

**by** auto

**show** ?thesis

**by** (simp add: 4)

qed

```

next
case (ICons x1a xs)
then show ?case
proof -
  have 01: filt (fuse xs (lfuse (x1a  $\odot$  xs))) (lsum (x1a  $\odot$  xs) (ilen xs)) =
    filt (fuse xs (fuse x1a (lfuse xs))) (lsum (x1a  $\odot$  xs) (ilen xs))
  by simp
  have 02: filt (fuse xs (fuse x1a (lfuse xs)))
    (lsum (x1a  $\odot$  xs) (ilen xs)) =
    filt (fuse xs (fuse x1a (lfuse xs)))
    ((ilen xs + ilen x1a)  $\odot$  lsum (xs) (ilen xs + ilen x1a))
  by simp
  have 03: filt (fuse xs (fuse x1a (lfuse xs)))
    ((ilen xs + ilen x1a)  $\odot$  lsum (xs) (ilen xs + ilen x1a)) =
    (inth (fuse xs (fuse x1a (lfuse xs))) (ilen xs + ilen x1a))  $\odot$ 
    filt (fuse xs (fuse x1a (lfuse xs))) (lsum (xs) (ilen xs + ilen x1a))
  by simp
  have 04: (inth (fuse xs (fuse x1a (lfuse xs))) (ilen xs + ilen x1a)) =
    (inth (fuse xs x1a) (ilen xs + ilen x1a))
  proof -
    have f1: inth xs (ilen xs) = inth x1a 0
    using ICons.prem(1) by force
    have f2: lastfirst (x1a  $\odot$  xs)
    using ICons.prem(1) lastfirst.simp(2) by blast
    then have inth x1a (ilen x1a) = inth ( inth xs 0) 0
    by simp
    then show ?thesis
    using f2 f1
    by (metis ICons.prem(1) add.right-neutral fuse-ilen-a
      fuse-inth-a ifirst-lfuse-ifirst
      inth-zero le0 le-add1 lfuse-ICons)
  qed
  have 05: (inth (fuse xs x1a) (ilen xs + ilen x1a)) = (inth x1a (ilen x1a))
  using ICons.prem(1) fuse-inth-a by force
  have 06: filt (fuse xs (fuse x1a (lfuse xs))) (lsum (xs) (ilen xs + ilen x1a)) =
    filt (fuse (fuse xs x1a) (lfuse xs)) (lsum (xs) (ilen (fuse xs x1a)))
  using FusionAssoc[of xs x1a (lfuse xs)]
  by (metis ICons.prem(1) fuse-ilen-a inth-zero lastfirst.simp(2) lastfirst-lfuse)
  have 07: lastfirst ( (fuse xs x1a)  $\odot$  xs )
  by (metis 05 ICons.prem(1) fuse-ilen-a lastfirst.simp(2))
  have 08: ilen (fuse xs x1a) > 0
  by (simp add: ICons.prem fuse-ilen-a)
  have 09: filt (fuse (fuse xs x1a) (lfuse xs)) (lsum (xs) (ilen (fuse xs x1a))) =
    filt (lfuse xs) (lsum (xs) 0)
  by (metis 07 08 ICons.prem(2) Suc-leI inth-Suc ilen.simp(2) less-Suc-eq-le
    ICons(1) plus-1-eq-Suc)
  have 10: filt (fuse xs (fuse x1a (lfuse xs)))
    ((ilen xs + ilen x1a)  $\odot$  lsum (xs) (ilen xs + ilen x1a)) =
    (inth x1a (ilen x1a))  $\odot$  filt (lfuse xs) (lsum (xs) 0)
  by (simp add: 04 05 06 09)

```



```

have 11: filt (lfuse (x1a  $\odot$  xxs)) (lsum (x1a  $\odot$  xxs) 0) =
  filt (fuse x1a (lfuse xxs)) ((ilen x1a)  $\odot$  (lsum xxs (ilen x1a)))
by simp
have 12: filt (fuse x1a (lfuse xxs)) ((ilen x1a)  $\odot$  (lsum xxs (ilen x1a))) =
  (inth (fuse x1a (lfuse xxs)) (ilen x1a))  $\odot$ 
  (filt (fuse x1a (lfuse xxs)) (lsum xxs (ilen x1a)))
by simp
have 13: (inth (fuse x1a (lfuse xxs)) (ilen x1a)) = (inth x1a (ilen x1a))
by (metis ICons.prems(1) eq-iff fuse-ilen-a fuse-inth
  ifirst-lfuse-ifirst lastfirst.simps(2) le-add1)
have 14: (filt (fuse x1a (lfuse xxs)) (lsum xxs (ilen x1a))) =
  filt (lfuse xxs) (lsum xxs 0)
using ICons.prems(1) ICons.prems(2) inth-zero ilen.simps(2) lastfirst.simps(2)
  ICons(1) by fastforce
have 15: filt (fuse x1a (lfuse xxs)) ((ilen x1a)  $\odot$  (lsum xxs (ilen x1a))) =
  (inth x1a (ilen x1a))  $\odot$  (filt (lfuse xxs) (lsum xxs 0))
by (simp add: 13 14)
show ?thesis using 10 15 by auto
qed
qed

```

**lemma** *filt-lfuse-lsum*:

```

assumes lastfirst (xs  $\odot$  xxs)
  ( $\forall j \leq \text{ilen } xxs. \text{ilen}(\text{inth } xxs \ j) > 0$ )
  ilen xs > 0
shows (filt (lfuse (xs  $\odot$  xxs)) (addzero (lsum (xs  $\odot$  xxs) 0))) =
  (ifirst xs)  $\odot$  (ilast xs)  $\odot$  (filt (lfuse xxs) (lsum xxs 0))
proof -
have 1: lfuse (xs  $\odot$  xxs) = fuse xs (lfuse xxs)
by simp
have 2: (filt (lfuse (xs  $\odot$  xxs)) (addzero (lsum (xs  $\odot$  xxs) 0))) =
  (filt (fuse xs (lfuse xxs)) (addzero (lsum (xs  $\odot$  xxs) 0)))
using 1 by simp
have 3: addzero (lsum (xs  $\odot$  xxs) 0) =
  0  $\odot$  (ilen xs)  $\odot$  (lsum xxs (ilen xs))
using lsum-addzero-ICons by auto
have 4: (filt (fuse xs (lfuse xxs)) (addzero (lsum (xs  $\odot$  xxs) 0))) =
  (filt (fuse xs (lfuse xxs)) (0  $\odot$  (ilen xs)  $\odot$  (lsum xxs (ilen xs))))
using 3 by auto
have 5: (filt (fuse xs (lfuse xxs)) (0  $\odot$  (ilen xs)  $\odot$  (lsum xxs (ilen xs)))) =
  (inth (fuse xs (lfuse xxs)) 0)  $\odot$ 
  (inth (fuse xs (lfuse xxs)) (ilen xs))  $\odot$ 
  (filt (fuse xs (lfuse xxs)) (lsum xxs (ilen xs)))
by simp
have 6: (inth (fuse xs (lfuse xxs)) 0) = (inth xs 0)
using assms by (metis fuse-inth ifirst-lfuse-ifirst ilen-gr-zero
  lastfirst.simps(2))
have 7: (inth (fuse xs (lfuse xxs)) (ilen xs)) = (inth xs (ilen xs))
using assms by (metis fuse-ilen-a fuse-inth ifirst-lfuse-ifirst
  lastfirst.simps(2) le-add1 order-refl)

```

```

have 8: index-sequence 0 (addzero (lsum xs 0))
  using assms lsum-addzero-idx by blast
have 9: (filt (fuse xs (lfuse xs)) (lsum xs (ilen xs))) =
  (filt (lfuse xs) (lsum xs 0))
  using assms filt-lfuse-lsum-a by blast
show ?thesis
  using 3 6 7 9 inth-ilen-ilast inth-zero-ifirst by force
qed

```

```

lemma filt-lfuse-lsum-1:
assumes lastfirst (xs  $\odot$  xs)
  ( $\forall j \leq \text{ilen } xs. \text{ilen}(\text{inth } xs \ j) > 0$ )
   $\text{ilen } xs > 0$ 
shows (filt (lfuse (xs  $\odot$  xs)) ((lsum (xs  $\odot$  xs) 0))) =
  (ilast xs)  $\odot$  (filt (lfuse xs) (lsum xs 0))
using assms by (simp,
  metis assms(1) eq-iff filt-lfuse-lsum-a fuse-ilen-a
  fuse-inth ifirst-lfuse-ifirst lastfirst.simps(2) le-add1)

```

```

lemma filt-lfuse-lsum-2:
assumes lastfirst (xs)
  ( $\forall j \leq \text{ilen } xs. \text{ilen}(\text{inth } xs \ j) > 0$ )
   $j \leq \text{ilen } xs$ 
shows (inth (filt (lfuse (xs)) ((lsum (xs) 0))) j) = ilast(inth xs j)
using assms
proof (induct xs arbitrary: j)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
  proof (cases j)
  case 0
  then show ?thesis using ICons.hyps ICons.prem1 filt-expand ifirst-lfuse-ifirst[of x1a xs]
    by simp
  (metis 0 add.right-neutral fuse-inth-a le0)
  next
  case (Suc nat)
  then show ?thesis using ICons.hyps ICons.prem1 by simp-all
  (metis ICons.prem1(1) Suc-le-mono filt-lfuse-lsum-a le-eq-less-or-eq old.nat.simps(4)
    old.nat.simps(5) zero-less-Suc)
  qed
qed

```

```

lemma filt-lfuse-lsum-3:
assumes lastfirst (xs)
  ( $\forall j \leq \text{ilen } xs. \text{ilen}(\text{inth } xs \ j) > 0$ )
   $j \leq \text{ilen } (\text{addzero } (\text{lsum } xs \ 0))$ 
shows (  $j=0 \longrightarrow (\text{inth } (\text{filt } (\text{lfuse } (xs)) (\text{addzero}(\text{lsum } (xs) \ 0))) j) = \text{ifirst}(\text{ifirst } xs)$  )
   $\wedge$ 

```

$$(j > 0 \longrightarrow (\text{inth } (\text{filt } (\text{lfuse } (x\text{xs})) (\text{addzero}(\text{lsum } (x\text{xs}) 0))) j) = \text{ilast}(\text{inth } x\text{xs } (j-1)))$$

**using** *assms*

**proof** –

**have** 1:  $j \leq \text{ilen } (\text{addzero } (\text{lsum } x\text{xs } 0)) \wedge j = 0 \longrightarrow$   
 $(\text{inth } (\text{filt } (\text{lfuse } (x\text{xs})) (\text{addzero}(\text{lsum } (x\text{xs}) 0))) j) = \text{ifirst}(\text{ifirst } x\text{xs})$

**using** *assms* **by** (*metis filt-ilen filt-inth lastfirst-lfuse*  
*lsum-addzero-ifirst*)

**have** 2:  $j \leq \text{ilen } (\text{addzero } (\text{lsum } x\text{xs } 0)) \wedge j > 0 \longrightarrow$   
 $(\text{inth } (\text{filt } (\text{lfuse } x\text{xs}) (\text{addzero}(\text{lsum } (x\text{xs}) 0))) j) =$   
 $(\text{inth } (\text{lfuse } x\text{xs}) (\text{inth } (\text{addzero}(\text{lsum } (x\text{xs}) 0)) j))$

**by** (*simp add: filt-ilen filt-inth*)

**have** 3:  $j \leq \text{ilen } (\text{addzero } (\text{lsum } x\text{xs } 0)) \wedge j > 0 \longrightarrow$   
 $\text{inth } (\text{addzero}(\text{lsum } (x\text{xs}) 0)) j = \text{inth } (\text{lsum } x\text{xs } 0) (j-1)$

**by** (*metis One-nat-def Suc-pred inth-Suc leD lsum-addzero-ilen lsum-addzero-inth neq0-conv*)

**have** 4:  $j \leq \text{ilen } (\text{addzero } (\text{lsum } x\text{xs } 0)) \wedge j > 0 \longrightarrow$   
 $(\text{inth } (\text{lfuse } x\text{xs}) (\text{inth } (\text{lsum } x\text{xs } 0) (j-1))) = \text{ilast}(\text{inth } x\text{xs } (j-1))$

**by** (*metis assms diff-is-0-eq' filt-ilen filt-lfuse-lsum-2 filt-inth*  
*le0 le-diff-conv lsum-addzero-ilen lsum-ilen neq0-conv*)

**show** *?thesis*

**using** 1 2 3 4 **by** (*simp add: assms(3)*)

**qed**

**lemma** *filt-lfuse-lsum-4*:

**assumes** *lastfirst (xxs)*

$(\forall j \leq \text{ilen } x\text{xs}. \text{ilen}(\text{inth } x\text{xs } j) > 0)$

**shows**  $(\text{inth } (\text{addzero } (\text{lsum } x\text{xs } 0)) (\text{ilen } (\text{addzero } (\text{lsum } x\text{xs } 0)))) \leq \text{ilen } (\text{lfuse } x\text{xs})$

**using** *assms*

**by** (*metis add-cancel-right-left eq-iff lfuse-ilen*  
*lsum-addzero-ilast lsum-ilen lsum-inth*)

**lemma** *filt-lfuse-lsum-5*:

**assumes** *lastfirst (xxs)*

$(\forall j \leq \text{ilen } x\text{xs}. \text{ilen}(\text{inth } x\text{xs } j) > 0)$

$i \leq \text{ilen } (\text{addzero } (\text{lsum } x\text{xs } 0))$

**shows**  $(\text{inth } (\text{addzero } (\text{lsum } x\text{xs } 0)) i) \leq \text{ilen } (\text{lfuse } x\text{xs})$

**using** *assms filt-lfuse-lsum-4[of xs] lsum-addzero-idx[of xs]*

**proof** –

**have** *f1*:  $\text{inth } (\text{addzero } (\text{lsum } x\text{xs } 0)) (\text{ilen } (\text{addzero } (\text{lsum } x\text{xs } 0))) \leq \text{ilen } (\text{lfuse } x\text{xs})$

**using**  $\langle \forall j \leq \text{ilen } x\text{xs}. 0 < \text{ilen } (\text{inth } x\text{xs } j) \rangle$

$\langle \llbracket \text{lastfirst } x\text{xs}; \forall j \leq \text{ilen } x\text{xs}. 0 < \text{ilen } (\text{inth } x\text{xs } j) \rrbracket \implies$

$\text{inth } (\text{addzero } (\text{lsum } x\text{xs } 0)) (\text{ilen } (\text{addzero } (\text{lsum } x\text{xs } 0))) \leq \text{ilen } (\text{lfuse } x\text{xs}) \rangle$

$\langle \text{lastfirst } x\text{xs} \rangle$  **by** *blast*

**have**  $\neg i < \text{ilen } (\text{addzero } (\text{lsum } x\text{xs } 0)) \vee$

$\text{inth } (\text{addzero } (\text{lsum } x\text{xs } 0)) i < \text{inth } (\text{addzero } (\text{lsum } x\text{xs } 0)) (\text{ilen } (\text{addzero } (\text{lsum } x\text{xs } 0)))$

**using**  $\langle \forall j \leq \text{ilen } x\text{xs}. 0 < \text{ilen } (\text{inth } x\text{xs } j) \implies \text{index-sequence } 0 (\text{addzero } (\text{lsum } x\text{xs } 0)) \rangle$

$\langle \forall j \leq \text{ilen } xs. 0 < \text{ilen } (\text{inth } xs \ j) \rangle \text{ idx-less-last-1}$  **by** *blast*  
**then show** *?thesis*  
**using** *f1*  $\langle i \leq \text{ilen } (\text{addzero } (\text{lsum } xs \ 0)) \rangle$  **by** *force*  
**qed**

**lemma** *lfuse-ilen-b*:

**assumes** *lastfirst xs*

**shows**  $(\forall i. 1 \leq i \wedge i \leq \text{ilen } (xs) \longrightarrow$   
 $(\forall j \leq \text{ilen } (\text{inth } (xs) \ i)).$   
 $\text{inth } ((\text{lsum } xs \ 0) \ ) \ (i-1) + j \leq \text{ilen } (\text{lfuse } xs))$

**proof** –

**have** *0*:  $(\forall i. 1 \leq i \wedge i \leq \text{ilen } (xs) \longrightarrow$   
 $(i-1) \leq \text{ilen } xs$   
 $)$

**by** *linarith*

**have** *1*:  $(\forall i. 1 \leq i \wedge i \leq \text{ilen } (xs) \longrightarrow$   
 $\text{inth } ((\text{lsum } xs \ 0) \ ) \ (i-1) =$   
 $(\sum k::\text{nat} = 0..(i-1). \text{ilen}(\text{inth } xs \ k))$   
 $)$

**by**  $(\text{metis } 0 \ \text{add.left-neutral lsum-inth})$

**have** *2*:  $\text{ilen } (\text{lfuse } xs) = (\sum k::\text{nat} = 0..(\text{ilen } xs). \text{ilen}(\text{inth } xs \ k))$

**using** *assms lfuse-ilen* **by** *blast*

**have** *3*:  $(\forall i. 1 \leq i \wedge i \leq \text{ilen } (xs) \longrightarrow$   
 $(\forall j \leq \text{ilen } (\text{inth } (xs) \ i)).$   
 $(\sum k::\text{nat} = 0..(i-1). \text{ilen}(\text{inth } xs \ k)) + j \leq$   
 $(\sum k::\text{nat} = 0..(i). \text{ilen}(\text{inth } xs \ k))$   
 $))$

**by**  $(\text{metis } (\text{no-types, lifting}) \ \text{add-le-cancel-left le-add-diff-inverse plus-1-eq-Suc}$   
 $\text{sum.atLeast0-atMost-Suc})$

**have** *4*:  $(\forall i. 1 \leq i \wedge i \leq \text{ilen } (xs) \longrightarrow$   
 $(\sum k::\text{nat} = 0..(i). \text{ilen}(\text{inth } xs \ k)) \leq$   
 $(\sum k::\text{nat} = 0..(\text{ilen } xs). \text{ilen}(\text{inth } xs \ k))$   
 $)$

**using** *sum.ub-add-nat*  $[\text{of } - \ - \ \lambda \ k. \text{ilen}(\text{inth } xs \ k) \ ]$

**by**  $(\text{metis } \text{Suc-eq-plus1 le-add1 le-add-diff-inverse le-simps}(1) \ \text{zero-less-Suc})$

**show** *?thesis*

**using** *1 2 3 4* **by** *fastforce*

**qed**

**lemma** *lsum-shift*:

**assumes** *lastfirst xs*

$(\forall j \leq \text{ilen } xs. \text{ilen}(\text{inth } xs \ j) > 0)$   
 $i \leq \text{ilen } xs$

**shows**  $\text{inth } (\text{lsum } xs \ a) \ i = a + \text{inth } (\text{lsum } xs \ 0) \ i$

**using** *assms* **by**  $(\text{simp add: lsum-inth})$

**lemma** *lsum-lfuse-inth-lsum-inth*:

```

assumes lastfirst xxs
  ( $\forall j \leq \text{ilen } xxs. \text{ilen}(\text{inth } xxs \ j) > 0$ )
shows ( $\forall i \leq \text{ilen } xxs.$ 
  ( $\forall j \leq \text{ilen}(\text{inth } xxs \ i).$ 
    ( $(\text{inth } (\text{lfuse } xxs) ((\text{inth } (\text{addzero } (\text{lsum } xxs \ 0)) \ i) + j)) =$ 
      ( $(\text{inth } (\text{inth } xxs \ i) \ j))$  ))
  )
using assms
proof
  (induct xxs)
  case (INil x)
  then show ?case
  by (metis inth.simps(1) add-cancel-right-left ilen.simps(1)
    le-zero-eq lfuse-INil lsum-addzero-ifirst)
  next
  case (ICons x1a xxs)
  then show ?case
  proof –
    have 1: ( $\forall i \leq \text{ilen } (x1a \odot xxs).$ 
      ( $\forall j \leq \text{ilen } (\text{inth } (x1a \odot xxs) \ i).$ 
        ( $\text{inth } (\text{lfuse } (x1a \odot xxs)) (\text{inth } (\text{addzero } (\text{lsum } (x1a \odot xxs) \ 0)) \ i + j) =$ 
          ( $\text{inth } (\text{inth } (x1a \odot xxs) \ i) \ j)$ )
        )
      )
    =
    ( $\forall j \leq \text{ilen } (\text{inth } (x1a \odot xxs) \ 0).$ 
      ( $\text{inth } (\text{lfuse } (x1a \odot xxs)) (\text{inth } (\text{addzero } (\text{lsum } (x1a \odot xxs) \ 0)) \ 0 + j) =$ 
        ( $\text{inth } (\text{inth } (x1a \odot xxs) \ 0) \ j) \wedge$ 
        ( $\forall i. 1 \leq i \wedge i \leq \text{ilen } (x1a \odot xxs) \longrightarrow$ 
          ( $\forall j \leq \text{ilen } (\text{inth } (x1a \odot xxs) \ i).$ 
            ( $\text{inth } (\text{lfuse } (x1a \odot xxs)) (\text{inth } (\text{addzero } (\text{lsum } (x1a \odot xxs) \ 0)) \ i + j) =$ 
              ( $\text{inth } (\text{inth } (x1a \odot xxs) \ i) \ j)$ )
          )
        )
      )
    )
    by (metis One-nat-def Suc-leI ilen-gr-zero not-gr-zero)
    have 2: ( $\forall j \leq \text{ilen } (\text{inth } (x1a \odot xxs) \ 0).$ 
      ( $\text{inth } (\text{lfuse } (x1a \odot xxs)) (\text{inth } (\text{addzero } (\text{lsum } (x1a \odot xxs) \ 0)) \ 0 + j) =$ 
        ( $\text{inth } (\text{inth } (x1a \odot xxs) \ 0) \ j)$ )
    )
    =
    ( $\forall j \leq \text{ilen } (x1a).$ 
      ( $\text{inth } (\text{lfuse } (x1a \odot xxs)) (j) = \text{inth } (x1a) \ j$ )
    )

    by (metis add-cancel-right-left inth-zero
      lsum-addzero-ifirst)
    have 3: ilast x1a = ifirst(ifirst xxs)
    using ICons.prems lastfirst.simps(2) by blast
    have 4: ( $\forall j \leq \text{ilen } (x1a).$ 
      ( $\text{inth } (\text{lfuse } (x1a \odot xxs)) (j) = \text{inth } (x1a) \ j$ )
    )
    by (metis ICons.prems(1) fuse-ilen-a fuse-inth
      ifirst-lfuse-ifirst lastfirst.simps(2) le-add1 le-trans lfuse-ICons)
    have 41: ( $\forall j \leq \text{ilen } (\text{inth } (x1a \odot xxs) \ 0).$ 
      ( $\text{inth } (\text{lfuse } (x1a \odot xxs)) (\text{inth } (\text{addzero } (\text{lsum } (x1a \odot xxs) \ 0)) \ 0 + j) =$ 
        ( $\text{inth } (\text{inth } (x1a \odot xxs) \ 0) \ j$ )
    )
    )
    using 2 4 by blast

```

**have 5:**  $(\forall i. 1 \leq i \wedge i \leq \text{ilen } (x1a \odot xs) \longrightarrow$   
 $(\forall j \leq \text{ilen } (\text{inth } (x1a \odot xs) i).$   
 $\text{inth } (\text{lfuse } (x1a \odot xs)) (\text{inth } (\text{addzero } (\text{lsum } (x1a \odot xs) 0)) i + j) =$   
 $\text{inth } (\text{inth } (x1a \odot xs) i) j)) =$   
 $(\forall i. 0 \leq i-1 \wedge i-1 \leq \text{ilen } (xs) \longrightarrow$   
 $(\forall j \leq \text{ilen } (\text{inth } (x1a \odot xs) i).$   
 $\text{inth } (\text{lfuse } (x1a \odot xs)) (\text{inth } (\text{addzero } (\text{lsum } (x1a \odot xs) 0)) i + j) =$   
 $\text{inth } (\text{inth } (x1a \odot xs) i) j))$   
**using 1 2 4 le-diff-conv by auto**  
**have 6:**  $(\forall i. 0 \leq i-1 \wedge i-1 \leq \text{ilen } (xs) \longrightarrow$   
 $(\forall j \leq \text{ilen } (\text{inth } (x1a \odot xs) i).$   
 $\text{inth } (\text{lfuse } (x1a \odot xs)) (\text{inth } (\text{addzero } (\text{lsum } (x1a \odot xs) 0)) i + j) =$   
 $\text{inth } (\text{inth } (x1a \odot xs) i) j)) =$   
 $(\forall i \leq \text{ilen } (xs).$   
 $(\forall j \leq \text{ilen } (\text{inth } (x1a \odot xs) (\text{Suc } i)).$   
 $\text{inth } (\text{lfuse } (x1a \odot xs)) (\text{inth } (\text{addzero } (\text{lsum } (x1a \odot xs) 0)) (\text{Suc } i) + j) =$   
 $\text{inth } (\text{inth } (x1a \odot xs) (\text{Suc } i)) j))$  **(is ?L = ?R)**  
**proof**  
**show ?L  $\implies$  ?R**  
**using 2 4 by simp**  
**show ?R  $\implies$  ?L**  
**using 2 4 by (metis One-nat-def Suc-pred gr0I)**  
**qed**  
**have 7:**  $(\forall i \leq \text{ilen } (xs).$   
 $(\forall j \leq \text{ilen } (\text{inth } (x1a \odot xs) (\text{Suc } i)).$   
 $\text{inth } (\text{lfuse } (x1a \odot xs)) (\text{inth } (\text{addzero } (\text{lsum } (x1a \odot xs) 0)) (\text{Suc } i) + j) =$   
 $\text{inth } (\text{inth } (x1a \odot xs) (\text{Suc } i)) j)) =$   
 $(\forall i \leq \text{ilen } (xs).$   
 $(\forall j \leq \text{ilen } (\text{inth } (xs) (i)).$   
 $\text{inth } (\text{lfuse } (x1a \odot xs)) (\text{inth } (\text{addzero } (\text{lsum } (x1a \odot xs) 0)) (\text{Suc } i) + j) =$   
 $\text{inth } (\text{inth } (xs) (i)) j))$   
**by simp**  
**have 8:**  $(\forall i \leq \text{ilen } (xs).$   
 $(\forall j \leq \text{ilen } (\text{inth } (xs) (i)).$   
 $\text{inth } (\text{lfuse } (x1a \odot xs)) (\text{inth } (\text{addzero } (\text{lsum } (x1a \odot xs) 0)) (\text{Suc } i) + j) =$   
 $\text{inth } (\text{inth } (xs) (i)) j)) =$   
 $(\forall i \leq \text{ilen } (xs).$   
 $(\forall j \leq \text{ilen } (\text{inth } (xs) (i)).$   
 $\text{inth } (\text{fuse } x1a (\text{lfuse } xs)) (\text{inth } ((\text{lsum } (x1a \odot xs) 0)) (i) + j) =$   
 $\text{inth } (\text{inth } (xs) (i)) j))$   
**by (metis inth-Suc lfuse-ICons lsum-addzero-ICons)**  
**have 9:**  $(\forall i \leq \text{ilen } (xs).$   
 $(\forall j \leq \text{ilen } (\text{inth } (xs) (i)).$   
 $\text{inth } (\text{fuse } x1a (\text{lfuse } xs)) (\text{inth } ((\text{lsum } (x1a \odot xs) 0)) (i) + j) =$   
 $\text{inth } (\text{inth } (xs) (i)) j))$   
 $=$   
 $(\forall i \leq \text{ilen } (xs).$   
 $(\forall j \leq \text{ilen } (\text{inth } (xs) (i)).$   
 $\text{inth } (\text{fuse } x1a (\text{lfuse } xs)) (\text{inth } ((\text{ilen } (x1a) \odot (\text{lsum } xs) (\text{ilen } x1a)) ) i + j) =$   
 $\text{inth } (\text{inth } (xs) (i)) j))$

**by simp**

**have 11:**  $(\forall i \leq \text{ilen } (xs)).$

$$\begin{aligned}
& (\forall j \leq \text{ilen } (\text{inth } (xs) (i))). \\
& \text{inth } (\text{fuse } x1a \text{ (lfuse } xs)) (\text{inth } ((\text{ilen}(x1a) \odot (\text{lsum } xs) (\text{ilen } x1a)) ) (i) + j) = \\
& \text{inth } (\text{inth } (xs) (i)) j) \\
& = \\
& ((\forall i. i=0 \longrightarrow \\
& (\forall j \leq \text{ilen } (\text{inth } (xs) (i))). \\
& \text{inth } (\text{fuse } x1a \text{ (lfuse } xs)) (\text{inth } ((\text{ilen}(x1a) \odot (\text{lsum } xs) (\text{ilen } x1a)) ) (i) + j) = \\
& \text{inth } (\text{inth } (xs) (i)) j)) \\
& \wedge \\
& (\forall i. 1 \leq i \wedge i \leq \text{ilen } (xs) \longrightarrow \\
& (\forall j \leq \text{ilen } (\text{inth } (xs) (i))). \\
& \text{inth } (\text{fuse } x1a \text{ (lfuse } xs)) (\text{inth } ((\text{ilen}(x1a) \odot (\text{lsum } xs) (\text{ilen } x1a)) ) (i) + j) = \\
& \text{inth } (\text{inth } (xs) (i)) j)) )
\end{aligned}$$

**by (metis One-nat-def Suc-leI gr-zeroI ilen-gr-zero)**

**have 12:**  $(\forall i. i=0 \longrightarrow$

$$\begin{aligned}
& (\forall j \leq \text{ilen } (\text{inth } (xs) (i))). \\
& \text{inth } (\text{fuse } x1a \text{ (lfuse } xs)) (\text{inth } ((\text{ilen}(x1a) \odot (\text{lsum } xs) (\text{ilen } x1a)) ) (i) + j) = \\
& \text{inth } (\text{inth } (xs) (i)) j) = \\
& (\forall j \leq \text{ilen } (\text{inth } (xs) (0))). \\
& \text{inth } (\text{fuse } x1a \text{ (lfuse } xs)) (\text{inth } ((\text{ilen}(x1a) \odot (\text{lsum } xs) (\text{ilen } x1a)) ) (0) + j) = \\
& \text{inth } (\text{inth } (xs) (0)) j)
\end{aligned}$$

**by blast**

**have 13:**  $(\forall j \leq \text{ilen } (\text{inth } (xs) (0))).$

$$\begin{aligned}
& \text{inth } (\text{fuse } x1a \text{ (lfuse } xs)) (\text{inth } ((\text{ilen}(x1a) \odot (\text{lsum } xs) (\text{ilen } x1a)) ) (0) + j) = \\
& \text{inth } (\text{inth } (xs) (0)) j) = \\
& (\forall j \leq \text{ilen } (\text{inth } (xs) (0))). \\
& \text{inth } (\text{fuse } x1a \text{ (lfuse } xs)) (\text{ilen}(x1a) + j) = \\
& \text{inth } (\text{inth } (xs) (0)) j)
\end{aligned}$$

**by auto**

**have 14:**  $(\forall j \leq \text{ilen } (\text{inth } (xs) (0))).$

$$\begin{aligned}
& \text{inth } (\text{fuse } x1a \text{ (lfuse } xs)) (\text{ilen}(x1a) + j) = \\
& \text{inth } (\text{inth } (xs) (0)) j) \\
& = \\
& (\forall j \leq \text{ilen } (\text{inth } (xs) (0))). \\
& \text{inth } ( \text{ (lfuse } xs)) (j) = \\
& \text{inth } (\text{inth } (xs) (0)) j)
\end{aligned}$$

**using ICons.premis fuse-inth-a ifirst-lfuse-ifirst lfuse-ilen-a**

**by (metis lastfirst.simps(2) le0)**

**have 15:**  $(\forall i. 1 \leq i \wedge i \leq \text{ilen } (xs) \longrightarrow$

$$\begin{aligned}
& (\forall j \leq \text{ilen } (\text{inth } (xs) (i))). \\
& \text{inth } (\text{fuse } x1a \text{ (lfuse } xs)) (\text{inth } ((\text{ilen}(x1a) \odot (\text{lsum } xs) (\text{ilen } x1a)) ) (i) + j) = \\
& \text{inth } (\text{inth } (xs) (i)) j) = \\
& (\forall i. 1 \leq i \wedge i \leq \text{ilen } (xs) \longrightarrow
\end{aligned}$$

$(\forall j \leq \text{ilen} \text{ (inth (xs) (i))}).$   
 $\text{inth (fuse x1a (lfuse xs)) (inth ( (lsum xs (ilen x1a)) ) (i-1) + j) =}$   
 $\text{inth (inth (xs) (i)) j)}$

**by** (*metis inth-Suc ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc*)

**have** 16:  $(\forall i. 1 \leq i \wedge i \leq \text{ilen} \text{ (xs)} \longrightarrow$   
 $\text{inth ( (lsum xs (ilen x1a)) ) (i-1) =}$   
 $\text{ilen x1a + inth (lsum xs 0) (i-1)})$

**by** (*simp add: le-diff-conv lsum-inth*)

**have** 17:  $(\forall i. 1 \leq i \wedge i \leq \text{ilen} \text{ (xs)} \longrightarrow$   
 $(\forall j \leq \text{ilen} \text{ (inth (xs) (i))}).$   
 $\text{inth (fuse x1a (lfuse xs)) (inth ( (lsum xs (ilen x1a)) ) (i-1) + j) =}$   
 $\text{inth (inth (xs) (i)) j)}$

=

$(\forall i. 1 \leq i \wedge i \leq \text{ilen} \text{ (xs)} \longrightarrow$   
 $(\forall j \leq \text{ilen} \text{ (inth (xs) (i))}).$   
 $\text{inth (fuse x1a (lfuse xs)) (ilen x1a + inth ((lsum xs 0) ) (i-1) + j) =}$   
 $\text{inth (inth (xs) (i)) j)}$

**using** 16 **by** *auto*

**have** 18:  $(\forall i. 1 \leq i \wedge i \leq \text{ilen} \text{ (xs)} \longrightarrow$   
 $(\forall j \leq \text{ilen} \text{ (inth (xs) (i))}).$   
 $\text{inth ((lsum xs 0) ) (i-1) + j \leq \text{ilen (lfuse xs))})$

**using** *ICons.premis lastfirst.simps(2) lfuse-ilen-b* **by** *blast*

**have** 19:  $(\forall i. 1 \leq i \wedge i \leq \text{ilen} \text{ (xs)} \longrightarrow$   
 $(\forall j \leq \text{ilen} \text{ (inth (xs) (i))}).$   
 $\text{inth (fuse x1a (lfuse xs)) (ilen x1a + inth ((lsum xs 0) ) (i-1) + j) =}$   
 $\text{inth ( (lfuse xs)) ( inth ((lsum xs 0) ) (i-1) + j))}$

**by** (*metis ICons.premis(1) ab-semigroup-add-class.add-ac(1) fuse-inth-a*

*ifirst-lfuse-ifirst lastfirst.simps(2) lfuse-ilen-b*)

**have** 20:  $(\forall i. 1 \leq i \wedge i \leq \text{ilen} \text{ (xs)} \longrightarrow$   
 $(\forall j \leq \text{ilen} \text{ (inth (xs) (i))}).$   
 $\text{inth (fuse x1a (lfuse xs)) (ilen x1a + inth ((lsum xs 0) ) (i-1) + j) =}$   
 $\text{inth (inth (xs) (i)) j)}$  =  
 $(\forall i. 1 \leq i \wedge i \leq \text{ilen} \text{ (xs)} \longrightarrow$   
 $(\forall j \leq \text{ilen} \text{ (inth (xs) (i))}).$   
 $\text{inth ( (lfuse xs)) ( inth ((lsum xs 0) ) (i-1) + j) =}$   
 $\text{inth (inth (xs) (i)) j)}$

**using** 19 **by** *auto*

**have** 201:  $(\forall i. i=0 \longrightarrow$   
 $(\forall j \leq \text{ilen} \text{ (inth (xs) (i))}).$   
 $\text{inth (fuse x1a (lfuse xs)) (inth ((ilen(x1a) \odot (lsum xs) (ilen x1a)) ) (i) + j) =}$   
 $\text{inth (inth (xs) (i)) j)}$   
 $\wedge$   
 $(\forall i. 1 \leq i \wedge i \leq \text{ilen} \text{ (xs)} \longrightarrow$   
 $(\forall j \leq \text{ilen} \text{ (inth (xs) (i))}).$   
 $\text{inth (fuse x1a (lfuse xs)) (inth ((ilen(x1a) \odot (lsum xs) (ilen x1a)) ) (i) + j) =}$   
 $\text{inth (inth (xs) (i)) j)}$  ) =



```

    (( $\forall j \leq \text{ilen} \ (\text{inth} \ (xs) \ (0)).$ 
       $\text{inth} \ ( \ (\text{lfuse} \ xs)) \ (j) =$ 
       $\text{inth} \ (\text{inth} \ (xs) \ (0)) \ j) \wedge$ 
      ( $\forall i. \ 1 \leq i \wedge i \leq \text{ilen} \ (xs) \longrightarrow$ 
        ( $\forall j \leq \text{ilen} \ (\text{inth} \ (xs) \ (i)).$ 
           $\text{inth} \ ( \ (\text{lfuse} \ xs)) \ ( \ \text{inth} \ ((\text{lsum} \ xs \ 0) \ ) \ (i-1) + j) =$ 
           $\text{inth} \ (\text{inth} \ (xs) \ (i)) \ j)) \ )$ 
      using 14 15 17 20 by auto
  have 21: lastfirst xs
    using ICons.prem lastfirst.simps(2) by blast
  have 22: ( $\forall j \leq \text{ilen} \ xs. \ \text{ilen}(\text{inth} \ xs \ j) > 0$ )
    using ICons.prem by auto
  have 23: ( $\forall i \leq \text{ilen} \ xs.$ 
    ( $\forall j \leq \text{ilen} \ (\text{inth} \ xs \ i).$ 
       $\text{inth} \ (\text{lfuse} \ xs) \ (\text{inth} \ (\text{addzero} \ (\text{lsum} \ xs \ 0)) \ i + j) =$ 
       $\text{inth} \ (\text{inth} \ xs \ i) \ j)$ 
    using 21 22 ICons.hyps by blast
  have 24: ( $\forall j \leq \text{ilen} \ (\text{inth} \ xs \ 0).$ 
     $\text{inth} \ (\text{lfuse} \ xs) \ (\text{inth} \ (\text{addzero} \ (\text{lsum} \ xs \ 0)) \ 0 + j) =$ 
     $\text{inth} \ ( \ \text{inth} \ xs \ 0) \ j) \wedge$ 
    ( $\forall i. \ 1 \leq i \wedge i \leq \text{ilen} \ xs \longrightarrow$ 
      ( $\forall j \leq \text{ilen} \ (\text{inth} \ xs \ i).$ 
         $\text{inth} \ (\text{lfuse} \ xs) \ (\text{inth} \ (\text{addzero} \ (\text{lsum} \ xs \ 0)) \ i + j) =$ 
         $\text{inth} \ (\text{inth} \ xs \ i) \ j)) \ )$ 
    using 23 by blast
  have 25: ( $\forall j \leq \text{ilen} \ (\text{inth} \ (xs) \ (0)).$ 
     $\text{inth} \ ( \ (\text{lfuse} \ xs)) \ (j) =$ 
     $\text{inth} \ (\text{inth} \ (xs) \ (0)) \ j)$ 
    by (metis 24 add-cancel-right-left lsum-addzero-ifirst)
  have 26: ( $\forall i. \ 1 \leq i \wedge i \leq \text{ilen} \ (xs) \longrightarrow$ 
     $\text{inth} \ (\text{addzero} \ (\text{lsum} \ xs \ 0)) \ i = \text{inth} \ ((\text{lsum} \ xs \ 0) \ ) \ (i-1)$ 
    )
    by (metis addzero-def inth-Suc less-eq-Suc-le less-le-trans lsum-ilen
      not-less-eq-eq ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc
      zero-less-one)
  have 27: ( $\forall i. \ 1 \leq i \wedge i \leq \text{ilen} \ (xs) \longrightarrow$ 
    ( $\forall j \leq \text{ilen} \ (\text{inth} \ (xs) \ (i)).$ 
       $\text{inth} \ ( \ (\text{lfuse} \ xs)) \ ( \ \text{inth} \ ((\text{lsum} \ xs \ 0) \ ) \ (i-1) + j) =$ 
       $\text{inth} \ (\text{inth} \ (xs) \ (i)) \ j) =$ 
      ( $\forall i. \ 1 \leq i \wedge i \leq \text{ilen} \ xs \longrightarrow$ 
        ( $\forall j \leq \text{ilen} \ (\text{inth} \ xs \ i).$ 
           $\text{inth} \ (\text{lfuse} \ xs) \ (\text{inth} \ (\text{addzero} \ (\text{lsum} \ xs \ 0)) \ i + j) =$ 
           $\text{inth} \ (\text{inth} \ xs \ i) \ j)$ 
        )
      by (simp add: 26)
    show ?thesis
    using 11 201 24 25 27 6 8 by auto
  qed

```

qed

**lemma** *lcpl-lsum-less-th-equal:*

**assumes** *index-sequence 0 l*

$(\text{inth } l \ (\text{ilen } l)) = \text{ilen } \sigma$

$(\forall \ i < \text{ilen } l. (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma) \models f \ \Delta \ g)$

$\text{ilen } \sigma > 0$

$i < \text{ilen } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ l) \ 0))$

**shows**  $(\text{inth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ l) \ 0)) \ (\text{Suc } i)) \leq$

$\text{ilen}(\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l))$

**using** *assms*

**by** (*metis (no-types, lifting) Suc-leI filt-lfuse-lsum-5 index-sequence-def lcpl-ilen-inth-gr-zero lcpl-lfuse-lastfirst neq0-conv*)

**lemma** *lcpl-lsum-ilen:*

**assumes** *index-sequence 0 l*

$(\text{inth } l \ (\text{ilen } l)) = \text{ilen } \sigma$

$(\forall \ i < \text{ilen } l. (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma) \models f \ \Delta \ g)$

$\text{ilen } \sigma > 0$

**shows**  $\text{ilen } (\text{addzero } (\text{lsum } ((\text{lcpl } f \ g \ \sigma \ l)) \ 0)) = \text{ilen } l$

**proof** –

**have** 1:  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen } (\text{lcpl } f \ g \ \sigma \ l) = \text{ilen } l - 1$

**using** *assms index-sequence-def lcpl-ilen* **by** *fastforce*

**have** 2:  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen } l > 0$

**using** *assms gr-zeroI index-sequence-def* **by** *fastforce*

**have** 3:  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen } (\text{ifirst } (\text{lcpl } f \ g \ \sigma \ l)) > 0$

**by** (*metis assms ilen-gr-zero lcpl-ilen-inth-gr-zero*)

**have** 4:  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen } (\text{lcpl } f \ g \ \sigma \ l) = 0 \longrightarrow$

$\text{ilen } (\text{addzero } (\text{lsum } ((\text{lcpl } f \ g \ \sigma \ l)) \ 0)) = \text{ilen } l$

**using** 1 2 3 *lsum-addzero-ilen* **by** *fastforce*

**have** 5:  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen } (\text{lcpl } f \ g \ \sigma \ l) > 0 \longrightarrow$

$\text{ilen } (\text{addzero } (\text{lsum } ((\text{lcpl } f \ g \ \sigma \ l)) \ 0)) = \text{ilen } l$

**by** (*simp add: 1 lsum-addzero-ilen*)

**show** *?thesis* **using** 4 5 **using** *assms(4)* **by** *blast*

**qed**

**lemma** *lcpl-lsum-inth:*

**assumes** *index-sequence 0 l*

$(\text{inth } l \ (\text{ilen } l)) = \text{ilen } \sigma$

$(\forall \ i < \text{ilen } l. (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma) \models f \ \Delta \ g)$

$\text{ilen } \sigma > 0$

$j \leq \text{ilen } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ l) \ 0))$

**shows**  $(j = 0 \longrightarrow$

$(\text{inth } (\text{filt } (\text{lfuse } ((\text{lcpl } f \ g \ \sigma \ l)))) (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ l)) \ 0))) \ j) =$

$\text{ifirst}(\text{ifirst } (\text{lcpl } f \ g \ \sigma \ l)) \ )$

$\wedge$

$$(j > 0 \longrightarrow (\text{inth } (\text{filt } (\text{lfuse } ((\text{lcpl } f \ g \ \sigma \ l)))) (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ l)) \ 0))) \ j) = \\ \text{ilast}(\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ (j-1)))$$

**proof** –

**have** 0:  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen } l > 0$

**using** *assms gr-zeroI index-sequence-def* **by** *fastforce*

**have** 1:  $\text{ilen } \sigma > 0 \longrightarrow \text{lastfirst } (\text{lcpl } f \ g \ \sigma \ l)$

**using** 0 *assms index-sequence-def lcpl-lfuse-lastfirst* **by** *blast*

**have** 2:  $\text{ilen } \sigma > 0 \longrightarrow$

$$(\forall j \leq \text{ilen } (\text{lcpl } f \ g \ \sigma \ l). \text{ilen}(\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ j) > 0)$$

**by** (*simp add: assms lcpl-ilen-inth-gr-zero*)

**have** 3:  $\text{ilen } \sigma > 0 \longrightarrow$

$$\text{ilen } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ l) \ 0)) =$$

$$\text{ilen } (\text{lcpl } f \ g \ \sigma \ l) + 1$$

**by** (*metis 2 One-nat-def Suc-eq-plus1 ilen-gr-zero*

*le-imp-less-or-eq lsum-addzero-ilen*)

**have** 4:  $\text{ilen } \sigma > 0 \longrightarrow$

$$(\text{inth } (\text{filt } (\text{lfuse } ((\text{lcpl } f \ g \ \sigma \ l)))) (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ l)) \ 0))) \ 0) = \\ \text{ifirst}(\text{ifirst } (\text{lcpl } f \ g \ \sigma \ l))$$

**by** (*simp add: 1 2 filt-lfuse-lsum-3*)

**have** 5:  $\text{ilen } \sigma > 0 \longrightarrow$

$$j \leq \text{ilen } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ l) \ 0)) \wedge j > 0 \longrightarrow$$

$$(\text{inth } (\text{filt } (\text{lfuse } ((\text{lcpl } f \ g \ \sigma \ l)))) (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ l)) \ 0))) \ (j)) \\ = \text{ilast}(\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ (j-1))$$

**by** (*simp add: 1 2 filt-lfuse-lsum-3*)

**from** 4 5 **show** *?thesis* **using** *assms(4) assms(5)* **by** *blast*

**qed**

**lemma** *lcpl-lsum-inth-a*:

**assumes** *index-sequence 0 l*

$$(\text{inth } l \ (\text{ilen } l)) = \text{ilen } \sigma$$

$$(\forall i < \text{ilen } l. (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma) \models f \ \triangle \ g)$$

$$\text{ilen } \sigma > 0$$

$$j \leq \text{ilen } l$$

$$\text{shows } (\text{inth } (\text{filt } (\text{lfuse } ((\text{lcpl } f \ g \ \sigma \ l)))) (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ l)) \ 0))) \ j) = \\ (\text{inth } l \ j)$$

**proof** –

**have** 1:  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen } l > 0$

**using** *assms gr-zeroI index-sequence-def* **by** *fastforce*

**have** 2:  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ l) \ 0)) = \text{ilen } l$

**by** (*simp add: assms lcpl-lsum-ilen*)

**have** 3:  $\text{ilen } \sigma > 0 \longrightarrow$

$$j \leq \text{ilen } l \longrightarrow$$

$$(j = 0 \longrightarrow$$

$$(\text{inth } (\text{filt } (\text{lfuse } ((\text{lcpl } f \ g \ \sigma \ l)))) (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ l)) \ 0))) \ j) = \\ \text{ifirst}(\text{ifirst } (\text{lcpl } f \ g \ \sigma \ l))$$

$\wedge$

$$(j > 0 \longrightarrow (\text{inth } (\text{filt } (\text{lfuse } ((\text{lcpl } f \ g \ \sigma \ l)))) (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ l)) \ 0))) \ j) = \\ \text{ilast}(\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ (j-1)))$$

by (metis 2 assms lcpl-lsum-inth)  
 have 4:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $j \leq \text{ilen } l \wedge j = 0 \longrightarrow \text{ifirst}(\text{ifirst } (\text{lcpl } f \ g \ \sigma \ l)) = (\text{inth } l \ j)$   
 using assms lcpl-ifirst[of  $l \ \sigma \ f \ g$ ]  
 by (metis 1 index-sequence-def)  
 have 5:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $j \leq \text{ilen } l \wedge j > 0 \longrightarrow j - 1 \leq \text{ilen } (\text{lcpl } f \ g \ \sigma \ l)$   
 using assms by (metis 1 diff-le-mono index-sequence-def lcpl-ilen)  
 have 6:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $j \leq \text{ilen } l \wedge j > 0 \longrightarrow \text{ilast}(\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ (j - 1)) = (\text{inth } l \ j)$   
 using lcpl-ilast-inth  
 by (metis 5 One-nat-def Suc-pred assms)  
 show ?thesis  
 using 3 4 6 using assms(4) assms(5) by auto  
 qed

**lemma** *lcpl-filt-lfuse-lsum:*

**assumes** *index-sequence* 0  $l$

$(\text{inth } l \ (\text{ilen } l)) = \text{ilen } \sigma$

$(\forall \ i < \text{ilen } l. (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma) \models f \ \Delta \ g)$

$\text{ilen } \sigma > 0$

**shows**  $(\text{filt } (\text{lfuse } ((\text{lcpl } f \ g \ \sigma \ l))) \ (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ l)) \ 0))) = l$

**using** *assms*

**proof** –

**have** 0:  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen } l > 0$

**using** *assms* gr-zeroI *index-sequence-def* **by** *fastforce*

**have** 1:  $\text{ilen } \sigma > 0 \longrightarrow$

$\text{ilen } (\text{filt } (\text{lfuse } ((\text{lcpl } f \ g \ \sigma \ l))) \ (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ l)) \ 0))) = \text{ilen } l$

**by** (*simp* *add*: *assms* *filt-ilen* *lcpl-lsum-ilen*)

**have** 2:  $\text{ilen } \sigma > 0 \longrightarrow$

$(\forall \ j. j \leq \text{ilen } l \longrightarrow$

$(\text{inth } (\text{filt } (\text{lfuse } ((\text{lcpl } f \ g \ \sigma \ l))) \ (\text{addzero}(\text{lsum } ((\text{lcpl } f \ g \ \sigma \ l)) \ 0))) \ j) =$

$(\text{inth } l \ j))$

**by** (*simp* *add*: *assms* *lcpl-lsum-inth-a*)

**from** 1 2 **show** ?thesis **by** (*simp* *add*: *assms*(4) *interval-eq-inth-eq*)

qed

**lemma** *lcpl-lfuse-filt-power:*

**assumes** *index-sequence* 0  $l$

$(\text{inth } l \ (\text{ilen } l)) = \text{ilen } \sigma$

$(\forall \ i < \text{ilen } l. (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma) \models f \ \Delta \ g)$

$\text{ilen } \sigma > 0$

**shows**  $\text{powerinterval } g \ (\text{filt } \sigma \ (\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l))) \ (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ l) \ 0))$

**proof** –

**have** 0:  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen } l > 0$

**using** *assms* gr-zeroI *index-sequence-def* **by** *fastforce*

**have** 01:  $(\forall \ i < \text{ilen } l.$

$g \ (\text{filt } (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma) \ (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma))))$

```

using assms cpl-projection by auto
have 02:  $ilen\ \sigma > 0 \longrightarrow (\forall\ i < ilen\ l.\ g\ (filt\ \sigma\ (inth\ (lcpl\ f\ g\ \sigma\ l)\ i))\ )$ 
using 0 assms index-sequence-def lcpl-lfuse-filt-power-help by blast
have 03:  $powerinterval\ g\ (filt\ \sigma\ (lfuse\ (lcpl\ f\ g\ \sigma\ l)))\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0)) =$ 
 $(\forall\ i < ilen\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0)).$ 
 $g\ (sub\ (inth\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))\ i)$ 
 $(inth\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))\ (Suc\ i))$ 
 $(filt\ \sigma\ (lfuse\ (lcpl\ f\ g\ \sigma\ l))))$ 
by (simp add: powerinterval-def)
have 04:  $ilen\ \sigma > 0 \longrightarrow ilen\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0)) = ilen\ l$ 
by (simp add: assms lcpl-lsum-ilen)
have 05:  $ilen\ \sigma > 0 \longrightarrow$ 
 $(\forall\ i < ilen\ l.$ 
 $(filt\ \sigma\ (inth\ (lcpl\ f\ g\ \sigma\ l)\ i)) =$ 
 $(sub\ (inth\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))\ i)$ 
 $(inth\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))\ (Suc\ i))$ 
 $(filt\ \sigma\ (lfuse\ (lcpl\ f\ g\ \sigma\ l))))$ 
proof –
have 06:  $ilen\ \sigma > 0 \longrightarrow$ 
 $(\forall\ i < ilen\ l.\ ilen\ (filt\ \sigma\ (inth\ (lcpl\ f\ g\ \sigma\ l)\ i)) =$ 
 $ilen\ (inth\ (lcpl\ f\ g\ \sigma\ l)\ i))$ 
using filt-ilen by blast
have 07:  $ilen\ \sigma > 0 \longrightarrow$ 
 $(\forall\ i < ilen\ l.\ ilen\ (inth\ (lcpl\ f\ g\ \sigma\ l)\ i) =$ 
 $ilen\ (imap\ (shift\ (inth\ l\ i))\ (cpl\ f\ g\ (sub\ (inth\ l\ i)\ (inth\ l\ (Suc\ i))\ \sigma))))$ 
using assms by (metis index-sequence-def lcpl-inth)
have 08:  $ilen\ \sigma > 0 \longrightarrow$ 
 $(\forall\ i < ilen\ l.$ 
 $ilen\ (imap\ (shift\ (inth\ l\ i))\ (cpl\ f\ g\ (sub\ (inth\ l\ i)\ (inth\ l\ (Suc\ i))\ \sigma))) =$ 
 $ilen\ (cpl\ f\ g\ (sub\ (inth\ l\ i)\ (inth\ l\ (Suc\ i))\ \sigma)))$ 
using ilen-imap by blast
have 09:  $ilen\ \sigma > 0 \longrightarrow$ 
 $(\forall\ i < ilen\ l.$ 
 $ilen\ (sub\ (inth\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))\ i)$ 
 $(inth\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))\ (Suc\ i))$ 
 $(filt\ \sigma\ (lfuse\ (lcpl\ f\ g\ \sigma\ l)))) =$ 
 $(inth\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))\ (Suc\ i)) -$ 
 $(inth\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))\ i)$ 
by (metis 04 assms filt-ilen ilen-sub lcpl-lsum-less-th-equal
lcpl-ilen-inth-gr-zero le-eq-less-or-eq lsum-addzero-inth-leq-Suc)
have 10:  $ilen\ \sigma > 0 \longrightarrow$ 
 $(\forall\ i < ilen\ l.\ (inth\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))\ (Suc\ i)) =$ 
 $(inth\ (0 \odot (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))\ (Suc\ i)))$ 
using 04 addzero-def by auto
have 11:  $ilen\ \sigma > 0 \longrightarrow$ 

```

```

    (∀ i<ilen l. (inth (0⊙(lsum (lcpl f g σ l) 0)) (Suc i)) =
      (∑ k::nat= 0..i. ilen(inth (lcpl f g σ l) k)) )
    using 04
  proof simp-all
    assume 0 < ilen σ ⟶ ilen (addzero (lsum (lcpl f g σ l) 0)) = ilen l
    show 0 < ilen σ ⟶
      (∀ i<ilen l. inth (lsum (lcpl f g σ l) 0) i =
        (∑ k = 0..i. ilen (inth (lcpl f g σ l) k)))
    proof
      assume 0 < ilen σ
      show ∀ i<ilen l. inth (lsum (lcpl f g σ l) 0) i =
        (∑ k = 0..i. ilen (inth (lcpl f g σ l) k))
      proof
        fix i
        show i < ilen l ⟶
          inth (lsum (lcpl f g σ l) 0) i = (∑ k = 0..i. ilen (inth (lcpl f g σ l) k))
        by (metis (no-types, lifting) 0 One-nat-def Suc-leI Suc-le-mono Suc-pred
          add.left-neutral assms(1) assms(4) index-sequence-def lcpl-ilen lsum-inth sum.cong)
      qed
    qed
  qed
  have 12: ilen σ > 0 ⟶
    (∀ i<ilen l. (inth (addzero (lsum (lcpl f g σ l) 0)) i) =
      (inth (0⊙(lsum (lcpl f g σ l) 0)) i) )

    using 04 addzero-def by auto
  have 13: ilen σ > 0 ⟶
    (∀ i<ilen l. (inth (0⊙(lsum (lcpl f g σ l) 0)) i) =
      (case i of 0 ⇒ 0
        | Suc j ⇒ (∑ k::nat= 0..(j). ilen(inth (lcpl f g σ l) k)) ))
    using 11 by (simp-all add: Nitpick.case-nat-unfold)
  have 14: ilen σ > 0 ⟶
    (∀ i<ilen l.
      (inth (addzero (lsum (lcpl f g σ l) 0)) (Suc i)) -
      (inth (addzero (lsum (lcpl f g σ l) 0)) i) =
      (∑ k::nat= 0..i. ilen(inth (lcpl f g σ l) k)) -
      (case i of 0 ⇒ 0
        | Suc j ⇒ (∑ k::nat= 0..(j). ilen(inth (lcpl f g σ l) k)) )) )

    using 10 11 12 13 by auto
  have 15: ilen σ > 0 ⟶
    (∀ i<ilen l.
      (∑ k::nat= 0..(i). ilen(inth (lcpl f g σ l) k)) -
      (case i of 0 ⇒ 0
        | Suc j ⇒ (∑ k::nat= 0..(j). ilen(inth (lcpl f g σ l) k)) )) =
      (case i of 0 ⇒ ilen(inth (lcpl f g σ l) 0)
        | Suc j ⇒ (∑ k::nat= 0..(Suc j). ilen(inth (lcpl f g σ l) k)) -
          (∑ k::nat= 0..(j). ilen(inth (lcpl f g σ l) k)) ))

    by (simp add: Nitpick.case-nat-unfold)

```

**have 16:**  $ilen \sigma > 0 \longrightarrow$   
 $(\forall i < ilen \ l.$   
 $(case \ i \ of \ 0 \Rightarrow ilen(inth \ (lcpl \ f \ g \ \sigma \ l) \ 0)$   
 $| \ Suc \ j \Rightarrow (\sum k::nat= 0..(Suc \ j). \ ilen(inth \ (lcpl \ f \ g \ \sigma \ l) \ k)) -$   
 $(\sum k::nat= 0..(j). \ ilen(inth \ (lcpl \ f \ g \ \sigma \ l) \ k)) \ ) =$   
 $(case \ i \ of \ 0 \Rightarrow ilen(inth \ (lcpl \ f \ g \ \sigma \ l) \ 0)$   
 $| \ Suc \ j \Rightarrow ilen(inth \ (lcpl \ f \ g \ \sigma \ l) \ (Suc \ j)) \ ) \ )$

**by** (*metis (no-types, lifting) Nitpick.case-nat-unfold add-diff-cancel-left'*  
*sum.atLeast0-atMost-Suc*)

**have 17:**  $ilen \sigma > 0 \longrightarrow$   
 $(\forall i < ilen \ l.$   
 $(case \ i \ of \ 0 \Rightarrow ilen(inth \ (lcpl \ f \ g \ \sigma \ l) \ 0)$   
 $| \ Suc \ j \Rightarrow ilen(inth \ (lcpl \ f \ g \ \sigma \ l) \ (Suc \ j)) \ ) =$   
 $ilen(inth \ (lcpl \ f \ g \ \sigma \ l) \ i))$

**by** (*simp add: Nitpick.case-nat-unfold*)

**have 18:**  $ilen \sigma > 0 \longrightarrow$   
 $(\forall i < ilen \ l. \ ilen \ (filt \ \sigma \ (inth \ (lcpl \ f \ g \ \sigma \ l) \ i)) =$   
 $ilen \ (sub \ (inth \ (addzero \ (lsum \ (lcpl \ f \ g \ \sigma \ l) \ 0)) \ i)$   
 $(inth \ (addzero \ (lsum \ (lcpl \ f \ g \ \sigma \ l) \ 0)) \ (Suc \ i))$   
 $(filt \ \sigma \ (lfuse \ (lcpl \ f \ g \ \sigma \ l)))) \ )$

**by** (*simp add: 09 14 15 16 17 filt-ilen*)

**have 19:**  $ilen \sigma > 0 \longrightarrow$   
 $(\forall i < ilen \ l.$   
 $(\forall j \leq ilen(inth \ (lcpl \ f \ g \ \sigma \ l) \ i).$   
 $(inth \ (addzero \ (lsum \ (lcpl \ f \ g \ \sigma \ l) \ 0)) \ (Suc \ i)) \leq$   
 $ilen \ (filt \ \sigma \ (lfuse \ (lcpl \ f \ g \ \sigma \ l)))) \ )$

**by** (*simp add: 04 assms filt-ilen lcpl-lsum-less-th-equal*)

**have 22:**  $ilen \sigma > 0 \longrightarrow lastfirst \ (lcpl \ f \ g \ \sigma \ l)$

**using** 0 *assms index-sequence-def lcpl-lfuse-lastfirst* **by** *blast*

**have 23:**  $ilen \sigma > 0 \longrightarrow (\forall j \leq ilen \ (lcpl \ f \ g \ \sigma \ l). \ ilen(inth \ (lcpl \ f \ g \ \sigma \ l) \ j) > 0)$

**by** (*simp add: assms lcpl-ilen-inth-gr-zero*)

**have 190:**  $ilen \sigma > 0 \longrightarrow$   
 $(\forall i < ilen \ l.$   
 $(\forall j \leq ilen(inth \ (lcpl \ f \ g \ \sigma \ l) \ i).$   
 $(inth \ (addzero \ (lsum \ (lcpl \ f \ g \ \sigma \ l) \ 0)) \ i) \leq$   
 $(inth \ (addzero \ (lsum \ (lcpl \ f \ g \ \sigma \ l) \ 0)) \ (Suc \ i))$   
 $))$

**by** (*simp add: 04 23 less-imp-le-nat lsum-addzero-inth-leq-Suc*)

**have 20:**  $ilen \sigma > 0 \longrightarrow$   
 $(\forall i < ilen \ l.$   
 $(\forall j \leq ilen(inth \ (lcpl \ f \ g \ \sigma \ l) \ i).$   
 $(inth \ (sub \ (inth \ (addzero \ (lsum \ (lcpl \ f \ g \ \sigma \ l) \ 0)) \ i)$   
 $(inth \ (addzero \ (lsum \ (lcpl \ f \ g \ \sigma \ l) \ 0)) \ (Suc \ i))$   
 $(filt \ \sigma \ (lfuse \ (lcpl \ f \ g \ \sigma \ l)))) \ j) =$   
 $(inth \ (filt \ \sigma \ (lfuse \ (lcpl \ f \ g \ \sigma \ l)))) \ )$

$$((\text{inth} (\text{addzero} (\text{lsum} (\text{lcpl } f \ g \ \sigma \ l) \ 0)) \ i) + j) \ ) \ ) \ )$$

**by** (*simp add: 14 15 16 17 19 190*)

**have 21:**  $\text{ilen } \sigma > 0 \longrightarrow$

$$\begin{aligned} & (\forall \ i < \text{ilen } l. \\ & \quad (\forall \ j \leq \text{ilen}(\text{inth} (\text{lcpl } f \ g \ \sigma \ l) \ i). \\ & \quad \quad (\text{inth} (\text{filt } \sigma (\text{lfuse} (\text{lcpl } f \ g \ \sigma \ l)))) \\ & \quad \quad ((\text{inth} (\text{addzero} (\text{lsum} (\text{lcpl } f \ g \ \sigma \ l) \ 0)) \ i) + j) \ ) = \\ & \quad \quad (\text{inth } \sigma (\text{inth} (\text{lfuse} (\text{lcpl } f \ g \ \sigma \ l)) \\ & \quad \quad \quad ((\text{inth} (\text{addzero} (\text{lsum} (\text{lcpl } f \ g \ \sigma \ l) \ 0)) \ i) + j) \ ) \ ) \ ) \ ) \end{aligned}$$

**by** (*simp add: filt-imap inth-imap*)

**have 24:**  $\text{ilen } \sigma > 0 \longrightarrow$

$$\begin{aligned} & (\forall \ i \leq \text{ilen} (\text{lcpl } f \ g \ \sigma \ l). \\ & \quad (\forall \ j \leq \text{ilen}(\text{inth} (\text{lcpl } f \ g \ \sigma \ l) \ i). \\ & \quad \quad ( (\text{inth} (\text{lfuse} (\text{lcpl } f \ g \ \sigma \ l)) ((\text{inth} (\text{addzero} (\text{lsum} (\text{lcpl } f \ g \ \sigma \ l) \ 0)) \ i) + j) \ ) = \\ & \quad \quad ( (\text{inth} (\text{inth} (\text{lcpl } f \ g \ \sigma \ l) \ i) \ j) \ ) \ ) \end{aligned}$$

**by** (*simp add: 22 23 lsum-lfuse-inth-lsum-inth*)

**have 241:**  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen} (\text{lcpl } f \ g \ \sigma \ l) = \text{ilen } l - 1$

**using** 0 *assms index-sequence-def lcpl-ilen* **by** *blast*

**have 25:**  $\text{ilen } \sigma > 0 \longrightarrow$

$$\begin{aligned} & (\forall \ i < \text{ilen } l. \\ & \quad (\forall \ j \leq \text{ilen}(\text{inth} (\text{lcpl } f \ g \ \sigma \ l) \ i). \\ & \quad \quad (\text{inth } \sigma (\text{inth} (\text{lfuse} (\text{lcpl } f \ g \ \sigma \ l)) \\ & \quad \quad \quad ((\text{inth} (\text{addzero} (\text{lsum} (\text{lcpl } f \ g \ \sigma \ l) \ 0)) \ i) + j) \ ) \ ) = \\ & \quad \quad (\text{inth} (\text{filt } \sigma (\text{inth} (\text{lcpl } f \ g \ \sigma \ l) \ i) \ j) \ ) \ ) \end{aligned}$$

**by** (*simp add: 06 24 241 assms filt-inth*)

**have 26:**  $\text{ilen } \sigma > 0 \longrightarrow$

$$\begin{aligned} & (\forall \ i < \text{ilen } l. \\ & \quad (\forall \ j \leq \text{ilen}(\text{inth} (\text{lcpl } f \ g \ \sigma \ l) \ i). \\ & \quad \quad (\text{inth} (\text{filt } \sigma (\text{inth} (\text{lcpl } f \ g \ \sigma \ l) \ i) \ j) = \\ & \quad \quad (\text{inth} (\text{sub} (\text{inth} (\text{addzero} (\text{lsum} (\text{lcpl } f \ g \ \sigma \ l) \ 0)) \ i) \\ & \quad \quad \quad (\text{inth} (\text{addzero} (\text{lsum} (\text{lcpl } f \ g \ \sigma \ l) \ 0)) (\text{Suc } i)) \\ & \quad \quad \quad (\text{filt } \sigma (\text{lfuse} (\text{lcpl } f \ g \ \sigma \ l)))) \ j) \ ) \ ) \end{aligned}$$

**by** (*simp add: 20 21 25*)

**from** 18 26 **show** *?thesis*

**by** (*simp add: filt-ilen interval-eq-inth-eq*)

**qed**

**show** *?thesis*

**using** 02 03 04 05 **by** (*simp add: assms(4)*)

**qed**

**lemma** *lcpl-lfuse-filt-ilen:*

**assumes** *index-sequence 0 l*

$$(\text{inth } l (\text{ilen } l)) = \text{ilen } \sigma$$

$$(\forall \ i < \text{ilen } l. (\text{sub} (\text{inth } l \ i) (\text{inth } l (\text{Suc } i)) \ \sigma) \models f \ \triangle \ g)$$

$$\text{ilen } \sigma > 0$$



**shows**  $(\text{inth } (\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ l) \ 0)) \ (\text{ilen}(\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ l) \ 0)))) =$   
 $\text{ilen } (\text{filt } \sigma \ (\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l)))$   
**proof** –  
**have** 0:  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen } l > 0$   
**using** *assms gr-zeroI index-sequence-def* **by** *fastforce*  
**have** 1:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $\text{ilen } (\text{filt } \sigma \ (\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l))) = \text{ilen } ( \ (\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l)))$   
**using** *filt-ilen* **by** *blast*  
**have** 2:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $\text{ilen } ( \ (\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l))) =$   
 $(\sum k::\text{nat} = 0..(\text{ilen } (\text{lcpl } f \ g \ \sigma \ l)). \text{ilen}(\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ k))$   
**using** 0 *assms index-sequence-def lfuse-ilen lcpl-lfuse-lastfirst* **by** *blast*  
**have** 3:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $\text{ilast}(\text{addzero } (\text{lsum } (\text{lcpl } f \ g \ \sigma \ l) \ 0)) = \text{ilast}(\text{lsum } (\text{lcpl } f \ g \ \sigma \ l) \ 0))$   
**using** *lsum-addzero-ilast* **by** *blast*  
**have** 4:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $\text{ilast}(\text{lsum } (\text{lcpl } f \ g \ \sigma \ l) \ 0)) =$   
 $(\sum k::\text{nat} = 0..(\text{ilen } (\text{lcpl } f \ g \ \sigma \ l)). \text{ilen}(\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ k))$   
**by** (*metis add-cancel-right-left lsum-ilast*)  
**show** ?thesis **using** 1 2 3 4 **by** (*simp add: assms(4)*)  
**qed**

## 8.3 Soundness of Projection Axioms

### 8.3.1 PJ1

**lemma** *PJ1sem:*

$(\sigma \models f \triangle (g \vee h) \longrightarrow f \triangle g \vee f \triangle h)$   
**by** (*simp add: projection-d-def*) *blast*

**lemma** *PJ1sema:*

$(\sigma \models f \triangle (g \vee h) = (f \triangle g \vee f \triangle h))$   
**by** (*simp add: projection-d-def*) *blast*

### 8.3.2 PJ2

**lemma** *PJ2sem:*

$(\sigma \models f \triangle \text{empty} = \text{empty})$

**proof** *auto*

**show**  $(\sigma \models f \triangle \text{empty}) \Longrightarrow \sigma \models \text{empty}$

**unfolding** *projection-d-def empty-defs index-sequence-def*

**by** (*metis filt.simps(2) ilen-ICons-1 neq0-conv*)

**show**  $\sigma \models \text{empty} \Longrightarrow (\sigma \models f \triangle \text{empty})$

**unfolding** *projection-d-def empty-defs index-sequence-def powerinterval-def*

**by** (*metis inth.simps(1) filt.simps(1) index-sequence-def ilen.simps(1)*  
*not-less-zero* )

**qed**

### 8.3.3 PJ3

**lemma** *PJ3help:*

$sub\ 0\ (ilen\ \sigma)\ \sigma = \sigma$   
**by** (*simp add: sub-zero-prefix*)

**lemma** *PJ3help1*:

**assumes**  $f\ \sigma \wedge 0 < ilen\ \sigma$

**shows**  $(\exists l. index\_sequence\ 0\ l \wedge$   
 $inth\ l\ (ilen\ l) = ilen\ \sigma \wedge$   
 $(\forall i < ilen\ l. f\ (sub\ (inth\ l\ i)\ (inth\ l\ (Suc\ i))\ \sigma)) \wedge$   
 $(\exists s1. ilen\ s1 = ilen\ l \wedge$   
 $(\forall i \leq ilen\ s1. inth\ s1\ i = inth\ \sigma\ (inth\ l\ i)) \wedge ilen\ s1 = Suc\ 0))$

**proof** –

**have** 1:  $index\_sequence\ 0\ \langle 0, ilen\ \sigma \rangle$

**by** (*simp add: assms index-sequence-def*)

**have** 2:  $inth\ \langle 0, ilen\ \sigma \rangle\ (ilen\ \langle 0, ilen\ \sigma \rangle) = ilen\ \sigma$

**by** *auto*

**have** 3:  $(\forall i < ilen\ \langle 0, ilen\ \sigma \rangle. f\ (sub\ (inth\ \langle 0, ilen\ \sigma \rangle\ i)\ (inth\ \langle 0, ilen\ \sigma \rangle\ (Suc\ i))\ \sigma))$

**by** (*simp add: PJ3help assms*)

**have** 4:  $ilen\ \langle inth\ \sigma\ (0), inth\ \sigma\ (ilen\ \sigma) \rangle = ilen\ \langle 0, ilen\ \sigma \rangle$

**by** *simp*

**have** 5:  $(\forall i \leq ilen\ \langle inth\ \sigma\ (0), inth\ \sigma\ (ilen\ \sigma) \rangle.$

$inth\ \langle inth\ \sigma\ (0), inth\ \sigma\ (ilen\ \sigma) \rangle\ i = inth\ \sigma\ (inth\ \langle 0, ilen\ \sigma \rangle\ i))$

**using** *antisym-conv2* **by** *fastforce*

**have** 6:  $ilen\ \langle inth\ \sigma\ (0), inth\ \sigma\ (ilen\ \sigma) \rangle = Suc\ 0$

**by** *simp*

**show** *?thesis*

**using** 1 2 3 4 5 6 **by** *blast*

**qed**

**lemma** *PJ3sem*:

$(\sigma \models f \triangle skip = (f \wedge more))$

**proof** –

**have** 1:  $(\sigma \models f \triangle skip) \implies (\sigma \models f \wedge more)$

**by** (*metis (mono-tags, lifting) One-nat-def PJ3help cpl-projection filt-expand index-sequence-def*  
 $more-defs powerinterval-def skip-defs unl-lift2 zero-less-one$ )

**have** 2:  $(\sigma \models f \wedge more) \implies (\sigma \models f \triangle skip)$

**by** (*simp add: projection-d-def skip-defs more-defs powerinterval-def,*  
 $metis PJ3help1 filt-ilen$ )

**show** *?thesis*

**using** 1 2 *unl-lift2* **by** *blast*

**qed**

### 8.3.4 PJ4

**lemma** *PJ4semchaina*:

**assumes**  $(\sigma \models f \triangle (g;h))$

**shows**  $(\sigma \models (f \triangle g) ; (f \triangle h))$

**proof** –

**have** 1:  $(\sigma \models f \triangle (g;h))$

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

```

have 2: ( $\exists l$ . index-sequence 0 l  $\wedge$ 
  inth l (ilen l) = ilen  $\sigma$   $\wedge$ 
  powerinterval f  $\sigma$  l  $\wedge$ 
  ( $\exists n \leq \text{ilen}$  (filt  $\sigma$  l). g (prefix n (filt  $\sigma$  l))  $\wedge$  h (suffix n (filt  $\sigma$  l))))
by (metis assms chop-defs projection-d-def)
obtain l where 3: index-sequence 0 l  $\wedge$ 
  inth l (ilen l) = ilen  $\sigma$   $\wedge$ 
  powerinterval f  $\sigma$  l  $\wedge$ 
  ( $\exists n \leq \text{ilen}$  (filt  $\sigma$  l). g (prefix n (filt  $\sigma$  l))  $\wedge$  h (suffix n (filt  $\sigma$  l)))
  using 2 by auto
have 4: index-sequence 0 l
  using 3 by auto
have 5: powerinterval f  $\sigma$  l
  using 3 by auto
have 6: inth l (ilen l) = ilen  $\sigma$ 
  using 3 by auto
have 7: ( $\exists n \leq \text{ilen}$  (filt  $\sigma$  l). g (prefix n (filt  $\sigma$  l))  $\wedge$  h (suffix n (filt  $\sigma$  l)))
  using 3 by auto
obtain n where 8:  $n \leq \text{ilen}$  (filt  $\sigma$  l)  $\wedge$  g (prefix n (filt  $\sigma$  l))  $\wedge$  h (suffix n (filt  $\sigma$  l))
  using 7 by auto
have 9:  $n \leq \text{ilen}$  (filt  $\sigma$  l)
  using 8 by auto
have 10: g (prefix n (filt  $\sigma$  l))
  using 8 by auto
have 11: h (suffix n (filt  $\sigma$  l))
  using 8 by auto
have 12: index-sequence 0 (prefix n l)
  by (metis 4 8 filt-ilen idx-split)
have 13: index-sequence (inth l n) (suffix n l)
  by (metis 4 9 filt-ilen idx-split)
have 14: index-sequence 0 ((imap (shiftn (inth l n)) (suffix n l)))
  using 13 idx-shiftn by blast
have 15: g (filt  $\sigma$  (prefix n l))
  by (metis 8 filt-ilen filt-prefix)
have 16: h (filt  $\sigma$  (suffix n l))
  by (metis 11 9 filt-ilen filt-suffix)
have 17: g (filt (prefix (inth l n)  $\sigma$ ) (prefix n l))
  by (metis (no-types, lifting) 12 15 4 6 8 filt-ilen filt-prefix-idx
    idx-less-equal ilast-prefix order-refl)
have 18: h (filt (suffix (inth l n)  $\sigma$ ) ((imap (shiftn (inth l n)) (suffix n l))))
proof –
  have 181: ilen((filt  $\sigma$  (suffix n l))) =
    ilen(filt (suffix (inth l n)  $\sigma$ ) ((imap (shiftn (inth l n)) (suffix n l))))
    by (simp add: filt-ilen)
  have 182: ( $\forall j \leq \text{ilen}((\text{filt } \sigma (\text{suffix } n \ l)))$ .
    (inth (filt  $\sigma$  (suffix n l)) j) =
    (inth  $\sigma$  ((inth l (n+j)))))
    )
    by (metis 9 filt-ilen filt-imap inth-imap inth-suffix
      suffix-ilen-good)

```

```

have 183: (∀ j ≤ ilen((filt σ (suffix n l))).
  (inth (filt (suffix (inth l n) σ) ((imap (shiftn (inth l n)) (suffix n l)))) j) =
  (inth (suffix (inth l n) σ) (inth (imap (shiftn (inth l n)) (suffix n l)) j))
)
by (simp add: filt-imap inth-imap)
have 184: (inth l n) ≤ ilen σ
  by (metis 4 6 9 filt-ilen idx-less-equal order-refl)
have 185: (∀ j ≤ ilen((filt σ (suffix n l))).
  (inth (imap (shiftn (inth l n)) (suffix n l)) j) =
  (inth (suffix n l) j) - (inth l n)
)
by (metis 4 6 9 filt-ilen inth-suffix idx-shiftn-suffix-inth
  suffix-ilen-good)
have 186: (∀ j ≤ ilen((filt σ (suffix n l))).
  (inth (suffix n l) j) - (inth l n) = (inth l (j+n)) - (inth l n)
)
by (metis 9 add.commute filt-ilen inth-suffix suffix-ilen-good)
have 187: (∀ j ≤ ilen((filt σ (suffix n l))).
  (inth l (j+n)) - (inth l n) ≤ ilen σ - (inth l n)
)
by (metis 4 6 9 add.commute diff-le-mono eq-imp-le filt-ilen
  idx-less-equal suffix-ilen-good nat-add-left-cancel-le
  ordered-cancel-comm-monoid-diff-class.add-diff-inverse)
have 188: (∀ j ≤ ilen((filt σ (suffix n l))).
  (inth (suffix (inth l n) σ) (inth (imap (shiftn (inth l n)) (suffix n l)) j)) =
  (inth σ ((inth (imap (shiftn (inth l n)) (suffix n l)) j) + (inth l n)))
)
using inth-suffix
by (simp add: 184 185 186 187 add.commute)
have 189: (∀ j ≤ ilen((filt σ (suffix n l))).
  (inth σ ((inth (imap (shiftn (inth l n)) (suffix n l)) j) + (inth l n))) =
  (inth σ ((inth (suffix n l) j)))
)
by (metis 13 185 filt-ilen idx-greater
  ordered-cancel-comm-monoid-diff-class.le-imp-diff-is-add)
have 190: (∀ j ≤ ilen((filt σ (suffix n l))).
  (inth (filt σ (suffix n l)) j) =
  (inth (filt (suffix (inth l n) σ) ((imap (shiftn (inth l n)) (suffix n l)))) j)
)
using 183 188 189 filt-expand by fastforce
show ?thesis
by (metis 16 181 190 interval-eq-inth-eq)
qed
have 19: powerinterval f (prefix (inth l n) σ) (prefix n l)
  by (metis 4 5 6 8 filt-ilen powerinterval-split)
have 20: powerinterval f (suffix (inth l n) σ) ((imap (shiftn (inth l n)) (suffix n l)))
  by (metis 4 5 6 9 filt-ilen powerinterval-split)
have 21: (inth l n) ≤ ilen σ
  by (metis 3 9 filt-ilen idx-less-equal order-refl)
have 22: inth (prefix n l) (ilen (prefix n l)) = (inth l n)

```

**by** (*metis 8 filt-ilen ilast-prefix*)  
**have 23:**  $\text{inth } ((\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l)))$   
 $\quad (\text{ilen } ((\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l)))) = \text{ilen } \sigma - (\text{inth } l \ n)$   
**by** (*metis 4 6 9 eq-imp-le filt-ilen ilen-imap idx-shiftn-suffix-inth*  
*suffix-ilen-good ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)  
**have 24:**  $l = \text{fuse } (\text{prefix } n \ l)$   
 $\quad (\text{imap } (\text{shift } (\text{inth } l \ n)) \ ((\text{imap } (\text{shiftn } (\text{inth } l \ n)) \ (\text{suffix } n \ l))))$   
**using** *fuse-prefix-suffix*  
**by** (*metis 13 14 8 filt-ilen lsk-ls*)  
**have 25:**  $(\exists l1. \text{index-sequence } 0 \ l1 \wedge$   
 $\quad \text{inth } l1 \ (\text{ilen } l1) = \text{ilen } (\text{prefix } (\text{inth } l \ n) \ \sigma) \wedge$   
 $\quad \text{powerinterval } f \ (\text{prefix } (\text{inth } l \ n) \ \sigma) \ l1 \wedge g \ (\text{filt } (\text{prefix } (\text{inth } l \ n) \ \sigma) \ l1))$   
**by** (*metis 12 17 19 21 22 prefix-ilen-good*)  
**have 26:**  $(\exists l2. \text{index-sequence } 0 \ l2 \wedge$   
 $\quad \text{inth } l2 \ (\text{ilen } l2) = \text{ilen } (\text{suffix } (\text{inth } l \ n) \ \sigma) \wedge$   
 $\quad \text{powerinterval } f \ (\text{suffix } (\text{inth } l \ n) \ \sigma) \ l2 \wedge h \ (\text{filt } (\text{suffix } (\text{inth } l \ n) \ \sigma) \ l2))$   
**using** *18 14 20 23 21 by auto*  
**have 27:**  $(\text{prefix } (\text{inth } l \ n) \ \sigma) \models f \triangle g$   
**by** (*metis 25 projection-d-def*)  
**have 28:**  $(\text{suffix } (\text{inth } l \ n) \ \sigma) \models f \triangle h$   
**by** (*metis 26 projection-d-def*)  
**show** *?thesis*  
**using** *21 27 28 chop-defs by auto*  
**qed**

**lemma** *PJ4semchainb:*

**assumes**  $(\sigma \models (f \triangle g) ; (f \triangle h))$

**shows**  $(\sigma \models f \triangle (g;h))$

**proof** –

**have 1:**  $(\exists n \leq \text{ilen } \sigma.$

$(\exists l. \text{index-sequence } 0 \ l \wedge$   
 $\quad \text{inth } l \ (\text{ilen } l) = \text{ilen } (\text{prefix } n \ \sigma) \wedge$   
 $\quad \text{powerinterval } f \ (\text{prefix } n \ \sigma) \ l \wedge g \ (\text{filt } (\text{prefix } n \ \sigma) \ l)) \wedge$   
 $(\exists l. \text{index-sequence } 0 \ l \wedge$   
 $\quad \text{inth } l \ (\text{ilen } l) = \text{ilen } (\text{suffix } n \ \sigma) \wedge$   
 $\quad \text{powerinterval } f \ (\text{suffix } n \ \sigma) \ l \wedge h \ (\text{filt } (\text{suffix } n \ \sigma) \ l)))$

**using** *assms by (metis chop-defs cpl-projection)*

**obtain** *cp where 2:*  $cp \leq \text{ilen } \sigma \wedge$

$(\exists l. \text{index-sequence } 0 \ l \wedge$   
 $\quad \text{inth } l \ (\text{ilen } l) = \text{ilen } (\text{prefix } cp \ \sigma) \wedge$   
 $\quad \text{powerinterval } f \ (\text{prefix } cp \ \sigma) \ l \wedge g \ (\text{filt } (\text{prefix } cp \ \sigma) \ l)) \wedge$   
 $(\exists l. \text{index-sequence } 0 \ l \wedge$   
 $\quad \text{inth } l \ (\text{ilen } l) = \text{ilen } (\text{suffix } cp \ \sigma) \wedge$   
 $\quad \text{powerinterval } f \ (\text{suffix } cp \ \sigma) \ l \wedge h \ (\text{filt } (\text{suffix } cp \ \sigma) \ l))$

**using** *1 by auto*

**have 3:**  $cp \leq \text{ilen } \sigma$

**using** *2 by auto*

**have 4:**  $(\exists l. \text{index-sequence } 0 \ l \wedge$   
 $\quad \text{inth } l \ (\text{ilen } l) = \text{ilen } (\text{prefix } cp \ \sigma) \wedge$   
 $\quad \text{powerinterval } f \ (\text{prefix } cp \ \sigma) \ l \wedge g \ (\text{filt } (\text{prefix } cp \ \sigma) \ l))$

```

using 2 by auto
obtain l1 where 5: index-sequence 0 l1  $\wedge$ 
  inth l1 (ilen l1) = ilen (prefix cp  $\sigma$ )  $\wedge$ 
  powerinterval f (prefix cp  $\sigma$ ) l1  $\wedge$  g (filt (prefix cp  $\sigma$ ) l1)
using 4 by auto
have 6: index-sequence 0 l1
using 5 by auto
have 7: inth l1 (ilen l1) = ilen (prefix cp  $\sigma$ )
using 5 by auto
have 8: powerinterval f (prefix cp  $\sigma$ ) l1
using 5 by auto
have 9: g (filt (prefix cp  $\sigma$ ) l1)
using 5 by auto
have 10: ( $\exists$  l. index-sequence 0 l  $\wedge$ 
  inth l (ilen l) = ilen (suffix cp  $\sigma$ )  $\wedge$ 
  powerinterval f (suffix cp  $\sigma$ ) l  $\wedge$  h (filt (suffix cp  $\sigma$ ) l))
using 2 by auto
obtain l2 where 11: index-sequence 0 l2  $\wedge$ 
  inth l2 (ilen l2) = ilen (suffix cp  $\sigma$ )  $\wedge$ 
  powerinterval f (suffix cp  $\sigma$ ) l2  $\wedge$  h (filt (suffix cp  $\sigma$ ) l2)
using 10 by auto
have 12: index-sequence 0 l2
using 11 by auto
have 13: inth l2 (ilen l2) = ilen (suffix cp  $\sigma$ )
using 11 by auto
have 14: powerinterval f (suffix cp  $\sigma$ ) l2
using 11 by auto
have 15: h (filt (suffix cp  $\sigma$ ) l2)
using 11 by auto
have 16: index-sequence 0 (fuse l1 (imap (shift cp) l2))
  by (metis 11 12 2 5 6 eq-imp-le idx-fuse-idx prefix-ilen-good
    suffix-ilen-good)
have 17: ilast l1 = ifirst (imap (shift cp) l2)
  by (metis 12 13 3 6 7 idx-fuse-ifirst-ilast prefix-ilen-good
    suffix-ilen-good)
have 18: inth (fuse l1 (imap (shift cp) l2)) (ilen l1) = cp
  by (metis 17 3 7 eq-imp-le fuse-ilen-a fuse-inth
    prefix-ilen-good le-add1)
have 19: ilast (fuse l1 (imap (shift cp) l2)) = ilen  $\sigma$ 
proof –
  have 191: ilast (fuse l1 (imap (shift cp) l2)) = ilast (imap (shift cp) l2)
    by (metis 17 eq-imp-le fuse-inth-a fuse-ilen-a)
  have 192: ilast (imap (shift cp) l2) = ilast l2 + cp
    by (metis shift-def ilen-imap inth-imap)
  have 193: ilast l2 + cp = ilen  $\sigma$ 
    by (metis 13 3 Nat.le-imp-diff-is-add suffix-ilen-good)
  show ?thesis using 191 192 193 by auto
qed
have 20: powerinterval f  $\sigma$  (fuse l1 (imap (shift cp) l2))
  using powerinterval-fuse[of l1 (l2) cp  $\sigma$  f]

```

```

using 12 13 14 3 5 by auto
have 21:  $\sigma = \text{fuse } (\text{prefix } cp \ \sigma) \ (\text{suffix } cp \ \sigma)$ 
  by (simp add: 3 fuse-prefix-suffix)
have 22:  $\text{inth } ((\text{fuse } l1 \ (\text{imap } (\text{shift } cp) \ l2))) \ (\text{ilen } l1) = cp$ 
  using 18 by blast
have 23:  $(\text{prefix } (\text{ilen } l1) \ (\text{filt } \sigma \ (\text{fuse } l1 \ (\text{imap } (\text{shift } cp) \ l2))) =$ 
   $(\text{filt } (\text{prefix } cp \ \sigma) \ l1)$ 
  by (metis 17 2 5 filt-prefix filt-prefix-idx fuse-ilen-a
  prefix-fuse prefix-ilen-good le-add1)
have 24:  $g \ (\text{prefix } (\text{ilen } l1) \ (\text{filt } \sigma \ (\text{fuse } l1 \ (\text{imap } (\text{shift } cp) \ l2))))$ 
  by (simp add: 23 9)
have 25:  $(\text{suffix } (\text{ilen } l1) \ (\text{filt } \sigma \ (\text{fuse } l1 \ (\text{imap } (\text{shift } cp) \ l2)))) =$ 
   $(\text{filt } (\text{suffix } cp \ \sigma) \ l2)$ 
  by (metis 12 13 17 2 23 filt-ilen filt-suffix filt-suffix-idx
  prefix-ilen-bound suffix-fuse suffix-ilen-good)
have 26:  $\text{ilen } l1 \leq \text{ilen } (\text{filt } \sigma \ (\text{fuse } l1 \ (\text{imap } (\text{shift } cp) \ l2)))$ 
  by (metis 23 filt-ilen prefix-ilen-bound)
have 27:  $(\exists l. \text{index-sequence } 0 \ l \wedge$ 
   $\text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma \wedge$ 
   $\text{powerinterval } f \ \sigma \ l \wedge$ 
   $(\exists n \leq \text{ilen } (\text{filt } \sigma \ l). \ g \ (\text{prefix } n \ (\text{filt } \sigma \ l)) \wedge h \ (\text{suffix } n \ (\text{filt } \sigma \ l))))$ 
  by (metis 11 16 19 20 24 25 26)
show ?thesis
by (metis 27 chop-fuse fuse-prefix-suffix ilast-ifirst projection-d-def)
qed

```

**lemma** *PJ4sem*:

$(\sigma \models f \triangle (g;h) = (f \triangle g) ; (f \triangle h))$

**using** *PJ4semchaina PJ4semchainb unl-lift2* **by** *blast*

### 8.3.5 PJ5

**lemma** *PJ5sem*:

$(\sigma \models f \triangle \text{init}(g) \longrightarrow \text{init}(g))$

**by** (*simp add: projection-d-def init-defs*)

( *metis filt-inth filt-ilen index-sequence-def ilen-gr-zero* )

### 8.3.6 PJ6

**lemma** *PJ6help1*:

**assumes** *index-sequence* 0 *l*

$(\text{inth } l \ (\text{ilen } l)) = (\text{ilen } \sigma)$

**shows**  $(\forall i. 0 \leq i \wedge i < \text{ilen } l \longrightarrow \text{ilen } (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)$   
 $= (\text{inth } l \ (\text{Suc } i)) - (\text{inth } l \ i))$

**proof**

**fix** *i*

**show**  $0 \leq i \wedge i < \text{ilen } l \longrightarrow$

$\text{ilen } (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma) = \text{inth } l \ (\text{Suc } i) - \text{inth } l \ i$

**using** *assms*

**by** (*simp add: index-sequence-def sub-def*)

(*metis Suc-lessI assms(1) idx-less-last-1 le-diff-iff le-eq-less-or-eq min.orderE*)

qed

**lemma** *PJ6help2:*

**assumes** *index-sequence 0 l*

$inth\ l\ (ilen\ l) = ilen\ \sigma \wedge$

$(\forall i < ilen\ l. inth\ l\ (Suc\ i) - inth\ l\ i = Suc\ 0)$

**shows**  $(\forall i \leq ilen\ l. inth\ l\ i = i)$

**proof**

**fix** *i*

**show**  $i \leq ilen\ l \longrightarrow inth\ l\ i = i$

**proof**

*(induct i)*

**case** *0*

**then show** *?case* **using** *assms index-sequence-def* **by** *blast*

**next**

**case** *(Suc i)*

**then show** *?case*

**by** (*metis One-nat-def Suc-eq-plus1 Suc-leD Suc-le-lessD assms idx-expand  
le-add-diff-inverse2 plus-1-eq-Suc*)

**qed**

qed

**lemma** *PJ6help3:*

**assumes** *index-sequence 0 l*

$inth\ l\ (ilen\ l) = ilen\ \sigma$

$(\forall i \leq ilen\ l. inth\ l\ i = i)$

**shows**  $(\forall i < ilen\ l. inth\ l\ (Suc\ i) - inth\ l\ i = Suc\ 0)$

**proof**

**fix** *i*

**show**  $i < ilen\ l \longrightarrow inth\ l\ (Suc\ i) - inth\ l\ i = Suc\ 0$

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

**qed**

**lemma** *PJ6help4:*

$(\exists l. index-sequence\ 0\ l \wedge l = [0.. \leq ilen\ \sigma] \wedge$

$inth\ l\ (ilen\ l) = ilen\ \sigma \wedge$

$(\forall i \leq ilen\ l. inth\ l\ i = i) \wedge$

$(\exists s1. ilen\ s1 = ilen\ l \wedge (\forall i \leq ilen\ s1. inth\ s1\ i = inth\ \sigma\ (i)) \wedge s1 = \sigma))$

**by** (*simp add: index-sequence-def upt-ilen upt-inth*)

**lemma** *PJ6help5:*

$(\exists l. index-sequence\ 0\ l \wedge$

$inth\ l\ (ilen\ l) = ilen\ \sigma \wedge$



$$\begin{aligned}
& (\forall i < \text{ilen } l. \text{ilen } (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma) = \text{Suc } 0) \wedge \\
& (\exists s1. \text{ilen } s1 = \text{ilen } l \wedge (\forall i \leq \text{ilen } s1. \text{inth } s1 \ i = \text{inth } \sigma \ (\text{inth } l \ i)) \wedge g \ s1)) \\
& = g \ \sigma \\
\text{proof } & - \\
\text{have } & (\exists l. \text{index-sequence } 0 \ l \wedge \\
& \text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma \wedge \\
& (\forall i < \text{ilen } l. \text{ilen } (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma) = \text{Suc } 0) \wedge \\
& (\exists s1. \text{ilen } s1 = \text{ilen } l \wedge (\forall i \leq \text{ilen } s1. \text{inth } s1 \ i = \text{inth } \sigma \ (\text{inth } l \ i)) \wedge g \ s1)) \\
& = \\
& (\exists l. \text{index-sequence } 0 \ l \wedge \\
& \text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma \wedge \\
& (\forall i < \text{ilen } l. \text{inth } l \ (\text{Suc } i) - \text{inth } l \ i = \text{Suc } 0) \wedge \\
& (\exists s1. \text{ilen } s1 = \text{ilen } l \wedge (\forall i \leq \text{ilen } s1. \text{inth } s1 \ i = \text{inth } \sigma \ (\text{inth } l \ i)) \wedge g \ s1)) \\
& \text{using } PJ6help1 \text{ by } (\text{metis zero-order}(1)) \\
\text{also have } & \dots = \\
& (\exists l. \text{index-sequence } 0 \ l \wedge \\
& \text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma \wedge \\
& (\forall i \leq \text{ilen } l. \text{inth } l \ i = i) \wedge \\
& (\exists s1. \text{ilen } s1 = \text{ilen } l \wedge (\forall i \leq \text{ilen } s1. \text{inth } s1 \ i = \text{inth } \sigma \ (\text{inth } l \ i)) \wedge g \ s1)) \\
& \\
& \text{using } PJ6help2 \ PJ6help3 \text{ by } \text{blast} \\
\text{also have } & \dots = \\
& (\exists l. \text{index-sequence } 0 \ l \wedge \\
& \text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma \wedge \\
& (\forall i \leq \text{ilen } l. \text{inth } l \ i = i) \wedge \\
& (\exists s1. \text{ilen } s1 = \text{ilen } l \wedge (\forall i \leq \text{ilen } s1. \text{inth } s1 \ i = \text{inth } \sigma \ (i)) \wedge g \ s1)) \\
& \\
& \text{by } \text{metis} \\
\text{also have } & \dots = \\
& (\exists l. \text{index-sequence } 0 \ l \wedge \\
& \text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma \wedge \\
& (\forall i \leq \text{ilen } l. \text{inth } l \ i = i) \wedge \\
& (\exists s1. \text{ilen } s1 = \text{ilen } l \wedge (\forall i \leq \text{ilen } s1. \text{inth } s1 \ i = \text{inth } \sigma \ (i)) \wedge s1 = \sigma \wedge g \ \sigma)) \\
& \\
& \text{by } (\text{metis interval-eq-inth-eq le-eq-less-or-eq}) \\
\text{also have } & \dots = \\
& g \ \sigma \\
& \\
& \text{using } PJ6help4 \text{ by } \text{blast} \\
\text{finally show } & (\exists l. \text{index-sequence } 0 \ l \wedge \\
& \text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma \wedge \\
& (\forall i < \text{ilen } l. \text{ilen } (\text{sub } (\text{inth } l \ i) (\text{inth } l \ (\text{Suc } i)) \ \sigma) = \text{Suc } 0) \wedge \\
& (\exists s1. \text{ilen } s1 = \text{ilen } l \wedge (\forall i \leq \text{ilen } s1. \text{inth } s1 \ i = \text{inth } \sigma \ (\text{inth } l \ i)) \wedge g \ s1)) \\
& = g \ \sigma \\
& \cdot \\
& \text{qed} \\
& \\
& \text{lemma } PJ6sem: \\
& (\sigma \models \text{skip } \Delta \ g = g) \\
& \text{proof } -
\end{aligned}$$

**have 1:**  $(\sigma \models \text{skip} \triangle g = g) =$   
 $((\exists l. \text{index-sequence } 0\ l \wedge \text{inth } l\ (\text{ilen } l) = \text{ilen } \sigma \wedge$   
 $(\forall i < \text{ilen } l. (\text{sub } (\text{inth } l\ i)\ (\text{inth } l\ (\text{Suc } i))\ \sigma) \models \text{skip}) \wedge g\ (\text{filt } \sigma\ l)) =$   
 $g\ \sigma)$   
**by** (*simp add: projection-d-def powerinterval-def*)  
**have 2:**  $(\exists l. \text{index-sequence } 0\ l \wedge \text{inth } l\ (\text{ilen } l) = \text{ilen } \sigma \wedge$   
 $(\forall i < \text{ilen } l. (\text{sub } (\text{inth } l\ i)\ (\text{inth } l\ (\text{Suc } i))\ \sigma) \models \text{skip}) \wedge g\ (\text{filt } \sigma\ l)) =$   
 $(\exists l\ s1. \text{index-sequence } 0\ l \wedge \text{inth } l\ (\text{ilen } l) = \text{ilen } \sigma \wedge$   
 $(\forall i < \text{ilen } l. (\text{sub } (\text{inth } l\ i)\ (\text{inth } l\ (\text{Suc } i))\ \sigma) \models \text{skip}) \wedge$   
 $\text{ilen } s1 = \text{ilen } l \wedge$   
 $(\forall i \leq \text{ilen } s1. (\text{inth } s1\ i) = (\text{inth } \sigma\ (\text{inth } l\ i)))) \wedge$   
 $g\ s1)$   
**using** *filt-expand* **by** *metis*  
**have 3:**  $(\exists l\ s1. \text{index-sequence } 0\ l \wedge \text{inth } l\ (\text{ilen } l) = \text{ilen } \sigma \wedge$   
 $(\forall i < \text{ilen } l. (\text{sub } (\text{inth } l\ i)\ (\text{inth } l\ (\text{Suc } i))\ \sigma) \models \text{skip}) \wedge$   
 $\text{ilen } s1 = \text{ilen } l \wedge$   
 $(\forall i \leq \text{ilen } s1. (\text{inth } s1\ i) = (\text{inth } \sigma\ (\text{inth } l\ i)))) \wedge$   
 $g\ s1) =$   
 $(\exists l. \text{index-sequence } 0\ l \wedge \text{inth } l\ (\text{ilen } l) = \text{ilen } \sigma \wedge$   
 $(\forall i < \text{ilen } l. \text{ilen } (\text{sub } (\text{inth } l\ i)\ (\text{inth } l\ (\text{Suc } i))\ \sigma) = \text{Suc } 0) \wedge$   
 $(\exists s1. \text{ilen } s1 = \text{ilen } l \wedge (\forall i \leq \text{ilen } s1. \text{inth } s1\ i = \text{inth } \sigma\ (\text{inth } l\ i)) \wedge g\ s1))$   
**by** (*simp add: skip-defs*)  
**have 4:**  $(\exists l. \text{index-sequence } 0\ l \wedge \text{inth } l\ (\text{ilen } l) = \text{ilen } \sigma \wedge$   
 $(\forall i < \text{ilen } l. \text{ilen } (\text{sub } (\text{inth } l\ i)\ (\text{inth } l\ (\text{Suc } i))\ \sigma) = \text{Suc } 0) \wedge$   
 $(\exists s1. \text{ilen } s1 = \text{ilen } l \wedge (\forall i \leq \text{ilen } s1. \text{inth } s1\ i = \text{inth } \sigma\ (\text{inth } l\ i)) \wedge g\ s1)) =$   
 $(g\ \sigma)$   
**by** (*simp add: PJ6help5*)  
**from 1 2 3 4 show ?thesis**  
**by** *simp*  
**qed**

### 8.3.7 PJ7

**lemma** *PJemptyImp:*

**assumes**  $\text{ilen } \sigma = 0$

**shows**  $(\sigma \models (f \triangle g) = g)$

**using** *assms*

**by** (*simp add: projection-d-def index-sequence-def powerinterval-def,*  
*auto,*  
*metis filt.simps(1) INil-ilen lessI not-less-zero old.nat.exhaust,*  
*metis inth.simps(1) filt.simps(1) suffix-ilast suffix-zero*  
*ilen.simps(1) not-less-zero*)

**lemma** *PJ7empty:*

**assumes**  $\text{ilen } \sigma = 0$

**shows**  $(\sigma \models f \triangle (g \triangle h) = (f \triangle g) \triangle h)$

**proof** –

**have 1:**  $(\sigma \models f \triangle (g \triangle h) = (g \triangle h))$

**using** *PJemptyImp assms* **by** *blast*

**have 2:**  $(\sigma \models (g \triangle h) = h)$

```

  using PJemptyImp assms by blast
have 3:  $(\sigma \models (f \triangle g) \triangle h = h)$ 
  using PJemptyImp assms by blast
from 1 2 3 show ?thesis by simp
qed

```

**lemma** *PJ7helpchain1a-help-1:*

**assumes** *index-sequence 0 l*

$(\text{inth } l (\text{ilen } l)) = \text{ilen } \sigma$

$(\forall i < \text{ilen } l. (\text{sub } (\text{inth } l i) (\text{inth } l (\text{Suc } i)) \sigma) \models f \triangle g)$

$\text{ilen } \sigma > 0$

**shows** *index-sequence 0 (lfuse ( (lcpl f g  $\sigma$  l)))*

**proof** –

**have** 0:  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen } l > 0$

**using** *assms gr-zeroI index-sequence-def* **by** *fastforce*

**have** 01:  $\text{ifirst } l = 0$

**using** *assms index-sequence-def* **by** *auto*

**have** 02:  $\text{lastfirst } (\text{lcpl } f g \sigma l)$

**using** *assms lcpl-lfuse-lastfirst* **by** *(simp add: projection-d-def)*

*(metis 0 01 assms(3) lcpl-lfuse-lastfirst)*

**have** 1:  $\text{ilen } \sigma > 0 \longrightarrow \text{ifirst } (\text{lfuse } (\text{lcpl } f g \sigma l)) = 0$

**using** *assms 0 01 02*

*lcpl-ifirst[of l  $\sigma$  f g]*

**by** *(metis (mono-tags, lifting) lastfirst-lfuse)*

**from** 1 0 **show** ?thesis **using** *assms lcpl-lfuse-idx[of l  $\sigma$  f g]* **by** *(simp add: projection-d-def)*

**qed**

**lemma** *PJ7helpchain1a-help-2:*

**assumes** *index-sequence 0 l*

$(\text{inth } l (\text{ilen } l)) = \text{ilen } \sigma$

$(\forall i < \text{ilen } l. (\text{sub } (\text{inth } l i) (\text{inth } l (\text{Suc } i)) \sigma) \models f \triangle g)$

$\text{ilen } \sigma > 0$

**shows** *powerinterval f  $\sigma$  (lfuse ( (lcpl f g  $\sigma$  l)))*

**proof** –

**have** 1:  $(\forall i < \text{ilen } l.$

$\text{powerinterval } f (\text{sub } (\text{inth } l i) (\text{inth } l (\text{Suc } i)) \sigma)$

$(\text{cpl } f g (\text{sub } (\text{inth } l i) (\text{inth } l (\text{Suc } i)) \sigma)))$

**using** *assms cpl-projection* **by** *blast*

**have** 2:  $\forall i < \text{ilen } l.$

$\forall ia < \text{ilen } (\text{cpl } f g (\text{sub } (\text{inth } l i) (\text{inth } l (\text{Suc } i)) \sigma)).$

$f (\text{sub } (\text{inth } (\text{cpl } f g (\text{sub } (\text{inth } l i) (\text{inth } l (\text{Suc } i)) \sigma)) ia)$

$(\text{inth } (\text{cpl } f g (\text{sub } (\text{inth } l i) (\text{inth } l (\text{Suc } i)) \sigma)) (\text{Suc } ia))$

$(\text{sub } (\text{inth } l i) (\text{inth } l (\text{Suc } i)) \sigma))$

**using** 1 **by** *(simp add: powerinterval-def)*

**have** 3:  $\forall i < \text{ilen } l.$

$\forall ia < \text{ilen } (\text{cpl } f g (\text{sub } (\text{inth } l i) (\text{inth } l (\text{Suc } i)) \sigma)).$

$(\text{sub } (\text{inth } (\text{cpl } f g (\text{sub } (\text{inth } l i) (\text{inth } l (\text{Suc } i)) \sigma)) ia)$

$(\text{inth } (\text{cpl } f g (\text{sub } (\text{inth } l i) (\text{inth } l (\text{Suc } i)) \sigma)) (\text{Suc } ia))$

$(\text{sub } (\text{inth } l i) (\text{inth } l (\text{Suc } i)) \sigma)) =$

$(\text{sub } (\text{inth } (\text{imap } (\text{shift } (\text{inth } l i)) (\text{cpl } f g (\text{sub } (\text{inth } l i) (\text{inth } l (\text{Suc } i)) \sigma)))) ia)$

$(\text{inth } (\text{imap } (\text{shift } (\text{inth } l \ i)) \ (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma))) \ (\text{Suc } ia))$   
 $\sigma)$

**proof** –

**have** 30:  $\forall i < \text{ilen } l.$

$\forall ia < \text{ilen } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)).$   
 $\forall j \leq \text{ilen } (\text{sub } (\text{inth } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ ia)$   
 $(\text{inth } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ (\text{Suc } ia))$   
 $(\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)).$   
 $(\text{inth } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ (\text{Suc } ia)) \leq$   
 $(\text{inth } l \ (\text{Suc } i)) - (\text{inth } l \ (i))$

**using** *assms* **by** (*metis* *add.commute* *cpl-projection* *idx-expand* *ilen-sub* *plus-1-eq-Suc*)

**have** 31:  $\forall i < \text{ilen } l.$

$\forall ia < \text{ilen } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)).$   
 $\text{ilen } (\text{sub } (\text{inth } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ ia)$   
 $(\text{inth } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ (\text{Suc } ia))$   
 $(\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)) =$   
 $\text{ilen } (\text{sub } (\text{inth } (\text{imap } (\text{shift } (\text{inth } l \ i))$   
 $(\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma))) \ ia)$   
 $(\text{inth } (\text{imap } (\text{shift } (\text{inth } l \ i))$   
 $(\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma))) \ (\text{Suc } ia))$   
 $\sigma)$

**using** *assms* **unfolding** *shift-def* *inth-imap*

**by** (*metis* (*no-types*, *lifting*) *PJ6help1* *add.commute* *cpl-projection* *idx-expand* *sub-sub-1* *le-add1* *le-add-same-cancel1* *plus-1-eq-Suc*)

**have** 32:  $\forall i < \text{ilen } l.$

$\forall ia < \text{ilen } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)).$   
 $\forall j \leq \text{ilen } (\text{sub } (\text{inth } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ ia)$   
 $(\text{inth } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ (\text{Suc } ia))$   
 $(\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)).$   
 $\text{inth } (\text{sub } (\text{inth } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ ia)$   
 $(\text{inth } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ (\text{Suc } ia))$   
 $(\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)) \ j =$   
 $\text{inth } (\text{sub } (\text{inth } (\text{imap } (\text{shift } (\text{inth } l \ i))$   
 $(\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma))) \ ia)$   
 $(\text{inth } (\text{imap } (\text{shift } (\text{inth } l \ i))$   
 $(\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma))) \ (\text{Suc } ia))$   
 $\sigma) \ j$

**using** *assms* 30 *sub-sub-1*[*of* - - -  $\sigma$ ]

**by** (*auto* *simp* *add: shift-def* *inth-imap* *cpl-projection* *idx-expand* )  
(*metis* *add.commute* *idx-expand* *plus-1-eq-Suc*)

**show** ?thesis **using** 31 32 *interval-eq-inth-eq* **by** (*simp* *add: interval-eq-inth-eq*)

**qed**

**have** 4:  $\forall i < \text{ilen } l.$

$\forall ia < \text{ilen } (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma)).$   
 $f \ (\text{sub } (\text{inth } (\text{imap } (\text{shift } (\text{inth } l \ i)) \ (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma))) \ ia)$   
 $(\text{inth } (\text{imap } (\text{shift } (\text{inth } l \ i)) \ (\text{cpl } f \ g \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma))) \ (\text{Suc } ia))$   
 $\sigma)$

**using** 2 3 **by** *auto*

**have** 5:  $\text{ilen}(\text{lcpl } f \ g \ \sigma \ l) = \text{ilen } l - 1$

**using** *assms index-sequence-def lcpl-ilen lcpl-ilen-zero* **by** *fastforce*  
**have** 6:  $\text{ilen } \sigma > 0 \longrightarrow \text{ilen } l > 0$   
**using** *assms gr-zeroI index-sequence-def* **by** *fastforce*  
**have** 7:  $\forall i < \text{ilen } l.$   
 $\quad \forall ia < \text{ilen } ((\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i)).$   
 $\quad \quad f \ (\text{sub } (\text{inth } (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i) \ ia)$   
 $\quad \quad \quad (\text{inth } (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i) \ (\text{Suc } ia))$   
 $\quad \quad \quad \sigma)$   
**using** *assms* **by** (*metis (no-types, lifting) 4 index-sequence-def ilen-imap lcpl-inth*)  
**have** 8:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $\quad (\forall i \leq \text{ilen}(\text{lcpl } f \ g \ \sigma \ l).$   
 $\quad \quad \forall ia < \text{ilen } ((\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i)).$   
 $\quad \quad \quad f \ (\text{sub } (\text{inth } (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i) \ ia)$   
 $\quad \quad \quad \quad (\text{inth } (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i) \ (\text{Suc } ia))$   
 $\quad \quad \quad \sigma))$   
**by** (*metis 5 6 7 One-nat-def Suc-pred le-imp-less-Suc*)  
**have** 9:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $\quad \text{powerinterval } f \ \sigma \ (\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l))) =$   
 $\quad (\forall i < \text{ilen } (\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l)).$   
 $\quad \quad f \ (\text{sub } (\text{inth } (\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l)) \ i) \ (\text{inth } (\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l)) \ (\text{Suc } i)) \ \sigma))$   
**by** (*simp add: powerinterval-def*)  
**have** 10:  $\text{ilen } \sigma > 0 \longrightarrow \text{lastfirst } (\text{lcpl } f \ g \ \sigma \ l)$   
**using** 6 *assms index-sequence-def lcpl-lfuse-lastfirst* **by** *blast*  
**have** 11:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $\quad (\forall j \leq \text{ilen } (\text{lcpl } f \ g \ \sigma \ l). \text{ilen}(\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ j) > 0)$   
  
**by** (*simp add: assms lcpl-ilen-inth-gr-zero*)  
**have** 12:  $\text{ilen } \sigma > 0 \longrightarrow$   
 $\quad (\forall i \leq \text{ilen}(\text{lcpl } f \ g \ \sigma \ l).$   
 $\quad \quad (\forall ia < \text{ilen } ((\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i)).$   
 $\quad \quad \quad f \ (\text{sub } (\text{inth } (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i) \ ia)$   
 $\quad \quad \quad \quad (\text{inth } (\text{inth } (\text{lcpl } f \ g \ \sigma \ l) \ i) \ (\text{Suc } ia))$   
 $\quad \quad \quad \sigma))) =$   
 $\quad \quad (\forall j < \text{ilen } (\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l)).$   
 $\quad \quad \quad f \ (\text{sub } (\text{inth } (\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l)) \ j)$   
 $\quad \quad \quad \quad (\text{inth } (\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l)) \ (\text{Suc } j))$   
 $\quad \quad \quad \sigma))$   
  
**using** *lfuse-split[of (lcpl f g σ l) f σ]* 10 11 **by** *auto*  
**show** *?thesis* **using** 12 8 9 **using** *assms(4)* **by** *blast*  
**qed**

**lemma** *PJ7helpchain1a-help-3:*

**assumes** *index-sequence 0 l*

$(\text{inth } l \ (\text{ilen } l)) = \text{ilen } \sigma$

$(\forall i < \text{ilen } l. (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ \sigma) \models f \ \Delta \ g)$

$h \ (\text{filt } \sigma \ l)$

$\text{ilen } \sigma > 0$

**shows**  $\text{ilast } (\text{lfuse } (\text{lcpl } f \ g \ \sigma \ l)) = \text{ilen } \sigma$

**proof** –

**have** 0:  $ilen \sigma > 0 \longrightarrow ilen l > 0$   
**using** *assms gr-zeroI index-sequence-def* **by** *fastforce*  
**have** 1:  $ilen \sigma > 0 \longrightarrow$   
 $ilast (lfuse (lcpl f g \sigma l)) = ilast(ilast ( (lcpl f g \sigma l)))$   
**using** *assms*  
**using** 0 *index-sequence-def lastfirst-lfuse-ilast lcpl-lfuse-lastfirst* **by** *blast*  
**have** 2:  $ilen \sigma > 0 \longrightarrow$   
 $ilast ( (lcpl f g \sigma l))$   
 $= (inth (lcpl f g \sigma l) (ilen l - 1))$   
**using** *assms* **by** (*simp add: 0 index-sequence-def lcpl-ilen*)  
**have** 3:  $ilen \sigma > 0 \longrightarrow$   
 $(inth (lcpl f g \sigma l) (ilen l - 1)) =$   
 $(imap (shift (inth l (ilen l - 1)))$   
 $(cpl f g (sub (inth l (ilen l - 1)) (inth l (Suc (ilen l - 1))) \sigma)))$   
  
**using** *assms* **by** (*simp add: 0 index-sequence-def lcpl-inth*)  
**have** 4:  $ilen \sigma > 0 \longrightarrow$   
 $ilast( ( (cpl f g (sub (inth l (ilen l - 1)) (inth l (Suc (ilen l - 1))) \sigma))) =$   
 $ilen ((sub (inth l (ilen l - 1)) (inth l (Suc (ilen l - 1))) \sigma))$   
**using** 0 *assms*  
**by** (*metis cpl-projection diff-less zero-less-one*)  
**have** 5:  $ilen \sigma > 0 \longrightarrow$   
 $ilen ((sub (inth l (ilen l - 1)) (inth l (Suc (ilen l - 1))) \sigma)) =$   
 $(inth l (Suc (ilen l - 1))) - (inth l (ilen l - 1))$   
  
**using** 0 *PJ6help1 assms diff-less zero-less-one* **by** *blast*  
**have** 6:  $ilen \sigma > 0 \longrightarrow$   
 $ilast ( (imap (shift (inth l (ilen l - 1)))$   
 $(cpl f g (sub (inth l (ilen l - 1)) (inth l (Suc (ilen l - 1))) \sigma))) =$   
 $(shift (inth l (ilen l - 1)))$   
 $(ilast (cpl f g (sub (inth l (ilen l - 1)) (inth l (Suc (ilen l - 1))) \sigma)))$   
  
**by** (*metis ilen-imap inth-imap*)  
**have** 7:  $ilen \sigma > 0 \longrightarrow$   
 $(shift (inth l (ilen l - 1)))$   
 $(ilast (cpl f g (sub (inth l (ilen l - 1)) (inth l (Suc (ilen l - 1))) \sigma))) =$   
 $(shift (inth l (ilen l - 1))) ((inth l (Suc (ilen l - 1))) - (inth l (ilen l - 1)))$   
  
**using** 4 5 **by** *auto*  
**have** 8:  $ilen \sigma > 0 \longrightarrow$   
 $(shift (inth l (ilen l - 1))) ((inth l (Suc (ilen l - 1))) - (inth l (ilen l - 1))) =$   
 $((inth l (Suc (ilen l - 1))) - (inth l (ilen l - 1))) + (inth l (ilen l - 1))$   
  
**by** (*simp add: shift-def*)  
**have** 9:  $ilen \sigma > 0 \longrightarrow$   
 $((inth l (Suc (ilen l - 1))) - (inth l (ilen l - 1))) + (inth l (ilen l - 1))$   
 $= (inth l (Suc (ilen l - 1)))$   
  
**using** *assms*  
**by** (*metis 0 add commute diff-add diff-less idx-expand plus-1-eq-Suc zero-less-one*)

**have** 10:  $ilen \sigma > 0 \longrightarrow (inth\ l\ (Suc\ (ilen\ l - 1))) = ilen\ \sigma$   
**by** (*simp add: 0 assms*)  
**show** ?thesis  
**using** 1 10 2 3 6 7 8 9 *assms*(5) **by** *presburger*  
**qed**

**lemma** *PJ7helpchain1a-help-4:*

**assumes** *index-sequence 0 l*  
 $(inth\ l\ (ilen\ l)) = ilen\ \sigma$   
 $(\forall\ i < ilen\ l. (sub\ (inth\ l\ i)\ (inth\ l\ (Suc\ i))\ \sigma) \models f \triangle g)$   
 $h\ (filt\ \sigma\ l)$   
 $ilen\ \sigma > 0$   
**shows**  $((filt\ \sigma\ (lfuse\ (lcpl\ f\ g\ \sigma\ l))) \models g \triangle h)$   
**proof** –  
**have** 0:  $ilen\ \sigma > 0 \longrightarrow ilen\ l > 0$   
**using** *assms gr-zeroI index-sequence-def* **by** *fastforce*  
**have** 1:  $ilen\ \sigma > 0 \longrightarrow index-sequence\ 0\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))$   
**by** (*simp add: assms lcpl-ilen-inth-gr-zero lsum-addzero-idx*)  
**have** 2:  $ilen\ \sigma > 0 \longrightarrow$   
 $(inth\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))\ (ilen\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0)))) =$   
 $ilen\ (filt\ \sigma\ (lfuse\ (lcpl\ f\ g\ \sigma\ l)))$   
**by** (*simp add: assms lcpl-lfuse-filt-ilen*)  
**have** 3:  $ilen\ \sigma > 0 \longrightarrow$   
 $powerinterval\ g\ (filt\ \sigma\ (lfuse\ (lcpl\ f\ g\ \sigma\ l)))\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0))$   
  
**by** (*simp add: assms lcpl-lfuse-filt-power*)  
**have** 4:  $ilen\ \sigma > 0 \longrightarrow$   
 $h\ (filt\ (filt\ \sigma\ (lfuse\ (lcpl\ f\ g\ \sigma\ l)))\ (addzero\ (lsum\ (lcpl\ f\ g\ \sigma\ l)\ 0)))$   
**by** (*simp add: assms filt-imap-filt lcpl-filt-lfuse-lsum*)  
**show** ?thesis **by** (*metis 1 2 3 4 assms*(5) *projection-d-def*)  
**qed**

**lemma** *PJ7helpchain1a:*

**assumes**  $ilen\ \sigma > 0$   
 $(\sigma \models (f \triangle g) \triangle h)$   
**shows**  $(\sigma \models f \triangle (g \triangle h))$   
**proof** –  
**have** 1:  $ilen\ \sigma > 0$   
**using** *assms* **by** *auto*  
**have** 2:  $(\exists\ l. index-sequence\ 0\ l \wedge inth\ l\ (ilen\ l) = ilen\ \sigma \wedge$   
 $powerinterval\ (LIFT(f \triangle g))\ \sigma\ l \wedge$   
 $h\ (filt\ \sigma\ l))$   
**using** *assms* **using** *cpl-projection* **by** *blast*  
**obtain** *l* **where** 3:  $index-sequence\ 0\ l \wedge inth\ l\ (ilen\ l) = ilen\ \sigma \wedge$   
 $powerinterval\ (LIFT(f \triangle g))\ \sigma\ l \wedge$   
 $h\ (filt\ \sigma\ l)$   
**using** 2 **by** *blast*  
**have** 4:  $index-sequence\ 0\ l \wedge inth\ l\ (ilen\ l) = ilen\ \sigma \wedge$   
 $(\forall\ i < ilen\ l. (sub\ (inth\ l\ i)\ (inth\ l\ (Suc\ i))\ \sigma) \models f \triangle g) \wedge$

```

      h (filt  $\sigma$  l)
using 3 by (simp add: powerinterval-def)
have 6:  $\text{ilen } l > 0$ 
  using 1 4 gr-zeroI index-sequence-def by fastforce
have 7:  $\text{index-sequence } 0 \text{ (lfuse (lcpl } f \ g \ \sigma \ l))$ 
by (metis (no-types, lifting) 1 4 PJ7helpchain1a-help-1)
have 8:  $\text{powerinterval } f \ \sigma \text{ (lfuse (lcpl } f \ g \ \sigma \ l))$ 
  using 4 6 PJ7helpchain1a-help-2 assms index-sequence-def by auto
have 9:  $(\text{inth (lfuse (lcpl } f \ g \ \sigma \ l)) (\text{ilen (lfuse (lcpl } f \ g \ \sigma \ l)))) = \text{ilen } \sigma$ 
  by (metis 1 4 PJ7helpchain1a-help-3)
have 10:  $((\text{filt } \sigma \text{ (lfuse (lcpl } f \ g \ \sigma \ l))) \models g \ \Delta \ h)$ 
  by (simp add: 1 4 6 PJ7helpchain1a-help-4)
show ?thesis
using 10 7 8 9 by (metis projection-d-def)
qed

```

```

lemma PJ7helpchain1b:
assumes  $\text{ilen } \sigma > 0$ 
       $(\sigma \models f \ \Delta (g \ \Delta \ h))$ 
shows  $(\sigma \models (f \ \Delta \ g) \ \Delta \ h)$ 
proof -
  have 1:  $\text{ilen } \sigma > 0$ 
  using assms by auto
  have 2:  $(\exists l. \text{index-sequence } 0 \ l \wedge \text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma \wedge$ 
     $\text{powerinterval } f \ \sigma \ l \wedge$ 
     $((\text{filt } \sigma \ l) \models g \ \Delta \ h))$ 
  using assms by (simp add: projection-d-def)
  obtain l where 3:  $\text{index-sequence } 0 \ l \wedge \text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma \wedge$ 
     $\text{powerinterval } f \ \sigma \ l \wedge$ 
     $((\text{filt } \sigma \ l) \models g \ \Delta \ h)$ 
  using 2 by blast
  have 4:  $(\exists la. \text{index-sequence } 0 \ la \wedge \text{inth } la \ (\text{ilen } la) = \text{ilen}(\text{filt } \sigma \ l) \wedge$ 
     $\text{powerinterval } g \ (\text{filt } \sigma \ l) \ la \wedge$ 
     $((\text{filt } (\text{filt } \sigma \ l) \ la) \models h))$ 
  using 3 using cpl-projection by blast
  obtain la where 5:  $\text{index-sequence } 0 \ la \wedge \text{inth } la \ (\text{ilen } la) = \text{ilen}(\text{filt } \sigma \ l) \wedge$ 
     $\text{powerinterval } g \ (\text{filt } \sigma \ l) \ la \wedge$ 
     $((\text{filt } (\text{filt } \sigma \ l) \ la) \models h)$ 
  using 4 by blast
  have 6:  $\text{ilen } l > 0$ 
  using 1 3 gr0I index-sequence-def by force
  have 7:  $\text{ilen}(\text{filt } \sigma \ l) = \text{ilen } l$ 
  by (simp add: filt-ilen)
  have 8:  $(\text{filt } (\text{filt } \sigma \ l) \ la) = (\text{filt } \sigma \ (\text{filt } l \ la))$ 
  using filt-imap-filt by blast
  have 9:  $(\text{inth } (\text{filt } l \ la) \ (\text{ilen } (\text{filt } l \ la)))) = \text{ilen } \sigma$ 
  by (metis 3 5 filt-expand order-refl)
  have 10:  $\text{ilen } la > 0$ 
  using 5 6 7 gr0I index-sequence-def by force

```



```

have 11:  $\text{ilen}(\text{filt } l \text{ la}) > 0$ 
  by (simp add: 10 filt-ilen)
have 12:  $\text{index-sequence } 0(\text{filt } l \text{ la})$ 
proof -
  have 111:  $\text{inth}(\text{filt } l \text{ la}) 0 = 0$ 
    by (metis 3 5 filt-inth index-sequence-def ilen-gr-zero)
  have 112:  $\text{ilen}(\text{filt } l \text{ la}) = \text{ilen } la$ 
    using filt-expand by blast
  have 113:  $(\forall i < \text{ilen } la. (\text{inth}(\text{filt } l \text{ la}) i) = (\text{inth } l (\text{inth } la i)))$ 
    by (simp add: filt-imap inth-imap)
  have 114:  $(\forall i < \text{ilen } la. (\text{inth}(\text{filt } l \text{ la}) (\text{Suc } i)) = (\text{inth } l (\text{inth } la (\text{Suc } i))))$ 
    by (simp add: filt-imap inth-imap)
  have 115:  $(\forall i < \text{ilen } la. (\text{inth } l (\text{inth } la i)) < (\text{inth } l (\text{inth } la (\text{Suc } i))))$ 
    by (metis 3 5 7 Suc-lessI idx-less-than idx-less-last-1 lessI less-imp-le-nat)
  show ?thesis by (simp add: 111 113 114 115 filt-ilen index-sequence-def)
qed
have 20:  $\text{powerinterval}(\text{LIFT}(f \triangle g)) \sigma(\text{filt } l \text{ la})$ 
proof -
  have 201:  $\text{powerinterval}(\text{LIFT}(f \triangle g)) \sigma(\text{filt } l \text{ la}) =$ 
     $(\forall i < \text{ilen } la. (\text{sub}(\text{inth } l (\text{inth } la i)) (\text{inth } l (\text{inth } la (\text{Suc } i))) \sigma) \models f \triangle g)$ 
    by (simp add: filt-imap inth-imap powerinterval-def)
  have 202:  $(\forall i < \text{ilen } la. (\text{sub}(\text{inth } l (\text{inth } la i)) (\text{inth } l (\text{inth } la (\text{Suc } i))) \sigma) \models f \triangle g) =$ 
     $(\forall i < \text{ilen } la.$ 
       $(\exists ll. \text{index-sequence } 0 ll$ 
         $\wedge \text{ilast } ll = \text{ilen}(\text{sub}(\text{inth } l (\text{inth } la i)) (\text{inth } l (\text{inth } la (\text{Suc } i))) \sigma) \wedge$ 
         $\text{powerinterval } f(\text{sub}(\text{inth } l (\text{inth } la i)) (\text{inth } l (\text{inth } la (\text{Suc } i))) \sigma) ll \wedge$ 
         $((\text{filt}(\text{sub}(\text{inth } l (\text{inth } la i)) (\text{inth } l (\text{inth } la (\text{Suc } i))) \sigma) ll) \models g)$ 
       $))$ 

    by (simp add: projection-d-def)
  have 203:  $(\forall i < \text{ilen } la. \text{ilen}(\text{sub}(\text{inth } l (\text{inth } la i)) (\text{inth } l (\text{inth } la (\text{Suc } i))) \sigma) =$ 
     $(\text{inth } l (\text{inth } la (\text{Suc } i))) - (\text{inth } l (\text{inth } la i)))$ 
    by (metis 3 5 7 Suc-leI Suc-lessI idx-less-equal idx-less-last-1 ilen-sub lessI less-imp-le-nat)
  have 2041:  $(\forall i < \text{ilen } la.$ 
     $(\text{inth } la (\text{Suc } i)) \leq \text{ilen } l$ 
   $)$ 

  using 5 7 Suc-lessI idx-less-last-1 by fastforce
  have 204:  $(\forall i < \text{ilen } la.$ 
     $\text{ilen}(\text{imap}(\text{shiftn}(\text{inth } l (\text{inth } la i))) (\text{sub}(\text{inth } la i) (\text{inth } la (\text{Suc } i)) l)) =$ 
     $(\text{inth } la (\text{Suc } i)) - (\text{inth } la i))$ 

    by (simp add: 5 PJ6help1 filt-ilen)
  have 205:  $(\forall i < \text{ilen } la.$ 
     $(\forall j \leq (\text{inth } la (\text{Suc } i)) - (\text{inth } la i).$ 
     $(\text{inth}(\text{sub}(\text{inth } la i) (\text{inth } la (\text{Suc } i)) l) j) =$ 
     $(\text{inth } l ((\text{inth } la i) + j))$ 
   $)$ 

```

)

**using** 2041 5 *index-sequence-def inth-sub order.strict-implies-order* **by** blast

**have** 2060:  $(\forall i < \text{ilen } la. \text{ inth } la \ i \leq \text{ inth } la \ (\text{Suc } i))$  )

**using** 5 *idx-expand* **by** fastforce

**have** 206:  $(\forall i < \text{ilen } la. (\forall j \leq \text{ inth } la \ (\text{Suc } i)) - \text{ inth } la \ i. (\text{ inth } l \ (\text{ inth } la \ i)) \leq \text{ inth } l \ ((\text{ inth } la \ i) + j))$  )

)

**using** 2041 3 2060 *idx-less-equal* **by** fastforce

**have** 207:  $(\forall i < \text{ilen } la. (\forall j \leq \text{ inth } la \ (\text{Suc } i)) - \text{ inth } la \ i. (\text{ inth } (\text{imap } (\text{shiftn } (\text{ inth } l \ (\text{ inth } la \ i)))) (\text{sub } (\text{ inth } la \ i) (\text{ inth } la \ (\text{Suc } i)) \ l)) \ j) = (\text{ inth } l \ ((\text{ inth } la \ i) + j)) - (\text{ inth } l \ (\text{ inth } la \ i))$  ))

**by** (simp add: 205 inth-*imap shiftn-def*)

**have** 208:  $(\forall i < \text{ilen } la. (\text{ inth } (\text{imap } (\text{shiftn } (\text{ inth } l \ (\text{ inth } la \ i)))) (\text{sub } (\text{ inth } la \ i) (\text{ inth } la \ (\text{Suc } i)) \ l)) \ 0) = 0$  )

**by** (simp add: 207)

**have** 209:  $(\forall i < \text{ilen } la. \text{ ilast } (\text{imap } (\text{shiftn } (\text{ inth } l \ (\text{ inth } la \ i)))) (\text{sub } (\text{ inth } la \ i) (\text{ inth } la \ (\text{Suc } i)) \ l)) = (\text{ inth } l \ (\text{ inth } la \ (\text{Suc } i))) - (\text{ inth } l \ (\text{ inth } la \ i))$  )

**using** 204 207 5 2060 **by** simp-all

**have** 210:  $(\forall i < \text{ilen } la. \text{ index-sequence } 0 \ (\text{imap } (\text{shiftn } (\text{ inth } l \ (\text{ inth } la \ i)))) (\text{sub } (\text{ inth } la \ i) (\text{ inth } la \ (\text{Suc } i)) \ l))$  )

**by** (metis 3 5 7 add.commute *idx-expand idx-shiftn idx-sub plus-1-eq-Suc*)

**have** 211:  $(\forall i < \text{ilen } la. \text{ powerinterval } f \ (\text{sub } (\text{ inth } l \ (\text{ inth } la \ i)) (\text{ inth } l \ (\text{ inth } la \ (\text{Suc } i)))) \ \sigma) (\text{imap } (\text{shiftn } (\text{ inth } l \ (\text{ inth } la \ i)))) (\text{sub } (\text{ inth } la \ i) (\text{ inth } la \ (\text{Suc } i)) \ l))$  )

**proof** –

**have** 2111:  $(\forall i < \text{ilen } la. \text{ powerinterval } f \ (\text{sub } (\text{ inth } l \ (\text{ inth } la \ i)) (\text{ inth } l \ (\text{ inth } la \ (\text{Suc } i)))) \ \sigma) (\text{imap } (\text{shiftn } (\text{ inth } l \ (\text{ inth } la \ i)))) (\text{sub } (\text{ inth } la \ i) (\text{ inth } la \ (\text{Suc } i)) \ l))$  ) =

$(\forall i < \text{ilen } la. (\forall ia < \text{ilen } (\text{sub } (\text{ inth } la \ i) (\text{ inth } la \ (\text{Suc } i)) \ l). \text{ powerinterval } f \ (\text{sub } (\text{ inth } l \ (\text{ inth } la \ i)) (\text{ inth } l \ (\text{ inth } la \ (\text{Suc } i)))) \ \sigma) (\text{imap } (\text{shiftn } (\text{ inth } l \ (\text{ inth } la \ i)))) (\text{sub } (\text{ inth } la \ i) (\text{ inth } la \ (\text{Suc } i)) \ l))$  )

```

      f (sub (inth (imap (shiftn (inth l (inth la i)))
        (sub (inth la i) (inth la (Suc i)) l)) ia)
        (inth (imap (shiftn (inth l (inth la i)))
          (sub (inth la i) (inth la (Suc i)) l)) (Suc ia))
        (sub (inth l (inth la i)) (inth l (inth la (Suc i)) σ) )
    )
  )
  by (simp add: powerinterval-def)
have 2112: ... =
  (∀ i < iLen la.
    (∀ ia < (inth la (Suc i)) - (inth la i).
      f (sub (inth (imap (shiftn (inth l (inth la i)))
        (sub (inth la i) (inth la (Suc i)) l)) ia)
        (inth (imap (shiftn (inth l (inth la i)))
          (sub (inth la i) (inth la (Suc i)) l)) (Suc ia))
        (sub (inth l (inth la i)) (inth l (inth la (Suc i)) σ) )
      )
    )
  )
  using 204 by auto
have 2113: ... =
  (∀ i < iLen la.
    (∀ ia < (inth la (Suc i)) - (inth la i).
      f (sub ((inth l ((inth la i) + ia)) - (inth l (inth la i)))
        ((inth l ((inth la i) + (Suc ia))) - (inth l (inth la i)))
        (sub (inth l (inth la i)) (inth l (inth la (Suc i)) σ) )
      )
    )
  )
  using 207 by auto
have 2114:
  (∀ i < iLen la.
    (let m = (inth la (Suc i)); n = (inth la i) in
      (∀ ia < m - n.
        (inth l (n + ia)) ≤ ((inth l (n + (Suc ia))) )
      )
    )
  )
  using 2041 3 2060 idx-less-equal by simp-all fastforce
have 2115:
  (∀ i < iLen la.
    (let m = (inth la (Suc i)); n = (inth la i) in
      (∀ ia < m - n.
        (inth l (n + ia)) - (inth l n) ≤ ((inth l ((Suc (n + ia)))) - (inth l n))
      )
    )
  )
  by (metis 2114 add-Suc-right diff-le-mono)
have 2116:
  (∀ i < iLen la.
    (let m = (inth la (Suc i)); n = (inth la i) in
      (∀ ia < m - n.

```

```

      ((inth l ((Suc (n+ia)))) - (inth l n)) ≤ (inth l m) - (inth l n)
    )
  )
)
by (metis 3 5 7 idx-expand Suc-leI add.commute diff-le-mono
idx-less-equal less-diff-conv plus-1-eq-Suc)
have 2117: (∀ i<ilen la.
  (let m = (inth la (Suc i)); n= (inth la i) in
    (inth l m) ≤ ilen σ ))
by (metis 12 9 Suc-lessI filt-inth filt-ilen idx-less-last-1
less-or-eq-imp-le)
have 2118: (∀ i<ilen la.
  (let m = (inth la (Suc i)); n= (inth la i) in
    (∀ ia< m - n.
      (n+(Suc ia)) ≤ ilen l )
    )
  )
)
using 5 7 idx-expand less-diff-conv by fastforce
have 21190: (∀ i<ilen la.
  (let m = (inth la (Suc i)); n= (inth la i) in
    (∀ ia< m - n.
      ((inth l (n +ia)) - (inth l n)) ≤ ((inth l (n+(Suc ia))) - (inth l n))
    ) ))
)
by (meson 2114 diff-le-mono)
have 21191: (∀ i<ilen la.
  (let m = (inth la (Suc i)); n= (inth la i) in
    (∀ ia< m - n.
      ((inth l (n+(Suc ia))) - (inth l n)) ≤
        ilen (sub (inth l n) (inth l m) σ)
    )))
)
by (simp add: 203 2116)
have 2119: (∀ i<ilen la.
  (let m = (inth la (Suc i)); n= (inth la i) in
    (∀ ia< m - n.
      ilen (sub ((inth l (n +ia)) - (inth l n))
        ((inth l (n+(Suc ia))) - (inth l n))
        (sub (inth l n) (inth l m) σ)) =
      ((inth l (n+(Suc ia))) - ((inth l (n +ia)))
    )
  )
)
)
by (metis (no-types, lifting) 206 21190 21191
diff-diff-add ilen-sub le-add-diff-inverse less-imp-le)
have 2120: (∀ i<ilen la.
  (let m = (inth la (Suc i)); n= (inth la i) in
    (∀ ia< m - n.
      (∀ j ≤ ((inth l (n+(Suc ia)))) - ((inth l (n +ia))).
    )
  )
)

```

```

      inth (sub ((inth l (n + ia)) - (inth l n))
        ((inth l (n+(Suc ia))) - (inth l n))
        (sub (inth l n) (inth l m) σ)) j =
      inth (sub (inth l n) (inth l m) σ) (((inth l (n + ia)) - (inth l n))+j)
    ) )))
  by (metis 2119 21190 21191 len-sub inth-sub)
have 212111: (∀ i < len la.
  (let m = (inth la (Suc i)); n = (inth la i) in
    (∀ ia < m - n.
      ((inth l (n + ia))) ≤ ((inth l (n+(Suc ia)))))
    )
  )
)

  using 2114 by blast
have 212112: (∀ i < len la.
  (let m = (inth la (Suc i)); n = (inth la i) in
    (∀ ia < m - n.
      ((inth l (n))) ≤ ((inth l (n+(ia)))))
    )
  )
)

  by (simp add: 206)
have 212113: (∀ i < len la.
  (let m = (inth la (Suc i)); n = (inth la i) in
    (∀ ia < m - n.
      (∀ j ≤ ((inth l (n+(Suc ia))))) - ((inth l (n + ia))).
      ((inth l (n + ia)) )+j ≤ (inth l m)
    )
  )
)

  using 212111 212112 206 2060 2116
  unfolding Let-def by fastforce
have 21211: (∀ i < len la.
  (let m = (inth la (Suc i)); n = (inth la i) in
    (∀ ia < m - n.
      (∀ j ≤ ((inth l (n+(Suc ia))))) - ((inth l (n + ia))).
      ((inth l (n + ia)) - (inth l n))+j ≤ (inth l m) - (inth l n)
    )
  )
)

  by (metis 212112 212113 Nat.add-diff-assoc2 diff-le-mono)
have 2121: (∀ i < len la.
  (let m = (inth la (Suc i)); n = (inth la i) in
    (∀ ia < m - n.
      (∀ j ≤ ((inth l (n+(Suc ia))))) - ((inth l (n + ia))).
      (inth (sub (inth l n) (inth l m) σ) (((inth l (n + ia)) - (inth l n))+j) ) =

```

```

      (inth σ ((inth l n) + (((inth l (n + ia)) - (inth l n)) + j)) )
    )
  )
)

by (metis 206 2117 21211 add-diff-inverse-nat inth-sub nat-diff-split
    not-less-zero order-refl)
have 2122: (∀ i < ilen la.
  (let m = (inth la (Suc i)); n = (inth la i) in
    (∀ ia < m - n.
      (∀ j ≤ ((inth l (n + (Suc ia)))) - ((inth l (n + ia))))
        (inth σ ((inth l n) + (((inth l (n + ia)) - (inth l n)) + j)) ) =
        (inth σ ( (((inth l (n + ia)) + j)) )
      )
    )
  )
)
by (metis 206 add.assoc le-add-diff-inverse less-imp-le-nat)
have 2123: (∀ i < ilen la.
  (let m = (inth la (Suc i)); n = (inth la i) in
    (∀ ia < m - n.
      (∀ j ≤ ((inth l (n + (Suc ia)))) - ((inth l (n + ia))))
        inth (sub (inth l (n + ia)) ((inth l (n + (Suc ia)))) σ) j =
        (inth σ ( (((inth l (n + ia)) + j)) )
      )
    )
  )
)
by (metis 2114 2118 3 eq-imp-le idx-less-equal inth-sub)
have 2124: (∀ i < ilen la.
  (let m = (inth la (Suc i)); n = (inth la i) in
    (∀ ia < m - n.
      ilen (sub (inth l (n + ia)) ((inth l (n + (Suc ia)))) σ) =
      ((inth l (n + (Suc ia)))) - ((inth l (n + ia)))
    )
  )
)
by (metis 2118 3 Suc-eq-plus1 Suc-le-lessD add-Suc-right idx-expand
    ilen-sub)
have 2125: (∀ i < ilen la.
  (let m = (inth la (Suc i)); n = (inth la i) in
    (∀ ia < m - n.
      (∀ j ≤ ((inth l (n + (Suc ia)))) - ((inth l (n + ia))))
        inth (sub ((inth l (n + ia)) - (inth l n))
          ((inth l (n + (Suc ia)) - (inth l n))
            (sub (inth l n) (inth l m) σ)) j =
        (inth (sub (inth l (n + ia)) ((inth l (n + (Suc ia)))) σ) j)
      )
    )
  )
)

```

)  
)  
)  
)

**by** (*metis 2120 2121 2122 2123*)

**have 2126:**  $(\forall i < \text{ilen } la.$

(let  $m = (\text{inth } la \text{ (Suc } i)); n = (\text{inth } la \text{ } i)$  in  
 $(\forall ia < m - n.$   
 $\text{ilen } (\text{sub } ((\text{inth } l \text{ (} n + ia)) - (\text{inth } l \text{ } n))$   
 $((\text{inth } l \text{ (} n + (\text{Suc } ia))) - (\text{inth } l \text{ } n))$   
 $(\text{sub } (\text{inth } l \text{ } n) (\text{inth } l \text{ } m) \sigma)) =$   
 $\text{ilen } (\text{sub } (\text{inth } l \text{ (} n + ia)) ((\text{inth } l \text{ (} n + (\text{Suc } ia)))) \sigma)$   
 $)$   
 $)$   
 $)$

**by** (*metis 2119 2124*)

**have 2127:**  $(\forall i < \text{ilen } la.$

(let  $m = (\text{inth } la \text{ (Suc } i)); n = (\text{inth } la \text{ } i)$  in  
 $(\forall ia < m - n.$   
 $(\text{sub } ((\text{inth } l \text{ (} n + ia)) - (\text{inth } l \text{ } n))$   
 $((\text{inth } l \text{ (} n + (\text{Suc } ia))) - (\text{inth } l \text{ } n))$   
 $(\text{sub } (\text{inth } l \text{ } n) (\text{inth } l \text{ } m) \sigma)) =$   
 $(\text{sub } (\text{inth } l \text{ (} n + ia)) ((\text{inth } l \text{ (} n + (\text{Suc } ia)))) \sigma)$   
 $)$   
 $)$   
 $)$

**using** *interval-eq-inth-eq 2125 2126*

**by** (*metis 2119*)

**have 2128:**  $(\forall i < \text{ilen } la.$

(let  $m = (\text{inth } la \text{ (Suc } i)); n = (\text{inth } la \text{ } i)$  in  
 $(\forall ia < m - n.$   
 $f \text{ (sub } (\text{inth } l \text{ (} n + ia)) ((\text{inth } l \text{ (} n + (\text{Suc } ia)))) \sigma)$   
 $)$   
 $)$   
 $)$

**by** (*metis 2041 3 add.commute add-Suc-right less-diff-conv less-le-trans*  
*powerinterval-def*)

**show** *?thesis* **using** *2111 2112 2113 2127 2128* **unfolding** *Let-def* **by** (*simp add: 2113*)

**qed**

**have 220:**  $(\forall i < \text{ilen } la.$

$((\text{filt } (\text{sub } (\text{inth } l \text{ (} \text{inth } la \text{ } i)) (\text{inth } l \text{ (} \text{inth } la \text{ (Suc } i))) \sigma)$   
 $(\text{imap } (\text{shiftm } (\text{inth } l \text{ (} \text{inth } la \text{ } i))) (\text{sub } (\text{inth } la \text{ } i) (\text{inth } la \text{ (Suc } i)) l)) \text{ )}$   
 $\models g)$   
 $)$

**proof** –

**have 2201:**  $(\forall i < \text{ilen } la.$

$$\begin{aligned} & \text{ilen } (\text{filt } (\text{sub } (\text{inth } l \text{ (inth } la \text{ } i)) \text{ (inth } l \text{ (inth } la \text{ (Suc } i))) } \sigma) \\ & \quad (\text{imap } (\text{shiftn } (\text{inth } l \text{ (inth } la \text{ } i))) \\ & \quad \quad (\text{sub } (\text{inth } la \text{ } i) \text{ (inth } la \text{ (Suc } i) \text{ } l)) \text{ } ) = \\ & \text{ilen } (\text{imap } (\text{shiftn } (\text{inth } l \text{ (inth } la \text{ } i))) \text{ (sub } (\text{inth } la \text{ } i) \text{ (inth } la \text{ (Suc } i) \text{ } l))} \\ & \quad ) \end{aligned}$$

**using** *filt-ilen by blast*

**have** 2202:  $(\forall \text{ } i < \text{ilen } la.$   

$$\begin{aligned} & \text{ilen } (\text{imap } (\text{shiftn } (\text{inth } l \text{ (inth } la \text{ } i))) \\ & \quad (\text{sub } (\text{inth } la \text{ } i) \text{ (inth } la \text{ (Suc } i) \text{ } l)) = \\ & \quad (\text{inth } la \text{ (Suc } i)) - (\text{inth } la \text{ } i) \end{aligned}$$

$$)$$

**using** 204 **by** *blast*

**have** 2203:  $(\forall \text{ } i < \text{ilen } la.$   

$$\begin{aligned} & (\forall \text{ } j \leq (\text{inth } la \text{ (Suc } i)) - (\text{inth } la \text{ } i) . \\ & \text{inth } ((\text{filt } (\text{sub } (\text{inth } l \text{ (inth } la \text{ } i)) \text{ (inth } l \text{ (inth } la \text{ (Suc } i))) } \sigma) \\ & \quad (\text{imap } (\text{shiftn } (\text{inth } l \text{ (inth } la \text{ } i))) \\ & \quad \quad (\text{sub } (\text{inth } la \text{ } i) \text{ (inth } la \text{ (Suc } i) \text{ } l)) \text{ } ) \text{ } j = \\ & \text{inth } (\text{sub } (\text{inth } l \text{ (inth } la \text{ } i)) \text{ (inth } l \text{ (inth } la \text{ (Suc } i))) } \sigma) \\ & \quad (\text{inth } (\text{imap } (\text{shiftn } (\text{inth } l \text{ (inth } la \text{ } i))) \\ & \quad \quad (\text{sub } (\text{inth } la \text{ } i) \text{ (inth } la \text{ (Suc } i) \text{ } l)) \text{ } j) \\ & \quad )) \end{aligned}$$

**by** *(simp add: filt-imap inth-imap)*

**have** 2204:  $(\forall \text{ } i < \text{ilen } la.$   

$$\begin{aligned} & (\forall \text{ } j \leq (\text{inth } la \text{ (Suc } i)) - (\text{inth } la \text{ } i) . \\ & \text{inth } (\text{sub } (\text{inth } l \text{ (inth } la \text{ } i)) \text{ (inth } l \text{ (inth } la \text{ (Suc } i))) } \sigma) \\ & \quad (\text{inth } (\text{imap } (\text{shiftn } (\text{inth } l \text{ (inth } la \text{ } i))) \\ & \quad \quad (\text{sub } (\text{inth } la \text{ } i) \text{ (inth } la \text{ (Suc } i) \text{ } l)) \text{ } j) = \\ & \text{inth } (\text{sub } (\text{inth } l \text{ (inth } la \text{ } i)) \text{ (inth } l \text{ (inth } la \text{ (Suc } i))) } \sigma) \\ & \quad ((\text{inth } (\text{sub } (\text{inth } la \text{ } i) \text{ (inth } la \text{ (Suc } i) \text{ } l) \text{ } j) - (\text{inth } l \text{ (inth } la \text{ } i))) \\ & \quad )) \end{aligned}$$

**by** *(simp add: inth-imap shiftn-def)*

**have** 2205:  $(\forall \text{ } i < \text{ilen } la.$   

$$\begin{aligned} & ( \\ & \quad (\text{inth } l \text{ (inth } la \text{ } i)) \leq (\text{inth } l \text{ (inth } la \text{ (Suc } i))) \wedge \\ & \quad (\text{inth } l \text{ (inth } la \text{ (Suc } i))) \leq \text{ilen } \sigma \\ & ) \end{aligned}$$

**by** *(metis 12 9 add.commute filt-ilen filt-imap idx-expand inth-imap plus-1-eq-Suc)*

**have** 2206 :  $(\forall \text{ } i < \text{ilen } la.$   

$$\begin{aligned} & (\forall \text{ } j \leq (\text{inth } la \text{ (Suc } i)) - (\text{inth } la \text{ } i) . \\ & \quad (\text{inth } (\text{sub } (\text{inth } la \text{ } i) \text{ (inth } la \text{ (Suc } i) \text{ } l) \text{ } j) = \\ & \quad \quad (\text{inth } l \text{ ((inth } la \text{ } i) + j)) \\ & \quad )) \end{aligned}$$

**using** 205 **by** *blast*



```

have 2207: (∀ i < ilen la.
  (∀ j ≤ (inth la (Suc i)) - (inth la i) .
    ((inth l (inth la i)) + ((inth (sub (inth la i) (inth la (Suc i)) l) j)
      - (inth l (inth la i)))) =
      (inth l ((inth la i) + j))
  ))

  by (simp add: 206 2206)
have 2208: (∀ i < ilen la.
  (∀ j ≤ (inth la (Suc i)) - (inth la i) .
    inth (sub (inth l (inth la i)) (inth l (inth la (Suc i)))) σ)
    ((inth (sub (inth la i) (inth la (Suc i)) l) j) - (inth l (inth la i))) =
    inth σ (inth l ((inth la i) + j))
  ))
  using 2041 206 2060 2205 3 idx-less-equal[of l σ]
  using Nat.le-diff-conv2 by auto
have 2209: (∀ i < ilen la.
  (∀ j ≤ (inth la (Suc i)) - (inth la i) .
    (inth (filt σ l) ((inth la i) + j)) = (inth σ (inth l ((inth la i) + j)))
  ))

  by (simp add: filt-imap inth-imap)
have 2210: (∀ i < ilen la.
  ilen (sub (inth la i) (inth la (Suc i)) (filt σ l)) =
  (inth la (Suc i)) - (inth la i)
  )

  using 5 PJ6help1 by blast
have 2211: (∀ i < ilen la.
  (∀ j ≤ (inth la (Suc i)) - (inth la i) .
    (inth (sub (inth la i) (inth la (Suc i)) (filt σ l)) j) =
    (inth σ (inth l ((inth la i) + j)))
  ))

  using 2209 5 idx-expand inth-sub by fastforce
have 2212: (∀ i < ilen la.
  (∀ j ≤ (inth la (Suc i)) - (inth la i) .
    inth ((filt (sub (inth l (inth la i)) (inth l (inth la (Suc i)))) σ)
      (imap (shiftn (inth l (inth la i)))
        (sub (inth la i) (inth la (Suc i)) l)) ) ) j =
    (inth (sub (inth la i) (inth la (Suc i)) (filt σ l)) j)
  ))

  by (simp add: 2203 2204 2208 2211)
have 2213: (∀ i < ilen la.
  ilen ((filt (sub (inth l (inth la i)) (inth l (inth la (Suc i)))) σ)
    (imap (shiftn (inth l (inth la i)))
      (sub (inth la i) (inth la (Suc i)) l)) ) ) =
  ilen (sub (inth la i) (inth la (Suc i)) (filt σ l))
  )

```

```

    by (metis 2202 2210 filt-ilen)
  have 2214: (∀ i < ilen la.
    (filt (sub (inth l (inth la i)) (inth l (inth la (Suc i))) σ)
      (imap (shiftn (inth l (inth la i))) (sub (inth la i) (inth la (Suc i) l))) =
      (sub (inth la i) (inth la (Suc i)) (filt σ l))
    )

  using interval-eq-inth-eq 2212 2213 using 2201 2202 by fastforce
  have 2215: (∀ i < ilen la.
    (sub (inth la i) (inth la (Suc i)) (filt σ l)) ⊨ g
    )
  using 5 powerinterval-def by blast
  show ?thesis by (simp add: 2214 2215)
qed
show ?thesis
using 201 202 203 209 210 211 220 by metis
qed
show ?thesis
by (metis 12 20 5 8 9 projection-d-def)
qed

```

```

lemma PJ7sem:
(σ ⊨ f Δ (g Δ h) = (f Δ g) Δ h)
proof -
  have 1: ilen σ > 0 ⟶ (σ ⊨ f Δ (g Δ h) = (f Δ g) Δ h)
    using PJ7helpchain1a PJ7helpchain1b unl-lift2 by blast
  have 2: ilen σ = 0 ⟶ (σ ⊨ f Δ (g Δ h) = (f Δ g) Δ h)
    using PJ7empty by blast
  from 1 2 show ?thesis by auto
qed

```

### 8.3.8 PJ8

```

lemma PJ8semhelp:
  assumes index-sequence 0 l
    inth l (ilen l) = ilen σ
    (∀ n na. na + n ≤ ilen σ ⟶ f (sub n (n + na) σ) ⟶ g (sub n (n + na) σ))
  shows
    (∀ i < ilen l. f (sub (inth l i) (inth l (Suc i)) σ)
      ⟶ g (sub (inth l i) (inth l (Suc i)) σ)
    )
  by (metis add.commute assms idx-expand
    ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc)

```

```

lemma PJ8sem:
(σ ⊨ ba(f ⟶ g) ⟶ (f Δ h) ⟶ (g Δ h))

```

**using** *PJ8semhelp* **by** (*simp add: projection-d-def ba-defs powerinterval-def*) *blast*

### 8.3.9 PJ9

**lemma** *PJ9sem*:

$(\sigma \models f \nabla (g \longrightarrow h) \longrightarrow f \triangle g \longrightarrow f \triangle h)$

**by** (*simp add: uprojection-d-def projection-d-def, metis*)

## 8.4 Axioms

**lemma** *BpGen*:

**assumes**  $\vdash f$

**shows**  $\vdash bp\ f$

**using** *assms*

**by** (*simp add: bp-d-def uprojection-d-def projection-d-def Valid-def*)

**lemma** *PJ1*:

$\vdash f \triangle (g \vee h) \longrightarrow f \triangle g \vee f \triangle h$

**using** *PJ1sem Valid-def* **by** *blast*

**lemma** *PJ2*:

$\vdash f \triangle empty = empty$

**using** *PJ2sem Valid-def* **by** *blast*

**lemma** *PJ3*:

$\vdash f \triangle skip = (f \wedge more)$

**using** *PJ3sem Valid-def* **by** *blast*

**lemma** *PJ4*:

$\vdash f \triangle (g;h) = (f \triangle g) ; (f \triangle h)$

**using** *PJ4sem Valid-def* **by** *blast*

**lemma** *PJ5*:

$\vdash f \triangle init(g) \longrightarrow init(g)$

**using** *PJ5sem Valid-def* **by** *blast*

**lemma** *PJ6*:

$\vdash skip \triangle g = g$

**using** *PJ6sem Valid-def* **by** *blast*

**lemma** *PJ7*:

$\vdash f \triangle (g \triangle h) = (f \triangle g) \triangle h$

**using** *PJ7sem Valid-def* **by** *blast*

**lemma** *PJ8*:

$\vdash ba(f \longrightarrow g) \longrightarrow (f \triangle h) \longrightarrow (g \triangle h)$

**using** *PJ8sem Valid-def* **by** *blast*

**lemma** *PJ9*:

$\vdash f \nabla (g \longrightarrow h) \longrightarrow f \triangle g \longrightarrow f \triangle h$

**using** *PJ9sem Valid-def* **by** *blast*

## 8.5 Time Reversal

**lemma** *filt-iapp*:

$\text{filt } w \ (l \ominus \langle x \rangle) = (\text{filt } w \ l) \ominus \langle (\text{inth } w \ x) \rangle$

**proof**

(*induct*  $l$ )

**case** (*INil*  $x$ )

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

**next**

**case** (*ICons*  $x1a \ l$ )

**then show** *?case*

**by** *simp*

**qed**

**lemma** *filt-irev*:

**assumes**  $\forall \ i \leq \text{ilen } l. (\text{inth } l \ i) \leq \text{ilen } w$

**shows**  $(\text{filt } (\text{irev } w) \ l) = (\text{irev } (\text{filt } w \ (\text{imap } (\lambda x. \text{ilen } w - x) \ (\text{irev } l))))$

**using** *assms*

**proof**

(*induct*  $l$ )

**case** (*INil*  $x$ )

**then show** *?case*

**proof** –

**have** *01*:  $\text{filt } (\text{irev } w) \ \langle x \rangle = \langle \text{inth } (\text{irev } w) \ x \rangle$

**by** *simp*

**have** *02*:  $\langle \text{inth } (\text{irev } w) \ x \rangle = \langle \text{inth } w \ (\text{ilen } w - x) \rangle$

**using** *INil.premis irev-inth* **by** *auto*

**have** *03*:  $\text{irev } (\text{filt } w \ (\text{imap } ((-) \ (\text{ilen } w)) \ \langle x \rangle)) = \langle \text{inth } w \ (\text{ilen } w - x) \rangle$

**by** *simp*

**from** *01 02 03* **show** *?thesis* **by** *auto*

**qed**

**next**

**case** (*ICons*  $x1a \ l$ )

**then show** *?case*

**proof** –

**have** *1*:  $\text{filt } (\text{irev } w) \ (x1a \odot l) =$

$(\text{inth } (\text{irev } w) \ x1a) \odot (\text{filt } (\text{irev } w) \ l)$

**by** *simp*

**have** *2*:  $(\text{inth } (\text{irev } w) \ x1a) = (\text{inth } w \ (\text{ilen } w - x1a))$

**using** *ICons.premis irev-inth* **by** *fastforce*

**have** *3*:  $\text{irev } (\text{filt } w \ (x1a \odot l)) = \text{irev } ((\text{inth } w \ x1a) \odot (\text{filt } w \ l))$

**by** *simp*

**have** *4*:  $\text{irev } (\text{filt } w \ (\text{imap } ((-) \ (\text{ilen } w)) \ (\text{irev } (x1a \odot l)))) =$   
 $\text{irev } (\text{filt } w \ (\text{imap } ((-) \ (\text{ilen } w)) \ ((\text{irev } l) \ominus \langle x1a \rangle)))$

**by** *simp*

**have** *5*:  $\text{irev } (\text{filt } w \ (\text{imap } ((-) \ (\text{ilen } w)) \ ((\text{irev } l) \ominus \langle x1a \rangle))) =$   
 $\text{irev } ((\text{filt } w \ (\text{imap } ((-) \ (\text{ilen } w)) \ (\text{irev } l))) \ominus \langle \text{inth } w \ (\text{ilen } w - x1a) \rangle)$

**by** (*simp add: filt-iapp*)

**have** 6:  $\text{irev } ( (\text{filt } w \ (\text{imap } ((-) \ (\text{ilen } w)) \ (\text{irev } l))) \ominus \langle \text{inth } w \ (\text{ilen } w - x1a) \ \rangle =$   
 $(\text{inth } w \ (\text{ilen } w - x1a)) \odot \text{irev} ( (\text{filt } w \ (\text{imap } ((-) \ (\text{ilen } w)) \ (\text{irev } l))))$   
**by** *auto*  
**have** 7:  $\forall i \leq \text{ilen } l. (\text{inth } l \ i) \leq \text{ilen } w$   
**using** *ICons(2)* **by** *auto*  
**have** 8:  $(\text{filt } (\text{irev } w) \ l) = \text{irev}(\text{filt } w \ (\text{imap } ((-) \ (\text{ilen } w)) \ (\text{irev } l)))$   
**using** 7 *ICons.hyps* **by** *blast*  
**show** *?thesis*  
**using** 2 5 8 **by** *auto*  
**qed**  
**qed**

**lemma** *ProjectionRevsema:*

**assumes**  $(\sigma \models (f \triangle g)^r)$

**shows**  $(\sigma \models (f^r) \triangle (g^r))$

**proof** –

**have** 1:  $(\sigma \models (f \triangle g)^r)$

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

**have** 2:  $\exists l. \text{index-sequence } 0 \ l \wedge \text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma \wedge$   
 $\text{powerinterval } f \ (\text{irev } \sigma) \ l \wedge g \ (\text{filt } (\text{irev } \sigma) \ l)$

**using** 1 **by** (*simp add: projection-d-def reverse-d-def*)

**obtain** *l* **where** 3:  $\text{index-sequence } 0 \ l \wedge \text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma \wedge$   
 $\text{powerinterval } f \ (\text{irev } \sigma) \ l \wedge g \ (\text{filt } (\text{irev } \sigma) \ l)$

**using** 2 **by** *auto*

**have** 4:  $\text{index-sequence } 0 \ l$

**using** 3 **by** *auto*

**have** 5:  $\text{inth } l \ (\text{ilen } l) = \text{ilen } \sigma$

**using** 3 **by** *auto*

**have** 6:  $\text{powerinterval } f \ (\text{irev } \sigma) \ l$

**using** 3 **by** *auto*

**have** 7:  $g \ (\text{filt } (\text{irev } \sigma) \ l)$

**using** 3 **by** *auto*

**have** 8:  $\forall i \leq \text{ilen } l. (\text{inth } l \ i) \leq \text{ilen } \sigma$

**using** 4 5 *idx-less-last-1 le-eq-less-or-eq* **by** *fastforce*

**have** 9:  $g \ (\text{irev}(\text{filt } \sigma \ (\text{imap } (\lambda x. \text{ilen } \sigma - x) \ (\text{irev } l)))$

**using** 7 8 *filt-irev* **by** *fastforce*

**have** 10:  $\text{inth } (\text{imap } ((-) \ (\text{ilen } \sigma)) \ (\text{irev } l)) \ 0 = 0$

**by** (*metis 5 diff-self-eq-0 ifirst-irev inth-imap*)

**have** 11:  $(\forall n < \text{ilen } l.$

$\text{inth } (\text{imap } ((-) \ (\text{ilen } \sigma)) \ (\text{irev } l)) \ n$

$< \text{inth } (\text{imap } ((-) \ (\text{ilen } \sigma)) \ (\text{irev } l)) \ (\text{Suc } n))$

**by** (*simp add: inth-imap irev-inth*)

(*metis (no-types, lifting) 4 5 Suc-diff-Suc Suc-less-eq diff-less-Suc diff-less-mono2*  
*index-sequence-def idx-less-last-1*)

**have** 12:  $\text{index-sequence } 0 \ (\text{imap } (\lambda x. \text{ilen } \sigma - x) \ (\text{irev } l))$

**by** (*simp add: 10 11 index-sequence-def*)

**have** 13:  $\text{ilen } (\text{imap } (\lambda x. \text{ilen } \sigma - x) \ (\text{irev } l)) = \text{ilen } l$

**by** *auto*

**have** 14:  $\text{inth } (\text{imap } ((-) \ (\text{ilen } \sigma)) \ (\text{irev } l)) \ (\text{ilen } (\text{imap } ((-) \ (\text{ilen } \sigma)) \ (\text{irev } l))) =$   
 $\text{ilen } \sigma$

by (metis 4 diff-zero index-sequence-def ilast-irev ilen-imap  
 inth-imap)  
 have 15:  $\forall i < \text{ilen } l.$   

$$(\text{inth } (\text{imap } ((-) (\text{ilen } \sigma)) (\text{irev } l)) i) \leq$$

$$(\text{inth } (\text{imap } ((-) (\text{ilen } \sigma)) (\text{irev } l)) (\text{Suc } i))$$
  
 by (simp add: 11 less-imp-le-nat)  
 have 16:  $\forall i < \text{ilen } l.$   

$$(\text{inth } (\text{imap } ((-) (\text{ilen } \sigma)) (\text{irev } l)) (\text{Suc } i)) \leq \text{ilen } \sigma$$
  
 by (simp add: inth-imap)  
 have 17:  $\forall i < \text{ilen } l.$   
 irev  

$$(\text{sub } (\text{inth } (\text{imap } ((-) (\text{ilen } \sigma)) (\text{irev } l)) i)$$

$$(\text{inth } (\text{imap } ((-) (\text{ilen } \sigma)) (\text{irev } l)) (\text{Suc } i)) \sigma) =$$

$$\text{sub } (\text{ilen } \sigma - (\text{inth } (\text{imap } ((-) (\text{ilen } \sigma)) (\text{irev } l)) (\text{Suc } i)))$$

$$(\text{ilen } \sigma - (\text{inth } (\text{imap } ((-) (\text{ilen } \sigma)) (\text{irev } l)) i))$$

$$(\text{irev } \sigma)$$
  
 using irev-sub  
 using 15 16 by blast  
 have 18:  $\forall i < \text{ilen } l.$   

$$\text{ilen } \sigma - (\text{inth } (\text{imap } ((-) (\text{ilen } \sigma)) (\text{irev } l)) (\text{Suc } i)) =$$

$$(\text{inth } (\text{irev } l) (\text{Suc } i))$$
  
 by (metis 10 12 13 14 Suc-leI diff-diff-cancel idx-less-than  
 ilen-gr-zero inth-imap less-le zero-less-Suc zero-less-diff)  
 have 19:  $\forall i < \text{ilen } l.$   

$$\text{ilen } \sigma - (\text{inth } (\text{imap } ((-) (\text{ilen } \sigma)) (\text{irev } l)) i) =$$

$$(\text{inth } (\text{irev } l) i)$$
  
 by (metis 10 18 5 diff-zero ifirst-irev less-SucI less-Suc-eq-0-disj)  
 have 20:  $\forall i < \text{ilen } l.$   

$$(\text{sub } (\text{ilen } \sigma - (\text{inth } (\text{imap } ((-) (\text{ilen } \sigma)) (\text{irev } l)) (\text{Suc } i)))$$

$$(\text{ilen } \sigma - (\text{inth } (\text{imap } ((-) (\text{ilen } \sigma)) (\text{irev } l)) i))$$

$$(\text{irev } \sigma)) =$$

$$(\text{sub } (\text{inth } (\text{irev } l) (\text{Suc } i))$$

$$(\text{inth } (\text{irev } l) i))$$

$$(\text{irev } \sigma))$$
  
 using 18 19 by simp  
 have 21:  $\forall i < \text{ilen } l.$   

$$(\text{sub } (\text{inth } (\text{irev } l) (\text{Suc } i))$$

$$(\text{inth } (\text{irev } l) i))$$

$$(\text{irev } \sigma)) =$$

$$(\text{sub } (\text{inth } l (\text{ilen } l - (\text{Suc } i)))$$

$$(\text{inth } l (\text{ilen } l - i))$$

$$(\text{irev } \sigma))$$
  
 by (simp add: irev-inth)  
 have 22:  $\forall i < \text{ilen } l.$

$f ( \text{sub} (\text{inth } l (\text{ilen } l - (\text{Suc } i)))$   
 $(\text{inth } l (\text{ilen } l - i))$   
 $(\text{irev } \sigma))$

**by** (*metis 6 Suc-diff-Suc Suc-less-eq diff-less-Suc powerinterval-def*)  
**have 24:**  $\forall i < \text{ilen } l.$   
 $f (\text{irev}$   
 $(\text{sub} (\text{inth} (\text{imap} ((-) (\text{ilen } \sigma)) (\text{irev } l)) i)$   
 $(\text{inth} (\text{imap} ((-) (\text{ilen } \sigma)) (\text{irev } l)) (\text{Suc } i)) \sigma))$

**by** (*simp add: 17 20 21 22*)  
**have 25:**  $\text{powerinterval } (\lambda s. f (\text{irev } s)) \sigma (\text{imap} ((-) (\text{ilen } \sigma)) (\text{irev } l))$   
**by** (*simp add: 24 powerinterval-def*)  
**have 26:**  $(\exists l. \text{index-sequence } 0 l \wedge$   
 $\text{inth } l (\text{ilen } l) = \text{ilen } \sigma \wedge$   
 $\text{powerinterval } (\lambda s. f (\text{irev } s)) \sigma l \wedge g (\text{irev}(\text{filt } \sigma l)))$

**using 12 14 25 9 by blast**  
**from 26 show ?thesis**  
**by** (*simp add: projection-d-def reverse-d-def*)  
**qed**

**lemma ProjectionRevsemb:**

**assumes**  $(\sigma \models (f^r) \triangle (g^r))$

**shows**  $(\sigma \models (f \triangle g)^r)$

**proof** –

**have 1:**  $(\exists l. \text{index-sequence } 0 l \wedge$   
 $\text{inth } l (\text{ilen } l) = \text{ilen } \sigma \wedge$   
 $\text{powerinterval } (\lambda s. f (\text{irev } s)) \sigma l \wedge g (\text{irev}(\text{filt } \sigma l)))$

**using** *assms* **by** (*simp add: projection-d-def reverse-d-def*)

**obtain l where 2:**  $\text{index-sequence } 0 l \wedge$   
 $\text{inth } l (\text{ilen } l) = \text{ilen } \sigma \wedge$   
 $\text{powerinterval } (\lambda s. f (\text{irev } s)) \sigma l \wedge g (\text{irev}(\text{filt } \sigma l))$

**using 1 by auto**

**have 3:**  $\text{index-sequence } 0 l$

**using 2 by auto**

**have 4:**  $\text{inth } l (\text{ilen } l) = \text{ilen } \sigma$

**using 2 by auto**

**have 5:**  $\text{powerinterval } (\lambda s. f (\text{irev } s)) \sigma l$

**using 2 by auto**

**have 6:**  $g (\text{irev}(\text{filt } \sigma l))$

**using 2 by auto**

**have 7:**  $\text{ilen } (\text{imap} ((-) (\text{ilen } \sigma)) (\text{irev } l)) = \text{ilen } l$

**by** *simp*

**have 8:**  $\forall i \leq \text{ilen } l. (\text{inth} (\text{imap} ((-) (\text{ilen } \sigma)) (\text{irev } l)) i) \leq \text{ilen } \sigma$

**by** (*simp add: inth-imap*)

**have 9:**  $g (\text{filt } (\text{irev } \sigma) (\text{imap} ((-) (\text{ilen } \sigma)) (\text{irev } l)))$

**by** (*metis 2 filt-irev idx-less-equal irev-ilen irev-irev-ident*  
*le-refl*)

**have 10:**  $\text{inth} (\text{imap} ((-) (\text{ilen } \sigma)) (\text{irev } l)) (\text{ilen } (\text{imap} ((-) (\text{ilen } \sigma)) (\text{irev } l))) = \text{ilen } \sigma$

**by** (*metis 2 diff-zero index-sequence-def ilast-irev ilen-imap*)

$\text{inth-imap}$  )  
**have 11:**  $\text{inth} (\text{imap} ((-) (\text{ilen } \sigma)) (\text{irev } l)) 0 = 0$   
**by** ( $\text{metis 4 diff-self-eq-0 ifirst-irev inth-imap}$  )  
**have 12:**  $(\forall n < \text{ilen } l.$   
 $\quad \text{inth} (\text{imap} ((-) (\text{ilen } \sigma)) (\text{irev } l)) n$   
 $\quad < \text{inth} (\text{imap} ((-) (\text{ilen } \sigma)) (\text{irev } l)) (\text{Suc } n))$   
**by** ( $\text{simp add: inth-imap irev-inth}$ )  
 $(\text{metis (no-types, lifting) 2 Suc-diff-Suc Suc-less-eq diff-less-Suc diff-less-mono2}$   
 $\text{index-sequence-def idx-less-last-1})$   
**have 13:**  $\text{index-sequence } 0 (\text{imap} (\lambda x. \text{ilen } \sigma - x) (\text{irev } l))$   
**by** ( $\text{simp add: 11 12 index-sequence-def}$ )  
**have 14:**  $\forall i < \text{ilen } l. f (\text{irev} (\text{sub} (\text{inth } l \ i) (\text{inth } l (\text{Suc } i)) \sigma))$   
**using 5 by** ( $\text{simp add: powerinterval-def}$ )  
**have 15:**  $\forall i < \text{ilen } l. (\text{inth } l (\text{Suc } i)) \leq \text{ilen } \sigma$   
**using 2 idx-expand by fastforce**  
**have 16:**  $\forall i < \text{ilen } l. (\text{inth } l \ i) \leq (\text{inth } l (\text{Suc } i))$   
**using 2 idx-expand by fastforce**  
**have 17:**  $\forall i < \text{ilen } l.$   
 $\quad f (\text{sub} ((\text{ilen } \sigma) - (\text{inth } l (\text{Suc } i))) ((\text{ilen } \sigma) - (\text{inth } l \ i)) (\text{irev } \sigma))$   
  
**using 14**  
**by** ( $\text{simp add: irev-sub 15 16}$ )  
**have 18:**  $\forall i < \text{ilen } l.$   
 $\quad (\text{inth} (\text{imap} ((-) (\text{ilen } \sigma)) (\text{irev } l)) \ i) =$   
 $\quad \text{ilen } \sigma - (\text{inth} (\text{irev } l) \ i)$   
  
**using inth-imap by blast**  
**have 19:**  $\forall i < \text{ilen } l.$   
 $\quad (\text{inth} (\text{imap} ((-) (\text{ilen } \sigma)) (\text{irev } l)) (\text{Suc } i)) =$   
 $\quad \text{ilen } \sigma - (\text{inth} (\text{irev } l) (\text{Suc } i))$   
  
**using inth-imap by blast**  
**have 20:**  $\forall i < \text{ilen } l.$   
 $\quad (\text{inth} (\text{irev } l) \ i) = (\text{inth } l (\text{ilen } l - i))$   
  
**by** ( $\text{simp add: irev-inth}$ )  
**have 21:**  $\forall i < \text{ilen } l.$   
 $\quad (\text{inth} (\text{irev } l) (\text{Suc } i)) = (\text{inth } l (\text{ilen } l - (\text{Suc } i)))$   
**by** ( $\text{simp add: irev-inth}$ )  
**have 22:**  $\forall i < \text{ilen } l.$   
 $\quad f (\text{sub} (\text{ilen } \sigma - (\text{inth } l (\text{ilen } l - i)))$   
 $\quad (\text{ilen } \sigma - (\text{inth } l (\text{ilen } l - (\text{Suc } i))))$   
 $\quad (\text{irev } \sigma))$   
  
**by** ( $\text{metis 17 Suc-diff-Suc Suc-less-eq diff-less-Suc}$ )  
**have 23:**  $\forall i < \text{ilen } l.$   
 $\quad f (\text{sub} (\text{ilen } \sigma - (\text{inth} (\text{irev } l) \ i))$   
 $\quad (\text{ilen } \sigma - (\text{inth} (\text{irev } l) (\text{Suc } i)))$   
 $\quad (\text{irev } \sigma))$



```

  by (simp add: 20 21 22)
have 24:  $\forall i < \text{ilen } l.$ 
   $f (\text{sub } (\text{inth } (\text{imap } ((-) (\text{ilen } \sigma)) (\text{irev } l)) i)$ 
     $(\text{inth } (\text{imap } ((-) (\text{ilen } \sigma)) (\text{irev } l)) (\text{Suc } i)) (\text{irev } \sigma))$ 
  by (simp add: 23 inth-imap)
have 25:  $\text{powerinterval } f (\text{irev } \sigma) (\text{imap } ((-) (\text{ilen } \sigma)) (\text{irev } l))$ 
  by (simp add: 24 powerinterval-def)
have 26:  $\exists l. \text{index-sequence } 0 l \wedge \text{inth } l (\text{ilen } l) = \text{ilen } \sigma \wedge$ 
   $\text{powerinterval } f (\text{irev } \sigma) l \wedge g (\text{filt } (\text{irev } \sigma) l)$ 
  using 10 13 25 9 by blast
from 26 show ?thesis by (simp add: projection-d-def reverse-d-def)
qed

```

**lemma** *ProjectionRev*:  
 $\vdash (f \triangle g)^r = f^r \triangle g^r$   
**using** *ProjectionRevsema ProjectionRevsemb unl-lift2* **by** *blast*

## 8.6 Theorems

### 8.6.1 Projection

**lemma** *PowerProjLen*:  
 $\vdash f \triangle \text{len } n = \text{power } (f \wedge \text{more}) n$   
**proof**  
 (*induct*  $n$ )  
**case** 0  
**then show** ?case **by** (*metis PJ2 len-d-def pow-0*)  
**next**  
**case** (*Suc*  $n$ )  
**then show** ?case  
**by** (*metis PJ3 PJ4 inteq-reflection len-d-def pow-Suc*)  
**qed**

**lemma** *ProjLenExist*:  
 $\vdash f \triangle (\exists n. \text{len } n) = (\exists n. f \triangle \text{len } n)$   
**by** (*simp add: Valid-def projection-d-def, blast*)

**lemma** *PowerProjLenExist*:  
 $\vdash (\exists n. f \triangle \text{len } n) = (\exists n. \text{power } (f \wedge \text{more}) n)$   
**using** *PowerProjLen* **by** (*simp add: Valid-def PowerProjLen, blast*)

**lemma** *RightProjImpProj*:  
**assumes**  $\vdash g1 \longrightarrow g2$   
**shows**  $\vdash f \triangle g1 \longrightarrow f \triangle g2$   
**using** *assms*  
**by** (*simp add: Valid-def projection-d-def, blast*)

**lemma** *LeftProjImpProj*:

**assumes**  $\vdash f1 \longrightarrow f2$   
**shows**  $\vdash f1 \triangle g \longrightarrow f2 \triangle g$   
**using** *assms*  
**by** (*simp add: Valid-def projection-d-def powerinterval-def, blast*)

**lemma** *RightProjEqvProj*:  
**assumes**  $\vdash g1 = g2$   
**shows**  $\vdash f \triangle g1 = f \triangle g2$   
**using** *assms*  
**by** (*metis (mono-tags, lifting) Valid-def inteq-reflection unl-lift2*)

**lemma** *LeftProjEqvProj*:  
**assumes**  $\vdash f1 = f2$   
**shows**  $\vdash f1 \triangle g = f2 \triangle g$   
**using** *assms*  
**by** (*metis (mono-tags, lifting) Valid-def inteq-reflection unl-lift2*)

**lemma** *ProjTrueEqvChopstar*:  
 $\vdash f \triangle \#True = f^*$   
**proof** –  
**have** 1:  $\vdash \#True = (\exists n. \text{len } n)$   
**by** (*simp add: Valid-def len-defs*)  
**have** 2:  $\vdash f \triangle \#True = f \triangle (\exists n. \text{len } n)$   
**using** 1 *RightProjEqvProj* **by** *blast*  
**have** 3:  $\vdash f \triangle (\exists n. \text{len } n) = (\exists n. \text{power } (f \wedge \text{more}) n)$   
**using** *PowerProjLenExist ProjLenExist* **by** *fastforce*  
**have** 4:  $\vdash (\exists n. \text{power } (f \wedge \text{more}) n) = f^*$   
**by** (*simp add: chopstar-d-def powerstar-d-def*)  
**show** ?thesis **using** 2 3 4 **by** *fastforce*  
**qed**

**lemma** *ProjChopstarEqvChopstarProj*:  
 $\vdash f \triangle (g^*) = (f \triangle g)^*$   
**proof** –  
**have** 1:  $\vdash f \triangle (g^*) = f \triangle (g \triangle \#True)$   
**by** (*metis ProjTrueEqvChopstar RightProjEqvProj inteq-reflection*)  
**have** 2:  $\vdash f \triangle (g \triangle \#True) = (f \triangle g) \triangle \#True$   
**by** (*simp add: PJ7*)  
**have** 3:  $\vdash (f \triangle g) \triangle \#True = (f \triangle g)^*$   
**by** (*simp add: ProjTrueEqvChopstar*)  
**show** ?thesis **using** 1 2 3 **by** *fastforce*  
**qed**

**lemma** *ProjAndImp*:  
 $\vdash f \triangle (g1 \wedge g2) \longrightarrow f \triangle g1 \wedge f \triangle g2$   
**by** (*meson Prop12 RightProjImpProj int-iffD1 lift-and-com*)

**lemma** *ProjOrDist*:

$\vdash \#True \triangle (f \vee g) = (\#True \triangle f \vee \#True \triangle g)$   
**using** *PJ1sema* **by** *blast*

**lemma** *StateImportProj*:

$\vdash ((init\ w) \wedge f \triangle g) = f \triangle ((init\ w) \wedge g)$   
**by** (*auto simp add: Valid-def init-defs projection-d-def filt-inth index-sequence-def*)

**lemma** *ProjStateAndNextEqvStateAndMoreChopProj*:

$\vdash f \triangle ((init\ w) \wedge \bigcirc g) = ((init\ w) \wedge (f \wedge more));(f \triangle g)$   
**proof** –  
**have** 2:  $\vdash (f \wedge more);(f \triangle g) = f \triangle \bigcirc g$   
**by** (*metis PJ3 PJ4 inteq-reflection next-d-def*)  
**have** 3:  $\vdash f \triangle ((init\ w)) \longrightarrow init\ w$   
**by** (*simp add: PJ5*)  
**have** 4:  $\vdash (init\ w \wedge f \triangle \bigcirc g) = f \triangle ((init\ w) \wedge \bigcirc g)$   
**by** (*simp add: StateImportProj*)  
**have** 5:  $\vdash f \triangle ((init\ w) \wedge \bigcirc g) \longrightarrow ((init\ w) \wedge (f \wedge more));(f \triangle g)$   
**using** 2 3 *ProjAndImp* **by** *fastforce*  
**from** 5 4 **show** *?thesis* **using** 2 **by** *fastforce*  
**qed**

**lemma** *ProjNext*:

$\vdash f \triangle \bigcirc g = (f \wedge more);(f \triangle g)$   
**by** (*metis PJ3 PJ4 inteq-reflection next-d-def*)

**lemma** *ProjWnext*:

$\vdash f \triangle (wnext\ g) = (empty \vee (f \wedge more));(f \triangle g)$   
**proof** –  
**have** 1:  $\vdash f \triangle (wnext\ g) = f \triangle (empty \vee \bigcirc g)$   
**by** (*simp add: RightProjEqvProj WnextEqvEmptyOrNext*)  
**have** 2:  $\vdash f \triangle (empty \vee \bigcirc g) = (empty \vee f \triangle (\bigcirc g))$   
**using** *PJ1sema PJ2* **by** *fastforce*  
**have** 3:  $\vdash f \triangle (\bigcirc g) = (f \wedge more);(f \triangle g)$   
**by** (*metis PJ3 PJ4 inteq-reflection next-d-def*)  
**show** *?thesis*  
**using** 1 2 3 **by** *fastforce*  
**qed**

**lemma** *ProjIntro*:

**assumes**  $\vdash f \wedge more \longrightarrow (g \wedge more); f$   
**shows**  $\vdash f \longrightarrow g \triangle \#True$   
**using** *assms CSIntro ProjTrueEqvChopstar* **by** *force*

**lemma** *RightBoxStateImportProj*:

$\vdash \Box((init\ w) \wedge f \triangle g) \longrightarrow f \triangle (\Box((init\ w) \wedge g))$   
**by** (*simp add: Valid-def always-defs init-defs projection-d-def*)  
*(metis filt-ilen filt-inth idx-bound-1 suffix-zero)*

**lemma** *LeftBoxStateImportProjhelp*:

$$\begin{aligned}
& (\forall n \leq \text{ilen } wa. w \langle \text{inth } wa \ n \rangle) \wedge \\
& (\exists l. \text{inth } l \ 0 = 0 \wedge \\
& \quad (\forall n < \text{ilen } l. \text{inth } l \ n < \text{inth } l \ (\text{Suc } n)) \wedge \\
& \quad \text{inth } l \ (\text{ilen } l) = \text{ilen } wa \wedge \\
& \quad (\forall i < \text{ilen } l. f \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ wa)) \wedge \\
& \quad g \ (\text{filt } wa \ l)) \longrightarrow \\
& (\exists l. \text{inth } l \ 0 = 0 \wedge \\
& \quad (\forall n < \text{ilen } l. \text{inth } l \ n < \text{inth } l \ (\text{Suc } n)) \wedge \\
& \quad \text{inth } l \ (\text{ilen } l) = \text{ilen } wa \wedge \\
& \quad (\forall i < \text{ilen } l. \\
& \quad \quad f \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ wa) \wedge \\
& \quad \quad (\forall n \leq \text{ilen } (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ wa). \\
& \quad \quad \quad w \langle \text{inth } (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ wa) \ n \rangle)) \wedge \\
& \quad g \ (\text{filt } wa \ l))
\end{aligned}$$

**proof**

**assume** *0*:  $(\forall n \leq \text{ilen } wa. w \langle \text{inth } wa \ n \rangle) \wedge$   
 $(\exists l. \text{inth } l \ 0 = 0 \wedge$   
 $(\forall n < \text{ilen } l. \text{inth } l \ n < \text{inth } l \ (\text{Suc } n)) \wedge$   
 $\text{inth } l \ (\text{ilen } l) = \text{ilen } wa \wedge$   
 $(\forall i < \text{ilen } l. f \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ wa)) \wedge$   
 $g \ (\text{filt } wa \ l))$

**show**  $\exists l. \text{inth } l \ 0 = 0 \wedge$   
 $(\forall n < \text{ilen } l. \text{inth } l \ n < \text{inth } l \ (\text{Suc } n)) \wedge$   
 $\text{inth } l \ (\text{ilen } l) = \text{ilen } wa \wedge$   
 $(\forall i < \text{ilen } l.$   
 $\quad f \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ wa) \wedge$   
 $\quad (\forall n \leq \text{ilen } (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ wa).$   
 $\quad \quad w \langle \text{inth } (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ wa) \ n \rangle)) \wedge$   
 $g \ (\text{filt } wa \ l)$

**proof** –

**have** *1*:  $(\forall n \leq \text{ilen } wa. w \langle \text{inth } wa \ n \rangle)$

**using** *0* **by** *auto*

**have** *2*:  $(\exists l. \text{inth } l \ 0 = 0 \wedge$   
 $(\forall n < \text{ilen } l. \text{inth } l \ n < \text{inth } l \ (\text{Suc } n)) \wedge$   
 $\text{inth } l \ (\text{ilen } l) = \text{ilen } wa \wedge$   
 $(\forall i < \text{ilen } l. f \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ wa)) \wedge$   
 $g \ (\text{filt } wa \ l))$

**using** *0* **by** *auto*

**obtain** *l* **where** *3*:  $\text{inth } l \ 0 = 0 \wedge$   
 $(\forall n < \text{ilen } l. \text{inth } l \ n < \text{inth } l \ (\text{Suc } n)) \wedge$   
 $\text{inth } l \ (\text{ilen } l) = \text{ilen } wa \wedge$   
 $(\forall i < \text{ilen } l. f \ (\text{sub } (\text{inth } l \ i) \ (\text{inth } l \ (\text{Suc } i)) \ wa)) \wedge$   
 $g \ (\text{filt } wa \ l)$

**using** *2* **by** *auto*

**have** *4*:  $\text{inth } l \ 0 = 0$

**using** *3* **by** *auto*

**have** *5*:  $(\forall n < \text{ilen } l. \text{inth } l \ n < \text{inth } l \ (\text{Suc } n))$

**using** *3* **by** *auto*

```

have 6:  $\text{inth } l (\text{ilen } l) = \text{ilen } wa$ 
  using 3 by auto
have 7:  $(\forall i < \text{ilen } l. f (\text{sub } ( \text{inth } l i) ( \text{inth } l (\text{Suc } i)) wa))$ 
  using 3 by auto
have 8:  $g (\text{filt } wa l)$ 
  using 3 by auto
have 9:  $(\forall i < \text{ilen } l.$ 
   $f (\text{sub } ( \text{inth } l i) ( \text{inth } l (\text{Suc } i)) wa) \wedge$ 
   $(\forall n \leq (\text{inth } l (\text{Suc } i)) - (\text{inth } l i).$ 
   $w \langle \text{inth } wa ((\text{inth } l i) + n) \rangle))$ 
by (metis 1 7 inth-last-stutter nat-le-iff-add nat-le-linear)
have 10:  $(\forall i < \text{ilen } l.$ 
   $\text{ilen } (\text{sub } ( \text{inth } l i) ( \text{inth } l (\text{Suc } i)) wa) =$ 
   $(\text{inth } l (\text{Suc } i)) - (\text{inth } l i) )$ 
  by (simp add: 3 PJ6help1 index-sequence-def)
have 11:  $(\forall i < \text{ilen } l.$ 
   $(\forall n \leq (\text{inth } l (\text{Suc } i)) - (\text{inth } l i).$ 
   $\text{inth } (\text{sub } ( \text{inth } l i) ( \text{inth } l (\text{Suc } i)) wa) n =$ 
   $\text{inth } wa ((\text{inth } l i) + n) )$ 
  using 3 index-sequence-def idx-expand by fastforce
have 12:  $(\forall i < \text{ilen } l.$ 
   $f (\text{sub } ( \text{inth } l i) ( \text{inth } l (\text{Suc } i)) wa) \wedge$ 
   $(\forall n \leq \text{ilen } (\text{sub } ( \text{inth } l i) ( \text{inth } l (\text{Suc } i)) wa).$ 
   $w \langle \text{inth } (\text{sub } ( \text{inth } l i) ( \text{inth } l (\text{Suc } i)) wa) n \rangle))$ 
  using 9 10 11 by simp
show ?thesis
using 12 3 by blast
qed
qed

```

**lemma** *LeftBoxStateImportProj:*

$\vdash \square(\text{init } w) \wedge f \triangle g \longrightarrow (f \wedge \square(\text{init } w)) \triangle g$

**using** *LeftBoxStateImportProjhelp*

**by** (simp add: index-sequence-def Valid-def always-defs init-defs projection-d-def powerinterval-def)  
blast

### 8.6.2 dp and bp

**lemma** *NotDpEqvBpNot:*

$\vdash (\neg(dp f)) = bp (\neg f)$

**by** (simp add: bp-d-def dp-d-def uprojection-d-def)

**lemma** *NotBpEqvDpNot:*

$\vdash (\neg(bp f)) = dp(\neg f)$

**by** (simp add: bp-d-def dp-d-def uprojection-d-def)

**lemma** *NowImpDp:*

$\vdash f \longrightarrow dp f$

**proof** –  
**have** 1:  $\vdash (\text{skip} \longrightarrow \# \text{True})$   
**by** *simp*  
**have** 2:  $\vdash \text{ba}(\text{skip} \longrightarrow \# \text{True})$   
**using** 1 **by** (*simp add: BaGen*)  
**have** 3:  $\vdash \text{ba}(\text{skip} \longrightarrow \# \text{True}) \longrightarrow (\text{skip} \triangle f \longrightarrow \# \text{True} \triangle f)$   
**using** PJ8 **by** *blast*  
**have** 4:  $\vdash (\text{skip} \triangle f \longrightarrow \# \text{True} \triangle f)$   
**using** 2 3 MP **by** *blast*  
**show** ?thesis  
**by** (*metis 4 PJ6 dp-d-def inteq-reflection*)  
**qed**

**lemma** *BpElim*:  
 $\vdash \text{bp } f \longrightarrow f$   
**proof** –  
**have** 1:  $\vdash \neg f \longrightarrow \text{dp } (\neg f)$   
**by** (*simp add: NowImpDp*)  
**hence** 2:  $\vdash \neg(\text{dp } (\neg f)) \longrightarrow f$   
**by** *auto*  
**from** 2 **show** ?thesis  
**by** (*simp add: bp-d-def dp-d-def uprojection-d-def*)  
**qed**

**lemma** *BpImpDpImpDp*:  
 $\vdash \text{bp } (f \longrightarrow g) \longrightarrow \text{dp } f \longrightarrow \text{dp } g$   
**proof** –  
**have** 1:  $\vdash \text{bp } (f \longrightarrow g) \longrightarrow (\# \text{True} \triangle f) \longrightarrow (\# \text{True} \triangle g)$   
**by** (*simp add: PJ9 bp-d-def*)  
**from** 1 **show** ?thesis **by** (*simp add: dp-d-def*)  
**qed**

**lemma** *BpContraPosImpDist*:  
 $\vdash \text{bp } (\neg g \longrightarrow \neg f) \longrightarrow (\text{bp } f) \longrightarrow (\text{bp } g)$   
**proof** –  
**have** 1:  $\vdash \text{bp } (\neg g \longrightarrow \neg f) \longrightarrow (\text{dp } (\neg g)) \longrightarrow (\text{dp } (\neg f))$   
**by** (*rule BpImpDpImpDp*)  
**hence** 2:  $\vdash \text{bp } (\neg g \longrightarrow \neg f) \longrightarrow (\neg (\text{dp } (\neg f))) \longrightarrow (\neg (\text{dp } (\neg g)))$  **by** *auto*  
**from** 2 **show** ?thesis  
**by** (*simp add: bp-d-def dp-d-def uprojection-d-def*)  
**qed**

**lemma** *BpImpDist*:  
 $\vdash \text{bp } (f \longrightarrow g) \longrightarrow (\text{bp } f) \longrightarrow (\text{bp } g)$   
**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:  $\vdash bp (\neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g))$  **by** (rule BpGen)  
**have** 4:  $\vdash bp (\neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g))$   
 $\longrightarrow$   
 $bp (f \longrightarrow g) \longrightarrow bp (\neg g \longrightarrow \neg f)$  **by** (rule BpContraPosImpDist)  
**have** 5:  $\vdash bp (f \longrightarrow g) \longrightarrow bp (\neg g \longrightarrow \neg f)$  **using** 3 4 MP **by** blast  
**have** 6:  $\vdash bp (\neg g \longrightarrow \neg f) \longrightarrow (bp f) \longrightarrow (bp g)$  **by** (rule BpContraPosImpDist)  
**from** 5 6 **show** ?thesis **using** lift-imp-trans **by** blast  
**qed**

**lemma** DpImpDpRule:  
**assumes**  $\vdash f \longrightarrow g$   
**shows**  $\vdash dp f \longrightarrow dp g$   
**proof** –  
**have** 1:  $\vdash f \longrightarrow g$  **using** assms **by** auto  
**hence** 2:  $\vdash \#True \triangle f \longrightarrow \#True \triangle g$   
**by** (metis BpGen MP PJ9 bp-d-def)  
**from** 2 **show** ?thesis **by** (simp add: dp-d-def)  
**qed**

**lemma** BpImpBpRule:  
**assumes**  $\vdash f \longrightarrow g$   
**shows**  $\vdash bp f \longrightarrow bp g$   
**proof** –  
**have** 1:  $\vdash f \longrightarrow g$  **using** assms **by** auto  
**hence** 2:  $\vdash \neg g \longrightarrow \neg f$  **by** auto  
**hence** 3:  $\vdash dp (\neg g) \longrightarrow dp (\neg f)$  **by** (rule DpImpDpRule)  
**hence** 4:  $\vdash \neg (dp (\neg f)) \longrightarrow \neg (dp (\neg g))$  **by** auto  
**from** 4 **show** ?thesis  
**by** (meson BpGen BpImpDist MP assms)  
**qed**

**lemma** DpEqvDpRule:  
**assumes**  $\vdash f = g$   
**shows**  $\vdash dp f = dp g$   
**proof** –  
**have** 1:  $\vdash f = g$  **using** assms **by** auto  
**hence** 2:  $\vdash \#True \triangle f = \#True \triangle g$   
**using** RightProjEqvProj **by** blast  
**from** 2 **show** ?thesis **by** (simp add: dp-d-def)  
**qed**

**lemma** BpEqvBpRule:  
**assumes**  $\vdash f = g$   
**shows**  $\vdash bp f = bp g$   
**proof** –  
**have** 1:  $\vdash f = g$  **using** assms **by** auto  
**hence** 2:  $\vdash (\neg f) = (\neg g)$  **by** auto  
**hence** 3:  $\vdash dp (\neg f) = dp (\neg g)$  **by** (rule DpEqvDpRule)  
**hence** 4:  $\vdash \neg (dp (\neg f)) = \neg (dp (\neg g))$  **by** auto  
**from** 4 **show** ?thesis

by (metis BpImpBpRule assms int-iffD1 int-iffI inteq-reflection)  
qed

**lemma** DpState:

$\vdash dp \ (init \ w) = (init \ w)$

by (metis NowImpDp PJ5 dp-d-def int-iffI)

**lemma** StateEqvBp:

$\vdash \ (init \ w) = bp \ (init \ w)$

**proof** –

have 1:  $\vdash (init \ w) \longrightarrow bp \ (init \ w)$

by (metis (no-types, lifting) DiState DpState Initprop(2) StateEqvBi bi-d-def bp-d-def dp-d-def int-iffD1 inteq-reflection uprojection-d-def)

have 2:  $\vdash bp \ (init \ w) \longrightarrow (init \ w)$  by (rule BpElim)

from 1 2 show ?thesis by fastforce

qed

**lemma** DpDpEqvDp:

$\vdash dp \ (dp \ f) = dp \ f$

**proof** –

have 2:  $\vdash \#True \triangle (\#True \triangle f) = (\#True \triangle \#True) \triangle f$

by (simp add: PJ7)

have 3:  $\vdash (\#True \triangle \#True) = \#True$

by (metis DpState Initprop(4) dp-d-def int-eq-true inteq-reflection)

show ?thesis by (metis 2 3 dp-d-def inteq-reflection)

qed

**lemma** BpBpEqvBp:

$\vdash bp \ (bp \ f) = bp \ f$

**proof** –

have 1:  $\vdash dp \ (dp \ (\neg \ f)) = dp \ (\neg \ f)$

using DpDpEqvDp by blast

have 2:  $\vdash (\neg \ (dp \ (dp \ (\neg \ f)))) = (\neg \ (dp \ (\neg \ f)))$

using 1 by auto

have 3:  $\vdash (\neg \ (dp \ (\neg \ f))) = bp \ f$

by (simp add: bp-d-def dp-d-def uprojection-d-def)

have 4:  $\vdash (\neg \ (dp \ (dp \ (\neg \ f)))) = bp \ (bp \ f)$

by (simp add: bp-d-def dp-d-def uprojection-d-def)

from 2 3 4 show ?thesis

by fastforce

qed

**lemma** DpOrEqv:

$\vdash \ dp \ (f \vee \ g) = (dp \ f \vee \ dp \ g)$

**proof** –

have 1:  $\vdash \#True \triangle (f \vee \ g) = (\#True \triangle f \vee \ \#True \triangle g)$



using *ProjOrDist* by *auto*  
 from 1 show ?thesis by (simp add: *dp-d-def*)  
 qed

lemma *BpAndEqv*:

$\vdash bp(f \wedge g) = (bp\ f \wedge bp\ g)$

proof –

have 1:  $\vdash dp((\neg f) \vee (\neg g)) = (dp\ (\neg f) \vee dp\ (\neg g))$

using *DpOrEqv* by *auto*

hence 2:  $\vdash (\neg(dp((\neg f) \vee (\neg g)))) = (\neg(dp\ (\neg f) \vee dp\ (\neg g)))$

by *auto*

have 3:  $\vdash (\neg(dp((\neg f) \vee (\neg g)))) = bp\ (\neg((\neg f) \vee (\neg g)))$

using *NotDpEqvBpNot* by *blast*

have 4:  $\vdash (\neg((\neg f) \vee (\neg g))) = (f \wedge g)$

by *auto*

hence 5:  $\vdash bp(\neg((\neg f) \vee (\neg g))) = bp(f \wedge g)$

by (simp add: *BpEqvBpRule*)

have 6:  $\vdash (\neg(dp\ (\neg f) \vee dp\ (\neg g))) = ((\neg(dp\ (\neg f))) \wedge (\neg(dp\ (\neg g))))$

by *auto*

have 7:  $\vdash ((\neg(dp\ (\neg f))) \wedge (\neg(dp\ (\neg g)))) = (bp\ f \wedge bp\ g)$

by (simp add: *bp-d-def dp-d-def uprojection-d-def*)

show ?thesis

by (metis 2 3 4 6 7 *inteq-reflection*)

qed

lemma *DpAndA*:

$\vdash dp(f \wedge g) \longrightarrow dp\ f$

proof –

have 1:  $\vdash \#True \triangle (f \wedge g) \longrightarrow \#True \triangle f$

by (meson *Prop12 RightProjImpProj int-iffD1 lift-and-com*)

from 1 show ?thesis by (simp add: *dp-d-def*)

qed

lemma *BpOrA*:

$\vdash bp\ f \longrightarrow bp(f \vee g)$

by (simp add: *BpImpBpRule intI*)

lemma *BpOrB*:

$\vdash bp\ g \longrightarrow bp(f \vee g)$

by (simp add: *BpImpBpRule intI*)

lemma *BpOrImpOr*:

$\vdash bp\ f \vee bp\ g \longrightarrow bp(f \vee g)$

using *BpOrA BpOrB* by *fastforce*

lemma *DpAndB*:

$\vdash dp(f \wedge g) \longrightarrow dp\ g$

proof –

have 1:  $\vdash \#True \triangle (f \wedge g) \longrightarrow \#True \triangle g$

by (meson Prop12 RightProjImpProj int-iffD2 lift-and-com)  
 from 1 show ?thesis by (simp add: dp-d-def)  
 qed

**lemma** *DpAndImpAnd*:

$\vdash dp (f \wedge g) \longrightarrow dp f \wedge dp g$

**proof** –

have 1:  $\vdash dp (f \wedge g) \longrightarrow dp f$  by (rule DpAndA)

have 2:  $\vdash dp (f \wedge g) \longrightarrow dp g$  by (rule DpAndB)

from 1 2 show ?thesis by fastforce

qed

**lemma** *DpSkipEqvMore*:

$\vdash dp skip = more$

**proof** –

have 1:  $\vdash dp skip = \#True \triangle skip$

by (simp add: dp-d-def)

have 2:  $\vdash \#True \triangle skip = (\#True \wedge more)$

using PJ3 by blast

have 3:  $\vdash (\#True \wedge more) = more$

by auto

from 1 2 3 show ?thesis by fastforce

qed

**lemma** *DpMoreEqvMore*:

$\vdash dp more = more$

by (metis DpDpEqvDp DpSkipEqvMore inteq-reflection)

**lemma** *BpEmptyEqvEmpty*:

$\vdash bp empty = empty$

by (metis DpMoreEqvMore NotDpEqvBpNot empty-d-def inteq-reflection)

**lemma** *DpEmptyEqvEmpty*:

$\vdash dp empty = empty$

**proof** –

have 1:  $\vdash dp empty = \#True \triangle empty$

by (simp add: dp-d-def)

have 2:  $\vdash \#True \triangle empty = empty$

by (simp add: PJ2)

from 1 2 show ?thesis by fastforce

qed

**lemma** *BpMoreEqvMore*:

$\vdash bp more = more$

by (metis DpEmptyEqvEmpty NotDpEqvBpNot NotEmptyEqvMore inteq-reflection)

**lemma** *NextDpImpDpNext*:

```

  ⊢ ○ (dp f) → dp (○ f)
proof -
  have 1: ⊢ dp(○ f) = #True △ (skip;f)
    by (simp add: dp-d-def next-d-def)
  have 2: ⊢ #True △ (skip;f) = (#True △ skip);(#True △ f)
    by (simp add: PJ4)
  have 3: ⊢ (#True △ skip) = (#True ∧ more)
    using PJ3 by blast
  have 4: ⊢ (#True ∧ more) = more
    by auto
  have 5: ⊢ skip;(#True △ f) → more;(#True △ f)
    by (metis DpSkipEqvMore LeftChopImpChop NowImpDp inteq-reflection)
  show ?thesis
    by (metis 2 3 4 5 dp-d-def inteq-reflection next-d-def)
qed

lemma BoxStateImportBp:
  ⊢ □(init w) → bp(□(init w))
by (simp add: Valid-def always-defs init-defs projection-d-def bp-d-def uprojection-d-def
    powerinterval-def)
  (metis (mono-tags, lifting) filt-expand idx-bound-1 suffix-zero)

lemma BoxStateEqvBpBoxState:
  ⊢ □(init w) = bp(□(init w))
proof -
  have 1: ⊢ bp(□(init w)) → □(init w)
    by (simp add: BpElim)
  have 2: ⊢ bp(□(init w)) = (¬(#True △ (¬ □(init w))))
    by (simp add: bp-d-def uprojection-d-def)
  have 2: ⊢ □(init w) → dp(□(init w))
    by (metis NowImpDp)
  have 2: ⊢ □(init w) → bp(□(init w))
    using BoxStateImportBp by auto
  from 1 2 show ?thesis by fastforce
qed

end

```

## 9 The First Occurrence Operator in finite ITL

```

theory First
imports
  Theorems TimeReversal
begin

```

Runtime verification (RV) has gained significant interest in recent years. The behaviour of a program can be verified in real time by analysing its evolving trace. This approach has two significant benefits over static verification techniques such as model checking. Firstly, it is only necessary to verify actual

execution paths rather than all possible paths. Secondly, it is possible to react at runtime should the program diverge from its specified behaviour. RV does not replace traditional verification techniques but it does provide an extra layer of security.

Linear Temporal Logic (LTL) is a popular formalism for writing specifications from which RV monitors can be derived automatically. By contrast, Interval Temporal Logic (ITL) has not been as widely represented in this field despite being more expressive and compositional. The principal issue is efficiency. ITL uses non-deterministic operators to IConstruct sequential and iterative specifications (chop and chop-star, respectively) and these introduce combinatorial complexity. Approaches to mitigate this include using a deterministic subset of ITL or adapting the semantics to include a deterministic chop operator. This work proposes an alternative approach, wholly within existing ITL, and based upon a new, derived operator called “first occurrence”.

A theory of first occurrence is developed and used to derive an algebra of RV monitors.

## 9.1 Definitions

### 9.1.1 Definitions Strict Initial and Final

**definition**  $bs-d :: ('a::world) formula \Rightarrow 'a formula$   
**where**

$$bs-d f \equiv LIFT(empty \vee ((bi f) ; skip))$$

**definition**  $bt-d :: ('a::world) formula \Rightarrow 'a formula$   
**where**

$$bt-d f \equiv LIFT(empty \vee (skip;(\Box f)))$$

**syntax**

$$-bs-d :: lift \Rightarrow lift ((bs -) [88] 87)$$

$$-bt-d :: lift \Rightarrow lift ((bt -) [88] 87)$$

**syntax** (*ASCII*)

$$-bs-d :: lift \Rightarrow lift ((bs -) [88] 87)$$

$$-bt-d :: lift \Rightarrow lift ((bt -) [88] 87)$$

**translations**

$$-bs-d \rightleftharpoons CONST bs-d$$

$$-bt-d \rightleftharpoons CONST bt-d$$

**definition**  $ds-d :: ('a::world) formula \Rightarrow 'a formula$   
**where**

$$ds-d f \equiv LIFT(\neg (bs (\neg f)))$$

**definition**  $dt-d :: ('a::world) formula \Rightarrow 'a formula$   
**where**

$$dt-d f \equiv LIFT(\neg (bt (\neg f)))$$

**syntax**

$$-ds-d :: lift \Rightarrow lift ((ds -) [88] 87)$$

$$-dt-d :: lift \Rightarrow lift ((dt -) [88] 87)$$

**syntax** (*ASCII*)

$-ds-d :: lift \Rightarrow lift ((ds \ -) [88] \ 87)$

$-dt-d :: lift \Rightarrow lift ((dt \ -) [88] \ 87)$

**translations**

$-ds-d \Rightarrow CONST \ ds-d$

$-dt-d \Rightarrow CONST \ dt-d$

### 9.1.2 Definition First and Last Operators

**definition**  $first-d :: ('a::world) \ formula \Rightarrow 'a \ formula$

**where**

$first-d \ f \equiv LIFT \ (f \wedge \ (bs \ (\neg \ f)))$

**definition**  $last-d :: ('a::world) \ formula \Rightarrow 'a \ formula$

**where**

$last-d \ f \equiv LIFT \ (f \wedge \ (bt \ (\neg \ f)))$

**syntax**

$-first-d :: lift \Rightarrow lift ((\triangleright \ -) [88] \ 87)$

$-last-d :: lift \Rightarrow lift ((\triangleleft \ -) [88] \ 87)$

**syntax** (*ASCII*)

$-first-d :: lift \Rightarrow lift ((first \ -) [88] \ 87)$

$-last-d :: lift \Rightarrow lift ((last \ -) [88] \ 87)$

**translations**

$-first-d \Rightarrow CONST \ first-d$

$-last-d \Rightarrow CONST \ last-d$

## 9.2 First and Time Reversal

**lemma** *BsEqvRule*:

**assumes**  $\vdash f = g$

**shows**  $\vdash bs \ f = bs \ g$

**proof** –

**have**  $1: \vdash f = g$  **using** *assms* **by** *auto*

**hence**  $2: \vdash bi(f) = bi(g)$  **by** (*simp add: BiEqvBi*)

**hence**  $3: \vdash bi(f);skip = bi(g);skip$  **by** (*simp add: LeftChopEqvChop*)

**hence**  $4: \vdash (empty \vee bi(f);skip) = (empty \vee bi(g);skip)$  **by** *auto*

**hence**  $5: \vdash bs(f) = bs(g)$  **by** (*simp add: bs-d-def*)

**from**  $1 \ 2 \ 3 \ 4 \ 5$  **show** *?thesis* **by** *auto*

**qed**

**lemma** *BtEqvRule*:

**assumes**  $\vdash f = g$

**shows**  $\vdash bt \ f = bt \ g$

**proof** –

**have**  $1: \vdash f = g$  **using** *assms* **by** *auto*

**hence**  $2: \vdash \Box(f) = \Box(g)$  **by** (*simp add: BoxEqvBox*)

hence 3:  $\vdash \text{skip};\Box(f) = \text{skip};\Box(g)$  **using** *RightChopEqvChop* **by** *blast*  
 hence 4:  $\vdash (\text{empty} \vee \text{skip};\Box(f)) = (\text{empty} \vee \text{skip};\Box(g))$  **by** *auto*  
 hence 5:  $\vdash \text{bt}(f) = \text{bt}(g)$  **by** (*simp add: bt-d-def*)  
 from 1 2 3 4 5 **show** *?thesis* **by** *auto*  
**qed**

**lemma** *FstEqvRule*:

assumes  $\vdash f = g$   
 shows  $\vdash \triangleright f = \triangleright g$

**proof** –

have 1:  $\vdash f = g$  **using** *assms* **by** *auto*  
 hence 2:  $\vdash (\neg f) = (\neg g)$  **by** *auto*  
 hence 3:  $\vdash \text{bs}(\neg f) = \text{bs}(\neg g)$  **by** (*simp add: BsEqvRule*)  
 hence 4:  $\vdash (f \wedge \text{bs}(\neg f)) = (g \wedge \text{bs}(\neg g))$  **using** 1 **by** *fastforce*  
 from 4 **show** *?thesis* **by** (*simp add: first-d-def*)

**qed**

**lemma** *LstEqvRule*:

assumes  $\vdash f = g$   
 shows  $\vdash \triangleleft f = \triangleleft g$

**proof** –

have 1:  $\vdash f = g$  **using** *assms* **by** *auto*  
 hence 2:  $\vdash (\neg f) = (\neg g)$  **by** *auto*  
 hence 3:  $\vdash \text{bt}(\neg f) = \text{bt}(\neg g)$  **by** (*simp add: BtEqvRule*)  
 hence 4:  $\vdash (f \wedge \text{bt}(\neg f)) = (g \wedge \text{bt}(\neg g))$  **using** 1 **by** *fastforce*  
 from 4 **show** *?thesis* **by** (*simp add: last-d-def*)

**qed**

**lemma** *RBsEqvBt*:

$\vdash (\text{bs } f)^r = (\text{bt } (f^r))$

**proof** –

have 1:  $\vdash (\text{bs } f)^r = (\text{empty} \vee ((\text{bi } f) ; \text{skip}))^r$   
     **by** (*simp add: bs-d-def*)  
 have 2:  $\vdash (\text{empty} \vee ((\text{bi } f) ; \text{skip}))^r = (\text{empty}^r \vee ((\text{bi } f) ; \text{skip})^r)$   
     **using** *ROr* **by** *blast*  
 have 3:  $\vdash (\text{empty}^r \vee ((\text{bi } f) ; \text{skip})^r) = (\text{empty} \vee (\text{skip}^r; (\text{bi } f)^r))$   
     **using** *REmptyEqvEmpty RevChop* **by** *fastforce*  
 have 4:  $\vdash (\text{empty} \vee (\text{skip}^r; (\text{bi } f)^r)) = (\text{empty} \vee (\text{skip}; \Box(f^r)))$   
     **by** (*metis 3 RBiEqvBox RevSkip int-eq*)  
 have 5:  $\vdash (\text{empty} \vee (\text{skip}; \Box(f^r))) = (\text{bt } (f^r))$   
     **by** (*simp add: bt-d-def*)  
 from 1 2 3 4 5 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *RRBsEqvBt*:

$\vdash (\text{bs } (f^r))^r = (\text{bt } (f))$

**proof** –

have 1:  $\vdash (\text{bs } (f^r))^r = \text{bt } ((f^r)^r)$  **using** *RBsEqvBt* **by** *blast*  
 have 2:  $\vdash \text{bt } ((f^r)^r) = \text{bt } f$  **using** *EqvReverseReverse* **using** *BtEqvRule* **by** *blast*  
 from 1 2 **show** *?thesis* **by** *fastforce*

qed

**lemma** *RBtEqvBs*:

$\vdash (bt\ f)^r = (bs\ (f^r))$

**proof** –

**have** 1:  $\vdash (bt\ f)^r = (empty \vee (skip; \square\ f))^r$

**by** (*simp add: bt-d-def*)

**have** 2:  $\vdash (empty \vee (skip; \square\ f))^r = (empty^r \vee (skip; \square\ f)^r)$

**using** *ROr* **by** *blast*

**have** 3:  $\vdash (empty^r \vee (skip; \square\ f)^r) = (empty \vee (\square\ f)^r; skip^r)$

**using** *REmptyEqvEmpty RevChop* **by** *fastforce*

**have** 4:  $\vdash (empty \vee (\square\ f)^r; skip^r) = (empty \vee (bi\ (f^r)); skip)$

**by** (*metis 3 RBoxEqvBi RevSkip int-eq*)

**show** *?thesis*

**by** (*metis 3 4 all-rev-eq(3) bs-d-def bt-d-def inteq-reflection*)

qed

**lemma** *RRBtEqvBs*:

$\vdash (bt\ (f^r))^r = (bs\ (f))$

**proof** –

**have** 1:  $\vdash (bt\ (f^r))^r = bs\ ((f^r)^r)$  **using** *RBtEqvBs* **by** *blast*

**have** 2:  $\vdash bs\ ((f^r)^r) = bs\ f$  **using** *EqvReverseReverse* **using** *BsEqvRule* **by** *blast*

**from** 1 2 **show** *?thesis* **by** *fastforce*

qed

**lemma** *RFirstEqvLast*:

$\vdash (\triangleright\ f)^r = (\triangleleft\ (f^r))$

**proof** –

**have** 1:  $\vdash (\triangleright\ f)^r = (f \wedge bs(\neg f))^r$  **by** (*simp add: first-d-def*)

**have** 2:  $\vdash (f \wedge bs(\neg f))^r = (f^r \wedge (bs\ (\neg f))^r)$  **using** *RAnd* **by** *blast*

**have** 3:  $\vdash (f^r \wedge (bs\ (\neg f))^r) = (f^r \wedge bt\ ((\neg f)^r))$  **using** *RBsEqvBt* **by** *fastforce*

**have** 4:  $\vdash (f^r \wedge bt\ ((\neg f)^r)) = (f^r \wedge bt\ (\neg(f^r)))$  **using** *RNot int-eq* **by** *fastforce*

**have** 5:  $\vdash (f^r \wedge bt\ (\neg(f^r))) = (\triangleleft\ (f^r))$  **by** (*simp add: last-d-def*)

**from** 1 2 3 4 5 **show** *?thesis* **by** *fastforce*

qed

**lemma** *RRFirstEqvLast*:

$\vdash (\triangleright\ (f^r))^r = (\triangleleft\ (f))$

**proof** –

**have** 1:  $\vdash (\triangleright\ (f^r))^r = \triangleleft\ ((f^r)^r)$  **using** *RFirstEqvLast* **by** *blast*

**have** 2:  $\vdash \triangleleft\ ((f^r)^r) = \triangleleft\ f$  **using** *EqvReverseReverse* **using** *LstEqvRule* **by** *blast*

**from** 1 2 **show** *?thesis* **by** *fastforce*

qed

**lemma** *RLastEqvFirst*:

$\vdash (\triangleleft\ f)^r = (\triangleright\ (f^r))$

**proof** –

**have** 1:  $\vdash (\triangleleft\ f)^r = (f \wedge bt(\neg f))^r$  **by** (*simp add: last-d-def*)

**have** 2:  $\vdash (f \wedge bt(\neg f))^r = (f^r \wedge (bt\ (\neg f))^r)$  **using** *RAnd* **by** *blast*

**have** 3:  $\vdash (f^r \wedge (bt\ (\neg f))^r) = (f^r \wedge bs\ ((\neg f)^r))$  **using** *RBtEqvBs* **by** *fastforce*

**have** 4:  $\vdash (f^r \wedge bs(\neg f)^r) = (f^r \wedge bs(\neg(f^r)))$  **using** *RNot int-eq* **by** *fastforce*  
**have** 5:  $\vdash (f^r \wedge bs(\neg(f^r))) = (\triangleright(f^r))$  **by** (*simp add: first-d-def*)  
**from** 1 2 3 4 5 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *RRLastEqvFirst*:

$\vdash (\triangleleft(f^r))^r = (\triangleright(f))$

**proof** –

**have** 1:  $\vdash (\triangleleft(f^r))^r = \triangleright((f^r)^r)$  **using** *RLastEqvFirst* **by** *blast*

**have** 2:  $\vdash \triangleright((f^r)^r) = \triangleright f$  **using** *EqvReverseReverse* **using** *FstEqvRule* **by** *blast*

**from** 1 2 **show** *?thesis* **by** *fastforce*

**qed**

## 9.3 Semantic Theorems

### 9.3.1 Semantics First and Last Operators

**lemma** *FstAndBisem*:

$(ilen\ \sigma > 0 \wedge (\sigma \models f) \wedge (\sigma \models bi(\neg f); skip)) =$   
 $(ilen\ \sigma > 0 \wedge (\sigma \models f) \wedge (\forall ia < ilen\ (\sigma). (prefix\ ia\ \sigma \models \neg f)))$

**proof** –

**have**  $(ilen\ \sigma > 0 \wedge (\sigma \models f) \wedge (\sigma \models bi(\neg f); skip)) =$   
 $(0 < ilen\ \sigma \wedge (\sigma \models f) \wedge$   
 $(\exists i. (i \leq ilen\ \sigma \longrightarrow (\forall ia \leq i. \neg (prefix\ ia\ (prefix\ i\ \sigma) \models f))) \wedge$   
 $ilen\ \sigma - i = Suc\ 0) \wedge i \leq ilen\ \sigma)$   
 $)$

**using** *le-trans* **by** (*auto simp add: chop-defs bi-defs skip-defs, blast*)

**also have** ... =

$(0 < ilen\ \sigma \wedge (\sigma \models f) \wedge$   
 $(\exists i. (i \leq ilen\ \sigma \longrightarrow (\forall ia \leq i. \neg (prefix\ ia\ (prefix\ i\ \sigma) \models f))) \wedge$   
 $i = ilen\ \sigma - Suc\ 0) \wedge i \leq ilen\ \sigma)$   
 $)$

**by** *auto*

**also have** ... =

$(0 < ilen\ \sigma \wedge (\sigma \models f) \wedge$   
 $(\forall ia \leq (ilen\ \sigma - Suc\ 0). \neg (prefix\ ia\ (prefix\ (ilen\ \sigma - Suc\ 0)\ \sigma) \models f))$   
 $)$

**using** *diff-le-self* **by** *blast*

**also have** ... =

$(ilen\ \sigma > 0 \wedge (\sigma \models f) \wedge$   
 $(\forall ia < ilen\ (\sigma). \neg (prefix\ ia\ (prefix\ (ilen\ \sigma - Suc\ 0)\ \sigma) \models f))$   
 $)$  **by** (*metis Suc-pred less-Suc-eq-le*)

**also have** ... =

$(ilen\ \sigma > 0 \wedge (\sigma \models f) \wedge$   
 $(\forall ia < ilen\ (\sigma). (prefix\ ia\ (prefix\ (ilen\ \sigma - Suc\ 0)\ \sigma) \models \neg f))$   
 $)$

**by** *auto*

**also have** ... =

$(ilen\ \sigma > 0 \wedge (\sigma \models f) \wedge (\forall ia < ilen\ (\sigma). (prefix\ ia\ \sigma \models \neg f)))$   
**by** (*simp add: pref-pref-help*)

**finally show**  $(ilen\ \sigma > 0 \wedge (\sigma \models f) \wedge (\sigma \models bi(\neg f); skip)) =$



$$(\text{ilen } \sigma > 0 \wedge (\sigma \models f) \wedge (\forall ia < \text{ilen } (\sigma). (\text{prefix } ia \ \sigma \models \neg f)) ) \ .$$

**qed**

**lemma** *Fstsem-0*:

$$(\sigma \models \triangleright f) =$$

$$(\sigma \models f \wedge \text{empty}) \vee (\text{ilen } \sigma > 0 \wedge (\sigma \models f) \wedge (\sigma \models \text{bi } (\neg f); \text{skip}))$$

**using** *empty-defs* **by** (*auto simp add: first-d-def bs-d-def*)

**lemma** *Emptysem*:

$$(\sigma \models f \wedge \text{empty}) = ((\sigma \models f) \wedge \text{ilen } \sigma = 0)$$

**using** *empty-defs* **by** *auto*

**lemma** *Fstsem*:

$$(\sigma \models \triangleright f) =$$

$$((\sigma \models f) \wedge \text{ilen } \sigma = 0) \vee$$

$$(\text{ilen } \sigma > 0 \wedge (\sigma \models f) \wedge (\forall ia < \text{ilen } (\sigma). (\text{prefix } ia \ \sigma \models \neg f)))$$

**using** *Fstsem-0 Emptysem FstAndBisem* **by** *metis*

**lemma** *Lstsem*:

$$(\sigma \models \triangleleft f) =$$

$$((\sigma \models f) \wedge \text{ilen } \sigma = 0) \vee$$

$$(\text{ilen } \sigma > 0 \wedge (\sigma \models f) \wedge (\forall ia < \text{ilen } \sigma. (\text{suffix } ((\text{ilen } \sigma) - ia) \ \sigma \models \neg f)))$$

**proof** –

$$\text{have } (\sigma \models \triangleleft f) = (\sigma \models (\triangleright (f^r))^r)$$

**using** *RRFirstEqvLast* **by** *fastforce*

$$\text{also have } \dots = (\text{irev } \sigma \models \triangleright (f^r))$$

**by** (*metis reverse-d-def*)

$$\text{also have } \dots =$$

$$((\text{irev } \sigma \models f^r) \wedge \text{ilen } (\text{irev } \sigma) = 0) \vee$$

$$(\text{ilen } (\text{irev } \sigma) > 0 \wedge (\text{irev } \sigma \models f^r) \wedge$$

$$(\forall ia < \text{ilen } (\text{irev } \sigma). (\text{prefix } ia \ (\text{irev } \sigma) \models \neg(f^r))))$$

**using** *Fstsem* **by** *blast*

$$\text{also have } \dots =$$

$$((\sigma \models f) \wedge \text{ilen } (\sigma) = 0) \vee$$

$$(\text{ilen } (\sigma) > 0 \wedge (\sigma \models f) \wedge$$

$$(\forall ia < \text{ilen } (\text{irev } \sigma). (\text{prefix } ia \ (\text{irev } \sigma) \models (\neg(f))^r))))$$

**by** (*simp add: reverse-d-def*)

$$\text{also have } \dots =$$

$$((\sigma \models f) \wedge \text{ilen } (\sigma) = 0) \vee$$

```

  ( ilen ( $\sigma$ ) > 0  $\wedge$  ( $\sigma \models f$ )  $\wedge$ 
    ( $\forall ia < i$ len (irev  $\sigma$ ). (irev (prefix ia (irev  $\sigma$ ))  $\models (\neg(f))$ )))
)
by (simp add: reverse-d-def)
also have ... =
  (
    ( ( $\sigma \models f$ )  $\wedge$  ilen ( $\sigma$ ) = 0)  $\vee$ 
    ( ilen ( $\sigma$ ) > 0  $\wedge$  ( $\sigma \models f$ )  $\wedge$ 
      ( $\forall ia < i$ len ( $\sigma$ ). ( suffix ((ilen  $\sigma$ ) - ia) ( $\sigma$ )  $\models (\neg(f))$ )))
  )
  by (simp add: irev-prefix)
finally show
  ( $\sigma \models \triangleleft f$ ) =
  ( ( ( $\sigma \models f$ )  $\wedge$  ilen  $\sigma$  = 0)  $\vee$ 
    ( ilen  $\sigma$  > 0  $\wedge$  ( $\sigma \models f$ )  $\wedge$ 
      ( $\forall ia < i$ len  $\sigma$ . (suffix ((ilen  $\sigma$ ) - ia)  $\sigma \models \neg f$ )) )
  ) .
qed

```

### 9.3.2 Various Semantic Lemmas

**lemma** *DiLensem*:

```

( $\sigma \models di$  ( $f \wedge len(i)$ )) =
  ( (prefix i  $\sigma \models f$ )  $\wedge$  i  $\leq i$ len  $\sigma$ )

```

**using** *prefix-ilen-good* **by** (*auto simp add: itl-defs*)

**lemma** *PrefixFstsem*:

```

( (prefix i  $\sigma \models \triangleright f$ )  $\wedge$  i  $\leq i$ len  $\sigma$ ) =
  ( i  $\leq i$ len  $\sigma \wedge$ 
    (
      ( (prefix i  $\sigma \models f$ )  $\wedge$  i = 0)  $\vee$ 
      ( i > 0  $\wedge$  (prefix i  $\sigma \models f$ )  $\wedge$  ( $\forall ia < i$ . (prefix ia  $\sigma \models \neg f$ )))
    )
  )

```

**proof** –

```

have 1: ( (prefix i  $\sigma \models \triangleright f$ ) ) =
  (
    ( ((prefix i  $\sigma \models f$ )  $\wedge$  ilen (prefix i  $\sigma$ ) = 0)  $\vee$ 
    ( ilen (prefix i  $\sigma$ ) > 0  $\wedge$  ((prefix i  $\sigma \models f$ )  $\wedge$ 
      ( $\forall ia < i$ len (prefix i  $\sigma$ ). (prefix ia (prefix i  $\sigma$ )  $\models \neg f$ )) )
    )
  )

```

**using** *Fstsem* **by** *blast*

```

hence 2: ( (prefix i  $\sigma \models \triangleright f$ )  $\wedge$  i  $\leq i$ len  $\sigma$ ) =
  ( i  $\leq i$ len  $\sigma \wedge$  (
    ( ((prefix i  $\sigma \models f$ )  $\wedge$  ilen (prefix i  $\sigma$ ) = 0)  $\vee$ 
    ( ilen (prefix i  $\sigma$ ) > 0  $\wedge$  ((prefix i  $\sigma \models f$ )  $\wedge$ 
      ( $\forall ia < i$ len (prefix i  $\sigma$ ). (prefix ia (prefix i  $\sigma$ )  $\models \neg f$ )) )
    )
  )

```

**by** *auto*

**hence 3:**  $((\text{prefix } i \ \sigma) \models \triangleright f) \wedge i \leq \text{ilen } \sigma =$   
 $(i \leq \text{ilen } \sigma \wedge$   
 $( (\text{prefix } i \ \sigma) \models f) \wedge i = 0) \vee$   
 $(i > 0 \wedge ((\text{prefix } i \ \sigma) \models f) \wedge (\forall ia < i. (\text{prefix } ia \ (\text{prefix } i \ \sigma) \models \neg f)))$   
 $)$   
 $)$

**by** *auto*

**hence 4:**  $((\text{prefix } i \ \sigma) \models \triangleright f) \wedge i \leq \text{ilen } \sigma =$   
 $(i \leq \text{ilen } \sigma \wedge$   
 $( (\text{prefix } i \ \sigma) \models f) \wedge i = 0) \vee$   
 $(i > 0 \wedge ((\text{prefix } i \ \sigma) \models f) \wedge (\forall ia < i. (\text{prefix } ia \ \sigma \models \neg f)))$   
 $)$   
 $)$

**using** *pref-pref-3* **using** *less-imp-add-positive* **by** *fastforce*

**from 4** **show** *?thesis* **by** *auto*

**qed**

**lemma** *PrefixFstAndsem:*

$( (\text{prefix } i \ \sigma \models \triangleright f \wedge g) \wedge i \leq \text{ilen } \sigma) =$   
 $(i \leq \text{ilen } \sigma \wedge$   
 $($   
 $( (\text{prefix } i \ \sigma \models f \wedge g) \wedge i = 0) \vee$   
 $(i > 0 \wedge (\text{prefix } i \ \sigma \models f \wedge g) \wedge (\forall ia < i. (\text{prefix } ia \ \sigma \models \neg f)))$   
 $)$   
 $)$

**using** *PrefixFstsem[of f i σ]* **by** *(metis unl-lift2)*

**lemma** *DiLenFstsem:*

$(\sigma \models di (\triangleright f \wedge \text{len}(i))) =$   
 $(i \leq \text{ilen } \sigma \wedge$   
 $($   
 $( (\text{prefix } i \ \sigma \models f) \wedge i = 0) \vee$   
 $(i > 0 \wedge (\text{prefix } i \ \sigma \models f) \wedge (\forall ia < i. (\text{prefix } ia \ \sigma \models \neg f)))$   
 $)$   
 $)$

**by** *(simp add: DiLensem PrefixFstsem)*

**lemma** *DiLenFstAndsem:*

$(\sigma \models di ((\triangleright f \wedge g) \wedge \text{len}(i))) =$   
 $(i \leq \text{ilen } \sigma \wedge$   
 $($   
 $( (\text{prefix } i \ \sigma \models f \wedge g) \wedge i = 0) \vee$   
 $(i > 0 \wedge (\text{prefix } i \ \sigma \models f \wedge g) \wedge (\forall ia < i. (\text{prefix } ia \ \sigma \models \neg f)))$   
 $)$   
 $)$

**using** *DiLensem PrefixFstAndsem* **by** *metis*

**lemma** *FstLenSamesem:*

$( (i \leq \text{ilen } \sigma \wedge$   
 $($

$$\begin{aligned}
& ( (prefix\ i\ \sigma \models f) \wedge i = 0) \vee \\
& ( i > 0 \wedge (prefix\ i\ \sigma \models f) \wedge (\forall ia < i. (prefix\ ia\ \sigma \models \neg f))) \\
& ) \\
& ) \wedge \\
& ( j \leq len\ \sigma \wedge \\
& ( \\
& ( (prefix\ j\ \sigma \models f) \wedge j = 0) \vee \\
& ( j > 0 \wedge (prefix\ j\ \sigma \models f) \wedge (\forall ia < j. (prefix\ ia\ \sigma \models \neg f))) \\
& ) \\
& ) \\
& ) \longrightarrow (i=j)
\end{aligned}$$

by (*metis not-less-iff-gr-or-eq unl-lift*)

## 9.4 Theorems

### 9.4.1 Fixed length intervals

**lemma** *LenZeroEqvEmpty*:

$\vdash len(0) = empty$

by (*simp add: len-d-def*)

**lemma** *LenOneEqvSkip*:

$\vdash len(1) = skip$

by (*simp add: len-d-def ChopEmpty*)

**lemma** *LenNPlusOneA*:

$\vdash len(n+1) = skip;len(n)$

by (*simp add: len-d-def*)

**lemma** *LenEqvLenChopLen*:

$\vdash len(i+j) = len(i);len(j)$

**proof**

(*induct i*)

**case** 0

**then show** ?case

by (*metis EmptyChop LenZeroEqvEmpty add.left-neutral inteq-reflection*)

**next**

**case** (*Suc i*)

**then show** ?case

by (*metis ChopAssoc add-Suc int-eq len-d-def pow-Suc*)

qed

**lemma** *LenNPlusOneB*:

$\vdash len(n+1) = len(n);skip$

**proof** –

**have** 1:  $\vdash len(n+1) = len(n);len(1)$  **by** (*rule LenEqvLenChopLen*)

**have** 2:  $\vdash len(1) = skip$  **by** (*rule LenOneEqvSkip*)

**hence** 3:  $\vdash len(n);len(1) = len(n);skip$  **using** *RightChopEqvChop* **by** *blast*

**from** 1 3 **show** ?thesis **by** *fastforce*

qed

**lemma** *LenCommute*:

$\vdash (skip;(len\ n)) = (len\ n);skip$

**proof**

(*induct n*)

**case** 0

**then show** ?*case*

**by** (*metis LenEqvLenChopLen LenNPlusOneA LenOneEqvSkip inteq-reflection*)

**next**

**case** (*Suc n*)

**then show** ?*case*

**by** (*metis LenEqvLenChopLen LenNPlusOneA LenOneEqvSkip inteq-reflection*)

qed

**lemma** *SkipTrueEqvTrueSkip*:

$\vdash skip;\#True = \#True;skip$

**using** *TrueChopSkipEqvSkipChopTrue* **by** *fastforce*

**lemma** *PowerCommute*:

$\vdash (f;(power\ f\ n)) = ((power\ f\ n);f)$

**proof**

(*induct n*)

**case** 0

**then show** ?*case* **using** *EmptyChop ChopEmpty pow-0* **by** (*metis int-eq*)

**next**

**case** (*Suc n*)

**then show** ?*case* **using** *ChopAssoc pow-Suc* **by** (*metis inteq-reflection*)

qed

**lemma** *PowerRev*:

$\vdash (power\ skip\ n)^r = (power\ skip\ n)$

**proof**

(*induct n*)

**case** 0

**then show** ?*case* **using** *REmptyEqvEmpty* **by** *auto*

**next**

**case** (*Suc n*)

**then show** ?*case* **using** *PowerCommute RevChop pow-Suc* **by** (*metis RevSkip int-eq*)

qed

**lemma** *RLenEqvLen*:

$\vdash (len\ k)^r = (len\ k)$

**proof**

(*induct k*)

**case** 0

**then show** ?*case*

**by** (*metis PowerRev len-d-def*)

**next**

**case** (*Suc k*)

**then show** *?case*  
**by** (*metis PowerRev len-d-def*)  
**qed**

**lemma** *PowerSkipEqLen*:  
 $\vdash (\text{power skip } n) = (\text{len } n)$   
**by** (*simp add: len-d-def*)

**lemma** *ExistsLen*:  
 $\vdash \exists k. \text{len}(k)$   
**by** (*simp add: len-defs Valid-def*)

**lemma** *AndExistsLen*:  
 $\vdash f = (f \wedge (\exists k. \text{len}(k)))$   
**using** *ExistsLen* **by** *fastforce*

**lemma** *AndExistsLenChop*:  
 $\vdash (f;g) = (\exists k. (f \wedge \text{len}(k));g)$   
**by** (*simp add: Valid-def len-defs chop-defs*)

**lemma** *AndExistsLenChopR*:  
 $\vdash (f;g) = (\exists k. f;(g \wedge \text{len}(k)))$   
**by** (*simp add: Valid-def len-defs chop-defs*)

**lemma** *LFixedAndDistr*:  
 $\vdash ((f0 \wedge \text{len}(k));g0 \wedge (f1 \wedge \text{len}(k));g1) = ((f0 \wedge f1) \wedge \text{len}(k));(g0 \wedge g1)$   
**by** (*auto simp add: min.absorb1 Valid-def len-defs chop-defs*)

**lemma** *RFixedAndDistr*:  
 $\vdash (f0;(g0 \wedge \text{len}(k)) \wedge f1;(g1 \wedge \text{len}(k))) = (f0 \wedge f1);((g0 \wedge g1) \wedge \text{len}(k))$   
**by** (*simp add: Valid-def min.absorb1 len-defs chop-defs*) (*metis diff-diff-cancel*)

**lemma** *LFixedAndDistrA*:  
 $\vdash ((f0 \wedge \text{len}(k));g0 \wedge (f1 \wedge \text{len}(k));g0) = ((f0 \wedge f1) \wedge \text{len}(k));g0$   
**proof** –  
**have** 1:  $\vdash ((f0 \wedge \text{len}(k));g0 \wedge (f1 \wedge \text{len}(k));g0) = ((f0 \wedge f1) \wedge \text{len}(k));(g0 \wedge g0)$   
**by** (*rule LFixedAndDistr*)  
**have** 2:  $\vdash ((f0 \wedge f1) \wedge \text{len}(k));(g0 \wedge g0) = ((f0 \wedge f1) \wedge \text{len}(k));g0$   
**by** *auto*  
**from** 1 2 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *LFixedAndDistrB*:  
 $\vdash ((f0 \wedge \text{len}(k));g0 \wedge (f0 \wedge \text{len}(k));g1) = (f0 \wedge \text{len}(k));(g0 \wedge g1)$   
**proof** –  
**have** 1:  $\vdash ((f0 \wedge \text{len}(k));g0 \wedge (f0 \wedge \text{len}(k));g1) = ((f0 \wedge f0) \wedge \text{len}(k));(g0 \wedge g1)$   
**by** (*rule LFixedAndDistr*)  
**have** 2:  $\vdash ((f0 \wedge f0) \wedge \text{len}(k));(g0 \wedge g1) = (f0 \wedge \text{len}(k));(g0 \wedge g1)$   
**by** *auto*

from 1 2 show ?thesis by fastforce  
qed

**lemma** *LFixedAndDistrB1*:

$\vdash (\text{len}(k);f \wedge \text{len}(k);g) = \text{len}(k);(f \wedge g)$

**proof** –

have 1:  $\vdash \text{len}(k);f = (\# \text{True} \wedge \text{len}(k));f$

by auto

have 2:  $\vdash \text{len}(k);g = (\# \text{True} \wedge \text{len}(k));g$

by auto

have 3:  $\vdash (\text{len}(k);f \wedge \text{len}(k);g) = ((\# \text{True} \wedge \text{len}(k));f \wedge (\# \text{True} \wedge \text{len}(k));g)$

using 1 2 by auto

have 4:  $\vdash ((\# \text{True} \wedge \text{len}(k));f \wedge (\# \text{True} \wedge \text{len}(k));g) = (\# \text{True} \wedge \text{len}(k));(f \wedge g)$

using *LFixedAndDistrB* by blast

have 5:  $\vdash (\# \text{True} \wedge \text{len}(k));(f \wedge g) = (\text{len}(k));(f \wedge g)$

by auto

from 1 2 3 4 5 show ?thesis by auto

qed

**lemma** *RFixedAndDistrA*:

$\vdash (f0;(g0 \wedge \text{len}(k)) \wedge f0;(g1 \wedge \text{len}(k))) = f0;((g0 \wedge g1) \wedge \text{len}(k))$

**proof** –

have 1:  $\vdash (f0;(g0 \wedge \text{len}(k)) \wedge f0;(g1 \wedge \text{len}(k))) = (f0 \wedge f0);((g0 \wedge g1) \wedge \text{len}(k))$

by (rule *RFixedAndDistr*)

have 2:  $\vdash (f0 \wedge f0);((g0 \wedge g1) \wedge \text{len}(k)) = f0;((g0 \wedge g1) \wedge \text{len}(k))$

by auto

from 1 2 show ?thesis by fastforce

qed

**lemma** *RFixedAndDistrB*:

$\vdash (f0;(g0 \wedge \text{len}(k)) \wedge f1;(g0 \wedge \text{len}(k))) = (f0 \wedge f1);(g0 \wedge \text{len}(k))$

**proof** –

have 1:  $\vdash (f0;(g0 \wedge \text{len}(k)) \wedge f1;(g0 \wedge \text{len}(k))) = (f0 \wedge f1);((g0 \wedge g0) \wedge \text{len}(k))$

by (rule *RFixedAndDistr*)

have 2:  $\vdash (f0 \wedge f1);((g0 \wedge g0) \wedge \text{len}(k)) = (f0 \wedge f1);(g0 \wedge \text{len}(k))$

by auto

from 1 2 show ?thesis by fastforce

qed

**lemma** *ChopSkipAndChopSkip*:

$\vdash (f0;\text{skip} \wedge f1;\text{skip}) = (f0 \wedge f1);\text{skip}$

**proof** –

have 1:  $\vdash (f0;(\# \text{True} \wedge \text{len}(1)) \wedge f1;(\# \text{True} \wedge \text{len}(1))) = (f0 \wedge f1);(\# \text{True} \wedge \text{len}(1))$

by (rule *RFixedAndDistrB*)

have 2:  $\vdash (\# \text{True} \wedge \text{len}(1)) = \text{skip}$

using *LenOneEqvSkip* by fastforce

hence 3:  $\vdash f0;(\# \text{True} \wedge \text{len}(1)) = f0;\text{skip}$

using *RightChopEqvChop* by blast

have 4:  $\vdash f1;(\# \text{True} \wedge \text{len}(1)) = f1;\text{skip}$

using 2 *RightChopEqvChop* by blast

**have** 5:  $\vdash (f0;(\#True \wedge len(1)) \wedge f1;(\#True \wedge len(1))) = (f0;skip \wedge f1;skip)$   
**using** 3 4 **by** *fastforce*  
**have** 6:  $\vdash (f0 \wedge f1);(\#True \wedge len(1)) = (f0 \wedge f1);skip$   
**using** 2 *RightChopEqvChop* **by** *blast*  
**from** 1 5 6 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *BiAndChopSkipEqv*:  
 $\vdash (bi (f \wedge g));skip = ((bi f);skip \wedge (bi g);skip)$   
**proof** –  
**have** 1:  $\vdash bi (f \wedge g) = ((bi f) \wedge (bi g))$   
**by** (*auto simp add: bi-defs Valid-def*)  
**hence** 2:  $\vdash (bi (f \wedge g));skip = (bi f \wedge bi g);skip$   
**by** (*rule LeftChopEqvChop*)  
**have** 3:  $\vdash (bi f \wedge bi g);skip = ((bi f);skip \wedge (bi g);skip)$   
**using** *ChopSkipAndChopSkip* **by** *fastforce*  
**from** 2 3 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *DiAndChopSkipEqv*:  
 $\vdash (di (f \wedge g));skip \longrightarrow (di f);skip \wedge (di g);skip$   
**proof** –  
**have** 1:  $\vdash di (f \wedge g) \longrightarrow (di f) \wedge (di g)$   
**by** (*simp add: DiAndImpAnd*)  
**hence** 2:  $\vdash (di (f \wedge g));skip \longrightarrow (di f \wedge di g);skip$   
**by** (*rule LeftChopImpChop*)  
**have** 3:  $\vdash (di f \wedge di g);skip = ((di f);skip \wedge (di g);skip)$   
**using** *ChopSkipAndChopSkip* **by** *fastforce*  
**from** 2 3 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *ChopEmptyAndEmpty*:  
 $\vdash (f;g \wedge empty) = (f \wedge g \wedge empty)$   
**by** (*simp add: Valid-def itl-defs*)  
*(metis prefix-ilen suffix-zero le-zero-eq)*

**lemma** *ChopSkipImpMore*:  
 $\vdash f;skip \longrightarrow more$   
**using** *ChopImpDiamond MoreEqvSkipChopTrue SkipTrueEqvTrueSkip TrueChopEqvDiamond* **by** *fastforce*

**lemma** *MoreEqvMoreChopTrue*:  
 $\vdash more = more;\#True$   
**proof** –  
**have** 1:  $\vdash more = skip;\#True$   
**using** *MoreEqvSkipChopTrue* **by** *blast*  
**have** 2:  $\vdash \#True = \#True;\#True$   
**by** (*auto simp add: Valid-def chop-defs*)  
**hence** 3:  $\vdash skip;\#True = skip;(\#True;\#True)$   
**using** *RightChopEqvChop* **by** *blast*  
**have** 4:  $\vdash skip;(\#True;\#True) = (skip;\#True);\#True$



**using** *ChopAssoc* **by** *blast*  
**have** 5:  $\vdash (skip; \# True); \# True = more; \# True$   
**using** *MoreEqvSkipChopTrue* **by** (*simp add: more-d-def next-d-def*)  
**from** 1 3 4 5 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *NotNotChopSkip*:  
 $\vdash (\neg((\neg f); skip)) = (empty \vee (f; skip))$   
**by** (*metis WprevEqvEmptyOrPrev prev-d-def wprev-d-def*)

**lemma** *NotChopFixed*:  
 $\vdash (\neg(f; (g \wedge len(k)))) = (\neg(\Diamond(g \wedge len(k))) \vee ((\neg f); (g \wedge len(k))))$   
**by** (*auto simp add: itl-defs Valid-def*)  
*(metis diff-diff-cancel)*

**lemma** *NotFixedChop*:  
 $\vdash (\neg((g \wedge len(k)); f)) = (\neg(di(g \wedge len(k))) \vee ((g \wedge len(k)); (\neg f)))$   
**by** (*auto simp add: itl-defs min.absorb1 Valid-def*)

**lemma** *NotChopNotSkip*:  
 $\vdash (\neg(f; skip)) = (empty \vee ((\neg f); skip))$   
**proof** –  
**have** 1:  $\vdash (\neg((\neg(\neg f)); skip)) = (empty \vee ((\neg f); skip))$  **using** *NotNotChopSkip* **by** *blast*  
**have** 2:  $\vdash (\neg((\neg(\neg f)); skip)) = (\neg(f; skip))$  **by** *auto*  
**from** 1 2 **show** ?thesis **by** *auto*  
**qed**

#### 9.4.2 Additional ITL theorems

**lemma** *BiOrBiImpBiOr*:  
 $\vdash bi\ f \vee bi\ g \longrightarrow bi(f \vee g)$   
**proof** –  
**have** 1:  $\vdash f \longrightarrow f \vee g$  **by** *auto*  
**hence** 2:  $\vdash bi\ f \longrightarrow bi(f \vee g)$  **by** (*rule BiImpBiRule*)  
**have** 3:  $\vdash g \longrightarrow f \vee g$  **by** *auto*  
**hence** 4:  $\vdash bi\ g \longrightarrow bi(f \vee g)$  **by** (*rule BiImpBiRule*)  
**from** 2 4 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *MoreAndBiImpBiChopSkip*:  
 $\vdash more \wedge bi\ f \longrightarrow (bi\ f); skip$   
**proof** –  
**have** 1:  $\vdash (bi\ f); skip = ((\neg(di(\neg f))); skip)$  **by** (*simp add: bi-d-def*)  
**have** 2:  $\vdash (\neg(\neg(di(\neg f))); skip) = (empty \vee (di(\neg f)); skip)$  **by** (*rule NotNotChopSkip*)  
**have** 3:  $\vdash empty \longrightarrow empty \vee di(\neg f)$  **by** *auto*  
**have** 4:  $\vdash (di(\neg f)); skip \longrightarrow di(\neg f)$  **using** *ChopImpDi DiEqvDiDi* **by** *fastforce*  
**hence** 5:  $\vdash (di(\neg f)); skip \longrightarrow empty \vee di(\neg f)$  **by** (*rule Prop05*)  
**have** 6:  $\vdash \neg(\neg(di(\neg f)); skip) \longrightarrow empty \vee di(\neg f)$  **using** 2 3 5 **by** *fastforce*  
**hence** 7:  $\vdash \neg(empty \vee di(\neg f)) \longrightarrow \neg(\neg(\neg(di(\neg f)); skip))$  **by** *fastforce*

**have** 8:  $\vdash (\neg(\neg(\neg(di(\neg f));skip))) = ((\neg(di(\neg f));skip))$  **by** *auto*  
**have** 9:  $\vdash (\neg(empty \vee di(\neg f))) = (more \wedge \neg(di(\neg f)))$   
**using** *NotEmptyEqvMore* **by** *auto*  
**have** 10:  $\vdash (more \wedge \neg(di(\neg f))) = (more \wedge bi\ f)$  **by** (*simp add: bi-d-def*)  
**from** 1 6 7 8 9 10 **show** *?thesis* **by** (*metis int-eq*)  
**qed**

**lemma** *DiChopImpDiB*:

$\vdash di(f;g) \longrightarrow di\ f$

**proof** –

**have** 1:  $\vdash f ; (g;\#True) \longrightarrow di\ f$  **by** (*rule ChopImpDi*)  
**have** 2:  $\vdash f ; (g;\#True) = (f;g);\#True$  **by** (*rule ChopAssoc*)  
**from** 1 2 **show** *?thesis* **by** (*metis di-d-def int-eq*)

**qed**

**lemma** *BiBiOrImpBi*:

$\vdash bi\ (bi\ f \vee bi\ g) \longrightarrow bi\ f \vee bi\ g$

**using** *BiElim* **by** *auto*

**lemma** *BiImpBiBiOr*:

$\vdash bi\ f \longrightarrow bi\ (bi\ f \vee bi\ g)$

**proof** –

**have** 1:  $\vdash bi\ f \longrightarrow bi\ f \vee bi\ g$  **by** *auto*  
**hence** 2:  $\vdash bi\ (bi\ f) \longrightarrow bi\ (bi\ f \vee bi\ g)$  **using** *BiImpBiRule* **by** *blast*  
**have** 3:  $\vdash bi\ (bi\ f) = bi\ f$  **using** *BiEqvBiBi* **by** *fastforce*  
**from** 2 3 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *BiImpBiBiOrB*:

$\vdash bi\ g \longrightarrow bi\ (bi\ f \vee bi\ g)$

**proof** –

**have** 1:  $\vdash bi\ g \longrightarrow bi\ f \vee bi\ g$  **by** *auto*  
**hence** 2:  $\vdash bi\ (bi\ g) \longrightarrow bi\ (bi\ f \vee bi\ g)$  **using** *BiImpBiRule* **by** *blast*  
**have** 3:  $\vdash bi\ (bi\ g) = bi\ g$  **using** *BiEqvBiBi* **by** *fastforce*  
**from** 2 3 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *BiBiOrEqvBi*:

$\vdash bi\ (bi\ f \vee bi\ g) = bi\ f \vee bi\ g$

**proof** –

**have** 1:  $\vdash bi\ (bi\ f \vee bi\ g) \longrightarrow bi\ f \vee bi\ g$  **by** (*rule BiBiOrImpBi*)  
**have** 2:  $\vdash bi\ f \longrightarrow bi\ (bi\ f \vee bi\ g)$  **by** (*rule BiImpBiBiOr*)  
**have** 3:  $\vdash bi\ g \longrightarrow bi\ (bi\ f \vee bi\ g)$  **by** (*rule BiImpBiBiOrB*)  
**have** 4:  $\vdash bi\ f \vee bi\ g \longrightarrow bi\ (bi\ f \vee bi\ g)$  **using** 2 3 **by** *fastforce*  
**from** 1 4 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *DiEqvOrDiChopSkipA*:

$\vdash di\ f = (f \vee di(f;skip))$

**proof** –

**have** 1:  $\vdash di\ f = f ; \#True$  **by** (*simp add: di-d-def*)  
**hence** 2:  $\vdash di\ f = f ; (empty \vee more)$  **by** (*simp add: empty-d-def*)  
**hence** 3:  $\vdash f ; (empty \vee more) = (f ; empty \vee f ; more)$  **using** *ChopOrEqv* **by** *blast*  
**have** 4:  $\vdash f ; empty = f$  **by** (*rule ChopEmpty*)  
**have** 5:  $\vdash more = skip ; \#True$  **using** *MoreEqvSkipChopTrue* **by** *blast*  
**hence** 6:  $\vdash f ; more = f ; (skip ; \#True)$  **using** *RightChopEqvChop* **by** *blast*  
**have** 7:  $\vdash f ; (skip ; \#True) = (f ; skip) ; \#True$  **by** (*rule ChopAssoc*)  
**from** 2 3 4 6 7 **show** *?thesis* **by** (*metis di-d-def int-eq*)  
**qed**

**lemma** *DiEqvOrDiChopSkipB*:

$\vdash di\ f = (f \vee (di\ f) ; skip)$

**proof** –

**have** 1:  $\vdash (di\ f) = (f \vee di(f ; skip))$  **by** (*rule DiEqvOrDiChopSkipA*)  
**have** 2:  $\vdash di(f ; skip) = (f ; skip) ; \#True$  **by** (*simp add: di-d-def*)  
**have** 3:  $\vdash (f ; skip) ; \#True = f ; (skip ; \#True)$  **by** (*rule ChopAssocB*)  
**have** 4:  $\vdash di(f ; skip) = f ; (skip ; \#True)$  **using** 2 3 **by** *fastforce*  
**have** 5:  $\vdash skip ; \#True = \#True ; skip$  **by** (*rule SkipTrueEqvTrueSkip*)  
**hence** 6:  $\vdash f ; (skip ; \#True) = f ; (\#True ; skip)$  **using** *RightChopEqvChop* **by** *blast*  
**have** 7:  $\vdash di(f ; skip) = f ; (\#True ; skip)$  **using** 4 6 **by** *fastforce*  
**have** 8:  $\vdash f ; (\#True ; skip) = (f ; \#True) ; skip$  **by** (*rule ChopAssoc*)  
**have** 9:  $\vdash (f ; \#True) ; skip = (di\ f) ; skip$  **by** (*simp add: di-d-def*)  
**have** 10:  $\vdash di(f ; skip) = (di\ f) ; skip$  **using** 7 8 9 **by** *fastforce*  
**hence** 11:  $\vdash (f \vee di(f ; skip)) = (f \vee (di\ f) ; skip)$  **by** *auto*  
**from** 1 11 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *BiEqvAndEmptyOrBiChopSkip*:

$\vdash bi\ f = (f \wedge (empty \vee (bi\ f) ; skip))$

**proof** –

**have** 1:  $\vdash di\ (\neg f) = (\neg f \vee (di\ (\neg f) ; skip))$  **by** (*rule DiEqvOrDiChopSkipB*)  
**have** 2:  $\vdash di\ (\neg f) = (\neg (bi\ f))$  **by** (*rule DiNotEqvNotBi*)  
**have** 3:  $\vdash (\neg (bi\ f)) = (\neg f \vee (di\ (\neg f) ; skip))$  **using** 1 2 **by** *fastforce*  
**hence** 4:  $\vdash bi\ f = (\neg (\neg f \vee (di\ (\neg f) ; skip)))$  **by** *auto*  
**have** 5:  $\vdash (\neg (\neg f \vee (di\ (\neg f) ; skip))) = (f \wedge \neg (di\ (\neg f) ; skip))$  **by** *auto*  
**have** 6:  $\vdash di\ (\neg f) ; skip = ((\neg (bi\ f)) ; skip)$  **by** (*simp add: 2 LeftChopEqvChop*)  
**hence** 7:  $\vdash (\neg (di\ (\neg f) ; skip)) = (\neg ((\neg (bi\ f)) ; skip))$  **by** *auto*  
**have** 8:  $\vdash (\neg ((\neg (bi\ f)) ; skip)) = (empty \vee (bi\ f) ; skip)$  **using** *NotNotChopSkip* **by** *blast*  
**hence** 9:  $\vdash (f \wedge \neg (di\ (\neg f) ; skip)) = (f \wedge (empty \vee (bi\ f) ; skip))$  **using** 7 8 **by** *fastforce*  
**from** 4 5 9 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *DiDiAndEqvDi*:

$\vdash di\ (di\ f \wedge di\ g) = (di\ f \wedge di\ g)$

**proof** –

**have** 1:  $\vdash bi\ (bi\ (\neg f) \vee bi\ (\neg g)) = (bi\ (\neg f) \vee bi\ (\neg g))$   
**by** (*meson BiBiOrImpBi BiImpBiBiOr BiImpBiBiOrB Prop02 int-iffI*)  
**have** 2:  $\vdash bi\ (\neg f) = (\neg (di\ f))$   
**by** (*simp add: bi-d-def*)  
**have** 3:  $\vdash bi\ (\neg g) = (\neg (di\ g))$

by (simp add: bi-d-def)  
 have 4:  $\vdash (bi (\neg f) \vee bi (\neg g)) = (\neg (di f) \vee \neg (di g))$   
 using 2 3 by fastforce  
 have 5:  $\vdash (\neg (di f) \vee \neg (di g)) = (\neg (di f \wedge di g))$   
 by auto  
 have 6:  $\vdash bi (bi (\neg f) \vee bi (\neg g)) = (\neg (di f \wedge di g))$   
 using 1 5 4 by fastforce  
 hence 7:  $\vdash (\neg (bi (bi (\neg f) \vee bi (\neg g)))) = (di f \wedge di g)$   
 by auto  
 have 8:  $\vdash (\neg (bi (bi (\neg f) \vee bi (\neg g)))) = di (\neg (bi (\neg f) \vee bi (\neg g)))$   
 using DiNotEqvNotBi by fastforce  
 have 9:  $\vdash (\neg (bi (\neg f) \vee bi (\neg g))) = (di f \wedge di g)$   
 using 1 7 by fastforce  
 hence 10:  $\vdash di (\neg (bi (\neg f) \vee bi (\neg g))) = di (di f \wedge di g)$   
 using DiEqvDi by blast  
 from 7 8 10 show ?thesis by fastforce  
 qed

lemma BiInduct:

$\vdash bi(f \longrightarrow wprev f) \wedge f \longrightarrow bi f$

proof –

have 1:  $\vdash \Box((f^r) \longrightarrow wnext(f^r)) \wedge f^r \longrightarrow \Box(f^r)$  using BoxInduct by blast  
 hence 2:  $\vdash (\Box((f^r) \longrightarrow wnext(f^r)) \wedge f^r \longrightarrow \Box(f^r))^r$  using ReverseEqv by blast  
 have 3:  $\vdash ((f^r)^r) = f$  by (simp add: EqvReverseReverse)  
 have 4:  $\vdash (\Box(f^r))^r = bi(f)$  using RRBBoxEqvBi by blast  
 have 5:  $\vdash ((f^r) \longrightarrow wnext(f^r))^r = (f^r \longrightarrow (wnext(f^r))^r)$  by (simp add: rev-fun2)  
 have 6:  $\vdash (wnext(f^r))^r = wprev(f)$  using RRWNextEqvWPrev by blast  
 have 7:  $\vdash (f^r \longrightarrow (wnext(f^r))^r) = (f \longrightarrow wprev(f))$  using 6 3 by fastforce  
 have 8:  $\vdash bi(f^r \longrightarrow (wnext(f^r))^r) = bi(f \longrightarrow wprev(f))$  using 7 3 BiEqvBi by blast  
 have 9:  $\vdash (\Box(f^r) \longrightarrow wnext(f^r))^r = bi((f^r) \longrightarrow wnext(f^r))^r$  using RBoxEqvBi by blast  
 have 10:  $\vdash (\Box(f^r) \longrightarrow wnext(f^r))^r = bi(f \longrightarrow wprev(f))$  using 8 9 5 int-eq by fastforce  
 have 11:  $\vdash (\Box((f^r) \longrightarrow wnext(f^r)) \wedge f^r \longrightarrow \Box(f^r))^r =$   
 $((\Box((f^r) \longrightarrow wnext(f^r)))^r \wedge (f^r)^r \longrightarrow (\Box(f^r))^r)$  by (metis int-eq rev-fun2)  
 have 12:  $\vdash ((\Box((f^r) \longrightarrow wnext(f^r)))^r \wedge (f^r)^r \longrightarrow (\Box(f^r))^r) =$   
 $(bi(f \longrightarrow wprev(f)) \wedge f \longrightarrow bi f)$  using 8 3 4 10 by fastforce  
 from 2 11 12 show ?thesis using MP by fastforce  
 qed

lemma PrevLoop:

assumes  $\vdash f \longrightarrow prev f$

shows  $\vdash \neg f$

proof –

have 1:  $\vdash f \longrightarrow prev f$  using assms by auto  
 hence 2:  $\vdash f \longrightarrow (more \wedge wprev f)$   
 by (metis ChopSkipImpMore Prop05 Prop12 WprevEqvEmptyOrPrev inteq-reflection lift-imp-trans prev-d-def)  
 hence 3:  $\vdash f \longrightarrow wprev f$  by auto  
 hence 4:  $\vdash bi(f \longrightarrow wprev f)$  by (rule BiGen)  
 have 5:  $\vdash bi(f \longrightarrow wprev f) \wedge f \longrightarrow bi f$  by (rule BiInduct)  
 hence 6:  $\vdash bi(f \longrightarrow wprev f) \longrightarrow (f \longrightarrow bi f)$  by fastforce

**have** 7:  $\vdash (f \longrightarrow bi\ f)$  **using** 4 6 *MP* **by** *blast*  
**have** 8:  $\vdash bi\ f \longrightarrow f$  **by** (rule *BiElim*)  
**have** 9:  $\vdash f = bi\ f$  **using** 7 8 **by** *fastforce*  
**have** 10:  $\vdash f \longrightarrow more$  **using** 2 **by** *auto*  
**hence** 11:  $\vdash bi\ f \longrightarrow bi\ more$  **using** *BiImpBiRule* **by** *blast*  
**have** 12:  $\vdash \neg(bi\ more)$  **using** *DiEmpty* *bi-d-def* *empty-d-def* **by** (*simp* *add*: *bi-d-def* *empty-d-def*)  
**from** 7 9 11 12 **show** *?thesis* **using** *MP* **by** *fastforce*  
**qed**

**lemma** *PrevImpNotPrevNot*:

$\vdash prev\ f \longrightarrow \neg(prev\ (\neg\ f))$   
**by** (*metis* (*no-types*, *lifting*) *NextImpNotNextNot* *RPrevEqvNext* *ReverseEqv* *inteq-reflection* *rev-fun1* *rev-fun2*)

**lemma** *BiEqvAndWprevBi*:

$\vdash bi\ f = (f \wedge wprev(bi\ f))$   
**by** (*metis* *BiEqvAndEmptyOrBiChopSkip* *WprevEqvEmptyOrPrev* *inteq-reflection* *prev-d-def*)

**lemma** *DiIntroLoop*:

**assumes**  $\vdash (f \wedge \neg\ g) \longrightarrow prev\ f$   
**shows**  $\vdash f \longrightarrow di\ g$   
**using** *assms* *DiamondIntro*[*of* *LIFT*( $f^r$ ) *LIFT*( $g^r$ )]  
**by** (*metis* (*no-types*, *lifting*) *RDiEqvDiamond* *RPrevEqvNext* *ReverseEqv* *inteq-reflection* *rev-fun2* *rev-fun1*)

**lemma** *DiEqvOrChopMore*:

$\vdash di\ f = (f \vee f;more)$   
**proof** –  
**have** 1:  $\vdash di\ f = f; \# True$  **by** (*simp* *add*: *di-d-def*)  
**hence** 2:  $\vdash di\ f = f; (empty \vee more)$  **by** (*simp* *add*: *empty-d-def*)  
**have** 3:  $\vdash f; (empty \vee more) = (f;empty \vee f;more)$  **by** (*simp* *add*: *ChopOrEqv*)  
**have** 4:  $\vdash f;empty = f$  **by** (rule *ChopEmpty*)  
**from** 2 3 4 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *DiAndDiEqvDiAndDiOrDiAndDi*:

$\vdash (di\ f \wedge di\ g) = (di(f \wedge di\ g) \vee di(g \wedge di\ f))$   
**proof** –  
**have** 1:  $\vdash di\ f = (f \vee f;more)$   
**using** *DiEqvOrChopMore* **by** *blast*  
**have** 2:  $\vdash di\ g = (g \vee g;more)$   
**using** *DiEqvOrChopMore* **by** *blast*  
**have** 3:  $\vdash (di\ f \wedge di\ g) = ((f \vee f;more) \wedge (g \vee g;more))$   
**using** 1 2 **by** *fastforce*  
**have** 4:  $\vdash ((f \vee f;more) \wedge (g \vee g;more)) =$   
 $((f \wedge g) \vee (f \wedge g;more) \vee (g \wedge f;more) \vee (f;more \wedge g;more))$   
**by** *auto*

**have** 5:  $\vdash \text{more} = \# \text{True}; \text{skip}$   
**using** *MoreEqvSkipChopTrue SkipTrueEqvTrueSkip* **by** *fastforce*  
**hence** 6:  $\vdash f; \text{more} = f; (\# \text{True}; \text{skip})$   
**using** *RightChopEqvChop* **by** *blast*  
**have** 7:  $\vdash f; (\# \text{True}; \text{skip}) = (f; \# \text{True}); \text{skip}$   
**by** (*rule ChopAssoc*)  
**have** 8:  $\vdash f; \text{more} = \text{prev} (di f)$   
**using** 6 7 **by** (*metis di-d-def int-eq prev-d-def*)  
**have** 9:  $\vdash g; \text{more} = g; (\# \text{True}; \text{skip})$   
**using** 5 *RightChopEqvChop* **by** *blast*  
**have** 10:  $\vdash g; (\# \text{True}; \text{skip}) = (g; \# \text{True}); \text{skip}$   
**by** (*rule ChopAssoc*)  
**have** 11:  $\vdash g; \text{more} = \text{prev} (di g)$   
**using** 9 10 **by** (*metis di-d-def int-eq prev-d-def*)  
**have** 12:  $\vdash (f; \text{more} \wedge g; \text{more}) = (\text{prev} (di f) \wedge \text{prev} (di g))$   
**using** 8 11 **by** *fastforce*  
**hence** 13:  $\vdash (f; \text{more} \wedge g; \text{more}) = \text{prev} (di f \wedge di g)$   
**by** (*metis ChopSkipAndChopSkip int-eq prev-d-def*)  
**have** 14:  $\vdash (di f \wedge di g) =$   
 $((f \wedge g) \vee (f \wedge g; \text{more}) \vee (g \wedge f; \text{more})) \vee (f; \text{more} \wedge g; \text{more})$   
**using** 3 4 **by** *auto*  
**have** 15:  $\vdash (di f \wedge di g) =$   
 $((f \wedge g) \vee (f \wedge g; \text{more}) \vee (g \wedge f; \text{more})) \vee \text{prev} (di f \wedge di g)$   
**by** (*metis 10 14 6 7 9 ChopSkipAndChopSkip di-d-def int-eq prev-d-def*)  
**hence** 16:  $\vdash (di f \wedge di g) \longrightarrow$   
 $((f \wedge g) \vee (f \wedge g; \text{more}) \vee (g \wedge f; \text{more})) \vee \text{prev} (di f \wedge di g)$   
**by** *fastforce*  
**hence** 17:  $\vdash (di f \wedge di g) \wedge \neg((f \wedge g) \vee (f \wedge g; \text{more}) \vee (g \wedge f; \text{more})) \longrightarrow$   
 $\text{prev} (di f \wedge di g)$   
**by** *fastforce*  
**hence** 18:  $\vdash (di f \wedge di g) \longrightarrow di((f \wedge g) \vee (f \wedge g; \text{more}) \vee (g \wedge f; \text{more}))$   
**using** *DiIntroLoop* **by** *blast*  
**have** 19:  $\vdash di((f \wedge g) \vee (f \wedge g; \text{more}) \vee (g \wedge f; \text{more})) =$   
 $(di(f \wedge g) \vee di(f \wedge g; \text{more}) \vee di(g \wedge f; \text{more}))$   
**by** (*meson DiOrEqv Prop06*)  
**have** 20:  $\vdash f \longrightarrow di f$   
**using** *DiIntro* **by** *blast*  
**hence** 21:  $\vdash f \wedge g \longrightarrow g \wedge di f$   
**by** *auto*  
**hence** 22:  $\vdash di(f \wedge g) \longrightarrow di(g \wedge di f)$   
**using** *DiImpDi* **by** *blast*  
**hence** 23:  $\vdash di(f \wedge g) \longrightarrow di(g \wedge di f) \vee di(f \wedge di g)$   
**by** *auto*  
**have** 24:  $\vdash g; \text{more} \longrightarrow di g$   
**by** (*simp add: ChopImpDi*)  
**hence** 25:  $\vdash f \wedge g; \text{more} \longrightarrow f \wedge di g$   
**by** *auto*  
**hence** 26:  $\vdash di(f \wedge g; \text{more}) \longrightarrow di(f \wedge di g)$   
**using** *DiImpDi* **by** *blast*  
**hence** 27:  $\vdash di(f \wedge g; \text{more}) \longrightarrow di(f \wedge di g) \vee di(g \wedge di f)$

by *auto*  
 have 28:  $\vdash f; \text{more} \longrightarrow di\ f$   
 by (*simp add: ChopImpDi*)  
 hence 29:  $\vdash g \wedge f; \text{more} \longrightarrow g \wedge di\ f$   
 by *auto*  
 hence 30:  $\vdash di(g \wedge f; \text{more}) \longrightarrow di(g \wedge di\ f)$   
 using *DiImpDi* by *blast*  
 hence 31:  $\vdash di(g \wedge f; \text{more}) \longrightarrow di(f \wedge di\ g) \vee di(g \wedge di\ f)$   
 by *auto*  
 have 32:  $\vdash di(f \wedge g) \vee di(f \wedge g; \text{more}) \vee di(g \wedge f; \text{more}) \longrightarrow$   
 $di(f \wedge di\ g) \vee di(g \wedge di\ f)$   
 using 23 27 31 by *fastforce*  
 have 33:  $\vdash di((f \wedge g) \vee (f \wedge g; \text{more}) \vee (g \wedge f; \text{more})) \longrightarrow$   
 $di(f \wedge di\ g) \vee di(g \wedge di\ f)$   
 using 19 32 by *fastforce*  
 have 34:  $\vdash (di\ f \wedge di\ g) \longrightarrow di(f \wedge di\ g) \vee di(g \wedge di\ f)$   
 using 18 33 by *fastforce*  
 have 35:  $\vdash f \longrightarrow di\ f$   
 using *DiIntro* by *blast*  
 hence 36:  $\vdash f \wedge di\ g \longrightarrow di\ f \wedge di\ g$   
 by *auto*  
 hence 37:  $\vdash di(f \wedge di\ g) \longrightarrow di(di\ f \wedge di\ g)$   
 using *DiImpDi* by *blast*  
 have 38:  $\vdash di(di\ f \wedge di\ g) = (di\ f \wedge di\ g)$   
 using *DiDiAndEqvDi* by *blast*  
 have 39:  $\vdash di(f \wedge di\ g) \longrightarrow di\ f \wedge di\ g$   
 using 37 38 by *fastforce*  
 have 40:  $\vdash g \longrightarrow di\ g$   
 using *DiIntro* by *blast*  
 hence 41:  $\vdash g \wedge di\ f \longrightarrow di\ f \wedge di\ g$   
 by *auto*  
 hence 42:  $\vdash di(g \wedge di\ f) \longrightarrow di(di\ f \wedge di\ g)$   
 using *DiImpDi* by *blast*  
 have 43:  $\vdash di(di\ f \wedge di\ g) = (di\ f \wedge di\ g)$   
 using *DiDiAndEqvDi* by *fastforce*  
 have 44:  $\vdash di(g \wedge di\ f) \longrightarrow di\ f \wedge di\ g$   
 using 42 43 by *fastforce*  
 have 45:  $\vdash di(f \wedge di\ g) \vee di(g \wedge di\ f) \longrightarrow di\ f \wedge di\ g$   
 using 39 44 by *fastforce*  
 from 34 45 show ?thesis by *fastforce*  
 qed

lemma *BoxStateEqvBiFinState*:

$\vdash \square (init\ w) = bi\ (fin\ (init\ w))$

proof –

have 1:  $\vdash \Diamond (\neg (init\ w)) = \#True ; (\neg (init\ w))$

by (*simp add: sometimes-d-def*)

have 2:  $\vdash \Diamond (init(\neg w)) = \#True ; init(\neg w)$

by (*simp add: sometimes-d-def*)

have 3:  $\vdash di(\#True \wedge fin\ (init(\neg w))) = \#True ; init(\neg w)$

```

    using DiAndFinEqvChopState by blast
  have 4:  $\vdash \Diamond (init(\neg w)) = di (\#True \wedge fin (init (\neg w)))$ 
    using 1 2 3 by fastforce
  have 5:  $\vdash (\neg (\Diamond (init(\neg w)))) = (\neg (di (\#True \wedge fin (init (\neg w)))))$ 
    using 4 by fastforce
  have 6:  $\vdash \Box (init w) = (\neg (di (\#True \wedge fin (init (\neg w)))))$ 
    using 5 always-d-def Initprop(2) by (metis int-eq)
  have 7:  $\vdash \Box (init w) = bi (\neg (fin (init (\neg w))))$ 
    using 6 by (simp add: bi-d-def)
  have 8:  $\vdash init (\neg w) = (\neg (init w))$ 
    using Initprop(2) by fastforce
  have 9:  $\vdash fin (init (\neg w)) = fin (\neg (init w))$ 
    using 8 FinEqvFin by blast
  have 10:  $\vdash fin (init (\neg w)) = (\neg (fin (init w)))$ 
    using 8 FinNotStateEqvNotFinState FinEqvFin by blast
  have 11:  $\vdash (\neg (fin (init (\neg w)))) = (fin (init w))$ 
    using 10 by fastforce
  have 12:  $\vdash bi (\neg (fin (init (\neg w)))) = bi (fin (init w))$ 
    using 11 by (simp add: BiEqvBi)
  have 13:  $\vdash \Box (init w) = bi (fin (init w))$ 
    using 7 12 by fastforce
  from 13 show ?thesis by simp
qed

```

**lemma** *DiamondStateEqvDiFinState*:

$\vdash \Diamond (init w) = di (fin (init w))$

**proof** –

```

  have 1:  $\vdash \Box (init (\neg w)) = bi (fin (init (\neg w)))$ 
    using BoxStateEqvBiFinState by blast
  have 2:  $\vdash (\neg (\Box (init (\neg w)))) = (\neg (bi (fin (init (\neg w)))))$ 
    using 1 by auto
  have 3:  $\vdash \Diamond (\neg (init (\neg w))) = di (\neg (fin (init (\neg w))))$ 
    using 2 by (simp add: always-d-def bi-d-def)
  have 4:  $\vdash \Diamond (init w) = di (\neg (fin (init (\neg w))))$ 
    by (metis 3 DiEqvNotBiNot DiState Initprop(2) StateEqvBi int-eq)
  have 5:  $\vdash \Diamond (init w) = di (fin (init w))$  using 4 FinNotStateEqvNotFinState
    by (metis DiEqvNotBiNot DiNotEqvNotBi inteq-reflection)
  from 1 2 3 4 5 show ?thesis by simp
qed

```

**lemma** *OrDiEqvDi*:

$\vdash (f \vee di f) = di f$

**proof** –

```

  have 1:  $\vdash f \longrightarrow di f$  using DiIntro by blast
  from 1 show ?thesis by auto
qed

```

**lemma** *AndDiEqv*:

$\vdash (f \wedge di f) = f$

**proof** –



**have** 1:  $\vdash f \longrightarrow di\ f$  **using** *DiIntro* **by** *blast*  
**from** 1 **show** *?thesis* **by** *auto*  
**qed**

**lemma** *BiEmptyEqvEmpty*:

$\vdash bi\ empty = empty$

**proof** –

**have** 1:  $\vdash bi\ empty = (\neg (di\ (\neg empty)))$  **by** (*simp add: bi-d-def*)  
**have** 2:  $\vdash (\neg (di\ (\neg empty))) = (\neg ((\neg empty); \# True))$  **by** (*simp add: di-d-def*)  
**have** 3:  $\vdash (\neg ((\neg empty); \# True)) = (\neg (more; \# True))$  **by** (*simp add: empty-d-def*)  
**have** 4:  $\vdash more; \# True = more$  **using** *MoreEqvMoreChopTrue* **by** *auto*  
**hence** 5:  $\vdash (\neg (more; \# True)) = (\neg more)$  **by** *fastforce*  
**from** 1 2 3 5 **show** *?thesis* **using** *NotEmptyEqvMore* **by** *fastforce*  
**qed**

**lemma** *EmptyChopSkipInduct*:

**assumes**  $\vdash empty \longrightarrow f$

$\vdash prev\ f \longrightarrow f$

**shows**  $\vdash f$

**proof** –

**have** 1:  $\vdash empty \longrightarrow f$  **using** *assms(1)* **by** *auto*  
**have** 2:  $\vdash prev\ f \longrightarrow f$  **using** *assms(2)* **by** *blast*  
**have** 3:  $\vdash (empty \vee prev\ f) \longrightarrow f$  **using** 1 2 **by** *fastforce*  
**have** 4:  $\vdash wprev\ f = (empty \vee prev\ f)$  **by** (*simp add: WprevEqvEmptyOrPrev*)  
**hence** 5:  $\vdash wprev\ f \longrightarrow f$  **using** 3 **by** *fastforce*  
**hence** 6:  $\vdash \neg f \longrightarrow \neg (wprev\ f)$  **by** *fastforce*  
**hence** 7:  $\vdash \neg f \longrightarrow prev\ (\neg f)$  **by** (*simp add: wprev-d-def*)  
**hence** 8:  $\vdash \neg \neg f$  **by** (*rule PrevLoop*)  
**from** 8 **show** *?thesis* **by** *auto*  
**qed**

**lemma** *MoreImpImpChopSkipEqv*:

$\vdash more \longrightarrow ( (f \longrightarrow g); skip = ((f; skip) \longrightarrow (g; skip)) )$

**proof** –

**have** 01:  $\vdash (f \longrightarrow g) = (\neg f \vee g)$  **by** *auto*  
**hence** 02:  $\vdash (f \longrightarrow g); skip = (\neg f \vee g); skip$  **by** (*simp add: LeftChopEqvChop*)  
**hence** 1:  $\vdash (more \wedge (f \longrightarrow g); skip) = (more \wedge (\neg f \vee g); skip)$  **by** *fastforce*  
**have** 2:  $\vdash (\neg f \vee g); skip = ((\neg f); skip \vee g; skip)$   
**using** *OrChopEqv* **by** *auto*  
**hence** 3:  $\vdash (more \wedge (\neg f \vee g); skip) = (more \wedge ((\neg f); skip \vee g; skip))$   
**by** *auto*  
**have** 4:  $\vdash (\neg ((\neg f); skip)) = (empty \vee (f; skip))$   
**using** *NotNotChopSkip* **by** *blast*  
**hence** 5:  $\vdash ((\neg f); skip) = (\neg (empty \vee (f; skip)))$   
**by** *fastforce*  
**have** 6:  $\vdash \neg (empty \vee (f; skip)) = (more \wedge \neg (f; skip))$   
**using** 5 *NotChopSkipEqvMoreAndNotChopSkip* **by** *fastforce*  
**have** 7:  $\vdash ((\neg f); skip \vee g; skip) = ( (more \wedge \neg (f; skip)) \vee g; skip)$   
**using** 5 6 **by** *fastforce*  
**hence** 8:  $\vdash (more \wedge (\neg f; skip \vee g; skip)) = (more \wedge ( (more \wedge \neg (f; skip)) \vee g; skip))$

by auto  
 have 9:  $\vdash (more \wedge ( (more \wedge \neg(f;skip)) \vee g;skip)) = (more \wedge (\neg(f;skip) \vee g;skip))$   
 by auto  
 have 10:  $\vdash (more \wedge (\neg(f;skip) \vee g;skip)) = (more \wedge ((f;skip) \longrightarrow (g;skip)))$   
 by auto  
 have 11:  $\vdash (more \wedge (f \longrightarrow g);skip) = (more \wedge ((f;skip) \longrightarrow (g;skip)))$   
 using 1 2 3 8 9 10 7 by fastforce  
 from 11 show ?thesis using MP by fastforce  
 qed

lemma MoreImpImpPrevEqv:  
 $\vdash more \longrightarrow ( prev(f \longrightarrow g) = (prev f \longrightarrow prev g) )$   
 by (simp add: MoreImpImpChopSkipEqv prev-d-def)

lemma BiBoxNotEqvNotTrueChopChopTrue:  
 $\vdash bi(\Box (\neg f)) = (\neg((\# True;f); \# True ))$   
 by (simp add: bi-d-def always-d-def di-d-def sometimes-d-def)

lemma DiAndEmptyEqvAndEmpty:  
 $\vdash (di f \wedge empty) = (f \wedge empty)$   
 proof –  
 have 1 :  $\vdash di f = (f \vee di f;skip)$   
 using DiEqvOrDiChopSkipB by blast  
 hence 2:  $\vdash (di f \wedge empty) = ((f \vee di f;skip) \wedge empty)$   
 by fastforce  
 have 3 :  $\vdash ((f \vee di f;skip) \wedge empty) = ((f \wedge empty) \vee (di f;skip \wedge empty))$   
 by auto  
 have 4:  $\vdash \neg(di f;skip \wedge empty)$   
 by (metis (no-types, hide-lams) ChopAssoc NotChopAndMoreAndEmpty TrueChopSkipEqvSkipChopTrue  
 di-d-def int-simps(17) inteq-reflection more-d-def next-d-def)  
 hence 5 :  $\vdash ((f \wedge empty) \vee (di f;skip \wedge empty)) = (f \wedge empty)$   
 by auto  
 from 2 3 5 show ?thesis by fastforce  
 qed

### 9.4.3 Strict initial intervals

lemma DsMoreDi:  
 $\vdash ds f = (more \wedge (di f);skip)$   
 proof –  
 have 1:  $\vdash ds f = (\neg(bs (\neg f)))$   
 by (simp add: ds-d-def)  
 have 2:  $\vdash (\neg(bs (\neg f))) = (\neg(empty \vee (bi (\neg f));skip))$   
 by (simp add: bs-d-def)  
 have 3:  $\vdash (\neg(empty \vee (bi (\neg f));skip)) = (\neg empty \wedge \neg((bi (\neg f));skip))$   
 by auto  
 have 4:  $\vdash (\neg empty \wedge \neg((bi (\neg f));skip)) = (more \wedge \neg((bi (\neg f));skip))$   
 using NotEmptyEqvMore by auto

**have** 5:  $\vdash (more \wedge \neg((bi \ (\neg f));skip)) = (more \wedge \neg(\neg(di \ f);skip))$   
**by** (*metis DiEqvNotBiNot DiIntro DiSkipEqvMore NotChopSkipEqvMoreAndNotChopSkip Prop10 RightChopImpMoreRule int-simps(4) inteq-reflection lift-and-com*)  
**have** 6:  $\vdash (more \wedge \neg(\neg(di \ f));skip)) = (more \wedge (empty \vee (di \ f);skip))$   
**using** *NotNotChopSkip* **by** *fastforce*  
**have** 7:  $\vdash (more \wedge (empty \vee (di \ f);skip)) = (more \wedge (di \ f);skip)$   
**using** *NotEmptyEqvMore* **by** *auto*  
**from** 1 2 3 4 5 6 7 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *DsDi*:

$\vdash ds \ f = (di \ f);skip$

**proof** –

**have** 1:  $\vdash ds \ f = (more \wedge (di \ f);skip)$  **by** (*rule DsMoreDi*)  
**have** 2:  $\vdash (di \ f);skip \longrightarrow more$  **by** (*metis DiIntro DiSkipEqvMore RightChopImpMoreRule int-eq*)  
**hence** 3:  $\vdash (more \wedge (di \ f);skip) = (di \ f);skip$  **by** *auto*  
**from** 1 2 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *BsEqvNotDsNot*:

$\vdash bs \ f = (\neg(ds \ (\neg f)))$

**proof** –

**have** 1:  $\vdash ds \ (\neg f) = (more \wedge (di \ (\neg f));skip)$   
**by** (*rule DsMoreDi*)  
**hence** 2:  $\vdash (\neg(ds \ (\neg f))) = (\neg(more \wedge (di \ (\neg f));skip))$   
**by** *auto*  
**have** 3:  $\vdash (\neg(more \wedge (di \ (\neg f));skip)) = (empty \vee \neg((di \ (\neg f));skip))$   
**using** *NotEmptyEqvMore* **by** *auto*  
**have** 4:  $\vdash (empty \vee \neg((di \ (\neg f));skip)) = (empty \vee \neg(\neg(bi \ f));skip))$   
**using** *DiNotEqvNotBi* **by** (*metis 3 inteq-reflection*)  
**have** 5:  $\vdash (\neg(\neg(bi \ f));skip) = (empty \vee (bi \ f);skip)$   
**by** (*rule NotNotChopSkip*)  
**hence** 6:  $\vdash (empty \vee \neg(\neg(bi \ f));skip) = (empty \vee (bi \ f);skip)$   
**by** *auto*  
**from** 2 3 4 6 **show** *?thesis* **by** (*metis bs-d-def inteq-reflection*)  
**qed**

**lemma** *NotBsEqvDsNot*:

$\vdash (\neg(bs \ f)) = ds \ (\neg f)$

**proof** –

**have** 1:  $\vdash bs \ f = (\neg(ds \ (\neg f)))$  **by** (*rule BsEqvNotDsNot*)  
**hence** 2:  $\vdash (\neg(bs \ f)) = (\neg\neg(ds \ (\neg f)))$  **by** *auto*  
**from** 2 **show** *?thesis* **by** *auto*  
**qed**

**lemma** *NotDsEqvBsNot*:

$\vdash (\neg(ds \ f)) = bs \ (\neg f)$

**proof** –

**have** 1:  $\vdash (\neg(ds \ f)) = (\neg\neg(bs \ (\neg f)))$  **by** (*simp add: ds-d-def*)  
**from** 1 **show** *?thesis* **by** *auto*

qed

**lemma** *NotDsAndEmpty*:

$\vdash \neg(ds\ f \wedge empty)$

**proof** –

**have** 1:  $\vdash ds\ f = (more \wedge (di\ f);skip)$  **by** (*rule DsMoreDi*)

**have** 2:  $\vdash more \wedge (di\ f);skip \wedge empty \longrightarrow \#False$  **using** *NotEmptyEqvMore* **by** *auto*

**from** 1 2 **show** *?thesis* **by** *fastforce*

qed

**lemma** *BsMoreEqvEmpty*:

$\vdash bs\ more = empty$

**proof** –

**have** 1:  $\vdash bs\ more = (empty \vee (bi\ more);skip)$  **by** (*simp add: bs-d-def*)

**have** 2:  $\vdash bi\ more \longrightarrow \#False$  **using** *DiEmpty NotEmptyEqvMore* **by** (*simp add: bi-d-def empty-d-def*)

**hence** 3:  $\vdash (bi\ more);skip \longrightarrow \#False;skip$  **using** *LeftChopImpChop* **by** *blast*

**have** 31:  $\vdash \#False;skip \longrightarrow \#False$  **by** (*simp add: Valid-def skip-defs chop-defs*)

**have** 32:  $\vdash (bi\ more);skip \longrightarrow \#False$  **using** 3 31 **by** *fastforce*

**hence** 4:  $\vdash (empty \vee ((bi\ more);skip)) = empty$  **by** *fastforce*

**from** 1 4 **show** *?thesis* **by** *fastforce*

qed

**lemma** *BsAndEqv*:

$\vdash (bs\ f \wedge bs\ g) = bs(f \wedge g)$

**proof** –

**have** 1:  $\vdash bs\ f = (empty \vee (bi\ f);skip)$

**by** (*simp add: bs-d-def*)

**have** 2:  $\vdash bs\ g = (empty \vee (bi\ g);skip)$

**by** (*simp add: bs-d-def*)

**have** 3:  $\vdash (bs\ f \wedge bs\ g) = ((empty \vee (bi\ f);skip) \wedge (empty \vee (bi\ g);skip))$

**using** 1 2 **by** *fastforce*

**have** 4:  $\vdash ((empty \vee (bi\ f);skip) \wedge (empty \vee (bi\ g);skip)) =$

$(empty \vee ((bi\ f);skip \wedge (bi\ g);skip))$

**by** *auto*

**have** 5:  $\vdash (((bi\ f);skip \wedge (bi\ g);skip)) = bi(f \wedge g);skip$

**using** *BiAndChopSkipEqv* **by** *fastforce*

**hence** 6:  $\vdash (empty \vee ((bi\ f);skip \wedge (bi\ g);skip)) = (empty \vee bi(f \wedge g);skip)$

**by** *auto*

**from** 3 4 6 **show** *?thesis* **by** (*metis bs-d-def inteq-reflection*)

qed

**lemma** *DsEqvRule*:

**assumes**  $\vdash f = g$

**shows**  $\vdash ds\ f = ds\ g$

**using** *assms* **using** *int-eq* **by** *force*

**lemma** *DsOrEqv*:

$\vdash (ds\ f \vee ds\ g) = ds(f \vee g)$

**proof** –

**have** 1:  $\vdash ds\ f = (\neg(bs\ (\neg f)))$  **by** (*simp add: ds-d-def*)

**have** 2:  $\vdash ds\ g = (\neg(bs\ (\neg\ g)))$  **by** (*simp add: ds-d-def*)  
**have** 3:  $\vdash (ds\ f \vee ds\ g) = (\neg(bs\ (\neg\ f)) \vee \neg(bs\ (\neg\ g)))$  **using** 1 2 **by** *fastforce*  
**have** 4:  $\vdash (\neg(bs\ (\neg\ f)) \vee \neg(bs\ (\neg\ g))) = (\neg(bs\ (\neg\ f) \wedge bs\ (\neg\ g)))$  **by** *auto*  
**have** 5:  $\vdash (bs\ (\neg\ f) \wedge bs\ (\neg\ g)) = bs\ (\neg\ f \wedge \neg\ g)$  **by** (*rule BsAndEqv*)  
**hence** 6:  $\vdash (\neg(bs\ (\neg\ f) \wedge bs\ (\neg\ g))) = (\neg(bs\ (\neg\ f \wedge \neg\ g)))$  **by** *auto*  
**have** 7:  $\vdash (\neg(bs\ (\neg\ f \wedge \neg\ g))) = ds\ (\neg\ (\neg\ f \wedge \neg\ g))$  **by** (*rule NotBsEqvDsNot*)  
**have** 8:  $\vdash (\neg(\neg\ f \wedge \neg\ g)) = (f \vee g)$  **by** *auto*  
**hence** 9:  $\vdash ds(\neg(\neg\ f \wedge \neg\ g)) = ds\ (f \vee g)$  **by** (*rule DsEqvRule*)  
**from** 3 4 6 7 9 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *BsOrImp*:

$\vdash bs\ f \vee bs\ g \longrightarrow bs(f \vee g)$

**proof** –

**have** 1:  $\vdash bi\ f \vee bi\ g \longrightarrow bi(f \vee g)$

**by** (*rule BiOrBiImpBiOr*)

**hence** 2:  $\vdash (bi\ f \vee bi\ g);skip \longrightarrow (bi(f \vee g));skip$

**by** (*rule LeftChopImpChop*)

**have** 3:  $\vdash (bi\ f);skip \vee (bi\ g);skip \longrightarrow (bi(f \vee g));skip$

**using** 1 *OrChopEqv* 2 **by** *fastforce*

**hence** 4:  $\vdash empty \vee (bi\ f);skip \vee (bi\ g);skip \longrightarrow empty \vee (bi(f \vee g));skip$

**by** *auto*

**hence** 5:  $\vdash (empty \vee (bi\ f);skip) \vee (empty \vee (bi\ g);skip) \longrightarrow empty \vee (bi(f \vee g));skip$

**by** *auto*

**from** 5 **show** *?thesis* **by** (*simp add: bs-d-def*)

**qed**

**lemma** *DsAndImp*:

$\vdash ds\ (f \wedge g) \longrightarrow ds\ f \wedge ds\ g$

**proof** –

**have** 1:  $\vdash bs\ (\neg\ f) \vee bs\ (\neg\ g) \longrightarrow bs(\neg\ f \vee \neg\ g)$  **by** (*rule BsOrImp*)

**have** 2:  $\vdash (\neg\ f \vee \neg\ g) = (\neg(f \wedge g))$  **by** *auto*

**hence** 3:  $\vdash bs(\neg\ f \vee \neg\ g) = bs\ (\neg(f \wedge g))$  **by** (*rule BsEqvRule*)

**have** 4:  $\vdash bs\ (\neg\ f) \vee bs\ (\neg\ g) \longrightarrow bs\ (\neg(f \wedge g))$  **using** 1 3 **by** *fastforce*

**have** 5:  $\vdash bs\ (\neg\ f) = (\neg(ds\ f))$  **using** *NotDsEqvBsNot* **by** *fastforce*

**have** 6:  $\vdash bs\ (\neg\ g) = (\neg(ds\ g))$  **using** *NotDsEqvBsNot* **by** *fastforce*

**have** 7:  $\vdash bs\ (\neg(f \wedge g)) = (\neg(ds\ (f \wedge g)))$  **using** *NotDsEqvBsNot* **by** *fastforce*

**have** 8:  $\vdash \neg(ds\ f) \vee \neg(ds\ g) \longrightarrow \neg(ds\ (f \wedge g))$  **using** 4 5 6 7 **by** *fastforce*

**hence** 9:  $\vdash \neg(ds\ f \wedge ds\ g) \longrightarrow \neg(ds\ (f \wedge g))$  **by** *auto*

**from** 9 **show** *?thesis* **by** *auto*

**qed**

**lemma** *DsAndImpElimL*:

$\vdash ds\ (f \wedge g) \longrightarrow ds\ f$

**using** *DsAndImp* **by** *fastforce*

**lemma** *DsAndImpElimR*:

$\vdash ds\ (f \wedge g) \longrightarrow ds\ g$

**using** *DsAndImp* **by** *fastforce*

**lemma** *BiImpBs*:

$\vdash bi\ f \longrightarrow bs\ f$

**proof** –

**have** 1:  $\vdash empty \longrightarrow empty \vee (bi\ f);skip$  **by** *auto*

**hence** 2:  $\vdash empty \wedge bi\ f \longrightarrow empty \vee (bi\ f);skip$  **by** *auto*

**have** 2:  $\vdash more \wedge bi\ f \longrightarrow (bi\ f);skip$  **by** (*rule MoreAndBiImpBiChopSkip*)

**hence** 3:  $\vdash more \wedge bi\ f \longrightarrow empty \vee (bi\ f);skip$  **by** *auto*

**have** 4:  $\vdash bi\ f = ((bi\ f \wedge empty) \vee (bi\ f \wedge more))$  **by** (*auto simp add: empty-d-def*)

**have** 5:  $\vdash (empty \vee (bi\ f);skip) = bs\ f$  **by** (*simp add: bs-d-def*)

**from** 2 3 4 5 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *BsImpBsBs*:

$\vdash bs\ f \longrightarrow bs\ (bs\ f)$

**proof** –

**have** 1:  $\vdash bi\ f \longrightarrow bs\ f$  **by** (*rule BiImpBs*)

**hence** 2:  $\vdash bi\ (bi\ f) \longrightarrow bi\ (bs\ f)$  **by** (*rule BiImpBiRule*)

**hence** 3:  $\vdash (bi\ f) \longrightarrow bi\ (bs\ f)$  **using** *BiEqvBiBi* **by** *fastforce*

**hence** 4:  $\vdash (bi\ f);skip \longrightarrow (bi\ (bs\ f));skip$  **by** (*rule LeftChopImpChop*)

**hence** 5:  $\vdash empty \vee (bi\ f);skip \longrightarrow empty \vee (bi\ (bs\ f));skip$  **by** *auto*

**from** 5 **show** *?thesis* **by** (*simp add: bs-d-def*)

**qed**

**lemma** *DsImpDi*:

$\vdash ds\ f \longrightarrow di\ f$

**proof** –

**have** 1:  $\vdash bi\ (\neg f) \longrightarrow bs\ (\neg f)$  **by** (*rule BiImpBs*)

**hence** 2:  $\vdash \neg(bs\ (\neg f)) \longrightarrow \neg(bi\ (\neg f))$  **by** *fastforce*

**from** 2 **show** *?thesis* **using** *NotBsEqvDsNot DiEqvNotBiNot* **by** *fastforce*

**qed**

**lemma** *BsImpBsRule*:

**assumes**  $\vdash f \longrightarrow g$

**shows**  $\vdash bs\ f \longrightarrow bs\ g$

**proof** –

**have** 1:  $\vdash f \longrightarrow g$  **using** *assms* **by** *auto*

**hence** 2:  $\vdash bi\ f \longrightarrow bi\ g$  **by** (*rule BiImpBiRule*)

**hence** 3:  $\vdash (bi\ f);skip \longrightarrow (bi\ g);skip$  **by** (*rule LeftChopImpChop*)

**hence** 4:  $\vdash empty \vee (bi\ f);skip \longrightarrow empty \vee (bi\ g);skip$  **by** *auto*

**from** 4 **show** *?thesis* **by** (*simp add: bs-d-def*)

**qed**

**lemma** *DsChopImpDsB*:

$\vdash ds\ (f;g) \longrightarrow ds\ f$

**proof** –

**have** 1:  $\vdash di\ (f;g) \longrightarrow di\ f$  **by** (*rule DiChopImpDiB*)

**hence** 2:  $\vdash (di\ (f;g));skip \longrightarrow (di\ f);skip$  **by** (*rule LeftChopImpChop*)

**from** 2 **show** *?thesis* **using** *DsDi* **by** *fastforce*

**qed**

**lemma** *NotBsImpBsNotChop*:  
 $\vdash bs (\neg f) \longrightarrow bs (\neg(f;g))$   
**proof** –  
**have** 1:  $\vdash ds (f;g) \longrightarrow ds f$  **by** (rule *DsChopImpDsB*)  
**hence** 2:  $\vdash \neg(ds f) \longrightarrow \neg(ds (f;g))$  **by** *fastforce*  
**from** 2 **show** *?thesis* **using** *NotDsEqvBsNot* **by** *fastforce*  
**qed**

**lemma** *BsOrBsEqvBsBiOrBi*:  
 $\vdash (bs f \vee bs g) = bs(bi f \vee bi g)$   
**proof** –  
**have** 1:  $\vdash (bs f \vee bs g) = ((empty \vee (bi f);skip) \vee (empty \vee (bi g);skip))$   
**by** (*simp add: bs-d-def*)  
**have** 2:  $\vdash ((empty \vee (bi f);skip) \vee (empty \vee (bi g);skip)) = (empty \vee (bi f);skip \vee (bi g);skip)$   
**by** *auto*  
**have** 3:  $\vdash ((bi f);skip \vee (bi g);skip) = (bi f \vee bi g);skip$   
**using** *OrChopEqv* **by** *fastforce*  
**hence** 4:  $\vdash (empty \vee (bi f);skip \vee (bi g);skip) = (empty \vee (bi f \vee bi g);skip)$   
**by** *auto*  
**have** 5:  $\vdash (bi f \vee bi g) = bi (bi f \vee bi g)$   
**by** (*meson BiBiOrImpBi BiImpBiBiOr BiImpBiBiOrB Prop02 int-iffI*)  
**hence** 6:  $\vdash (bi f \vee bi g);skip = bi (bi f \vee bi g);skip$   
**by** (*simp add: LeftChopEqvChop*)  
**hence** 7:  $\vdash (empty \vee bi (bi f \vee bi g);skip) = (empty \vee (bi f \vee bi g);skip)$   
**by** *auto*  
**have** 8:  $\vdash (empty \vee (bi f \vee bi g);skip) = bs(bi f \vee bi g)$  **using** *bs-d-def*  
**by** (*metis 4 5 inteq-reflection*)  
**from** 1 2 4 8 **show** *?thesis* **by** (*metis inteq-reflection*)  
**qed**

**lemma** *DiOrDsEqvDi*:  
 $\vdash di f \vee ds f = di f$   
**proof** –  
**have** 1:  $\vdash di f \longrightarrow di f \vee ds f$  **by** *auto*  
**have** 2:  $\vdash di f \longrightarrow di f$  **by** *auto*  
**have** 3:  $\vdash ds f \longrightarrow di f$  **by** (rule *DsImpDi*)  
**have** 4:  $\vdash di f \vee ds f \longrightarrow di f$  **using** 2 3 **by** *auto*  
**from** 1 4 **show** *?thesis* **by** *auto*  
**qed**

**lemma** *DiAndDsEqvDs*:  
 $\vdash (di f \wedge ds f) = ds f$   
**proof** –  
**have** 1:  $\vdash di f \wedge ds f \longrightarrow ds f$  **by** *auto*  
**have** 2:  $\vdash ds f \longrightarrow ds f$  **by** *auto*  
**have** 3:  $\vdash ds f \longrightarrow di f$  **by** (rule *DsImpDi*)  
**have** 4:  $\vdash ds f \longrightarrow di f \wedge ds f$  **using** 2 3 **by** *auto*  
**from** 1 4 **show** *?thesis* **by** *auto*  
**qed**

**lemma** *OrDsEqvDi*:

$\vdash (f \vee ds\ f) = di\ f$

**proof** –

**have** 1:  $\vdash ds\ f = (di\ f);skip$  **by** (rule *DsDi*)

**hence** 2:  $\vdash (f \vee ds\ f) = (f \vee (di\ f);skip)$  **by** *auto*

**from** 2 **show** *?thesis* **using** *DiEqvOrDiChopSkipB[of f]* **by** *fastforce*

**qed**

**lemma** *AndBsEqvBi*:

$\vdash (f \wedge bs\ f) = bi\ f$

**proof** –

**have** 1:  $\vdash (f \wedge bs\ f) = (f \wedge (empty \vee (bi\ f);skip))$  **by** (*simp add: bs-d-def*)

**from** 1 **show** *?thesis*

**by** (*meson BiEqvAndEmptyOrBiChopSkip Prop04 int-simps(4)*)

**qed**

**lemma** *BsEqvBsBi*:

$\vdash bs\ f = bs\ (bi\ f)$

**proof** –

**have** 1:  $\vdash bs\ f = (empty \vee (bi\ f);skip)$  **by** (*simp add: bs-d-def*)

**have** 2:  $\vdash bi\ f = bi\ (bi\ f)$  **by** (rule *BiEqvBiBi*)

**hence** 3:  $\vdash (bi\ f);skip = bi\ (bi\ f);skip$  **using** *LeftChopEqvChop* **by** *blast*

**hence** 4:  $\vdash (empty \vee (bi\ f);skip) = (empty \vee bi\ (bi\ f);skip)$  **by** *auto*

**from** 1 4 **show** *?thesis* **by** (*simp add: bs-d-def*)

**qed**

**lemma** *StateImpBs*:

$\vdash init\ w \longrightarrow bs\ (init\ w)$

**proof** –

**have** 1:  $\vdash init\ w = bi\ (init\ w)$  **by** (rule *StateEqvBi*)

**have** 2:  $\vdash bi\ (init\ w) \longrightarrow bs\ (init\ w)$  **by** (rule *BiImpBs*)

**from** 1 2 **show** *?thesis* **using** *StateImpBi* **by** *fastforce*

**qed**

**lemma** *DsAndDsEqvDsAndDiOrDsAndDi*:

$\vdash (ds\ f \wedge ds\ g) = (ds\ (f \wedge di\ g) \vee ds\ (g \wedge di\ f))$

**proof** –

**have** 1:  $\vdash (di\ f \wedge di\ g) = (di\ (f \wedge di\ g) \vee di\ (g \wedge di\ f))$

**by** (rule *DiAndDiEqvDiAndDiOrDiAndDi*)

**hence** 2:  $\vdash (di\ f \wedge di\ g);skip = (di\ (f \wedge di\ g) \vee di\ (g \wedge di\ f));skip$

**by** (rule *LeftChopEqvChop*)

**have** 3:  $\vdash (di\ f \wedge di\ g);skip = ((di\ f);skip \wedge (di\ g);skip)$

**using** *ChopSkipAndChopSkip* **by** *fastforce*

**have** 4:  $\vdash ((di\ f);skip \wedge (di\ g);skip) = (di\ (f \wedge di\ g) \vee di\ (g \wedge di\ f));skip$

**using** 2 3 **by** *fastforce*

**have** 5:  $\vdash (di\ (f \wedge di\ g) \vee di\ (g \wedge di\ f));skip = (di\ (f \wedge di\ g);skip \vee di\ (g \wedge di\ f);skip)$

**using** *OrChopEqv* **by** *blast*

**have** 6:  $\vdash ds\ f = (di\ f);skip$

**using** *DsDi* **by** *blast*



```

have 7: ⊢ ds g = (di g);skip
  using DsDi by blast
have 8: ⊢ ((di f);skip ∧ (di g);skip) = (ds f ∧ ds g)
  using 6 7 by fastforce
have 9: ⊢ ds(f ∧ di g) = di(f ∧ di g);skip
  using DsDi by blast
have 10: ⊢ ds(g ∧ di f) = di(g ∧ di f);skip
  using DsDi by blast
have 11: ⊢ (di(f ∧ di g);skip ∨ di(g ∧ di f);skip) = (ds(f ∧ di g) ∨ ds(g ∧ di f))
  using 9 10 by fastforce
from 4 5 8 11 show ?thesis by (meson Prop04)
qed

```

**lemma** *BsEqvBiMoreImpChop*:

⊢ bs f = bi(more → f;skip)

**proof** –

```

have 1: ⊢ bs f = (empty ∨ (bi f;skip))
  by (simp add: bs-d-def)
have 2: ⊢ (empty ∨ (bi f;skip)) = ((¬(¬(bi f)));skip)
  using NotNotChopSkip by fastforce
have 3: ⊢ ¬((¬(bi f));skip) = (¬(di (¬ f);skip))
  by (simp add: bi-d-def)
have 4: ⊢ (¬(di (¬ f);skip)) = (¬(((¬ f) ;# True);skip))
  by (simp add: di-d-def)
have 5: ⊢ (¬(((¬ f) ;# True);skip)) = (¬((¬ f) ;(# True;skip)))
  using ChopAssocB by fastforce
have 6: ⊢ (¬((¬ f) ;(# True;skip))) = (¬((¬ f) ;(skip;# True)))
  by (metis 5 SkipTrueEqvTrueSkip int-eq)
have 7: ⊢ (¬((¬ f) ;(skip;# True))) = (¬(((¬ f) ;skip);# True))
  using ChopAssoc by fastforce
have 8: ⊢ (¬(((¬ f) ;skip);# True)) = (¬(di ((¬ f);skip)))
  by (simp add: di-d-def)
have 9: ⊢ (¬(di ((¬ f);skip))) = bi (¬((¬ f) ;skip))
  using NotDiEqvBiNot by blast
have 10: ⊢ bi (¬((¬ f) ;skip)) = bi( empty ∨ (f;skip))
  using NotNotChopSkip using BiEqvBi by blast
have 11: ⊢ bi( empty ∨ (f;skip)) = bi( ¬ more ∨ (f;skip))
  by (simp add: empty-d-def)
have 12: ⊢ ( ¬ more ∨ (f;skip)) = (more → f;skip) by auto
have 13: ⊢ bi( ¬ more ∨ (f;skip)) = bi(more → f;skip) using 12 using BiEqvBi by blast
have 14: ⊢ bs f = (¬ (((¬ f);skip);# True)) using 1 2 3 4 5 6 7 by fastforce
have 15: ⊢ (¬ (((¬ f);skip);# True)) = bi(more → f;skip) using 8 9 10 11 13 by fastforce
from 14 15 show ?thesis by fastforce
qed

```

**lemma** *BoxMoreStateEqvBsFinState*:

⊢ □(more → ¬ (init w)) = bs(¬(fin(init w)))

**proof** –

```

have 1: ⊢ □(more → ¬ (init w)) = (¬(◇(¬(more → ¬ (init w)))))
  by (simp add: always-d-def)

```

**have** 01:  $\vdash (\neg(\text{more} \longrightarrow \neg(\text{init } w))) = (\text{init } w \wedge \text{more})$  **by** *auto*  
**hence** 2:  $\vdash \neg(\Diamond(\neg(\text{more} \longrightarrow \neg(\text{init } w)))) = (\neg(\# \text{True}; (\text{init } w \wedge \text{more})))$   
**by** (*metis int-eq int-iffD1 int-simps(14) int-simps(6) sometimes-d-def*)  
**have** 3:  $\vdash \text{more} = \# \text{True}; \text{skip}$   
**using** *MoreEqvSkipChopTrue SkipTrueEqvTrueSkip* **by** *fastforce*  
**have** 4:  $\vdash (\text{init } w \wedge \text{more}) = (\text{init } w \wedge (\# \text{True}; \text{skip}))$   
**using** 3 **by** *auto*  
**have** 5:  $\vdash (\text{init } w \wedge (\# \text{True}; \text{skip})) = ((\text{init } w \wedge \text{empty}); (\# \text{True}; \text{skip}))$   
**using** *StateAndEmptyChop* **by** *fastforce*  
**have** 6:  $\vdash (\text{init } w \wedge \text{more}) = ((\text{init } w \wedge \text{empty}); (\# \text{True}; \text{skip}))$   
**using** 4 5 **by** *fastforce*  
**have** 7:  $\vdash (\# \text{True}; (\text{init } w \wedge \text{more})) = (\# \text{True}; ((\text{init } w \wedge \text{empty}); (\# \text{True}; \text{skip})))$   
**using** 6 *RightChopEqvChop* **by** *blast*  
**have** 8:  $\vdash (\# \text{True}; ((\text{init } w \wedge \text{empty}); (\# \text{True}; \text{skip}))) =$   
 $((\# \text{True}; (\text{init } w \wedge \text{empty})); (\# \text{True}; \text{skip}))$   
**using** *ChopAssoc* **by** *blast*  
**have** 9:  $\vdash (((\# \text{True}; (\text{init } w \wedge \text{empty})); (\# \text{True}; \text{skip}))) =$   
 $((((\# \text{True}; (\text{init } w \wedge \text{empty})); \# \text{True}); \text{skip}))$   
**using** *ChopAssoc* **by** *blast*  
**have** 10:  $\vdash (\# \text{True}; (\text{init } w \wedge \text{more})) =$   
 $((((\# \text{True}; (\text{init } w \wedge \text{empty})); \# \text{True}); \text{skip}))$   
**using** 7 8 9 **by** *fastforce*  
**hence** 11:  $\vdash (\neg(\# \text{True}; (\text{init } w \wedge \text{more}))) =$   
 $(\neg(((\# \text{True}; (\text{init } w \wedge \text{empty})); \# \text{True}); \text{skip})))$   
**by** *auto*  
**have** 12:  $\vdash \neg(((\# \text{True}; (\text{init } w \wedge \text{empty})); \# \text{True}); \text{skip})) =$   
 $\text{empty} \vee (\neg((\# \text{True}; (\text{init } w \wedge \text{empty})); \# \text{True}); \text{skip}))$   
**using** *NotChopNotSkip* **by** *fastforce*  
**have** 13:  $\vdash (\neg((\# \text{True}; (\text{init } w \wedge \text{empty})); \# \text{True})) = \text{bi}(\Box(\neg(\text{init } w \wedge \text{empty})))$   
**using** *BiBoxNotEqvNotTrueChopChopTrue* **by** *fastforce*  
**hence** 14:  $\vdash (\neg((\# \text{True}; (\text{init } w \wedge \text{empty})); \# \text{True})); \text{skip} =$   
 $(\text{bi}(\Box(\neg(\text{init } w \wedge \text{empty})))); \text{skip}$   
**using** *RightChopEqvChop* **by** (*simp add: LeftChopEqvChop*)  
**hence** 15:  $\vdash \text{empty} \vee (\neg((\# \text{True}; (\text{init } w \wedge \text{empty})); \# \text{True})); \text{skip} =$   
 $\text{empty} \vee (\text{bi}(\Box(\neg(\text{init } w \wedge \text{empty})))); \text{skip}$   
**by** *auto*  
**have** 16:  $\vdash (\neg(((\# \text{True}; (\text{init } w \wedge \text{empty})); \# \text{True}); \text{skip}))) =$   
 $(\text{empty} \vee (\text{bi}(\Box(\neg(\text{init } w \wedge \text{empty})))); \text{skip}))$   
**using** 12 15 **using** 14 *NotChopNotSkip int-eq* **by** *fastforce*  
**have** 171:  $\vdash (\neg(\text{init } w \wedge \text{empty})) = (\neg(\text{init } w) \vee \neg \text{empty})$   
**by** *auto*  
**hence** 172:  $\vdash \Box(\neg(\text{init } w \wedge \text{empty})) = \Box(\neg(\text{init } w) \vee \neg \text{empty})$   
**by** (*simp add: BoxEqvBox*)  
**hence** 173:  $\vdash \text{bi}(\Box(\neg(\text{init } w \wedge \text{empty}))) = \text{bi}(\Box(\neg(\text{init } w) \vee \neg \text{empty}))$   
**by** (*simp add: BiEqvBi*)  
**hence** 174:  $\vdash \text{bi}(\Box(\neg(\text{init } w \wedge \text{empty}))); \text{skip} = \text{bi}(\Box(\neg(\text{init } w) \vee \neg \text{empty})); \text{skip}$   
**using** *LeftChopEqvChop* **by** *blast*  
**hence** 17:  $\vdash (\text{empty} \vee (\text{bi}(\Box(\neg(\text{init } w \wedge \text{empty}))); \text{skip})) =$   
 $(\text{empty} \vee (\text{bi}(\Box(\neg(\text{init } w) \vee \neg \text{empty}))); \text{skip}))$   
**by** *auto*

**have 181:**  $\vdash (\neg(\text{init } w) \vee \neg \text{empty}) = (\neg \text{empty} \vee \neg(\text{init } w))$   
**by** *auto*  
**hence 18:**  $\vdash \square (\neg(\text{init } w) \vee \neg \text{empty}) = \square (\neg \text{empty} \vee \neg(\text{init } w))$   
**by** (*simp add: BoxEqvBox*)  
**have 191:**  $\vdash (\neg \text{empty} \vee \neg(\text{init } w)) = (\text{empty} \longrightarrow \neg(\text{init } w))$   
**by** *auto*  
**hence 19:**  $\vdash \square (\neg \text{empty} \vee \neg(\text{init } w)) = \square(\text{empty} \longrightarrow \neg(\text{init } w))$   
**by** (*simp add: BoxEqvBox*)  
**have 20:**  $\vdash \square(\text{empty} \longrightarrow \neg(\text{init } w)) = \text{fin } (\neg(\text{init } w))$   
**by** (*simp add: fin-d-def*)  
**have 21:**  $\vdash \text{fin } (\neg(\text{init } w)) = (\neg(\text{fin } (\text{init } w)))$   
**using** *FinEqvFin FinNotStateEqvNotFinState Initprop(2)* **by** *fastforce*  
**have 22:**  $\vdash \text{bi}(\square (\neg(\text{init } w) \vee \neg \text{empty})) = \text{bi } (\neg(\text{fin } (\text{init } w)))$   
**using** *18 19 20 21 BiEqvBi* **by** (*metis int-eq*)  
**hence 23:**  $\vdash (\text{bi}(\square (\neg(\text{init } w) \vee \neg \text{empty}))); \text{skip} = (\text{bi } (\neg(\text{fin } (\text{init } w)))); \text{skip}$   
**using** *RightChopEqvChop* **by** (*simp add: LeftChopEqvChop*)  
**hence 24:**  $\vdash (\text{empty} \vee (\text{bi}(\square (\neg(\text{init } w) \vee \neg \text{empty}))); \text{skip}) =$   
 $(\text{empty} \vee (\text{bi } (\neg(\text{fin } (\text{init } w)))); \text{skip})$   
**by** *auto*  
**hence 25:**  $\vdash (\text{empty} \vee (\text{bi } (\neg(\text{fin } (\text{init } w)))); \text{skip}) = \text{bs}(\neg(\text{fin } (\text{init } w)))$   
**by** (*simp add: bs-d-def*)  
**show** *?thesis*  
**by** (*metis 01 1 11 16 172 18 19 20 21 25 inteq-reflection sometimes-d-def*)  
**qed**

**lemma** *BsFalseEqvEmpty:*

$\vdash \text{bs } \# \text{False} = \text{empty}$

**proof** –

**have 1:**  $\vdash \text{bs } \# \text{False} = (\text{empty} \vee \text{bi } \# \text{False}; \text{skip})$

**by** (*simp add: bs-d-def*)

**have 2:**  $\vdash \neg(\text{bi } \# \text{False}; \text{skip})$

**by** (*metis BiEqvAndWprevBi MoreEqvSkipChopTrue NotChopSkipEqvMoreAndNotChopSkip*  
*SkipTrueEqvTrueSkip int-eq int-iffD1 int-simps(14) int-simps(19) int-simps(2)*  
*int-simps(21)*)

**from 1 2 show** *?thesis* **by** *fastforce*

**qed**

#### 9.4.4 First occurrence

**lemma** *FstWithAndImp:*

$\vdash \triangleright f \wedge g \longrightarrow \triangleright (f \wedge g)$

**proof** –

**have 1:**  $\vdash (\triangleright f \wedge g) = ((f \wedge (\text{bs } (\neg f)))) \wedge g$

**by** (*simp add: first-d-def*)

**have 2:**  $\vdash ((f \wedge (\text{bs } (\neg f)))) \wedge g = (f \wedge \neg(ds f)) \wedge g$

**using** *NotDsEqvBsNot* **by** *fastforce*

**have 3:**  $\vdash \neg(ds f) \longrightarrow \neg(ds(f \wedge g))$

**using** *DsAndImpElimL* **by** *fastforce*

**hence 4:**  $\vdash f \wedge \neg(ds f) \wedge g \longrightarrow f \wedge g \wedge \neg(ds(f \wedge g))$

**by** *auto*

**have** 5:  $\vdash (f \wedge g \wedge \neg(ds(f \wedge g))) = ((f \wedge g) \wedge (bs(\neg(f \wedge g))))$   
**using** *NotDsEqvBsNot* **by** *fastforce*  
**have** 6:  $\vdash ((f \wedge g) \wedge (bs(\neg(f \wedge g)))) = \triangleright(f \wedge g)$   
**by** (*simp add: first-d-def*)  
**from** 1 2 4 5 6 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *FstWithOrEqv*:

$\vdash \triangleright(f \vee g) = ((\triangleright f \wedge bs(\neg g)) \vee (\triangleright g \wedge bs(\neg f)))$   
**proof** –  
**have** 1:  $\vdash \triangleright(f \vee g) = ((f \vee g) \wedge bs(\neg(f \vee g)))$   
**by** (*simp add: first-d-def*)  
**have** 2:  $\vdash (\neg(f \vee g)) = (\neg f \wedge \neg g)$   
**by** *auto*  
**hence** 3:  $\vdash bs(\neg(f \vee g)) = bs(\neg f \wedge \neg g)$   
**using** *BsEqvRule* **by** *blast*  
**have** 4:  $\vdash bs(\neg f \wedge \neg g) = (bs(\neg f) \wedge bs(\neg g))$   
**using** *BsAndEqv* **by** *fastforce*  
**have** 5:  $\vdash ((f \vee g) \wedge bs(\neg(f \vee g))) = ((f \vee g) \wedge bs(\neg f) \wedge bs(\neg g))$   
**using** 3 4 **by** *fastforce*  
**have** 6:  $\vdash ((f \vee g) \wedge bs(\neg f) \wedge bs(\neg g)) =$   
 $((f \wedge bs(\neg f)) \wedge bs(\neg g)) \vee (g \wedge bs(\neg f) \wedge bs(\neg g))$   
**by** *auto*  
**have** 7:  $\vdash ((f \wedge bs(\neg f)) \wedge bs(\neg g)) = (\triangleright f \wedge bs(\neg g))$   
**by** (*simp add: first-d-def*)  
**have** 8:  $\vdash (g \wedge bs(\neg f) \wedge bs(\neg g)) = ((g \wedge bs(\neg g)) \wedge bs(\neg f))$   
**by** *auto*  
**have** 9:  $\vdash ((g \wedge bs(\neg g)) \wedge bs(\neg f)) = (\triangleright g \wedge bs(\neg f))$   
**by** (*simp add: first-d-def*)  
**have** 10:  $\vdash ((f \wedge bs(\neg f)) \wedge bs(\neg g)) \vee (g \wedge bs(\neg f) \wedge bs(\neg g)) =$   
 $(\triangleright f \wedge bs(\neg g)) \vee (\triangleright g \wedge bs(\neg f))$   
**using** 7 8 9 **by** *fastforce*  
**from** 1 5 6 10 **show** *?thesis* **by** (*metis* 7 8 9 *int-eq*)  
**qed**

**lemma** *FstFstAndEqvFstAnd*:

$\vdash \triangleright(\triangleright f \wedge g) = (\triangleright f \wedge g)$   
**proof** –  
**have** 1:  $\vdash (\triangleright f \wedge g) = ((f \wedge (bs(\neg f))) \wedge g)$  **by** (*simp add: first-d-def*)  
**hence** 2:  $\vdash \triangleright f \wedge g \longrightarrow (bs(\neg f))$  **by** *auto*  
**hence** 3:  $\vdash \triangleright f \wedge g \longrightarrow \triangleright f \wedge g \wedge (bs(\neg f))$  **by** *auto*  
**have** 4:  $\vdash \neg f \longrightarrow \neg f \vee \neg(bs(\neg f)) \vee \neg g$  **by** *auto*  
**hence** 5:  $\vdash bs(\neg f) \longrightarrow bs(\neg f \vee \neg(bs(\neg f)) \vee \neg g)$  **using** *BsImpBsRule* **by** *blast*  
**have** 6:  $\vdash (\neg f \vee \neg(bs(\neg f)) \vee \neg g) = (\neg(f \wedge bs(\neg f) \wedge g))$  **by** *auto*  
**hence** 7:  $\vdash bs(\neg f \vee \neg(bs(\neg f)) \vee \neg g) = bs(\neg(f \wedge bs(\neg f) \wedge g))$  **using** *BsEqvRule* **by** *blast*  
**have** 8:  $\vdash ((f \wedge bs(\neg f)) \wedge g) = (\triangleright f \wedge g)$  **by** (*simp add: first-d-def*)  
**hence** 9:  $\vdash (\neg(f \wedge bs(\neg f) \wedge g)) = (\neg(\triangleright f \wedge g))$  **by** *auto*  
**hence** 10:  $\vdash bs(\neg(f \wedge bs(\neg f) \wedge g)) = bs(\neg(\triangleright f \wedge g))$  **using** *BsEqvRule* **by** *blast*  
**have** 11:  $\vdash \triangleright f \wedge g \longrightarrow (\triangleright f \wedge g) \wedge bs(\neg(\triangleright f \wedge g))$  **using** 3 5 7 10 **by** *fastforce*  
**hence** 12:  $\vdash \triangleright f \wedge g \longrightarrow \triangleright(\triangleright f \wedge g)$  **by** (*simp add: first-d-def*)

**have** 13:  $\vdash \triangleright(\triangleright f \wedge g) = ((\triangleright f \wedge g) \wedge \text{bs } (\neg(\triangleright f \wedge g)))$  **by** (*simp add: first-d-def*)  
**hence** 14:  $\vdash \triangleright(\triangleright f \wedge g) \longrightarrow \triangleright f \wedge g$  **by** *auto*  
**from** 12 14 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *FstTrue*:

$\vdash \triangleright \#True = \text{empty}$

**proof** –

**have** 1:  $\vdash \triangleright \#True = (\#True \wedge \text{bs } (\neg \#True))$

**by** (*simp add: first-d-def*)

**have** 2:  $\vdash \text{bs } (\neg \#True) = (\text{empty} \vee (\text{bi } (\neg \#True)); \text{skip})$

**by** (*simp add: bs-d-def*)

**have** 3:  $\vdash \neg(\text{bi } (\neg \#True))$

**using** *BiElim* **by** *fastforce*

**have** 4:  $\vdash \neg((\text{bi } (\neg \#True)); \text{skip})$

**by** (*metis AndChopA BiEqvAndEmptyOrBiChopSkip MoreEqvSkipChopTrue*

*NotChopSkipEqvMoreAndNotChopSkip SkipTrueEqvTrueSkip int-eq int-simps(14) int-simps(21)*)

**have** 5:  $\vdash \text{bs } (\neg \#True) = \text{empty}$

**using** 2 4 **by** *fastforce*

**from** 1 5 **show** ?thesis **by** *fastforce*

**qed**

**lemma** *FstFalse*:

$\vdash \neg(\triangleright \#False)$

**proof** –

**have** 1:  $\vdash \triangleright \#False = (\#False \wedge \text{bs } \#True)$  **by** (*simp add: first-d-def*)

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

**qed**

**lemma** *FstChopFalseEqvFalse*:

$\vdash \neg(\triangleright f ; \#False)$

**by** (*simp add: Valid-def chop-defs*)

**lemma** *FstEmpty*:

$\vdash \triangleright \text{empty} = \text{empty}$

**proof** –

**have** 1:  $\vdash \triangleright \text{empty} = (\text{empty} \wedge \text{bs } (\neg \text{empty}))$  **by** (*simp add: first-d-def*)

**have** 2:  $\vdash \text{bs } (\neg \text{empty}) = (\text{empty} \vee \text{bi } (\neg \text{empty}); \text{skip})$  **by** (*simp add: bs-d-def*)

**from** 1 2 **show** ?thesis **by** *fastforce*

**qed**

**lemma** *FstAndEmptyEqvAndEmpty*:

$\vdash (\triangleright f \wedge \text{empty}) = (f \wedge \text{empty})$

**proof** –

**have** 1:  $\vdash (\triangleright f \wedge \text{empty}) = ((f \wedge \text{bs } (\neg f)) \wedge \text{empty})$  **by** (*simp add: first-d-def*)

**have** 2:  $\vdash \text{bs } (\neg f) = (\text{empty} \vee \text{bi } (\neg f); \text{skip})$  **by** (*simp add: bs-d-def*)

**from** 1 2 **show** ?thesis **by** *fastforce*

**qed**

**lemma** *FstEmptyOrEqvEmpty*:

$\vdash \triangleright(\text{empty} \vee f) = \text{empty}$

**proof** –

**have** 1:  $\vdash \triangleright(\text{empty} \vee f) = ((\triangleright \text{empty} \wedge \text{bs } (\neg f)) \vee (\triangleright f \wedge \text{bs } (\neg \text{empty})))$  **using** *FstWithOrEqv* **by** *blast*

**have** 2:  $\vdash (\neg \text{empty}) = \text{more}$  **by** (*simp add: empty-d-def*)

**hence** 3:  $\vdash \text{bs } (\neg \text{empty}) = \text{bs more}$  **using** *BsEqvRule* **by** *blast*

**have** 4:  $\vdash \text{bs more} = \text{empty}$  **using** *BsMoreEqvEmpty* **by** *blast*

**have** 5:  $\vdash (\triangleright f \wedge \text{bs } (\neg \text{empty})) = (\triangleright f \wedge \text{empty})$  **using** 3 4 **by** *fastforce*

**have** 6:  $\vdash \triangleright \text{empty} = \text{empty}$  **using** *FstEmpty* **by** *blast*

**hence** 7:  $\vdash (\triangleright \text{empty} \wedge \text{bs } (\neg f)) = (\text{empty} \wedge \text{bs } (\neg f))$  **by** *auto*

**have** 8:  $\vdash (\text{empty} \wedge \text{bs } (\neg f)) = (\text{empty} \wedge (\text{empty} \vee \text{bi } (\neg f); \text{skip}))$  **by** (*simp add: bs-d-def*)

**have** 9:  $\vdash (\text{empty} \wedge (\text{empty} \vee \text{bi } (\neg f); \text{skip})) = \text{empty}$  **by** *auto*

**have** 10:  $\vdash (\text{empty} \wedge \text{bs } (\neg f)) = \text{empty}$  **using** 8 9 **by** *auto*

**have** 11:  $\vdash ((\triangleright \text{empty} \wedge \text{bs } (\neg f)) \vee (\triangleright f \wedge \text{bs } (\neg \text{empty}))) =$   
 $(\text{empty} \vee (\triangleright f \wedge \text{empty}))$  **using** 7 10 5 **by** *fastforce*

**have** 12:  $\vdash (\text{empty} \vee (\triangleright f \wedge \text{empty})) = \text{empty}$  **by** *auto*

**from** 1 11 12 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *FstChopEmptyEqvFstChopFstEmpty*:

$\vdash (\triangleright f; g \wedge \text{empty}) = (\triangleright f; \triangleright g \wedge \text{empty})$

**proof** –

**have** 1:  $\vdash (\triangleright f; g \wedge \text{empty}) = (\triangleright f \wedge g \wedge \text{empty})$  **using** *ChopEmptyAndEmpty* **by** *blast*

**have** 2:  $\vdash (\triangleright g \wedge \text{empty}) = (g \wedge \text{empty})$  **using** *FstAndEmptyEqvAndEmpty* **by** *blast*

**hence** 3:  $\vdash (\triangleright f \wedge g \wedge \text{empty}) = (\triangleright f \wedge \triangleright g \wedge \text{empty})$  **by** *auto*

**have** 4:  $\vdash (\triangleright f; \triangleright g \wedge \text{empty}) = (\triangleright f \wedge \triangleright g \wedge \text{empty})$  **using** *ChopEmptyAndEmpty* **by** *blast*

**from** 1 3 4 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *FstMoreEqvSkip*:

$\vdash \triangleright \text{more} = \text{skip}$

**proof** –

**have** 1:  $\vdash \triangleright \text{more} = (\text{more} \wedge \text{bs } (\neg \text{more}))$  **by** (*simp add: first-d-def*)

**have** 2:  $\vdash (\text{more} \wedge \text{bs } (\neg \text{more})) = (\text{more} \wedge (\text{empty} \vee \text{bi } (\neg \text{more}); \text{skip}))$  **by** (*simp add: bs-d-def*)

**have** 3:  $\vdash (\text{more} \wedge (\text{empty} \vee \text{bi } (\neg \text{more}); \text{skip})) = (\text{more} \wedge \text{bi } (\neg \text{more}); \text{skip})$  **using** *empty-d-def*  
**using** *MoreAndEmptyOrEqvMoreAnd* **by** *fastforce*

**have** 4:  $\vdash (\text{more} \wedge ((\text{bi } (\neg \text{more})); \text{skip})) = ((\text{bi } (\neg \text{more})); \text{skip})$  **using** *ChopSkipImpMore* **by** *fastforce*

**have** 5:  $\vdash ((\text{bi } (\neg \text{more})); \text{skip}) = \text{bi empty}; \text{skip}$  **by** (*simp add: empty-d-def*)

**have** 6:  $\vdash \text{bi empty} = \text{empty}$  **using** *BiEmptyEqvEmpty* **by** *auto*

**hence** 7:  $\vdash \text{bi empty}; \text{skip} = \text{empty}; \text{skip}$  **using** *LeftChopEqvChop* **by** *blast*

**have** 8:  $\vdash \text{empty}; \text{skip} = \text{skip}$  **using** *EmptyChop* **by** *blast*

**from** 1 2 3 4 5 7 8 **show** *?thesis* **by** (*metis int-eq*)

**qed**

**lemma** *FstEqvBsNotAndDi*:

$\vdash \triangleright f = (\text{bs } (\neg f) \wedge \text{di } f)$

**proof** –

**have** 1:  $\vdash \text{bs } (\neg f) = (\neg(\text{ds } f))$  **by** (*simp add: ds-d-def*)

**hence** 2:  $\vdash (\text{bs } (\neg f) \wedge \text{di } f) = (\neg(\text{ds } f) \wedge \text{di } f)$  **by** *auto*

**have** 3:  $\vdash \text{di } f = (\text{ds } f \vee f)$  **using** *OrDsEqvDi* **by** *fastforce*

**hence** 4:  $\vdash (\neg(\text{ds } f) \wedge \text{di } f) = (\neg(\text{ds } f) \wedge (\text{ds } f \vee f))$  **by** *auto*

**have** 5 :  $\vdash (\neg(ds\ f) \wedge (ds\ f \vee f)) = (\neg(ds\ f) \wedge f)$  **by** *auto*  
**have** 6 :  $\vdash (\neg(ds\ f) \wedge f) = (f \wedge bs\ (\neg\ f))$  **using** 1 **by** *auto*  
**from** 2 4 5 6 **show** *?thesis* **by** (*metis first-d-def int-eq*)  
**qed**

**lemma** *FstOrDiEqvDi*:

$\vdash (\triangleright f \vee di\ f) = di\ f$

**proof** –

**have** 1:  $\vdash (\triangleright f \vee di\ f) = ((f \wedge bs\ (\neg\ f)) \vee di\ f)$  **by** (*simp add: first-d-def*)  
**have** 2:  $\vdash ((f \wedge bs\ (\neg\ f)) \vee di\ f) = ((f \vee di\ f) \wedge (bs\ (\neg\ f) \vee di\ f))$  **by** *auto*  
**have** 3:  $\vdash (f \vee di\ f) = di\ f$   
**by** (*metis 2 DiIntro RRDiamondEqvDi int-eq Prop02 Prop03 Prop11 Prop12*)  
**hence** 4:  $\vdash ((f \vee di\ f) \wedge (bs\ (\neg\ f) \vee di\ f)) = (di\ f \wedge (bs\ (\neg\ f) \vee di\ f))$  **by** *auto*  
**have** 5:  $\vdash (di\ f \wedge (bs\ (\neg\ f) \vee di\ f)) = di\ f$  **by** *auto*  
**from** 1 2 4 5 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *FstAndDiEqvFst*:

$\vdash (\triangleright f \wedge di\ f) = \triangleright f$

**proof** –

**have** 1:  $\vdash (\triangleright f \wedge di\ f) = ((f \wedge bs\ (\neg\ f)) \wedge di\ f)$  **by** (*simp add: first-d-def*)  
**have** 2:  $\vdash (f \wedge di\ f) = f$  **by** (*meson DiIntro Prop10 Prop11*)  
**hence** 3:  $\vdash (f \wedge bs\ (\neg\ f) \wedge di\ f) = (f \wedge bs\ (\neg\ f))$  **by** *auto*  
**from** 1 3 **show** *?thesis* **by** (*metis first-d-def int-iffD2 int-iffI Prop12*)

**qed**

**lemma** *DiEqvDiFst*:

$\vdash di\ f = di\ (\triangleright f)$

**proof** –

**have** 1:  $\vdash di\ (\triangleright f) = di\ (f \wedge bs\ (\neg\ f))$   
**by** (*simp add: first-d-def*)  
**have** 2:  $\vdash di\ (f \wedge bs\ (\neg\ f)) \longrightarrow di\ f \wedge di\ (bs\ (\neg\ f))$   
**using** *DiAndImpAnd* **by** *auto*  
**hence** 3:  $\vdash di\ (f \wedge bs\ (\neg\ f)) \longrightarrow di\ f$   
**by** *auto*  
**have** 4:  $\vdash di\ (\triangleright f) \longrightarrow di\ f$  **using** 1 3  
**by** *fastforce*  
**have** 5:  $\vdash (di\ f \wedge empty) = (f \wedge empty)$   
**using** *DiAndEmptyEqvAndEmpty* **by** *blast*  
**have** 6:  $\vdash (\triangleright f \wedge empty) = (f \wedge empty)$   
**using** *FstAndEmptyEqvAndEmpty* **by** *auto*  
**have** 7:  $\vdash di\ f \wedge empty \longrightarrow \triangleright f$   
**using** 5 6 **by** *fastforce*  
**have** 8:  $\vdash \triangleright f \longrightarrow di\ (\triangleright f)$   
**using** *DiIntro* **by** *auto*  
**have** 9:  $\vdash di\ f \wedge empty \longrightarrow di\ (\triangleright f)$   
**using** 7 8 **using** *lift-imp-trans* **by** *blast*  
**hence** 10:  $\vdash empty \longrightarrow (di\ f \longrightarrow di\ (\triangleright f))$   
**by** *auto*  
**have** 11:  $\vdash prev\ (di\ f \longrightarrow di\ (\triangleright f)) \longrightarrow more$

by (*simp add: ChopSkipImpMore prev-d-def*)  
**have 12:**  $\vdash \text{more} \longrightarrow (\text{prev} (di\ f \longrightarrow di\ (\triangleright f)) = (\text{prev}(di\ f) \longrightarrow \text{prev}(di\ (\triangleright f))))$   
 using *MoreImpImpPrevEqv* by *auto*  
**have 13:**  $\vdash (\text{more} \wedge \text{prev} (di\ f \longrightarrow di\ (\triangleright f))) = (\text{more} \wedge (\text{prev}(di\ f) \longrightarrow \text{prev}(di\ (\triangleright f))))$   
 using 12 by *fastforce*  
**have 14:**  $\vdash \text{prev} (di\ f \longrightarrow di\ (\triangleright f)) = (\text{more} \wedge (\text{prev}(di\ f) \longrightarrow \text{prev}(di\ (\triangleright f))))$   
 using 11 13 by *fastforce*  
**have 15:**  $\vdash di\ f = (f \vee ds\ f)$   
 using *OrDsEqvDi* by *fastforce*  
**have 16:**  $\vdash di\ f = (di\ f \wedge (bs\ (\neg f) \vee \neg(bs\ (\neg f))))$   
 by *auto*  
**have 17:**  $\vdash (di\ f \wedge (bs\ (\neg f) \vee \neg(bs\ (\neg f)))) = ((di\ f \wedge bs\ (\neg f)) \vee (di\ f \wedge \neg(bs\ (\neg f))))$   
 by *auto*  
**have 18:**  $\vdash (di\ f \wedge bs\ (\neg f)) = ((f \vee ds\ f) \wedge bs\ (\neg f))$   
 using 15 by *auto*  
**have 19:**  $\vdash ((f \vee ds\ f) \wedge bs\ (\neg f)) = ((f \wedge bs\ (\neg f)) \vee (ds\ f \wedge bs\ (\neg f)))$   
 by *auto*  
**have 20:**  $\vdash \neg(ds\ f \wedge bs\ (\neg f))$   
 by (*simp add: ds-d-def*)  
**have 21:**  $\vdash ((f \wedge bs\ (\neg f)) \vee (ds\ f \wedge bs\ (\neg f))) = (f \wedge bs\ (\neg f))$   
 using 20 by *auto*  
**have 22:**  $\vdash (di\ f \wedge bs\ (\neg f)) = (f \wedge bs\ (\neg f))$   
 using 18 19 21 by *fastforce*  
**have 23:**  $\vdash (f \wedge bs\ (\neg f)) = \triangleright f$   
 by (*simp add: first-d-def*)  
**have 24:**  $\vdash (\triangleright f) \longrightarrow di\ (\triangleright f)$   
 using *DiIntro* by *auto*  
**have 25:**  $\vdash (f \wedge bs\ (\neg f)) \longrightarrow di\ (\triangleright f)$   
 using 23 24 by *fastforce*  
**have 26:**  $\vdash (di\ f \wedge bs\ (\neg f)) \longrightarrow di\ (\triangleright f)$   
 using 25 22 by *fastforce*  
**hence 27:**  $\vdash (di\ f \wedge bs\ (\neg f) \wedge (\text{prev} (di\ f \longrightarrow di\ (\triangleright f)))) \longrightarrow di\ (\triangleright f)$   
 by *auto*  
**have 28:**  $\vdash (di\ f \wedge \neg(bs\ (\neg f))) = (di\ f \wedge ds\ f)$   
 by (*simp add: ds-d-def*)  
**hence 29:**  $\vdash (di\ f \wedge \neg(bs\ (\neg f)) \wedge (\text{prev} (di\ f \longrightarrow di\ (\triangleright f)))) =$   
 $(di\ f \wedge ds\ f \wedge (\text{prev} (di\ f \longrightarrow di\ (\triangleright f))))$   
 by *auto*  
**have 30:**  $\vdash ds\ f = \text{prev}(di\ f)$   
 using *DsDi* by (*metis prev-d-def*)  
**hence 31:**  $\vdash (di\ f \wedge ds\ f \wedge (\text{prev} (di\ f \longrightarrow di\ (\triangleright f)))) =$   
 $(di\ f \wedge \text{prev}(di\ f) \wedge (\text{prev} (di\ f \longrightarrow di\ (\triangleright f))))$   
 by *auto*  
**have 32:**  $\vdash \text{prev} (di\ f \longrightarrow di\ (\triangleright f)) \longrightarrow (\text{prev}(di\ f) \longrightarrow \text{prev}(di\ (\triangleright f)))$   
 using 14 by *auto*  
**hence 33:**  $\vdash di\ f \wedge \text{prev}(di\ f) \wedge \text{prev} (di\ f \longrightarrow di\ (\triangleright f)) \longrightarrow$   
 $di\ f \wedge \text{prev}(di\ f) \wedge (\text{prev}(di\ f) \longrightarrow \text{prev}(di\ (\triangleright f)))$   
 by *auto*  
**have 34:**  $\vdash di\ f \wedge \text{prev}(di\ f) \wedge (\text{prev}(di\ f) \longrightarrow \text{prev}(di\ (\triangleright f))) \longrightarrow \text{prev}(di\ (\triangleright f))$   
 by *auto*



**have** 35:  $\vdash \text{prev}(di (\triangleright f)) = (di (\triangleright f)); \text{skip}$   
**by** (*simp add: prev-d-def*)  
**have** 36:  $\vdash (di (\triangleright f)); \text{skip} \longrightarrow di(di (\triangleright f))$   
**using** *ChopImpDi* **by** *auto*  
**have** 37:  $\vdash di(di (\triangleright f)) = di (\triangleright f)$   
**using** *DiEqvDiDi* **by** *fastforce*  
**have** 38:  $\vdash di f \wedge \text{prev}(di f) \wedge (\text{prev}(di f) \longrightarrow \text{prev}(di (\triangleright f))) \longrightarrow di (\triangleright f)$   
**using** 37 36 35 34 **by** *fastforce*  
**have** 39:  $\vdash di f \wedge \neg(bs (\neg f)) \wedge (\text{prev}(di f \longrightarrow di (\triangleright f))) \longrightarrow di (\triangleright f)$   
**using** 29 31 33 38 **by** *fastforce*  
**hence** 40:  $\vdash \neg(bs (\neg f)) \wedge (\text{prev}(di f \longrightarrow di (\triangleright f))) \longrightarrow (di f \longrightarrow di (\triangleright f))$   
**by** *fastforce*  
**have** 41:  $\vdash bs (\neg f) \wedge (\text{prev}(di f \longrightarrow di (\triangleright f))) \longrightarrow (di f \longrightarrow di (\triangleright f))$   
**using** 27 **by** *fastforce*  
**have** 42:  $\vdash (\neg(bs (\neg f)) \vee bs (\neg f)) \wedge (\text{prev}(di f \longrightarrow di (\triangleright f))) \longrightarrow (di f \longrightarrow di (\triangleright f))$   
**using** 40 41 **by** *fastforce*  
**have** 43:  $\vdash (\neg(bs (\neg f)) \vee bs (\neg f))$   
**by** *auto*  
**have** 44:  $\vdash (\text{prev}(di f \longrightarrow di (\triangleright f))) \longrightarrow (di f \longrightarrow di (\triangleright f))$   
**using** 42 43 **by** *fastforce*  
**have** 45:  $\vdash di f \longrightarrow di (\triangleright f)$   
**using** 10 44 *EmptyChopSkipInduct* **by** *blast*  
**from** 4 45 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *FstDiEqvFst*:

$\vdash \triangleright(di f) = \triangleright f$

**proof** –

**have** 1:  $\vdash \triangleright(di f) = (di f \wedge bs (\neg (di f)))$  **by** (*simp add: first-d-def*)  
**have** 2:  $\vdash \neg (di f) = bi (\neg f)$  **by** (*simp add: NotDiEqvBiNot*)  
**hence** 3:  $\vdash bs (\neg (di f)) = bs (bi (\neg f))$  **using** *BsEqvRule* **by** *blast*  
**have** 4:  $\vdash bs (bi (\neg f)) = bs (\neg f)$  **using** *BsEqvBsBi* **by** *fastforce*  
**hence** 5:  $\vdash (di f \wedge bs (\neg (di f))) = (di f \wedge bs (\neg f))$  **using** 3 **by** *fastforce*  
**have** 6:  $\vdash di f = (f \vee ds f)$  **using** *OrDsEqvDi* **by** *fastforce*  
**hence** 7:  $\vdash (di f \wedge bs (\neg f)) = ((f \vee ds f) \wedge bs (\neg f))$  **by** *auto*  
**have** 8:  $\vdash ((f \vee ds f) \wedge bs (\neg f)) = ((f \wedge bs (\neg f)) \vee (ds f \wedge bs (\neg f)))$  **by** *auto*  
**have** 9:  $\vdash \neg(ds f \wedge bs (\neg f))$  **by** (*simp add: ds-d-def*)  
**have** 10:  $\vdash (f \wedge bs (\neg f)) = \triangleright f$  **by** (*simp add: first-d-def*)  
**have** 11:  $\vdash ((f \wedge bs (\neg f)) \vee (ds f \wedge bs (\neg f))) = \triangleright f$  **using** 9 10 **by** *fastforce*  
**from** 1 5 7 8 11 **show** *?thesis* **by** (*metis int-eq*)

**qed**

**lemma** *DiAndFstOrEqvFstOrDiAnd*:

$\vdash (di f \wedge (\triangleright f \vee g)) = (\triangleright f \vee (di f \wedge g))$

**proof** –

**have** 1:  $\vdash (di f \wedge (\triangleright f \vee g)) = (\triangleright f \wedge di f) \vee (di f \wedge g)$  **by** *auto*  
**have** 2:  $\vdash (\triangleright f \wedge di f) = \triangleright f$  **using** *FstAndDiEqvFst* **by** *blast*  
**from** 1 2 **show** *?thesis* **by** *auto*

**qed**

**lemma** *DiOrFstAndEqvDi*:

$\vdash di\ f \vee (\triangleright f \wedge g) = di\ f$

**proof** –

**have** 1:  $\vdash (di\ f \vee (\triangleright f \wedge g)) = ((\triangleright f \vee di\ f) \wedge (di\ f \vee g))$  **by** *auto*

**have** 2:  $\vdash (\triangleright f \vee di\ f) = di\ f$  **using** *FstOrDiEqvDi* **by** *blast*

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

**qed**

**lemma** *FstDiAndDiEqv*:

$\vdash \triangleright(di\ f \wedge di\ g) = ((\triangleright f \wedge di\ g) \vee (\triangleright g \wedge di\ f))$

**proof** –

**have** 1:  $\vdash \triangleright(di\ f \wedge di\ g) = ((di\ f \wedge di\ g) \wedge bs(\neg(di\ f \wedge di\ g)))$  **by** (*simp add: first-d-def*)

**have** 2:  $\vdash (\neg(di\ f \wedge di\ g)) = (bi(\neg f) \vee bi(\neg g))$  **by** (*auto simp add: bi-d-def*)

**hence** 3:  $\vdash bs(\neg(di\ f \wedge di\ g)) = bs(bi(\neg f) \vee bi(\neg g))$  **using** *BsEqvRule* **by** *blast*

**hence** 4:  $\vdash ((di\ f \wedge di\ g) \wedge bs(\neg(di\ f \wedge di\ g))) =$   
 $(di\ f \wedge di\ g \wedge bs(bi(\neg f) \vee bi(\neg g)))$  **by** *auto*

**have** 5:  $\vdash (bs(\neg f) \vee bs(\neg g)) = bs(bi(\neg f) \vee bi(\neg g))$  **using** *BsOrBsEqvBsBiOrBi* **by** *blast*

**hence** 6:  $\vdash (di\ f \wedge di\ g \wedge bs(bi(\neg f) \vee bi(\neg g))) =$   
 $(di\ f \wedge di\ g \wedge (bs(\neg f) \vee bs(\neg g)))$  **by** *auto*

**have** 7:  $\vdash (di\ f \wedge di\ g \wedge (bs(\neg f) \vee bs(\neg g))) =$   
 $((bs(\neg f) \wedge di\ f \wedge di\ g) \vee (di\ f \wedge bs(\neg g) \wedge di\ g))$  **by** *auto*

**have** 8:  $\vdash \triangleright f = (bs(\neg f) \wedge di\ f)$  **using** *FstEqvBsNotAndDi* **by** *blast*

**hence** 9:  $\vdash (bs(\neg f) \wedge di\ f \wedge di\ g) = (\triangleright f \wedge di\ g)$  **by** *auto*

**have** 10:  $\vdash \triangleright g = (bs(\neg g) \wedge di\ g)$  **using** *FstEqvBsNotAndDi* **by** *blast*

**hence** 11:  $\vdash (di\ f \wedge bs(\neg g) \wedge di\ g) = (di\ f \wedge \triangleright g)$  **by** *auto*

**show** *?thesis*

**by** (*metis 11 4 6 7 9 first-d-def inteq-reflection lift-and-com*)

**qed**

**lemma** *BiNotFstEqvBiNot*:

$\vdash bi(\neg(\triangleright f)) = bi(\neg f)$

**proof** –

**have** 1:  $\vdash di\ f = di(\triangleright f)$  **using** *DiEqvDiFst* **by** *blast*

**hence** 2:  $\vdash (\neg(di\ f)) = (\neg(di(\triangleright f)))$  **by** *auto*

**from** 1 2 **show** *?thesis* **using** *NotDiEqvBiNot* **by** *fastforce*

**qed**

**lemma** *BsNotFstEqvBsNot*:

$\vdash bs(\neg(\triangleright f)) = bs(\neg f)$

**proof** –

**have** 1:  $\vdash bs(\neg(\triangleright f)) = (empty \vee bi(\neg(\triangleright f));skip)$  **by** (*simp add: bs-d-def*)

**have** 2:  $\vdash bi(\neg(\triangleright f)) = bi(\neg f)$  **using** *BiNotFstEqvBiNot* **by** *blast*

**hence** 3:  $\vdash bi(\neg(\triangleright f));skip = bi(\neg f);skip$  **using** *LeftChopEqvChop* **by** *blast*

**hence** 4:  $\vdash (empty \vee bi(\neg(\triangleright f));skip) = (empty \vee bi(\neg f);skip)$  **by** *auto*

**from** 1 4 **show** *?thesis* **by** (*simp add: bs-d-def*)

**qed**

**lemma** *FstState*:

$\vdash \triangleright(init\ w) = (empty \wedge init\ w)$

**proof** –

**have** 1:  $\vdash \triangleright (init\ w) = (init\ w \wedge bs\ (\neg(init\ w)))$  **by** (*simp add: first-d-def*)  
**hence** 2:  $\vdash \triangleright (init\ w) \longrightarrow init\ w$  **by** *auto*  
**have** 3:  $\vdash init\ w \longrightarrow bs\ (init\ w)$  **using** *StateImpBs* **by** *auto*  
**have** 4:  $\vdash \triangleright (init\ w) \longrightarrow bs\ (init\ w)$  **using** 2 3 **by** *fastforce*  
**have** 5:  $\vdash \triangleright (init\ w) \longrightarrow bs\ (\neg(init\ w))$  **using** 1 **by** *auto*  
**have** 6:  $\vdash \triangleright (init\ w) \longrightarrow bs\ (init\ w) \wedge bs\ (\neg(init\ w))$  **using** 4 5 **by** *fastforce*  
**have** 7:  $\vdash (bs\ (init\ w) \wedge bs\ (\neg(init\ w))) = (bs\ ((init\ w) \wedge \neg(init\ w)))$  **using** *BsAndEqv* **by** *blast*  
**have** 8:  $\vdash ((init\ w) \wedge \neg(init\ w)) = \#False$  **by** *auto*  
**hence** 9:  $\vdash (bs\ ((init\ w) \wedge \neg(init\ w))) = bs\ \#False$  **using** *BsEqvRule* **by** *blast*  
**have** 10:  $\vdash bs\ \#False = empty$  **using** *BsFalseEqvEmpty* **by** *auto*  
**have** 11:  $\vdash \triangleright (init\ w) \longrightarrow empty$  **using** 10 9 7 6 **by** *fastforce*  
**have** 12:  $\vdash \triangleright (init\ w) \longrightarrow empty \wedge init\ w$  **using** 11 2 **by** *fastforce*  
**have** 13:  $\vdash empty \wedge init\ w \longrightarrow empty$  **by** *auto*  
**hence** 14:  $\vdash empty \wedge init\ w \longrightarrow empty \vee bi\ (\neg(init\ w));skip$  **by** *auto*  
**hence** 15:  $\vdash empty \wedge init\ w \longrightarrow bs\ (\neg(init\ w))$  **by** (*simp add: bs-d-def*)  
**have** 16:  $\vdash empty \wedge init\ w \longrightarrow init\ w$  **by** *auto*  
**have** 17:  $\vdash empty \wedge init\ w \longrightarrow init\ w \wedge bs\ (\neg(init\ w))$  **using** 16 15 **by** *auto*  
**hence** 18:  $\vdash empty \wedge init\ w \longrightarrow \triangleright(init\ w)$  **by** (*simp add: first-d-def*)  
**from** 12 18 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *FstStateAndBsNotEmpty*:

$\vdash (\triangleright (init\ w) \wedge bs\ (\neg\ empty)) = \triangleright (init\ w)$

**proof** –

**have** 1:  $\vdash (\triangleright (init\ w) \wedge bs\ (\neg\ empty)) = (\triangleright (init\ w) \wedge bs\ more)$   
**using** *BsEqvRule NotEmptyEqvMore* **by** (*simp add: empty-d-def*)  
**have** 2:  $\vdash (\triangleright (init\ w) \wedge bs\ more) = (\triangleright (init\ w) \wedge empty)$   
**using** *BsMoreEqvEmpty* **by** *fastforce*  
**have** 3:  $\vdash \triangleright (init\ w) = (empty \wedge (init\ w))$   
**using** *FstState* **by** *blast*  
**hence** 4:  $\vdash (\triangleright (init\ w) \wedge empty) = (empty \wedge (init\ w) \wedge empty)$   
**by** *auto*  
**have** 5:  $\vdash (empty \wedge (init\ w) \wedge empty) = (empty \wedge (init\ w))$   
**by** *auto*  
**have** 6:  $\vdash (empty \wedge (init\ w)) = \triangleright(init\ w)$   
**using** *FstState* **by** *fastforce*  
**from** 1 2 4 5 6 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *FstStateImpFstStateOr*:

$\vdash \triangleright(init\ w) \longrightarrow \triangleright(init\ w \vee f)$

**proof** –

**have** 1:  $\vdash \triangleright(init\ w) = (empty \wedge init\ w)$   
**using** *FstState* **by** *blast*  
**have** 2:  $\vdash (empty \wedge init\ w) = (empty \wedge (empty \vee bi\ (\neg f);skip) \wedge init\ w)$   
**by** *auto*  
**have** 3:  $\vdash (empty \wedge (empty \vee bi\ (\neg f);skip) \wedge init\ w) =$   
 $(empty \wedge bs\ (\neg f) \wedge init\ w)$   
**by** (*simp add: bs-d-def*)  
**have** 4:  $\vdash (empty \wedge bs\ (\neg f) \wedge init\ w) = (empty \wedge init\ w \wedge bs\ (\neg f))$

by auto  
 have 5:  $\vdash (\text{empty} \wedge \text{init } w) = \triangleright (\text{init } w)$   
 using *FstState* by fastforce  
 hence 6:  $\vdash (\text{empty} \wedge \text{init } w \wedge \text{bs } (\neg f)) = (\triangleright (\text{init } w) \wedge \text{bs } (\neg f))$   
 by auto  
 have 7:  $\vdash \triangleright (\text{init } w) \wedge \text{bs } (\neg f) \longrightarrow (\triangleright (\text{init } w) \wedge \text{bs } (\neg f)) \vee (\triangleright f \wedge \text{bs } (\neg(\text{init } w)))$   
 by auto  
 have 8:  $\vdash \triangleright (\text{init } w \vee f) = ((\triangleright (\text{init } w) \wedge \text{bs } (\neg f)) \vee (\triangleright f \wedge \text{bs } (\neg(\text{init } w))))$   
 using *FstWithOrEqv* by blast  
 show ?thesis using 1 3 8 by fastforce  
 qed

**lemma** *FstLenSame*:

$(\forall \sigma. (\sigma \models \text{di } (\triangleright f \wedge \text{len}(i)) \wedge \text{di } (\triangleright f \wedge \text{len}(j))) \longrightarrow (i=j))$   
 by (simp add: *DiLenFstsem FstLenSamesem*)

**lemma** *FstLenSame-1*:

$\vdash \text{di } (\triangleright f \wedge \text{len}(i)) \wedge \text{di } (\triangleright f \wedge \text{len}(j)) \longrightarrow (\#i=\#j)$   
 using *FstLenSame Valid-def* by fastforce

**lemma** *FstAndLenSame*:

$(\forall \sigma. (\sigma \models \text{di } ((\triangleright f \wedge g1) \wedge \text{len}(i)) \wedge \text{di } ((\triangleright f \wedge g2) \wedge \text{len}(j))) \longrightarrow (i=j))$   
 using *linorder-neqE-nat* by (simp add: *DiLenFstAndsem*) blast

**lemma** *FstAndLenSame-1*:

$\vdash \text{di } ((\triangleright f \wedge g1) \wedge \text{len}(i)) \wedge \text{di } ((\triangleright f \wedge g2) \wedge \text{len}(j)) \longrightarrow (\#i=\#j)$   
 using *FstAndLenSame Valid-def* by fastforce

**lemma** *FstLenSameChop*:

$(\forall \sigma. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i));h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(j));h2) \longrightarrow (i=j))$

**proof**

fix  $\sigma$   
 show  $(\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i));h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(j));h2) \longrightarrow (i=j)$   
**proof**  
 assume 0:  $(\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i));h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(j));h2)$   
 have 1:  $(\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i));h1)$  using 0 by auto  
 have 2:  $(\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i));h1) \longrightarrow$   
      $(\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i));\# \text{True})$  by (metis *ChopImpDi Valid-def di-d-def unl-lift2*)  
 have 3:  $(\sigma \models \text{di}((\triangleright f \wedge g1) \wedge \text{len}(i)))$  using 1 2 by (simp add: *di-d-def*)  
 have 4:  $(\sigma \models ((\triangleright f \wedge g2) \wedge \text{len}(j));h2)$  using 0 by auto  
 have 5:  $(\sigma \models ((\triangleright f \wedge g2) \wedge \text{len}(j));h2) \longrightarrow$   
      $(\sigma \models ((\triangleright f \wedge g2) \wedge \text{len}(j));\# \text{True})$  by (metis *ChopImpDi Valid-def di-d-def unl-lift2*)  
 have 6:  $(\sigma \models \text{di}((\triangleright f \wedge g2) \wedge \text{len}(j)))$  using 4 5 by (simp add: *di-d-def*)  
 have 7:  $(\sigma \models \text{di}((\triangleright f \wedge g1) \wedge \text{len}(i)) \wedge \text{di}((\triangleright f \wedge g2) \wedge \text{len}(j)))$  using 3 6 by auto  
 thus  $(i=j)$  using *FstAndLenSame* by blast  
 qed  
 qed

**lemma** *FstLenSameChop-1*:

$\vdash ((\triangleright f \wedge g1) \wedge \text{len}(i));h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(j));h2 \longrightarrow (\#i=\#j)$   
**using** *FstLenSameChop Valid-def* **by** *fastforce*

**lemma** *DiImpExistsOneDiLenAndFst*:

$(\forall \sigma. (\sigma \models \text{di } f) \longrightarrow (\exists! k. (\sigma \models \text{di}(\triangleright f \wedge \text{len}(k))))))$

**proof**

**fix**  $\sigma$

**show**  $(\sigma \models \text{di } f) \longrightarrow (\exists! k. (\sigma \models \text{di}(\triangleright f \wedge \text{len}(k))))$

**proof**

**assume**  $0: (\sigma \models \text{di } f)$

**have**  $1: (\sigma \models \text{di}(\triangleright f))$

**using**  $0$  *DiEqvDiFst Valid-def* **by** *force*

**have**  $2: (\sigma \models \triangleright f) = ((\sigma \models \triangleright f) \wedge (\exists k. (\sigma \models \text{len}(k))))$

**using** *AndExistsLen[of TEMP  $\triangleright f$ ]* **by** (*simp add: Valid-def*)

**have**  $3: ((\sigma \models \triangleright f) \wedge (\exists k. (\sigma \models \text{len}(k)))) =$   
 $(\exists k. (\sigma \models \triangleright f) \wedge (\sigma \models \text{len}(k)))$

**by** *auto*

**have**  $4: (\sigma \models \text{di}(\triangleright f)) = (\exists k. (\sigma \models \text{di}(\triangleright f \wedge \text{len}(k))))$

**using**  $2\ 3$  **by** (*metis 1 DiLensem di-defs*)

**have**  $5: (\exists k. (\sigma \models \text{di}(\triangleright f \wedge \text{len}(k))))$

**using**  $1$  **using**  $4$  **by** *auto*

**then obtain**  $i$  **where**  $6: (\sigma \models \text{di}(\triangleright f \wedge \text{len}(i)))$  **by** *blast*

**from**  $5$  **obtain**  $j$  **where**  $7: (\sigma \models \text{di}(\triangleright f \wedge \text{len}(j)))$  **by** *blast*

**have**  $8: (\sigma \models \text{di}(\triangleright f \wedge \text{len}(i))) \wedge (\sigma \models \text{di}(\triangleright f \wedge \text{len}(j)))$

**using**  $6\ 7$  **by** *auto*

**hence**  $9: (\sigma \models \text{di}(\triangleright f \wedge \text{len}(i)) \wedge \text{di}(\triangleright f \wedge \text{len}(j)))$

**by** *simp*

**hence**  $10: i=j$

**using** *FstLenSame* **by** *blast*

**have**  $11: \bigwedge j. (\sigma \models \text{di}(\triangleright f \wedge \text{len}(j))) \longrightarrow (j=i)$

**using**  $9\ 10$  **using** *FstLenSame* **by** *auto*

**thus**  $(\exists! k. (\sigma \models \text{di}(\triangleright f \wedge \text{len}(k))))$

**using**  $11\ 5$  **by** *blast*

**qed**

**qed**

**lemma** *DiImpExistsOneDiLenAndFst-1*:

$\vdash \text{di } f \longrightarrow (\exists! k. (\text{di}(\triangleright f \wedge \text{len}(k))))$

**using** *Valid-def DiImpExistsOneDiLenAndFst* **by** *fastforce*

**lemma** *LFstAndDist-help*:

$(\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(k));h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(k));h2) =$   
 $(\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k));(h1 \wedge h2))$

**using** *LFixedAndDistr* **by** *fastforce*

**lemma** *LFstAndDist-help-1*:

$(\exists k. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(k));h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(k));h2)) =$   
 $(\exists k. (\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k));(h1 \wedge h2)))$

**proof**

**assume**  $0: \exists k. \sigma \models ((\triangleright f \wedge g1) \wedge \text{len } k) ; h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len } k) ; h2$

**obtain**  $k$  **where** 1:  $\sigma \models ((\triangleright f \wedge g1) \wedge \text{len } k) ; h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len } k) ; h2$   
**using** 0 **by** *auto*  
**hence** 2:  $(\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k)); (h1 \wedge h2))$   
**using** *LFstAndDist-help* **by** *blast*  
**show**  $(\exists k. (\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k)); (h1 \wedge h2) ))$   
**using** 2 **by** *auto*  
**next**  
**assume** 3:  $(\exists k. (\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k)); (h1 \wedge h2) ))$   
**obtain**  $k$  **where** 4:  $(\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k)); (h1 \wedge h2) )$   
**using** 3 **by** *auto*  
**hence** 5:  $(\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(k)); h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(k)); h2)$   
**using** *LFstAndDist-help* **by** *blast*  
**show**  $(\exists k. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(k)); h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(k)); h2))$   
**using** 5 **by** *auto*  
**qed**

**lemma** *LFstAndDistrsem*:

$(\forall \sigma. (\sigma \models ((\triangleright f \wedge g1); h1 \wedge (\triangleright f \wedge g2); h2) = (\triangleright f \wedge g1 \wedge g2); (h1 \wedge h2)))$   
**proof**  
**fix**  $\sigma$   
**show**  $(\sigma \models ((\triangleright f \wedge g1); h1 \wedge (\triangleright f \wedge g2); h2) = (\triangleright f \wedge g1 \wedge g2); (h1 \wedge h2))$   
**proof** –  
**have** 1:  $(\sigma \models (\triangleright f \wedge g1); h1) = (\exists i. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i)); h1) )$   
**using** *AndExistsLenChop[of TEMP  $(\triangleright f \wedge g1)$ ]* **by** *fastforce*  
**have** 2:  $(\sigma \models (\triangleright f \wedge g2); h2) = (\exists j. (\sigma \models ((\triangleright f \wedge g2) \wedge \text{len}(j)); h2) )$   
**using** *AndExistsLenChop[of TEMP  $(\triangleright f \wedge g2)$ ]* **by** *fastforce*  
**have** 3:  $(\sigma \models (\triangleright f \wedge g1); h1 \wedge (\triangleright f \wedge g2); h2) =$   
 $( (\exists i j. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i)); h1 \wedge$   
 $((\triangleright f \wedge g2) \wedge \text{len}(j)); h2) )$   
 $)$   
**using** 1 2 **by** *auto*  
**have** 4:  $( (\exists i j. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i)); h1 \wedge$   
 $((\triangleright f \wedge g2) \wedge \text{len}(j)); h2) )$   
 $) =$   
 $( (\exists k. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(k)); h1 \wedge$   
 $((\triangleright f \wedge g2) \wedge \text{len}(k)); h2) )$   
 $)$   
**using** *FstLenSameChop* **by** *blast*  
**have** 5:  $(\exists k. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(k)); h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(k)); h2)) =$   
 $(\exists k. (\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k)); (h1 \wedge h2) ))$   
**using** *LFstAndDist-help-1* **by** *blast*  
**have** 6 :  $(\exists k. (\sigma \models (((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)) \wedge \text{len}(k)); (h1 \wedge h2) )) =$   
 $(\sigma \models ((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)); (h1 \wedge h2))$   
**using** *AndExistsLenChop[of TEMP  $((\triangleright f \wedge g1) \wedge \triangleright f \wedge g2)$ ]* **by** *fastforce*  
**have** 7 :  $(\sigma \models ((\triangleright f \wedge g1) \wedge (\triangleright f \wedge g2)); (h1 \wedge h2)) =$   
 $(\sigma \models (\triangleright f \wedge g1 \wedge g2); (h1 \wedge h2))$   
**by** (*auto simp add: chop-defs*)  
**from** 3 4 5 6 7 **show** *?thesis* **by** *auto*  
**qed**  
**qed**

**lemma** *LFstAndDistr*:

$\vdash ((\triangleright f \wedge g1);h1 \wedge (\triangleright f \wedge g2);h2) = (\triangleright f \wedge g1 \wedge g2);(h1 \wedge h2)$

**using** *LFstAndDistrsem* **by** *fastforce*

**lemma** *LFstAndDistrA*:

$\vdash ((\triangleright f \wedge g1);h \wedge (\triangleright f \wedge g2);h) = (\triangleright f \wedge g1 \wedge g2);h$

**proof** –

**have** 1:  $\vdash ((\triangleright f \wedge g1);h \wedge (\triangleright f \wedge g2);h) = (\triangleright f \wedge g1 \wedge g2);(h \wedge h)$  **using** *LFstAndDistr* **by** *blast*

**have** 2:  $\vdash (\triangleright f \wedge g1 \wedge g2);(h \wedge h) = (\triangleright f \wedge g1 \wedge g2);h$  **by** *auto*

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

**qed**

**lemma** *LFstAndDistrB*:

$\vdash ((\triangleright f \wedge g);h1 \wedge (\triangleright f \wedge g);h2) = (\triangleright f \wedge g);(h1 \wedge h2)$

**proof** –

**have** 1:  $\vdash ((\triangleright f \wedge g);h1 \wedge (\triangleright f \wedge g);h2) = (\triangleright f \wedge g \wedge g);(h1 \wedge h2)$  **using** *LFstAndDistr* **by** *blast*

**have** 2:  $\vdash (\triangleright f \wedge g \wedge g);(h1 \wedge h2) = (\triangleright f \wedge g);(h1 \wedge h2)$  **by** *auto*

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

**qed**

**lemma** *LFstAndDistrC*:

$\vdash ((\triangleright f);h1 \wedge (\triangleright f);h2) = (\triangleright f);(h1 \wedge h2)$

**proof** –

**have** 1:  $\vdash ((\triangleright f \wedge \#True);h1 \wedge (\triangleright f \wedge \#True);h2) = (\triangleright f \wedge \#True \wedge \#True);(h1 \wedge h2)$

**using** *LFstAndDistr* **by** *blast*

**have** 2:  $\vdash (\triangleright f \wedge \#True);h1 = (\triangleright f);h1$

**by** *auto*

**have** 3:  $\vdash (\triangleright f \wedge \#True);h2 = (\triangleright f);h2$

**by** *auto*

**have** 4:  $\vdash (\triangleright f \wedge \#True \wedge \#True);(h1 \wedge h2) = (\triangleright f);(h1 \wedge h2)$

**by** *auto*

**from** 1 2 3 4 **show** *?thesis* **by** *auto*

**qed**

**lemma** *LFstAndDistrD*:

$\vdash (di(\triangleright f \wedge g1) \wedge di(\triangleright f \wedge g2)) = di(\triangleright f \wedge g1 \wedge g2)$

**proof** –

**have** 1:  $\vdash ((\triangleright f \wedge g1);\#True \wedge (\triangleright f \wedge g2);\#True) = (\triangleright f \wedge g1 \wedge g2);(\#True \wedge \#True)$

**using** *LFstAndDistr* **by** *blast*

**have** 2:  $\vdash (\triangleright f \wedge g1);\#True = di(\triangleright f \wedge g1)$

**by** (*simp add: di-d-def*)

**have** 3:  $\vdash (\triangleright f \wedge g2);\#True = di(\triangleright f \wedge g2)$

**by** (*simp add: di-d-def*)

**have** 4:  $\vdash (\triangleright f \wedge g1 \wedge g2);(\#True \wedge \#True) = di(\triangleright f \wedge g1 \wedge g2)$

**by** (*simp add: di-d-def*)

**from** 1 2 3 4 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *LstAndDistr*:

$$\vdash (h1;(\triangleleft f \wedge g1) \wedge h2;(\triangleleft f \wedge g2)) = (h1 \wedge h2);(\triangleleft f \wedge g1 \wedge g2)$$

**proof –**

$$\text{have } 1: \vdash ((\triangleright(f^r) \wedge g1^r);(h1^r) \wedge (\triangleright(f^r) \wedge (g2^r));(h2^r)) =$$

$$(\triangleright(f^r) \wedge (g1^r) \wedge (g2^r));((h1^r) \wedge (h2^r))$$

**using** *LFstAndDistr* **by** *blast*

$$\text{hence } 2: \vdash ((\triangleright(f^r) \wedge g1^r);(h1^r) \wedge (\triangleright(f^r) \wedge (g2^r));(h2^r))^r =$$

$$((\triangleright(f^r) \wedge (g1^r) \wedge (g2^r));((h1^r) \wedge (h2^r)))^r$$

**using** *1 REqvRule* **by** *blast*

$$\text{have } 3: \vdash (((\triangleright(f^r) \wedge g1^r);(h1^r))^r \wedge ((\triangleright(f^r) \wedge (g2^r));(h2^r))^r) =$$

$$((\triangleright(f^r) \wedge g1^r);(h1^r) \wedge (\triangleright(f^r) \wedge (g2^r));(h2^r))^r$$

**using** *RAnd* **by** *fastforce*

$$\text{have } 4: \vdash ((h1^r)^r;(\triangleright(f^r) \wedge g1^r)^r \wedge (h2^r)^r;(\triangleright(f^r) \wedge (g2^r))^r) =$$

$$(((\triangleright(f^r) \wedge g1^r);(h1^r))^r \wedge ((\triangleright(f^r) \wedge (g2^r));(h2^r))^r)$$

**using** *RevChop* **by** *fastforce*

$$\text{have } 5: \vdash (h1^r)^r = h1$$

**using** *EqvReverseReverse* **by** *blast*

$$\text{have } 6: \vdash (h2^r)^r = h2$$

**using** *EqvReverseReverse* **by** *blast*

$$\text{have } 7: \vdash (g1^r)^r = g1$$

**using** *EqvReverseReverse* **by** *blast*

$$\text{have } 8: \vdash (g2^r)^r = g2$$

**using** *EqvReverseReverse* **by** *blast*

$$\text{have } 9: \vdash (f^r)^r = f$$

**using** *EqvReverseReverse* **by** *blast*

$$\text{have } 10: \vdash (\triangleright(f^r) \wedge g1^r)^r = ((\triangleright(f^r))^r \wedge (g1^r)^r)$$

**using** *RAnd* **by** *blast*

$$\text{have } 11: \vdash (\triangleright(f^r) \wedge g2^r)^r = ((\triangleright(f^r))^r \wedge (g2^r)^r)$$

**using** *RAnd* **by** *blast*

$$\text{have } 12: \vdash (\triangleright(f^r))^r = \triangleleft(f)$$

**using** *RRFirstEqvLast* **by** *blast*

$$\text{have } 13: \vdash (((\triangleright(f^r))^r \wedge (g1^r)^r) = (\triangleleft f \wedge g1)$$

**using** *12 7* **by** *fastforce*

$$\text{have } 14: \vdash (((\triangleright(f^r))^r \wedge (g2^r)^r) = (\triangleleft f \wedge g2)$$

**using** *12 8* **by** *fastforce*

$$\text{have } 15: \vdash (h1;(\triangleleft f \wedge g1) \wedge h2;(\triangleleft f \wedge g2)) =$$

$$((h1^r)^r;(\triangleright(f^r) \wedge g1^r)^r \wedge (h2^r)^r;(\triangleright(f^r) \wedge (g2^r))^r)$$

**using** *14 13 10 11 5 6* **by** (*metis 4 int-eq*)

$$\text{have } 16: \vdash (((\triangleright(f^r) \wedge (g1^r) \wedge (g2^r));((h1^r) \wedge (h2^r)))^r) =$$

$$((h1^r) \wedge (h2^r))^r;((\triangleright(f^r) \wedge (g1^r) \wedge (g2^r))^r$$

**by** (*simp add: RevChop*)

$$\text{have } 17: \vdash ((\triangleright(f^r)) \wedge (g1^r) \wedge (g2^r))^r = ((\triangleright(f^r))^r \wedge (g1^r)^r \wedge (g2^r)^r)$$

**by** (*metis inteq-reflection rev-fun2*)

$$\text{have } 18: \vdash ((\triangleright(f^r))^r \wedge (g1^r)^r \wedge (g2^r)^r) = (\triangleleft f \wedge g1 \wedge g2)$$

**using** *12 7 8* **by** *fastforce*

$$\text{have } 19: \vdash ((h1^r) \wedge (h2^r))^r = (h1 \wedge h2)$$

**using** *RRAnd* **by** *auto*

$$\text{have } 20: \vdash (((h1^r) \wedge (h2^r))^r;((\triangleright(f^r)) \wedge (g1^r) \wedge (g2^r))^r) =$$



$(h1 \wedge h2);(\triangleleft f \wedge g1 \wedge g2)$   
**using** 19 17 18 **using** *ChopEqvChop* **by** (*metis int-eq*)  
**from** 15 4 3 2 16 20 **show** *?thesis* **using** *int-eq* **by** *metis*  
**qed**

**lemma** *LstAndDistrA*:

$\vdash (h;(\triangleleft f \wedge g1) \wedge h;(\triangleleft f \wedge g2)) = h;(\triangleleft f \wedge g1 \wedge g2)$   
**proof** –  
**have** 1:  $\vdash (h;(\triangleleft f \wedge g1) \wedge h;(\triangleleft f \wedge g2)) = (h \wedge h);(\triangleleft f \wedge g1 \wedge g2)$   
**using** *LstAndDistr* **by** *blast*  
**have** 2:  $\vdash (h \wedge h);(\triangleleft f \wedge g1 \wedge g2) = h;(\triangleleft f \wedge g1 \wedge g2)$   
**by** *auto*  
**from** 1 2 **show** *?thesis* **by** *auto*  
**qed**

**lemma** *LstAndDistrB*:

$\vdash (h1;(\triangleleft f \wedge g) \wedge h2;(\triangleleft f \wedge g)) = (h1 \wedge h2);(\triangleleft f \wedge g)$   
**proof** –  
**have** 1:  $\vdash (h1;(\triangleleft f \wedge g) \wedge h2;(\triangleleft f \wedge g)) = (h1 \wedge h2);(\triangleleft f \wedge g \wedge g)$   
**using** *LstAndDistr* **by** *blast*  
**have** 2:  $\vdash (h1 \wedge h2);(\triangleleft f \wedge g \wedge g) = (h1 \wedge h2);(\triangleleft f \wedge g)$   
**by** *auto*  
**from** 1 2 **show** *?thesis* **by** *auto*  
**qed**

**lemma** *LstAndDistrC*:

$\vdash (h1;(\triangleleft f) \wedge h2;(\triangleleft f)) = (h1 \wedge h2);(\triangleleft f)$   
**proof** –  
**have** 1:  $\vdash (h1;(\triangleleft f \wedge \#True) \wedge h2;(\triangleleft f \wedge \#True)) = (h1 \wedge h2);(\triangleleft f \wedge \#True \wedge \#True)$   
**using** *LstAndDistr* **by** *blast*  
**have** 2:  $\vdash (h1 \wedge h2);(\triangleleft f \wedge \#True \wedge \#True) = (h1 \wedge h2);(\triangleleft f)$   
**by** *auto*  
**have** 3:  $\vdash h1;(\triangleleft f \wedge \#True) = h1;(\triangleleft f)$   
**by** *auto*  
**have** 4:  $\vdash h2;(\triangleleft f \wedge \#True) = h2;(\triangleleft f)$   
**by** *auto*  
**from** 1 2 3 4 **show** *?thesis* **by** *auto*  
**qed**

**lemma** *LstAndDistrD*:

$\vdash (\Diamond(\triangleleft f \wedge g1) \wedge \Diamond(\triangleleft f \wedge g2)) = \Diamond(\triangleleft f \wedge g1 \wedge g2)$   
**proof** –  
**have** 1:  $\vdash (\#True;(\triangleleft f \wedge g1) \wedge \#True;(\triangleleft f \wedge g2)) = (\#True \wedge \#True);(\triangleleft f \wedge g1 \wedge g2)$   
**using** *LstAndDistr* **by** *blast*  
**have** 2:  $\vdash (\#True \wedge \#True);(\triangleleft f \wedge g1 \wedge g2) = \Diamond(\triangleleft f \wedge g1 \wedge g2)$   
**by** (*simp add: sometimes-d-def*)  
**have** 3:  $\vdash \#True;(\triangleleft f \wedge g1) = \Diamond(\triangleleft f \wedge g1)$   
**by** (*simp add: sometimes-d-def*)  
**have** 4:  $\vdash \#True;(\triangleleft f \wedge g2) = \Diamond(\triangleleft f \wedge g2)$   
**by** (*simp add: sometimes-d-def*)

from 1 2 3 4 show ?thesis by fastforce  
qed

lemma NotFstChop:

$\vdash (\neg(\triangleright f ; g)) = (\neg(di(\triangleright f)) \vee (\triangleright f ; (\neg g)))$

proof –

have 1:  $\vdash g \longrightarrow \#True$  by auto

hence 2:  $\vdash \triangleright f ; g \longrightarrow \triangleright f ; \#True$  using RightChopImpChop by blast

hence 3:  $\vdash \triangleright f ; g \longrightarrow di(\triangleright f)$  by (simp add: di-d-def)

hence 4:  $\vdash \neg(di(\triangleright f)) \longrightarrow \neg(\triangleright f ; g)$  by auto

have 5:  $\vdash (\triangleright f ; (\neg g) \longrightarrow \neg(\triangleright f ; g)) = ((\triangleright f ; (\neg g)) \wedge (\triangleright f ; g) \longrightarrow \#False)$  by auto

have 6:  $\vdash ((\triangleright f ; (\neg g)) \wedge (\triangleright f ; g)) = \triangleright f ; (\neg g \wedge g)$  using LFstAndDistrC by blast

have 7:  $\vdash \neg(\triangleright f ; (\neg g \wedge g))$  by (simp add: FstChopFalseEqvFalse)

have 8:  $\vdash \triangleright f ; (\neg g) \longrightarrow \neg(\triangleright f ; g)$  using 5 6 7 by fastforce

have 9:  $\vdash \neg(di(\triangleright f)) \vee (\triangleright f ; (\neg g)) \longrightarrow \neg(\triangleright f ; g)$  using 4 8 by fastforce

have 10:  $\vdash di(\triangleright f) \vee \neg(di(\triangleright f))$  by auto

hence 11:  $\vdash (\triangleright f ; \#True) \vee \neg(di(\triangleright f))$  by (simp add: di-d-def)

hence 12:  $\vdash (\triangleright f ; (g \vee \neg g)) \vee \neg(di(\triangleright f))$  by auto

have 13:  $\vdash (\triangleright f ; (g \vee \neg g)) = ((\triangleright f ; g) \vee (\triangleright f ; (\neg g)))$  using ChopOrEqv by fastforce

have 14:  $\vdash ((\triangleright f ; g) \vee (\triangleright f ; (\neg g))) \vee \neg(di(\triangleright f))$  using 12 13 by fastforce

hence 15:  $\vdash \neg(\triangleright f ; g) \longrightarrow \neg(di(\triangleright f)) \vee (\triangleright f ; (\neg g))$  by auto

from 9 15 show ?thesis by fastforce

qed

lemma BsNotFstChop:

$\vdash bs(\neg(\triangleright f ; g)) = (empty \vee \neg(di(\triangleright f)) \vee (\triangleright f ; bs(\neg g)))$

proof –

have 1:  $\vdash bs(\neg(\triangleright f ; g)) = (empty \vee bi(\neg(\triangleright f ; g)); skip)$

by (simp add: bs-d-def)

have 2:  $\vdash (empty \vee bi(\neg(\triangleright f ; g)); skip) = (empty \vee (\neg(di(\triangleright f ; g))); skip)$

by (metis 1 NotDiEqvBiNot int-eq)

have 3:  $\vdash (empty \vee (\neg(di(\triangleright f ; g))); skip) = (empty \vee (\neg((\triangleright f ; g); \#True)); skip)$

by (simp add: di-d-def)

have 4:  $\vdash (\neg((\triangleright f ; g); \#True)); skip = (\neg(\triangleright f ; (g; \#True))); skip$

by (metis ChopAssocB LeftChopEqvChop int-simps(15) inteq-reflection)

hence 5:  $\vdash (empty \vee (\neg((\triangleright f ; g); \#True)); skip) = (empty \vee (\neg(\triangleright f ; (g; \#True))); skip)$

by auto

have 6:  $\vdash (empty \vee (\neg(\triangleright f ; (g; \#True))); skip) = (empty \vee (\neg(\triangleright f ; di(g))); skip)$

by (simp add: di-d-def)

have 7:  $\vdash (empty \vee (\neg(\triangleright f ; di(g))); skip) = (empty \vee \neg(\neg(\neg(\triangleright f ; di(g))); skip))$

by auto

have 8:  $\vdash \neg(\neg(\neg(\triangleright f ; di(g))); skip) = (\neg(empty \vee (\triangleright f ; di(g)); skip))$

using NotNotChopSkip by fastforce

hence 9:  $\vdash (empty \vee \neg(\neg(\neg(\triangleright f ; di(g))); skip)) = (empty \vee \neg(empty \vee (\triangleright f ; di(g)); skip))$

by auto

have 10:  $\vdash (empty \vee \neg(empty \vee (\triangleright f ; di(g)); skip)) = (empty \vee (more \wedge \neg((\triangleright f ; di(g)); skip)))$

by (meson 6 7 9 NotChopSkipEqvMoreAndNotChopSkip Prop04 Prop06)

have 11:  $\vdash (empty \vee (more \wedge \neg((\triangleright f ; di(g)); skip))) = (empty \vee \neg((\triangleright f ; di(g)); skip))$

by (auto simp add: empty-d-def)

have 12:  $\vdash (empty \vee \neg((\triangleright f ; di(g)); skip)) = (empty \vee \neg(\triangleright f ; (di(g); skip)))$

using *ChopAssocB 11* by *fastforce*  
 have 13:  $\vdash (\neg(\triangleright f; (di(g); skip))) = (\neg(\triangleright f; (ds(g))))$   
 using *DsDi* using *RightChopEqvChop* by *fastforce*  
 hence 14:  $\vdash (empty \vee \neg(\triangleright f; (di(g); skip))) = (empty \vee \neg(\triangleright f; (ds(g))))$   
 by *auto*  
 have 15:  $\vdash (empty \vee \neg(\triangleright f; (ds(g)))) = (empty \vee \neg(di(\triangleright f)) \vee (\triangleright f; (\neg(ds\ g))))$   
 using *NotFstChop* by *fastforce*  
 have 16:  $\vdash (\triangleright f; (\neg(ds\ g))) = (\triangleright f; (bs(\neg g)))$   
 using *NotDsEqvBsNot* *RightChopEqvChop* by *blast*  
 hence 17:  $\vdash ((empty \vee \neg(di(\triangleright f))) \vee (\triangleright f; (\neg(ds\ g)))) = ((empty \vee \neg(di(\triangleright f))) \vee (\triangleright f; (bs(\neg g))))$   
 by *auto*  
 show *?thesis*  
 by (*metis 11 12 15 16 2 4 DsDi*  
*NotChopSkipEqvMoreAndNotChopSkip bs-d-def di-d-def inteq-reflection*)  
 qed

lemma *FstFstChopEqvFstChopFst*:

$\vdash \triangleright(\triangleright f; g) = \triangleright f; \triangleright g$   
 proof –  
 have 1:  $\vdash \triangleright(\triangleright f; g) = ((\triangleright f; g) \wedge bs(\neg(\triangleright f; g)))$   
 by (*simp add: first-d-def*)  
 have 2:  $\vdash bs(\neg(\triangleright f; g)) = (empty \vee \neg(di(\triangleright f)) \vee (\triangleright f; bs(\neg g)))$   
 using *BsNotFstChop* by *auto*  
 hence 3:  $\vdash ((\triangleright f; g) \wedge bs(\neg(\triangleright f; g))) = ((\triangleright f; g) \wedge (empty \vee \neg(di(\triangleright f)) \vee (\triangleright f; bs(\neg g))))$   
 by *auto*  
 have 4:  $\vdash ((\triangleright f; g) \wedge (empty \vee \neg(di(\triangleright f)) \vee (\triangleright f; bs(\neg g)))) =$   
 $((\triangleright f; g) \wedge empty) \vee ((\triangleright f; g) \wedge \neg(di(\triangleright f))) \vee ((\triangleright f; g) \wedge (\triangleright f; bs(\neg g)))$   
 by *auto*  
 have 5:  $\vdash \neg((\triangleright f; g) \wedge \neg(di(\triangleright f)))$   
 using *ChopImpDi* by *fastforce*  
 hence 6:  $\vdash (((\triangleright f; g) \wedge empty) \vee ((\triangleright f; g) \wedge \neg(di(\triangleright f))) \vee ((\triangleright f; g) \wedge (\triangleright f; bs(\neg g)))) =$   
 $((\triangleright f; g) \wedge empty) \vee ((\triangleright f; g) \wedge (\triangleright f; bs(\neg g)))$   
 by *auto*  
 have 7:  $\vdash ((\triangleright f; g) \wedge (\triangleright f; (bs(\neg g)))) = ((\triangleright f; (g \wedge (bs(\neg g)))))$   
 using *LFstAndDistrC* by *blast*  
 hence 8:  $\vdash (((\triangleright f; g) \wedge empty) \vee ((\triangleright f; g) \wedge (\triangleright f; (bs(\neg g))))) =$   
 $((\triangleright f; g) \wedge empty) \vee ((\triangleright f; (g \wedge (bs(\neg g)))))$   
 by *auto*  
 have 9:  $\vdash (((\triangleright f; g) \wedge empty) \vee ((\triangleright f; (g \wedge (bs(\neg g))))) = (((\triangleright f; g) \wedge empty) \vee \triangleright f; \triangleright g)$   
 by (*simp add: first-d-def*)  
 have 10:  $\vdash ((\triangleright f; g) \wedge empty) = ((\triangleright f; \triangleright g) \wedge empty)$   
 using *FstChopEmptyEqvFstChopFstEmpty* by *blast*  
 hence 11:  $\vdash (((\triangleright f; g) \wedge empty) \vee \triangleright f; \triangleright g) = (((\triangleright f; \triangleright g) \wedge empty) \vee \triangleright f; \triangleright g)$   
 by *auto*  
 have 12:  $\vdash (((\triangleright f; \triangleright g) \wedge empty) \vee \triangleright f; \triangleright g) = \triangleright f; \triangleright g$   
 by *auto*  
 from 1 3 4 6 8 9 11 12 show *?thesis* by (*metis inteq-reflection*)  
 qed

lemma *FstFixFst*:

$\vdash \triangleright(\triangleright f) = \triangleright f$   
**proof** –  
**have** 1:  $\vdash \triangleright f = (\triangleright f); \text{empty}$  **using** *ChopEmpty* **by** (*metis int-eq*)  
**hence** 2:  $\vdash \triangleright(\triangleright f) = \triangleright((\triangleright f); \text{empty})$  **using** *FstEqvRule* **by** *blast*  
**have** 3:  $\vdash \triangleright((\triangleright f); \text{empty}) = \triangleright f; \triangleright \text{empty}$  **using** *FstFstChopEqvFstChopFst* **by** *auto*  
**have** 4:  $\vdash \triangleright f; \triangleright \text{empty} = \triangleright f; \text{empty}$  **using** *FstEmpty* **using** *RightChopEqvChop* **by** *blast*  
**have** 5:  $\vdash \triangleright f; \text{empty} = \triangleright f$  **using** *ChopEmpty* **by** *blast*  
**from** 2 3 4 5 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *FstCSEqvEmpty*:

$\vdash \triangleright(f^*) = \text{empty}$   
**proof** –  
**have** 1:  $\vdash \triangleright(f^*) = \triangleright(\text{empty} \vee ((f \wedge \text{more}); f^*))$  **using** *ChopstarEqv FstEqvRule* **by** *blast*  
**from** 1 **show** *?thesis* **using** *FstEmptyOrEqvEmpty* **by** *fastforce*  
**qed**

**lemma** *FstIterFixFst*:

$\vdash \text{power}(\triangleright f) n = \triangleright(\text{power}(\triangleright f) n)$   
**proof**  
*(induct n)*  
**case** 0  
**then show** *?case*  
**proof** –  
**have** 1:  $\vdash \text{power}(\triangleright f) 0 = \text{empty}$  **by** *auto*  
**have** 2:  $\vdash \text{empty} = \triangleright \text{empty}$  **using** *FstEmpty* **by** *auto*  
**have** 3:  $\vdash \triangleright \text{empty} = \triangleright(\text{power}(\triangleright f) 0)$  **by** *auto*  
**from** 1 2 3 **show** *?thesis* **by** *auto*  
**qed**  
**next**  
**case** (*Suc n*)  
**then show** *?case*  
**proof** –  
**have** 4:  $\vdash (\text{power}(\triangleright f) (\text{Suc } n)) = (\triangleright f) ; (\text{power}(\triangleright f) n)$   
**by** (*simp*)  
**have** 5:  $\vdash (\triangleright f) ; (\text{power}(\triangleright f) n) = (\triangleright f) ; \triangleright (\text{power}(\triangleright f) n)$   
**using** *RightChopEqvChop Suc.hyps* **by** *blast*  
**have** 6:  $\vdash (\triangleright f) ; \triangleright (\text{power}(\triangleright f) n) = \triangleright(\triangleright f; (\text{power}(\triangleright f) n))$   
**using** *FstFstChopEqvFstChopFst* **by** *fastforce*  
**have** 7:  $\vdash \triangleright(\triangleright f; (\text{power}(\triangleright f) n)) = \triangleright(\text{power}(\triangleright f) (\text{Suc } n))$   
**by** *simp*  
**from** 4 5 6 7 **show** *?thesis* **by** *fastforce*  
**qed**  
**qed**

**lemma** *DsImpNotFst*:

$\vdash \text{ds } f \longrightarrow (\neg(\triangleright f))$   
**proof** –  
**have** 1:  $\vdash (\text{ds } f \wedge \triangleright f) = (\text{ds } f \wedge (f \wedge \text{bs } (\neg f)))$  **by** (*simp add: first-d-def*)  
**have** 2:  $\vdash (\text{ds } f \wedge (f \wedge \text{bs } (\neg f))) = (\text{ds } f \wedge f \wedge \neg(\text{ds } f))$  **using** *NotDsEqvBsNot* **by** *fastforce*

from 1 2 show ?thesis by fastforce  
qed

**lemma** *FstLenAndEqvLenAnd*:

$\vdash \triangleright(\text{len}(k) \wedge f) = (\text{len}(k) \wedge f)$

**proof** –

have 1:  $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow \text{ds}(\text{len}(k))$

using *DsAndImpElimL* by fastforce

hence 2:  $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow (\text{di}(\text{len}(k))); \text{skip}$

using *DsDi* by fastforce

hence 3:  $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow ((\text{len}(k); \# \text{True}); \text{skip})$   
by (*simp add: di-d-def*)

hence 4:  $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow (\text{len}(k); (\# \text{True}; \text{skip}))$   
using *ChopAssocB* by fastforce

hence 5:  $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow (\text{len}(k); (\text{skip}; \# \text{True}))$   
by (*metis SkipTrueEqvTrueSkip inteq-reflection*)

hence 6:  $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow (\text{len}(k); (\text{skip}; \# \text{True})) \wedge \text{len}(k)$   
by *auto*

hence 7:  $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow (\text{len}(k); (\text{skip}; \# \text{True})) \wedge \text{len}(k); \text{empty}$   
using *ChopEmpty* by (*metis int-eq*)

hence 8:  $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow (\text{len}(k); ((\text{skip}; \# \text{True}) \wedge \text{empty}))$   
using *LFixedAndDistrB1* by fastforce

have 9:  $\vdash \neg(\text{len}(k); ((\text{skip}; \# \text{True}) \wedge \text{empty}))$

by (*simp add: empty-d-def more-d-def next-d-def chop-defs Valid-def*)

have 10:  $\vdash \text{len}(k) \wedge f \longrightarrow \neg(\text{ds}(\text{len}(k) \wedge f))$   
using 8 9 by fastforce

hence 11:  $\vdash \text{len}(k) \wedge f \longrightarrow \text{bs}(\neg(\text{len}(k) \wedge f))$   
using *NotDsEqvBsNot* by fastforce

hence 12:  $\vdash \text{len}(k) \wedge f \longrightarrow (\text{len}(k) \wedge f) \wedge \text{bs}(\neg(\text{len}(k) \wedge f))$   
by *auto*

hence 13:  $\vdash \text{len}(k) \wedge f \longrightarrow \triangleright(\text{len}(k) \wedge f)$   
by (*simp add: first-d-def*)

have 14:  $\vdash \triangleright(\text{len}(k) \wedge f) \longrightarrow \text{len}(k) \wedge f$   
by (*auto simp add: first-d-def*)

from 13 14 show ?thesis by fastforce

qed

**lemma** *FstAndElimL*:

$\vdash \triangleright f \longrightarrow f$

by (*auto simp add: first-d-def*)

**lemma** *FstImpNotDiChopSkip*:

$\vdash \triangleright f \longrightarrow \neg(\text{di } f; \text{skip})$

**proof** –

have 1:  $\vdash \triangleright f \longrightarrow \text{bs}(\neg f)$  by (*auto simp add: first-d-def*)

hence 2:  $\vdash \triangleright f \longrightarrow \neg(\text{ds } f)$  using *NotDsEqvBsNot* by fastforce

have 3:  $\vdash \text{ds } f = \text{di } f ; \text{skip}$  using *DsDi* by blast

hence 4:  $\vdash (\neg(\text{ds } f)) = (\neg(\text{di } f; \text{skip}))$  by *auto*

from 2 4 show ?thesis by fastforce

qed

**lemma** *FstImpNotDiChopSkipB*:

$\vdash \triangleright f \longrightarrow \neg(di\ f; skip)$

**proof** –

**have** 1:  $\vdash \triangleright f \longrightarrow bs\ (\neg f)$

**by** (*auto simp add: first-d-def*)

**hence** 2:  $\vdash \triangleright f \longrightarrow \neg(ds\ f)$

**using** *NotDsEqvBsNot* **by** *fastforce*

**have** 3:  $\vdash ds\ f = di\ f ; skip$

**using** *DsDi* **by** *blast*

**have** 4:  $\vdash di\ f ; skip = (f; \# True); skip$

**by** (*simp add: di-d-def*)

**have** 5:  $\vdash (f; \# True); skip = f; (\# True; skip)$

**using** *ChopAssocB* **by** *blast*

**have** 6:  $\vdash f; (\# True; skip) = f; (skip; \# True)$

**using** *SkipTrueEqvTrueSkip* **using** *TrueChopSkipEqvSkipChopTrue RightChopEqvChop* **by** *blast*

**have** 7:  $\vdash f; (skip; \# True) = (f; skip); \# True$

**using** *ChopAssoc* **by** *blast*

**have** 8:  $\vdash (f; skip); \# True = di(f; skip)$

**by** (*simp add: di-d-def*)

**have** 9:  $\vdash (\neg(ds\ f)) = (\neg(di(f; skip)))$

**using** 3 4 5 6 7 8 **by** *fastforce*

**from** 2 9 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *FstImpDiEqv*:

$\vdash \triangleright f \longrightarrow (di\ f = f)$

**proof** –

**have** 1:  $\vdash \triangleright f \longrightarrow \neg(di\ f; skip)$  **using** *FstImpNotDiChopSkip* **by** *blast*

**have** 2:  $\vdash di\ f \longrightarrow f \vee (di\ f; skip)$  **using** *DiEqvOrDiChopSkipB* **by** *fastforce*

**have** 3:  $\vdash \triangleright f \wedge di\ f \longrightarrow (f \vee (di\ f; skip)) \wedge \neg(di\ f; skip)$  **using** 1 2 **by** *fastforce*

**have** 4:  $\vdash ((f \vee (di\ f; skip)) \wedge \neg(di\ f; skip)) = (f \wedge \neg(di\ f; skip))$  **by** *auto*

**have** 5:  $\vdash \triangleright f \wedge di\ f \longrightarrow f \wedge \neg(di\ f; skip)$  **using** 3 4 **by** *fastforce*

**hence** 6:  $\vdash \triangleright f \wedge di\ f \longrightarrow f$  **by** *fastforce*

**hence** 7:  $\vdash \triangleright f \longrightarrow (di\ f \longrightarrow f)$  **using** *FstAndElimL* **by** *fastforce*

**have** 8:  $\vdash f \longrightarrow di\ f$  **using** *DiIntro* **by** *auto*

**hence** 9:  $\vdash \triangleright f \longrightarrow (f \longrightarrow (di\ f))$  **by** *auto*

**from** 7 9 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *FstAndDiFstAndEqvFstAnd*:

$\vdash (\triangleright f \wedge di(\triangleright f \wedge g)) = (\triangleright f \wedge g)$

**proof** –

**have** 1:  $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow \triangleright f$

**by** *auto*

**have** 2:  $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow di(\triangleright f \wedge g)$

**by** *auto*

**have** 3:  $\vdash di(\triangleright f \wedge g) = ((\triangleright f \wedge g) \vee di((\triangleright f \wedge g); skip))$

**using** *DiEqvOrDiChopSkipA* **by** *blast*

**have** 4:  $\vdash di((\triangleright f \wedge g); skip) = ((\triangleright f \wedge g); skip); \# True$

by (*simp add: di-d-def*)  
 have 5:  $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow (\triangleright f \wedge g) \vee ((\triangleright f \wedge g); skip); \# True$   
 using 2 3 4 by *fastforce*  
 have 6:  $\vdash \triangleright f \wedge g \longrightarrow f$   
 using *FstAndElimL* by *fastforce*  
 hence 7:  $\vdash ((\triangleright f \wedge g); skip); \# True \longrightarrow (f; skip); \# True$   
 by (*simp add: LeftChopImpChop*)  
 hence 8:  $\vdash ((\triangleright f \wedge g); skip); \# True \longrightarrow di(f; skip)$   
 by (*simp add: di-d-def*)  
 have 9:  $\vdash \triangleright f \longrightarrow \neg(di(f; skip))$   
 using *FstImpNotDiChopSkipB* by *blast*  
 have 10:  $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow ((\triangleright f \wedge g) \vee di(f; skip))$   
 using 5 8 by *fastforce*  
 have 11:  $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow \neg(di(f; skip)) \wedge ((\triangleright f \wedge g) \vee di(f; skip))$   
 using 9 10 1 by *fastforce*  
 have 12:  $\vdash (\neg(di(f; skip)) \wedge ((\triangleright f \wedge g) \vee di(f; skip))) = (\neg(di(f; skip)) \wedge ((\triangleright f \wedge g)))$   
 by *auto*  
 have 13:  $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow (\triangleright f \wedge g)$   
 using 11 12 by *auto*  
 have 14:  $\vdash (\triangleright f \wedge g) \longrightarrow \triangleright f$   
 by *auto*  
 hence 15:  $\vdash (\triangleright f \wedge g) \longrightarrow di(\triangleright f \wedge g)$   
 using *DiIntro* by *auto*  
 have 16:  $\vdash (\triangleright f \wedge g) \longrightarrow \triangleright f \wedge di(\triangleright f \wedge g)$   
 using 14 15 by *auto*  
 from 13 16 show ?thesis by *fastforce*  
 qed

lemma *FstAndDiImpBsNotAndDi*:

$\vdash (\triangleright f \wedge di\ g) \longrightarrow (bs\ (\neg(di\ f \wedge g)))$

proof –

have 1:  $\vdash (\triangleright f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow ds(di\ f \wedge g)$   
 by (*auto simp add: ds-d-def*)  
 hence 2:  $\vdash (\triangleright f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow ds(di\ f)$   
 using *DsAndImp* by *fastforce*  
 hence 3:  $\vdash (\triangleright f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow di(di\ f); skip$   
 using *DsDi* by *fastforce*  
 hence 4:  $\vdash (\triangleright f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow di\ f; skip$   
 using *DiEqvDiDi* by (*metis int-eq*)  
 hence 5:  $\vdash (\triangleright f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow ds\ f$   
 using *DsDi* by *fastforce*  
 hence 6:  $\vdash (\triangleright f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow \neg(\triangleright f)$   
 using *DsImpNotFst* by *fastforce*  
 from 6 show ?thesis by *auto*  
 qed

lemma *FstFstOrEqvFstOrL*:

$\vdash \triangleright(\triangleright f \vee g) = \triangleright(f \vee g)$

proof –

have 1:  $\vdash \triangleright(f \vee g) = ((f \vee g) \wedge bs\ (\neg(f \vee g)))$

by (*simp add: first-d-def*)  
 have 2:  $\vdash (\neg(f \vee g)) = (\neg f \wedge \neg g)$   
 by *auto*  
 hence 3:  $\vdash bs(\neg(f \vee g)) = bs(\neg f \wedge \neg g)$   
 using *BsEqvRule* by *blast*  
 have 4:  $\vdash bs(\neg f \wedge \neg g) = (bs(\neg f) \wedge bs(\neg g))$   
 using *BsAndEqv* by *fastforce*  
 hence 5:  $\vdash ((f \vee g) \wedge bs(\neg(f \vee g))) = ((f \vee g) \wedge bs(\neg f) \wedge bs(\neg g))$   
 using 4 3 by *fastforce*  
 have 6:  $\vdash ((f \vee g) \wedge bs(\neg f) \wedge bs(\neg g)) =$   
 $((f \wedge bs(\neg f)) \vee (g \wedge bs(\neg f))) \wedge bs(\neg g)$   
 by *auto*  
 have 7:  $\vdash (((f \wedge bs(\neg f)) \vee (g \wedge bs(\neg f))) \wedge bs(\neg g)) =$   
 $((\triangleright f \vee (g \wedge bs(\neg f))) \wedge bs(\neg g))$   
 by (*simp add: first-d-def*)  
 have 8:  $\vdash ((\triangleright f \vee (g \wedge bs(\neg f))) \wedge bs(\neg g)) =$   
 $((\triangleright f \vee g) \wedge (\triangleright f \vee bs(\neg f))) \wedge bs(\neg g)$   
 by *auto*  
 have 9:  $\vdash (((\triangleright f \vee g) \wedge (\triangleright f \vee bs(\neg f))) \wedge bs(\neg g)) =$   
 $((\triangleright f \vee g) \wedge ((f \wedge bs(\neg f)) \vee bs(\neg f))) \wedge bs(\neg g)$   
 by (*simp add: first-d-def*)  
 have 10:  $\vdash (((\triangleright f \vee g) \wedge ((f \wedge bs(\neg f)) \vee bs(\neg f))) \wedge bs(\neg g)) =$   
 $((\triangleright f \vee g) \wedge bs(\neg f) \wedge bs(\neg g))$   
 by *auto*  
 have 11:  $\vdash ((\triangleright f \vee g) \wedge bs(\neg f) \wedge bs(\neg g)) =$   
 $((\triangleright f \vee g) \wedge bs(\neg(\triangleright f)) \wedge bs(\neg g))$   
 using *BsNotFstEqvBsNot* by *fastforce*  
 have 12:  $\vdash ((\triangleright f \vee g) \wedge bs(\neg(\triangleright f)) \wedge bs(\neg g)) =$   
 $((\triangleright f \vee g) \wedge bs(\neg(\triangleright f) \wedge \neg g))$   
 using *BsAndEqv* by *fastforce*  
 have 13:  $\vdash (\neg(\triangleright f) \wedge \neg g) = (\neg(\triangleright f \vee g))$   
 by *auto*  
 hence 14:  $\vdash bs(\neg(\triangleright f) \wedge \neg g) = bs(\neg(\triangleright f \vee g))$   
 using *BsEqvRule* by *blast*  
 hence 15:  $\vdash ((\triangleright f \vee g) \wedge bs(\neg(\triangleright f) \wedge \neg g)) = ((\triangleright f \vee g) \wedge bs(\neg(\triangleright f \vee g)))$   
 by *auto*  
 have 16:  $\vdash ((\triangleright f \vee g) \wedge bs(\neg(\triangleright f \vee g))) = \triangleright(\triangleright f \vee g)$   
 by (*simp add: first-d-def*)  
 from 16 15 12 11 10 9 8 7 6 5 1 show ?thesis by (*metis int-eq*)  
 qed

lemma *FstFstOrEqvFstOrR*:

$\vdash \triangleright(f \vee \triangleright g) = \triangleright(f \vee g)$

proof –

have 1:  $\vdash (f \vee \triangleright g) = (\triangleright g \vee f)$  by *auto*

hence 2:  $\vdash \triangleright(f \vee \triangleright g) = \triangleright(\triangleright g \vee f)$  using *FstEqvRule* by *blast*

have 3:  $\vdash \triangleright(\triangleright g \vee f) = \triangleright(g \vee f)$  using *FstFstOrEqvFstOrL* by *blast*

have 4:  $\vdash (g \vee f) = (f \vee g)$  by *auto*

hence 5:  $\vdash \triangleright(g \vee f) = \triangleright(f \vee g)$  using *FstEqvRule* by *blast*

from 2 3 5 show ?thesis by *fastforce*



qed

**lemma** *FstFstOrEqvFstOr*:

$\vdash \triangleright(\triangleright f \vee \triangleright g) = \triangleright(f \vee g)$

**proof** –

**have** 1:  $\vdash \triangleright(\triangleright f \vee \triangleright g) = \triangleright(f \vee \triangleright g)$  **using** *FstFstOrEqvFstOrL* **by** *blast*

**have** 2:  $\vdash \triangleright(f \vee \triangleright g) = \triangleright(f \vee g)$  **using** *FstFstOrEqvFstOrR* **by** *blast*

**from** 1 2 **show** *?thesis* **by** *fastforce*

qed

**lemma** *FstLenEqvLen*:

$\vdash \triangleright(\text{len}(k)) = \text{len}(k)$

**proof** –

**have** 1:  $\vdash \triangleright(\text{len}(k) \wedge \# \text{True}) = (\text{len}(k) \wedge \# \text{True})$  **using** *FstLenAndEqvLenAnd* **by** *blast*

**have** 2:  $\vdash (\text{len}(k) \wedge \# \text{True}) = \text{len}(k)$  **by** *auto*

**hence** 3:  $\vdash \triangleright(\text{len}(k) \wedge \# \text{True}) = \triangleright(\text{len}(k))$  **using** *FstEqvRule* **by** *blast*

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

qed

**lemma** *FstSkip*:

$\vdash \triangleright \text{skip} = \text{skip}$

**proof** –

**have** 1:  $\vdash \text{skip} = \text{len}(1)$  **using** *LenOneEqvSkip* **by** *fastforce*

**hence** 2:  $\vdash \triangleright \text{skip} = \triangleright(\text{len}(1))$  **using** *FstEqvRule* **by** *blast*

**have** 3:  $\vdash \triangleright(\text{len}(1)) = \text{len}(1)$  **using** *FstLenEqvLen* **by** *blast*

**from** 1 2 3 **show** *?thesis* **using** *LenOneEqvSkip* **by** *fastforce*

qed

**lemma** *HaltStateEqvFstFinState*:

$\vdash \text{halt}(\text{init } w) = \triangleright(\text{fin}(\text{init } w))$

**proof** –

**have** 1:  $\vdash \text{halt}(\text{init } w) = \square(\text{empty} = (\text{init } w))$  **by** (*simp add: halt-d-def*)

**have** 21:  $\vdash (\text{empty} = (\text{init } w)) = (((\text{empty} \longrightarrow (\text{init } w)) \wedge ((\text{init } w) \longrightarrow \text{empty})))$   
**by** *auto*

**hence** 2:  $\vdash \square(\text{empty} = (\text{init } w)) = (\square((\text{empty} \longrightarrow (\text{init } w)) \wedge ((\text{init } w) \longrightarrow \text{empty})))$   
**by** (*simp add: BoxEqvBox*)

**have** 3:  $\vdash (\square((\text{empty} \longrightarrow (\text{init } w)) \wedge ((\text{init } w) \longrightarrow \text{empty}))) =$   
 $(\square((\text{empty} \longrightarrow (\text{init } w))) \wedge \square((\text{init } w) \longrightarrow \text{empty}))$   
**by** (*metis 21 BoxAndBoxEqvBoxRule int-eq*)

**have** 4:  $\vdash ((\text{init } w) \longrightarrow \text{empty}) = (\text{more} \longrightarrow \neg(\text{init } w))$   
**by** (*auto simp add: empty-d-def*)

**hence** 5:  $\vdash \square((\text{init } w) \longrightarrow \text{empty}) = \square(\text{more} \longrightarrow \neg(\text{init } w))$  **using** *BoxEqvBox* **by** *blast*

**have** 6:  $\vdash \square(\text{more} \longrightarrow \neg(\text{init } w)) = \text{bs}(\neg(\text{fin}(\text{init } w)))$  **using** *BoxMoreStateEqvBsFinState* **by** *blast*

**have** 7:  $\vdash \square((\text{empty} \longrightarrow (\text{init } w))) = \text{fin}(\text{init } w)$  **by** (*simp add: fin-d-def*)

**have** 8:  $\vdash (\square((\text{empty} \longrightarrow (\text{init } w))) \wedge \square((\text{init } w) \longrightarrow \text{empty})) =$   
 $(\text{fin}(\text{init } w) \wedge \text{bs}(\neg(\text{fin}(\text{init } w))))$  **using** 5 6 7 **by** *fastforce*

**from** 1 2 3 8 **show** *?thesis* **by** (*metis first-d-def inteq-reflection*)

qed

**lemma** *FstLenEqvLenFst*:

$\vdash \triangleright(\text{len } k ; f) = \text{len } k ; \triangleright f$

**proof** –

**have** 1:  $\vdash \text{len } k ; f = \triangleright(\text{len } k) ; f$  **using** *FstLenEqvLen LeftChopEqvChop* **by** *fastforce*  
**have** 2:  $\vdash \triangleright(\text{len } k ; f) = \triangleright(\triangleright(\text{len } k) ; f)$  **using** 1 *FstEqvRule* **by** *blast*  
**have** 3:  $\vdash \triangleright(\triangleright(\text{len } k) ; f) = \triangleright(\text{len } k) ; \triangleright f$  **using** *FstFstChopEqvFstChopFst* **by** *blast*  
**have** 4:  $\vdash \triangleright(\text{len } k) ; \triangleright f = \text{len } k ; \triangleright f$  **using** *FstLenEqvLen LeftChopEqvChop* **by** *fastforce*  
**from** 2 3 4 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *FstNextEqvNextFst*:

$\vdash \triangleright(\bigcirc f) = \bigcirc(\triangleright f)$

**proof** –

**have** 1:  $\vdash \triangleright(\bigcirc f) = \triangleright(\text{skip} ; f)$  **using** *FstEqvRule* **by** (*simp add: next-d-def*)  
**have** 2:  $\vdash \text{skip} ; f = \triangleright \text{skip} ; f$  **using** *FstSkip* **using** *LeftChopEqvChop* **by** *fastforce*  
**have** 3:  $\vdash \triangleright(\text{skip} ; f) = \triangleright(\triangleright \text{skip} ; f)$  **using** 2 *FstEqvRule LeftChopEqvChop* **by** *blast*  
**have** 4:  $\vdash \triangleright(\triangleright \text{skip} ; f) = \triangleright \text{skip} ; \triangleright f$  **using** 3 *FstFstChopEqvFstChopFst* **by** *blast*  
**have** 5:  $\vdash \triangleright \text{skip} ; \triangleright f = \text{skip} ; \triangleright f$  **using** 4 *FstSkip LeftChopEqvChop* **by** *blast*  
**have** 6:  $\vdash \text{skip} ; \triangleright f = \bigcirc(\triangleright f)$  **by** (*simp add: next-d-def*)  
**from** 1 2 3 4 5 6 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *FstDiamondStateEqvHalt*:

$\vdash \triangleright(\Diamond(\text{init } w)) = \text{halt }(\text{init } w)$

**proof** –

**have** 1:  $\vdash \Diamond(\text{init } w) = \Diamond((\text{init } w) \wedge \# \text{True})$  **by** *simp*  
**have** 2:  $\vdash \text{fin }(\text{init } w) ; \# \text{True} = \Diamond((\text{init } w) \wedge \# \text{True})$  **using** 1 *FinChopEqvDiamond* **by** *blast*  
**have** 3:  $\vdash \text{fin }(\text{init } w) ; \# \text{True} = \text{di }(\text{fin }(\text{init } w))$  **by** (*simp add: di-d-def*)  
**have** 4:  $\vdash \Diamond(\text{init } w) = (\text{di }(\text{fin }(\text{init } w)))$  **using** 1 2 3 **by** *fastforce*  
**have** 5:  $\vdash \triangleright(\Diamond(\text{init } w)) = \triangleright(\text{di }(\text{fin }(\text{init } w)))$  **using** 4 *FstEqvRule* **by** *blast*  
**hence** 6:  $\vdash \triangleright(\Diamond(\text{init } w)) = \triangleright(\text{fin }(\text{init } w))$  **using** *FstDiEqvFst* **by** *fastforce*  
**hence** 7:  $\vdash \triangleright(\Diamond(\text{init } w)) = \text{halt }(\text{init } w)$  **using** *HaltStateEqvFstFinState* **by** *fastforce*  
**from** 7 **show** *?thesis* **by** *simp*

**qed**

**lemma** *FstBoxStateEqvStateAndEmpty*:

$\vdash \triangleright(\Box(\text{init } w)) = ((\text{init } w) \wedge \text{empty})$

**proof** –

**have** 1:  $\vdash ((\text{init } w) \wedge (\Box(\text{init } w))^*) = \Box(\text{init } w)$   
**using** *BoxCSEqvBox* **by** *blast*  
**have** 2:  $\vdash \Box(\text{init } w) = ((\text{init } w) \wedge (\Box(\text{init } w))^*)$   
**using** 1 **by** *auto*  
**hence** 3:  $\vdash \Box(\text{init } w) = ((\text{init } w) \wedge (\Box(\text{init } w))^*)$   
**by** *blast*  
**have** 4:  $\vdash ((\text{init } w) \wedge \text{empty}) ; (\Box(\text{init } w))^* = ((\text{init } w) \wedge (\Box(\text{init } w))^*)$   
**using** *StateAndEmptyChop* **by** *blast*  
**have** 5:  $\vdash ((\text{init } w) \wedge (\Box(\text{init } w))^*) = ((\text{init } w) \wedge \text{empty}) ; (\Box(\text{init } w))^*$   
**using** 4 **by** *fastforce*  
**have** 6:  $\vdash \Box(\text{init } w) = ((\text{init } w) \wedge \text{empty}) ; (\Box(\text{init } w))^*$   
**using** 3 5 **by** *fastforce*  
**have** 7:  $\vdash ((\text{init } w) \wedge \text{empty}) ; (\Box(\text{init } w))^* = \triangleright(\text{init } w) ; (\Box(\text{init } w))^*$

```

using FstState by (metis AndChopCommute int-eq)
have 8:  $\vdash \square (init\ w) = \triangleright (init\ w) ; (\square (init\ w))^*$ 
using 6 7 by fastforce
have 9:  $\vdash \triangleright (\square (init\ w)) = \triangleright (\triangleright (init\ w) ; (\square (init\ w))^*)$ 
using 8 FstEqvRule by blast
have 10:  $\vdash \triangleright (\triangleright (init\ w) ; (\square (init\ w))^*) = \triangleright (init\ w) ; \triangleright ((\square (init\ w))^*)$ 
using FstFstChopEqvFstChopFst by blast
have 11:  $\vdash \triangleright (init\ w) ; \triangleright ((\square (init\ w))^*) = \triangleright (init\ w) ; empty$ 
using RightChopEqvChop FstCSEqvEmpty by blast
have 12:  $\vdash \triangleright (init\ w) ; empty = \triangleright (init\ w)$ 
using RightChopEqvChop ChopEmpty by blast
have 13:  $\vdash \triangleright (init\ w) = ((init\ w) \wedge empty)$ 
using FstState by fastforce
from 9 10 11 12 13 show ?thesis by fastforce
qed

```

**lemma** *FstAndFstStarEqvFst*:

```

 $\vdash (\triangleright f \wedge (\triangleright f)^*) = \triangleright f$ 
proof –
  have 1:  $\vdash (\triangleright f)^* = (empty \vee (\triangleright f);(\triangleright f)^*)$ 
    using CSEqvOrChopCS by fastforce
  have 2:  $\vdash ((\triangleright f)^* \wedge \triangleright f) = ((empty \vee (\triangleright f);(\triangleright f)^*) \wedge \triangleright f)$ 
    using 1 by fastforce
  have 3:  $\vdash ((empty \vee (\triangleright f);(\triangleright f)^*) \wedge \triangleright f) = ((empty \wedge \triangleright f) \vee ((\triangleright f);(\triangleright f)^* \wedge \triangleright f))$ 
    by auto
  have 4:  $\vdash ((\triangleright f)^* \wedge \triangleright f) = ((empty \wedge \triangleright f) \vee ((\triangleright f);(\triangleright f)^* \wedge \triangleright f))$ 
    using 2 3 by fastforce
  have 5:  $\vdash ((\triangleright f);(\triangleright f)^* \wedge \triangleright f) = ((\triangleright f);(\triangleright f)^* \wedge \triangleright f; empty)$ 
    using ChopEmpty by (metis inteq-reflection)
  have 6:  $\vdash ((\triangleright f);(\triangleright f)^* \wedge \triangleright f; empty) = (\triangleright f);((\triangleright f)^* \wedge empty)$ 
    using LFstAndDistrC by blast
  have 7:  $\vdash ((\triangleright f)^* \wedge empty) = empty$ 
    using EmptyImpCS by fastforce
  have 8:  $\vdash (\triangleright f);((\triangleright f)^* \wedge empty) = \triangleright f$ 
    using 7 ChopEmpty by (metis inteq-reflection)
  have 9:  $\vdash ((\triangleright f);(\triangleright f)^* \wedge \triangleright f) = \triangleright f$ 
    using 5 6 8 by fastforce
  have 10:  $\vdash ((\triangleright f)^* \wedge \triangleright f) = ((empty \wedge \triangleright f) \vee \triangleright f)$ 
    using 4 9 by fastforce
  have 11:  $\vdash ((empty \wedge \triangleright f) \vee \triangleright f) = \triangleright f$ 
    by auto
  have 12:  $\vdash ((\triangleright f)^* \wedge \triangleright f) = \triangleright f$ 
    using 10 11 by fastforce
  from 12 show ?thesis by auto
qed

```

**lemma** *HaltStateEqvFstHaltState*:

```

 $\vdash halt(init\ w) = \triangleright (halt(init\ w))$ 
proof –
  have 1:  $\vdash halt\ (init\ w) = \triangleright (fin\ (init\ w))$ 

```

by (simp add: HaltStateEqvFstFinState)  
 have 2:  $\vdash \triangleright (fin (init w)) = \triangleright (\triangleright (fin (init w)))$   
 using FstEqvRule FstFixFst by fastforce  
 have 3:  $\vdash \triangleright (\triangleright (fin (init w))) = \triangleright (halt (init w))$   
 using FstEqvRule HaltStateEqvFstFinState by fastforce  
 from 1 2 3 show ?thesis by fastforce  
 qed

**lemma** DiHaltAndDiHaltAndEqvDiHaltAndAnd:

$\vdash (di(halt (init w) \wedge f) \wedge di(halt (init w) \wedge g)) = di(halt (init w) \wedge f \wedge g)$   
**proof** –  
 have 1:  $\vdash (di(halt (init w) \wedge f) \wedge di(halt (init w) \wedge g)) =$   
 $(di(\triangleright (fin (init w)) \wedge f) \wedge di(\triangleright (fin (init w)) \wedge g))$   
 using HaltStateEqvFstFinState by (metis LFstAndDistrD inteq-reflection)  
 have 2:  $\vdash (di(\triangleright (fin (init w)) \wedge f) \wedge di(\triangleright (fin (init w)) \wedge g)) =$   
 $di(\triangleright (fin (init w)) \wedge f \wedge g)$   
 using LFstAndDistrD by fastforce  
 have 3:  $\vdash di(\triangleright (fin (init w)) \wedge f \wedge g) = di(halt (init w) \wedge f \wedge g)$   
 using HaltStateEqvFstFinState by (metis DiEqvDi int-eq lift-and-com)  
 from 1 2 3 show ?thesis using int-eq by metis  
 qed

**lemma** counter-ex-lhs:

$\vdash ((\triangleright (len(5)) \wedge \triangleright (len(2))) ; (len(5) \vee len(2))) = \#False$   
**proof** –  
 have 1:  $\vdash ((\triangleright (len(5)) \wedge \triangleright (len(2))) ; (len(5) \vee len(2))) =$   
 $(len(5) \wedge len(2)); (len(5) \vee len(2))$   
 by (metis FstLenAndEqvLenAnd FstLenEqvLen LeftChopEqvChop inteq-reflection)  
 have 2:  $\vdash (len(5) \wedge len(2)) = \#False$   
 by (simp add: Valid-def len-defs)  
 have 3:  $\vdash ((len(5) \wedge len(2)); (len(5) \vee len(2))) = (\#False; (len(5) \vee len(2)))$   
 by (simp add: 2 LeftChopEqvChop)  
 have 4:  $\vdash (\#False; (len(5) \vee len(2))) = \#False$   
 by (simp add: Valid-def chop-defs)  
 from 1 3 4 show ?thesis by fastforce  
 qed

**lemma** counter-ex-rhs:

$\vdash ((\triangleright (len(5)) ; (len(5) \vee len(2))) \wedge (\triangleright (len(2)) ; (len(5) \vee len(2)))) = len(7)$   
**proof** –  
 have 1:  $\vdash (\triangleright (len(5)) ; (len(5) \vee len(2))) =$   
 $len(5); (len(5) \vee len(2))$   
 using FstLenEqvLen LeftChopEqvChop by blast  
 have 2:  $\vdash (\triangleright (len(2)) ; (len(5) \vee len(2))) =$   
 $len(2); (len(5) \vee len(2))$   
 using FstLenEqvLen LeftChopEqvChop by blast  
 have 3:  $\vdash len(5); (len(5) \vee len(2)) =$

```

      ((len(5);len(5)) ∨ (len(5);len(2)))
    by (simp add: ChopOrEqv)
  have 4: ⊢ ((len(5);len(5)) ∨ (len(5);len(2))) =
    (len(10) ∨ len(7))
    using LenEqvLenChopLen inteq-reflection by fastforce
  have 5: ⊢ len(2) ;(len(5) ∨ len(2)) =
    ((len(2);len(5)) ∨ (len(2);len(2)))
    by (simp add: ChopOrEqv)
  have 6: ⊢ ((len(2);len(5)) ∨ (len(2);len(2))) =
    (len(7) ∨ len(4))
    using LenEqvLenChopLen inteq-reflection by fastforce
  have 7: ⊢ ((len(10) ∨ len(7)) ∧ (len(7) ∨ len(4))) =
    ((len(7) ∨ len(10)) ∧ (len(7) ∨ len(4)))
    by fastforce
  have 8: ⊢ ((len(7) ∨ len(10)) ∧ (len(7) ∨ len(4))) =
    (len(7) ∨ (len(10) ∧ len(4)))
    by fastforce
  have 9: ⊢ (len(10) ∧ len(4)) = #False
    by (simp add: Valid-def len-defs)
  have 10 : ⊢ (len(7) ∨ (len(10) ∧ len(4))) = len(7)
    using 9 by auto
  have 11: ⊢ ((▷ (len(5)) ; (len(5) ∨ len(2))) ∧ (▷ (len(2)) ; (len(5) ∨ len(2)))) =
    (len(5);(len(5) ∨ len(2)) ∧ len(2) ;(len(5) ∨ len(2)))
    using 1 2 by fastforce
  have 12: ⊢ (len(5);(len(5) ∨ len(2)) ∧ len(2) ;(len(5) ∨ len(2))) = len(7)
    by (metis 10 4 6 7 8 ChopOrEqv inteq-reflection)
  from 11 12 show ?thesis by fastforce
qed

```

end

## 10 Monitors

```

theory Monitor
imports First

```

```

begin

```

The RV monitors language is introduced plus the algebraic properties of the monitor operators.

### 10.1 Syntax

```

datatype ('a :: world) monitor =
  mFIRST-d 'a formula ((FIRST -) [84] 83)
| mUPTO-d 'a monitor 'a monitor ((- UPTO -) [84,84] 83)
| mTHRU-d 'a monitor 'a monitor ((- THRU -) [84,84] 83)
| mTHEN-d 'a monitor 'a monitor ((- THEN -) [84,84] 83)
| mWITH-d 'a monitor 'a formula ((- WITH -) [84,84] 83)

```

```

fun MON :: ('a::world) monitor  $\Rightarrow$  'a formula
where (MON (FIRST f)) = LIFT( $\triangleright$  f)
      | (MON (a UPTO b)) = LIFT( $\triangleright$ ((MON a)  $\vee$  (MON b) ))
      | (MON (a THRU b)) = LIFT( $\triangleright$ (di(MON a)  $\wedge$  di(MON b)))
      | (MON (a THEN b)) = LIFT((MON a);(MON b))
      | (MON (a WITH f)) = LIFT((MON a)  $\wedge$  f)

syntax
-MON :: 'a monitor  $\Rightarrow$  lift (( $\mathcal{M}$  -) [80] 80)

translations
-MON == CONST MON

definition eq-d :: ('a:: world) monitor  $\Rightarrow$  'a monitor  $\Rightarrow$  bool (( -  $\simeq$  -) [84,84] 83)
where
  eq-d a b  $\equiv$  ( $\vdash$  ( $\mathcal{M}$  a) = ( $\mathcal{M}$  b))

lemma MonEqRefl:
  a  $\simeq$  a
by (simp add: eq-d-def)

lemma MonEqSym:
  assumes a  $\simeq$  b
  shows b  $\simeq$  a
using assms by (metis eq-d-def inteq-reflection)

lemma MonEqTrans:
  assumes a  $\simeq$  b
      b  $\simeq$  c
  shows a  $\simeq$  c
using assms(1) assms(2) by (metis eq-d-def inteq-reflection)

lemma MonEq:
  (a  $\simeq$  b) = ( $\vdash$  ( $\mathcal{M}$  a) = ( $\mathcal{M}$  b))
by (simp add: eq-d-def)

lemma MonEqSubstWith:
  assumes a  $\simeq$  b
  shows (a WITH f)  $\simeq$  (b WITH f)
using assms by (metis MON.simps(5) eq-d-def inteq-reflection lift-and-com)

lemma MonEqSubstThen:
  assumes a1  $\simeq$  b1
      a2  $\simeq$  b2
  shows (a1 THEN a2)  $\simeq$  (b1 THEN b2)
using assms(1) assms(2) by (simp add: ChopEqvChop eq-d-def)

lemma MonEqSubstUpto:

```

**assumes**  $a1 \simeq b1$   
 $a2 \simeq b2$   
**shows**  $(a1 \text{ UPTO } a2) \simeq (b1 \text{ UPTO } b2)$   
**using**  $assms(1) \text{ } assms(2)$  **by**  $(metis \text{ (mono-tags, lifting) } MON.simps(2) \text{ eq-d-def int-eq MonEqRefl})$

**lemma** *MonEqSubstThru*:

**assumes**  $a1 \simeq b1$   
 $a2 \simeq b2$   
**shows**  $(a1 \text{ THRU } a2) \simeq (b1 \text{ THRU } b2)$   
**using**  $assms(1) \text{ } assms(2)$  **by**  $(metis \text{ (mono-tags, lifting) } MON.simps(3) \text{ eq-d-def int-eq MonEqRefl})$

## 10.2 Derived Monitors

**definition** *HALT-d* ::  $('a :: world) \text{ formula} \Rightarrow 'a \text{ monitor}$   
**where**  $HALT-d \ w \equiv FIRST(LIFT(fn \ (init \ w)))$

**definition** *LEN-d* ::  $nat \Rightarrow ('a :: world) \text{ monitor}$   
**where**  
 $LEN-d \ k \equiv FIRST \ (LIFT(len \ k))$

**definition** *EMPTY-d* ::  $('a :: world) \text{ monitor}$   
**where**  
 $EMPTY-d \equiv FIRST \ (LIFT(empty))$

**definition** *SKIP-d* ::  $('a :: world) \text{ monitor}$   
**where**  
 $SKIP-d \equiv FIRST \ (LIFT(skip))$

**syntax**  
 $-HALT-d \ :: \ lift \Rightarrow 'a \text{ monitor} \quad ((HALT \ -) \ [84] \ 83)$   
 $-LEN-d \ :: \ nat \Rightarrow 'a \text{ monitor} \quad ((LEN \ -) \ [84] \ 83)$   
 $-EMPTY-d \ :: 'a \text{ monitor} \quad ((EMPTY) \ )$   
 $-SKIP-d \ :: 'a \text{ monitor} \quad ((SKIP))$

**syntax** (*ASCII*)  
 $-HALT-d \ :: \ lift \Rightarrow 'a \text{ monitor} \quad ((HALT \ -) \ [84] \ 83)$   
 $-LEN-d \ :: \ nat \Rightarrow 'a \text{ monitor} \quad ((LEN \ -) \ [84] \ 83)$   
 $-EMPTY-d \ :: 'a \text{ monitor} \quad ((EMPTY))$   
 $-SKIP-d \ :: 'a \text{ monitor} \quad ((SKIP))$

### translations

$-HALT-d \ \Rightarrow \ CONST \ HALT-d$   
 $-LEN-d \ \Rightarrow \ CONST \ LEN-d$   
 $-EMPTY-d \ \Rightarrow \ CONST \ EMPTY-d$   
 $-SKIP-d \ \Rightarrow \ CONST \ SKIP-d$

**definition** *GUARD-d* ::  $('a :: world) \text{ formula} \Rightarrow 'a \text{ monitor}$   
**where**

$GUARD-d\ w \equiv (EMPTY\ WITH\ LIFT(init\ w))$

**primrec**  $TIMES-d :: ('a :: world)\ monitor \Rightarrow nat \Rightarrow 'a\ monitor$

**where**

$TIMES-0 : TIMES-d\ a\ 0 = EMPTY$

|  $TIMES-Suc: TIMES-d\ a\ (Suc\ k) = (a\ THEN\ (TIMES-d\ a\ k))$

**syntax**

$-GUARD-d :: lift \Rightarrow 'a\ monitor \quad ((GUARD\ -)\ [84]\ 83)$

$-TIMES-d :: ['a\ monitor, nat] \Rightarrow 'a\ monitor \quad ((- \ TIMES\ -)\ [84,84]\ 83)$

**syntax** (ASCII)

$-GUARD-d :: lift \Rightarrow 'a\ monitor \quad ((GUARD\ -)\ [84]\ 83)$

$-TIMES-d :: ['a\ monitor, nat] \Rightarrow 'a\ monitor \quad ((- \ TIMES\ -)\ [84,84]\ 83)$

**translations**

$-GUARD-d \equiv CONST\ GUARD-d$

$-TIMES-d \equiv CONST\ TIMES-d$

**definition**  $FAIL-d :: ('a :: world)\ monitor$

**where**

$FAIL-d \equiv GUARD\ (\#False)$

**definition**  $ALWAYS-d :: ('a :: world)\ monitor \Rightarrow 'a\ formula \Rightarrow 'a\ monitor$

**where**

$ALWAYS-d\ a\ w \equiv (a\ WITH\ LIFT((bi\ (fn\ (init\ w))))))$

**definition**  $SOMETIME-d :: ('a :: world)\ monitor \Rightarrow 'a\ formula \Rightarrow 'a\ monitor$

**where**

$SOMETIME-d\ a\ w \equiv (a\ WITH\ LIFT((di\ (fn\ (init\ w))))))$

**definition**  $LIMIT-d :: ('a :: world)\ formula \Rightarrow 'a\ formula$

**where**

$LIMIT-d\ f \equiv LIFT(bs\ (\neg\ f))$

**definition**  $UNTIL-d :: ('a :: world)\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ monitor$

**where**

$UNTIL-d\ w1\ w2 \equiv (HALT\ w2)\ WITH\ (LIFT(bm\ w1))$

**syntax**

$-FAIL-d :: 'a\ monitor \quad (FAIL)$

$-ALWAYS-d :: ['a\ monitor, lift] \Rightarrow 'a\ monitor \quad ((- \ ALWAYS\ -)\ [84,84]\ 83)$

$-SOMETIME-d :: ['a\ monitor, lift] \Rightarrow 'a\ monitor \quad ((- \ SOMETIME\ -)\ [84,84]\ 83)$

$-LIMIT-d :: lift \Rightarrow lift \quad ((Limit\ -)\ [84]\ 83)$

$-UNTIL-d :: [lift, lift] \Rightarrow 'a\ monitor \quad ((- \ UNTIL\ -)\ [84,84]\ 83)$

**syntax** (ASCII)

$-FAIL-d :: 'a\ monitor \quad (FAIL)$

$-ALWAYS-d :: ['a\ monitor, lift] \Rightarrow 'a\ monitor \quad ((- \ ALWAYS\ -)\ [84,84]\ 83)$



$-SOMETIME-d :: ['a \text{ monitor}, lift] \Rightarrow 'a \text{ monitor } ((- \text{ SOMETIME } -) [84,84] \ 83)$   
 $-LIMIT-d :: lift \Rightarrow lift \quad ((Limit \ -) [84] \ 83)$   
 $-UNTIL-d :: [lift, lift] \Rightarrow 'a \text{ monitor } \quad ((- \text{ UNTIL } -) [84,84] \ 83)$

**translations**

$-FAIL-d \quad \Rightarrow \text{CONST FAIL-}d$   
 $-ALWAYS-d \quad \Rightarrow \text{CONST ALWAYS-}d$   
 $-SOMETIME-d \quad \Rightarrow \text{CONST SOMETIME-}d$   
 $-LIMIT-d \quad \Rightarrow \text{CONST LIMIT-}d$   
 $-UNTIL-d \quad \Rightarrow \text{CONST UNTIL-}d$

**definition**  $WITHIN-d :: ('a :: world) \text{ monitor} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ monitor}$   
**where**

$WITHIN-d \ a \ f \equiv (a \text{ WITH LIFT}(Limit \ f))$

**syntax**

$-WITHIN-d :: ['a \text{ monitor}, lift] \Rightarrow 'a \text{ monitor } ((- \text{ WITHIN } -) [84,84] \ 83)$

**syntax (ASCII)**

$-WITHIN-d :: ['a \text{ monitor}, lift] \Rightarrow 'a \text{ monitor } ((- \text{ WITHIN } -) [84,84] \ 83)$

**translations**

$-WITHIN-d \quad \Rightarrow \text{CONST WITHIN-}d$

**definition**  $AND-d :: ('a :: world) \text{ monitor} \Rightarrow 'a \text{ monitor} \Rightarrow 'a \text{ monitor}$   
**where**

$AND-d \ a \ b \equiv (a \text{ WITH LIFT}(\mathcal{M} \ b))$

**definition**  $ITERATE-d :: ('a :: world) \text{ monitor} \Rightarrow 'a \text{ monitor} \Rightarrow 'a \text{ monitor}$   
**where**

$ITERATE-d \ a \ b \equiv (a \text{ WITH } (LIFT \ (\mathcal{M} \ b)^*))$

**syntax**

$-AND-d \quad :: ['a \text{ monitor}, 'a \text{ monitor}] \Rightarrow 'a \text{ monitor } ((- \text{ AND } -) [84,84] \ 83)$   
 $-ITERATE-d :: ['a \text{ monitor}, 'a \text{ monitor}] \Rightarrow 'a \text{ monitor } ((- \text{ ITERATE } -) [84,84] \ 83)$

**syntax (ASCII)**

$-AND-d \quad :: ['a \text{ monitor}, 'a \text{ monitor}] \Rightarrow 'a \text{ monitor } ((- \text{ AND } -) [84,84] \ 83)$   
 $-ITERATE-d :: ['a \text{ monitor}, 'a \text{ monitor}] \Rightarrow 'a \text{ monitor } ((- \text{ ITERATE } -) [84,84] \ 83)$

**translations**

$-AND-d \quad \Rightarrow \text{CONST AND-}d$   
 $-ITERATE-d \Rightarrow \text{CONST ITERATE-}d$

**definition**  $STAR-d :: ('a :: world) \text{ monitor} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ monitor}$   
**where**

$STAR\text{-}d\ a\ f \equiv ((FIRST\ LIFT(\Diamond\ f))\ ITERATE\ (a))$

**definition**  $REPEAT\text{-}d :: ('a :: world)\ monitor \Rightarrow 'a\ formula \Rightarrow 'a\ monitor$

**where**

$REPEAT\text{-}d\ a\ w \equiv ((HALT\ w)\ ITERATE\ (a\ WITH\ LIFT(keep(\neg\ (init\ w))))))$

**syntax**

$\text{-}STAR\text{-}d :: ['a\ monitor, lift] \Rightarrow 'a\ monitor\ ((\text{-}\ STAR\ \text{-})\ [84,84]\ 83)$

$\text{-}REPEAT\text{-}d :: ['a\ monitor, lift] \Rightarrow 'a\ monitor\ ((\text{-}\ REPEATUNTIL\ \text{-})\ [84,84]\ 83)$

**syntax** (*ASCII*)

$\text{-}STAR\text{-}d :: ['a\ monitor, lift] \Rightarrow 'a\ monitor\ ((\text{-}\ STAR\ \text{-})\ [84,84]\ 83)$

$\text{-}REPEAT\text{-}d :: ['a\ monitor, lift] \Rightarrow 'a\ monitor\ ((\text{-}\ REPEATUNTIL\ \text{-})\ [84,84]\ 83)$

**translations**

$\text{-}STAR\text{-}d \equiv CONST\ STAR\text{-}d$

$\text{-}REPEAT\text{-}d \equiv CONST\ REPEAT\text{-}d$

### 10.3 Monitor Laws

**lemma**  $MFixFst$ :

$\vdash (\mathcal{M}\ a) = \triangleright (\mathcal{M}\ a)$

**proof**

(*induct a*)

**case** ( $mFIRST\text{-}d\ x$ )

**then show**  $?case$

**proof** –

**have**  $1: \vdash (\mathcal{M}\ (FIRST\ x)) = \triangleright x$  **by** *simp*

**have**  $2: \vdash \triangleright x = \triangleright (\triangleright x)$  **using**  $FstFixFst$  **by** *fastforce*

**have**  $3: \vdash \triangleright (\triangleright x) = \triangleright (\mathcal{M}\ (FIRST\ x))$  **by** *simp*

**from**  $1\ 2\ 3$  **show**  $?thesis$  **by** *fastforce*

**qed**

**next**

**case** ( $mUPTO\text{-}d\ a1\ a2$ )

**then show**  $?case$

**proof** –

**have**  $1: \vdash (\mathcal{M}\ (a1\ UPTO\ a2)) = \triangleright (\mathcal{M}\ a1) \vee (\mathcal{M}\ a2)$

**by** (*simp*)

**have**  $2: \vdash \triangleright (\mathcal{M}\ a1) \vee (\mathcal{M}\ a2) = \triangleright (\triangleright ((\mathcal{M}\ a1) \vee (\mathcal{M}\ a2)))$

**using**  $FstFixFst$  **by** *fastforce*

**have**  $3: \vdash \triangleright (\triangleright ((\mathcal{M}\ a1) \vee (\mathcal{M}\ a2))) = \triangleright (\mathcal{M}\ (a1\ UPTO\ a2))$

**using**  $2$  **by** *simp*

**from**  $1\ 2\ 3$  **show**  $?thesis$  **by** *fastforce*

**qed**

**next**

**case** ( $mTHRU\text{-}d\ a1\ a2$ )

**then show**  $?case$

**proof** –

**have**  $1: \vdash (\mathcal{M}\ (a1\ THRU\ a2)) = \triangleright (di\ (\mathcal{M}\ a1) \wedge di(\mathcal{M}\ a2))$

**by** (*simp*)

```

have 2:  $\vdash \triangleright (di(\mathcal{M} a1) \wedge di(\mathcal{M} a2)) = \triangleright(\triangleright(di(\mathcal{M} a1) \wedge di(\mathcal{M} a2)))$ 
  using FstFixFst by fastforce
have 3:  $\vdash \triangleright(\triangleright(di(\mathcal{M} a1) \wedge di(\mathcal{M} a2))) = \triangleright(\mathcal{M} (a1 \text{ THRU } a2))$ 
  using 2 by simp
from 1 2 3 show ?thesis by fastforce
qed
next
case (mTHEN-d a1 a2)
then show ?case
proof -
have 1:  $\vdash (\mathcal{M} (a1 \text{ THEN } a2)) = (\mathcal{M} a1) ; (\mathcal{M} a2)$ 
  by (simp)
have 2:  $\vdash (\mathcal{M} a1) ; (\mathcal{M} a2) = \triangleright(\mathcal{M} a1) ; \triangleright(\mathcal{M} a2)$ 
  using ChopEqvChop mTHEN-d.hyps(1) mTHEN-d.hyps(2) by blast
have 3:  $\vdash \triangleright(\mathcal{M} a1) ; \triangleright(\mathcal{M} a2) = \triangleright(\triangleright(\mathcal{M} a1) ; (\mathcal{M} a2))$ 
  using FstFstChopEqvFstChopFst by fastforce
have 4:  $\vdash \triangleright(\triangleright(\mathcal{M} a1) ; (\mathcal{M} a2)) = \triangleright((\mathcal{M} a1) ; (\mathcal{M} a2))$ 
  using FstEqvRule LeftChopEqvChop mTHEN-d.hyps(1) by (metis inteq-reflection)
have 5:  $\vdash \triangleright((\mathcal{M} a1) ; (\mathcal{M} a2)) = \triangleright(\mathcal{M} (a1 \text{ THEN } a2))$ 
  using 4 by simp
from 1 2 3 4 5 show ?thesis by fastforce
qed
next
case (mWITH-d a x2)
then show ?case
proof -
have 1:  $\vdash (\mathcal{M} (a \text{ WITH } x2)) = ((\mathcal{M} a) \wedge (x2))$ 
  by (simp)
have 2:  $\vdash ((\mathcal{M} a) \wedge (x2)) = \triangleright(\mathcal{M} a) \wedge (x2)$ 
  using mWITH-d.hyps by fastforce
have 3:  $\vdash \triangleright(\mathcal{M} a) \wedge (x2) = \triangleright(\triangleright(\mathcal{M} a) \wedge (x2))$ 
  using FstFstAndEqvFstAnd by fastforce
have 4:  $\vdash \triangleright(\triangleright(\mathcal{M} a) \wedge (x2)) = \triangleright((\mathcal{M} a) \wedge (x2))$ 
  using 2 FstEqvRule by fastforce
have 5:  $\vdash \triangleright((\mathcal{M} a) \wedge (x2)) = \triangleright(\mathcal{M} (a \text{ WITH } x2))$ 
  using 4 by simp
from 1 2 3 4 5 show ?thesis by (metis inteq-reflection)
qed
qed

lemma MGuardFalseEqvFalse:
 $\vdash \mathcal{M}(GUARD \#False) = \#False$ 
proof -
have 1:  $\vdash \mathcal{M}(GUARD \#False) = \mathcal{M}(EMPTY \text{ WITH } LIFT(init \#False))$  by (simp add: GUARD-d-def)
have 2:  $\vdash \mathcal{M}(EMPTY \text{ WITH } LIFT(init \#False)) = (\mathcal{M}(EMPTY) \wedge (init \#False))$  by (simp)
have 3:  $\vdash \#False = (init \#False)$  by (simp add: init-defs Valid-def)
have 4:  $\vdash (\mathcal{M}(EMPTY) \wedge (init \#False)) = (\mathcal{M}(EMPTY) \wedge \#False)$  using 3 by auto
have 5:  $\vdash (\mathcal{M}(EMPTY) \wedge \#False) = \#False$  by simp
have 6:  $\vdash (\mathcal{M}(EMPTY) \wedge (init \#False)) = \#False$  using 4 5 by simp
have 7:  $\vdash \mathcal{M}(EMPTY \text{ WITH } LIFT(init \#False)) = \#False$  using 2 6 by fastforce

```

have 8:  $\vdash \mathcal{M}(\text{GUARD } \#False) = \#False$  using 1 7 by fastforce  
 from 8 show ?thesis by auto  
 qed

lemma *MFirstFalseEqvFalse*:

$\vdash \mathcal{M}(\text{FIRST LIFT } \#False) = \#False$

proof –

have 1:  $\vdash \mathcal{M}(\text{FIRST LIFT } \#False) = \triangleright \#False$  by (simp )

have 2:  $\vdash \mathcal{M}(\text{FIRST LIFT } \#False) = \#False$  using FstFalse by fastforce

from 2 show ?thesis by auto

qed

lemma *MFailAlt*:

$\vdash \mathcal{M} \text{ FAIL} = \#False$

proof –

have 1:  $\vdash \mathcal{M} \text{ FAIL} = \mathcal{M} (\text{GUARD } (\#False))$  by (simp add: FAIL-d-def)

have 2:  $\vdash \mathcal{M}(\text{GUARD } (\#False)) = \#False$  using MGuardFalseEqvFalse by auto

from 1 2 show ?thesis by fastforce

qed

lemma *MFailEqvFirstFalseWithinEmpty*:

$\text{FAIL} \simeq ((\text{FIRST LIFT } \#False) \text{ WITHIN } \text{empty})$

proof –

have 1:  $\vdash \mathcal{M} ( (\text{FIRST LIFT } \#False) \text{ WITHIN } ( \text{empty} ) ) =$   
 $\mathcal{M}((\text{FIRST LIFT } \#False) \text{ WITH LIFT}(\text{Limit empty}) )$

by (simp add: WITHIN-d-def)

have 2:  $\vdash \mathcal{M}((\text{FIRST LIFT } \#False) \text{ WITH LIFT}(\text{Limit empty}) ) =$   
 $(\mathcal{M}(\text{FIRST LIFT } \#False) \wedge (\text{Limit empty}) )$

by (simp )

have 3:  $\vdash \mathcal{M}((\text{FIRST LIFT } \#False) \text{ WITH LIFT}(\text{Limit empty}) ) = \#False$   
 using MFirstFalseEqvFalse by auto

have 4:  $\vdash \mathcal{M} ( (\text{FIRST LIFT } \#False) \text{ WITHIN } ( \text{empty} ) ) = \#False$   
 using 1 3 by fastforce

have 5:  $\vdash \mathcal{M} ( \text{FAIL} ) = \#False$

using MFailAlt by simp

from 4 5 show ?thesis using MonEq by (metis int-eq)

qed

lemma *MEmptyAlt*:

$\vdash \mathcal{M} \text{ EMPTY} = \text{empty}$

proof –

have 1:  $\vdash \mathcal{M} ( \text{EMPTY} ) = \mathcal{M} ( (\text{FIRST LIFT } \text{empty}) )$  by (simp add: EMPTY-d-def)

have 2:  $\vdash \mathcal{M} ( (\text{FIRST LIFT } \text{empty}) ) = \triangleright \text{empty}$  by (simp)

have 3:  $\vdash \triangleright \text{empty} = \text{empty}$  using FstEmpty by auto

from 1 2 3 show ?thesis by fastforce

qed

lemma *MSkipAlt*:

$\vdash \mathcal{M} \text{ SKIP} = \text{skip}$

proof –

**have** 1:  $\vdash \mathcal{M} \text{ SKIP} = \mathcal{M} (\text{FIRST LIFT skip})$  **by** (*simp add: SKIP-d-def*)  
**have** 2:  $\vdash \mathcal{M} (\text{FIRST LIFT skip}) = \triangleright \text{skip}$  **by** (*simp*)  
**have** 3:  $\vdash \triangleright \text{skip} = \text{skip}$  **using** *FstSkip* **by** *simp*  
**from** 1 2 3 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *MGuardAlt*:

$\vdash \mathcal{M} (\text{GUARD}(w)) = (\text{empty} \wedge \text{init } w)$

**proof** –

**have** 1:  $\vdash \mathcal{M}(\text{GUARD}(w)) = \mathcal{M}(\text{EMPTY WITH } (\text{LIFT } (\text{init } w)))$  **by** (*simp add: GUARD-d-def*)  
**have** 2:  $\vdash \mathcal{M}(\text{EMPTY WITH } (\text{LIFT } (\text{init } w))) = (\mathcal{M}(\text{EMPTY}) \wedge (\text{init } w))$  **by** (*simp*)  
**have** 3:  $\vdash (\mathcal{M}(\text{EMPTY}) \wedge (\text{init } w)) = (\text{empty} \wedge (\text{init } w))$  **using** *MEmptyAlt* **by** *fastforce*  
**have** 4:  $\vdash (\text{empty} \wedge (\text{init } w)) = (\text{empty} \wedge \text{init } w)$  **by** *simp*  
**from** 1 2 3 4 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *MLengthAlt*:

$\vdash \mathcal{M} (\text{LEN}(k)) = \text{len}(k)$

**proof** –

**have** 1:  $\vdash \mathcal{M}(\text{LEN}(k)) = \mathcal{M}(\text{FIRST LIFT}(\text{len}(k)))$  **by** (*simp add: LEN-d-def*)  
**have** 2:  $\vdash \mathcal{M}(\text{FIRST LIFT}(\text{len}(k))) = \triangleright(\text{len}(k))$  **by** (*simp*)  
**have** 3:  $\vdash \triangleright(\text{len}(k)) = \text{len}(k)$  **using** *FstLenEqvLen* **by** *blast*  
**from** 1 2 3 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *MAlwaysAlt*:

$\vdash \mathcal{M}(a \text{ ALWAYS } w) = (\mathcal{M}(a) \wedge \square (\text{init } w))$

**proof** –

**have** 1:  $\vdash \mathcal{M}(a \text{ ALWAYS } w) = \mathcal{M}(a \text{ WITH LIFT}(bi \text{ (fin (init } w)))))$   
**by** (*simp add: ALWAYS-d-def*)  
**have** 2:  $\vdash \mathcal{M}(a \text{ WITH LIFT}(bi \text{ (fin (init } w))))) = (\mathcal{M}(a) \wedge (bi \text{ (fin (init } w)))))$   
**by** (*simp*)  
**have** 3:  $\vdash (\mathcal{M}(a) \wedge (bi \text{ (fin (init } w))))) = (\mathcal{M}(a) \wedge \square (\text{init } w))$   
**using** *BoxStateEqvBiFinState* **by** *fastforce*  
**from** 1 2 3 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *MSometimeAlt*:

$\vdash \mathcal{M}(a \text{ SOMETIME } w) = (\mathcal{M}(a) \wedge \diamond (\text{init } w))$

**proof** –

**have** 1:  $\vdash \mathcal{M}(a \text{ SOMETIME } w) = \mathcal{M}(a \text{ WITH LIFT}(di \text{ (fin (init } w)))))$   
**by** (*simp add: SOMETIME-d-def*)  
**have** 2:  $\vdash \mathcal{M}(a \text{ WITH LIFT}(di \text{ (fin (init } w))))) = (\mathcal{M}(a) \wedge (di \text{ (fin (init } w)))))$   
**by** (*simp*)  
**have** 3:  $\vdash \mathcal{M}(a \text{ WITH LIFT}(di \text{ (fin (init } w))))) = (\mathcal{M}(a) \wedge \diamond (\text{init } w))$   
**using** *DiamondStateEqvDiFinState* **by** *fastforce*  
**from** 1 2 3 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *MWithinAlt*:

$\vdash \mathcal{M}(a \text{ WITHIN } f) = (\mathcal{M}(a) \wedge (bs \ (\neg f)))$

**proof** –

**have** 1:  $\vdash \mathcal{M}(a \text{ WITHIN } f) = \mathcal{M}(a \text{ WITH LIFT}(bs \ (\neg f)))$

**by** (*simp add: WITHIN-d-def LIMIT-d-def*)

**have** 2:  $\vdash \mathcal{M}(a \text{ WITH LIFT}(bs \ (\neg f))) = (\mathcal{M}(a) \wedge (bs \ (\neg f)))$

**by** (*simp*)

**from** 1 2 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *MTimesAlt*:

$\vdash \mathcal{M}(a \text{ TIMES } k) = \text{power } (\mathcal{M}(a)) \ k$

**proof**

(*induct k*)

**case** 0

**then show** *?case*

**proof** –

**have** 1:  $\vdash \mathcal{M}(a \text{ TIMES } 0) = \mathcal{M} \text{ EMPTY}$  **by** *simp*

**have** 2:  $\vdash \mathcal{M} \text{ EMPTY} = \text{empty}$  **using** *MEEmptyAlt* **by** *simp*

**have** 3:  $\vdash \text{empty} = \text{power } (\mathcal{M} \ a) \ 0$  **by** *simp*

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

**qed**

**next**

**case** (*Suc k*)

**then show** *?case*

**proof** –

**have** 1:  $\vdash \mathcal{M}(a \text{ TIMES } \text{Suc } k) = \mathcal{M}(a \text{ THEN } (a \text{ TIMES } k))$

**by** *simp*

**have** 2:  $\vdash \mathcal{M}(a \text{ THEN } (a \text{ TIMES } k)) = (\mathcal{M} \ a);(\mathcal{M} \ (a \text{ TIMES } k))$

**by** (*simp*)

**have** 3:  $\vdash (\mathcal{M} \ a);(\mathcal{M}(a \text{ TIMES } k)) = (\mathcal{M} \ a);(\text{power } (\mathcal{M} \ a) \ k)$

**using** *RightChopEqvChop Suc.hyps* **by** *blast*

**have** 4:  $\vdash (\mathcal{M} \ a);(\text{power } (\mathcal{M} \ a) \ k) = \text{power } (\mathcal{M} \ a) \ (\text{Suc } k)$

**by** *simp*

**from** 1 2 3 4 **show** *?thesis* **by** *fastforce*

**qed**

**qed**

**lemma** *MUptoAlt*:

$\vdash \mathcal{M}(a \text{ UPTO } b) = ((\mathcal{M} \ a) \wedge bi \ (\neg(\mathcal{M} \ b))) \vee ((\mathcal{M} \ b) \wedge bi \ (\neg(\mathcal{M} \ a))) \vee ((\mathcal{M} \ a) \wedge (\mathcal{M} \ b))$

**proof** –

**have** 1:  $\vdash \mathcal{M}(a \text{ UPTO } b) = \triangleright((\mathcal{M} \ a) \vee (\mathcal{M} \ b))$

**by** (*simp*)

**have** 2:  $\vdash \triangleright((\mathcal{M} \ a) \vee (\mathcal{M} \ b)) = ((\triangleright(\mathcal{M} \ a) \wedge (bs \ (\neg(\mathcal{M} \ b)))) \vee (\triangleright(\mathcal{M} \ b) \wedge (bs \ (\neg(\mathcal{M} \ a)))))$

**using** *FstWithOrEqv* **by** *blast*

**have** 3:  $\vdash ((\triangleright(\mathcal{M} \ a) \wedge (bs \ (\neg(\mathcal{M} \ b)))) \vee (\triangleright(\mathcal{M} \ b) \wedge (bs \ (\neg(\mathcal{M} \ a))))) =$

$((\mathcal{M} \ a) \wedge ((\mathcal{M} \ b) \vee \neg(\mathcal{M} \ b)) \wedge (bs \ (\neg(\mathcal{M} \ b)))) \vee$

$((\mathcal{M} \ b) \wedge ((\mathcal{M} \ a) \vee \neg(\mathcal{M} \ a)) \wedge (bs \ (\neg(\mathcal{M} \ a))))$

**using** *MFixFst* **by** *fastforce*

**have** 4:  $\vdash (((\mathcal{M} \ a) \wedge ((\mathcal{M} \ b) \vee \neg(\mathcal{M} \ b)) \wedge (bs \ (\neg(\mathcal{M} \ b)))) \vee$

$$((\mathcal{M} \ b) \wedge ((\mathcal{M} \ a) \vee \neg(\mathcal{M} \ a)) \wedge (bs \ (\neg(\mathcal{M} \ a)))) =$$

$$(((\mathcal{M} \ a) \wedge ((\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b))) \vee (\neg(\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b)))) \vee$$

$$((\mathcal{M} \ b) \wedge ((\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a))) \vee (\neg(\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a))))$$
**by** *auto*

**have 5:**  $\vdash (((\mathcal{M} \ a) \wedge ((\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b))) \vee (\neg(\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b)))) \vee$   
 $((\mathcal{M} \ b) \wedge ((\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a))) \vee (\neg(\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a)))) =$   
 $((\mathcal{M} \ a) \wedge (\triangleright(\mathcal{M} \ b)) \vee (\neg(\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b)))) \vee$   
 $((\mathcal{M} \ b) \wedge (\triangleright(\mathcal{M} \ a)) \vee (\neg(\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a))))$ 
**by** (*simp add: first-d-def*)

**have 6:**  $\vdash (((\mathcal{M} \ a) \wedge (\triangleright(\mathcal{M} \ b)) \vee (\neg(\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b)))) \vee$   
 $((\mathcal{M} \ b) \wedge (\triangleright(\mathcal{M} \ a)) \vee (\neg(\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a)))) =$   
 $((\mathcal{M} \ a) \wedge ((\mathcal{M} \ b)) \vee (\neg(\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b)))) \vee$   
 $((\mathcal{M} \ b) \wedge ((\mathcal{M} \ a)) \vee (\neg(\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a))))$ 
**using** *MFixFst* **by** *fastforce*

**have 7:**  $\vdash (\neg(\mathcal{M} \ b) \wedge bs \ (\neg(\mathcal{M} \ b))) = bi(\neg(\mathcal{M} \ b))$   
**using** *AndBsEqvBi* **by** *blast*

**have 8:**  $\vdash (\neg(\mathcal{M} \ a) \wedge bs \ (\neg(\mathcal{M} \ a))) = bi(\neg(\mathcal{M} \ a))$   
**using** *AndBsEqvBi* **by** *blast*

**have 9:**  $\vdash (((\mathcal{M} \ a) \wedge ((\mathcal{M} \ b)) \vee ((\neg(\mathcal{M} \ b)) \wedge bs(\neg(\mathcal{M} \ b)))) \vee$   
 $((\mathcal{M} \ b) \wedge ((\mathcal{M} \ a)) \vee ((\neg(\mathcal{M} \ a)) \wedge bs(\neg(\mathcal{M} \ a)))) =$   
 $((\mathcal{M} \ a) \wedge ((\mathcal{M} \ b)) \vee (bi(\neg(\mathcal{M} \ b)))) \vee$   
 $((\mathcal{M} \ b) \wedge ((\mathcal{M} \ a)) \vee (bi(\neg(\mathcal{M} \ a))))$ 
**using** 7 8 **by** *fastforce*

**have 10:**  $\vdash (((\mathcal{M} \ a) \wedge ((\mathcal{M} \ b)) \vee (bi(\neg(\mathcal{M} \ b)))) \vee$   
 $((\mathcal{M} \ b) \wedge ((\mathcal{M} \ a)) \vee (bi(\neg(\mathcal{M} \ a)))) =$   
 $((\mathcal{M} \ a) \wedge (\mathcal{M} \ b)) \vee ((\mathcal{M} \ a) \wedge bi(\neg(\mathcal{M} \ b))) \vee$   
 $((\mathcal{M} \ b) \wedge (\mathcal{M} \ a)) \vee ((\mathcal{M} \ b) \wedge bi(\neg(\mathcal{M} \ a)))$ 
**by** *auto*

**have 11:**  $\vdash (((\mathcal{M} \ a) \wedge (\mathcal{M} \ b)) \vee ((\mathcal{M} \ a) \wedge bi(\neg(\mathcal{M} \ b))) \vee$   
 $((\mathcal{M} \ b) \wedge (\mathcal{M} \ a)) \vee ((\mathcal{M} \ b) \wedge bi(\neg(\mathcal{M} \ a)))) =$   
 $((\mathcal{M} \ a) \wedge bi(\neg(\mathcal{M} \ b))) \vee ((\mathcal{M} \ b) \wedge bi(\neg(\mathcal{M} \ a))) \vee ((\mathcal{M} \ a) \wedge (\mathcal{M} \ b))$ 
**by** *auto*

**show** *?thesis*  
**by** (*metis 10 11 2 3 4 5 6 9 MON.simps(2) int-eq*)  
**qed**

**lemma** *MThruAlt:*

$\vdash \mathcal{M}(a \text{ THRU } b) = (((\mathcal{M} \ a) \wedge di(\mathcal{M} \ b)) \vee ((\mathcal{M} \ b) \wedge di(\mathcal{M} \ a)))$

**proof** –

**have 1:**  $\vdash \mathcal{M}(a \text{ THRU } b) = \triangleright(di(\mathcal{M} \ a) \wedge di(\mathcal{M} \ b))$

**by** (*simp*)

**have 2:**  $\vdash \triangleright(di(\mathcal{M} \ a) \wedge di(\mathcal{M} \ b)) = ((\triangleright(\mathcal{M} \ a) \wedge di(\mathcal{M} \ b)) \vee (\triangleright(\mathcal{M} \ b) \wedge di(\mathcal{M} \ a)))$

**using** *FstDiAndDiEqv* **by** *auto*

**have 3:**  $\vdash ((\triangleright(\mathcal{M} \ a) \wedge di(\mathcal{M} \ b)) \vee (\triangleright(\mathcal{M} \ b) \wedge di(\mathcal{M} \ a))) =$   
 $((\mathcal{M} \ a) \wedge di(\mathcal{M} \ b)) \vee ((\mathcal{M} \ b) \wedge di(\mathcal{M} \ a))$

**using** *MFixFst* **by** *fastforce*

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

**qed**

**lemma** *MHaltAlt:*

$\vdash \mathcal{M}(\text{HALT } w) = \text{halt}(\text{init } w)$   
**proof** –  
 have 1:  $\vdash \mathcal{M}(\text{HALT } w) = \mathcal{M}(\text{FIRST LIFT}(\text{fin } (\text{init } w)))$  **by** (*simp add: HALT-d-def*)  
 have 2:  $\vdash \mathcal{M}(\text{FIRST LIFT}(\text{fin } (\text{init } w))) = \triangleright (\text{fin } (\text{init } w))$  **by** (*simp*)  
 have 3:  $\vdash \triangleright (\text{fin } (\text{init } w)) = \text{halt}(\text{init } w)$  **using** *HaltStateEqvFstFinState* **by** *fastforce*  
 from 1 2 3 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** *MFailUpto*:

$(\text{FAIL UPTO } a) \simeq (a)$

**proof** –  
 have 1:  $\vdash \mathcal{M}(\text{FAIL UPTO } a) = \triangleright (\mathcal{M} \text{ FAIL} \vee (\mathcal{M} a))$  **by** (*simp*)  
 have 2:  $\vdash (\mathcal{M} \text{ FAIL} \vee \mathcal{M} a) = (\#False \vee \mathcal{M} a)$  **using** *MFailAlt* **by** *auto*  
 have 3:  $\vdash \triangleright (\mathcal{M} \text{ FAIL} \vee (\mathcal{M} a)) = \triangleright (\#False \vee (\mathcal{M} a))$  **using** 2 *FstEqvRule* **by** *blast*  
 have 4:  $\vdash (\#False \vee (\mathcal{M} a)) = \mathcal{M} a$  **by** *simp*  
 have 5:  $\vdash \triangleright (\#False \vee (\mathcal{M} a)) = \triangleright (\mathcal{M} a)$  **using** 4 *FstEqvRule* **by** *blast*  
 have 6:  $\vdash \triangleright (\mathcal{M} a) = \mathcal{M} a$  **using** *MFixFst* **by** *fastforce*  
 from 1 2 3 4 5 6 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MFailThru*:

$(\text{FAIL THRU } (a)) \simeq \text{FAIL}$

**proof** –  
 have 1:  $\vdash \mathcal{M}(\text{FAIL THRU } (a)) = \triangleright (di(\mathcal{M} \text{ FAIL}) \wedge di(\mathcal{M} a))$   
     **by** (*simp*)  
 have 2:  $\vdash \triangleright (di(\mathcal{M} \text{ FAIL}) \wedge di(\mathcal{M} a)) = \triangleright (di(\#False) \wedge di(\mathcal{M} a))$   
     **using** *MFailAlt* **by** (*metis 1 int-eq*)  
 have 3:  $\vdash di \#False = \#False$   
     **by** (*simp add: di-defs Valid-def*)  
 hence 4:  $\vdash \triangleright (di(\#False) \wedge di(\mathcal{M} a)) = \triangleright (\#False) \wedge di(\mathcal{M} a)$   
     **by** (*metis 2 inteq-reflection*)  
 have 5:  $\vdash \triangleright (\#False) \wedge di(\mathcal{M} a) = \triangleright \#False$   
     **using** *FstEqvRule* **by** *fastforce*  
 have 6:  $\vdash \triangleright \#False = \#False$  **using** *FstFalse*  
     **by** *auto*  
 have 7:  $\vdash \#False = \mathcal{M} \text{ FAIL}$   
     **using** *MFailAlt* **by** *auto*  
 from 1 2 4 5 6 7 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MFailAnd*:

$(\text{FAIL AND } a) \simeq \text{FAIL}$

**proof** –  
 have 1:  $\vdash \mathcal{M}(\text{FAIL AND } a) = (\mathcal{M} \text{ FAIL} \wedge (\mathcal{M} a))$  **by** (*simp add: AND-d-def*)  
 have 2:  $\vdash (\mathcal{M} \text{ FAIL} \wedge (\mathcal{M} a)) = (\#False \wedge (\mathcal{M} a))$  **using** *MFailAlt* **by** *fastforce*  
 have 3:  $\vdash (\#False \wedge (\mathcal{M} a)) = \#False$  **by** *auto*  
 have 4:  $\vdash \mathcal{M}(\text{FAIL AND } a) = \#False$  **using** 1 2 3 **by** *fastforce*  
 have 5:  $\vdash \#False = \mathcal{M} \text{ FAIL}$  **using** *MFailAlt* **by** *auto*  
 from 1 2 3 4 5 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**



**lemma** *MThenFail*:

$(a \text{ THEN } \text{FAIL}) \simeq \text{FAIL}$

**proof** –

**have** 1:  $\vdash \mathcal{M} (a \text{ THEN } \text{FAIL}) = (\mathcal{M} a);(\mathcal{M} \text{ FAIL})$  **by** (*simp*)

**have** 2:  $\vdash (\mathcal{M} a);(\mathcal{M} \text{ FAIL}) = (\mathcal{M} a);\#False$  **by** (*simp add: MFailAlt RightChopEqvChop*)

**have** 3:  $\vdash (\mathcal{M} a);\#False = \#False$  **by** (*simp add: chop-d-def Valid-def*)

**have** 4:  $\vdash \#False = \mathcal{M} \text{ FAIL}$  **using** *MFailAlt* **by** *auto*

**from** 1 2 3 4 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

**qed**

**lemma** *MFailThen*:

$(\text{FAIL THEN } a) \simeq \text{FAIL}$

**proof** –

**have** 1:  $\vdash \mathcal{M}(\text{FAIL THEN } a) = (\mathcal{M} \text{ FAIL});(\mathcal{M} a)$  **by** (*simp*)

**have** 2:  $\vdash (\mathcal{M} \text{ FAIL});(\mathcal{M} a) = \#False;(\mathcal{M} a)$  **using** *MFailAlt* **using** *LeftChopEqvChop* **by** *blast*

**have** 3:  $\vdash \#False;(\mathcal{M} a) = \#False$  **by** (*simp add: chop-d-def Valid-def*)

**have** 4:  $\vdash \#False = \mathcal{M} \text{ FAIL}$  **using** *MFailAlt* **by** *auto*

**from** 1 2 3 4 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

**qed**

**lemma** *MFailWith*:

$(\text{FAIL WITH } f) \simeq \text{FAIL}$

**proof** –

**have** 1:  $\vdash \mathcal{M}(\text{FAIL WITH } f) = ((\mathcal{M} \text{ FAIL}) \wedge f)$  **by** (*simp*)

**have** 2:  $\vdash ((\mathcal{M} \text{ FAIL}) \wedge f) = (\#False \wedge f)$  **using** *MFailAlt* **by** *auto*

**have** 3:  $\vdash (\#False \wedge f) = \#False$  **by** *simp*

**have** 4:  $\vdash \#False = \mathcal{M} \text{ FAIL}$  **using** *MFailAlt* **by** *auto*

**from** 1 2 3 4 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

**qed**

**lemma** *MWithFalse*:

$(a \text{ WITH } (\text{LIFT}(\#False))) \simeq \text{FAIL}$

**proof** –

**have** 1:  $\vdash \mathcal{M} (a \text{ WITH } \text{LIFT}(\#False)) = ((\mathcal{M} a) \wedge \#False)$  **by** (*simp*)

**have** 2:  $\vdash ((\mathcal{M} a) \wedge \#False) = \mathcal{M} \text{ FAIL}$  **using** *MFailAlt* **by** *auto*

**from** 1 2 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

**qed**

**lemma** *MWithTrue*:

$(a \text{ WITH } (\text{LIFT}(\#True))) \simeq a$

**proof** –

**have** 1:  $\vdash \mathcal{M} (a \text{ WITH } \text{LIFT}(\#True)) = ((\mathcal{M} a) \wedge \#True)$  **by** (*simp*)

**have** 2:  $\vdash ((\mathcal{M} a) \wedge \#True) = \mathcal{M} a$  **by** *simp*

**from** 1 2 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

**qed**

**lemma** *MEmptyUpto*:

$(\text{EMPTY UPTO } a) \simeq \text{EMPTY}$

**proof** –

**have** 1:  $\vdash \mathcal{M} (EMPTY \text{ UPTO } a) = \triangleright(\mathcal{M} EMPTY \vee (\mathcal{M} a))$  **by** (*simp*)  
**have** 2:  $\vdash \mathcal{M} EMPTY = empty$  **using** *MEEmptyAlt* **by** *auto*  
**hence** 3:  $\vdash (\mathcal{M} EMPTY \vee (\mathcal{M} a)) = (empty \vee (\mathcal{M} a))$  **by** *auto*  
**hence** 4:  $\vdash \triangleright(\mathcal{M} EMPTY \vee \mathcal{M} a) = \triangleright(empty \vee \mathcal{M} a)$  **using** *FstEqvRule* **by** *blast*  
**have** 5:  $\vdash \triangleright(empty \vee \mathcal{M} a) = empty$  **using** *FstEmptyOrEqvEmpty* **by** *blast*  
**have** 6:  $\vdash empty = \mathcal{M} EMPTY$  **using** *MEEmptyAlt* **by** *auto*  
**from** 1 4 5 6 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MEEmptyThru*:

$(EMPTY \text{ THRU } a) \simeq (a)$

**proof** –

**have** 1:  $\vdash \mathcal{M}(EMPTY \text{ THRU } a) = \triangleright(di(\mathcal{M} EMPTY) \wedge di(\mathcal{M} a))$  **by** (*simp*)  
**have** 2:  $\vdash di(\mathcal{M} EMPTY) = di \text{ empty}$  **using** *MEEmptyAlt DiEqvDi* **by** *blast*  
**hence** 3:  $\vdash (di(\mathcal{M} EMPTY) \wedge di(\mathcal{M} a)) = (di \text{ empty} \wedge di(\mathcal{M} a))$  **by** *auto*  
**hence** 4:  $\vdash (di \text{ empty} \wedge di(\mathcal{M} a)) = di(\mathcal{M} a)$  **using** *DiEmpty* **by** *auto*  
**have** 5:  $\vdash (di(\mathcal{M} EMPTY) \wedge di(\mathcal{M} a)) = di(\mathcal{M} a)$  **using** 3 4 **by** *fastforce*  
**hence** 6:  $\vdash \triangleright(di(\mathcal{M} EMPTY) \wedge di(\mathcal{M} a)) = \triangleright(di(\mathcal{M} a))$  **using** *FstEqvRule* **by** *blast*  
**have** 7:  $\vdash \triangleright(di(\mathcal{M} a)) = \triangleright(\mathcal{M} a)$  **using** *FstDiEqvFst* **by** *blast*  
**have** 8:  $\vdash \triangleright(\mathcal{M} a) = (\mathcal{M} a)$  **using** *MFixFst* **by** *fastforce*  
**from** 1 6 7 8 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MThenEmpty*:

$(a \text{ THEN } EMPTY) \simeq (a)$

**proof** –

**have** 1:  $\vdash \mathcal{M}(a \text{ THEN } EMPTY) = (\mathcal{M} a); (\mathcal{M} EMPTY)$  **by** (*simp*)  
**have** 2:  $\vdash (\mathcal{M} a); (\mathcal{M} EMPTY) = (\mathcal{M} a); empty$  **by** (*simp add: MEEmptyAlt RightChopEqvChop*)  
**have** 3:  $\vdash (\mathcal{M} a); empty = (\mathcal{M} a)$  **using** *ChopEmpty* **by** *auto*  
**from** 1 2 3 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MEEmptyThen*:

$(EMPTY \text{ THEN } a) \simeq a$

**proof** –

**have** 1:  $\vdash \mathcal{M}(EMPTY \text{ THEN } a) = (\mathcal{M} EMPTY); (\mathcal{M} a)$  **by** (*simp*)  
**have** 2:  $\vdash (\mathcal{M} EMPTY); (\mathcal{M} a) = empty; (\mathcal{M} a)$  **by** (*simp add: MEEmptyAlt LeftChopEqvChop*)  
**have** 3:  $\vdash empty; (\mathcal{M} a) = (\mathcal{M} a)$  **by** (*simp add: EmptyChop*)  
**from** 1 2 3 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MEEmptyIterate*:

$(EMPTY \text{ ITERATE } b) \simeq EMPTY$

**proof** –

**have** 1:  $\vdash \mathcal{M}(EMPTY \text{ ITERATE } b) = \mathcal{M}(EMPTY \text{ WITH LIFT } (\mathcal{M} b)^*)$   
**by** (*simp add: ITERATE-d-def*)  
**have** 2:  $\vdash \mathcal{M}(EMPTY \text{ WITH LIFT } (\mathcal{M} b)^*) = (\mathcal{M} EMPTY \wedge (\mathcal{M} b)^*)$   
**by** (*simp*)  
**have** 3:  $\vdash (\mathcal{M} EMPTY \wedge (\mathcal{M} b)^*) = (empty \wedge (\mathcal{M} b)^*)$   
**using** *MEEmptyAlt* **by** *auto*

**have** 4:  $\vdash (\text{empty} \wedge (\mathcal{M} b)^*) = (\text{empty} \wedge (\text{empty} \vee (((\mathcal{M} b) \wedge \text{more}); (\mathcal{M} b)^*)))$   
**using** *ChopstarEqv* **by** *fastforce*  
**have** 5:  $\vdash (\text{empty} \wedge (\text{empty} \vee (((\mathcal{M} b) \wedge \text{more}); (\mathcal{M} b)^*))) = \text{empty}$   
**by** *auto*  
**have** 6:  $\vdash \mathcal{M}(\text{EMPTY ITERATE } b) = \mathcal{M} \text{ EMPTY}$   
**using** 1 2 3 4 5 *MEmpyAlt* **by** *fastforce*  
**from** 6 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MIterateIdemp*:

$(a \text{ ITERATE } a) \simeq (a)$

**proof** –

**have** 1:  $\vdash \mathcal{M}(a \text{ ITERATE } a) = \mathcal{M}(a \text{ WITH LIFT } (\mathcal{M} a)^*)$  **by** (*simp add: ITERATE-d-def*)  
**have** 2:  $\vdash \mathcal{M}(a \text{ WITH LIFT } (\mathcal{M} a)^*) = ((\mathcal{M} a) \wedge (\mathcal{M} a)^*)$  **by** (*simp*)  
**have** 3:  $\vdash ((\mathcal{M} a) \wedge (\mathcal{M} a)^*) = (\triangleright(\mathcal{M} a) \wedge (\triangleright(\mathcal{M} a))^*)$  **using** *MFixFst*  
**by** (*metis ImpCS inteq-reflection Prop10*)  
**have** 4:  $\vdash (\triangleright(\mathcal{M} a) \wedge (\triangleright(\mathcal{M} a))^*) = \triangleright(\mathcal{M} a)$  **using** *FstAndFstStarEqvFst* **by** *fastforce*  
**have** 5:  $\vdash \triangleright(\mathcal{M} a) = \mathcal{M} a$  **using** *MFixFst* **by** *fastforce*  
**from** 1 2 3 4 5 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MUptoIdemp*:

$(a \text{ UPTO } a) \simeq (a)$

**proof** –

**have** 1:  $\vdash \mathcal{M}(a \text{ UPTO } a) = \triangleright((\mathcal{M} a) \vee (\mathcal{M} a))$  **by** *auto*  
**have** 2:  $\vdash \triangleright((\mathcal{M} a) \vee (\mathcal{M} a)) = \triangleright(\mathcal{M} a)$  **using** *FstEqvRule* **by** *fastforce*  
**have** 3:  $\vdash \triangleright(\mathcal{M} a) = (\mathcal{M} a)$  **using** *MFixFst* **by** *fastforce*  
**from** 1 2 3 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MThruIdemp*:

$(a \text{ THRU } a) \simeq (a)$

**proof** –

**have** 1:  $\vdash \mathcal{M}(a \text{ THRU } a) = \triangleright(\text{di}(\mathcal{M} a) \wedge \text{di}(\mathcal{M} a))$  **by** *auto*  
**have** 2:  $\vdash \triangleright(\text{di}(\mathcal{M} a) \wedge \text{di}(\mathcal{M} a)) = \triangleright(\text{di}(\mathcal{M} a))$  **using** *FstEqvRule* **by** *fastforce*  
**have** 3:  $\vdash \triangleright(\text{di}(\mathcal{M} a)) = \triangleright(\mathcal{M} a)$  **using** *FstDiEqvFst* **by** *blast*  
**have** 4:  $\vdash \triangleright(\mathcal{M} a) = (\mathcal{M} a)$  **using** *MFixFst* **by** *fastforce*  
**from** 1 2 3 4 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MAndIdemp*:

$(a \text{ AND } a) \simeq (a)$

**proof** –

**have** 1:  $\vdash \mathcal{M}(a \text{ AND } a) = ((\mathcal{M} a) \wedge (\mathcal{M} a))$  **by** (*simp add: AND-d-def*)  
**have** 2:  $\vdash ((\mathcal{M} a) \wedge (\mathcal{M} a)) = (\mathcal{M} a)$  **by** *fastforce*  
**from** 1 2 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MWithIdemp*:

$((a \text{ WITH } f) \text{ WITH } f) \simeq (a \text{ WITH } f)$

**proof** –

have 1:  $\vdash \mathcal{M}((a \text{ WITH } f) \text{ WITH } f) = (((\mathcal{M} a) \wedge (f)) \wedge (f))$  **by** *auto*  
 have 2:  $\vdash (((\mathcal{M} a) \wedge (f)) \wedge (f)) = ((\mathcal{M} a) \wedge (f))$  **by** *fastforce*  
 have 3:  $\vdash ((\mathcal{M} a) \wedge (f)) = \mathcal{M}(a \text{ WITH } f)$  **by** *auto*  
 from 1 2 3 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MUptoCommut*:

$(a \text{ UPTO } b) \simeq (b \text{ UPTO } a)$

**proof** –

have 1:  $\vdash \mathcal{M}(a \text{ UPTO } b) = \triangleright((\mathcal{M} a) \vee (\mathcal{M} b))$  **by** (*simp*)  
 have 2:  $\vdash ((\mathcal{M} a) \vee (\mathcal{M} b)) = ((\mathcal{M} b) \vee (\mathcal{M} a))$  **by** *auto*  
 hence 3:  $\vdash \triangleright((\mathcal{M} a) \vee (\mathcal{M} b)) = \triangleright((\mathcal{M} b) \vee (\mathcal{M} a))$  **using** *FstEqvRule* **by** *blast*  
 have 4:  $\vdash \triangleright((\mathcal{M} b) \vee (\mathcal{M} a)) = \mathcal{M}(b \text{ UPTO } a)$  **by** *auto*  
 from 1 3 4 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MThruCommut*:

$(a \text{ THRU } b) \simeq (b \text{ THRU } a)$

**proof** –

have 1:  $\vdash \mathcal{M}(a \text{ THRU } b) = \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))$  **by** (*simp*)  
 have 2:  $\vdash (di(\mathcal{M} a) \wedge di(\mathcal{M} b)) = (di(\mathcal{M} b) \wedge di(\mathcal{M} a))$  **by** *auto*  
 hence 3:  $\vdash \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b)) = \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} a))$  **using** *FstEqvRule* **by** *blast*  
 have 4:  $\vdash \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} a)) = \mathcal{M}(b \text{ THRU } a)$  **by** *auto*  
 from 1 3 4 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MAndCommut*:

$(a \text{ AND } b) \simeq (b \text{ AND } a)$

**proof** –

have 1:  $\vdash \mathcal{M}(a \text{ AND } b) = ((\mathcal{M} a) \wedge (\mathcal{M} b))$  **by** (*simp add: AND-d-def*)  
 have 2:  $\vdash ((\mathcal{M} a) \wedge (\mathcal{M} b)) = ((\mathcal{M} b) \wedge (\mathcal{M} a))$  **by** *auto*  
 have 3:  $\vdash ((\mathcal{M} b) \wedge (\mathcal{M} a)) = \mathcal{M}(b \text{ AND } a)$  **by** (*simp add: AND-d-def*)  
 from 1 2 3 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MWithCommut*:

$((a \text{ WITH } f) \text{ WITH } g) \simeq ((a \text{ WITH } g) \text{ WITH } f)$

**proof** –

have 1:  $\vdash \mathcal{M}((a \text{ WITH } f) \text{ WITH } g) = (((\mathcal{M} a) \wedge (f)) \wedge (g))$  **by** *auto*  
 have 2:  $\vdash (((\mathcal{M} a) \wedge (f)) \wedge (g)) = (((\mathcal{M} a) \wedge (g)) \wedge (f))$  **by** *auto*  
 have 3:  $\vdash (((\mathcal{M} a) \wedge (g)) \wedge (f)) = \mathcal{M}((a \text{ WITH } g) \text{ WITH } f)$  **by** *auto*  
 from 1 2 3 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MWithAbsorp*:

$((a \text{ WITH } f) \text{ WITH } g) \simeq (a \text{ WITH } LIFT(f \wedge g))$

**proof** –

have 1:  $\vdash \mathcal{M}((a \text{ WITH } f) \text{ WITH } g) = (((\mathcal{M} a) \wedge (f)) \wedge (g))$  **by** *auto*  
 have 2:  $\vdash (((\mathcal{M} a) \wedge (f)) \wedge (g)) = ((\mathcal{M} a) \wedge (f \wedge g))$  **by** *auto*

from 1 2 show ?thesis by (simp add: MonEq)  
qed

lemma MUptoAssoc:

$((a \text{ UPTO } b) \text{ UPTO } c) \simeq (a \text{ UPTO } (b \text{ UPTO } c))$

proof –

have 1:  $\vdash \mathcal{M}((a \text{ UPTO } b) \text{ UPTO } c) = \triangleright(\mathcal{M}(a \text{ UPTO } b) \vee (\mathcal{M} c))$   
by (simp)  
have 2:  $\vdash \triangleright(\mathcal{M}(a \text{ UPTO } b) \vee (\mathcal{M} c)) = \triangleright(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b)) \vee (\mathcal{M} c))$   
by auto  
have 3:  $\vdash \triangleright(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b)) \vee (\mathcal{M} c)) = \triangleright(((\mathcal{M} a) \vee (\mathcal{M} b)) \vee (\mathcal{M} c))$   
using FstFstOrEqvFstOrL by blast  
have 4:  $\vdash (((\mathcal{M} a) \vee (\mathcal{M} b)) \vee (\mathcal{M} c)) = ((\mathcal{M} a) \vee ((\mathcal{M} b) \vee (\mathcal{M} c)))$   
by auto  
hence 5:  $\vdash \triangleright(((\mathcal{M} a) \vee (\mathcal{M} b)) \vee (\mathcal{M} c)) = \triangleright((\mathcal{M} a) \vee ((\mathcal{M} b) \vee (\mathcal{M} c)))$   
using FstEqvRule by blast  
have 6:  $\vdash \triangleright((\mathcal{M} a) \vee ((\mathcal{M} b) \vee (\mathcal{M} c))) = \triangleright((\mathcal{M} a) \vee \triangleright((\mathcal{M} b) \vee (\mathcal{M} c)))$   
using FstFstOrEqvFstOrR by fastforce  
have 7:  $\vdash \triangleright((\mathcal{M} a) \vee \triangleright((\mathcal{M} b) \vee (\mathcal{M} c))) = \triangleright((\mathcal{M} a) \vee \mathcal{M}(b \text{ UPTO } c))$   
by auto  
have 8:  $\vdash \triangleright((\mathcal{M} a) \vee \mathcal{M}(b \text{ UPTO } c)) = \mathcal{M}(a \text{ UPTO } (b \text{ UPTO } c))$   
by auto

from 1 2 3 5 6 7 8 show ?thesis using MonEq by (metis int-eq)  
qed

lemma MThruAssoc:

$((a \text{ THRU } b) \text{ THRU } c) \simeq (a \text{ THRU } (b \text{ THRU } c))$

proof –

have 1:  $\vdash \mathcal{M}((a \text{ THRU } b) \text{ THRU } c) = \triangleright(di(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) \wedge di(\mathcal{M} c)$   
by auto  
have 2:  $\vdash di(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) = di((di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$   
using DiEqvDiFst by fastforce  
have 3:  $\vdash di((di(\mathcal{M} a) \wedge di(\mathcal{M} b))) = (di(\mathcal{M} a) \wedge di(\mathcal{M} b))$   
using DiDiAndEqvDi by blast  
have 4:  $\vdash di(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) = (di(\mathcal{M} a) \wedge di(\mathcal{M} b))$   
using 2 3 by fastforce  
hence 5:  $\vdash (di(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) \wedge di(\mathcal{M} c)) = (di(\mathcal{M} a) \wedge di(\mathcal{M} b) \wedge di(\mathcal{M} c))$   
by auto  
have 6:  $\vdash (di(\mathcal{M} b) \wedge di(\mathcal{M} c)) = di(di(\mathcal{M} b) \wedge di(\mathcal{M} c))$   
using DiDiAndEqvDi by fastforce  
have 7:  $\vdash di(di(\mathcal{M} b) \wedge di(\mathcal{M} c)) = di(\triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c)))$   
using DiEqvDiFst by blast  
have 8:  $\vdash (di(\mathcal{M} b) \wedge di(\mathcal{M} c)) = di(\triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c)))$   
using 6 7 by fastforce  
hence 9:  $\vdash (di(\mathcal{M} a) \wedge di(\mathcal{M} b) \wedge di(\mathcal{M} c)) = (di(\mathcal{M} a) \wedge di(\triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))))$   
by auto  
have 10:  $\vdash (di(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) \wedge di(\mathcal{M} c)) =$   
 $(di(\mathcal{M} a) \wedge di(\triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))))$   
using 5 9 by fastforce  
hence 11:  $\vdash \triangleright(di(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) \wedge di(\mathcal{M} c)) =$

$\triangleright(di(\mathcal{M} a) \wedge di(\triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))))$   
**using** *FstEqvRule* **by** *fastforce*  
**have** 12:  $\vdash \triangleright(di(\mathcal{M} a) \wedge di(\triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c)))) = \mathcal{M}(a \text{ THRU } (b \text{ THRU } c))$   
**by** *auto*  
**from** 1 11 12 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MAndAssoc*:

$((a \text{ AND } b) \text{ AND } c) \simeq (a \text{ AND } (b \text{ AND } c))$

**proof** –

**have** 1:  $\vdash \mathcal{M}((a \text{ AND } b) \text{ AND } c) = ((\mathcal{M} a) \wedge (\mathcal{M} b) \wedge (\mathcal{M} c))$   
**using** *AND-d-def* **by** (*metis MON.simps(5) MWithAbsorp eq-d-def*)  
**have** 2:  $\vdash ((\mathcal{M} a) \wedge (\mathcal{M} b) \wedge (\mathcal{M} c)) = \mathcal{M}(a \text{ AND } (b \text{ AND } c))$   
**using** *AND-d-def* **by** (*simp add: AND-d-def*)  
**from** 1 2 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MThenAssoc*:

$((a \text{ THEN } b) \text{ THEN } c) \simeq (a \text{ THEN } (b \text{ THEN } c))$

**proof** –

**have** 1:  $\vdash \mathcal{M}((a \text{ THEN } b) \text{ THEN } c) = ((\mathcal{M} a);(\mathcal{M} b));(\mathcal{M} c)$  **by** *auto*  
**have** 2:  $\vdash ((\mathcal{M} a);(\mathcal{M} b));(\mathcal{M} c) = (\mathcal{M} a);((\mathcal{M} b);(\mathcal{M} c))$  **using** *ChopAssocB* **by** *blast*  
**have** 3:  $\vdash (\mathcal{M} a);((\mathcal{M} b);(\mathcal{M} c)) = \mathcal{M}(a \text{ THEN } (b \text{ THEN } c))$  **by** *auto*  
**from** 1 2 3 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MUptoThruAbsorp*:

$(a \text{ UPTO } (a \text{ THRU } b)) \simeq a$

**proof** –

**have** 1:  $\vdash \mathcal{M}(a \text{ UPTO } (a \text{ THRU } b)) = \triangleright((\mathcal{M} a) \vee \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$   
**by** *simp*  
**have** 2:  $\vdash \triangleright((\mathcal{M} a) \vee \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) =$   
 $\triangleright((\mathcal{M} a) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$   
**using** *FstFstOrEqvFstOrR* **by** *auto*  
**have** 3:  $\vdash ((\mathcal{M} a) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b))) =$   
 $((\mathcal{M} a) \vee di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b))$   
**by** *auto*  
**have** 4:  $\vdash (((\mathcal{M} a) \vee di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b))) =$   
 $((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))$   
**using** *OrDiEqvDi* **by** *fastforce*  
**have** 5:  $\vdash ((\mathcal{M} a) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b))) =$   
 $((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))$   
**using** 3 4 **by** *auto*  
**hence** 6:  $\vdash \triangleright((\mathcal{M} a) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b))) =$   
 $\triangleright((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))$   
**using** *FstEqvRule* **by** *blast*  
**have** 7:  $\vdash \triangleright((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b))) =$   
 $((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$   
 $bs(\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b))))$   
**by** (*auto simp add: first-d-def*)

**have 8:**  $\vdash ((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b))) =$   
 $((di(\mathcal{M} a) \wedge (\mathcal{M} a)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$   
**by auto**  
**hence 9:**  $\vdash (\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))) =$   
 $(\neg((di(\mathcal{M} a) \wedge (\mathcal{M} a)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$   
**by fastforce**  
**have 10:**  $\vdash (\neg((di(\mathcal{M} a) \wedge (\mathcal{M} a)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) =$   
 $(\neg(((\mathcal{M} a)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$   
**using AndDiEqv using 5 by auto**  
**have 11:**  $\vdash (\neg(((\mathcal{M} a)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) =$   
 $(\neg(\mathcal{M} a) \wedge \neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$   
**by auto**  
**have 12:**  $\vdash (\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))) =$   
 $(\neg(\mathcal{M} a) \wedge \neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$   
**using 9 10 11 by auto**  
**hence 13:**  $\vdash bs(\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))) =$   
 $bs(\neg(\mathcal{M} a) \wedge \neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))$   
**using BsEqvRule by blast**  
**have 14:**  $\vdash bs((\neg(\mathcal{M} a)) \wedge \neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) =$   
 $(bs((\neg(\mathcal{M} a))) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$   
**using BsAndEqv by fastforce**  
**have 141:**  $\vdash bs(\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))) =$   
 $(bs((\neg(\mathcal{M} a))) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$   
**using 13 14 by fastforce**  
**hence 15:**  $\vdash ((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$   
 $bs(\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))) =$   
 $((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$   
 $bs((\neg(\mathcal{M} a))) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$   
**by auto**  
**have 16:**  $\vdash ((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$   
 $bs((\neg(\mathcal{M} a))) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) =$   
 $((bs((\neg(\mathcal{M} a))) \wedge di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$   
 $bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$   
**by auto**  
**have 17:**  $\vdash ((bs((\neg(\mathcal{M} a))) \wedge di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$   
 $bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) =$   
 $((\triangleright(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$   
 $bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$   
**using FstEqvBsNotAndDi by fastforce**  
**have 18:**  $\vdash ((\triangleright(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$   
 $bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) =$   
 $((\mathcal{M} a) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$   
 $bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$   
**using MFixFst by fastforce**  
**have 19:**  $\vdash (((\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$   
 $bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) =$   
 $((\mathcal{M} a) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$   
**by auto**  
**have 20:**  $\vdash (\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) = (\neg(di(\mathcal{M} a)) \vee \neg(di(\mathcal{M} b)))$   
**by auto**

**have 21:**  $\vdash (\neg(di(\mathcal{M} a)) \vee \neg(di(\mathcal{M} b))) = ((bi(\neg(\mathcal{M} a))) \vee (bi(\neg(\mathcal{M} b))))$   
**by** (*simp add: bi-d-def*)  
**have 22:**  $\vdash (\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) = ((bi(\neg(\mathcal{M} a))) \vee (bi(\neg(\mathcal{M} b))))$   
**using** 20 21 **by** *auto*  
**hence 23:**  $\vdash bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) = bs((bi(\neg(\mathcal{M} a))) \vee (bi(\neg(\mathcal{M} b))))$   
**using** *BsEqvRule* **by** *blast*  
**have 24:**  $\vdash bs((bi(\neg(\mathcal{M} a))) \vee (bi(\neg(\mathcal{M} b)))) = bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b))$   
**using** *BsOrBsEqvBsBiOrBi* **by** *fastforce*  
**have 25:**  $\vdash bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))) = (bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b)))$   
**using** 23 24 **using** *BsOrBsEqvBsBiOrBi* **by** *fastforce*  
**hence 26:**  $\vdash ((\mathcal{M} a) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) =$   
 $((\mathcal{M} a) \wedge (bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b))))$   
**by** *auto*  
**have 27:**  $\vdash ((\mathcal{M} a) \wedge (bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b)))) =$   
 $(\triangleright(\mathcal{M} a) \wedge (bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b))))$   
**using** *MFixFst* **by** *fastforce*  
**have 28:**  $\vdash (\triangleright(\mathcal{M} a) \wedge (bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b)))) =$   
 $((\mathcal{M} a) \wedge bs(\neg(\mathcal{M} a)) \wedge (bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b))))$   
**by** (*auto simp add: first-d-def*)  
**have 29:**  $\vdash ((\mathcal{M} a) \wedge bs(\neg(\mathcal{M} a)) \wedge (bs(\neg(\mathcal{M} a)) \vee bs(\neg(\mathcal{M} b)))) =$   
 $((\mathcal{M} a) \wedge bs(\neg(\mathcal{M} a)))$   
**by** *auto*  
**have 30:**  $\vdash ((\mathcal{M} a) \wedge bs(\neg(\mathcal{M} a))) = \triangleright(\mathcal{M} a)$   
**by** (*simp add: first-d-def*)  
**have 31:**  $\vdash \triangleright(\mathcal{M} a) = (\mathcal{M} a)$   
**using** *MFixFst* **by** *fastforce*  
**have 32:**  $\vdash \mathcal{M}(a \text{ UPTO } (a \text{ THRU } b)) =$   
 $((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$   
 $bs(\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))))$   
**using** 1 2 6 7 **by** *fastforce*  
**have 33:**  $\vdash ((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$   
 $bs(\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b))))) =$   
 $((\mathcal{M} a) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b))))$   
**using** 15 16 17 18 19 **by** (*metis int-eq*)  
**have 34:**  $\vdash ((\mathcal{M} a) \wedge bs(\neg(di(\mathcal{M} a) \wedge di(\mathcal{M} b)))) = (\mathcal{M} a)$   
**using** 26 27 28 29 30 31 **by** (*metis int-eq*)  
**from** 32 33 34 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MThruUptoAbsorp*:

$(a \text{ THRU } (a \text{ UPTO } b)) \simeq (a)$

**proof** –

**have 1:**  $\vdash \mathcal{M}(a \text{ THRU } (a \text{ UPTO } b)) = \triangleright(di(\mathcal{M} a) \wedge di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))))$   
**by** *simp*  
**have 2:**  $\vdash \triangleright(di(\mathcal{M} a) \wedge di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b)))) =$   
 $\triangleright(di(\mathcal{M} a) \wedge di(((\mathcal{M} a) \vee (\mathcal{M} b))))$   
**by** (*metis 1 DiEqvDiFst int-eq*)  
**have 3:**  $\vdash \triangleright(di(\mathcal{M} a) \wedge di(((\mathcal{M} a) \vee (\mathcal{M} b)))) =$   
 $\triangleright(di(\mathcal{M} a) \wedge (di(\mathcal{M} a) \vee di(\mathcal{M} b)))$   
**by** (*metis DiOrEqv FstEqvRule inteq-reflection lift-and-com*)



have 4:  $\vdash (di(\mathcal{M} a) \wedge (di(\mathcal{M} a) \vee di(\mathcal{M} b))) = (di(\mathcal{M} a))$   
     **by** *auto*  
 hence 5:  $\vdash \triangleright(di(\mathcal{M} a) \wedge (di(\mathcal{M} a) \vee di(\mathcal{M} b))) = \triangleright(di(\mathcal{M} a))$   
     **using** *FstEqvRule* **by** *blast*  
 have 6:  $\vdash \triangleright(di(\mathcal{M} a)) = \triangleright(\mathcal{M} a)$   
     **using** *FstDiEqvFst* **by** *blast*  
 have 7:  $\vdash \triangleright(\mathcal{M} a) = (\mathcal{M} a)$   
     **using** *MFixFst* **by** *fastforce*  
 from 1 2 3 5 6 7 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)  
**qed**

**lemma** *MUptoThruDistrib*:

$(a \text{ UPTO } (b \text{ THRU } c)) \simeq ((a \text{ UPTO } b) \text{ THRU } (a \text{ UPTO } c))$

**proof** –

have 1:  $\vdash \mathcal{M}((a \text{ UPTO } b) \text{ THRU } (a \text{ UPTO } c)) =$   
      $\triangleright( di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} c))) )$   
     **by** *simp*  
 have 2:  $\vdash ( di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} c))) ) =$   
      $( di(((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge di(((\mathcal{M} a) \vee (\mathcal{M} c))) )$   
     **using** *DiEqvDiFst* **by** *fastforce*  
 have 3:  $\vdash ( di(((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge di(((\mathcal{M} a) \vee (\mathcal{M} c))) ) =$   
      $(( di(\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge (di(\mathcal{M} a) \vee di(\mathcal{M} c)))$   
     **using** *DiOrEqv* **by** *fastforce*  
 have 4:  $\vdash (( di(\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge (di(\mathcal{M} a) \vee di(\mathcal{M} c))) =$   
      $(di(\mathcal{M} a) \vee (di(\mathcal{M} b) \wedge di(\mathcal{M} c)))$   
     **by** *auto*  
 have 5:  $\vdash ( di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} c))) ) =$   
      $(di(\mathcal{M} a) \vee (di(\mathcal{M} b) \wedge di(\mathcal{M} c)))$   
     **using** 2 3 4 **by** *fastforce*  
 hence 6:  $\vdash \triangleright( di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge di(\triangleright((\mathcal{M} a) \vee (\mathcal{M} c))) ) =$   
      $\triangleright( di(\mathcal{M} a) \vee (di(\mathcal{M} b) \wedge di(\mathcal{M} c))) )$   
     **using** *FstEqvRule* **by** *blast*  
 have 7:  $\vdash \triangleright( di(\mathcal{M} a) \vee (di(\mathcal{M} b) \wedge di(\mathcal{M} c))) =$   
      $\triangleright( \triangleright(di(\mathcal{M} a)) \vee \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))) )$   
     **using** *FstFstOrEqvFstOr* **by** *fastforce*  
 have 8:  $\vdash \triangleright(di(\mathcal{M} a)) = \triangleright((\mathcal{M} a))$   
     **using** *FstDiEqvFst* **by** *blast*  
 have 9:  $\vdash \triangleright((\mathcal{M} a)) = (\mathcal{M} a)$   
     **using** *MFixFst* **by** *fastforce*  
 have 10:  $\vdash \triangleright(di(\mathcal{M} a)) = (\mathcal{M} a)$   
     **using** 8 9 **by** *fastforce*  
 hence 11:  $\vdash (\triangleright(di(\mathcal{M} a)) \vee \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))) =$   
      $((\mathcal{M} a) \vee \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c)))$   
     **by** *auto*  
 hence 12:  $\vdash \triangleright(\triangleright(di(\mathcal{M} a)) \vee \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))) =$   
      $\triangleright((\mathcal{M} a) \vee \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c)))$   
     **using** *FstEqvRule* **by** *blast*  
 have 13:  $\vdash \triangleright((\mathcal{M} a) \vee \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))) = \mathcal{M}(a \text{ UPTO } (b \text{ THRU } c))$   
     **by** *simp*  
 from 1 6 7 12 13 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

**lemma** *MThruUptoDistrib*:

$$(a \text{ THRU } (b \text{ UPTO } c)) \simeq ((a \text{ THRU } b) \text{ UPTO } (a \text{ THRU } c))$$

**proof** –

**have** 1:  $\vdash \mathcal{M}((a \text{ THRU } b) \text{ UPTO } (a \text{ THRU } c)) =$   
 $\triangleright(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b)) \vee \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} c)))$   
**by** *simp*  
**have** 2:  $\vdash \triangleright(\triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} b)) \vee \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} c))) =$   
 $\triangleright((di(\mathcal{M} a) \wedge di(\mathcal{M} b)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} c)))$   
**using** *FstFstOrEqvFstOr* **by** *auto*  
**have** 3:  $\vdash ((di(\mathcal{M} a) \wedge di(\mathcal{M} b)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} c))) =$   
 $(di(\mathcal{M} a) \wedge (di(\mathcal{M} b) \vee di(\mathcal{M} c)))$  **by** *auto*  
**have** 4:  $\vdash (di(\mathcal{M} a) \wedge (di(\mathcal{M} b) \vee di(\mathcal{M} c))) =$   
 $(di(\mathcal{M} a) \wedge di((\mathcal{M} b) \vee (\mathcal{M} c)))$  **using** *DiOrEqv* **by** *fastforce*  
**have** 5:  $\vdash (di(\mathcal{M} a) \wedge di((\mathcal{M} b) \vee (\mathcal{M} c))) =$   
 $(di(\mathcal{M} a) \wedge di(\triangleright((\mathcal{M} b) \vee (\mathcal{M} c))))$  **using** *DiEqvDiFst* **by** *fastforce*  
**have** 6:  $\vdash ((di(\mathcal{M} a) \wedge di(\mathcal{M} b)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} c))) =$   
 $(di(\mathcal{M} a) \wedge di(\triangleright((\mathcal{M} b) \vee (\mathcal{M} c))))$  **using** 3 4 5 **by** *fastforce*  
**hence** 7:  $\vdash \triangleright((di(\mathcal{M} a) \wedge di(\mathcal{M} b)) \vee (di(\mathcal{M} a) \wedge di(\mathcal{M} c))) =$   
 $\triangleright(di(\mathcal{M} a) \wedge di(\triangleright((\mathcal{M} b) \vee (\mathcal{M} c))))$  **using** *FstEqvRule* **by** *blast*  
**have** 8:  $\vdash \triangleright(di(\mathcal{M} a) \wedge di(\triangleright((\mathcal{M} b) \vee (\mathcal{M} c)))) =$   
 $\mathcal{M}(a \text{ THRU } (b \text{ UPTO } c))$  **by** *simp*  
**from** 1 2 7 8 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

**lemma** *MThruUptoRDistrib*:

$$((a \text{ THRU } b) \text{ UPTO } c) \simeq ((a \text{ UPTO } c) \text{ THRU } (b \text{ UPTO } c))$$

**proof** –

**have** 1:  $((a \text{ THRU } b) \text{ UPTO } c) \simeq (c \text{ UPTO } (a \text{ THRU } b))$   
**using** *MUptoCommut* **by** *auto*  
**have** 2:  $(c \text{ UPTO } (a \text{ THRU } b)) \simeq ((c \text{ UPTO } a) \text{ THRU } (c \text{ UPTO } b))$   
**using** *MUptoThruDistrib* **by** *auto*  
**have** 3:  $(c \text{ UPTO } a) \simeq (a \text{ UPTO } c)$   
**using** *MUptoCommut* **by** *auto*  
**have** 4:  $(c \text{ UPTO } b) \simeq (b \text{ UPTO } c)$   
**using** *MUptoCommut* **by** *auto*  
**have** 5:  $((c \text{ UPTO } a) \text{ THRU } (c \text{ UPTO } b)) \simeq ((a \text{ UPTO } c) \text{ THRU } (b \text{ UPTO } c))$   
**using** 3 **by** (*simp add: MonEqRefl MonEqSubstThru*)  
**have** 6:  $((a \text{ UPTO } c) \text{ THRU } (c \text{ UPTO } b)) \simeq ((c \text{ UPTO } b) \text{ THRU } (a \text{ UPTO } c))$   
**using** *MThruCommut* **by** *auto*  
**have** 7:  $((c \text{ UPTO } b) \text{ THRU } (a \text{ UPTO } c)) \simeq ((b \text{ UPTO } c) \text{ THRU } (a \text{ UPTO } c))$   
**using** 4 **by** (*simp add: MonEqRefl MonEqSubstThru*)  
**from** 1 2 5 6 7 **show** *?thesis* **using** *MThruCommut MonEq* **by** (*metis int-eq*)

qed

**lemma** *MUptoThruRDistrib*:

$$((a \text{ UPTO } b) \text{ THRU } c) \simeq ((a \text{ THRU } c) \text{ UPTO } (b \text{ THRU } c))$$

**proof** –

**have** 1:  $((a \text{ UPTO } b) \text{ THRU } c) \simeq (c \text{ THRU } (a \text{ UPTO } b))$

```

    using MThruCommut by auto
  have 2: (c THRU (a UPTO b))  $\simeq$  ((c THRU a) UPTO (c THRU b))
    using MThruUptoDistrib by auto
  have 3: (c THRU a)  $\simeq$  (a THRU c)
    using MThruCommut by auto
  have 4: (c THRU b)  $\simeq$  (b THRU c)
    using MThruCommut by auto
  have 5: ((c THRU a) UPTO (c THRU b))  $\simeq$  ((a THRU c) UPTO (c THRU b))
    using 3 by (simp add: MonEqRefl MonEqSubstUpto)
  have 6: ((a THRU c) UPTO (c THRU b))  $\simeq$  ((c THRU b) UPTO (a THRU c))
    using MUptoCommut by auto
  have 7: ((c THRU b) UPTO (a THRU c))  $\simeq$  ((b THRU c) UPTO (a THRU c))
    using 4 by (simp add: MonEqRefl MonEqSubstUpto)
  from 1 2 5 6 7 show ?thesis using MUptoCommut MonEq by (metis int-eq)
qed

```

**lemma** *MWithAndDistrib*:

```

((a AND b) WITH f)  $\simeq$  ((a WITH f) AND (b WITH f))
proof –
  have 1:  $\vdash \mathcal{M}((a \text{ AND } b) \text{ WITH } f) = (\mathcal{M}(a \text{ AND } b) \wedge f)$ 
    by (simp)
  have 2:  $\vdash \mathcal{M}(a \text{ AND } b) = \mathcal{M}(a \text{ WITH LIFT}(\mathcal{M} \ b))$ 
    by (simp add: AND-d-def)
  have 3:  $\vdash (\mathcal{M}(a \text{ AND } b) \wedge f) = (\mathcal{M}(a \text{ WITH LIFT}(\mathcal{M} \ b)) \wedge f)$ 
    using 2 by auto
  have 4:  $\vdash \mathcal{M}(a \text{ WITH (LIFT( } \mathcal{M} \ b) \wedge f)) = (\mathcal{M}(a) \wedge \mathcal{M}(b) \wedge f)$ 
    by simp
  have 5:  $\vdash (\mathcal{M}(a) \wedge \mathcal{M}(b) \wedge f) = ((\mathcal{M}(a) \wedge f) \wedge (\mathcal{M}(b) \wedge f))$ 
    by auto
  have 6:  $\vdash ((\mathcal{M}(a) \wedge f) \wedge (\mathcal{M}(b) \wedge f)) = (\mathcal{M}(a \text{ WITH } f) \wedge \mathcal{M}(b \text{ WITH } f))$ 
    by simp
  have 7:  $\vdash (\mathcal{M}(a \text{ WITH } f) \wedge \mathcal{M}(b \text{ WITH } f)) = \mathcal{M}((a \text{ WITH } f) \text{ WITH LIFT}(\mathcal{M}(b \text{ WITH } f)))$ 
    by simp
  have 8:  $\vdash \mathcal{M}((a \text{ WITH } f) \text{ WITH LIFT}(\mathcal{M}(b \text{ WITH } f))) = \mathcal{M}((a \text{ WITH } f) \text{ AND } (b \text{ WITH } f))$ 
    by (simp add: AND-d-def)
  from 1 2 3 4 5 6 7 8 show ?thesis using MonEq by (metis AND-d-def MWithAbsorp int-eq)
qed

```

**lemma** *MHaltWithAndDistrib*:

```

(((HALT w) WITH f) AND ((HALT w) WITH g))  $\simeq$  ((HALT w) WITH LIFT(f  $\wedge$  g))
proof –
  have 1:  $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ AND } ((\text{HALT } w) \text{ WITH } g)) =$ 
     $\mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ WITH LIFT}(\mathcal{M}((\text{HALT } w) \text{ WITH } g)))$ 
    by (simp add: AND-d-def)
  have 2:  $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ WITH LIFT}(\mathcal{M}((\text{HALT } w) \text{ WITH } g))) =$ 
     $(\mathcal{M}(\text{HALT } w) \wedge f \wedge \mathcal{M}(\text{HALT } w) \wedge g)$ 
    by auto
  have 3:  $\vdash (\mathcal{M}(\text{HALT } w) \wedge f \wedge \mathcal{M}(\text{HALT } w) \wedge g) = (\mathcal{M}(\text{HALT } w) \wedge f \wedge g)$ 
    by auto
  have 4:  $\vdash (\mathcal{M}(\text{HALT } w) \wedge f \wedge g) = \mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \wedge g))$ 

```

by *auto*  
 from 1 2 3 4 show ?thesis using MonEq by (metis int-eq)  
 qed

**lemma** *MHaltWithUptoHaltWithEqvHaltWithOr*:

$((\text{HALT } w) \text{ WITH } f) \text{ UPTO } ((\text{HALT } w) \text{ WITH } g)) \simeq ((\text{HALT } w) \text{ WITH LIFT}(f \vee g))$

**proof** –

have 1:  $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ UPTO } ((\text{HALT } w) \text{ WITH } g)) =$   
 $\triangleright(\mathcal{M}((\text{HALT } w) \text{ WITH } f) \vee \mathcal{M}((\text{HALT } w) \text{ WITH } g))$

by (*simp*)

have 2:  $\vdash \triangleright(\mathcal{M}((\text{HALT } w) \text{ WITH } f) \vee \mathcal{M}((\text{HALT } w) \text{ WITH } g)) =$   
 $\triangleright((\mathcal{M}(\text{HALT } w) \wedge f) \vee (\mathcal{M}(\text{HALT } w) \wedge g))$

by *auto*

have 3:  $\vdash ((\mathcal{M}(\text{HALT } w) \wedge f) \vee (\mathcal{M}(\text{HALT } w) \wedge g)) = (\mathcal{M}(\text{HALT } w) \wedge (f \vee g))$

by *auto*

have 4:  $\vdash \triangleright((\mathcal{M}(\text{HALT } w) \wedge f) \vee (\mathcal{M}(\text{HALT } w) \wedge g)) = \triangleright(\mathcal{M}(\text{HALT } w) \wedge (f \vee g))$   
 using 3 *FstEqvRule* by *blast*

have 5:  $\vdash \triangleright(\mathcal{M}(\text{HALT } w) \wedge (f \vee g)) = \triangleright(\mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \vee g)))$

by *simp*

have 6:  $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH LIFT}(f \vee g))) = \triangleright(\mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \vee g)))$   
 using *MFixFst* by *blast*

from 1 2 3 4 5 6 show ?thesis using MonEq by (metis int-eq)

qed

**lemma** *MHaltWithThruHaltWithEqvHaltWithAndHaltWith*:

$((\text{HALT } w) \text{ WITH } f) \text{ THRU } ((\text{HALT } w) \text{ WITH } g)) \simeq (((\text{HALT } w) \text{ WITH } f) \text{ AND } ((\text{HALT } w) \text{ WITH } g))$

**proof** –

have 1:  $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ THRU } ((\text{HALT } w) \text{ WITH } g)) =$   
 $\triangleright(\text{di}(\mathcal{M}(\text{HALT } w) \wedge f) \wedge \text{di}(\mathcal{M}(\text{HALT } w) \wedge g))$

by *simp*

have 2:  $\vdash (\text{di}(\mathcal{M}(\text{HALT } w) \wedge f) \wedge \text{di}(\mathcal{M}(\text{HALT } w) \wedge g)) =$   
 $(\text{di}(\text{halt}(\text{init } w) \wedge f) \wedge \text{di}(\text{halt}(\text{init } w) \wedge g))$

using *MHaltAlt DiEqvDi*

by (metis (no-types, lifting) integ-reflection lift-and-com)

have 3:  $\vdash (\text{di}(\text{halt}(\text{init } w) \wedge f) \wedge \text{di}(\text{halt}(\text{init } w) \wedge g)) =$   
 $\text{di}(\text{halt}(\text{init } w) \wedge f \wedge g)$

using *DiHaltAndDiHaltAndEqvDiHaltAndAnd* by *fastforce*

have 4:  $\vdash \text{di}(\text{halt}(\text{init } w) \wedge f \wedge g) = \text{di}(\mathcal{M}(\text{HALT } w) \wedge f \wedge g)$

by (metis *DiEqvDi MHaltAlt integ-reflection lift-and-com*)

have 5:  $\vdash (\text{di}(\mathcal{M}(\text{HALT } w) \wedge f) \wedge \text{di}(\mathcal{M}(\text{HALT } w) \wedge g)) = \text{di}(\mathcal{M}(\text{HALT } w) \wedge f \wedge g)$

using 2 3 4 by *fastforce*

have 6:  $\vdash \triangleright(\text{di}(\mathcal{M}(\text{HALT } w) \wedge f) \wedge \text{di}(\mathcal{M}(\text{HALT } w) \wedge g)) = \triangleright(\text{di}(\mathcal{M}(\text{HALT } w) \wedge f \wedge g))$

using 5 *FstEqvRule* by *blast*

have 7:  $\vdash \triangleright(\text{di}(\mathcal{M}(\text{HALT } w) \wedge f \wedge g)) = \triangleright(\mathcal{M}(\text{HALT } w) \wedge f \wedge g)$

using *FstDiEqvFst* by *fastforce*

have 8:  $\vdash \triangleright(\mathcal{M}(\text{HALT } w) \wedge f \wedge g) = \triangleright(\mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \wedge g)))$

by *simp*

have 9:  $\vdash \mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \wedge g)) = \triangleright(\mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \wedge g)))$

using *MFixFst* by *blast*

**have 10:**  $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ THRU } ((\text{HALT } w) \text{ WITH } g)) = \mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \wedge g))$   
**using** 1 2 3 4 5 6 7 8 9 *int-eq* **by** *metis*  
**have 11:**  $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ AND } ((\text{HALT } w) \text{ WITH } g)) = \mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \wedge g))$   
**using** *MHaltWithAndDistrib* **using** *eq-d-def* **by** *blast*  
**have 12:**  $\vdash \mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \wedge g)) = \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ AND } ((\text{HALT } w) \text{ WITH } g))$   
**using** 11 **by** *fastforce*  
**from 10 12 show ?thesis using MonEq by (metis int-eq)**  
**qed**

**lemma** *MThenAndDistrib*:

$(a \text{ THEN } (b \text{ AND } c)) \simeq ((a \text{ THEN } b) \text{ AND } (a \text{ THEN } c))$

**proof** –

**have 1:**  $\vdash \mathcal{M}(a \text{ THEN } (b \text{ AND } c)) = (\mathcal{M}(a)) ; (\mathcal{M}(b \text{ AND } c))$   
**by** *simp*  
**have 2:**  $\vdash (\mathcal{M}(a)) ; (\mathcal{M}(b \text{ AND } c)) = (\mathcal{M}(a)) ; (\mathcal{M}(b) \wedge \mathcal{M}(c))$   
**by** (*simp add: AND-d-def*)  
**have 3:**  $\vdash (\mathcal{M}(a)) ; (\mathcal{M}(b) \wedge \mathcal{M}(c)) = \triangleright (\mathcal{M}(a)) ; (\mathcal{M}(b) \wedge \mathcal{M}(c))$   
**using** *MFixFst LeftChopEqvChop* **by** *blast*  
**have 4:**  $\vdash \triangleright (\mathcal{M}(a)) ; (\mathcal{M}(b) \wedge \mathcal{M}(c)) = ((\triangleright (\mathcal{M}(a)) ; (\mathcal{M}(b))) \wedge (\triangleright (\mathcal{M}(a)) ; (\mathcal{M}(c))))$   
**using** *LFstAndDistrC* **by** *fastforce*  
**have 5:**  $\vdash ((\triangleright (\mathcal{M}(a)) ; (\mathcal{M}(b))) \wedge (\triangleright (\mathcal{M}(a)) ; (\mathcal{M}(c)))) =$   
 $(( (\mathcal{M}(a)) ; (\mathcal{M}(b))) \wedge ( (\mathcal{M}(a)) ; (\mathcal{M}(c)) ))$  **using** *MFixFst*  
**by** (*metis 4 inteq-reflection*)  
**have 6:**  $\vdash (( (\mathcal{M}(a)) ; (\mathcal{M}(b))) \wedge ( (\mathcal{M}(a)) ; (\mathcal{M}(c)) )) =$   
 $(\mathcal{M}(a \text{ THEN } b) \wedge \mathcal{M}(a \text{ THEN } c))$   
**by** *simp*  
**have 7:**  $\vdash (\mathcal{M}(a \text{ THEN } b) \wedge \mathcal{M}(a \text{ THEN } c)) = \mathcal{M}((a \text{ THEN } b) \text{ AND } (a \text{ THEN } c))$   
**by** (*simp add: AND-d-def*)  
**from 1 2 3 4 5 6 7 show ?thesis using MonEq by (metis int-eq)**  
**qed**

**lemma** *MThenUptoDistrib*:

$(a \text{ THEN } (b \text{ UPTO } c)) \simeq ((a \text{ THEN } b) \text{ UPTO } (a \text{ THEN } c))$

**proof** –

**have 1:**  $\vdash (\mathcal{M}(a \text{ THEN } (b \text{ UPTO } c))) = ((\mathcal{M} a); (\triangleright ((\mathcal{M} b) \vee (\mathcal{M} c))))$   
**by** *simp*  
**have 2:**  $\vdash ((\mathcal{M} a); (\triangleright ((\mathcal{M} b) \vee (\mathcal{M} c)))) = (\triangleright (\mathcal{M} a); (\triangleright ((\mathcal{M} b) \vee (\mathcal{M} c))))$   
**by** (*simp add: MFixFst LeftChopEqvChop*)  
**have 3:**  $\vdash (\triangleright (\mathcal{M} a); (\triangleright ((\mathcal{M} b) \vee (\mathcal{M} c)))) = ((\triangleright (\triangleright (\mathcal{M} a); ((\mathcal{M} b) \vee (\mathcal{M} c))))$   
**using** *FstFstChopEqvFstChopFst* **by** *fastforce*  
**have 4:**  $\vdash \triangleright (\mathcal{M} a); ((\mathcal{M} b) \vee (\mathcal{M} c)) = (\mathcal{M} a); ((\mathcal{M} b) \vee (\mathcal{M} c))$   
**using** *MFixFst* **by** (*metis LeftChopEqvChop inteq-reflection*)  
**have 5:**  $\vdash (\mathcal{M} a); ((\mathcal{M} b) \vee (\mathcal{M} c)) = ((\mathcal{M} a); (\mathcal{M} b) \vee (\mathcal{M} a); (\mathcal{M} c))$   
**by** (*simp add: ChopOrEqv*)  
**have 6:**  $\vdash ((\mathcal{M} a); (\mathcal{M} b) \vee (\mathcal{M} a); (\mathcal{M} c)) = (\mathcal{M}(a \text{ THEN } b) \vee \mathcal{M}(a \text{ THEN } c))$   
**by** *simp*  
**have 7:**  $\vdash \triangleright (\mathcal{M} a); ((\mathcal{M} b) \vee (\mathcal{M} c)) = (\mathcal{M}(a \text{ THEN } b) \vee \mathcal{M}(a \text{ THEN } c))$   
**using** 6 5 4 **by** *fastforce*  
**have 8:**  $\vdash \triangleright (\triangleright (\mathcal{M} a); ((\mathcal{M} b) \vee (\mathcal{M} c))) = \triangleright (\mathcal{M}(a \text{ THEN } b) \vee \mathcal{M}(a \text{ THEN } c))$

using 7 by (simp add: FstEqvRule)  
 have 9:  $\vdash \triangleright(\mathcal{M}(a \text{ THEN } b) \vee \mathcal{M}(a \text{ THEN } c)) = \mathcal{M}((a \text{ THEN } b) \text{ UPTO } (a \text{ THEN } c))$   
 by simp  
 from 9 7 1 2 3 show ?thesis by (metis eq-d-def inteq-reflection)  
 qed

**lemma** *MTheinthruDistrib*:  
 $(a \text{ THEN } (b \text{ THRU } c)) \simeq ((a \text{ THEN } b) \text{ THRU } (a \text{ THEN } c))$   
**proof** –  
 have 1:  $\vdash \mathcal{M}(a \text{ THEN } (b \text{ THRU } c)) = (\mathcal{M} a); \triangleright (di(\mathcal{M} b) \wedge di(\mathcal{M} c))$   
 by simp  
 have 2:  $\vdash (\mathcal{M} a); \triangleright (di(\mathcal{M} b) \wedge di(\mathcal{M} c)) = \triangleright(\mathcal{M} a); \triangleright (di(\mathcal{M} b) \wedge di(\mathcal{M} c))$   
 by (simp add: MFixFst LeftChopEqvChop)  
 have 3:  $\vdash \triangleright(\mathcal{M} a); \triangleright (di(\mathcal{M} b) \wedge di(\mathcal{M} c)) = \triangleright(\triangleright(\mathcal{M} a); (di(\mathcal{M} b) \wedge di(\mathcal{M} c)))$   
 using FstFstChopEqvFstChopFst by fastforce  
 have 4:  $\vdash \triangleright(\mathcal{M} a); (di(\mathcal{M} b) \wedge di(\mathcal{M} c)) = (\triangleright(\mathcal{M} a); di(\mathcal{M} b) \wedge \triangleright(\mathcal{M} a); di(\mathcal{M} c))$   
 by (meson LFstAndDistrC Prop11)  
 have 5:  $\vdash (\triangleright(\mathcal{M} a); di(\mathcal{M} b) \wedge \triangleright(\mathcal{M} a); di(\mathcal{M} c)) = ((\mathcal{M} a); di(\mathcal{M} b) \wedge (\mathcal{M} a); di(\mathcal{M} c))$   
 using MFixFst by (metis 4 int-eq)  
 have 6:  $\vdash (\mathcal{M} a); di(\mathcal{M} b) = (\mathcal{M} a); ((\mathcal{M} b); \# \text{True})$   
 by (simp add: di-d-def)  
 have 7:  $\vdash (\mathcal{M} a); ((\mathcal{M} b); \# \text{True}) = ((\mathcal{M} a); (\mathcal{M} b)); \# \text{True}$   
 by (simp add: ChopAssoc)  
 have 8:  $\vdash ((\mathcal{M} a); (\mathcal{M} b)); \# \text{True} = di((\mathcal{M} a); (\mathcal{M} b))$   
 by (simp add: di-d-def)  
 have 9:  $\vdash (\mathcal{M} a); di(\mathcal{M} b) = di((\mathcal{M} a); (\mathcal{M} b))$   
 using 8 7 6 by fastforce  
 have 10:  $\vdash (\mathcal{M} a); di(\mathcal{M} c) = (\mathcal{M} a); ((\mathcal{M} c); \# \text{True})$   
 by (simp add: di-d-def)  
 have 11:  $\vdash (\mathcal{M} a); ((\mathcal{M} c); \# \text{True}) = ((\mathcal{M} a); (\mathcal{M} c)); \# \text{True}$   
 by (simp add: ChopAssoc)  
 have 12:  $\vdash ((\mathcal{M} a); (\mathcal{M} c)); \# \text{True} = di((\mathcal{M} a); (\mathcal{M} c))$   
 by (simp add: di-d-def)  
 have 13:  $\vdash (\mathcal{M} a); di(\mathcal{M} c) = di((\mathcal{M} a); (\mathcal{M} c))$   
 using 12 11 10 by fastforce  
 have 14:  $\vdash ((\mathcal{M} a); di(\mathcal{M} b) \wedge (\mathcal{M} a); di(\mathcal{M} c)) = (di((\mathcal{M} a); (\mathcal{M} b)) \wedge di((\mathcal{M} a); (\mathcal{M} c)))$   
 using 13 9 by fastforce  
 have 15:  $\vdash (di((\mathcal{M} a); (\mathcal{M} b)) \wedge di((\mathcal{M} a); (\mathcal{M} c))) = (di(\mathcal{M}(a \text{ THEN } b)) \wedge di(\mathcal{M}(a \text{ THEN } c)))$   
 by simp  
 have 16:  $\vdash \triangleright(\mathcal{M} a); (di(\mathcal{M} b) \wedge di(\mathcal{M} c)) = (di(\mathcal{M}(a \text{ THEN } b)) \wedge di(\mathcal{M}(a \text{ THEN } c)))$   
 using 15 14 4 5 by fastforce  
 have 17:  $\vdash \triangleright(\triangleright(\mathcal{M} a); (di(\mathcal{M} b) \wedge di(\mathcal{M} c))) = \triangleright(di(\mathcal{M}(a \text{ THEN } b)) \wedge di(\mathcal{M}(a \text{ THEN } c)))$   
 using 16 by (simp add: FstEqvRule)  
 have 18:  $\vdash \triangleright(di(\mathcal{M}(a \text{ THEN } b)) \wedge di(\mathcal{M}(a \text{ THEN } c))) = \mathcal{M}((a \text{ THEN } b) \text{ THRU } (a \text{ THEN } c))$   
 by simp  
 from 18 16 1 2 3 show ?thesis by (metis eq-d-def int-eq)  
 qed

end

## 11 Finite ITL Examples

```
theory Example
imports
  FOTheorems TimeReversal
begin
```

### 11.1 Example 1

```
definition F1 :: nat statefun  $\Rightarrow$  temporal
where F1 w  $\equiv$  TEMP  $\Box$  ( #0  $\leq$  $w )
```

```
definition Init1 :: nat statefun  $\Rightarrow$  temporal
where Init1 w  $\equiv$  TEMP $w = #0
```

```
lemma init1:
  ( $\langle s0, s1, s2 \rangle \models \text{len}(2) \wedge \text{Init1 } w$ ) = ( $(w \ s0) = 0$ )
by (simp add: Init1-def itl-defs)
```

```
lemma exist-test-F1 :
   $\vdash \exists \exists w. F1 \ w$ 
proof -
  have 1:  $\bigwedge w. \vdash F1 \ w$  by (simp add: itl-defs F1-def Valid-def)
  from 1 show ?thesis by (meson EExI MP)
qed
```

### 11.2 Example 2

```
locale Test =
  fixes v :: state  $\Rightarrow$  nat
  fixes v1 :: state  $\Rightarrow$  nat
  fixes y :: state  $\Rightarrow$  bool
  fixes z :: state  $\Rightarrow$  int
  fixes F2 :: nat statefun  $\Rightarrow$  temporal
  fixes F3 :: bool statefun  $\Rightarrow$  temporal
  fixes F4 :: int statefun  $\Rightarrow$  temporal
  fixes F5 :: nat statefun  $\Rightarrow$  temporal
  fixes Init2 :: nat statefun  $\Rightarrow$  temporal
  fixes Init3 :: bool statefun  $\Rightarrow$  temporal
  defines F2  $\equiv$  ( $\lambda v. \text{TEMP } \Box$  ( #0  $\leq$  $v ))
  defines F3  $\equiv$  ( $\lambda p. \text{TEMP } \Box$  ( $p  $\vee$   $\neg$  $p ))
  defines F4  $\equiv$  ( $\lambda z. \text{TEMP } \Box$  ( #0  $\leq$  $z  $\vee$  $z < #0 ))
  defines F5  $\equiv$  ( $\lambda v. \text{TEMP } \$v = \#0 \wedge v \text{ gets } \$v + \#1$ )
  defines Init2  $\equiv$  ( $\lambda v. \text{TEMP } \$v = \#0$ )
  defines Init3  $\equiv$  ( $\lambda p. \text{TEMP } \$p$ )
```

**lemma** (in *Test*) *currentval-test* :  
 $(s \models (\$v = \#0)) = ( (v \text{ (inth } s \ 0)) = 0 )$   
**by** (*simp add: itl-def*)

**lemma** (in *Test*) *nextempty-test* :  
 $(\langle s0 \rangle \models v\$) = (\epsilon \ x. x=x)$   
**by** (*simp add: itl-def*)

**lemma** (in *Test*) *nextempty-test-1* :  
 $(\langle s0 \rangle \models v\$ = v\$)$   
**by** *simp*

**lemma** (in *Test*) *nextempty-test-2* :  
 $(\langle s0 \rangle \models v\$ = v1\$)$   
**by** (*simp add: Test.nextempty-test*)

**lemma** (in *Test*) *nextcurrent-test*:  
 $(\langle s0, s1 \rangle \models \text{skip} \wedge (\$v = \#0) \wedge (v\$ = \$v + \#1)) = (((v \ s0) = 0) \wedge ((v \ s1) = 1 \ ))$   
**unfolding** *itl-defs* **by** *auto*

**lemma** (in *Test*) *nextcurrentfinpenult-test*:  
 $(\langle s0, s1, s2, s3 \rangle \models \text{len}(\mathcal{J}) \wedge v =: !v - \#1 \wedge v \leftarrow \#3 \wedge \$v = \#0 \wedge v := \$v + \#1) =$   
 $((v \ s0) = 0) \wedge ((v \ s1) = 1 \wedge (v \ (s2)) = 2 \wedge ((v \ s3) = 3 \ ))$   
**by** (*simp add: itl-def itl-defs*)  
(*metis One-nat-def add-Suc-shift add-diff-cancel-right' le-add1 min.orderE min-OR numeral-2-eq-2*  
*numeral-3-eq-3 plus-1-eq-Suc zero-neq-one*)

**lemma** (in *Test*) *stable-test*:  
 $(\langle s0, s1, s2, s3 \rangle \models \text{len}(\mathcal{J}) \wedge \text{stable } v \wedge \$v = \#0) =$   
 $((v \ s0) = 0 \wedge (v \ s1) = 0 \wedge (v \ s2) = 0 \wedge (v \ s3) = 0)$   
**using** *nat-le-linear* **by** (*simp add: itl-defs Nitpick.case-nat-unfold, auto*)

**lemma** (in *Test*) *revnextcurrentfinpenult-test*:  
 $(\langle s0, s1, s2, s3 \rangle \models (\text{len } \mathcal{J} \wedge v! = !v - \#1 \wedge !v = \#3 \wedge \$v = \#0 \wedge v\$ = \$v + \#1)^r) =$   
 $((v \ s3) = 0) \wedge ((v \ s2) = 1 \wedge (v \ (s1)) = 2 \wedge ((v \ s0) = 3 \ ))$   
**unfolding** *reverse-d-def len-defs current-val-d-def next-val-d-def*  
*penult-val-d-def fin-val-d-def* **by** *auto*

**lemma** (in *Test*) *exist-test-F2* :  
 $\vdash \exists \exists \ v. F2 \ v$   
**proof** –  
**have** *1*:  $\vdash F2 \ v$  **by** (*simp add: itl-defs F2-def Valid-def*)  
**from** *1* **show** *?thesis* **by** (*meson EExI MP*)  
**qed**

**lemma** (in *Test*) *exist-test-F3* :  
 $\vdash \exists \exists \ y. F3 \ y$



```

proof –
  have 1:  $\vdash F3\ y$  by (simp add: itl-defs F3-def Valid-def)
  from 1 show ?thesis by (meson EExI MP)
qed

```

### 11.3 Example 3

```

locale Test1 =
  fixes  $v :: state \Rightarrow nat$ 
  fixes  $F5 :: nat \text{ statefun} \Rightarrow nat \Rightarrow temporal$ 
  defines  $F5 \equiv (\lambda\ v\ n.\ TEMP\ \$v = \#0 \wedge v\ gets\ \$v + \#1 \wedge fin(\$v = \#n))$ 

```

**lemma** (**in** Test1) *test-E-F5-1*:

```

(
   $x\ (inth\ w\ (0::nat)) = (0::nat) \wedge$ 
   $(\forall i < ilen\ w.\ x\ (inth\ w\ (Suc\ i)) = Suc\ (x\ (inth\ w\ i))) \wedge$ 
   $x\ (inth\ w\ (ilen\ w)) = n \longrightarrow$ 
  (
     $x\ (inth\ w\ (0::nat)) = (0::nat) \wedge$ 
     $(\forall i \leq ilen\ w.\ x\ (inth\ w\ (i)) = i) \wedge$ 
     $x\ (inth\ w\ (ilen\ w)) = n$ 
  )
proof auto
show  $\bigwedge i.\ x\ (inth\ w\ 0) = 0 \Longrightarrow$ 
   $\forall i < ilen\ w.\ x\ (inth\ w\ (Suc\ i)) = Suc\ (x\ (inth\ w\ i)) \Longrightarrow$ 
   $n = x\ (inth\ w\ (ilen\ w)) \Longrightarrow i \leq ilen\ w \Longrightarrow x\ (inth\ w\ i) = i$ 
proof –
  fix  $i$ 
  assume 0:  $x\ (inth\ w\ 0) = 0$ 
  assume 1:  $\forall i < ilen\ w.\ x\ (inth\ w\ (Suc\ i)) = Suc\ (x\ (inth\ w\ i))$ 
  assume 2:  $n = x\ (inth\ w\ (ilen\ w))$ 
  assume 3:  $i \leq ilen\ w$ 
  show  $x\ (inth\ w\ i) = i$ 
  using 0 1 2 3
  proof (induct i)
  case 0
  then show ?case by simp
  next
  case ( $Suc\ i$ )
  then show ?case by simp
  qed
qed
qed

```

**lemma** (**in** Test1) *test-E-F5-2*:

```

(
   $x\ (inth\ w\ (0::nat)) = (0::nat) \wedge$ 
   $(\forall i \leq ilen\ w.\ x\ (inth\ w\ (i)) = i) \wedge$ 
   $x\ (inth\ w\ (ilen\ w)) = n \longrightarrow$ 

```

$$\begin{aligned}
& x \text{ ( } \textit{inth} \ w \ (0::\textit{nat})) = (0::\textit{nat}) \wedge \\
& (\forall i < \textit{ilen} \ w. \ x \text{ ( } \textit{inth} \ w \ (\textit{Suc} \ i)) = \textit{Suc} \ (x \text{ ( } \textit{inth} \ w \ i))) \wedge \\
& x \text{ ( } \textit{inth} \ w \ (\textit{ilen} \ w)) = n
\end{aligned}$$

by *simp*

**lemma** (in *Test1*) *test-E-F5-3*:

$$\begin{aligned}
& ( \\
& \quad x \text{ ( } \textit{inth} \ w \ (0::\textit{nat})) = (0::\textit{nat}) \wedge \\
& \quad (\forall i < \textit{ilen} \ w. \ x \text{ ( } \textit{inth} \ w \ (\textit{Suc} \ i)) = \textit{Suc} \ (x \text{ ( } \textit{inth} \ w \ i))) \wedge \\
& \quad x \text{ ( } \textit{inth} \ w \ (\textit{ilen} \ w)) = n = \\
& ( \\
& \quad x \text{ ( } \textit{inth} \ w \ (0::\textit{nat})) = (0::\textit{nat}) \wedge \\
& \quad (\forall i \leq \textit{ilen} \ w. \ x \text{ ( } \textit{inth} \ w \ (i)) = i) \wedge \\
& \quad x \text{ ( } \textit{inth} \ w \ (\textit{ilen} \ w)) = n
\end{aligned}$$

using *test-E-F5-1 test-E-F5-2* by *auto*

**lemma** (in *Test1*) *test-E-F5-4*:

$$\begin{aligned}
& (\exists x::\textit{state} \Rightarrow \textit{nat}. \\
& \quad x \text{ ( } \textit{inth} \ w \ (0::\textit{nat})) = (0::\textit{nat}) \wedge \\
& \quad (\forall i < \textit{ilen} \ w. \ x \text{ ( } \textit{inth} \ w \ (\textit{Suc} \ i)) = \textit{Suc} \ (x \text{ ( } \textit{inth} \ w \ i))) \wedge \\
& \quad x \text{ ( } \textit{inth} \ w \ (\textit{ilen} \ w)) = n = \\
& (\exists x::\textit{state} \Rightarrow \textit{nat}. \\
& \quad x \text{ ( } \textit{inth} \ w \ (0::\textit{nat})) = (0::\textit{nat}) \wedge \\
& \quad (\forall i \leq \textit{ilen} \ w. \ x \text{ ( } \textit{inth} \ w \ (i)) = i) \wedge \\
& \quad x \text{ ( } \textit{inth} \ w \ (\textit{ilen} \ w)) = n
\end{aligned}$$

by (*simp add: Test1.test-E-F5-3*)

**lemma** (in *Test1*) *test-E-F5*:

$$\vdash (\exists \exists v. (F5 \ v \ n)) \longrightarrow (\textit{len} \ n)$$

by (*auto simp add: Valid-def F5-def exist-state-d-def itl-defs sub-def* )  
 (*metis Test1.test-E-F5-1 nat-le-linear*)

## 11.4 Example 4

**locale** *Testrev* =

**fixes**  $x :: \textit{state} \Rightarrow \textit{nat}$

**fixes**  $F1 :: \textit{nat} \textit{ statefun} \Rightarrow \textit{temporal}$

**defines**  $F1 \equiv (\lambda v. \textit{TEMP} \ \$v = \#0 \wedge \textit{skip} \wedge v := \$v + \#1)$

**lemma** (in *Testrev*) *testrev1*:

$$(\sigma \models F1 \ (x)) = (\textit{ilen} \ \sigma = 1 \wedge (x \text{ ( } \textit{inth} \ \sigma \ 0)) = 0 \wedge (x \text{ ( } \textit{inth} \ \sigma \ 1)) = 1)$$

by (*simp add: F1-def itl-defs next-assign-d-def, auto*)

**lemma** (in *Testrev*) *testrev2*:

$$(\sigma \models (F1 \ (x))^r) = (\textit{ilen} \ \sigma = 1 \wedge (x \text{ ( } \textit{inth} \ \sigma \ 0)) = 1 \wedge (x \text{ ( } \textit{inth} \ \sigma \ 1)) = 0)$$

**proof** –

**have**  $(\sigma \models (F1 \ (x))^r) = (\sigma \models (\$x = \#0 \wedge \textit{skip} \wedge x := \$x + \#1)^r)$

by (*simp add: F1-def*)

```

also have ... =
  ( $\sigma \models ((\$x = \#0)^r \wedge skip^r \wedge (x := \$x + \#1)^r)$ )
by (simp add: all-rev-eq)
also have ... =
  ( $\sigma \models (!x = \#0) \wedge skip \wedge (x! = !x + \#1)$ )
using RRAnd all-rev-eq
by (simp add: all-rev-eq(1) all-rev-eq(12) all-rev-eq(3) all-rev-eq(8) all-rev-eq(9)
    next-assign-d-def)
also have ... =
  ( $\sigma \models ((x\$ = \#0) \wedge skip \wedge (\$x = x\$ + \#1))$ )
by (simp add: itl-defs, auto)
also have ... =
  ( $ilen \sigma = 1 \wedge (x (inth \sigma 0)) = 1 \wedge (x (inth \sigma 1)) = 0$ )
by (simp add: itl-defs, auto)
finally show ( $\sigma \models (F1 (x))^r = (ilen \sigma = 1 \wedge (x (inth \sigma 0)) = 1 \wedge (x (inth \sigma 1)) = 0)$  .
qed

```

## 11.5 Example 5

**lemma** *revnextcurrentfinpenult*:

$\vdash (v\$ = \$v)^r = (v! = !v)$

**proof** –

**have** 1:  $\vdash (v\$ = \$v)^r = ( (v\$)^r = (\$v)^r )$  **by** (*simp add: rev-fun2*)

**have** 2:  $\vdash ((v\$)^r = (v!))$  **by** (*simp add: rev-next*)

**have** 3:  $\vdash ((\$v)^r = (!v))$  **by** (*simp add: rev-current*)

**have** 4:  $\vdash (( (v\$)^r = (\$v)^r ) = ( (v!) = (!v) ))$  **by** (*metis 1 2 3 inteq-reflection*)

**from** 1 4 **show** *?thesis* **by** *fastforce*

**qed**

**end**

## 12 Monitor Example

**theory** *MonitorExample*

**imports**

*FOTheorems Monitor*

**begin**

**locale** *Test* =

**fixes** *v* :: *state*  $\Rightarrow$  *nat*

**fixes** *y* :: *state*  $\Rightarrow$  *bool*

**fixes** *z* :: *state*  $\Rightarrow$  *nat*

**fixes** *F2* :: *nat statefun*  $\Rightarrow$  *temporal*

**fixes** *F3* :: *bool statefun*  $\Rightarrow$  *temporal*

**fixes** *F4* :: *nat statefun*  $\Rightarrow$  *temporal*

```

fixes  $F5 :: \text{nat statefun} \Rightarrow \text{temporal}$ 
fixes  $\text{Init2} :: \text{nat statefun} \Rightarrow \text{temporal}$ 
fixes  $\text{Init3} :: \text{bool statefun} \Rightarrow \text{temporal}$ 
fixes  $\text{Mon1} :: \text{state monitor}$ 
fixes  $\text{Mon2} :: \text{state monitor}$ 
fixes  $\text{Mon3} :: \text{state monitor}$ 
fixes  $\text{Mon4} :: \text{state monitor}$ 
fixes  $\text{Mon5} :: \text{state monitor}$ 
fixes  $\text{Mon6} :: \text{state monitor}$ 
defines  $F2 \equiv (\lambda v. \text{TEMP } \Box ( \#0 \leq \$v ))$ 
defines  $F3 \equiv (\lambda p. \text{TEMP } \Box ( \$p \vee \neg \$p ))$ 
defines  $F4 \equiv (\lambda z. \text{TEMP } \$z = \#0 \wedge z \text{ gets } \$z + \#1)$ 
defines  $F5 \equiv (\lambda z. \text{TEMP } \text{fin}(\$z = \#4))$ 
defines  $\text{Init2} \equiv (\lambda v. \text{TEMP } \$v = \#0)$ 
defines  $\text{Init3} \equiv (\lambda p. \text{TEMP } \$p)$ 
defines  $\text{Mon1} \equiv \text{FIRST}( F2 v )$ 
defines  $\text{Mon2} \equiv \text{EMPTY UPTO Mon1}$ 
defines  $\text{Mon3} \equiv \text{Mon1 WITH } (F2 v)$ 
defines  $\text{Mon4} \equiv \text{Mon2 THEN Mon1}$ 
defines  $\text{Mon5} \equiv \text{Mon3 THRU Mon4}$ 
defines  $\text{Mon6} \equiv (\text{FIRST } F4 z) \text{ WITH } (F5 z)$ 

```

**lemma** (in *Test*) *test*:

$\vdash \mathcal{M}(\text{Mon1}) = \text{empty}$

**proof** –

**have** 1:  $\vdash \mathcal{M}(\text{Mon1}) = \triangleright(\Box ( \#0 \leq \$v ))$

**using** *F2-def Mon1-def* **by** *fastforce*

**have** 2:  $\vdash \Box ( \#0 \leq \$v )$

**by** (*simp add: Valid-def itl-defs*)

**have** 3:  $\vdash \triangleright(\Box ( \#0 \leq \$v )) = \text{empty}$

**using** 2 **by** (*metis FstTrue int-eq int-eq-true*)

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

**qed**

**lemma** (in *Test*) *test1*:

$\vdash \mathcal{M}(\text{Mon2}) = \text{empty}$

**proof** –

**have** 1:  $\vdash \mathcal{M}(\text{Mon2}) = \mathcal{M}(\text{EMPTY UPTO Mon1})$

**using** *Mon2-def* **by** *fastforce*

**have** 2:  $\vdash \mathcal{M}(\text{EMPTY UPTO Mon1}) = \triangleright(\mathcal{M}(\text{EMPTY}) \vee \mathcal{M}(\text{Mon1}))$

**by** *fastforce*

**have** 3:  $\vdash \triangleright(\mathcal{M}(\text{EMPTY}) \vee \mathcal{M}(\text{Mon1})) = \triangleright(\text{empty} \vee \text{empty})$

**using** *test* **by** (*metis 2 MEmptyAlt int-eq*)

**have** 4:  $\vdash \triangleright(\text{empty} \vee \text{empty}) = \text{empty}$

**using** *FstEmptyOrEqvEmpty* **by** *blast*

**from** 1 2 3 4 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** (in *Test*) *test2*:

$\vdash \mathcal{M}(\text{Mon3}) = \text{empty}$

**proof** –  
**have** 1:  $\vdash \mathcal{M}(\text{Mon3}) = \mathcal{M}(\text{Mon1 WITH } (F2\ v))$  **using** *Mon3-def* **by** *fastforce*  
**have** 2:  $\vdash \mathcal{M}(\text{Mon1 WITH } (F2\ v)) = (\mathcal{M}(\text{Mon1}) \wedge (F2\ v))$  **by** *fastforce*  
**have** 3:  $\vdash (\mathcal{M}(\text{Mon1}) \wedge (F2\ v)) = (\text{empty} \wedge (F2\ v))$  **using** *test* **by** *fastforce*  
**have** 4:  $\vdash (F2\ v)$  **by** (*simp add: F2-def Valid-def itl-defs*)  
**have** 5:  $\vdash (\text{empty} \wedge (F2\ v)) = \text{empty}$  **using** 4 **by** *fastforce*  
**from** 1 2 3 5 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** (in *Test*) *test3*:

$\vdash \mathcal{M}(\text{Mon4}) = \text{empty}$

**proof** –

**have** 1:  $\vdash \mathcal{M}(\text{Mon4}) = \mathcal{M}(\text{Mon2 THEN Mon1})$   
**using** *Mon4-def* **by** *fastforce*  
**have** 2:  $\vdash \mathcal{M}(\text{Mon2 THEN Mon1}) = (\mathcal{M}(\text{Mon2}) \ ;\ (\mathcal{M}(\text{Mon1}) \ ))$   
**by** *fastforce*  
**have** 3:  $\vdash (\mathcal{M}(\text{Mon2}) \ ;\ (\mathcal{M}(\text{Mon1}) \ )) = \text{empty};\text{empty}$   
**using** *test test1* **using** *ChopEqvChop* **by** *blast*  
**have** 4:  $\vdash \text{empty};\ \text{empty} = \text{empty}$   
**by** (*simp add: ChopEmpty*)  
**from** 1 2 3 4 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** (in *Test*) *test4*:

$\vdash \mathcal{M}(\text{Mon5}) = \text{empty}$

**proof** –

**have** 1:  $\vdash \mathcal{M}(\text{Mon5}) = \mathcal{M}(\text{Mon3 THRU Mon4})$   
**using** *Mon5-def* **by** *fastforce*  
**have** 2:  $\vdash \mathcal{M}(\text{Mon3 THRU Mon4}) = \triangleright(\text{di}(\mathcal{M}(\text{Mon3})) \wedge \text{di}(\mathcal{M}(\text{Mon4})))$   
**by** *fastforce*  
**have** 3:  $\vdash (\text{di}(\mathcal{M}(\text{Mon3})) \wedge \text{di}(\mathcal{M}(\text{Mon4}))) = (\text{di}(\text{empty}) \wedge \text{di}(\text{empty}))$   
**using** *test3 test2* **by** (*metis inteq-reflection lift-and-com*)  
**hence** 4:  $\vdash \triangleright(\text{di}(\mathcal{M}(\text{Mon3})) \wedge \text{di}(\mathcal{M}(\text{Mon4}))) = \triangleright(\text{di}(\text{empty}) \wedge \text{di}(\text{empty}))$   
**by** (*simp add: FstEqvRule*)  
**have** 5:  $\vdash \triangleright(\text{di}(\text{empty}) \wedge \text{di}(\text{empty})) = \triangleright(\text{di}(\text{empty}))$   
**by** *simp*  
**have** 6:  $\vdash \triangleright(\text{di}(\text{empty})) = \text{empty}$   
**using** *FstDiEqvFst FstEmpty* **by** *fastforce*  
**from** 6 5 4 2 1 **show** *?thesis* **by** *fastforce*  
**qed**

**lemma** (in *Test*) *test5*:

$\vdash \mathcal{M}(\text{Mon6}) = (\triangleright(\$z = \#0 \ \wedge \ z \ \text{gets} \ \$z + \#1) \wedge \text{fin}(\$z = \#4) \ )$

**proof** –

**have** 1:  $\vdash \mathcal{M}(\text{Mon6}) = (\mathcal{M}(\text{FIRST } F4\ z) \wedge (F5\ z))$   
**using** *Mon6-def* **by** *fastforce*  
**have** 2:  $\vdash (\mathcal{M}(\text{FIRST } F4\ z) \wedge (F5\ z)) = (\triangleright(F4\ z) \wedge \text{fin}(\$z = \#4))$   
**using** *F5-def* **by** *fastforce*  
**have** 3:  $\vdash (\triangleright(F4\ z) \wedge \text{fin}(\$z = \#4)) = (\triangleright(\$z = \#0 \ \wedge \ z \ \text{gets} \ \$z + \#1) \wedge \text{fin}(\$z = \#4) \ )$   
**using** *F4-def* **by** *fastforce*

from 1 2 3 show ?thesis by fastforce  
qed

lemma (in Test) test5-1:

$\vdash \triangleright (\$z=\#0 \wedge z \text{ gets } \$z+\#1) \wedge \text{fin}(\$z=\#4) \longrightarrow$   
 $\triangleright ((\$z=\#0 \wedge z \text{ gets } \$z+\#1) \wedge \text{fin}(\$z=\#4))$

using FstWithAndImp by blast

lemma (in Test) test5-2:

$(s \models (\$z=\#0 \wedge z \text{ gets } \$z+\#1) \wedge \text{fin}(\$z=\#4)) =$   
 $(z (\text{inth } s \ 0) = 0 \wedge (\forall i < \text{ilen } s. z (\text{inth } s \ (\text{Suc } i)) = \text{Suc}(z (\text{inth } s \ i))) \wedge$   
 $z (\text{inth } s \ (\text{ilen } s)) = 4)$   
 by (simp add: itl-defs sub-def)

lemma (in Test) test5-3:

$(z (\text{inth } s \ 0) = 0 \wedge (\forall i < \text{ilen } s. z (\text{inth } s \ (\text{Suc } i)) = \text{Suc}(z (\text{inth } s \ i))) \wedge$   
 $z (\text{inth } s \ (\text{ilen } s)) = 4)$   
 $\implies$   
 $(z (\text{inth } s \ 0) = 0 \wedge (\forall i \leq \text{ilen } s. z (\text{inth } s \ i) = i)$   
 $\wedge z (\text{inth } s \ (\text{ilen } s)) = 4)$

proof –

assume 0:  $(z (\text{inth } s \ 0) = 0 \wedge (\forall i < \text{ilen } s. z (\text{inth } s \ (\text{Suc } i)) = \text{Suc}(z (\text{inth } s \ i))) \wedge$   
 $z (\text{inth } s \ (\text{ilen } s)) = 4)$

show  $(z (\text{inth } s \ 0) = 0 \wedge (\forall i \leq \text{ilen } s. z (\text{inth } s \ i) = i)$   
 $\wedge z (\text{inth } s \ (\text{ilen } s)) = 4)$

proof –

have 1:  $z (\text{inth } s \ 0) = 0$  using 0 by auto

have 2:  $z (\text{inth } s \ (\text{ilen } s)) = 4$  using 0 by auto

have 3:  $(\forall i \leq \text{ilen } s. z (\text{inth } s \ i) = i)$

proof

fix i

show  $i \leq \text{ilen } s \longrightarrow z (\text{inth } s \ i) = i$

proof

(induct i)

case 0

then show ?case by (simp add: 1)

next

case (Suc i)

then show ?case by (simp add: 0)

qed

qed

from 1 2 3 show ?thesis by auto

qed

qed

lemma (in Test) test5-4:

$(z (\text{inth } s \ 0) = 0 \wedge (\forall i \leq \text{ilen } s. z (\text{inth } s \ i) = i)$   
 $\wedge z (\text{inth } s \ (\text{ilen } s)) = 4) \implies$

$(z \text{ (inth } s \ 0) = 0 \ \wedge (\forall \ i < \text{ilen } s. \ z \text{ (inth } s \ (\text{Suc } i)) = \text{Suc}(z \text{ (inth } s \ i))) \wedge$   
 $z \text{ (inth } s \ (\text{ilen } s)) = 4)$

**proof** –

**assume** 0:  $(z \text{ (inth } s \ 0) = 0 \ \wedge (\forall \ i \leq \text{ilen } s. \ z \text{ (inth } s \ i) = i)$

$\wedge z \text{ (inth } s \ (\text{ilen } s)) = 4)$

**show**  $(z \text{ (inth } s \ 0) = 0 \ \wedge (\forall \ i < \text{ilen } s. \ z \text{ (inth } s \ (\text{Suc } i)) = \text{Suc}(z \text{ (inth } s \ i))) \wedge$   
 $z \text{ (inth } s \ (\text{ilen } s)) = 4)$

**proof** –

**have** 1:  $z \text{ (inth } s \ 0) = 0$  **using** 0 **by** *auto*

**have** 2:  $z \text{ (inth } s \ (\text{ilen } s)) = 4$  **using** 0 **by** *auto*

**have** 3:  $(\forall \ i < \text{ilen } s. \ z \text{ (inth } s \ (\text{Suc } i)) = \text{Suc}(z \text{ (inth } s \ i)))$  **by** (*simp add: 0*)

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

**qed**

**qed**

**lemma** (**in** *Test*) *test5-5*:

$(z \text{ (inth } s \ 0) = 0 \ \wedge (\forall \ i < \text{ilen } s. \ z \text{ (inth } s \ (\text{Suc } i)) = \text{Suc}(z \text{ (inth } s \ i))) \wedge$   
 $z \text{ (inth } s \ (\text{ilen } s)) = 4)$

=

$(z \text{ (inth } s \ 0) = 0 \ \wedge (\forall \ i \leq \text{ilen } s. \ z \text{ (inth } s \ i) = i)$   
 $\wedge z \text{ (inth } s \ (\text{ilen } s)) = 4)$

**using** *test5-3 test5-4* **by** *blast*

**lemma** (**in** *Test*) *test5-6* :

$(z \text{ (inth } s \ 0) = 0 \ \wedge (\forall \ i \leq \text{ilen } s. \ z \text{ (inth } s \ i) = i)$

$\wedge z \text{ (inth } s \ (\text{ilen } s)) = 4) =$

$(\text{ilen } s = 4 \wedge (\forall \ i \leq \text{ilen } s. \ z \text{ (inth } s \ i) = i) )$

**by** *auto*

**lemma** (**in** *Test*) *test5-7* :

$(s \models (\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) =$

$(\text{ilen } s = 4 \wedge (\forall \ i \leq \text{ilen } s. \ z \text{ (inth } s \ i) = i) )$

**using** *test5-6 test5-5 test5-2* **by** *fastforce*

**lemma** (**in** *Test*) *test5-8* :

$(s \models \triangleright((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))) =$

(

$(s \models (\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{ilen } s = 0) \vee$

$(0 < \text{ilen } s \wedge (s \models \$z = \#0 \wedge z \text{ gets } \$z + \#1 \wedge \text{fin}(\$z = \#4)) \wedge$

$(\forall \ ia < \text{ilen } s. \ (\text{prefix } ia \ s \models \neg((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))))$

)

**using** *Fstsem[of TEMP (\$z = \#0 \wedge z \text{ gets } \\$z + \#1) \wedge \text{fin}(\\$z = \#4)]*

**by** *simp*

**lemma** (**in** *Test*) *test5-9* :

$\neg(s \models (\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{ilen } s = 0)$

using test5-7 by simp

lemma (in Test) test5-10:

$(s \models (\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))$

$\implies$

$0 < \text{ilen } s \wedge$

$(\forall ia < \text{ilen } s. (\text{prefix } ia \ s \models \neg((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))))$

proof -

assume 0:  $s \models (\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)$

show  $0 < \text{ilen } s \wedge$

$(\forall ia < \text{ilen } s. (\text{prefix } ia \ s \models \neg((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))))$

proof -

have 1:  $0 < \text{ilen } s$  using test5-7 0 by simp

have 2:  $(\forall ia < \text{ilen } s. (\text{prefix } ia \ s \models \neg((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))))$

proof

fix ia

show  $ia < \text{ilen } s \implies$

$(\text{prefix } ia \ s \models \neg((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)))$

proof -

have 1:  $(\text{prefix } ia \ s \models \neg((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))) =$   
 $(\neg((\text{prefix } ia \ s \models ((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))))$

by auto

have 2:  $(\text{prefix } ia \ s \models ((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))) =$   
 $(\text{ilen}(\text{prefix } ia \ s) = 4 \wedge (\forall i \leq \text{ilen}(\text{prefix } ia \ s). z(\text{inth}(\text{prefix } ia \ s) \ i) = i))$   
 using test5-7 by simp

have 3:  $ia < \text{ilen } s \implies \neg(\text{ilen}(\text{prefix } ia \ s) = 4 \wedge$   
 $(\forall i \leq \text{ilen}(\text{prefix } ia \ s). z(\text{inth}(\text{prefix } ia \ s) \ i) = i))$

using 0 using test5-7 by auto

from 1 2 3 show ?thesis by blast

qed

qed

from 1 2 show ?thesis by auto

qed

qed

lemma (in Test) test5-11 :

$(s \models \triangleright((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))) =$

$(s \models (\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))$

using test5-8 test5-9 test5-10 by fastforce

lemma (in Test) test5-12 :

$\vdash \triangleright((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) = ((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))$

using test5-11 by (simp add: Valid-def)

end



## 13 Filter on Intervals

```
theory IFilter
  imports
    Interval
begin
```

The ifilter operator on intervals is defined. The definition of ifilter is slightly more complicated than the one for lists as an interval has at least one state and one needs to ensure that the ifilter operator always returns an interval. The lemmas involving the ifilter on intervals are similar to those for the filter operator on lists only a bit more complicated.

### 13.1 Definitions

```
definition opfx :: 'a interval  $\Rightarrow$  'a interval  $\Rightarrow$  bool
  where opfx xs ys = ( $\exists$  zs. ys = xs  $\ominus$  zs  $\vee$  ys = xs)
```

```
definition sopfx :: 'a interval  $\Rightarrow$  'a interval  $\Rightarrow$  bool
  where sopfx xs ys  $\longleftrightarrow$  opfx xs ys  $\wedge$  xs  $\neq$  ys
```

```
interpretation opfx-order: order opfx sopfx
```

```
proof standard
```

```
  show  $\bigwedge x y. \text{sopfx } x y = (\text{opfx } x y \wedge \neg \text{opfx } y x)$ 
```

```
  by (metis add-leD1 add-le-same-cancel1 ilen-iapp le-add1 not-one-le-zero opfx-def
    sopfx-def)
```

```
  show  $\bigwedge x. \text{opfx } x x$ 
```

```
    by (auto simp add: opfx-def sopfx-def)
```

```
  show  $\bigwedge x y z. \text{opfx } x y \Longrightarrow \text{opfx } y z \Longrightarrow \text{opfx } x z$ 
```

```
    by (auto simp add: opfx-def sopfx-def)
```

```
  show  $\bigwedge x y. \text{opfx } x y \Longrightarrow \text{opfx } y x \Longrightarrow x = y$ 
```

```
  by (metis add-leD1 add-le-same-cancel1 ilen-iapp le-add1 not-one-le-zero opfx-def)
```

```
qed
```

```
definition osfx :: 'a interval  $\Rightarrow$  'a interval  $\Rightarrow$  bool
  where osfx xs ys = ( $\exists$  zs. ys = zs  $\ominus$  xs  $\vee$  ys = xs)
```

```
definition sosfx :: 'a interval  $\Rightarrow$  'a interval  $\Rightarrow$  bool
```

```
  where sosfx xs ys  $\longleftrightarrow$  osfx xs ys  $\wedge$  xs  $\neq$  ys
```

```
interpretation osfx-order: order osfx sosfx
```

```
proof standard
```

```
  show  $\bigwedge x y. \text{sosfx } x y = (\text{osfx } x y \wedge \neg \text{osfx } y x)$ 
```

```
  by (metis irev-iapp irev-irev-ident opfx-def opfx-order.dual-order.antisym
    osfx-def sosfx-def)
```

```
  show  $\bigwedge x. \text{osfx } x x$ 
```

```
    by (auto simp add: osfx-def sosfx-def)
```

```
  show  $\bigwedge x y z. \text{osfx } x y \Longrightarrow \text{osfx } y z \Longrightarrow \text{osfx } x z$ 
```

```
    by (auto simp add: osfx-def sosfx-def)
```

```
    (metis iapp-assoc)
```

**show**  $\bigwedge x y. \text{osfx } x y \implies \text{osfx } y x \implies x = y$   
**using**  $\langle \bigwedge y x. \text{sosfx } x y = (\text{osfx } x y \wedge \neg \text{osfx } y x) \rangle$  *sosfx-def* **by** *blast*  
**qed**

**primrec** *idistinct* ::  $'a \text{ interval} \Rightarrow \text{bool}$   
**where** *idistinct*  $\langle x \rangle \longleftrightarrow \text{True}$   
 $| \text{ idistinct } (x \odot xs) \longleftrightarrow x \notin \text{iset } xs \wedge \text{ idistinct } xs$

**primrec** *iremdups* ::  $'a \text{ interval} \Rightarrow 'a \text{ interval}$   
**where** *iremdups*  $\langle x \rangle = \langle x \rangle$   
 $| \text{ iremdups } (x \odot xs) = (\text{if } x \in \text{iset } xs \text{ then } \text{iremdups } xs \text{ else } x \odot \text{iremdups } xs)$

**primrec** *ifilter* ::  $('a \Rightarrow \text{bool}) \Rightarrow 'a \text{ interval} \Rightarrow 'a \text{ interval}$   
**where** *ifilter*  $P \langle x \rangle = \langle x \rangle$   
 $| \text{ ifilter } P (x \odot xs) = (\text{if } (\exists y \in \text{iset } xs. P y) \text{ then}$   
 $\quad (\text{if } P x \text{ then } x \odot \text{ifilter } P xs \text{ else } \text{ifilter } P xs)$   
 $\quad \text{else } \langle x \rangle)$

**primrec** *nfilter* ::  $('a \Rightarrow \text{bool}) \Rightarrow 'a \text{ interval} \Rightarrow \text{nat} \Rightarrow \text{nat interval}$   
**where** *nfilter*  $P \langle x \rangle n = \langle n \rangle$   
 $| \text{ nfilter } P (x \odot xs) n = (\text{if } (\exists y \in \text{iset } xs. P y) \text{ then}$   
 $\quad (\text{if } P x \text{ then } n \odot (\text{nfilter } P xs (\text{Suc } n))$   
 $\quad \text{else } \text{nfilter } P xs (\text{Suc } n))$   
 $\quad \text{else } \langle n \rangle)$

**primrec** *prefixes* ::  $'a \text{ interval} \Rightarrow 'a \text{ interval interval}$   
**where**  
 $\text{ prefixes } \langle x \rangle = \langle \langle x \rangle \rangle$   
 $| \text{ prefixes } (x \odot xs) = \langle x \rangle \odot (\text{imap } ((\odot) x) (\text{prefixes } xs))$

**primrec** *suffixes* ::  $'a \text{ interval} \Rightarrow 'a \text{ interval interval}$   
**where**  
 $\text{ suffixes } \langle x \rangle = \langle \langle x \rangle \rangle$   
 $| \text{ suffixes } (x \odot xs) = (x \odot xs) \odot (\text{suffixes } xs)$

## 13.2 Lemmas

### 13.2.1 opfx and sopfx

**lemma** *opfxI* [*intro?*]:  
**assumes**  $ys = xs \ominus zs \vee ys = xs$   
**shows**  $\text{opfx } xs \ ys$   
**using** *assms* **unfolding** *opfx-def* **by** *blast*

**lemma** *opfxE* [*elim?*]:  
**assumes**  $\text{opfx } xs \ ys$   
**obtains**  $zs$  **where**  $ys = xs \ominus zs \vee ys = xs$   
**using** *assms* **unfolding** *opfx-def* **by** *blast*

**lemma** *sopfxI'* [*intro?*]:  
 $ys = xs \ominus (zs) \implies \text{sopfx } xs \ ys$   
**unfolding** *sopfx-def opfx-def*  
**by** (*metis add-le-same-cancel1 ilen-iapp le-add1 not-one-le-zero*)

**lemma** *sopfxE'* [*elim?*]:  
**assumes** *sopfx xs ys*  
**obtains** *zs* **where**  $ys = xs \ominus zs$   
**using** *assms* **unfolding** *sopfx-def opfx-def* **by** *blast*

**lemma** *opfx-INil* [*simp*]:  
 $\text{opfx } \langle \text{ifirst } xs \rangle \ xs$   
**unfolding** *opfx-def*  
**by** (*metis iapp-INil ifirst-iapp2 interval.exhaust inth.simps(1)*)

**lemma** *opfx-snoc* [*simp*]:  
 $\text{opfx } xs \ (ys \ominus \langle y \rangle) \longleftrightarrow xs = ys \ominus \langle y \rangle \vee \text{opfx } xs \ ys$   
**unfolding** *opfx-def*  
**by** (*metis iapp-eq-iapp-conv2 iapp-not-INil*)

**lemma** *ICons-pfx-ICons* [*simp*]:  
 $\text{opfx } (x \odot xs) \ (y \odot ys) = (x = y \wedge \text{opfx } xs \ ys)$   
**by** (*auto simp add: opfx-def*)

**lemma** *opfx-code* [*code*]:  
 $\text{opfx } \langle \text{ifirst } xs \rangle \ xs \longleftrightarrow \text{True}$   
 $\text{opfx } (x \odot xs) \ \langle y \rangle \longleftrightarrow \text{False}$   
 $\text{opfx } (x \odot xs) \ (y \odot ys) \longleftrightarrow (x = y \wedge \text{opfx } xs \ ys)$   
**proof** *simp-all*  
**show**  $\neg \text{opfx } (x \odot xs) \ \langle y \rangle$   
**by** (*simp add: opfx-def*)  
**qed**

**lemma** *same-opfx-opfx* [*simp*]:  
 $\text{opfx } (xs \ominus ys) \ (xs \ominus zs) = \text{opfx } ys \ zs$   
**by** (*induct xs*) *simp-all*

**lemma** *same-opfx-INil* [*simp*]:  
 $\text{opfx } (xs \ominus ys) \ (xs \ominus \langle y \rangle) = (ys = \langle y \rangle)$   
**by** (*meson same-iapp-eq opfx-order.less-le-not-le opfx-snoc sopfxI'*)

**lemma** *opfx-opfx* [*simp*]:  
**assumes** *opfx xs ys*  
**shows**  $\text{opfx } xs \ (ys \ominus zs)$   
**using** *assms* **unfolding** *opfx-def* **by** *fastforce*

**lemma** *iapp-opfxD*:  
**assumes** *opfx (xs  $\ominus$  ys) zs*  
**shows**  $\text{opfx } xs \ zs$

**using** *assms* **by** (*auto simp add: opfx-def*)

**lemma** *opfx-ICons*:

*opfx xs (y⊙ys) = (xs = ⟨y⟩ ∨ (∃ zs. xs = y⊙zs ∧ opfx zs ys))*  
**by** (*case-tac xs*) (*auto simp add: opfx-def*)

**lemma** *opfx-iapp*:

*opfx xs (ys⊖zs) = (opfx xs ys ∨ (∃ us. xs = ys ⊖ us ∧ opfx us zs))*  
**proof** (*induct zs rule: irev-induct*)  
**case** (*INil y*)  
**then show** ?*case* **by** (*meson opfx-snoc same-opfx-opfx*)  
**next**  
**case** (*snoc x xs*)  
**then show** ?*case* **by** (*metis iapp-assoc opfx-snoc*)  
**qed**

**lemma** *iapp-one-opfx*:

**assumes** *opfx xs ys*  
*ilen xs < ilen ys*  
**shows** *opfx (xs⊖ ⟨inth ys ((ilen xs)+1)⟩) ys*  
**using** *assms*  
**proof** (*unfold opfx-def*)  
**assume** 1:  $\exists zs. ys = xs \ominus zs \vee ys = xs$   
**then obtain** *sk* **where** 2:  $ys = xs \ominus sk \vee ys = xs$   
**by** *fastforce*  
**assume** 3:  $ilen xs < ilen ys$   
**have** 4:  $ilen sk \geq 0$   
**by** *auto*  
**have** 5:  $\exists v. ys = xs \ominus ((ifirst sk) \odot v) \vee ys = xs \ominus \langle ifirst sk \rangle$   
**by** (*metis 2 3 iapp-INil less-irrefl opfx-def opfx-INil*)  
**have** 6:  $ys \neq xs$   
**using** 3 **by** *blast*  
**have** 7:  $\langle ifirst sk \rangle = inth (xs \ominus sk) ((ilen xs) + 1)$   
**by** (*simp add: iapp-inth*)  
**have** 8:  $ys = xs \ominus sk$   
**using** 2 6 **by** *auto*  
**have** 9:  $inth (xs \ominus sk) ((ilen xs) + 1) = inth (ys) ((ilen xs) + 1)$   
**using** 8 **by** *blast*  
**thus**  $\exists zs. ys = (xs \ominus \langle inth ys ((ilen xs) + 1) \rangle) \ominus zs \vee ys = xs \ominus \langle inth ys ((ilen xs) + 1) \rangle$   
**using** 5 7 9  
**by** *simp*  
**qed**

**lemma** *opfx-ilen-le*:

**assumes** *opfx xs ys*  
**shows**  $ilen xs \leq ilen ys$   
**using** *assms* **by** (*auto simp add: opfx-def*)

**lemma** *opfx-same-cases*:

**assumes** *opfx xs1 ys*

```

      opfx xs2 ys
shows   opfx xs1 xs2  $\vee$  opfx xs2 xs1
using  assms unfolding opfx-def using iapp-eq-iapp-conv2
by     metis

lemma opfx-ilen-opfx:
  assumes opfx ps xs
          opfx qs xs
          ilen ps  $\leq$  ilen qs
  shows   opfx ps qs
using    assms
by (auto simp: opfx-def)
   (metis assms(2) dual-order.antisym iapp-eq-iapp-conv opfx-iapp opfx-ilen-le)

lemma iset-mono-opfx:
  assumes opfx xs ys
  shows   iset xs  $\subseteq$  iset ys
using    assms by (auto simp: opfx-def)

lemma prefix-is-opfx:
  opfx (prefix n xs) xs
by (metis Suc-eq-plus1 add-diff-inverse-nat iapp-prefix-suffix
    prefix-ilen-gr-1 le-simps(2) less-Suc-eq-0-disj not-less opfx-def)

lemma imap-mono-opfx:
  assumes opfx xs ys
  shows   opfx (imap f xs) (imap f ys)
using    assms by (auto simp: opfx-def)

lemma opfx-ilen-less:
  assumes sopfx xs ys
  shows   ilen xs < ilen ys
using    assms by (auto simp: sopfx-def opfx-def)

lemma opfx-snocD:
  assumes opfx (xs  $\ominus$   $\langle x \rangle$ ) ys
  shows   sopfx xs ys
using    assms
using    opfx-order.less-le-trans sopfxI' by blast

lemma sopfx-simps [simp, code]:
  sopfx xs  $\langle y \rangle \longleftrightarrow$  False
  sopfx  $\langle x \rangle$  (x  $\odot$  xs)  $\longleftrightarrow$  True
  sopfx (x  $\odot$  xs) (y  $\odot$  ys)  $\longleftrightarrow$  x = y  $\wedge$  sopfx xs ys
proof (simp-all add: sopfx-def)
  show opfx xs  $\langle y \rangle \longrightarrow$  xs =  $\langle y \rangle$ 
    by (metis iapp-not-INil opfxE)
  show opfx  $\langle x \rangle$  (x  $\odot$  xs)
    by (simp add: opfx-ICons)
  show (x = y  $\wedge$  opfx xs ys  $\wedge$  (x = y  $\longrightarrow$  xs  $\neq$  ys)) = (x = y  $\wedge$  opfx xs ys  $\wedge$  xs  $\neq$  ys)

```

by *auto*  
qed

lemma *prefix-sopfx*:  
 assumes *sopfx xs ys*  
 shows *sopfx (prefix n xs) ys*  
 using *assms*  
 proof (induct *n* arbitrary: *xs ys*)  
 case 0  
 then show ?case  
 proof (cases *ys*)  
 case (INil *x1*)  
 then show ?thesis  
 using 0.prem1 by *auto*  
 next  
 case (ICons *x21 x22*)  
 then show ?thesis  
 using 0.prem1 *opfx-order.le-less-trans prefix-is-opfx* by *blast*  
 qed  
 next  
 case (Suc *n*)  
 then show ?case  
 using *opfx-order.order.strict-trans1 prefix-is-opfx* by *blast*  
 qed

lemma *osfxI* [*intro?*]:  
 assumes  $ys = zs \ominus xs \vee ys = xs$   
 shows *osfx xs ys*  
 using *assms* **unfolding** *osfx-def* by *blast*

lemma *osfxE* [*elim?*]:  
 assumes *osfx xs ys*  
 obtains *zs* **where**  $ys = zs \ominus xs \vee ys = xs$   
 using *assms* **unfolding** *osfx-def* by *blast*

lemma *osfx-tl* [*simp*]:  
 assumes  $ilen\ xs > 0$   
 shows *osfx (suffix 1 xs) xs*  
 using *assms*  
 proof (induct *xs*)  
 case (INil *x*)  
 then show ?case by *simp*  
 next  
 case (ICons *x1a xs*)  
 then show ?case by (auto *simp: osfx-def*) (*metis iapp.simps(1)*)  
 qed

lemma *osfx-suffix* [*simp*]:  
 shows *osfx (suffix i xs) xs*

**proof** –  
**have** 1:  $i > \text{ilen } xs \implies \text{osfx } (\text{suffix } i \text{ } xs) \text{ } xs$   
**by** (metis ilast-iapp ilen-snoc-1 neq0-conv osfxI suffix-gr suffix-ilen-last suffix-zero)  
**have** 2:  $i \leq \text{ilen } xs \wedge i = 0 \implies \text{osfx } (\text{suffix } i \text{ } xs) \text{ } xs$   
**by** simp  
**have** 3:  $i \leq \text{ilen } xs \wedge 0 < i \implies \text{osfx } (\text{suffix } i \text{ } xs) \text{ } xs$   
**unfolding** osfx-def **using** iapp-prefix-suffix[of i-1 xs]  
**by** (metis One-nat-def Suc-eq-plus1 Suc-pred less-le-trans)  
**show** ?thesis  
**using** 1 2 3 leI **by** blast  
**qed**

**lemma** osfx-prefix [simp]:  
**assumes** osfx xs ys  
**shows**  $ys = (\text{prefix } (\text{ilen } ys - \text{ilen } xs - 1) \text{ } ys) \ominus xs \quad \vee \quad ys = xs$   
**using** assms  
**by** (auto simp add: osfx-def)  
(metis diff-zero prefix-iapp prefix-ilen)

**lemma** osfx-snoc [simp]:  
 $\text{osfx } xs \text{ } (ys \ominus \langle y \rangle) \iff xs = \langle y \rangle \vee (\exists zs. xs = zs \ominus \langle y \rangle \wedge \text{osfx } zs \text{ } ys)$   
**proof** (cases xs)  
**case** (INil x1)  
**then show** ?thesis **by** (metis ilast-iapp ilast-iapp2 osfxI osfx-prefix)  
**next**  
**case** (ICons x21 x22)  
**then show** ?thesis  
**proof** auto  
**show**  $xs = x21 \odot x22 \implies \text{osfx } (x21 \odot x22) \text{ } (ys \ominus \langle y \rangle) \implies \exists zs. x21 \odot x22 = zs \ominus \langle y \rangle \wedge \text{osfx } zs \text{ } ys$   
**using** iapp-eq-iapp-conv2 iapp-not-INil **unfolding** osfx-def  
**by** (metis interval.distinct(1) )  
**show**  $\bigwedge zs. xs = zs \ominus \langle y \rangle \implies x21 \odot x22 = zs \ominus \langle y \rangle \implies \text{osfx } zs \text{ } ys \implies \text{osfx } (zs \ominus \langle y \rangle) \text{ } (ys \ominus \langle y \rangle)$   
**unfolding** osfx-def **by** (metis iapp-assoc )  
**qed**  
**qed**

**lemma** snoc-osfx-snoc [simp]:  
 $\text{osfx } (xs \ominus \langle x \rangle) \text{ } (ys \ominus \langle y \rangle) = (x = y \wedge \text{osfx } xs \text{ } ys)$   
**by** (simp add: osfx-def)  
(metis iapp-assoc iapp-eq-conv)

**lemma** same-osfx-osfx [simp]:  
 $\text{osfx } (ys \ominus xs) \text{ } (zs \ominus xs) = \text{osfx } ys \text{ } zs$   
**unfolding** osfx-def  
**by** (metis iapp-assoc iapp-same-eq)

**lemma** same-suffix-INil [simp]:  
 $\text{osfx } (ys \ominus xs) \text{ } (x \odot xs) = (ys = \langle x \rangle)$

**unfolding** *osfx-def*  
**by** (*metis iapp-INil iapp-assoc iapp-not-INil iapp-same-eq*)

**lemma** *osfx-ICons*:  
 $osfx\ xs\ (y \odot ys) \longleftrightarrow xs = y \odot ys \vee osfx\ xs\ ys$   
**unfolding** *osfx-def*  
**by** (*auto simp: ICons-eq-iapp-conv*)

**lemma** *osfx-iapp*:  
 $osfx\ xs\ (ys \ominus zs) \longleftrightarrow$   
 $osfx\ xs\ zs \vee (\exists xs'. xs = xs' \ominus zs \wedge osfx\ xs'\ ys)$   
**by** (*auto simp: osfx-def iapp-eq-iapp-conv2*)

**lemma** *osfx-ilen*:  
**assumes** *osfx xs ys*  
**shows**  $ilen\ xs \leq ilen\ ys$   
**using** *assms* **by** (*auto simp add: osfx-def*)

**lemma** *osfx-same-cases*:  
**assumes** *osfx xs<sub>1</sub> ys*  
 $osfx\ xs_2\ ys$   
**shows**  $osfx\ xs_1\ xs_2 \vee osfx\ xs_2\ xs_1$   
**using** *assms* **unfolding** *osfx-def* **by** (*metis iapp-eq-iapp-conv2*)

**lemma** *osfx-ilen-osfx*:  
**assumes** *osfx ps xs*  
 $osfx\ qs\ xs$   
 $ilen\ ps \leq ilen\ qs$   
**shows**  $osfx\ ps\ qs$   
**using** *assms*  
**by** (*auto simp: osfx-def iapp-eq-iapp-conv2*)

**lemma** *osfx-ilen-less*:  
**assumes** *sosfx xs ys*  
**shows**  $ilen\ xs < ilen\ ys$   
**using** *assms* **by** (*auto simp: sosfx-def osfx-def*)

**lemma** *osfx-IConsD'*:  
**assumes** *osfx (x ⊙ xs) ys*  
**shows** *sosfx xs ys*  
**using** *assms*  
**by** (*metis One-nat-def add-diff-cancel-right' diff-is-0-eq ilen.simps(2) not-one-le-zero osfx-ICons*  
 $osfx-ilen\ osfx-order.dual-order.refl\ osfx-order.dual-order.trans\ plus-1-eq-Suc\ sosfx-def$ )

**lemma** *suffix-sosfx*:  
**assumes** *sosfx xs ys*  
**shows**  $sosfx\ (suffix\ n\ xs)\ ys$   
**using** *assms*  
**proof** (*induct n arbitrary: xs ys*)



```

case 0
then show ?case by simp
next
case (Suc n)
then show ?case
  proof (cases xs)
  case (INil x1)
  then show ?thesis using Suc.prems by auto
  next
  case (ICons x21 x22)
  then show ?thesis
  using Suc.hyps Suc.prems osfx-IConsD' osfx-order.less-imp-le by fastforce
  qed
qed

```

```

lemma osfx-tl [simp]:
  assumes ilen xs > 0
  shows osfx (suffix 1 xs) xs
using assms
proof (induct xs)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case using osfx-IConsD' by force
qed

```

```

lemma INil-osfx [simp]:
  osfx ⟨ilast xs⟩ xs
by (metis osfx-suffix suffix-ilen-last)

```

```

lemma osfx-INil [simp]:
  (osfx xs ⟨ilast xs⟩) = (xs = ⟨ilast xs⟩)
using osfx-order.antisym by fastforce

```

```

lemma osfx-IConsI:
assumes osfx xs ys
shows osfx xs (x ⊙ ys)
using assms
by (simp add: osfx-ICons)

```

```

lemma osfx-IConsD:
assumes osfx (x ⊙ xs) ys
shows osfx xs ys
using assms
by (meson osfx-IConsD' osfx-order.less-imp-le)

```

```

lemma osfx-iappI:

```

```

assumes osfx xs ys
shows osfx xs (zs  $\ominus$  ys)
using assms by (metis iapp-assoc osfx-def)

lemma osfx-iappD:
assumes osfx (zs  $\ominus$  xs) ys
shows osfx xs ys
using assms osfxI osfx-order.dual-order.trans by blast

lemma sosfx-iset-subset:
assumes sosfx xs ys
shows iset xs  $\subseteq$  iset ys
using assms by (auto simp: sosfx-def osfx-def)

lemma iset-mono-osfx:
assumes osfx xs ys
shows iset xs  $\subseteq$  iset ys
using assms by (auto simp: osfx-def)

lemma osfx-IConsD2:
assumes osfx (x  $\odot$  xs) (y  $\odot$  ys)
shows osfx xs ys
using assms
proof –
  assume osfx (x  $\odot$  xs) (y  $\odot$  ys)
  then obtain zs where y  $\odot$  ys = zs  $\ominus$  (x  $\odot$  xs)  $\vee$  y  $\odot$  ys = x  $\odot$  xs
    using osfxE by blast
  then show ?thesis
    by (metis assms interval.inject(2) osfx-IConsD osfx-ICons)
qed

lemma osfx-to-opfx [code]:
  osfx xs ys  $\longleftrightarrow$  opfx (irev xs) (irev ys)
unfolding opfx-def
by (metis irev-iapp irev-irev-ident osfx-def)

lemma sosfx-to-sopfx [code]:
  sosfx xs ys  $\longleftrightarrow$  sopfx (irev xs) (irev ys)
by (auto simp: osfx-to-opfx sosfx-def sopfx-def)

lemma imap-mono-osfx:
assumes osfx xs ys
shows osfx (imap f xs) (imap f ys)
using assms by (auto elim!: osfxE intro: osfxI)

lemma prefix-subset:
assumes k  $\leq$  len xs
shows iset (prefix k xs)  $\subseteq$  iset xs
using assms by (simp add: prefix-is-opfx iset-mono-opfx)

```

**lemma** *suffix-subset*:  
**assumes**  $k \leq \text{ilen } xs$   
**shows**  $\text{iset } (\text{suffix } k \text{ } xs) \leq \text{iset } xs$   
**using** *assms* **by** (*simp add: iset-mono-osfx*)

### 13.2.2 idistinct and iremdups

**lemma** *idistinct-iapp* [*simp*]:  
 $\text{idistinct } (xs \ominus ys) = (\text{idistinct } xs \wedge \text{idistinct } ys \wedge \text{iset } xs \cap \text{iset } ys = \{\})$   
**by** (*induct xs*) *auto*

**lemma** *idistinct-osfx*:  
**assumes** *idistinct ys*  
 $\text{osfx } xs \text{ } ys$   
**shows** *idistinct xs*  
**using** *assms*  
**proof** (*clarsimp elim!: osfxE*)  
**show**  $\bigwedge zs. \text{idistinct } ys \implies ys = zs \ominus xs \vee ys = xs \implies \text{idistinct } xs$   
**using** *idistinct-iapp* **by** *blast*  
**qed**

**lemma** *idistinct-tl*:  
**assumes** *idistinct xs*  
 $\text{ilen } xs > 0$   
**shows** *idistinct (suffix 1 xs)*  
**using** *assms*  
**by** (*cases xs*) *auto*

**lemma** *idistinct-irev* [*simp*]:  
 $\text{idistinct } (\text{irev } xs) = \text{idistinct } xs$   
**by** (*induct xs*) *auto*

**lemma** *iset-iremdups* [*simp*]:  
 $\text{iset } (\text{iremdups } xs) = \text{iset } xs$   
**by** (*induct xs*) *auto*

**lemma** *idistinct-iremdups* [*iff*]:  
 $\text{idistinct } (\text{iremdups } xs)$   
**by** (*induct xs*) *auto*

**lemma** *idistinct-iremdups-id*:  
**assumes** *idistinct xs*  
**shows**  $\text{iremdups } xs = xs$   
**using** *assms*  
**by** (*induct xs*) *auto*

**lemma** *iremdups-id-iff-idistinct* [*simp*]:  
 $\text{iremdups } xs = xs \longleftrightarrow \text{idistinct } xs$   
**by** (*metis idistinct-iremdups idistinct-iremdups-id*)

```

lemma idistinct-imap:
  idistinct (imap f xs) = (idistinct xs  $\wedge$  inj-on f (iset xs))
by (induct xs) auto

```

```

lemma idistinct-prefix [simp]:
assumes idistinct xs
shows idistinct (prefix i xs)
using assms
proof (induct xs arbitrary: i)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
  proof (cases i)
  case 0
  then show ?thesis by auto
  next
  case (Suc nat)
  then show ?thesis
  using ICons.hyps ICons.prems prefix-is-opfx iset-mono-opfx by fastforce
  qed
qed

```

```

lemma idistinct-suffix [simp]:
assumes idistinct xs
shows idistinct (suffix i xs)
using assms
proof
  (induct xs arbitrary: i)
  case (INil x)
  then show ?case by simp
  next
  case (ICons x1a xs)
  then show ?case
    proof (cases i)
    case 0
    then show ?thesis using ICons.prems by auto
    next
    case (Suc nat)
    then show ?thesis using iapp-prefix-suffix ICons.hyps ICons.prems by auto
    qed
  qed

```

```

lemma idistinct-conv-inth:
  idistinct xs =
    ( $\forall$  i  $\leq$  ilen xs. ( $\forall$  j  $\leq$  ilen xs. i  $\neq$  j  $\longrightarrow$  inth xs i  $\neq$  inth xs j))
proof

```

```

(induction xs)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
  by (auto simp add: ICons inth-ICons inth-and-iset split: nat.split-asm)
blast+
qed

```

```

lemma idistinct-Ex1:
assumes idistinct xs
       $x \in \text{iset } xs$ 
shows  $(\exists !i. i \leq \text{ilen } xs \wedge (\text{inth } xs \ i) = x)$ 
using assms
by (metis idistinct-conv-inth inth-and-iset)

```

```

lemma inj-on-inth:
assumes idistinct xs
shows  $(\forall i \in I. i \leq \text{ilen } xs \implies \text{inj-on } (\text{inth } xs) \ I)$ 
using assms
by (meson idistinct-conv-inth inj-onI)

```

```

lemma bij-betw-inth:
assumes idistinct xs
       $A = \{..< \text{ilen } xs+1\}$ 
       $B = \text{iset } xs$ 
shows  $\text{bij-betw } ((\text{inth}) \ xs) \ A \ B$ 
using assms unfolding bij-betw-def
proof (auto intro!: inj-on-inth simp: )
  show  $\bigwedge xa. \text{idistinct } xs \implies$ 
     $A = \{..< \text{Suc } (\text{ilen } xs)\} \implies$ 
     $B = \text{iset } xs \implies xa < \text{Suc } (\text{ilen } xs) \implies \text{inth } xs \ xa \in \text{iset } xs$ 
  by (meson inth-and-iset less-Suc-eq-le)
  show  $\bigwedge x. \text{idistinct } xs \implies$ 
     $A = \{..< \text{Suc } (\text{ilen } xs)\} \implies$ 
     $B = \text{iset } xs \implies x \in \text{iset } xs \implies x \in \text{inth } xs \text{ ' } \{..< \text{Suc } (\text{ilen } xs)\}$ 
  using inth-and-iset by (metis image-iff lessThan-iff less-Suc-eq-le)
qed

```

```

lemma card-idistinct:
assumes  $\text{card } (\text{iset } xs) = \text{ilen } xs + 1$ 
shows  $\text{idistinct } xs$ 
using assms
proof
  (induct xs)
  case (INil x)
  then show ?case by simp
next

```

```

case (ICons x1a xs)
then show ?case
  proof (cases x1a  $\in$  iset xs)
    case True
    then show ?thesis
    by (metis ICons.prems iremdups.simps(2) iset-iremdups add-Suc card-ilen ilen.simps(2)
      not-less-eq-eq plus-1-eq-Suc)
    next
    case False
    then show ?thesis using ICons.hyps ICons.prems by auto
  qed
qed

```

```

lemma finite-interval:
  assumes finite A
  shows ( $A \neq \{\}$   $\longrightarrow$  ( $\exists$  xs. iset xs = A))
using assms
proof (induct rule:finite-induct)
case empty
then show ?case by simp
next
case (insert x F)
then show ?case by (metis insert-is-Un interval.set(1) iset-iapp)
qed

```

```

lemma finite-idistinct-interval:
assumes finite A
   $A \neq \{\}$ 
shows ( $\exists$  xs. iset xs = A  $\wedge$  idistinct xs)
using assms by (metis idistinct-iremdups finite-interval iset-iremdups)

```

```

lemma iremdups-eq-INil-iff [simp]:
  (iremdups xs =  $\langle x \rangle$ ) = ( $\forall$  i  $\leq$  ilen xs. (inth xs i) = x)
proof
  (induct xs)
  case (INil x)
  then show ?case by auto
  next
  case (ICons x1a xs)
  then show ?case
    proof –
      have 1: ( $\forall$  i  $\leq$  ilen (x1a  $\odot$  xs). inth (x1a  $\odot$  xs) i = x) =
        (x1a = x  $\wedge$  ( $\forall$  i  $\leq$  ilen (xs). inth (xs) i = x))
        by auto
        (metis Nitpick.case-nat-unfold Suc-eq-plus1 le-diff-conv)
      have 2: (iremdups (x1a  $\odot$  xs) =  $\langle x \rangle$ ) =
        ((if x1a  $\in$  iset xs then iremdups xs else x1a  $\odot$  iremdups xs) =  $\langle x \rangle$ )
        by simp
      have 3: ((if x1a  $\in$  iset xs then iremdups xs else x1a  $\odot$  iremdups xs) =  $\langle x \rangle$ ) =

```

```

      (iremdups xs = ⟨x⟩ ∧ x1a ∈ iset xs)
    by auto
  have 4: (iremdups xs = ⟨x⟩ ∧ x1a ∈ iset xs) =
    ((∀ i ≤ ilen xs). inth (xs) i = x) ∧ x1a ∈ iset xs
    using ICons.hyps by blast
  have 5: (∀ i ≤ ilen xs). inth (xs) i = x ∧ x1a ∈ iset xs ⟶ x1a = x
    by (metis inth-and-iset)
  show ?thesis
  by (metis 1 2 3 5 ICons.hyps iset-iremdups interval.set-intros(1))
qed
qed

```

**lemma** *iremdups-eq-INil-right-iff* [simp]:  
 $(\langle x \rangle = \text{iremdups } xs) = (\forall i \leq \text{ilen } xs. (\text{inth } xs \ i) = x)$   
 by (metis iremdups-eq-INil-iff)

**lemma** *ilen-remdups-leq* [iff]:  
 $\text{ilen}(\text{iremdups } xs) \leq \text{ilen } xs$   
 by (induct xs) auto

**lemma** *ilen-iremdups-eq*[iff]:  
 $(\text{ilen}(\text{iremdups } xs) = \text{ilen } xs) = (\text{iremdups } xs = xs)$   
**proof**  
 (induct xs)  
 case (INil x)  
 then show ?case by simp  
 next  
 case (ICons x1a xs)  
 then show ?case  
 by (metis ilen-remdups-leq iremdups.simps(2) add-left-cancel ilen.simps(2) not-less-eq-eq  
 plus-1-eq-Suc)  
 qed

**lemma** *idistinct-card*:  
**assumes** *idistinct xs*  
**shows**  $\text{card}(\text{iset } xs) = \text{Suc}(\text{ilen } xs)$   
**using** *assms*  
**by** (induct xs) *simp-all*

### 13.2.3 prefixes and suffixes

**lemma** *in-iset-prefixes* [simp]:  
 $xs \in \text{iset}(\text{prefixes } ys) \longleftrightarrow \text{opfx } xs \ ys$   
**proof**  
 (induct xs arbitrary: ys)  
 case (INil x)  
 then show ?case  
**proof** (cases ys)  
 case (INil x1)  
 then show ?thesis using opfxE by force

```

next
case (ICons x21 x22)
then show ?thesis unfolding opfx-def by auto
qed
next
case (ICons x1a xs)
then show ?case
  proof (cases ys)
  case (INil x1)
  then show ?thesis by (simp add: opfx-code(2))
  next
  case (ICons x21 x22)
  then show ?thesis using ICons.hyps by auto
  qed
qed

```

```

lemma ilen-prefixes [simp]:
  ilen (prefixes xs) = ilen xs
by (induction xs) auto

```

```

lemma idistinct-prefixes [intro]:
  idistinct (prefixes xs)
proof (induction xs)
case (INil x)
then show ?case by (auto simp: idistinct-imap)
next
case (ICons x1a xs)
then show ?case by (auto simp: idistinct-imap
  (meson inj-onI interval.inject(2)))
qed

```

```

lemma prefixes-snoc [simp]:
  prefixes (xs ⊖ ⟨x⟩) = (prefixes xs) ⊖ ⟨xs ⊖ ⟨x⟩⟩
by (induction xs) auto

```

```

lemma ifirst-prefixes [simp]:
  ifirst (prefixes xs) = ⟨ifirst xs⟩
by (cases xs) auto

```

```

lemma ilast-prefixes [simp]:
  ilast (prefixes xs) = xs
by (induction xs)
  (simp-all add: ilast-imap inth-imap)

```

```

lemma prefixes-iapp:
  prefixes (xs ⊖ ys) =
    prefixes xs ⊖ imap (λys'. xs ⊖ ys') (prefixes ys)
proof

```



```

(induction xs arbitrary: ys)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
  proof –
    have 1:  $\text{prefixes } ((x1a \odot xs) \ominus ys) = \langle x1a \rangle \odot (\text{imap } ((\odot) x1a) (\text{prefixes } (xs \ominus ys)))$ 
      by simp
    have 2:  $\text{prefixes } (xs \ominus ys) = \text{prefixes } xs \ominus \text{imap } ((\ominus) xs) (\text{prefixes } ys)$ 
      by (simp add: ICons.IH)
    have 3:  $\langle x1a \rangle \odot (\text{imap } ((\odot) x1a) (\text{prefixes } (xs \ominus ys))) =$ 
       $\langle x1a \rangle \odot (\text{imap } ((\odot) x1a) (\text{prefixes } xs \ominus \text{imap } ((\ominus) xs) (\text{prefixes } ys)))$ 

    using 2 by auto
    show ?thesis by (simp add: 3)
  qed
qed

```

```

lemma prefixes-eq-snoc:
   $\text{prefixes } ys = xs \ominus \langle x \rangle \longleftrightarrow$ 
   $(\exists z \text{ zs. } ys = zs \ominus \langle z \rangle \wedge xs = \text{prefixes } zs) \wedge x = ys$ 
proof (cases ys rule: irev-cases)
case (INil x)
then show ?thesis using iapp-not-INil by (metis prefixes.simps(1))
next
case (snoc ys y)
then show ?thesis by auto
qed

```

```

lemma iset-prefixes-eq:
   $\text{iset } (\text{prefixes } xs) = \{ys. \text{opfx } ys \text{ } xs\}$ 
by auto

```

```

lemma card-iset-prefixes [simp]:
   $\text{card } (\text{iset } (\text{prefixes } xs)) = \text{Suc } (\text{ilen } xs)$ 
by (simp add: idistinct-card idistinct-prefixes)

```

```

lemma iset-prefixes-append:
   $\text{iset } (\text{prefixes } (xs \ominus ys)) = \text{iset } (\text{prefixes } xs) \cup \{xs \ominus ys' \mid ys'. ys' \in \text{iset } (\text{prefixes } ys)\}$ 
by (subst prefixes-iapp) auto

```

```

lemma in-iset-suffixes [simp]:
   $xs \in \text{iset}(\text{suffixes } ys) \longleftrightarrow \text{osfx } xs \text{ } ys$ 
proof (induct ys)
case (INil x)
then show ?case using osfx-prefix by fastforce
next
case (ICons x1a ys)

```

**then show** *?case* **by** (*simp add: osfx-ICons*)  
**qed**

**lemma** *interval-sfx-INil*:  
 $iset(suffixes \langle x \rangle) = \{\langle x \rangle\}$   
**by** *simp*

**lemma** *interval-sfx-ICons*:  
 $iset(suffixes (x \odot xs)) = \{x \odot xs\} \cup iset(suffixes xs)$   
**by** *auto*

**lemma** *iset-suffixes-sfx*:  
 $iset(suffixes xs) = \{suffix\ i\ xs \mid i. i \leq len\ xs\}$

**proof**

(*induction xs*)

**case** (*INil x*)

**then show** *?case* **by** *auto*

**next**

**case** (*ICons x1a xs*)

**then show** *?case*

**proof** –

**have** 1:  $\{suffix\ i\ (x1a \odot xs) \mid i. i \leq len\ (x1a \odot xs)\} =$   
 $\{suffix\ i\ (x1a \odot xs) \mid i. i = 0\} \cup \{suffix\ i\ (x1a \odot xs) \mid i. 0 < i \wedge i \leq len\ (x1a \odot xs)\}$

**using** *neq0-conv* **by** (*auto, fastforce, force*)

**have** 2:  $\{suffix\ i\ (x1a \odot xs) \mid i. 0 < i \wedge i \leq len\ (x1a \odot xs)\} =$   
 $\{suffix\ i\ (xs) \mid i. i \leq len\ (xs)\}$

**by** (*auto,*

*metis Nitpick.case-nat-unfold Suc-eq-plus1 le-diff-conv neq0-conv,*

*metis Nitpick.case-nat-unfold Suc-le-mono add-diff-cancel-left' gr-implies-not-zero*

*less-Suc0 not-less-eq plus-1-eq-Suc*)

**show** *?thesis* **using** 1 2 *ICons.IH* **by** *auto*

**qed**

**qed**

**lemma** *iset-suffixes-osfx*:  
 $iset(suffixes xs) = \{ys. osfx\ ys\ xs\}$   
**by** *auto*

**lemma** *idistinct-suffixes [intro]*:  
 $idistinct(suffixes xs)$

**proof** (*induct xs*)

**case** (*INil x*)

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

**next**

**case** (*ICons x1a xs*)

**then show** *?case*

**by** *auto (metis Suc-n-not-n len.simps(2) osfx-IConsD osfx-ICons osfx-order.antisym plus-1-eq-Suc)*

**qed**

**lemma** *ilen-suffixes* [simp]:

$ilen\ (suffixes\ xs) = ilen\ xs$

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

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

$suffixes\ (xs \ominus \langle x \rangle) = (imap\ (\lambda\ ys.\ ys \ominus \langle x \rangle)\ (suffixes\ xs)) \ominus \langle \langle x \rangle \rangle$

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

**lemma** *ifirst-suffixes* [simp]:

$ifirst\ (suffixes\ xs) = xs$

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

**lemma** *ilast-suffixes* [simp]:

$ilast\ (suffixes\ xs) = \langle ilast\ xs \rangle$

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

**lemma** *suffixes-iapp*:

$suffixes\ (xs \ominus ys) = imap\ (\lambda\ xs'.\ xs' \ominus ys)\ (suffixes\ xs) \ominus (suffixes\ ys)$

**proof**

(*induction xs arbitrary: ys*)

**case** (*INil x*)

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

**next**

**case** (*ICons x1a ys*)

**then show** ?*case*

**proof** (*cases ys*)

**case** (*INil x1*)

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

**next**

**case** (*ICons x21 x22*)

**then show** ?*thesis* **by** (*auto simp add: ICons.IH*)

**qed**

**qed**

**lemma** *card-iset-suffixes* [simp]:

$card\ (iset\ (suffixes\ xs)) = Suc\ (ilen\ xs)$

**by** (*simp add: idistinct-card idistinct-suffixes*)

**lemma** *iset-suffixes-iapp*:

$iset\ (suffixes\ (xs \ominus ys)) = \{xs' \ominus ys \mid xs'.\ xs' \in iset\ (suffixes\ xs)\} \cup iset\ (suffixes\ ys)$

**proof** (*subst suffixes-iapp*)

**show**  $iset\ (imap\ (\lambda xs'.\ xs' \ominus ys)\ (suffixes\ xs) \ominus suffixes\ ys) =$

$\{xs' \ominus ys \mid xs'.\ xs' \in iset\ (suffixes\ xs)\} \cup iset\ (suffixes\ ys)$

**proof** (*cases xs*)

**case** (*INil x1*)

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

**next**

**case** (*ICons x21 x22*)

**then show** ?*thesis* **by** *force*

qed  
qed

**lemma** *imap-first-suffixes* [simp]:  
 $\text{imap } (\lambda xs. \text{inth } xs \ 0) (\text{suffixes } xs) = xs$   
**by** (*induct xs*) *auto*

**lemma** *suffixes-conv-prefixes*:  
 $(\text{suffixes } xs) = \text{irev } (\text{imap } \text{irev } (\text{prefixes } (\text{irev } xs)))$   
**by** (*induction xs*) *auto*

**lemma** *prefixes-conv-suffixes*:  
 $(\text{prefixes } xs) = \text{irev } (\text{imap } \text{irev } (\text{suffixes } (\text{irev } xs)))$   
**by** (*induction xs*) (*auto simp add: irev-imap*)

**lemma** *prefixes-irev*:  
 $\text{prefixes } (\text{irev } xs) = \text{irev } (\text{imap } \text{irev } (\text{suffixes } xs))$   
**by** (*induction xs*) *auto*

**lemma** *suffixes-irev*:  
 $\text{suffixes } (\text{irev } xs) = \text{irev } (\text{imap } \text{irev } (\text{prefixes } xs))$   
**by** (*induction xs*) (*auto simp add: irev-imap*)

**lemma** *inth-suffixes*:  
**assumes**  $i \leq \text{ilen}(\text{suffixes } xs)$   
**shows**  $\text{inth } (\text{suffixes } xs) \ i = (\text{suffix } i \ xs)$   
**using** *assms*  
**proof** (*induct xs arbitrary:i*)  
**case** (*INil x*)  
**then show** ?*case* **by** *simp*  
**next**  
**case** (*ICons x1a xs*)  
**then show** ?*case* **by** (*simp add: Nitpick.case-nat-unfold*)  
**qed**

**lemma** *suffix-suffixes*:  
**assumes**  $i \leq \text{ilen } (\text{suffixes } xs)$   
**shows**  $\text{suffix } i (\text{suffixes } xs) = \text{suffixes } (\text{suffix } i \ xs)$   
**using** *assms*  
**proof** (*induct xs arbitrary:i*)  
**case** (*INil x*)  
**then show** ?*case* **by** *simp*  
**next**  
**case** (*ICons x1a xs*)  
**then show** ?*case* **by** (*simp add: Nitpick.case-nat-unfold*)  
**qed**

### 13.2.4 ifilter and nfilter

**lemma** *sfxfilter-ilen-a:*

**assumes**  $\exists \text{ } ys \in \text{iset } (\text{suffixes } \langle x \rangle). P \text{ } ys$   
**shows**  $\text{ilen } (\text{ifilter } P \text{ } (\text{suffixes } \langle x \rangle)) = 0$   
**using** *assms by simp*

**lemma** *sfxfilter-ilen-b:*

**assumes**  $(\exists \text{ } ys \in \text{iset } (\text{suffixes } (x \odot xs)). P \text{ } ys)$   
 $(\exists \text{ } ys \in \text{iset } (\text{suffixes } xs). P \text{ } ys)$   
 $P \text{ } (x \odot xs)$   
**shows**  $\text{ilen } (\text{ifilter } P \text{ } (\text{suffixes } (x \odot xs))) = \text{ilen}(\text{ifilter } P \text{ } (\text{suffixes } xs)) + 1$   
**using** *assms by simp*

**lemma** *sfxfilter-ilen-c:*

**assumes**  $(\exists \text{ } ys \in \text{iset } (\text{suffixes } (x \odot xs)). P \text{ } ys)$   
 $(\exists \text{ } ys \in \text{iset } (\text{suffixes } xs). P \text{ } ys)$   
 $\neg P \text{ } (x \odot xs)$   
**shows**  $\text{ilen } (\text{ifilter } P \text{ } (\text{suffixes } (x \odot xs))) = \text{ilen}(\text{ifilter } P \text{ } (\text{suffixes } xs))$   
**using** *assms by simp*

**lemma** *sfxfilter-ilen-d:*

**assumes**  $(\exists \text{ } ys \in \text{iset } (\text{suffixes } (x \odot xs)). P \text{ } ys)$   
 $\neg(\exists \text{ } ys \in \text{iset } (\text{suffixes } xs). P \text{ } ys)$   
**shows**  $\text{ilen } (\text{ifilter } P \text{ } (\text{suffixes } (x \odot xs))) = 0$   
**using** *assms by simp*

**lemma** *ifilter-ilen-a:*

**assumes**  $\exists \text{ } ys \in \text{iset } \langle x \rangle. P \text{ } ys$   
**shows**  $\text{ilen } (\text{ifilter } P \text{ } \langle x \rangle) = 0$   
**using** *assms by simp*

**lemma** *nfilter-ilen-a:*

**assumes**  $\exists \text{ } ys \in \text{iset } \langle x \rangle. P \text{ } ys$   
**shows**  $\text{ilen } (\text{nfilter } P \text{ } \langle x \rangle \text{ } n) = 0$   
**using** *assms by simp*

**lemma** *ifilter-ilen-b:*

**assumes**  $(\exists \text{ } ys \in \text{iset } (x \odot xs). P \text{ } ys)$   
 $(\exists \text{ } ys \in \text{iset } (xs). P \text{ } ys)$   
 $P \text{ } (x)$   
**shows**  $\text{ilen } (\text{ifilter } P \text{ } (x \odot xs)) = \text{ilen}(\text{ifilter } P \text{ } (xs)) + 1$   
**using** *assms by simp*

**lemma** *nfilter-ilen-b:*

**assumes**  $(\exists \text{ } ys \in \text{iset } (x \odot xs). P \text{ } ys)$   
 $(\exists \text{ } ys \in \text{iset } (xs). P \text{ } ys)$   
 $P \text{ } (x)$   
**shows**  $\text{ilen } (\text{nfilter } P \text{ } (x \odot xs) \text{ } n) = \text{ilen}(\text{nfilter } P \text{ } (xs) \text{ } (\text{Suc } n)) + 1$   
**using** *assms by simp*

**lemma** *ifilter-ilen-c:*

**assumes**  $(\exists \text{ } ys \in \text{iset } (x \odot xs)). P \text{ } ys$   
           $(\exists \text{ } ys \in \text{iset } (xs). P \text{ } ys)$   
           $\neg P \text{ } (x)$   
**shows**  $\text{ilen } (\text{ifilter } P \text{ } (x \odot xs)) = \text{ilen}(\text{ifilter } P \text{ } (xs))$   
**using** *assms by simp*

**lemma** *nfilter-ilen-c:*

**assumes**  $(\exists \text{ } ys \in \text{iset } (x \odot xs)). P \text{ } ys$   
           $(\exists \text{ } ys \in \text{iset } (xs). P \text{ } ys)$   
           $\neg P \text{ } (x)$   
**shows**  $\text{ilen } (\text{nfilter } P \text{ } (x \odot xs) \text{ } n) = \text{ilen}(\text{nfilter } P \text{ } (xs) \text{ } (\text{Suc } n))$   
**using** *assms by simp*

**lemma** *ifilter-ilen-d:*

**assumes**  $(\exists \text{ } ys \in \text{iset } (x \odot xs)). P \text{ } ys$   
           $\neg(\exists \text{ } ys \in \text{iset } (xs). P \text{ } ys)$   
**shows**  $\text{ilen } (\text{ifilter } P \text{ } (x \odot xs)) = 0$   
**using** *assms by simp*

**lemma** *nfilter-ilen-d:*

**assumes**  $(\exists \text{ } ys \in \text{iset } (x \odot xs)). P \text{ } ys$   
           $\neg(\exists \text{ } ys \in \text{iset } (xs). P \text{ } ys)$   
**shows**  $\text{ilen } (\text{nfilter } P \text{ } (x \odot xs) \text{ } n) = 0$   
**using** *assms by simp*

**lemma** *sfxfilter-inth-a:*

**assumes**  $(\exists \text{ } ys \in \text{iset } (\text{suffixes } \langle x \rangle)). P \text{ } ys$   
           $j \leq \text{ilen}(\text{ifilter } P \text{ } (\text{suffixes } \langle x \rangle))$   
**shows**  $\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } \langle x \rangle)) \text{ } j = \langle x \rangle$   
**using** *assms by simp*

**lemma** *sfxfilter-inth-b1:*

**assumes**  $(\exists \text{ } ys \in \text{iset } (\text{suffixes } (x \odot xs)). P \text{ } ys)$   
           $(\exists \text{ } ys \in \text{iset } (\text{suffixes } xs). P \text{ } ys)$   
           $P \text{ } (x \odot xs)$   
**shows**  $\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } (x \odot xs))) \text{ } 0 = x \odot xs$   
**using** *assms by simp*

**lemma** *sfxfilter-inth-b2:*

**assumes**  $(\exists \text{ } ys \in \text{iset } (\text{suffixes } (x \odot xs)). P \text{ } ys)$   
           $(\exists \text{ } ys \in \text{iset } (\text{suffixes } xs). P \text{ } ys)$   
           $P \text{ } (x \odot xs)$   
           $(\text{Suc } j) \leq \text{ilen } (\text{ifilter } P \text{ } (\text{suffixes } (x \odot xs)))$   
**shows**  $\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } (x \odot xs))) \text{ } (\text{Suc } j) = \text{inth } (\text{ifilter } P \text{ } (\text{suffixes } xs)) \text{ } j$   
**using** *assms by auto*

**lemma** *sfxfilter-inth-c:*

**assumes**  $(\exists \text{ } ys \in \text{iset } (\text{suffixes } (x \odot xs)). P \text{ } ys)$   
           $(\exists \text{ } ys \in \text{iset } (\text{suffixes } xs). P \text{ } ys)$

$\neg P (x \odot xs)$   
 $j \leq \text{ilen} (\text{ifilter } P (\text{suffixes } (x \odot xs)))$   
**shows**  $\text{inth} (\text{ifilter } P (\text{suffixes } (x \odot xs))) j = \text{inth} (\text{ifilter } P (\text{suffixes } (xs))) j$   
**using** *assms* **by** *auto*

**lemma** *sxfilter-inth-d*:  
**assumes**  $(\exists \text{ } ys \in \text{iset} (\text{suffixes } (x \odot xs)). P \text{ } ys)$   
 $\neg(\exists \text{ } ys \in \text{iset} (\text{suffixes } xs). P \text{ } ys)$   
 $j \leq \text{ilen} (\text{ifilter } P (\text{suffixes } (x \odot xs)))$   
**shows**  $\text{inth} (\text{ifilter } P (\text{suffixes } (x \odot xs))) j = x \odot xs$   
**using** *assms* **by** *auto*

**lemma** *nfilter-inth-a*:  
**assumes**  $(\exists \text{ } ys \in \text{iset} (\langle x \rangle). P \text{ } ys)$   
 $j \leq \text{ilen}(\text{nfilter } P (\langle x \rangle) \text{ } n)$   
**shows**  $\text{inth} (\text{nfilter } P (\langle x \rangle) \text{ } n) j = n$   
**using** *assms* **by** *auto*

**lemma** *ifilter-inth-a*:  
**assumes**  $(\exists \text{ } ys \in \text{iset} (\langle x \rangle). P \text{ } ys)$   
 $j \leq \text{ilen}(\text{ifilter } P (\langle x \rangle))$   
**shows**  $\text{inth} (\text{ifilter } P (\langle x \rangle)) j = x$   
**using** *assms* **by** *simp*

**lemma** *nfilter-inth-b1*:  
**assumes**  $(\exists \text{ } ys \in \text{iset} (\langle x \odot xs \rangle). P \text{ } ys)$   
 $(\exists \text{ } ys \in \text{iset} (\langle xs \rangle). P \text{ } ys)$   
 $P (x)$   
**shows**  $\text{inth} (\text{nfilter } P (\langle x \odot xs \rangle) \text{ } n) 0 = n$   
**using** *assms* **by** *simp*

**lemma** *ifilter-inth-b1*:  
**assumes**  $(\exists \text{ } ys \in \text{iset} (\langle x \odot xs \rangle). P \text{ } ys)$   
 $(\exists \text{ } ys \in \text{iset} (\langle xs \rangle). P \text{ } ys)$   
 $P (x)$   
**shows**  $\text{inth} (\text{ifilter } P (\langle x \odot xs \rangle)) 0 = x$   
**using** *assms* **by** *simp*

**lemma** *nfilter-inth-b2*:  
**assumes**  $(\exists \text{ } ys \in \text{iset} (\langle x \odot xs \rangle). P \text{ } ys)$   
 $(\exists \text{ } ys \in \text{iset} (\langle xs \rangle). P \text{ } ys)$   
 $P (x)$   
 $(\text{Suc } j) \leq \text{ilen} (\text{nfilter } P (\langle x \odot xs \rangle) \text{ } n)$   
**shows**  $\text{inth} (\text{nfilter } P (\langle x \odot xs \rangle) \text{ } n) (\text{Suc } j) = \text{inth} (\text{nfilter } P (\langle xs \rangle) (\text{Suc } n)) j$   
**using** *assms* **by** *auto*

**lemma** *ifilter-inth-b2*:  
**assumes**  $(\exists \text{ } ys \in \text{iset} (\langle x \odot xs \rangle). P \text{ } ys)$   
 $(\exists \text{ } ys \in \text{iset} (\langle xs \rangle). P \text{ } ys)$   
 $P (x)$

$(\text{Suc } j) \leq \text{ilen } (\text{ifilter } P \ (x \odot xs))$   
**shows**  $\text{inth } (\text{ifilter } P \ (x \odot xs)) \ (\text{Suc } j) = \text{inth } (\text{ifilter } P \ (xs)) \ j$   
**using** *assms* **by** *auto*

**lemma** *nfilter-inth-c:*

**assumes**  $(\exists \ ys \in \text{iset } (x \odot xs)). \ P \ ys$   
 $(\exists \ ys \in \text{iset } (xs). \ P \ ys)$   
 $\neg \ P \ (x)$   
 $j \leq \text{ilen } (\text{nfilter } P \ (x \odot xs)) \ n$   
**shows**  $\text{inth } (\text{nfilter } P \ (x \odot xs)) \ n \ j = \text{inth } (\text{nfilter } P \ (xs)) \ (\text{Suc } n) \ j$   
**using** *assms* **by** *auto*

**lemma** *ifilter-inth-c:*

**assumes**  $(\exists \ ys \in \text{iset } (x \odot xs)). \ P \ ys$   
 $(\exists \ ys \in \text{iset } (xs). \ P \ ys)$   
 $\neg \ P \ (x)$   
 $j \leq \text{ilen } (\text{ifilter } P \ (x \odot xs))$   
**shows**  $\text{inth } (\text{ifilter } P \ (x \odot xs)) \ j = \text{inth } (\text{ifilter } P \ (xs)) \ j$   
**using** *assms* **by** *auto*

**lemma** *nfilter-inth-d:*

**assumes**  $(\exists \ ys \in \text{iset } (x \odot xs)). \ P \ ys$   
 $\neg(\exists \ ys \in \text{iset } (xs). \ P \ ys)$   
 $j \leq \text{ilen } (\text{nfilter } P \ (x \odot xs)) \ n$   
**shows**  $\text{inth } (\text{nfilter } P \ (x \odot xs)) \ n \ j = n$   
**using** *assms* **by** *auto*

**lemma** *ifilter-inth-d:*

**assumes**  $(\exists \ ys \in \text{iset } (x \odot xs)). \ P \ ys$   
 $\neg(\exists \ ys \in \text{iset } (xs). \ P \ ys)$   
 $j \leq \text{ilen } (\text{ifilter } P \ (x \odot xs))$   
**shows**  $\text{inth } (\text{ifilter } P \ (x \odot xs)) \ j = x$   
**using** *assms* **by** *auto*

**lemma** *sfxfilter-inth-ICons:*

$\text{inth } (\text{ifilter } P \ (\text{suffixes } (x \odot xs))) \ j =$   
 $(\text{if } (\exists \ ys \in \text{iset } (\text{suffixes } xs). \ P \ ys) \ \text{then}$   
 $(\text{if } P \ (x \odot xs) \ \text{then}$   
 $(\text{if } j=0 \ \text{then } (x \odot xs) \ \text{else } \text{inth } (\text{ifilter } P \ (\text{suffixes } xs)) \ (j-1))$   
 $\text{else } \text{inth } (\text{ifilter } P \ (\text{suffixes } xs)) \ j)$   
 $\text{else } (x \odot xs))$   
**by** (*simp add: Nitpick.case-nat-unfold*)

**lemma** *sfxfilter-inth-ICons-a:*

$\text{inth } (\text{ifilter } P \ (\text{suffixes } (x \odot xs))) \ j =$   
 $(\text{if } (\exists \ ys \in \text{iset } (\text{suffixes } xs). \ P \ ys) \ \text{then}$   
 $(\text{if } P \ (x \odot xs) \ \text{then}$   
 $(\text{case } j \ \text{of } 0 \Rightarrow x \odot xs \mid \text{Suc } m \Rightarrow \text{inth } (\text{ifilter } P \ (\text{suffixes } xs)) \ m)$   
 $\text{else } \text{inth } (\text{ifilter } P \ (\text{suffixes } xs)) \ j)$   
 $\text{else } (x \odot xs))$



**by** (*simp add: Nitpick.case-nat-unfold*)

**lemma** *nfilter-inth-ICons:*

*inth (nfilter P ( (x⊙xs)) n) j =*  
*(if (∃ ys ∈ iset ( xs). P ys) then*  
*(if P (x) then*  
*(if j=0 then (n) else inth (nfilter P ( xs) (Suc n)) (j-1))*  
*else inth (nfilter P (xs) (Suc n)) j)*  
*else (n))*

**by** (*simp add: Nitpick.case-nat-unfold*)

**lemma** *nfilter-inth-ICons-a:*

*inth (nfilter P ( (x⊙xs)) n) j =*  
*(if (∃ ys ∈ iset ( xs). P ys) then*  
*(if P (x) then*  
*(case j of 0 ⇒ n | Suc m ⇒ inth (nfilter P ( xs) (Suc n)) m)*  
*else inth (nfilter P (xs) (Suc n)) j)*  
*else (n))*

**by** (*simp add: Nitpick.case-nat-unfold*)

**lemma** *ifilter-inth-ICons:*

*inth (ifilter P ( (x⊙xs))) j =*  
*(if (∃ ys ∈ iset ( xs). P ys) then*  
*(if P (x) then*  
*(if j=0 then (x) else inth (ifilter P ( xs)) (j-1))*  
*else inth (ifilter P (xs)) j)*  
*else (x))*

**by** (*simp add: Nitpick.case-nat-unfold*)

**lemma** *ifilter-inth-ICons-a:*

*inth (ifilter P ( (x⊙xs))) j =*  
*(if (∃ ys ∈ iset ( xs). P ys) then*  
*(if P (x) then*  
*(case j of 0 ⇒ x | (Suc m) ⇒ inth (ifilter P ( xs)) m)*  
*else inth (ifilter P (xs)) j)*  
*else (x))*

**by** (*simp add: Nitpick.case-nat-unfold*)

**lemma** *sfxfilter-inth:*

**assumes** ( $\exists$  *ys* ∈ *iset (suffices xs). P ys*)  
*i ≤ ilen (ifilter P (suffices xs))*  
**shows** *P (inth (ifilter P (suffices xs)) i)*

**using** *assms*

**proof**

(*induction xs arbitrary: i*)

**case** (*INil x*)

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

**next**

**case** (*ICons x1a xs*)

**then show** *?case*

```

proof (cases  $\exists a \in \text{iset } (\text{suffixes } xs). P a$ )
case True
then show ?thesis using ICons by (auto simp add: nat.split-sels(2))
next
case False
then show ?thesis using ICons by simp
qed
qed

```

```

lemma nfilter-ilen:
assumes ( $\exists x \in \text{iset } xs. P x$ )
shows  $\text{ilen}(\text{nfilter } P \text{ } xs \text{ } n) = \text{ilen } (\text{ifilter } P \text{ } xs)$ 
using assms
proof
  (induction xs arbitrary: n)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case by simp
qed

```

```

lemma nfilter-upper-bound:
assumes ( $\exists x \in \text{iset } xs. P x$ )
   $i \leq \text{ilen } (\text{nfilter } P \text{ } xs \text{ } n)$ 
shows  $(\text{inth } (\text{nfilter } P \text{ } xs \text{ } n) \text{ } i) \leq n + \text{ilen } xs$ 
using assms
proof
  (induct xs arbitrary: i n)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
proof (cases  $\exists x \in \text{iset } xs. P x$ )
case True
then show ?thesis
  proof (cases i)
    case 0
    then show ?thesis using ICons True
    by simp-all
    (metis ICons.prem(1) ICons.prem(2) add-Suc nfilter-ilen-c)
  next
  case (Suc nat)
  then show ?thesis using ICons True by simp-all fastforce
  qed
next
case False

```

```

then show ?thesis using Cons by simp
qed
qed

lemma nfilter-lower-bound:
  assumes  $(\exists x \in \text{iset } xs. P x)$ 
            $i \leq \text{ilen } (nfilter P xs n)$ 
  shows  $n \leq (\text{inth } (nfilter P xs n) i)$ 
using assms
proof
  (induction xs arbitrary: i n)
  case (INil x)
  then show ?case by simp
  next
  case (ICons x1a xs)
  then show ?case
  proof (cases  $\exists x \in \text{iset } xs. P x$ )
  case True
  then show ?thesis
  proof (cases i)
  case 0
  then show ?thesis using ICons True Suc-leD by (simp-all, blast)
  next
  case (Suc nat)
  then show ?thesis using ICons True by simp-all fastforce
  qed
  next
  case False
  then show ?thesis using Cons False
  by auto
  qed
qed

```

```

lemma nfilter-ifilter:
  assumes  $(\exists x \in \text{iset } xs. P x)$ 
            $i \leq \text{ilen } (nfilter P xs n)$ 
  shows  $(\text{inth } xs ((\text{inth } (nfilter P xs n) i) - n)) = (\text{inth } (ifilter P xs) i)$ 
using assms
proof
  (induct xs arbitrary: i n)
  case (INil x)
  then show ?case by simp
  next
  case (ICons x1a xs)
  then show ?case
  proof -
  have 1:  $(\exists x \in \text{iset } xs. P x) \wedge P x1a \wedge i = 0 \longrightarrow$ 
            $\text{inth } (x1a \odot xs) (\text{inth } (nfilter P (x1a \odot xs) n) i - n) =$ 
            $\text{inth } (ifilter P (x1a \odot xs)) i$ 
  by auto
  

```

```

have 2:  $\bigwedge j. (\exists x \in \text{iset } xs. P x) \wedge P x1a \wedge j < \text{ilen } (nfilter P (x1a \odot xs) n) \longrightarrow$ 
 $\text{inth } (x1a \odot xs) (\text{inth } (nfilter P (x1a \odot xs) n) (Suc j) - n) =$ 
 $\text{inth } (ifilter P (x1a \odot xs)) (Suc j)$ 
by auto
(metis ICons.hyps inth.simps(2) Suc-diff-Suc inth-Suc less-Suc-eq-le
less-eq-Suc-le nfilter-lower-bound)
have 3:  $(\exists x \in \text{iset } xs. P x) \wedge \neg P x1a \wedge i = 0 \longrightarrow$ 
 $\text{inth } (x1a \odot xs) (\text{inth } (nfilter P (x1a \odot xs) n) i - n) =$ 
 $\text{inth } (ifilter P (x1a \odot xs)) i$ 
using nfilter-lower-bound[of xs P 0 Suc n] ICons
by auto
(metis inth.simps(2) Suc-diff-Suc inth-Suc le0 less-eq-Suc-le)
have 4:  $\bigwedge j. (\exists x \in \text{iset } xs. P x) \wedge \neg P x1a \wedge j < \text{ilen } (nfilter P (x1a \odot xs) n) \longrightarrow$ 
 $\text{inth } (x1a \odot xs) (\text{inth } (nfilter P (x1a \odot xs) n) (Suc j) - n) =$ 
 $\text{inth } (ifilter P (x1a \odot xs)) (Suc j)$ 
by auto
(metis ICons.hyps inth.simps(2) Suc-diff-Suc inth-Suc less-eq-Suc-le
nfilter-lower-bound)
have 5:  $\neg(\exists x \in \text{iset } xs. P x) \longrightarrow \text{inth } (x1a \odot xs) (\text{inth } (nfilter P (x1a \odot xs) n) i - n) =$ 
 $\text{inth } (ifilter P (x1a \odot xs)) i$ 
by simp
from 1 2 3 4 5 show ?thesis
by (metis ICons.prem(2) add-less-cancel-left le-0-eq le-SucE le-imp-less-Suc plus-1-eq-Suc
zero-induct)

```

**qed**  
**qed**

```

lemma iset-ifilter [simp]:
assumes  $(\exists x \in \text{iset } xs. P x)$ 
shows  $\text{iset } (ifilter P xs) = \{x. x \in \text{iset } xs \wedge P x\}$ 
using assms
proof
(induct xs)
case (INil x)
then show ?case by (simp add: Collect-conv-if)
next
case (ICons x1a xs)
then show ?case by auto
qed

```

```

lemma iset-nfilter [simp]:
assumes  $(\exists x \in \text{iset } xs. P x)$ 
shows  $\text{iset } (nfilter P xs n) = \{n+k | k. k \leq \text{ilen } xs \wedge P (\text{inth } xs k)\}$ 
using assms
proof
(induction xs arbitrary: n)
case (INil x)
then show ?case by auto
next
case (ICons x1a xs)

```

```

then show ?case
proof -
  have 1:  $(\exists x \in \text{iset } xs. P x) \wedge P x1a \longrightarrow$ 
     $\text{iset } (\text{nfilter } P (x1a \odot xs) n) =$ 
     $\{n + k \mid k. k \leq \text{ilen } (x1a \odot xs) \wedge P (\text{inth } (x1a \odot xs) k)\}$ 
  using ICons by auto
  (metis (mono-tags, lifting) Nitpick.case-nat-unfold One-nat-def Suc-le-mono Suc-pred
    gr-implies-not0)+
  have 2:  $(\exists x \in \text{iset } xs. P x) \wedge \neg P x1a \longrightarrow$ 
     $\text{iset } (\text{nfilter } P (x1a \odot xs) n) =$ 
     $\{n + k \mid k. k \leq \text{ilen } (x1a \odot xs) \wedge P (\text{inth } (x1a \odot xs) k)\}$ 
  using ICons by auto
  (metis (mono-tags, lifting) Nitpick.case-nat-unfold One-nat-def Suc-eq-plus1 Suc-pred
    le-0-eq le-diff-conv not-le-imp-less)
  have 3:  $\neg(\exists x \in \text{iset } xs. P x) \longrightarrow$ 
     $\text{iset } (\text{nfilter } P (x1a \odot xs) n) =$ 
     $\{n + k \mid k. k \leq \text{ilen } (x1a \odot xs) \wedge P (\text{inth } (x1a \odot xs) k)\}$ 
  using ICons by auto
  (metis Nitpick.case-nat-unfold Suc-eq-plus1 le-diff-conv inth-iset)
  show ?thesis
  using 1 2 3 by linarith
qed
qed

```

```

lemma iset-minus-ifilter-out:
assumes  $(\exists z \in \text{iset } xs. (\lambda x. \neg(x=y)) z)$ 
shows  $\text{iset } xs - \{y\} = \text{iset}(\text{ifilter } (\lambda x. \neg(x=y)) xs)$ 
using assms
by (induct xs) auto

```

```

lemma ifilter-ifilter [simp]:
assumes  $\exists x \in \text{iset } (\text{ifilter } Q xs). P x$ 
   $\exists x \in \text{iset } xs. Q x$ 
   $\exists x \in \text{iset } xs. P x \wedge Q x$ 
shows  $\text{ifilter } P (\text{ifilter } Q xs) = \text{ifilter } (\lambda x. P x \wedge Q x) xs$ 
using assms
by (induct xs) auto

```

```

lemma ilen-nfilter-le [simp]:
   $\text{ilen } (\text{nfilter } P xs n) \leq \text{ilen } xs$ 
by (induct xs arbitrary: n) (auto simp add: le-SucI)

```

```

lemma ilen-ifilter-le [simp]:
   $\text{ilen } (\text{ifilter } P xs) \leq \text{ilen } xs$ 
by (induct xs) (auto simp add: le-SucI)

```

```

lemma sfxfilter-bound:
assumes  $(\exists ys \in \text{iset } (\text{suffixes } xs). P ys)$ 
shows  $\text{ilen } (\text{ifilter } P (\text{suffixes } xs)) \leq \text{ilen } xs$ 
using assms by (metis ilen-ifilter-le ilen-suffixes)

```

**lemma** *ifilter-bound*:

**assumes**  $(\exists \text{ ys} \in \text{iset } (xs). P \text{ ys})$   
**shows**  $\text{ilen } (\text{ifilter } P \text{ } (xs)) \leq \text{ilen } xs$   
**using** *assms by auto*

**lemma** *sfxfilter-inth-bound*:

**assumes**  $(\exists \text{ ys} \in \text{iset } (\text{suffixes } xs). P \text{ ys})$   
 $j \leq \text{ilen } (\text{ifilter } P \text{ } (\text{suffixes } xs))$   
**shows**  $\text{ilen } ((\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } xs)) \text{ } j)) \leq \text{ilen } xs$   
**using** *assms iset-ifilter[of suffixes xs P]*  
**by** *(metis (mono-tags, lifting) in-iset-suffixes mem-Collect-eq inth-iset osfx-ilen)*

**lemma** *sfxfilter-inth-suffix*:

**assumes**  $(\exists \text{ ys} \in \text{iset } (\text{suffixes } xs). P \text{ ys})$   
 $j \leq \text{ilen } (\text{ifilter } P \text{ } (\text{suffixes } xs))$   
**shows**  $\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } xs)) \text{ } j =$   
 $\text{suffix } (\text{ilen } xs - \text{ilen } (\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } xs)) \text{ } j)) \text{ } xs$

**using** *assms*

**proof**

*(induction xs arbitrary: j)*

**case** *(INil x)*

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

**next**

**case** *(ICons x1a xs)*

**then show** *?case*

**proof** –

**have** 1:  $(\exists \text{ ys} \in \text{iset } (\text{suffixes } xs). P \text{ ys}) \wedge P \text{ } (x1a \odot xs) \wedge j=0 \longrightarrow$   
 $\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } (x1a \odot xs))) \text{ } j =$   
 $\text{suffix } (\text{ilen } (x1a \odot xs) - \text{ilen } (\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } (x1a \odot xs))) \text{ } j)) \text{ } (x1a \odot xs)$

**using** *Cons by auto*

**have** 2:  $(\exists \text{ ys} \in \text{iset } (\text{suffixes } xs). P \text{ ys}) \wedge \neg P \text{ } (x1a \odot xs) \wedge j=0 \longrightarrow$   
 $\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } (x1a \odot xs))) \text{ } j =$   
 $\text{suffix } (\text{ilen } (x1a \odot xs) - \text{ilen } (\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } (x1a \odot xs))) \text{ } j)) \text{ } (x1a \odot xs)$

**using** *ICons sfxfilter-inth-bound[of xs P 0] Suc-diff-le by auto*

**have** 3:  $\bigwedge x. (\exists \text{ ys} \in \text{iset } (\text{suffixes } xs). P \text{ ys}) \wedge P \text{ } (x1a \odot xs) \wedge$   
 $x < \text{ilen } (\text{ifilter } P \text{ } (\text{suffixes } (x1a \odot xs))) \longrightarrow$   
 $\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } (x1a \odot xs))) \text{ } (\text{Suc } x) =$   
 $\text{suffix } (\text{ilen } (x1a \odot xs) - \text{ilen } (\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } (x1a \odot xs))) \text{ } (\text{Suc } x))) \text{ } (x1a \odot xs)$

**using** *ICons sfxfilter-inth-bound[of xs P] Suc-diff-le in-iset-suffixes less-Suc-eq-le by auto*

**have** 4:  $\bigwedge x. (\exists \text{ ys} \in \text{iset } (\text{suffixes } xs). P \text{ ys}) \wedge \neg P \text{ } (x1a \odot xs) \wedge$   
 $x < \text{ilen } (\text{ifilter } P \text{ } (\text{suffixes } (x1a \odot xs))) \longrightarrow$   
 $\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } (x1a \odot xs))) \text{ } (\text{Suc } x) =$   
 $\text{suffix } (\text{ilen } (x1a \odot xs) - \text{ilen } (\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } (x1a \odot xs))) \text{ } (\text{Suc } x))) \text{ } (x1a \odot xs)$

**using** *ICons sfxfilter-inth-bound[of xs P] Suc-diff-le by auto*

**have** 5:  $\neg(\exists \text{ ys} \in \text{iset } (\text{suffixes } xs). P \text{ ys}) \longrightarrow$   
 $\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } (x1a \odot xs))) \text{ } j =$   
 $\text{suffix } (\text{ilen } (x1a \odot xs) - \text{ilen } (\text{inth } (\text{ifilter } P \text{ } (\text{suffixes } (x1a \odot xs))) \text{ } j)) \text{ } (x1a \odot xs)$

**using** *Cons by auto*

**show** *?thesis*

by (metis 1 2 3 4 5 ICons.premis(2) Suc-less-SucD le-SucE le-imp-less-Suc le-zero-eq  
 zero-induct)  
 qed  
 qed

**lemma** *initfilter-sfxfilter-exists*:

( $\exists$   $ys \in \text{iset}(\text{suffixes } xs). P(\text{inth } ys \ 0)$ ) = ( $\exists$   $x \in \text{iset } xs. P \ x$ )  
 by (metis inth-and-iset inth-imap ilen-suffixes imap-first-suffixes)

**lemma** *initfilter-sfxfilter*:

**assumes**  $\exists$   $ys \in \text{iset}(\text{suffixes } xs). P(\text{inth } ys \ 0)$   
**shows**  $\text{ifilter } P \ xs = \text{imap}(\lambda s. (\text{inth } s \ 0))(\text{ifilter}(\lambda ys. P(\text{inth } ys \ 0))(\text{suffixes } xs))$   
**using** *assms*  
**proof**  
 (induction  $xs$ )  
**case** (*INil*  $x$ )  
**then show** ?case **by** *simp*  
**next**  
**case** (*ICons*  $x1a \ xs$ )  
**then show** ?case  
**proof** –  
**have** 1: ( $\exists$   $x \in \text{iset } xs. P \ x$ )  $\wedge$   $P \ x1a \longrightarrow$   
 $\text{ifilter } P \ (x1a \odot xs) =$   
 $\text{imap}(\lambda s. \text{inth } s \ 0)(\text{ifilter}(\lambda ys. P(\text{inth } ys \ 0))(\text{suffixes } (x1a \odot xs)))$   
**using** *ICons initfilter-sfxfilter-exists[of xs P]* **by** *auto*  
**have** 2: ( $\exists$   $x \in \text{iset } xs. P \ x$ )  $\wedge$   $\neg P \ x1a \longrightarrow$   
 $\text{ifilter } P \ (x1a \odot xs) =$   
 $\text{imap}(\lambda s. \text{inth } s \ 0)(\text{ifilter}(\lambda ys. P(\text{inth } ys \ 0))(\text{suffixes } (x1a \odot xs)))$   
**using** *ICons initfilter-sfxfilter-exists[of xs P]* **by** *auto*  
**have** 3:  $\neg(\exists$   $x \in \text{iset } xs. P \ x) \longrightarrow$   
 $\text{ifilter } P \ (x1a \odot xs) =$   
 $\text{imap}(\lambda s. \text{inth } s \ 0)(\text{ifilter}(\lambda ys. P(\text{inth } ys \ 0))(\text{suffixes } (x1a \odot xs)))$   
**using** *ICons initfilter-sfxfilter-exists[of xs P]* **by** *auto*  
**show** ?thesis **using** 1 2 3 **by** *blast*  
 qed  
 qed

**lemma** *ifilter-inth*:

**assumes** ( $\exists$   $x \in \text{iset } xs. P \ x$ )  
 $i \leq \text{ilen}(\text{ifilter } P \ xs)$   
**shows** ( $\exists$   $k \leq \text{ilen } xs. \text{inth}(\text{ifilter } P \ xs) \ i = \text{inth } xs \ k$ )  
**using** *assms using iset-ifilter[of xs P]*  
**by** (metis (no-types, lifting) inth-and-iset mem-Collect-eq)

**lemma** *interval-sfx-inth-zero*:

$\text{iset } xs = \{(\text{inth } ys \ 0) \mid ys. ys \in \text{iset}(\text{suffixes } xs)\}$   
**proof**  
 (induct  $xs$ )  
**case** (*INil*  $x$ )

```

then show ?case
by auto
next
case (ICons x1a xs)
then show ?case
  by auto (metis inth-zero)
qed

```

```

lemma interval-sfx-1:
assumes ys ∈ iset(suffixes xs)
shows (inth ys 0) ∈ iset xs
using assms interval-sfx-inth-zero by fastforce

```

```

lemma sum-ilen-ifilter-compl-help:
assumes ∃ x ∈ iset xs. P x
        ∃ x ∈ iset xs. ¬ P x
shows ilen xs > 0
using assms
proof
(induct xs)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
  using ilen-ICons-1 by blast
qed

```

```

lemma ifilter-id-conv:
assumes ∃ x ∈ iset xs. P x
shows (ifilter P xs = xs) = (∀ x ∈ iset xs. P x)
using assms
proof
(induct xs)
case (INil x)
then show ?case by auto
next
case (ICons x1a xs)
then show ?case
  by (auto simp add: interval-iset-nonempty)
  (metis ilen-ifilter-le Suc-n-not-le-n ilen.simps(2) plus-1-eq-Suc)
qed

```

```

lemma sum-ilen-ifilter-compl:
assumes ∃ x ∈ iset xs. P x
        ∃ x ∈ iset xs. ¬ P x
shows ilen(ifilter P xs) + ilen(ifilter (λx. ¬P x) xs) + 1 = ilen xs
using assms
proof
(induct xs)

```



```

case (INil x)
then show ?case by auto
next
case (ICons x1a xs)
then show ?case
  proof –
    have 1:  $(\exists x \in \text{iset } xs. P x) \wedge (\exists x \in \text{iset } xs. \neg P x) \wedge P x1a \longrightarrow$ 
       $\text{ilen} (\text{ifilter } P (x1a \odot xs)) + \text{ilen} (\text{ifilter } (\lambda x. \neg P x) (x1a \odot xs)) + 1 =$ 
       $\text{ilen} (x1a \odot xs)$ 
    using ICons by auto
    have 2:  $(\exists x \in \text{iset } xs. P x) \wedge (\exists x \in \text{iset } xs. \neg P x) \wedge \neg P x1a \longrightarrow$ 
       $\text{ilen} (\text{ifilter } P (x1a \odot xs)) + \text{ilen} (\text{ifilter } (\lambda x. \neg P x) (x1a \odot xs)) + 1 =$ 
       $\text{ilen} (x1a \odot xs)$ 
    using ICons by auto
    have 3:  $\neg(\exists x \in \text{iset } xs. P x) \longrightarrow$ 
       $\text{ilen} (\text{ifilter } P (x1a \odot xs)) + \text{ilen} (\text{ifilter } (\lambda x. \neg P x) (x1a \odot xs)) + 1 =$ 
       $\text{ilen} (x1a \odot xs)$ 
    using ICons by auto
      (metis ifilter-id-conv)
    have 4:  $(\exists x \in \text{iset } xs. P x) \wedge \neg(\exists x \in \text{iset } xs. \neg P x) \wedge P x1a \longrightarrow$ 
       $\text{ilen} (\text{ifilter } P (x1a \odot xs)) + \text{ilen} (\text{ifilter } (\lambda x. \neg P x) (x1a \odot xs)) + 1 =$ 
       $\text{ilen} (x1a \odot xs)$ 
    using ICons by auto
    have 5:  $(\exists x \in \text{iset } xs. P x) \wedge \neg(\exists x \in \text{iset } xs. \neg P x) \wedge \neg P x1a \longrightarrow$ 
       $\text{ilen} (\text{ifilter } P (x1a \odot xs)) + \text{ilen} (\text{ifilter } (\lambda x. \neg P x) (x1a \odot xs)) + 1 =$ 
       $\text{ilen} (x1a \odot xs)$ 
    using ICons by auto
      (metis ifilter-id-conv)
    show ?thesis
    using 1 2 3 4 5 by blast
  qed
qed

```

**lemma** *ifilter-ilen-imp*:

```

assumes  $\exists x \in \text{iset } xs. P x \wedge Q x$ 
shows  $\text{ilen} (\text{ifilter } (\lambda x. P x \wedge Q x) xs) \leq \text{ilen} (\text{ifilter } P xs)$ 
using assms
proof (induct xs)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case by force
qed

```

**lemma** *subset-ifilter*:

```

assumes  $(\exists x \in \text{iset } xs. P x)$ 
shows  $\text{iset} (\text{ifilter } P xs) \leq \text{iset} (\text{ifilter } (\lambda x. P x \vee Q x) xs)$ 
proof –
  have 1:  $(\exists x \in \text{iset } xs. P x \vee Q x)$ 

```

using *assms* by *blast*  
 have 2:  $\text{iset} (\text{ifilter} (\lambda x. P x \vee Q x) xs) = \{x \in \text{iset } xs. P x \vee Q x\}$   
 using *assms* *iset-ifilter*[of *xs* ( $\lambda x. P x \vee Q x$ )] by *blast*  
 have 3:  $\text{iset} (\text{ifilter } P xs) = \{x \in \text{iset } xs. P x\}$   
 using *assms* *iset-ifilter* by *auto*  
 have 4:  $\{x \in \text{iset } xs. P x\} \leq \{x \in \text{iset } xs. P x \vee Q x\}$   
 by *auto*  
 show ?thesis by (simp add: 2 3 4)  
 qed

lemma *iset-ifilter-not*:

assumes  $\exists x \in \text{iset } xs. P x$   
 $\exists x \in \text{iset } xs. \neg (P x)$   
 shows  $\text{iset} (\text{ifilter} (\lambda x. \neg (P x)) xs) = \text{iset } xs - \text{iset} (\text{ifilter } P xs)$   
 using *assms*  
 proof (induct *xs*)  
 case (INil *x*)  
 then show ?case by *auto*  
 next  
 case (ICons *x1a xs*)  
 then show ?case  
 proof -  
 have 1:  $(\exists x \in \text{iset } xs. P x) \wedge (\exists x \in \text{iset } xs. \neg (P x)) \wedge P x1a \longrightarrow$   
 $\text{iset} (\text{ifilter} (\lambda x. \neg P x) (x1a \odot xs)) = \text{iset} (x1a \odot xs) - \text{iset} (\text{ifilter } P (x1a \odot xs))$   
 using *ICons* by *auto*  
 have 2:  $(\exists x \in \text{iset } xs. P x) \wedge (\exists x \in \text{iset } xs. \neg (P x)) \wedge \neg P x1a \longrightarrow$   
 $\text{iset} (\text{ifilter} (\lambda x. \neg P x) (x1a \odot xs)) = \text{iset} (x1a \odot xs) - \text{iset} (\text{ifilter } P (x1a \odot xs))$   
 using *ICons* by *auto*  
 have 3:  $(\exists x \in \text{iset } xs. P x) \wedge \neg(\exists x \in \text{iset } xs. \neg (P x)) \wedge P x1a \longrightarrow$   
 $\text{iset} (\text{ifilter} (\lambda x. \neg P x) (x1a \odot xs)) = \text{iset} (x1a \odot xs) - \text{iset} (\text{ifilter } P (x1a \odot xs))$   
 using *ICons* by *auto*  
 have 4:  $(\exists x \in \text{iset } xs. P x) \wedge \neg(\exists x \in \text{iset } xs. \neg (P x)) \wedge \neg P x1a \longrightarrow$   
 $\text{iset} (\text{ifilter} (\lambda x. \neg P x) (x1a \odot xs)) = \text{iset} (x1a \odot xs) - \text{iset} (\text{ifilter } P (x1a \odot xs))$   
 using *ICons* by *auto*  
 have 5:  $\neg(\exists x \in \text{iset } xs. P x) \longrightarrow$   
 $\text{iset} (\text{ifilter} (\lambda x. \neg P x) (x1a \odot xs)) = \text{iset} (x1a \odot xs) - \text{iset} (\text{ifilter } P (x1a \odot xs))$   
 using *ICons* by *auto*  
 show ?thesis using 1 2 3 4 5 by *linarith*  
 qed  
 qed

lemma *iset-ifilter-cap*:

assumes  $\exists x \in \text{iset } xs. f x$   
 $\exists x \in \text{iset } xs. \neg f x$   
 shows  $\text{iset} (\text{ifilter } f xs) \cap \text{iset} (\text{ifilter} (\lambda x. \neg f x) xs) = \{\}$   
 using *assms* by *auto*

lemma *ifilter-inth-or*:

assumes  $\exists x \in \text{iset } xs. P x$   
 shows  $\exists x \in \text{iset} (\text{ifilter} (\lambda x. P x \vee Q x) xs). P x$

**proof** –

**have** 1:  $\exists i \leq \text{ilen}(\text{ifilter } (\lambda x. P x \vee Q x) xs). P (\text{inth } (\text{ifilter } (\lambda x. P x) xs) i)$   
**using** *assms by (metis (mono-tags, lifting) iset-ifilter ilen-gr-zero mem-Collect-eq inth-iset)*  
**obtain** *i* **where** 2:  $i \leq \text{ilen}(\text{ifilter } (\lambda x. P x \vee Q x) xs) \wedge P (\text{inth } (\text{ifilter } (\lambda x. P x) xs) i) \wedge$   
 $(\text{inth } (\text{ifilter } (\lambda x. P x) xs) i) \in \text{iset } (\text{ifilter } P xs)$   
**using** 1  
**by** *(metis (mono-tags, lifting) iset-ifilter assms ilen-gr-zero mem-Collect-eq inth-iset)*  
**have** 3:  $\text{iset } (\text{ifilter } P xs) \leq \text{iset } (\text{ifilter } (\lambda x. P x \vee Q x) xs)$   
**using** *assms subset-ifilter[of xs P] by auto*  
**have** 4:  $(\text{inth } (\text{ifilter } (\lambda x. P x) xs) i) \in \text{iset } (\text{ifilter } P xs)$   
**using** 2 **by** *auto*  
**have** 5:  $(\text{inth } (\text{ifilter } (\lambda x. P x) xs) i) \in \text{iset } (\text{ifilter } (\lambda x. P x \vee Q x) xs)$   
**using** 3 4 **by** *blast*  
**from** 2 5 **show** *?thesis* **by** *blast*  
**qed**

**lemma** *ifilter-iapp1*:

**assumes**  $\forall x \in \text{iset } xs. \neg P x$   
 $\exists y \in \text{iset } ys. P y$   
**shows**  $\text{ifilter } P (xs \ominus ys) = \text{ifilter } P ys$   
**using** *assms*  
**by** *(induct xs) auto*

**lemma** *ifilter-iapp2*:

**assumes**  $\forall x \in \text{iset } xs. \neg P x$   
 $\exists y \in \text{iset } ys. P y$   
**shows**  $\text{ifilter } P (ys \ominus xs) = \text{ifilter } P ys$   
**using** *assms*  
**by** *(induct ys) auto*

**lemma** *ifilter-iapp [simp]*:

**assumes**  $(\exists x \in \text{iset } (xs \ominus ys). P x)$   
 $(\exists x \in \text{iset } xs. P x)$   
 $(\exists x \in \text{iset } ys. P x)$   
**shows**  $\text{ifilter } P (xs \ominus ys) = (\text{ifilter } P xs) \ominus (\text{ifilter } P ys)$   
**using** *assms*  
**proof**  
*(induct xs arbitrary: ys)*  
**case** *(INil x)*  
**then show** *?case* **by** *simp*  
**next**  
**case** *(ICons x1a xs)*  
**then show** *?case*  
**proof**  
*(cases  $(\exists x \in \text{iset } xs. P x)$ )*  
**case** *True*  
**then show** *?thesis* **using** *ICons* **by** *auto*  
**next**

```

    case False
    then show ?thesis using ICons using ifilter-iapp1[of xs P]
    using iset-IConsD iset-iapp by auto
qed
qed

lemma ifilter-irev:
  assumes  $(\exists x \in \text{iset } xs. P x)$ 
  shows  $\text{irev } (\text{ifilter } P \text{ } xs) = \text{ifilter } P \text{ } (\text{irev } xs)$ 
using assms
proof
  (induct xs)
  case (INil x)
  then show ?case by simp
next
  case (ICons x1a xs)
  then show ?case
    proof (cases  $(\exists x \in \text{iset } xs. P x)$ )
    case True
    then show ?thesis
      proof (cases  $P \text{ } x1a$ )
      case True
      then show ?thesis using ICons True
        by (simp add: ifilter-iapp1)
      next
      case False
      then show ?thesis using ICons False  $(\exists x \in \text{iset } xs. P x)$ 
        by (simp add: ifilter-iapp2)
    qed
    next
    case False
    then show ?thesis using False ICons
      by (simp add: ifilter-iapp1)
    qed
qed
qed

```

```

lemma ifilter-True:
  assumes  $\forall x \in \text{iset } xs. P x$ 
  shows  $\text{ifilter } P \text{ } xs = xs$ 
using assms
by (meson ifilter-id-conv ilen-ifilter-le inth-iset)

```

```

lemma nfilter-imap:
  assumes  $\exists x \in \text{iset } (\text{imap } f \text{ } xs). P x$ 
  shows  $\text{nfilter } P \text{ } (\text{imap } f \text{ } xs) \text{ } n = (\text{nfilter } (P \circ f) \text{ } xs \text{ } n)$ 
using assms
by (induct xs arbitrary: n) auto

```

**lemma** *ifilter-imap*:

**assumes**  $\exists x \in \text{iset } (\text{imap } f \text{ } xs).$   $P \ x$

**shows**  $\text{ifilter } P \ (\text{imap } f \text{ } xs) = \text{imap } f \ (\text{ifilter } (P \circ f) \text{ } xs)$

**using** *assms*

**by** (*induct xs*) *auto*

**lemma** *ilen-nfilter-imap[simp]*:

**assumes**  $\exists x \in \text{iset } (\text{imap } f \text{ } xs).$   $P \ x$

**shows**  $\text{ilen } (\text{nfilter } P \ (\text{imap } f \text{ } xs) \ n) = \text{ilen}(\text{nfilter } (P \circ f) \text{ } xs \ n)$

**using** *assms* **by** (*simp add:nfilter-imap*)

**lemma** *ilen-ifilter-imap[simp]*:

**assumes**  $\exists x \in \text{iset } (\text{imap } f \text{ } xs).$   $P \ x$

**shows**  $\text{ilen } (\text{ifilter } P \ (\text{imap } f \text{ } xs)) = \text{ilen}(\text{ifilter } (P \circ f) \text{ } xs)$

**using** *assms* **by** (*simp add:ifilter-imap*)

**lemma** *nfilter-is-subset [simp]*:

**assumes**  $\exists x \in \text{iset } xs.$   $P \ x$

**shows**  $\text{iset } (\text{nfilter } P \text{ } xs \ n) \leq \{n+k \mid k. k \leq \text{ilen } xs\}$

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

**lemma** *ifilter-is-subset [simp]*:

**assumes**  $\exists x \in \text{iset } xs.$   $P \ x$

**shows**  $\text{iset } (\text{ifilter } P \text{ } xs) \leq \text{iset } xs$

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

**lemma** *ilen-nfilter-less*:

**assumes**  $\exists x \in \text{iset } xs.$   $P \ x$

$x \in \text{iset } xs$

$\neg P \ x$

**shows**  $\text{ilen}(\text{nfilter } P \text{ } xs \ n) < \text{ilen } xs$

**using** *assms*

**proof**

(*induct xs arbitrary: n*)

**case** (*INil x*)

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

**next**

**case** (*ICons x1a xs*)

**then show** *?case* **using** *le-imp-less-Suc* *ilen-nfilter-le* **by** (*simp-all, blast*)

**qed**

**lemma** *ilen-ifilter-less*:

**assumes**  $\exists x \in \text{iset } xs.$   $P \ x$

$x \in \text{iset } xs$

$\neg P \ x$

**shows**  $\text{ilen}(\text{ifilter } P \text{ } xs) < \text{ilen } xs$

**using** *assms*

**proof**

(*induct xs*)

```

case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
    using ilen-ifilter-le le-imp-less-Suc by (simp-all, blast)
qed

```

```

lemma nfilter-iset:
assumes  $\exists x \in \text{iset } xs. P x$ 
shows  $\text{iset } (\text{nfilter } P \text{ } xs \text{ } n) = \{n+i \mid i. i \leq \text{ilen } xs \wedge P(\text{inth } xs \text{ } i)\}$ 
using assms by auto

```

```

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

```

```

lemma ifilter-cong[fundef-cong]:
assumes xs = ys
    ( $\bigwedge x. x \in \text{iset } ys \implies P x = Q x$ )
shows  $\text{ifilter } P \text{ } xs = \text{ifilter } Q \text{ } ys$ 
using assms by (induct ys arbitrary: xs) auto

```

```

lemma iremdups-ifilter:
assumes  $\exists x \in \text{iset } xs. P x$ 
shows  $\text{iremdups}(\text{ifilter } P \text{ } xs) = \text{ifilter } P \text{ } (\text{iremdups } xs)$ 
using assms
by (induct xs) auto

```

```

lemma idistinct-imap-ifilter:
assumes  $\exists x \in \text{iset } xs. P x$ 
    idistinct (imap f xs)
shows  $\text{idistinct } (\text{imap } f \text{ } (\text{ifilter } P \text{ } xs))$ 
using assms by (induct xs) auto

```

```

lemma idistinct-nfilter [simp]:
    idistinct (nfilter P xs n)
by (induction xs arbitrary: n) auto

```

```

lemma idistinct-ifilter [simp]:
assumes idistinct xs
shows  $\text{idistinct } (\text{ifilter } P \text{ } xs)$ 
using assms by (induct xs) auto

```

```

lemma idistinct-ilen-ifilter:
assumes  $\exists x \in \text{iset } xs. P x$ 

```

$idistinct\ xs$   
**shows**  $ilen\ (ifilter\ P\ xs) + 1 = card\ (\{x. P\ x\}\ Int\ iset\ xs)$   
**using**  $assms$  **by**  $(induct\ xs)\ auto$

**lemma**  $idx-nfilter-mono$ :  
**assumes**  $\exists\ x \in iset\ xs . P\ x$   
 $na < ilen\ (nfilter\ P\ xs\ n)$   
**shows**  $inth\ (nfilter\ P\ xs\ n)\ na < inth\ (nfilter\ P\ xs\ n)\ (Suc\ na)$   
**using**  $assms$   
**proof**  $(induct\ xs\ arbitrary:\ n\ na)$   
**case**  $(INil\ x)$   
**then show**  $?case$  **by**  $simp$   
**next**  
**case**  $(ICons\ x1a\ xs)$   
**then show**  $?case$   
**proof** –  
**have**  $1$ :  $(\exists\ x \in iset\ xs . P\ x) \wedge P\ x1a \wedge na = 0 \longrightarrow$   
 $inth\ (nfilter\ P\ (x1a \odot xs)\ n)\ na < inth\ (nfilter\ P\ (x1a \odot xs)\ n)\ (Suc\ na)$   
**using**  $ICons$  **by**  $auto$   
 $(meson\ Suc-le-lessD\ ilen-gr-zero\ nfilter-lower-bound)+$   
**have**  $5$ :  $\neg(\exists\ x \in iset\ xs . P\ x) \longrightarrow$   
 $inth\ (nfilter\ P\ (x1a \odot xs)\ n)\ na < inth\ (nfilter\ P\ (x1a \odot xs)\ n)\ (Suc\ na)$   
**using**  $ICons\ gr-implies-not-zero$  **by**  $(auto, fastforce)$   
**show**  $?thesis$   
**using**  $1\ 5\ ICons.hyps\ ICons.prem(2)\ less-Suc-eq-0-disj$  **by**  $auto$   
**qed**  
**qed**

**lemma**  $idx-nfilter$ :  
**assumes**  $\exists\ x \in iset\ xs . P\ x$   
**shows**  $index-sequence\ (ifirst((nfilter\ P\ xs\ n)))\ (nfilter\ P\ xs\ n)$   
**using**  $assms$  **by**  $(simp\ add:\ index-sequence-def\ idx-nfilter-mono)$

**lemma**  $idx-nfilter-expand$ :  
**assumes**  $\exists\ x \in iset\ xs . P\ x$   
**shows**  $\forall\ na < ilen\ (nfilter\ P\ xs\ n). inth\ (nfilter\ P\ xs\ n)\ na < inth\ (nfilter\ P\ xs\ n)\ (Suc\ na)$   
**using**  $assms\ idx-nfilter$  **by**  $(simp\ add:\ index-sequence-def)$

**lemma**  $idx-nfilter-gr-eq$ :  
**assumes**  $\exists\ x \in iset\ xs . P\ x$   
 $k \leq j$   
 $j \leq ilen(nfilter\ P\ xs\ n)$   
**shows**  $inth\ (nfilter\ P\ xs\ n)\ k \leq inth\ (nfilter\ P\ xs\ n)\ j$   
**using**  $assms$  **by**  $(meson\ idx-nfilter\ idx-less-eq)$

**lemma**  $idx-nfilter-gr$ :  
**assumes**  $\exists\ x \in iset\ xs . P\ x$

**shows**  $(\forall j . k < j \wedge j \leq \text{ilen}(\text{nfilter } P \text{ } xs \text{ } n) \longrightarrow$   
 $\text{inth } (\text{nfilter } P \text{ } xs \text{ } n) \text{ } k < \text{inth } (\text{nfilter } P \text{ } xs \text{ } n) \text{ } j)$   
**using** *assms*  
**by** (*meson Suc-leI dual-order.strict-trans1 idx-nfilter-expand idx-nfilter-gr-eq*)

**lemma** *idx-nfilter-less-eq*:  
**assumes**  $\exists x \in \text{iset } xs . P \text{ } x$   
 $k \leq \text{ilen}(\text{nfilter } P \text{ } xs \text{ } n)$   
**shows**  $(\forall j \leq k . \text{inth } (\text{nfilter } P \text{ } xs \text{ } n) \text{ } j \leq \text{inth } (\text{nfilter } P \text{ } xs \text{ } n) \text{ } k)$   
**using** *assms* **by** (*simp add: idx-nfilter-gr-eq*)

**lemma** *idx-nfilter-less*:  
**assumes**  $\exists x \in \text{iset } xs . P \text{ } x$   
 $k \leq \text{ilen}(\text{nfilter } P \text{ } xs \text{ } n)$   
**shows**  $(\forall j < k . \text{inth } (\text{nfilter } P \text{ } xs \text{ } n) \text{ } j < \text{inth } (\text{nfilter } P \text{ } xs \text{ } n) \text{ } k)$   
**using** *assms*  
**by** (*simp add: idx-nfilter-gr*)

**lemma** *nfilter-prefix-iset-0*:  
**assumes**  $k \leq \text{ilen } xs$   
 $\exists x \in \text{iset } (\text{prefix } k \text{ } xs) . P \text{ } x$   
**shows**  $\text{ilen } (\text{nfilter } P \text{ } (\text{prefix } k \text{ } xs) \text{ } n) \leq \text{ilen } (\text{nfilter } P \text{ } xs \text{ } n)$   
**using** *assms*  
**proof** (*induct xs arbitrary: n k*)  
**case** (*INil x*)  
**then show** *?case* **by** *simp*  
**next**  
**case** (*ICons x1a xs*)  
**then show** *?case*  
**proof** (*cases k*)  
**case** *0*  
**then show** *?thesis* **using** *ICons* **by** *simp*  
**next**  
**case** (*Suc nat*)  
**then show** *?thesis* **using** *ICons prefix-subset* **by** *simp blast*  
**qed**  
**qed**

**lemma** *nfilter-subset*:  
**assumes**  $\exists x \in \text{iset } (\text{prefix } k \text{ } xs) . P \text{ } x$   
 $k \leq \text{ilen } xs$   
**shows**  $\text{iset } (\text{nfilter } P \text{ } (\text{prefix } k \text{ } xs) \text{ } n) \leq \text{iset } (\text{nfilter } P \text{ } xs \text{ } n)$   
**using** *assms*  
**proof** (*induct xs arbitrary:k n*)  
**case** (*INil x*)  
**then show** *?case* **by** *auto*  
**next**  
**case** (*ICons x1a xs*)  
**then show** *?case*



```

proof (cases k)
case 0
then show ?thesis using ICons by simp
next
case (Suc nat)
then show ?thesis using ICons by (auto simp add: inth-iset)
qed
qed

```

```

lemma nfilter-prefix-iset:
assumes  $\exists x \in \text{iset } (\text{prefix } k \text{ } xs) . P \ x$ 
            $k \leq \text{ilen } xs$ 
shows  $\text{iset } (nfilter \ P \ (\text{prefix } k \text{ } xs) \ n) = \{n+i \mid i. i \leq k \wedge P(\text{inth } xs \ i)\}$ 
using assms
by auto

```

```

lemma nfilter-suffix-iset:
assumes  $\exists x \in \text{iset } (\text{suffix } k \text{ } xs) . P \ x$ 
            $k \leq \text{ilen } xs$ 
shows  $\text{iset } (nfilter \ P \ (\text{suffix } k \text{ } xs) \ n) = \{n+i \mid i. i \leq \text{ilen } xs - k \wedge P(\text{inth } xs \ (k+i))\}$ 
using assms by auto

```

```

lemma nfilter-suffix-iset-a:
assumes  $\exists x \in \text{iset } (\text{suffix } k \text{ } xs) . P \ x$ 
            $k \leq \text{ilen } xs$ 
shows  $\text{iset } (nfilter \ P \ (\text{suffix } k \text{ } xs) \ n) = \{n+(j-k) \mid j. k \leq j \wedge j \leq \text{ilen } xs \wedge P(\text{inth } xs \ j)\}$ 
using assms Nat.le-diff-conv2 by auto fastforce

```

```

lemma nfilter-suffix-iset-b:
assumes  $\exists x \in \text{iset } (\text{suffix } k \text{ } xs) . P \ x$ 
            $k \leq \text{ilen } xs$ 
shows  $\text{iset } (nfilter \ P \ (\text{suffix } k \text{ } xs) \ n) = \{n+i \mid i. i \leq \text{ilen } xs - k \wedge P(\text{inth } xs \ (i+k))\}$ 
using assms nfilter-suffix-iset by (auto simp add: add.commute)

```

```

lemma nfilter-card:
assumes  $\exists x \in \text{iset } xs . P \ x$ 
shows  $\text{ilen } (nfilter \ P \ xs \ n) + 1 = \text{card } (\text{iset } (nfilter \ P \ xs \ n))$ 
using assms
by (induct xs arbitrary: n) auto

```

```

lemma nfilter-prefix-iset-1:
assumes  $k \leq \text{ilen } xs$ 
            $\exists x \in \text{iset } (\text{prefix } k \text{ } xs) . P \ x$ 
shows  $\text{iset } (\text{prefix } (\text{ilen}(nfilter \ P \ (\text{prefix } k \text{ } xs) \ n)) \ (nfilter \ P \ xs \ n)) =$ 
            $\{( \text{inth } (nfilter \ P \ xs \ n) \ i) \mid i. i \leq \text{ilen}(nfilter \ P \ (\text{prefix } k \text{ } xs) \ n)\}$ 
using prefix-iset[of (ilen(nfilter P (prefix k xs) n)) (nfilter P xs n)]
by (simp add: assms(1) assms(2) nfilter-prefix-iset-0)

```

```

lemma nfilter-prefix-subset:
assumes  $\exists x \in \text{iset } xs . P \ x$ 

```

$k \leq \text{ilen } (\text{nfilter } P \text{ } xs \text{ } n)$   
**shows**  $\text{iset } (\text{prefix } k \text{ } (\text{nfilter } P \text{ } xs \text{ } n)) \leq \text{iset } (\text{nfilter } P \text{ } xs \text{ } n)$   
**using** *assms*  
**using** *prefix-subset* **by** *blast*

**lemma** *ilen-ifilter-conv-card*:  
**assumes**  $\exists x \in \text{iset } xs. P \text{ } x$   
**shows**  $\text{ilen } (\text{ifilter } P \text{ } xs) + 1 = \text{card } \{i. i \leq \text{ilen } xs \wedge P (\text{inth } xs \text{ } i)\}$   
**proof** –  
**have** 1:  $\text{ilen } (\text{ifilter } P \text{ } xs) = \text{ilen}(\text{nfilter } P \text{ } xs \text{ } 0)$   
**by** (*simp add: assms nfilter-ilen*)  
**have** 2:  $\text{ilen } (\text{nfilter } P \text{ } xs \text{ } 0) + 1 = \text{card } (\text{iset } (\text{nfilter } P \text{ } xs \text{ } 0))$   
**by** (*meson assms nfilter-card*)  
**have** 3:  $\text{iset } (\text{nfilter } P \text{ } xs \text{ } 0) = \{i | i. i \leq \text{ilen } xs \wedge P(\text{inth } xs \text{ } i)\}$   
**using** *assms* **by** *auto*  
**show** *?thesis*  
**using** 1 2 3 **by** *auto*  
**qed**

**lemma** *ifilter-nfilter-prefix-ilen-0*:  
**assumes**  $P (\text{inth } xs \text{ } ((\text{inth } (\text{nfilter } P \text{ } xs \text{ } n) \text{ } k) - n))$   
 $k \leq \text{ilen } (\text{ifilter } P \text{ } xs)$   
**shows**  $(\exists x \in \text{iset } xs. P \text{ } x)$   
**using** *assms*  
**by** (*induction xs arbitrary: k*) *auto*

**lemma** *nfilter-ilen-n-zero*:  
**assumes**  $(\exists x \in \text{iset } xs. P \text{ } x)$   
**shows**  $\text{ilen } (\text{nfilter } P \text{ } xs \text{ } n) = \text{ilen } (\text{nfilter } P \text{ } xs \text{ } 0)$   
**using** *assms*  
**proof**  
*(induction xs arbitrary: n)*  
**case** (*INil x*)  
**then show** *?case* **by** *simp*  
**next**  
**case** (*ICons x1a xs*)  
**then show** *?case*  
**proof** (*cases n*)  
**case** 0  
**then show** *?thesis* **using** *ICons* **by** *blast*  
**next**  
**case** (*Suc nat*)  
**then show** *?thesis* **using** *ICons* **by** (*simp add: nfilter-ilen*)  
**qed**  
**qed**

**lemma** *nfilter-inth-n-zero-a*:  
**assumes**  $(\exists x \in \text{iset } xs. P \text{ } x)$

$k \leq \text{ilen } (\text{nfilter } P \text{ } xs \text{ } n)$   
**shows**  $n \leq (\text{inth } (\text{nfilter } P \text{ } xs \text{ } n) \text{ } k)$   
**using** *assms* **by** (*simp add: nfilter-lower-bound*)

**lemma** *nfilter-inth-n-zero:*

**assumes**  $(\exists x \in \text{iset } xs. P \text{ } x)$   
 $k \leq \text{ilen } (\text{nfilter } P \text{ } xs \text{ } n)$   
**shows**  $(\text{inth } (\text{nfilter } P \text{ } xs \text{ } n) \text{ } k) - n = \text{inth } (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k$   
**using** *assms*  
**proof**  
*(induction xs arbitrary: n k)*  
**case** (*INil x*)  
**then show** ?*case* **by** *simp*  
**next**  
**case** (*ICons x1a xs*)  
**then show** ?*case*  
**proof** (*cases*  $(\exists x \in \text{iset } xs. P \text{ } x)$ )  
**case** *True*  
**then show** ?*thesis*  
**proof** (*cases* *P x1a*)  
**case** *True*  
**then show** ?*thesis*  
**proof** (*cases* *k*)  
**case** *0*  
**then show** ?*thesis* **using** *ICons True* **by** *simp*  
**next**  
**case** (*Suc nat*)  
**then show** ?*thesis* **using** *ICons True*  $(\exists x \in \text{iset } xs. P \text{ } x)$   
**proof** *auto*  
**fix** *x*  
**assume** *a0*:  $k = \text{Suc } nat$   
**assume** *a1*:  $(\bigwedge k \text{ } n. k \leq \text{ilen } (\text{nfilter } P \text{ } xs \text{ } n) \implies$   
 $\text{inth } (\text{nfilter } P \text{ } xs \text{ } n) \text{ } k - n = \text{inth } (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k)$   
**assume** *a2*:  $nat \leq \text{ilen } (\text{nfilter } P \text{ } xs \text{ } (\text{Suc } n))$   
**assume** *a3*: *P x1a*  
**assume** *a4*:  $x \in \text{iset } xs$   
**assume** *a5*: *P x*  
**show**  $\text{inth } (\text{nfilter } P \text{ } xs \text{ } (\text{Suc } n)) \text{ } nat - n = \text{inth } (\text{nfilter } P \text{ } xs \text{ } (\text{Suc } 0)) \text{ } nat$   
**proof** –  
**have** *1*:  $\text{inth } (\text{nfilter } P \text{ } xs \text{ } (\text{Suc } n)) \text{ } nat - (\text{Suc } n) = \text{inth } (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } nat$   
**using** *a1 a2 a3 a4 a5* **by** *blast*  
**have** *2*:  $\text{inth } (\text{nfilter } P \text{ } xs \text{ } (\text{Suc } 0)) \text{ } nat - (\text{Suc } 0) = \text{inth } (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } nat$   
**by** (*metis a2 a1 a4 a5 nfilter-ilen-n-zero*)  
**from** *1 2* **show** ?*thesis*  
**by** (*metis One-nat-def Suc-diff-le*  $(\exists x \in \text{iset } xs. P \text{ } x)$  *a2 diff-Suc-Suc*  
*nfilter-ilen-n-zero nfilter-lower-bound*  
*ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc*)  
**qed**  
**qed**  
**qed**

```

next
case False
then show ?thesis using ICons True
proof auto
  fix x
  assume b0:  $\neg P\ x1a$ 
  assume b1:  $(\bigwedge k\ n. k \leq \text{ilen } (\text{nfilter } P\ xs\ n) \implies$ 
     $\text{inth } (\text{nfilter } P\ xs\ n)\ k - n = \text{inth } (\text{nfilter } P\ xs\ 0)\ k)$ 
  assume b2:  $k \leq \text{ilen } (\text{nfilter } P\ xs\ (\text{Suc } n))$ 
  assume b3:  $x \in \text{iset } xs$ 
  assume b4:  $P\ x$ 
  show  $\text{inth } (\text{nfilter } P\ xs\ (\text{Suc } n))\ k - n = \text{inth } (\text{nfilter } P\ xs\ (\text{Suc } 0))\ k$ 
  proof -
    have 3:  $\text{inth } (\text{nfilter } P\ xs\ (\text{Suc } n))\ k - (\text{Suc } n) = \text{inth } (\text{nfilter } P\ xs\ 0)\ k$ 
      using b1 b2 by blast
    have 4:  $\text{inth } (\text{nfilter } P\ xs\ (\text{Suc } 0))\ k - (\text{Suc } 0) = \text{inth } (\text{nfilter } P\ xs\ 0)\ k$ 
      by (metis True b1 b2 nfilter-ilen-n-zero)
    show ?thesis
    by (metis 3 4 One-nat-def Suc-diff-le True b2 diff-Suc-Suc nfilter-ilen-n-zero
      nfilter-lower-bound ordered-cancel-comm-monoid-diff-class.add-diff-inverse
      plus-1-eq-Suc)
  qed
qed
qed
next
case False
then show ?thesis using ICons False by auto
qed
qed

```

lemma *nfilter-n-zero*:

```

assumes  $(\exists\ x \in \text{iset } xs. P\ x)$ 
shows  $(\text{nfilter } P\ xs\ n) = \text{imap } (\lambda i. i+n) (\text{nfilter } P\ xs\ 0)$ 
using assms
proof -
  have 1:  $\text{ilen } (\text{nfilter } P\ xs\ n) = \text{ilen } (\text{imap } (\lambda i. i+n) (\text{nfilter } P\ xs\ 0))$ 
    using assms nfilter-ilen-n-zero by fastforce
  have 2:  $\bigwedge k. k \leq \text{ilen } (\text{nfilter } P\ xs\ n) \longrightarrow$ 
     $\text{inth } (\text{nfilter } P\ xs\ n)\ k = \text{inth } (\text{imap } (\lambda i. i+n) (\text{nfilter } P\ xs\ 0))\ k$ 
    using assms nfilter-inth-n-zero[of xs P - n]
    by (metis inth-imap le-add-diff-inverse2 nfilter-lower-bound)
  show ?thesis by (simp add: 1 2 interval-eq-inth-eq)
qed

```

lemma *nfilter-n-zero-a*:

```

assumes  $(\exists\ x \in \text{iset } xs. P\ x)$ 
shows  $(\text{nfilter } P\ xs\ 0) = \text{imap } (\lambda i. i-n) (\text{nfilter } P\ xs\ n)$ 
proof -
  have 1:  $\text{ilen } (\text{nfilter } P\ xs\ 0) = \text{ilen } (\text{imap } (\lambda i. i-n) (\text{nfilter } P\ xs\ n))$ 
    by (metis assms ilen-imap nfilter-ilen-n-zero)

```

**have** 2:  $\bigwedge k. k \leq \text{ilen } (\text{nfilter } P \text{ } xs \ 0) \longrightarrow$   
 $\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k = \text{inth } (\text{imap } (\lambda i. i - n) (\text{nfilter } P \text{ } xs \ n)) \ k$   
**using** *assms*  
**by** (*simp add: 1 inth-imap nfilter-inth-n-zero*)  
**show** *?thesis*  
**using** 1 2 *interval-eq-inth-eq* **by** *blast*  
**qed**

**lemma** *nfilter-count:*

**assumes**  $(\exists x \in \text{iset } xs. P \ x)$   
**shows**  $\text{card } \{(\text{inth } (\text{nfilter } P \text{ } xs \ n) \ k) \mid k. k \leq \text{ilen } (\text{nfilter } P \text{ } xs \ n)\} =$   
 $\text{ilen } (\text{nfilter } P \text{ } xs \ n) + 1$   
**using** *assms nfilter-card[of xs P n] iset-nfilter[of xs P n]*  
*inth-and-iset* **by** (*simp add: iset-inth*)

**lemma** *nfilter-holds:*

**assumes**  $(\exists x \in \text{iset } xs. P \ x)$   
**shows**  $(\forall x \in \text{iset } (\text{nfilter } P \text{ } xs \ n). P \ (\text{inth } xs \ (x - n)))$   
**using** *assms* **by** *auto*

**lemma** *nfilter-holds-not:*

**assumes**  $(\exists x \in \text{iset } xs. P \ x)$   
**shows**  $(\forall x \in (\{i + n \mid i. i \leq \text{ilen } xs\} - (\text{iset } (\text{nfilter } P \text{ } xs \ n)))) . \neg P \ (\text{inth } xs \ (x - n)))$   
**using** *assms* **by** *auto*

**lemma** *nfilter-holds-a:*

**assumes**  $(\exists x \in \text{iset } xs. P \ x)$   
**shows**  $(\forall i \leq \text{ilen } xs. (i + n) \in \text{iset } (\text{nfilter } P \text{ } xs \ n) \longrightarrow P \ (\text{inth } xs \ i))$   
**using** *assms* **by** *auto*

**lemma** *nfilter-holds-not-a:*

**assumes**  $(\exists x \in \text{iset } xs. P \ x)$   
**shows**  $(\forall i \leq \text{ilen } xs. P \ (\text{inth } xs \ i) \longrightarrow (i + n) \in \text{iset } (\text{nfilter } P \text{ } xs \ n))$   
**using** *assms* **by** *auto*

**lemma** *nfilter-holds-b:*

**assumes**  $(\exists x \in \text{iset } xs. P \ x)$   
**shows**  $(\forall i \leq \text{ilen } xs. (i + n) \in \text{iset } (\text{nfilter } P \text{ } xs \ n) = P \ (\text{inth } xs \ i))$   
**using** *assms* **by** *auto*

**lemma** *nfilter-holds-c:*

**assumes**  $(\exists x \in \text{iset } xs. P \ x)$   
 $i \leq \text{ilen } xs$   
**shows**  $(i + n) \in \text{iset } (\text{nfilter } P \text{ } xs \ n) = P \ (\text{inth } xs \ i)$   
**by** (*simp add: assms(1) assms(2)*)

**lemma** *nfilter-holds-d:*

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

$n \leq i$   
 $i \leq \text{ilen } xs + n$   
**shows**  $i \in \text{iset } (\text{nfilter } P \text{ } xs \text{ } n) = P (\text{inth } xs \text{ } (i-n))$   
**using** *assms*  
**by** (*metis diff-add le-diff-conv nfilter-holds-b*)

**lemma** *nfilter-holds-not-b*:  
**assumes**  $(\exists x \in \text{iset } xs. P \text{ } x)$   
 $n \leq i$   
 $i \leq \text{ilen } xs + n$   
**shows**  $i \notin \text{iset } (\text{nfilter } P \text{ } xs \text{ } n) = (\neg P (\text{inth } xs \text{ } (i-n)))$   
**using** *assms* **by** *auto*

**lemma** *nfilter-disjoint-iset-coset*:  
**assumes**  $(\exists x \in \text{iset } xs. P \text{ } x)$   
**shows**  $(\{i+n \mid i. i \leq \text{ilen } xs\} - (\text{iset } (\text{nfilter } P \text{ } xs \text{ } n))) \cap (\text{iset } (\text{nfilter } P \text{ } xs \text{ } n)) = \{\}$   
**using** *assms* **by** *auto*

**lemma** *nfilter-not-before*:  
**assumes**  $(\exists x \in \text{iset } xs. P \text{ } x)$   
 $i < (\text{inth } (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } 0)$   
**shows**  $\neg P (\text{inth } xs \text{ } i)$   
**proof** –  
**have**  $0: (\text{inth } (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } 0) \leq \text{ilen } xs$   
**by** (*metis add.left-neutral assms(1) ilen-gr-zero nfilter-upper-bound*)  
**have**  $1: i \notin \text{iset } (\text{nfilter } P \text{ } xs \text{ } 0)$   
**using** *assms*  
**proof** (*induction xs arbitrary: i*)  
**case** (*INil x*)  
**then show** *?case* **by** *simp*  
**next**  
**case** (*ICons x1a xs*)  
**then show** *?case*  
**by** (*metis idx-nfilter idx-greater inth-and-iset leD*)  
**qed**  
**have**  $2: i \notin \text{iset } (\text{nfilter } P \text{ } xs \text{ } 0) \wedge i \leq \text{ilen } xs \longrightarrow \neg P (\text{inth } xs \text{ } (i))$   
**by** (*metis add.right-neutral nfilter-holds-not-a inth-iset*)  
**have**  $3: i \leq \text{ilen } xs$   
**using**  $0$  *assms(2)* **by** *linarith*  
**from**  $0 \ 1 \ 2 \ 3$  **show** *?thesis* **by** *auto*  
**qed**

**lemma** *nfilter-n-not-before*:  
**assumes**  $(\exists x \in \text{iset } (\text{suffix } n \text{ } xs). P \text{ } x)$   
 $n \leq \text{ilen } xs$   
 $n \leq i$   
 $i < (\text{inth } (\text{nfilter } P \text{ } (\text{suffix } n \text{ } xs) \text{ } n) \text{ } 0)$   
**shows**  $\neg P (\text{inth } xs \text{ } (i))$   
**proof** –  
**have**  $0: (\text{inth } (\text{nfilter } P \text{ } (\text{suffix } n \text{ } xs) \text{ } n) \text{ } 0) \leq \text{ilen } xs$

```

  by (metis assms(1) assms(2) ilen-gr-zero suffix-ilen-good
      le-add-diff-inverse nfilter-upper-bound)
have 1:  $i \notin \text{iset } (nfilter\ P\ (\text{suffix } n\ xs)\ n)$ 
  using assms
  proof (induction xs arbitrary: i)
  case (INil x)
  then show ?case by simp
  next
  case (ICons x1a xs)
  then show ?case
    by (metis idx-nfilter idx-greater inth-and-iset leD)
  qed
have 2:  $i \notin \text{iset } (nfilter\ P\ (\text{suffix } n\ xs)\ n) \wedge n \leq i \wedge i \leq \text{ilen } xs \longrightarrow \neg P\ (\text{inth } xs\ (i))$ 
  using assms nfilter-holds-not-a[of  $(\text{suffix } n\ xs)\ P\ n$ ]
  by (metis add-le-imp-le-right inth-suffix suffix-ilen le-add-diff-inverse2
      ordered-cancel-comm-monoid-diff-class.add-diff-inverse)
have 3:  $n \leq i \wedge i \leq \text{ilen } xs$ 
  using 0 assms(3) assms(4) by linarith
from 0 1 2 3 show ?thesis by auto
qed

```

**lemma** *nfilter-not-after:*

```

assumes  $(\exists x \in \text{iset } xs. P\ x)$ 
         $(\text{inth } (nfilter\ P\ xs\ 0)\ (\text{ilen } (nfilter\ P\ xs\ 0))) < i$ 
         $i \leq \text{ilen } xs$ 
shows  $\neg P\ (\text{inth } xs\ (i))$ 
proof -
have 1:  $i \notin \text{iset } (nfilter\ P\ xs\ 0)$ 
  using assms
  proof (induction xs arbitrary: i)
  case (INil x)
  then show ?case by auto
  next
  case (ICons x1a xs)
  then show ?case
    by (metis idx-nfilter-gr-eq inth-and-iset leD le-refl)
  qed
have 2:  $i \notin \text{iset } (nfilter\ P\ xs\ 0) \wedge i \leq \text{ilen } xs \longrightarrow \neg P\ (\text{inth } xs\ (i))$ 
  by (metis add.right-neutral nfilter-holds-b inth-iset)
have 3:  $i \leq \text{ilen } xs$ 
  by (simp add: assms(3))
from 1 2 3 show ?thesis by auto
qed

```

**lemma** *nfilter-n-not-after:*

```

assumes  $(\exists x \in \text{iset } (\text{suffix } n\ xs). P\ x)$ 
         $n \leq \text{ilen } xs$ 
         $(\text{inth } (nfilter\ P\ (\text{suffix } n\ xs)\ n)\ (\text{ilen } (nfilter\ P\ (\text{suffix } n\ xs)\ n))) < i$ 
         $i \leq \text{ilen } xs$ 
shows  $\neg P\ (\text{inth } xs\ (i))$ 

```

```

proof –
  have 1:  $i \notin \text{iset } (\text{nfilter } P \text{ (suffix } n \text{ xs) } n)$ 
    using assms
    proof (induction xs arbitrary: i)
    case (INil x)
    then show ?case by auto
    next
    case (ICons x1a xs)
    then show ?case
      by (metis idx-nfilter-gr-eq inth-and-iset leD le-eq-less-or-eq)
    qed
  have 2:  $i \notin \text{iset } (\text{nfilter } P \text{ (suffix } n \text{ xs) } n) \wedge n \leq i \wedge i \leq \text{ilen xs} \longrightarrow \neg P \text{ (inth xs (i))}$ 
    using assms nfilter-holds-not-a[of suffix n xs P n]
    by (metis add-le-imp-le-left inth-suffix suffix-ilen le-add-diff-inverse2
      ordered-cancel-comm-monoid-diff-class.add-diff-inverse)
  have 3:  $n \leq i \wedge i \leq \text{ilen xs}$ 
    by (meson assms(1) assms(3) assms(4) dual-order.strict-implies-order dual-order.strict-trans2
      nfilter-inth-n-zero-a order-refl)
  from 1 2 3 show ?thesis by auto
qed

```

**lemma** *nfilter-not-between-help-a:*

```

assumes ( $\bigwedge k. i.$ 
   $\exists a \in \text{iset xs. } P a \implies$ 
   $k < \text{ilen } (\text{nfilter } P \text{ xs } 0) \implies$ 
   $\text{inth } (\text{nfilter } P \text{ xs } 0) k < i \implies$ 
   $i < \text{inth } (\text{nfilter } P \text{ xs } 0) (\text{Suc } k) \implies$ 
   $\text{inth } (\text{nfilter } P \text{ xs } 0) (\text{Suc } k) \leq \text{ilen xs} \implies$ 
   $i \notin \text{iset } (\text{nfilter } P \text{ xs } 0))$ 
   $\exists a \in \text{iset } (x1a \odot \text{xs}). P a$ 
   $k < \text{ilen } (\text{nfilter } P \text{ (x1a } \odot \text{ xs) } 0)$ 
   $\text{inth } (\text{nfilter } P \text{ (x1a } \odot \text{ xs) } 0) k < i$ 
   $i < \text{inth } (\text{nfilter } P \text{ (x1a } \odot \text{ xs) } 0) (\text{Suc } k)$ 
   $\text{inth } (\text{nfilter } P \text{ (x1a } \odot \text{ xs) } 0) (\text{Suc } k) \leq \text{ilen } (x1a \odot \text{xs})$ 
   $\exists x \in \text{iset xs. } P x$ 
   $P x1a$ 
shows  $i \notin \text{iset } (\text{nfilter } P \text{ (x1a } \odot \text{ xs) } 0)$ 
proof –
  have 1:  $k=0 \implies i \notin \text{iset } (\text{nfilter } P \text{ (x1a } \odot \text{ xs) } 0)$ 
    using assms by auto
    (metis One-nat-def Suc-less-eq2 diff-Suc-1 ilen-gr-zero nfilter-not-before
      nfilter-inth-n-zero)
  have 2:  $\bigwedge n. k = (\text{Suc } n) \implies i \notin \text{iset } (\text{nfilter } P \text{ (x1a } \odot \text{ xs) } 0)$ 
    using assms
    proof auto
    fix n
    fix x
    fix ka
    assume a0: k = Suc n

```



```

assume a1: ( $\bigwedge k\ i.\ k < \text{ilen} (\text{nfilter } P\ xs\ 0) \implies$ 
   $\text{inth} (\text{nfilter } P\ xs\ 0)\ k < i \implies$ 
   $i < \text{inth} (\text{nfilter } P\ xs\ 0)\ (\text{Suc } k) \implies$ 
   $\text{inth} (\text{nfilter } P\ xs\ 0)\ (\text{Suc } k) \leq \text{ilen } xs \implies$ 
   $\neg P (\text{inth } xs\ i)$ )
assume a2:  $n < \text{ilen} (\text{nfilter } P\ xs\ (\text{Suc } 0))$ 
assume a3:  $\text{inth} (\text{nfilter } P\ xs\ (\text{Suc } 0))\ n < \text{Suc } ka$ 
assume a4:  $\text{Suc } ka < \text{inth} (\text{nfilter } P\ xs\ (\text{Suc } 0))\ (\text{Suc } n)$ 
assume a5:  $\text{inth} (\text{nfilter } P\ xs\ (\text{Suc } 0))\ (\text{Suc } n) \leq \text{Suc } (\text{ilen } xs)$ 
assume a6:  $P\ x1a$ 
assume a7:  $x \in \text{iset } xs$ 
assume a8:  $P\ x$ 
assume a9:  $i = \text{Suc } ka$ 
assume a10:  $P (\text{inth } xs\ ka)$ 
show False
proof –
  have 3:  $n < \text{ilen} (\text{nfilter } P\ xs\ 0) \implies$ 
     $\text{inth} (\text{nfilter } P\ xs\ 0)\ n < ka \implies$ 
     $ka < \text{inth} (\text{nfilter } P\ xs\ 0)\ (\text{Suc } n) \implies$ 
     $\text{inth} (\text{nfilter } P\ xs\ 0)\ (\text{Suc } n) \leq \text{ilen } xs \implies$ 
     $\neg P (\text{inth } xs\ ka)$ 
    using a1[of  $n\ ka$ ] by auto
  have 4:  $n < \text{ilen} (\text{nfilter } P\ xs\ 0)$ 
    by (metis a2 a7 a8 nfilter-ilen-n-zero)
  have 5:  $\text{inth} (\text{nfilter } P\ xs\ 0)\ (\text{Suc } n) \leq \text{ilen } xs$ 
    by (metis 4 Suc-leI add commute add.right-neutral assms(7) nfilter-upper-bound)
  have 6:  $\exists x \in \text{iset } xs.\ P\ x$ 
    using assms(7) by auto
  have 7:  $\text{Suc } 0 \leq \text{ilen} (\text{nfilter } P\ xs\ (\text{Suc } n))$ 
    by (metis One-nat-def Suc-le-mono Suc-mono a2 a7 a8 ilen-gr-zero le-SucE
      nfilter-ilen-n-zero not-add-less1 plus-1-eq-Suc)
  have 8:  $ka < \text{inth} (\text{nfilter } P\ xs\ 0)\ (\text{Suc } n)$ 
    using nfilter-inth-n-zero[of  $xs\ P\ \text{Suc } n\ \text{Suc } 0$ ] a4 6 a2 by linarith
  have 9:  $\text{inth} (\text{nfilter } P\ xs\ 0)\ n < ka$ 
    using a2 a3 a7 a8 nfilter-inth-n-zero[of  $xs\ P\ n\ \text{Suc } 0$ ]
    by (metis One-nat-def add commute dual-order.strict-implies-order less-diff-conv2
      nfilter-lower-bound plus-1-eq-Suc)
  have 10:  $\neg P (\text{inth } xs\ ka)$ 
    using 3 4 9 8 5 by auto
  from a10 10 show ?thesis by auto
qed
qed
show ?thesis using 1 2
using gr0-implies-Suc by blast
qed

lemma nfilter-not-between-help-b:
assumes ( $\bigwedge k\ i.$ 
   $\exists a \in \text{iset } xs.\ P\ a \implies$ 
   $k < \text{ilen} (\text{nfilter } P\ xs\ 0) \implies$ 

```

```

  inth (nfilter P xs 0) k < i ==>
  i < inth (nfilter P xs 0) (Suc k) ==>
  inth (nfilter P xs 0) (Suc k) ≤ ilen xs ==>
  i ∉ iset (nfilter P xs 0)
  ∃ a ∈ iset (x1a ⊙ xs). P a
  k < ilen (nfilter P (x1a ⊙ xs) 0)
  inth (nfilter P (x1a ⊙ xs) 0) k < i
  i < inth (nfilter P (x1a ⊙ xs) 0) (Suc k)
  inth (nfilter P (x1a ⊙ xs) 0) (Suc k) ≤ ilen (x1a ⊙ xs)
  ∃ x ∈ iset xs. P x
  ¬ P x1a
shows i ∉ iset (nfilter P (x1a ⊙ xs) 0)
proof -
have 1: k=0 ==> i ∉ iset (nfilter P (x1a ⊙ xs) 0)
  using assms
proof auto
  fix x :: 'a and ka :: nat
  assume a1: inth (nfilter P xs (Suc 0)) 0 < Suc ka
  assume a2: x ∈ iset xs
  assume a3: P x
  assume a4: ⋀ k i. [k < ilen (nfilter P xs 0); inth (nfilter P xs 0) k < i;
    i < inth (nfilter P xs 0) (Suc k); inth (nfilter P xs 0) (Suc k) ≤ ilen xs]
    ==> ¬ P (inth xs i)
  assume a5: P (inth xs ka)
  assume a6: 0 < ilen (nfilter P xs (Suc 0))
  assume a7: Suc ka < inth (nfilter P xs (Suc 0)) (Suc 0)
  assume a8: inth (nfilter P xs (Suc 0)) (Suc 0) ≤ Suc (ilen xs)
  have f9: ∃ a. a ∈ iset xs ∧ P a
  using a3 a2 by blast
  then have f10: 0 ≤ ilen (nfilter P xs (Suc 0)) ⟶ inth (nfilter P xs (Suc 0)) 0 ≠ 0
  by (metis le-zero-eq nfilter-lower-bound not-less-eq-eq)
  have f11: ilen (nfilter P xs (Suc 0)) = ilen (nfilter P xs 0)
  using f9 by (meson nfilter-ilen-n-zero)
  obtain nn :: nat ⇒ nat ⇒ nat where
  f12: inth (nfilter P xs (Suc 0)) (Suc 0) = Suc (nn (inth (nfilter P xs (Suc 0)) (Suc 0)) ka) ∧
    ka < nn (inth (nfilter P xs (Suc 0)) (Suc 0)) ka
  using a7 by (meson Suc-less-eq2)
  then have inth (nfilter P xs 0) (Suc 0) = nn (inth (nfilter P xs (Suc 0)) (Suc 0)) ka
  using f9 a6 by (metis One-nat-def diff-Suc-1 le-zero-eq neq0-conv
    nfilter-inth-n-zero not-less-eq-eq)
  then have ¬ 0 ≤ ilen (nfilter P xs 0)
  using f12 f11 f10 f9 a8 a6 a5 a4 a1
  by (metis One-nat-def Suc-le-mono diff-Suc-1 less-Suc-eq-0-disj nfilter-inth-n-zero)
  then show False
  by blast
qed
have 2: ⋀ n. k = (Suc n) ==> i ∉ iset (nfilter P (x1a ⊙ xs) 0)
  using assms
proof auto
  fix n

```

```

fix x
fix ka
assume a0: k = Suc n
assume a1: ( $\bigwedge k i. k < \text{ilen } (\text{nfilter } P \text{ } xs \ 0) \implies$ 
   $\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k < i \implies$ 
   $i < \text{inth } (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } k) \implies$ 
   $\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } k) \leq \text{ilen } xs \implies$ 
   $\neg P \ (\text{inth } xs \ i)$ )
assume a2:  $\text{Suc } n < \text{ilen } (\text{nfilter } P \text{ } xs \ (\text{Suc } 0))$ 
assume a3:  $\text{inth } (\text{nfilter } P \text{ } xs \ (\text{Suc } 0)) \ (\text{Suc } n) < \text{Suc } ka$ 
assume a4:  $\text{Suc } ka < \text{inth } (\text{nfilter } P \text{ } xs \ (\text{Suc } 0)) \ (\text{Suc } (\text{Suc } n))$ 
assume a5:  $\text{inth } (\text{nfilter } P \text{ } xs \ (\text{Suc } 0)) \ (\text{Suc } (\text{Suc } n)) \leq \text{Suc } (\text{ilen } xs)$ 
assume a6:  $\neg P \ x1a$ 
assume a7:  $x \in \text{iset } xs$ 
assume a8:  $P \ x$ 
assume a9:  $i = \text{Suc } ka$ 
assume a10:  $P \ (\text{inth } xs \ ka)$ 
show False
proof -
  have 3:  $\text{Suc } n < \text{ilen } (\text{nfilter } P \text{ } xs \ 0) \implies$ 
     $\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } n) < ka \implies$ 
     $ka < \text{inth } (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } (\text{Suc } n)) \implies$ 
     $\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } (\text{Suc } n)) \leq \text{ilen } xs \implies$ 
     $\neg P \ (\text{inth } xs \ ka)$ 
  using a1[of Suc n ka] by auto
  have 4:  $(\text{Suc } n) < \text{ilen } (\text{nfilter } P \text{ } xs \ 0)$ 
  by (metis a2 a7 a8 nfilter-ilen-n-zero)
  have 5:  $\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } (\text{Suc } n)) \leq \text{ilen } xs$ 
  by (metis 4 Suc-leI a7 a8 add.left-neutral nfilter-upper-bound)
  have 6:  $\exists x \in \text{iset } xs. P \ x$ 
  by (simp add: assms(7))
  have 7:  $\text{Suc } 0 \leq \text{ilen } (\text{nfilter } P \text{ } xs \ (\text{Suc } n))$ 
  by (metis One-nat-def Suc-le-mono Suc-mono a2 a7 a8 ilen-gr-zero le-SucE
    nfilter-ilen-n-zero not-add-less1 plus-1-eq-Suc)
  have 8:  $ka < \text{inth } (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } (\text{Suc } n))$ 
  using nfilter-inth-n-zero[of xs P Suc (Suc n) Suc 0] a4 6 a2 by linarith
  have 9:  $\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } n) < ka$ 
  using a2 a3 a7 a8 nfilter-inth-n-zero[of xs P Suc n Suc 0]
  by (metis One-nat-def add commute dual-order.strict-implies-order less-diff-conv2
    nfilter-lower-bound plus-1-eq-Suc)
  have 10:  $\neg P \ (\text{inth } xs \ ka)$ 
  using 3 4 9 8 5 by auto
  from a10 10 show ?thesis by auto
qed
qed
show ?thesis using 1 2
using gr0-implies-Suc by blast
qed

```

```

lemma nfilter-not-between-help:
  assumes  $(\exists x \in \text{iset } xs. P x)$ 
     $k < \text{ilen } (\text{nfilter } P \text{ } xs \ 0)$ 
     $(\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) < i$ 
     $i < (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } k))$ 
     $(\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } k)) \leq \text{ilen } xs$ 
  shows  $i \notin \text{iset } (\text{nfilter } P \text{ } xs \ 0)$ 
using assms
proof (induction xs arbitrary: i k)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
  proof (cases  $(\exists x \in \text{iset } xs. P x)$ )
    case True
    then show ?thesis
      proof (cases P x1a)
        case True
        then show ?thesis using ICons True using nfilter-not-between-help-a[of xs P x1a k i]
          by fastforce
        next
        case False
        then show ?thesis using ICons False using nfilter-not-between-help-b[of xs P x1a k i]
          by simp
      qed
    next
    case False
    then show ?thesis using ICons False by auto
  qed
qed

```

```

lemma nfilter-not-between:
  assumes  $(\exists x \in \text{iset } xs. P x)$ 
     $(\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) < i$ 
     $i < (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } k))$ 
     $k < \text{ilen } (\text{nfilter } P \text{ } xs \ 0)$ 
  shows  $\neg P (\text{inth } xs \ (i))$ 
proof –
  have 0:  $(\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } k)) \leq \text{ilen } xs$ 
    by (metis Suc-leI add-cancel-right-left assms(1) assms(4) nfilter-upper-bound)
  have 1:  $i \leq \text{ilen } xs$ 
    using 0 assms(3) by linarith
  have 2:  $k < \text{ilen } (\text{nfilter } P \text{ } xs \ 0) \wedge (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) < i \wedge$ 
     $i < (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } k)) \wedge (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } k)) \leq \text{ilen } xs \longrightarrow$ 
     $i \notin \text{iset } (\text{nfilter } P \text{ } xs \ 0)$ 
  using assms(1)
  proof (induction xs arbitrary: i k)
    case (INil x)
    then show ?case by simp

```

```

next
case (ICons x1a xs)
then show ?case using assms nfilter-not-between-help
by metis
qed
have  $\exists i : i \notin \text{iset } (nfilter\ P\ xs\ 0) \wedge i \leq \text{ilen } xs \longrightarrow \neg P\ (\text{inth } xs\ (i))$ 
by (metis add.right-neutral nfilter-holds-b inth-iset)
from 0 1 2 3 show ?thesis using assms by blast
qed

```

```

lemma idx-imp-idistinct:
assumes index-sequence (inth xs 0) xs
shows idistinct xs
using assms
proof (induction xs)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
by (auto simp add: idx-expand1
(metis idx-expand1 idx-greater-first inth-and-iset le-zero-eq not-less not-less-iff-gr-or-eq))
qed

```

```

lemma idx-iset-eq:
assumes index-sequence (inth xs 0) xs
      index-sequence (inth ys 0) ys
      iset xs = iset ys
shows xs = ys
using assms
proof
(induction xs arbitrary: ys)
case (INil x)
then show ?case
by (metis idistinct.simps(2) iapp-INil iapp-not-INil idx-imp-idistinct
interval.exhaust interval-in-iset-conv-decomp-last)
next
case (ICons x1a xs)
then show ?case
proof (cases ys)
case (INil x1)
then show ?thesis
using ICons.prem1 ICons.prem3 idx-less-last-1 le-eq-less-or-eq inth-iset by fastforce
next
case (ICons x21 x22)
then show ?thesis
proof (cases x1a = x21)
case True

```

```

then show ?thesis
proof -
  have 1:  $\text{ilen } (x1a \odot xs) = \text{ilen } (x21 \odot x22)$ 
    by (metis  $\text{ICons.prem}(1)$   $\text{ICons.prem}(2)$   $\text{ICons.prem}(3)$   $\text{idistinct-card}$   $\text{idx-imp-idistinct}$ 
         $\text{ICons nat.inject}$ )
  have 2:  $\text{index-sequence } (\text{inth } xs \ 0) \ xs$ 
    using  $\text{ICons.prem}(1)$   $\text{idx-expand1}$  by auto
  have 3:  $\text{index-sequence } (\text{inth } x22 \ 0) \ x22$ 
    using  $\text{ICons.prem}(2)$   $\text{idx-expand1}$   $\text{ICons}$  by auto
  have 4:  $\text{idistinct } xs$ 
    using 2  $\text{idx-imp-idistinct}$  by auto
  have 5:  $\text{idistinct } x22$ 
    by (simp add: 3  $\text{idx-imp-idistinct}$ )
  have 6:  $\text{iset } (x1a \odot xs) = \{x1a\} \cup \text{iset } xs$ 
    by auto
  have 7:  $x1a \notin \text{iset } xs$ 
    by (meson  $\text{ICons.prem}(1)$   $\text{idistinct.simp}(2)$   $\text{idx-imp-idistinct}$ )
  have 8:  $\text{iset } (x21 \odot x22) = \{x21\} \cup \text{iset } x22$ 
    by auto
  have 9:  $x21 \notin \text{iset } x22$ 
    using  $\text{ICons.prem}(2)$   $\text{idx-imp-idistinct}$   $\text{ICons}$  by fastforce
  have 10:  $\text{iset } xs = \text{iset } x22$ 
    using 7 9  $\text{ICons.prem}(3)$   $\text{True}$   $\text{ICons}$  by fastforce
  have 11:  $xs = x22$ 
    using 10 2 3  $\text{ICons.IH}$  by blast
  have 12:  $x1a \odot xs = x21 \odot x22$ 
    by (simp add: 11  $\text{True}$ )
  show ?thesis by (simp add: 12  $\text{ICons}$ )
qed
next
case False
then show ?thesis using  $\text{ICons}$   $\text{idx-imp-idistinct}$   $\text{interval-hd-in-iset}$   $\text{idx-expand1}$ 
by (metis (full-types)  $\text{ICons.prem}(1)$   $\text{ICons.prem}(2)$   $\text{ICons.prem}(3)$   $\text{idistinct.simp}(2)$ 
     $\text{inth-zero}$   $\text{linorder-neqE-nat}$ )
qed
qed
qed

```

**lemma**  $\text{ifilter-nfilter-prefix-idx-a}$ :

```

assumes  $P \ (\text{inth } xs \ (\text{inth } (\text{nfilter } P \ xs \ 0) \ k))$ 
         $k \leq \text{ilen } (\text{ifilter } P \ xs)$ 
shows  $\text{index-sequence } (\text{inth } (\text{prefix } k \ (\text{nfilter } P \ xs \ 0)) \ 0) \ (\text{prefix } k \ (\text{nfilter } P \ xs \ 0))$ 
using  $\text{assms}$  by (auto simp add:  $\text{index-sequence-def}$ )
        (metis  $\text{diff-zero}$   $\text{ifilter-nfilter-prefix-ilen-0}$   $\text{idx-nfilter-mono}$ )

```

**lemma**  $\text{ifilter-nfilter-prefix-idx-a-1}$ :

```

assumes  $\exists x \in \text{iset } xs. P \ x$ 
         $k \leq \text{ilen } (\text{ifilter } P \ xs)$ 
shows  $\text{index-sequence } (\text{inth } (\text{prefix } k \ (\text{nfilter } P \ xs \ 0)) \ 0) \ (\text{prefix } k \ (\text{nfilter } P \ xs \ 0))$ 
using  $\text{assms}$  by (auto simp add:  $\text{index-sequence-def}$ )

```

(meson idx-nfilter-mono)

**lemma** *ifilter-nfilter-suffix-idx-a:*

**assumes**  $P \text{ (inth } xs \text{ ( (inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k)) \text{ ) )}$   
 $k \leq \text{ilen (ifilter } P \text{ } xs)$

**shows**  $\text{index-sequence (inth (suffix } k \text{ (nfilter } P \text{ } xs \text{ } 0)) \text{ } 0) \text{ (suffix } k \text{ (nfilter } P \text{ } xs \text{ } 0))}$

**using** *assms by (simp add: index-sequence-def)*

(metis add.commute diff-zero ifilter-nfilter-prefix-ilen-0 idx-nfilter-mono less-diff-conv)

**lemma** *ifilter-nfilter-suffix-idx-a-1:*

**assumes**  $\exists x \in \text{iset } xs. P \text{ } x$

$k \leq \text{ilen (ifilter } P \text{ } xs)$

**shows**  $\text{index-sequence (inth (suffix } k \text{ (nfilter } P \text{ } xs \text{ } 0)) \text{ } 0) \text{ (suffix } k \text{ (nfilter } P \text{ } xs \text{ } 0))}$

**using** *assms*

**by** (simp add: index-sequence-def idx-nfilter-mono nfilter-ilen)

**lemma** *ifilter-nfilter-prefix-idx-b:*

**assumes**  $P \text{ (inth } xs \text{ ( (inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k)) \text{ ) )}$   
 $k \leq \text{ilen (ifilter } P \text{ } xs)$

**shows**  $\text{index-sequence (inth (nfilter } P \text{ (prefix ((inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ ) } xs) \text{ } 0) \text{ } 0)$   
 $(\text{nfilter } P \text{ (prefix ((inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ ) } xs) \text{ } 0)$

**using** *assms idx-nfilter[of (prefix (inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ } xs) \text{ } P \text{ } 0]*

**by** (metis add.right-neutral ilast-iffirst inth-suffix le0 le-refl inth-iset)

**lemma** *ifilter-nfilter-prefix-idx-b-1:*

**assumes**  $\exists x \in \text{iset } xs. P \text{ } x$

$k \leq \text{ilen (ifilter } P \text{ } xs)$

**shows**  $\text{index-sequence (inth (nfilter } P \text{ (prefix ((inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ ) } xs) \text{ } 0) \text{ } 0)$   
 $(\text{nfilter } P \text{ (prefix ((inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ ) } xs) \text{ } 0)$

**using** *assms*

**by** (metis ifilter-nfilter-prefix-idx-b nfilter-holds nfilter-ilen nfilter-inth-n-zero inth-iset)

**lemma** *ifilter-nfilter-suffix-idx-b:*

**assumes**  $P \text{ (inth } xs \text{ ( (inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k)) \text{ ) )}$   
 $k \leq \text{ilen (ifilter } P \text{ } xs)$

**shows**  $\text{index-sequence (inth (nfilter } P \text{ (suffix ((inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ ) } xs)$   
 $(\text{inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k)) \text{ } 0)$   
 $(\text{nfilter } P \text{ (suffix ((inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ ) } xs)$   
 $(\text{inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k))$

**using** *assms*

**by** (auto simp add: index-sequence-def)

(metis idx-nfilter-mono ifirst-suffix ilen-gr-zero

suffix-ilen-code ilen-nfilter-le less-le-trans not-less0 inth-iset)

**lemma** *ifilter-nfilter-suffix-idx-b-1:*

**assumes**  $\exists x \in \text{iset } xs. P \text{ } x$

$k \leq \text{ilen (ifilter } P \text{ } xs)$

**shows**  $\text{index-sequence (inth (nfilter } P \text{ (suffix ((inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ ) } xs)$   
 $(\text{inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k)) \text{ } 0)$

$(\text{nfilter } P (\text{suffix } ((\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k}) ) \text{ xs}) (\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k}))$   
**using** *assms*  
**by** (*metis ifilter-nfilter-suffix-idx-b nfilter-holds nfilter-ilen nfilter-inth-n-zero inth-iset*)

**lemma** *ifilter-nfilter-prefix-iset-eq*:  
**assumes**  $P (\text{inth } \text{xs} \quad ( (\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k})) )$   
 $k \leq \text{ilen } (\text{ifilter } P \text{ xs})$   
**shows**  $\text{iset } (\text{prefix } k (\text{nfilter } P \text{ xs } 0)) =$   
 $\text{iset } (\text{nfilter } P (\text{prefix } ((\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k}) ) \text{ xs}) 0)$

**proof** –  
**have** 1:  $(\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k}) \leq \text{ilen } \text{xs}$   
**by** (*metis assms(1) assms(2) diff-zero ifilter-nfilter-prefix-ilen-0 ilen-gr-zero nfilter-ilen nfilter-upper-bound ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)  
**have** 2:  $\exists x \in \text{iset } \text{xs} . P \text{ x}$   
**using** 1 *assms(1) inth-iset* **by** *blast*  
**have** 3:  $\{ i. i \leq \text{ilen}(\text{prefix } ((\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k})) \text{ xs}) \wedge$   
 $P (\text{inth } (\text{prefix } ((\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k})) \text{ xs}) i) \} =$   
 $\{ i. i \leq (\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k}) \wedge P (\text{inth } \text{xs } i) \}$   
**using** 1 **by** *auto*  
**have** 4:  $\exists x \in \text{iset}(\text{prefix } ((\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k})) \text{ xs}). P \text{ x}$   
**by** (*metis 1 assms(1) inth-prefix prefix-ilen-good inth-iset order-refl*)  
**have** 5:  $\{ i. i \leq (\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k}) \wedge P (\text{inth } \text{xs } i) \} =$   
 $\{ i. i \leq (\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k}) \wedge i \in \text{iset}(\text{nfilter } P \text{ xs } 0) \}$   
**using** 4 1 2 *nfilter-holds-b* **by** *auto*  
**have** 6:  $\{ i. i \leq (\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k}) \wedge i \in \text{iset}(\text{nfilter } P \text{ xs } 0) \} =$   
 $\{ i. i \leq (\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k}) \wedge$   
 $i \in \{ (\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ j}) \mid j. j \leq \text{ilen}(\text{nfilter } P \text{ xs } 0) \} \}$   
**by** (*auto simp add: inth-and-iset*)  
**have** 7:  $\{ i. i \leq (\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k}) \wedge$   
 $i \in \{ (\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ j}) \mid j. j \leq \text{ilen}(\text{nfilter } P \text{ xs } 0) \} \} =$   
 $\{ (\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ j}) \mid j. j \leq k \}$   
**using** *assms 2* **by** (*auto simp add: nfilter-ilen,*  
*metis dual-order.antisym idx-nfilter idx-less-eq le-cases nfilter-ilen,*  
*metis idx-nfilter-less-eq nfilter-ilen*)  
**have** 8:  $k \leq \text{ilen } (\text{nfilter } P \text{ xs } 0)$   
**by** (*simp add: 2 assms(2) nfilter-ilen*)  
**have** 9:  $\{ (\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ j}) \mid j. j \leq k \} = \text{iset } (\text{prefix } k (\text{nfilter } P \text{ xs } 0))$   
**using** 8 2 **using** *prefix-iset* **by** *force*  
**have** 10:  $\text{iset } (\text{nfilter } P (\text{prefix } ((\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k}) ) \text{ xs}) 0) =$   
 $\{ i. i \leq ((\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k})) \wedge$   
 $P (\text{inth } (\text{prefix } ((\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k})) \text{ xs}) i) \}$   
**using** 4 1 *nfilter-prefix-iset* **by** *auto*  
**have** 11:  $\text{ilen}(\text{prefix } ((\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k})) \text{ xs}) = ((\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k}))$   
**using** 1 *prefix-ilen-good* **by** *blast*  
**have** 12:  $\{ i. i \leq ((\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k})) \wedge$   
 $P (\text{inth } (\text{prefix } ((\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k})) \text{ xs}) i) \} =$   
 $\{ i. i \leq \text{ilen}(\text{prefix } ((\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k})) \text{ xs}) \wedge$   
 $P (\text{inth } (\text{prefix } ((\text{inth } (\text{nfilter } P \text{ xs } 0) \text{ k})) \text{ xs}) i) \}$   
**using** 11 **by** *auto*



```

show ?thesis
using 10 12 3 5 6 7 9 by auto
qed

```

**lemma** *ifilter-nfilter-prefix-iset-eq-1:*

```

  assumes  $\exists x \in \text{iset } xs . P x$ 
            $k \leq \text{ilen } (\text{ifilter } P \text{ } xs)$ 
  shows    $\text{iset } (\text{prefix } k (\text{nfilter } P \text{ } xs \ 0)) =$ 
            $\text{iset } (\text{nfilter } P (\text{prefix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) ) \ xs) \ 0)$ 

```

**proof** –

```

  have 1:  $(\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \leq \text{ilen } xs$ 
    by (metis assms(1) assms(2) diff-zero ilen-gr-zero
        nfilter-ilen nfilter-upper-bound ordered-cancel-comm-monoid-diff-class.add-diff-inverse)

```

```

  have 2:  $\exists x \in \text{iset } xs . P x$ 
    using 1 assms(1) inth-iset by blast

```

```

  have 3:  $\{ i. i \leq \text{ilen}(\text{prefix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k)) \ xs) \wedge$ 
            $P (\text{inth } (\text{prefix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k)) \ xs) \ i) \} =$ 

```

```

            $\{ i. i \leq (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \wedge P (\text{inth } xs \ i) \}$ 

```

```

  using 1 by auto

```

```

  have 4:  $\exists x \in \text{iset}(\text{prefix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k)) \ xs). P x$ 
    using assms 1 2 nfilter-holds[of xs P 0]
    by (metis ilast-prefix le-refl nfilter-ilen nfilter-inth-n-zero inth-iset)

```

```

  have 5:  $\{ i. i \leq (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \wedge P (\text{inth } xs \ i) \} =$ 
            $\{ i. i \leq (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \wedge i \in \text{iset}(\text{nfilter } P \text{ } xs \ 0) \}$ 

```

```

    using 4 1 2 nfilter-holds-b by auto

```

```

  have 6:  $\{ i. i \leq (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \wedge i \in \text{iset}(\text{nfilter } P \text{ } xs \ 0) \} =$ 
            $\{ i. i \leq (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \wedge$ 
            $i \in \{ (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ j) \mid j. j \leq \text{ilen}(\text{nfilter } P \text{ } xs \ 0) \} \}$ 

```

```

    by (auto simp add: inth-and-iset)

```

```

  have 7:  $\{ i. i \leq (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \wedge$ 
            $i \in \{ (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ j) \mid j. j \leq \text{ilen}(\text{nfilter } P \text{ } xs \ 0) \} \} =$ 
            $\{ (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ j) \mid j. j \leq k \}$ 

```

```

    using assms 2 by (auto simp add: nfilter-ilen,
        metis dual-order.antisym idx-nfilter idx-less-eq le-cases nfilter-ilen,
        metis idx-nfilter-less-eq nfilter-ilen)

```

```

  have 8:  $k \leq \text{ilen } (\text{nfilter } P \text{ } xs \ 0)$ 

```

```

    by (simp add: 2 assms(2) nfilter-ilen)

```

```

  have 9:  $\{ (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ j) \mid j. j \leq k \} = \text{iset } (\text{prefix } k (\text{nfilter } P \text{ } xs \ 0))$ 

```

```

    using 8 2 using prefix-iset by force

```

```

  have 10:  $\text{iset } (\text{nfilter } P (\text{prefix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) ) \ xs) \ 0) =$ 
            $\{ i. i \leq ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k)) \wedge$ 
            $P (\text{inth } (\text{prefix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k)) \ xs) \ i) \}$ 

```

```

    using 4 1 nfilter-prefix-iset by auto

```

```

  have 11:  $\text{ilen}(\text{prefix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k)) \ xs) = ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k))$ 

```

```

    using 1 prefix-ilen-good by blast

```

```

  have 12:  $\{ i. i \leq ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k)) \wedge$ 
            $P (\text{inth } (\text{prefix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k)) \ xs) \ i) \} =$ 
            $\{ i. i \leq \text{ilen}(\text{prefix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k)) \ xs) \wedge$ 
            $P (\text{inth } (\text{prefix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k)) \ xs) \ i) \}$ 

```

using 11 by auto  
 show ?thesis  
 using 10 12 3 5 6 7 9 by auto  
 qed

lemma ifilter-nfilter-suffix-iset-eq:

assumes  $P \text{ (inth } xs \text{ ( (inth (nfilter P xs 0) k)) )}$   
 $k \leq \text{ilen (ifilter P xs)}$   
 shows  $\text{iset (nfilter P (suffix (inth (nfilter P xs 0) k) xs) (inth (nfilter P xs 0) k))} =$   
 $\text{iset (suffix k (nfilter P xs 0))}$

proof –

have 1:  $\text{inth (nfilter P xs 0) k} \leq \text{ilen xs}$   
 by (metis assms(1) assms(2) diff-zero ifilter-nfilter-prefix-ilen-0 ilen-gr-zero  
 nfilter-ilen nfilter-upper-bound ordered-cancel-comm-monoid-diff-class.add-diff-inverse)

have 2:  $\exists x \in \text{iset xs} . P x$

using 1 assms(1) inth-iset by blast

have 4:  $\exists x \in \text{iset (suffix ((inth (nfilter P xs 0) k)) xs)} . P x$

using 1 assms(1) inth-and-iset by force

have 10:  $\text{iset (nfilter P (suffix ((inth (nfilter P xs 0) k) xs) (inth (nfilter P xs 0) k))} =$   
 $\{ ( \text{inth (nfilter P xs 0) k} ) + i \mid i. i \leq \text{ilen xs} - ( \text{inth (nfilter P xs 0) k} ) \wedge$   
 $P ( \text{inth xs (i+(inth (nfilter P xs 0) k)) } ) \}$

using nfilter-suffix-iset-b[of ((inth (nfilter P xs 0) k)) xs P (inth (nfilter P xs 0) k)]

using 1 4 by blast

have 5:  $\{ ( \text{inth (nfilter P xs 0) k} ) + i \mid i. i \leq \text{ilen xs} - ( \text{inth (nfilter P xs 0) k} ) \wedge$   
 $P ( \text{inth xs (i+(inth (nfilter P xs 0) k)) } ) \}$

=

$\{ ( \text{inth (nfilter P xs 0) k} ) + i \mid i. i \leq \text{ilen xs} - ( \text{inth (nfilter P xs 0) k} ) \wedge$   
 $i + ( \text{inth (nfilter P xs 0) k} ) \in \text{iset (nfilter P xs 0)} \}$

using 4 1 2 nfilter-holds-b by auto

have 51:  $\{ ( \text{inth (nfilter P xs 0) k} ) + i \mid i. i \leq \text{ilen xs} - ( \text{inth (nfilter P xs 0) k} ) \wedge$   
 $i + ( \text{inth (nfilter P xs 0) k} ) \in \text{iset (nfilter P xs 0)} \} =$   
 $\{ ( \text{inth (nfilter P xs 0) k} ) + i \mid i.$   
 $( \text{inth (nfilter P xs 0) k} ) \leq i + ( \text{inth (nfilter P xs 0) k} ) \wedge$   
 $i + ( \text{inth (nfilter P xs 0) k} ) \leq \text{ilen xs} \wedge$   
 $i + ( \text{inth (nfilter P xs 0) k} ) \in \text{iset (nfilter P xs 0)} \}$

using 1 by auto

have 52:  $\{ ( \text{inth (nfilter P xs 0) k} ) + i \mid i.$   
 $( \text{inth (nfilter P xs 0) k} ) \leq i + ( \text{inth (nfilter P xs 0) k} ) \wedge$   
 $i + ( \text{inth (nfilter P xs 0) k} ) \leq \text{ilen xs} \wedge$   
 $i + ( \text{inth (nfilter P xs 0) k} ) \in \text{iset (nfilter P xs 0)} \} =$   
 $\{ j. ( \text{inth (nfilter P xs 0) k} ) \leq j \wedge j \leq \text{ilen xs} \wedge j \in \text{iset (nfilter P xs 0)} \}$

by (metis (no-types, lifting) add-diff-cancel-right' le0 le-add-diff-inverse  
 le-add-same-cancel2)

have 53:  $\{ j. ( \text{inth (nfilter P xs 0) k} ) \leq j \wedge j \leq \text{ilen xs} \wedge j \in \text{iset (nfilter P xs 0)} \} =$   
 $\{ j. ( \text{inth (nfilter P xs 0) k} ) \leq j \wedge$   
 $j \leq \text{ilen xs} \wedge j \in \{ ( \text{inth (nfilter P xs 0) jj} ) \mid jj. jj \leq \text{ilen (nfilter P xs 0)} \} \}$

**by** (*auto simp add: inth-and-iset*)  
**have** 54:  $\{ j. (inth (nfilter P xs 0) k) \leq j \wedge$   
 $j \leq ilen xs \wedge j \in \{ (inth (nfilter P xs 0) jj) \mid jj. jj \leq ilen (nfilter P xs 0) \} \} =$   
 $\{ (inth (nfilter P xs 0) j) \mid j. k \leq j \wedge j \leq ilen (nfilter P xs 0) \}$   
  
**using** *assms 2* **by** (*auto,*  
*metis dual-order.antisym idx-nfilter-less-eq le-cases nfilter-ilen,*  
*metis idx-nfilter-gr-eq,*  
*metis add-cancel-right-left nfilter-upper-bound*)  
**have** 8:  $k \leq ilen (nfilter P xs 0)$   
**by** (*simp add: 2 assms(2) nfilter-ilen*)  
**have** 9:  $\{ (inth (nfilter P xs 0) j) \mid j. k \leq j \wedge j \leq ilen (nfilter P xs 0) \} =$   
 $iset (suffix k (nfilter P xs 0))$   
**using** 8 2 **using** *suffix-iset-a* **by** *blast*  
**show** ?thesis  
**using** 10 5 51 52 53 54 9 **by** *simp*  
**qed**

**lemma** *ifilter-nfilter-suffix-iset-eq-1:*

**assumes**  $\exists x \in iset xs . P x$   
 $k \leq ilen (ifilter P xs)$   
**shows**  $iset (nfilter P (suffix (inth (nfilter P xs 0) k) xs) (inth (nfilter P xs 0) k)) =$   
 $iset (suffix k (nfilter P xs 0))$

**proof** –

**have** 1:  $(inth (nfilter P xs 0) k) \leq ilen xs$   
**by** (*metis assms(1) assms(2) diff-zero ilen-gr-zero*  
*nfilter-ilen nfilter-upper-bound ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)  
**have** 2:  $\exists x \in iset xs . P x$   
**using** *assms(1)* **by** *blast*  
**have** 4:  $\exists x \in iset(suffix ((inth (nfilter P xs 0) k)) xs). P x$   
**using** *nfilter-holds[of xs P 0]*  
**by** (*metis 1 2 assms(2) ifirst-suffix le0 nfilter-ilen nfilter-inth-n-zero inth-iset*)  
**have** 10:  $iset (nfilter P (suffix ((inth (nfilter P xs 0) k) xs) (inth (nfilter P xs 0) k))) =$   
 $\{ (inth (nfilter P xs 0) k) + i \mid i. i \leq ilen xs - (inth (nfilter P xs 0) k) \wedge$   
 $P (inth xs (i + (inth (nfilter P xs 0) k))) \}$

**using** *nfilter-suffix-iset-b[of (inth (nfilter P xs 0) k) xs P (inth (nfilter P xs 0) k)]*  
**using** 1 4 **by** *blast*

**have** 5:  $\{ (inth (nfilter P xs 0) k) + i \mid i. i \leq ilen xs - (inth (nfilter P xs 0) k) \wedge$   
 $P (inth xs (i + (inth (nfilter P xs 0) k))) \}$   
 $=$   
 $\{ (inth (nfilter P xs 0) k) + i \mid i. i \leq ilen xs - (inth (nfilter P xs 0) k) \wedge$   
 $i + (inth (nfilter P xs 0) k) \in iset(nfilter P xs 0) \}$

**using** 4 1 2 *nfilter-holds-b* **by** *auto*

**have** 51:  $\{ (inth (nfilter P xs 0) k) + i \mid i. i \leq ilen xs - (inth (nfilter P xs 0) k) \wedge$   
 $i + (inth (nfilter P xs 0) k) \in iset(nfilter P xs 0) \} =$   
 $\{ (inth (nfilter P xs 0) k) + i \mid i.$   
 $(inth (nfilter P xs 0) k) \leq i + (inth (nfilter P xs 0) k) \wedge$

$$i + (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \leq \text{ilen } xs \wedge \\ i + (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \in \text{iset}(\text{nfilter } P \text{ } xs \ 0)\}$$

**using** 1 **by** *auto*

**have** 52:  $\{ (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) + i \mid i. \\ (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \leq i + (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \wedge \\ i + (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \leq \text{ilen } xs \wedge \\ i + (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \in \text{iset}(\text{nfilter } P \text{ } xs \ 0) \} = \\ \{ j. (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \leq j \wedge j \leq \text{ilen } xs \wedge j \in \text{iset}(\text{nfilter } P \text{ } xs \ 0) \}$

**by** (*metis* (*no-types*, *lifting*) *add-diff-cancel-right'* *le0* *le-add-diff-inverse* *le-add-same-cancel2*)

**have** 53:  $\{ j. (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \leq j \wedge j \leq \text{ilen } xs \wedge j \in \text{iset}(\text{nfilter } P \text{ } xs \ 0) \} = \\ \{ j. (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \leq j \wedge \\ j \leq \text{ilen } xs \wedge j \in \{ (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ jj) \mid jj. jj \leq \text{ilen}(\text{nfilter } P \text{ } xs \ 0) \} \}$

**by** (*auto simp add: inth-and-iset*)

**have** 54:  $\{ j. (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \leq j \wedge \\ j \leq \text{ilen } xs \wedge j \in \{ (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ jj) \mid jj. jj \leq \text{ilen}(\text{nfilter } P \text{ } xs \ 0) \} \} = \\ \{ (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ j) \mid j. k \leq j \wedge j \leq \text{ilen } (\text{nfilter } P \text{ } xs \ 0) \}$

**using** *assms* 2 **by** (*auto*,  
*metis* *dual-order.antisym* *idx-nfilter-less-eq* *le-cases* *nfilter-ilen*,  
*simp add: 2 idx-nfilter-less-eq*,  
*metis add-cancel-right-left nfilter-upper-bound*)

**have** 8:  $k \leq \text{ilen } (\text{nfilter } P \text{ } xs \ 0)$

**by** (*simp add: 2 assms(2) nfilter-ilen*)

**have** 9:  $\{ (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ j) \mid j. k \leq j \wedge j \leq \text{ilen } (\text{nfilter } P \text{ } xs \ 0) \} = \\ \text{iset } (\text{suffix } k \ (\text{nfilter } P \text{ } xs \ 0))$

**using** 8 2 **using** *suffix-iset-a* **by** *blast*

**show** *?thesis*

**using** 10 5 51 52 53 54 9 **by** *simp*

**qed**

**lemma** *nfilter-nfilter-prefix*:

**assumes**  $P \ (\text{inth } xs \ (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k)) \wedge \\ k \leq \text{ilen } (\text{ifilter } P \text{ } xs)$

**shows**  $(\text{prefix } k \ (\text{nfilter } P \text{ } xs \ 0)) = \\ (\text{nfilter } P \ (\text{prefix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \ ) \ xs) \ 0)$

**using** *assms*

**by** (*meson ifilter-nfilter-prefix-idx-a ifilter-nfilter-prefix-idx-b* *ifilter-nfilter-prefix-iset-eq idx-iset-eq*)

**lemma** *nfilter-nfilter-prefix-1*:

**assumes**  $\exists x \in \text{iset } xs. P \ x \wedge \\ k \leq \text{ilen } (\text{ifilter } P \text{ } xs)$

**shows**  $(\text{prefix } k \ (\text{nfilter } P \text{ } xs \ 0)) = \\ (\text{nfilter } P \ (\text{prefix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) \ ) \ xs) \ 0)$

**using** *assms*

by (meson ifilter-nfilter-prefix-idx-a-1 ifilter-nfilter-prefix-idx-b-1  
 ifilter-nfilter-prefix-iset-eq-1 idx-iset-eq)

**lemma** *nfilter-nfilter-suffix*:

**assumes**  $P$  (*inth*  $xs$  ( (*inth* (*nfilter*  $P$   $xs$  0)  $k$ )) )  
 $k \leq \text{ilen}$  (*ifilter*  $P$   $xs$ )  
**shows** (*suffix*  $k$  (*nfilter*  $P$   $xs$  0)) =  
 (*nfilter*  $P$  (*suffix* ((*inth* (*nfilter*  $P$   $xs$  0)  $k$ ) )  $xs$ ) (*inth* (*nfilter*  $P$   $xs$  0)  $k$ ))

**using** *assms*

**by** (metis ifilter-nfilter-suffix-idx-a ifilter-nfilter-suffix-idx-b  
 ifilter-nfilter-suffix-iset-eq idx-iset-eq)

**lemma** *nfilter-nfilter-suffix-1*:

**assumes**  $\exists x \in \text{iset } xs . P x$   
 $k \leq \text{ilen}$  (*ifilter*  $P$   $xs$ )  
**shows** (*suffix*  $k$  (*nfilter*  $P$   $xs$  0)) =  
 (*nfilter*  $P$  (*suffix* ((*inth* (*nfilter*  $P$   $xs$  0)  $k$ ) )  $xs$ ) (*inth* (*nfilter*  $P$   $xs$  0)  $k$ ))

**proof** –

**have** 1:  $P$  (*inth*  $xs$  ( (*inth* (*nfilter*  $P$   $xs$  0)  $k$ )) )  
**by** (metis *assms*(1) *assms*(2) *nfilter-holds* *nfilter-ilen* *nfilter-inth-n-zero* *inth-iset*)  
**show** ?thesis **using** *assms* 1 *nfilter-nfilter-suffix* **by** *auto*  
**qed**

**lemma** *nfilter-imap-ifilter*:

**assumes** ( $\exists x \in \text{iset } xs . P x$ )  
**shows** *imap* ( $\lambda n. (\text{inth } xs \ n)$ ) (*nfilter*  $P$   $xs$  0) = *ifilter*  $P$   $xs$

**proof** –

**have** 1:  $\text{ilen}$  (*imap* ( $\lambda n. (\text{inth } xs \ n)$ ) (*nfilter*  $P$   $xs$  0)) =  $\text{ilen}$  (*ifilter*  $P$   $xs$ )  
**by** (*simp* *add*: *assms* *nfilter-ilen*)  
**have** 2:  $\bigwedge i. i \leq \text{ilen}$  (*imap* ( $\lambda n. (\text{inth } xs \ n)$ ) (*nfilter*  $P$   $xs$  0))  $\longrightarrow$   
 (*inth* (*imap* ( $\lambda n. (\text{inth } xs \ n)$ ) (*nfilter*  $P$   $xs$  0))  $i$ ) =  
 (*inth* (*ifilter*  $P$   $xs$ )  $i$ )  
**by** (metis *assms* *diff-zero* *ilen-imap* *inth-imap* *nfilter-ifilter*)  
**from** 1 2 **show** ?thesis  
**using** *interval-eq-inth-eq* **by** *blast*

**qed**

**lemma** *ifilter-nfilter-prefix*:

**assumes**  $P$  (*inth*  $xs$  ( (*inth* (*nfilter*  $P$   $xs$  0)  $k$ )) )  
 $k \leq \text{ilen}$  (*ifilter*  $P$   $xs$ )  
**shows** (*prefix*  $k$  (*ifilter*  $P$   $xs$  )) =  
 (*ifilter*  $P$  (*prefix* ((*inth* (*nfilter*  $P$   $xs$  0)  $k$ ) )  $xs$ ) )

**proof** –

**have** 1:  $\exists x \in \text{iset } xs . P x$   
**by** (metis *assms*(1) *assms*(2) *diff-zero* *ifilter-nfilter-prefix-ilen-0*)  
**have** 2: (*ifilter*  $P$   $xs$  ) = *imap* ( $\lambda s. \text{inth } xs \ s$ ) (*nfilter*  $P$   $xs$  0)  
**by** (*simp* *add*: 1 *nfilter-imap-ifilter*)  
**have** 3: (*prefix*  $k$  (*ifilter*  $P$   $xs$  )) =

```

      (prefix k (imap (λs. inth xs s) (nfilter P xs 0)))
    by (simp add: 2)
  have 4: (prefix k (imap (λs. inth xs s) (nfilter P xs 0))) =
    imap (λs. inth xs s) (prefix k (nfilter P xs 0))
    by (simp add: 1 assms(2) imap-prefix nfilter-ilen)
  have 5: ∃ x ∈ iset(prefix ((inth (nfilter P xs 0) k)) xs). P x
    by (metis 1 add.left-neutral assms(1) assms(2) ilast-prefix inth-and-iset
      nfilter-ilen nfilter-upper-bound order-refl)
  have 6: (ifilter P (prefix ((inth (nfilter P xs 0) k) ) xs) ) =
    imap (λs. (inth (prefix ((inth (nfilter P xs 0) k) ) xs) s))
      (nfilter P (prefix ((inth (nfilter P xs 0) k) ) xs) 0)
    by (simp add: 5 nfilter-imap-ifilter)
  have 7: imap (λs. inth xs s) (prefix k (nfilter P xs 0)) =
    imap (λs. inth xs s) (nfilter P (prefix ((inth (nfilter P xs 0) k) ) xs) 0)
    by (simp add: assms(1) assms(2) nfilter-nfilter-prefix)
  have 8: imap (λs. inth xs s) (nfilter P (prefix ((inth (nfilter P xs 0) k) ) xs) 0) =
    imap (λs. (inth (prefix ((inth (nfilter P xs 0) k) ) xs) s))
      (nfilter P (prefix ((inth (nfilter P xs 0) k) ) xs) 0)

    using 1 5 by simp
  show ?thesis
  by (simp add: 3 4 6 7 8)
qed

```

**lemma ifilter-nfilter-prefix-1:**

```

  assumes ∃ x ∈ iset xs . P x
    k ≤ ilen (ifilter P xs)
  shows (prefix k (ifilter P xs)) =
    (ifilter P (prefix ((inth (nfilter P xs 0) k) ) xs) )

```

**proof** –

```

  have 1: P (inth xs ( (inth (nfilter P xs 0) k) ) )
    by (metis assms(1) assms(2) nfilter-holds nfilter-ilen nfilter-inth-n-zero inth-iset)
  show ?thesis using assms 1 ifilter-nfilter-prefix by auto
qed

```

**lemma ifilter-nfilter-prefix-ilen:**

```

  assumes P (inth xs ( (inth (nfilter P xs 0) k) ) )
    k ≤ ilen (ifilter P xs)
  shows ilen(prefix k (ifilter P xs)) =
    ilen(ifilter P (prefix ((inth (nfilter P xs 0) k) ) xs))

```

**using** assms

**by** (simp add: ifilter-nfilter-prefix)

**lemma ifilter-nfilter-prefix-ilen-1:**

```

  assumes ∃ x ∈ iset xs . P x
    k ≤ ilen (ifilter P xs)
  shows ilen(prefix k (ifilter P xs)) =
    ilen(ifilter P (prefix ((inth (nfilter P xs 0) k) ) xs))

```

**using** assms

**by** (simp add: ifilter-nfilter-prefix-1)

**lemma** *ifilter-nfilter-prefix-inth*:

**assumes**  $P \text{ (inth } xs \text{ ( (inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k)) \text{ ) )}$   
 $k \leq \text{ilen (ifilter } P \text{ } xs)$   
 $j \leq \text{ilen(prefix } k \text{ (ifilter } P \text{ } xs))}$   
**shows**  $\text{inth (prefix } k \text{ (ifilter } P \text{ } xs)) } j =$   
 $\text{inth (ifilter } P \text{ (prefix ((inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ ) } xs)) } j$

**using** *assms*

**by** (*simp add: ifilter-nfilter-prefix*)

**lemma** *ifilter-nfilter-prefix-inth-1*:

**assumes**  $\exists x \in \text{iset } xs . P \text{ } x$   
 $k \leq \text{ilen (ifilter } P \text{ } xs)$   
 $j \leq \text{ilen(prefix } k \text{ (ifilter } P \text{ } xs))}$   
**shows**  $\text{inth (prefix } k \text{ (ifilter } P \text{ } xs)) } j =$   
 $\text{inth (ifilter } P \text{ (prefix ((inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ ) } xs)) } j$

**using** *assms*

**by** (*simp add: ifilter-nfilter-prefix-1*)

**lemma** *nfilter-imap-shift*:

**assumes**  $\exists x \in \text{iset } xs . P \text{ } x$   
**shows**  $\text{imap } (\lambda s. \text{inth } xs \text{ (s+n)}) \text{ (nfilter } P \text{ } xs \text{ } 0) =$   
 $\text{imap } (\lambda s. \text{inth } xs \text{ } s) \text{ (nfilter } P \text{ } xs \text{ } n)$

**proof** –

**have** 1:  $\text{ilen (imap } (\lambda s. \text{inth } xs \text{ (s+n)}) \text{ (nfilter } P \text{ } xs \text{ } 0))} =$   
 $\text{ilen (imap } (\lambda s. \text{inth } xs \text{ } s) \text{ (nfilter } P \text{ } xs \text{ } n))}$

**by** (*metis assms(1) ilen-imap nfilter-ilen-n-zero*)

**have** 2:  $\bigwedge i. \text{inth (imap } (\lambda s. \text{inth } xs \text{ (s+n)}) \text{ (nfilter } P \text{ } xs \text{ } 0)) } i =$   
 $(\text{inth } xs \text{ ( (inth (nfilter } P \text{ } xs \text{ } 0) \text{ } i) + n))$

**by** (*simp add: inth-imap*)

**have** 3:  $\bigwedge i. \text{inth (imap } (\lambda s. \text{inth } xs \text{ } s) \text{ (nfilter } P \text{ } xs \text{ } n)) } i =$   
 $(\text{inth } xs \text{ ( (inth (nfilter } P \text{ } xs \text{ } n) \text{ } i))}$

**by** (*simp add: inth-imap*)

**have** 4:  $\bigwedge i. (\text{inth (nfilter } P \text{ } xs \text{ } 0) \text{ } i) + n = (\text{inth (nfilter } P \text{ } xs \text{ } n) \text{ } i)$

**by** (*metis assms(1) inth-imap nfilter-n-zero*)

**show** *?thesis*

**by** (*metis 1 2 3 4 interval-eq-inth-eq*)

**qed**

**lemma** *nfilter-imap-shift-suffix*:

**assumes**  $\exists x \in \text{iset}(\text{suffix ((inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k)) \text{ } xs) . P \text{ } x$

**shows**  $\text{imap } (\lambda s. \text{inth } xs \text{ (s+(inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k)))$   
 $(\text{nfilter } P \text{ (suffix ((inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ ) } xs) \text{ } 0) =$   
 $\text{imap } (\lambda s. \text{inth } xs \text{ } s)$   
 $(\text{nfilter } P \text{ (suffix ((inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ ) } xs) (inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k))$

**proof** –

**have** 1:  $\text{ilen (imap } (\lambda s. \text{inth } xs \text{ (s+(inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k)))$   
 $(\text{nfilter } P \text{ (suffix ((inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ ) } xs) \text{ } 0)) =$   
 $\text{ilen (imap } (\lambda s. \text{inth } xs \text{ } s)$   
 $(\text{nfilter } P \text{ (suffix ((inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ ) } xs) (inth (nfilter } P \text{ } xs \text{ } 0) \text{ } k)))$

**by** (*metis* *assms* *ilen-imap* *nfilter-ilen-n-zero*)  
**have** 2:  $\bigwedge i. \text{inth} (\text{imap} (\lambda s. \text{inth} xs (s + (\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k))))$   
 $(\text{nfilter } P (\text{suffix} ((\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ } ) \text{ } xs) \text{ } 0)) \text{ } i =$   
 $\text{inth } xs$   
 $((\text{inth} (\text{nfilter } P (\text{suffix} ((\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ } ) \text{ } xs) \text{ } 0) \text{ } i) + (\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k))$

**using** *inth-imap* **by** *blast*  
**have** 3:  $\bigwedge i.$   
 $\text{inth} (\text{imap} (\lambda s. \text{inth} xs \text{ } s)$   
 $(\text{nfilter } P (\text{suffix} ((\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ } ) \text{ } xs)$   
 $(\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k))) \text{ } i =$   
 $\text{inth } xs (\text{inth} (\text{nfilter } P (\text{suffix} ((\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ } ) \text{ } xs)$   
 $(\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k)) \text{ } i)$

**using** *inth-imap* **by** *blast*  
**have** 4:  $\bigwedge i. (\text{inth} (\text{nfilter } P (\text{suffix} ((\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ } ) \text{ } xs) \text{ } 0) \text{ } i)$   
 $+ (\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k) =$   
 $(\text{inth} (\text{nfilter } P (\text{suffix} ((\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ } ) \text{ } xs) (\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k)) \text{ } i)$   
**by** (*metis* *assms* *inth-imap* *nfilter-n-zero*)  
**show** *?thesis*  
**by** (*metis* 1 2 3 4 *interval-eq-inth-eq*)  
**qed**

**lemma** *ifilter-nfilter-suffix*:

**assumes**  $P (\text{inth } xs \text{ } ( (\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k)) \text{ } )$   
 $k \leq \text{ilen} (\text{ifilter } P \text{ } xs)$   
**shows**  $(\text{suffix } k (\text{ifilter } P \text{ } xs)) =$   
 $(\text{ifilter } P (\text{suffix} ((\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ } ) \text{ } xs) \text{ } )$

**proof** –

**have** 1:  $\exists x \in \text{iset } xs . P \text{ } x$   
**by** (*metis* *assms*(1) *assms*(2) *diff-zero* *ifilter-nfilter-prefix-ilen-0*)  
**have** 2:  $(\text{ifilter } P \text{ } xs) = \text{imap} (\lambda s. \text{inth } xs \text{ } s) (\text{nfilter } P \text{ } xs \text{ } 0)$   
**by** (*simp* *add*: 1 *nfilter-imap-ifilter*)  
**have** 3:  $(\text{suffix } k (\text{ifilter } P \text{ } xs)) =$   
 $(\text{suffix } k (\text{imap} (\lambda s. \text{inth } xs \text{ } s) (\text{nfilter } P \text{ } xs \text{ } 0)))$   
**by** (*simp* *add*: 2)  
**have** 4:  $(\text{suffix } k (\text{imap} (\lambda s. \text{inth } xs \text{ } s) (\text{nfilter } P \text{ } xs \text{ } 0))) =$   
 $\text{imap} (\lambda s. \text{inth } xs \text{ } s) (\text{suffix } k (\text{nfilter } P \text{ } xs \text{ } 0))$   
**by** (*simp* *add*: 1 *assms*(2) *imap-suffix* *nfilter-ilen*)  
**have** 5:  $\exists x \in \text{iset}(\text{suffix} ((\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k)) \text{ } xs). P \text{ } x$   
**by** (*metis* 1 *add-cancel-right-left* *assms*(1) *assms*(2) *ifirst-suffix*  
*ilen-gr-zero* *inth-and-iset* *nfilter-ilen* *nfilter-upper-bound*)  
**have** 6:  $(\text{ifilter } P (\text{suffix} ((\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ } ) \text{ } xs) \text{ } ) =$   
 $\text{imap} (\lambda s. \text{inth} (\text{suffix} ((\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ } ) \text{ } xs) \text{ } s)$   
 $(\text{nfilter } P (\text{suffix} ((\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ } ) \text{ } xs) \text{ } 0)$   
**by** (*simp* *add*: 5 *nfilter-imap-ifilter*)  
**have** 7:  $\text{imap} (\lambda s. \text{inth} (\text{suffix} ((\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ } ) \text{ } xs) \text{ } s)$   
 $(\text{nfilter } P (\text{suffix} ((\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k) \text{ } ) \text{ } xs) \text{ } 0) =$   
 $\text{imap} (\lambda s. \text{inth } xs \text{ } (s + (\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } k)))$



$(\text{nfilter } P (\text{suffix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) ) \ xs) \ 0)$

**using** 1 5 **by** (*simp add: add.commute*)

**have** 8:  $\text{imap } (\lambda s. \text{inth } xs \ (s + (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k)))$   
 $(\text{nfilter } P (\text{suffix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) ) \ xs) \ 0) =$   
 $\text{imap } (\lambda s. \text{inth } xs \ s)$   
 $(\text{nfilter } P (\text{suffix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) ) \ xs) (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k))$

**using** 5 *nfilter-imap-shift-suffix* **by** *metis*

**have** 9:  $\text{imap } (\lambda s. \text{inth } xs \ s)$   
 $(\text{nfilter } P (\text{suffix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) ) \ xs) (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k)) =$   
 $\text{imap } (\lambda s. \text{inth } xs \ s)$   
 $(\text{suffix } k (\text{nfilter } P \text{ } xs \ 0))$

**by** (*simp add: assms(1) assms(2) nfilter-nfilter-suffix*)

**show** ?thesis

**by** (*simp add: 3 4 6 7 8 9*)

**qed**

**lemma** *ifilter-nfilter-suffix-1:*

**assumes**  $\exists x \in \text{iset } xs. P \ x$

$k \leq \text{ilen } (\text{ifilter } P \text{ } xs)$

**shows**  $(\text{suffix } k (\text{ifilter } P \text{ } xs) ) =$   
 $(\text{ifilter } P (\text{suffix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) ) \ xs) )$

**proof** –

**have** 1:  $\exists x \in \text{iset } xs. P \ x$

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

**have** 2:  $(\text{ifilter } P \text{ } xs) = \text{imap } (\lambda s. \text{inth } xs \ s) (\text{nfilter } P \text{ } xs \ 0)$

**by** (*simp add: 1 nfilter-imap-ifilter*)

**have** 3:  $(\text{suffix } k (\text{ifilter } P \text{ } xs) ) =$   
 $(\text{suffix } k (\text{imap } (\lambda s. \text{inth } xs \ s) (\text{nfilter } P \text{ } xs \ 0)))$

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

**have** 4:  $(\text{suffix } k (\text{imap } (\lambda s. \text{inth } xs \ s) (\text{nfilter } P \text{ } xs \ 0))) =$   
 $\text{imap } (\lambda s. \text{inth } xs \ s) (\text{suffix } k (\text{nfilter } P \text{ } xs \ 0))$

**by** (*simp add: 1 assms(2) imap-suffix nfilter-ilen*)

**have** 5:  $\exists x \in \text{iset}(\text{suffix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k)) \ xs). P \ x$

**using** *assms inth-iset 2 inth-suffix nfilter-holds[of xs P 0]*

**by** (*metis add.right-neutral diff-zero ilen-imap le0*)

**have** 6:  $(\text{ifilter } P (\text{suffix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) ) \ xs) ) =$   
 $\text{imap } (\lambda s. \text{inth } (\text{suffix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) ) \ xs) \ s)$   
 $(\text{nfilter } P (\text{suffix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) ) \ xs) \ 0)$

**by** (*simp add: 5 nfilter-imap-ifilter*)

**have** 7:  $\text{imap } (\lambda s. \text{inth } (\text{suffix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) ) \ xs) \ s)$   
 $(\text{nfilter } P (\text{suffix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) ) \ xs) \ 0) =$   
 $\text{imap } (\lambda s. \text{inth } xs \ (s + (\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k)))$   
 $(\text{nfilter } P (\text{suffix } ((\text{inth } (\text{nfilter } P \text{ } xs \ 0) \ k) ) \ xs) \ 0)$

**using** 1 5

**by** (*metis* *add.right-neutral inth-suffix suffix-suffix le0*)  
**have** 8: *imap* ( $\lambda s. \text{inth } xs \ (s + (\text{inth } (\text{nfilter } P \ xs \ 0) \ k))$ )  
 $(\text{nfilter } P \ (\text{suffix } ((\text{inth } (\text{nfilter } P \ xs \ 0) \ k) \ ) \ xs) \ 0) =$   
 $\text{imap } (\lambda s. \text{inth } xs \ s)$   
 $(\text{nfilter } P \ (\text{suffix } ((\text{inth } (\text{nfilter } P \ xs \ 0) \ k) \ ) \ xs) \ (\text{inth } (\text{nfilter } P \ xs \ 0) \ k))$   
  
**using** 5 *nfilter-imap-shift-suffix* **by** *metis*  
**have** 9: *imap* ( $\lambda s. \text{inth } xs \ s$ )  
 $(\text{nfilter } P \ (\text{suffix } ((\text{inth } (\text{nfilter } P \ xs \ 0) \ k) \ ) \ xs) \ (\text{inth } (\text{nfilter } P \ xs \ 0) \ k)) =$   
 $\text{imap } (\lambda s. \text{inth } xs \ s)$   
 $(\text{suffix } k \ (\text{nfilter } P \ xs \ 0))$   
  
**by** (*simp* *add: assms(1) assms(2) nfilter-nfilter-suffix-1*)  
**show** ?thesis  
**by** (*simp* *add: 3 4 6 7 8 9*)  
**qed**

**lemma** *ifilter-nfilter-suffix-ilen:*  
**assumes**  $P \ (\text{inth } xs \ ( (\text{inth } (\text{nfilter } P \ xs \ 0) \ k) \ )$   
 $k \leq \text{ilen } (\text{ifilter } P \ xs)$   
**shows**  $\text{ilen}(\text{suffix } k \ (\text{ifilter } P \ xs)) =$   
 $\text{ilen}(\text{ifilter } P \ (\text{suffix } ((\text{inth } (\text{nfilter } P \ xs \ 0) \ k) \ ) \ xs))$   
**using** *assms*  
**by** (*simp* *add: ifilter-nfilter-suffix*)

**lemma** *ifilter-nfilter-suffix-ilen-1:*  
**assumes**  $\exists x \in \text{iset } xs. P \ x$   
 $k \leq \text{ilen } (\text{ifilter } P \ xs)$   
**shows**  $\text{ilen}(\text{suffix } k \ (\text{ifilter } P \ xs)) =$   
 $\text{ilen}(\text{ifilter } P \ (\text{suffix } ((\text{inth } (\text{nfilter } P \ xs \ 0) \ k) \ ) \ xs))$   
**using** *assms*  
**by** (*simp* *add: ifilter-nfilter-suffix-1*)

**lemma** *ifilter-nfilter-suffix-inth:*  
**assumes**  $P \ (\text{inth } xs \ ( (\text{inth } (\text{nfilter } P \ xs \ 0) \ k) \ )$   
 $k \leq \text{ilen } (\text{ifilter } P \ xs)$   
 $j \leq \text{ilen}(\text{suffix } k \ (\text{ifilter } P \ xs))$   
**shows**  $\text{inth } (\text{suffix } k \ (\text{ifilter } P \ xs)) \ j =$   
 $\text{inth } (\text{ifilter } P \ (\text{suffix } ((\text{inth } (\text{nfilter } P \ xs \ 0) \ k) \ ) \ xs)) \ j$   
**using** *assms*  
**by** (*simp* *add: ifilter-nfilter-suffix*)

**lemma** *ifilter-nfilter-suffix-inth-1:*  
**assumes**  $\exists x \in \text{iset } xs. P \ x$   
 $k \leq \text{ilen } (\text{ifilter } P \ xs)$   
 $j \leq \text{ilen}(\text{suffix } k \ (\text{ifilter } P \ xs))$   
**shows**  $\text{inth } (\text{suffix } k \ (\text{ifilter } P \ xs)) \ j =$   
 $\text{inth } (\text{ifilter } P \ (\text{suffix } ((\text{inth } (\text{nfilter } P \ xs \ 0) \ k) \ ) \ xs)) \ j$   
**using** *assms*  
**by** (*simp* *add: ifilter-nfilter-suffix-1*)

```

lemma nfilter-ilast:
  assumes  $P \text{ (ilast (prefix k xs))}$ 
     $k \leq \text{ilen xs}$ 
  shows  $(\text{inth xs } ((\text{ilast (nfilter P (prefix k xs) n)) - n)) = (\text{ilast (prefix k xs)})$ 
using assms
proof (induction xs arbitrary: n k)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
  proof (auto simp add: min.absorb1 split:nat.split)
    fix x2
    fix x
    assume a0:  $(\bigwedge k n. P \text{ (inth xs k)} \implies$ 
       $k \leq \text{ilen xs} \implies$ 
       $\text{inth xs } (\text{inth (nfilter P (prefix k xs) n) (ilen (nfilter P (prefix k xs) n)) - n) =$ 
       $\text{inth xs k})$ 
    assume a1:  $P \text{ (inth xs x2)}$ 
    assume a2:  $x \in \text{iset (prefix x2 xs)}$ 
    assume a3:  $P x$ 
    assume a4:  $k = \text{Suc x2}$ 
    assume a5:  $x2 \leq \text{ilen xs}$ 
    assume a6:  $\text{inth (nfilter P (prefix x2 xs) (Suc n)) (ilen (nfilter P (prefix x2 xs) (Suc n)))}$ 
       $\leq n$ 
    show  $x1a = \text{inth xs x2}$ 
    using a0 a1 a2 a3 a6 by (metis le-refl nfilter-inth-n-zero-a not-less-eq-eq)
  next
    fix x2
    fix x
    fix x2a
    assume b0:  $(\bigwedge k n. P \text{ (inth xs k)} \implies$ 
       $k \leq \text{ilen xs} \implies$ 
       $\text{inth xs } (\text{inth (nfilter P (prefix k xs) n) (ilen (nfilter P (prefix k xs) n)) - n) =$ 
       $\text{inth xs k})$ 
    assume b1:  $P \text{ (inth xs x2)}$ 
    assume b2:  $x \in \text{iset (prefix x2 xs)}$ 
    assume b3:  $P x$ 
    assume b4:  $k = \text{Suc x2}$ 
    assume b5:  $x2 \leq \text{ilen xs}$ 
    assume b6:  $\text{inth (nfilter P (prefix x2 xs) (Suc n))}$ 
       $(\text{ilen (nfilter P (prefix x2 xs) (Suc n))) - n = \text{Suc x2a}$ 
    show  $\text{inth xs x2a} = \text{inth xs x2}$ 
    using b0 b1 b2 b3 b4 b5 b6
    by (metis Suc-eq-plus1 add-diff-cancel-left' diff-diff-left plus-1-eq-Suc)
  next
    fix x2
    assume c0:  $(\bigwedge k n. P \text{ (inth xs k)} \implies$ 
       $k \leq \text{ilen xs} \implies$ 

```

```

      inth xs (inth (nfilter P (prefix k xs) n) (ilen (nfilter P (prefix k xs) n)) - n) =
        inth xs k)
assume c1: P (inth xs x2)
assume c2:  $\forall x \in \text{iset} \text{ (prefix } x2 \text{ xs). } \neg P \ x$ 
assume c3:  $k = \text{Suc } x2$ 
assume c4:  $x2 \leq \text{ilen } xs$ 
show x1a = inth xs x2
using c0 c1 c2 c3 c4 ilast-prefix inth-iset by fastforce
qed
qed

```

```

lemma nfilter-ifirst:
assumes P (ifirst (suffix k xs))
       $k \leq \text{ilen } xs$ 
shows ifirst (nfilter P (suffix k xs) n) = n
using assms
proof (induction xs arbitrary: k n)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case
      by (auto split:nat.split)
qed

```

```

lemma ifilter-ilast:
assumes P (ilast (prefix n xs))
       $n \leq \text{ilen } xs$ 
shows ilast (ifilter P (prefix n xs)) = (ilast (prefix 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 (INil x1)
      then show ?thesis
      by simp
      next
      case (ICons x21 x22)
      then show ?thesis using Suc inth-iset by (auto, force)
      qed
    qed

```

```

lemma ifilter-ifirst:
assumes P (ifirst (suffix n xs))
shows ifirst (ifilter P (suffix n xs)) = (ifirst (suffix n xs))
using assms

```

```

proof (induction xs arbitrary: n)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)
then show ?case by (auto split:nat.split)
qed

```

```

lemma ifilter-inth-aa:
  assumes ( $\exists x \in \text{iset } xs. P\ x$ )
     $n \leq \text{ilen } (\text{ifilter } P\ xs)$ 
  shows  $P\ (\text{inth } (\text{ifilter } P\ xs)\ n)$ 
using assms iset-ifilter inth-iset[of n (ifilter P xs)] by simp

```

```

lemma ifilter-ilen-zero-conv-a:
  assumes ( $\exists x \in \text{iset } xs. P\ x$ )
     $\text{ilen } (\text{ifilter } P\ xs) = 0$ 
  shows ( $\exists k \leq \text{ilen } xs. P\ (\text{inth } xs\ k) \wedge$ 
     $(\forall j \leq \text{ilen } xs. j \neq k \longrightarrow \neg P\ (\text{inth } xs\ j)))$ 
proof –
  have 1:  $P\ (\text{inth } xs\ (\text{inth } (\text{nfilter } P\ xs\ 0)\ 0))$ 
    by (metis nfilter-imap-ifilter assms(1) assms(2) ifilter-inth-aa inth-imap
      le-numeral-extra(3))
  have 2:  $(\text{inth } (\text{nfilter } P\ xs\ 0)\ 0) \leq \text{ilen } xs$ 
    by (metis assms(1) diff-zero ilen-gr-zero nfilter-upper-bound
      ordered-cancel-comm-monoid-diff-class.add-diff-inverse)
  have 3:  $(\forall j \leq \text{ilen } xs. j \neq (\text{inth } (\text{nfilter } P\ xs\ 0)\ 0) \longrightarrow \neg P\ (\text{inth } xs\ j))$ 
    using assms nfilter-not-after[of xs P ] nfilter-not-before[of xs P]
    by (metis linorder-neqE-nat nfilter-ilen)
  show ?thesis
using 1 2 3 by blast
qed

```

```

lemma ifilter-ilen-zero-conv-c:
  ( $\exists k \leq \text{ilen } xs. P\ (\text{inth } xs\ k) \wedge$ 
     $(\forall j \leq \text{ilen } xs. j \neq k \longrightarrow \neg P\ (\text{inth } xs\ j))) =$ 
  ( $\exists k \leq \text{ilen } xs. P\ (\text{inth } xs\ k) \wedge$ 
     $(\forall j \leq \text{ilen } xs. j < k \vee k < j \longrightarrow \neg P\ (\text{inth } xs\ j)))$ 

```

```

using antisym-conv3 by auto

```

```

lemma ifilter-ilen-zero-conv-d:
  ( $\exists k \leq \text{ilen } xs. P\ (\text{inth } xs\ k) \wedge$ 
     $(\forall j \leq \text{ilen } xs. j < k \vee k < j \longrightarrow \neg P\ (\text{inth } xs\ j))) =$ 
  ( $\exists k \leq \text{ilen } xs. P\ (\text{inth } xs\ k) \wedge$ 
     $(\forall j. j < k \longrightarrow \neg P\ (\text{inth } xs\ j)) \wedge$ 
     $(\forall j \leq \text{ilen } xs. k < j \longrightarrow \neg P\ (\text{inth } xs\ j))$ 
  )

```

by auto

**lemma** *ifilter-ilen-zero-conv-b*:

**assumes**  $(\exists k \leq \text{ilen } xs. P (\text{inth } xs k) \wedge$   
 $(\forall j \leq \text{ilen } xs. j \neq k \longrightarrow \neg P (\text{inth } xs j)))$

**shows**  $(\exists x \in \text{iset } xs. P x) \wedge \text{ilen } (\text{ifilter } P xs) = 0$

**proof** –

**have** 1:  $(\exists x \in \text{iset } xs. P x)$

**using** *assms inth-iset* **by** auto

**obtain** *k* **where** 2:  $k \leq \text{ilen } xs \wedge P (\text{inth } xs k) \wedge$   
 $(\forall j \leq \text{ilen } xs. j \neq k \longrightarrow \neg P (\text{inth } xs j))$

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

**have** 3:  $\text{ilen } (\text{ifilter } P xs) = 0$

**using** 1 2

**proof** (*induct xs arbitrary: k*)

**case** (*INil x*)

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

**next**

**case** (*ICons x1a xs*)

**then show** ?*case*

**proof** (*cases k*)

**case** 0

**then show** ?*thesis* **using** *ICons*

**by** (*metis Suc-le-mono ifilter-ilen-d inth-Suc inth-and-iset ilen.simps(2)*  
*le0 not-less-eq-eq order-refl plus-1-eq-Suc*)

**next**

**case** (*Suc nat*)

**then show** ?*thesis* **using** *ICons.hyps ICons.prem*s

**by** (*metis Suc-le-mono Zero-not-Suc add-diff-cancel-left' ifilter-ilen-c*  
*ilen-gr-zero inth-Suc inth-zero iset-IConsD ilen.simps(2) plus-1-eq-Suc*)

**qed**

**qed**

**show** ?*thesis*

**using** 1 3 **by** *blast*

**qed**

**lemma** *ifilter-ilen-zero-conv*:

$((\exists x \in \text{iset } xs. P x) \wedge \text{ilen } (\text{ifilter } P xs) = 0) =$

$(\exists k \leq \text{ilen } xs. P (\text{inth } xs k) \wedge$   
 $(\forall j \leq \text{ilen } xs. j \neq k \longrightarrow \neg P (\text{inth } xs j)))$

**using** *ifilter-ilen-zero-conv-a[of xs P] ifilter-ilen-zero-conv-b[of xs P]*

**by** *blast*

**lemma** *ifilter-ilen-zero-conv-1*:

$((\exists x \in \text{iset } xs. P x) \wedge \text{ilen } (\text{ifilter } P xs) = 0) =$

$(\exists k \leq \text{ilen } xs. P (\text{inth } xs k) \wedge$   
 $(\forall j \leq \text{ilen } xs. j < k \vee k < j \longrightarrow \neg P (\text{inth } xs j)))$

**proof** –

**have** 1:  $((\exists x \in \text{iset } xs. P x) \wedge \text{ilen } (\text{ifilter } P xs) = 0) =$

```

      (∃ k ≤ ilen xs. P (inth xs k) ∧
        (∀ j ≤ ilen xs. j ≠ k → ¬ P (inth xs j)))
    by (simp add: ifilter-ilen-zero-conv)
  have 2: (∃ k ≤ ilen xs. P (inth xs k) ∧
    (∀ j ≤ ilen xs. j ≠ k → ¬ P (inth xs j))) =
    (∃ k ≤ ilen xs. P (inth xs k) ∧
      (∀ j ≤ ilen xs. j < k ∨ k < j → ¬ P (inth xs j)))
    by fastforce
  show ?thesis by (simp add: 1 2)
qed

```

**lemma** *ifilter-ilen-zero-conv-2*:

```

  ((∃ x ∈ iset xs. P x) ∧ ilen (ifilter P xs) = 0) =
  (∃ k ≤ ilen xs. P (inth xs k) ∧
    (∀ j. j < k → ¬ P (inth xs j)) ∧
    (∀ j ≤ ilen xs. k < j → ¬ P (inth xs j)))
)
proof –
  have 1: ((∃ x ∈ iset xs. P x) ∧ ilen (ifilter P xs) = 0) =
    (∃ k ≤ ilen xs. P (inth xs k) ∧
      (∀ j ≤ ilen xs. j < k ∨ k < j → ¬ P (inth xs j)))
    by (simp add: ifilter-ilen-zero-conv-1)
  have 2: (∃ k ≤ ilen xs. P (inth xs k) ∧
    (∀ j ≤ ilen xs. j < k ∨ k < j → ¬ P (inth xs j))) =
    (∃ k ≤ ilen xs. P (inth xs k) ∧
      (∀ j. j < k → ¬ P (inth xs j)) ∧
      (∀ j ≤ ilen xs. k < j → ¬ P (inth xs j)))
    )
  using dual-order.strict-trans1 by auto
  from 1 2 show ?thesis by auto
qed

```

**lemma** *ifilter-suffixes-imap-help-0*:

**assumes**  $j \leq \text{inth}(\text{nfilter } P \text{ xs } 0) \ 0$

$\exists x \in \text{iset xs. } P \ x$

**shows**  $(\text{ifilter } P (\text{suffix } (\text{inth}(\text{nfilter } P \text{ xs } 0) \ 0) \text{ xs})) = (\text{ifilter } P (\text{suffix } j \text{ xs}))$

**using** *assms*

**proof** (*induct xs arbitrary:j*)

**case** (*INil x*)

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

**next**

**case** (*ICons x1a xs*)

**then show** ?*case*

**proof** (*cases*  $\exists a \in \text{iset xs. } P \ a$ )

**case** *True*

**then show** ?*thesis*

**proof** (*cases*  $P \ x1a$ )

**case** *True*

**then show** ?*thesis* **using** *ICons.prem*s

**by** (*metis le-zero-eq nfilter-inth-ICons*)

```

next
case False
then show ?thesis
proof (cases j)
case 0
then show ?thesis using ICons False  $\exists a \in \text{iset } xs. P a$ 
by (metis ifilter-nfilter-suffix-1 suffix-zero le0)
next
case (Suc nat)
then show ?thesis using ICons False  $\exists a \in \text{iset } xs. P a$  by (auto split: nat.split)
(metis One-nat-def add-diff-cancel-left' le0 nfilter-inth-n-zero plus-1-eq-Suc)
qed
qed
next
case False
then show ?thesis
using ICons by auto
qed
qed

```

```

lemma ifilter-suffixes-imap-help-0-a:
assumes  $j \leq \text{inth}(\text{nfilter } P (\text{suffixes } xs) 0) 0$ 
 $\exists x \in \text{iset } (\text{suffixes } xs). P x$ 
shows  $(\text{ifilter } P (\text{suffixes } (\text{suffix } (\text{inth}(\text{nfilter } P (\text{suffixes } xs) 0) 0) xs))) =$ 
 $(\text{ifilter } P (\text{suffixes } (\text{suffix } j xs)))$ 
proof -
have 1:  $(\text{suffix } (\text{inth}(\text{nfilter } P (\text{suffixes } xs) 0) 0) (\text{suffixes } xs)) =$ 
 $(\text{suffixes } (\text{suffix } (\text{inth}(\text{nfilter } P (\text{suffixes } xs) 0) 0) xs))$ 
by (metis assms(2) diff-zero ilen-gr-zero nfilter-upper-bound
ordered-cancel-comm-monoid-diff-class.add-diff-inverse suffix-suffixes)
have 2:  $\text{inth}(\text{nfilter } P (\text{suffixes } xs) 0) 0 \leq \text{ilen } (\text{suffixes } xs)$ 
by (metis add.left-neutral assms(2) ilen-gr-zero nfilter-upper-bound)
have 3:  $(\text{suffix } j (\text{suffixes } xs)) = (\text{suffixes } (\text{suffix } j xs))$ 
using 2 suffix-suffixes assms le-trans by blast
show ?thesis
using 1 3 assms ifilter-suffixes-imap-help-0 by fastforce
qed

```

```

lemma ifilter-suffixes-imap-help-1:
assumes  $j \leq \text{inth}(\text{nfilter } P xs 0) 1$ 
 $0 < \text{ilen}(\text{ifilter } P xs)$ 
 $\text{inth}(\text{nfilter } P xs 0) 0 < j$ 
 $\exists x \in \text{iset } xs. P x$ 
shows  $(\text{ifilter } P (\text{suffix } (\text{inth}(\text{nfilter } P xs 0) 1) xs)) = (\text{ifilter } P (\text{suffix } j xs))$ 
using assms
proof (induct xs arbitrary: j)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)

```



```

then show ?case
proof (cases  $\exists a \in \text{iset } xs. P a$ )
case True
then show ?thesis
proof (cases  $P x1a$ )
case True
then show ?thesis
proof (cases  $j$ )
case 0
then show ?thesis
using ICons.prems by blast
next
case (Suc nat)
then show ?thesis using ICons True  $\langle \exists a \in \text{iset } xs. P a \rangle$  ifilter-suffixes- $\text{imap-help-0}$ [of -  $P xs$ ]
nfilter-inth-n-zero[of  $xs P$ ]
proof auto
fix  $x :: 'a$ 
assume  $j = \text{Suc nat}$ 
assume  $a1: \text{Suc nat} \leq \text{inth } (nfilter P xs (\text{Suc } 0)) 0$ 
assume  $a2: \bigwedge k n. k \leq \text{ilen } (nfilter P xs n) \implies$ 
 $\text{inth } (nfilter P xs n) k - n = \text{inth } (nfilter P xs 0) k$ 
assume  $a3: \bigwedge j. j \leq \text{inth } (nfilter P xs 0) 0 \implies$ 
 $\text{ifilter } P (\text{suffix } (\text{inth } (nfilter P xs 0) 0) xs) = \text{ifilter } P (\text{suffix } j xs)$ 
have  $\text{Suc } (\text{inth } (nfilter P xs 0) 0) = \text{inth } (nfilter P xs 1) 0$ 
using  $a2 a1$  by (metis (no-types) One-nat-def Suc-le-D diff-Suc-Suc diff-zero le0)
then show  $\text{ifilter } P (\text{case } \text{inth } (nfilter P xs (\text{Suc } 0)) 0 \text{ of } 0 \Rightarrow x1a \odot xs \mid$ 
 $\text{Suc } n \Rightarrow \text{suffix } n xs) = \text{ifilter } P (\text{suffix } nat xs)$ 
using  $a3 a2 a1$  by (simp add: Nitpick.case-nat-unfold)
qed
qed
next
case False
then show ?thesis using False ICons  $\langle \exists a \in \text{iset } xs. P a \rangle$ 
proof (auto split: nat.split)
fix  $x2 :: nat$ 
fix  $x$ 
fix  $x2a$ 
assume  $a0: \neg P x1a$ 
assume  $a1: (\bigwedge j. j \leq \text{inth } (nfilter P xs 0) (\text{Suc } 0) \implies$ 
 $\text{inth } (nfilter P xs 0) 0 < j \implies$ 
 $\text{ifilter } P (\text{suffix } (\text{inth } (nfilter P xs 0) (\text{Suc } 0)) xs) = \text{ifilter } P (\text{suffix } j xs))$ 
assume  $a2: x2a \leq x2$ 
assume  $a3: 0 < \text{ilen } (\text{ifilter } P xs)$ 
assume  $a4: \text{inth } (nfilter P xs (\text{Suc } 0)) 0 < \text{Suc } x2a$ 
assume  $a5: x \in \text{iset } xs$ 
assume  $a6: P x$ 
assume  $a7: \text{inth } (nfilter P xs (\text{Suc } 0)) (\text{Suc } 0) = \text{Suc } x2$ 
assume  $a8: j = \text{Suc } x2a$ 
show  $\text{ifilter } P (\text{suffix } x2 xs) = \text{ifilter } P (\text{suffix } x2a xs)$ 
proof -

```

```

have 1:  $j > 0$ 
  using a8 by auto
have 2:  $x2a = j - 1$ 
  by (simp add: a8)
have 3:  $x2 = \text{inth} (\text{nfilter } P \text{ } xs \text{ } (\text{Suc } 0)) (\text{Suc } 0) - (\text{Suc } 0)$ 
  by (simp add: a7)
have 4:  $(\text{inth} (\text{nfilter } P \text{ } xs \text{ } (\text{Suc } 0)) (\text{Suc } 0) - (\text{Suc } 0)) = (\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) (\text{Suc } 0))$ 
  by (metis Suc-leI a3 a5 a6 nfilter-ilen nfilter-inth-n-zero)
have 5:  $0 < \text{ilen} (\text{ifilter } P \text{ } xs)$ 
  by (simp add: a3)
have 6:  $\text{ifilter } P (\text{suffix } x2 \text{ } xs) = \text{ifilter } P (\text{suffix} (\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) (\text{Suc } 0)) \text{ } xs)$ 
  by (simp add: 3 4)
have 7:  $\text{ifilter } P (\text{suffix } x2a \text{ } xs) = \text{ifilter } P (\text{suffix } (j - 1) \text{ } xs)$ 
  using 2 by blast
have 8:  $j - 1 \leq \text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } 1$ 
  by (metis 2 3 4 One-nat-def a2)
have 9:  $(\text{inth} (\text{nfilter } P \text{ } xs \text{ } (\text{Suc } 0)) \text{ } 0) - (\text{Suc } 0) = (\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } 0)$ 
  using a5 a6 nfilter-inth-n-zero by fastforce
have 10:  $(\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } 0) < j - (\text{Suc } 0)$ 
  using nfilter-inth-ICons[of  $P \text{ } x1a \text{ } xs \text{ } -$ ]
    ifilter-inth-aa[of  $xs \text{ } P \text{ } j - 1$ ]
    nfilter-inth-n-zero[of  $xs \text{ } P$ ] nfilter-holds[of  $x1a \odot xs \text{ } P$ ]
  using 9 2 True a0 a3 a4 a5 a6 a8
  by (metis 1 One-nat-def Suc-n-not-le-n diff-Suc-less less-Suc-eq less-imp-diff-less
    less-imp-le-nat nfilter-ilen nfilter-lower-bound)
have 11:  $\text{ifilter } P (\text{suffix} (\text{inth} (\text{nfilter } P \text{ } xs \text{ } 0) \text{ } 1) \text{ } xs) = \text{ifilter } P (\text{suffix } (j - 1) \text{ } xs)$ 
  by (metis 10 8 One-nat-def a1)
show ?thesis by (metis 11 3 4 7 One-nat-def)
qed
qed
qed
next
case False
then show ?thesis
  using ICons.premis(2) by auto
qed
qed

lemma ifilter-suffixes-imap-help-j:
assumes  $j \leq \text{inth}(\text{nfilter } P \text{ } xs \text{ } 0) (\text{Suc } i)$ 
   $i < \text{ilen}(\text{ifilter } P \text{ } xs)$ 
   $\text{inth}(\text{nfilter } P \text{ } xs \text{ } 0) \text{ } i < j$ 
   $\exists x \in \text{iset } xs. P \text{ } x$ 
shows  $(\text{ifilter } P (\text{suffix} (\text{inth}(\text{nfilter } P \text{ } xs \text{ } 0) (\text{Suc } i)) \text{ } xs)) = (\text{ifilter } P (\text{suffix } j \text{ } xs))$ 
  using assms
proof (induct xs arbitrary: j i)
case (INil x)
then show ?case by simp
next
case (ICons x1a xs)

```

```

then show ?case
proof (cases (∃ a ∈ iset xs. P a))
case True
then show ?thesis
proof (cases P x1a)
case True
then show ?thesis using ICons True (∃ a ∈ iset xs. P a)
proof (auto split: nat.split)
fix x
fix x2::nat
fix x2a
assume a0: (∧ j. j ≤ inth (nfilter P xs 0) (Suc i) ⇒
i < ilen (ifilter P xs) ⇒
inth (nfilter P xs 0) i < j ⇒
ifilter P (suffix (inth (nfilter P xs 0) (Suc i)) xs) = ifilter P (suffix j xs))
assume a1: x2a ≤ x2
assume a2: i < Suc (ilen (ifilter P xs))
assume a3: (case i of 0 ⇒ 0 | Suc x ⇒ inth (nfilter P xs (Suc 0)) x) < Suc x2a
assume a4: P x1a
assume a5: x ∈ iset xs
assume a6: P x
assume a7: inth (nfilter P xs (Suc 0)) i = Suc x2
assume a8: j = Suc x2a
show ifilter P (suffix x2 xs) = ifilter P (suffix x2a xs)
proof -
have 1: i=0 ⟶ ifilter P (suffix x2 xs) = ifilter P (suffix x2a xs)
using a0 a1 a2 a3 a4 a5 a6 a7 a8
by (metis One-nat-def add-diff-cancel-left' ilen-gr-zero nfilter-inth-n-zero
plus-1-eq-Suc ifilter-suffixes-imap-help-0)
have 2: i>0 ⟶ (inth (nfilter P xs (Suc 0)) (i-1)) - (Suc 0) = inth (nfilter P xs 0) (i-1)
using a0 a1 a2 a3 a4 a5 a6 a7 a8
by (metis One-nat-def Suc-leI add-leD1 le-add-diff-inverse2
less-Suc-eq-le nfilter-ilen nfilter-inth-n-zero)
have 3: i>0 ⟶ inth (nfilter P xs (Suc 0)) (i-1) < Suc x2a
using a0 a1 a2 a3 a4 a5 a6 a7 a8
by (metis Nitpick.case-nat-unfold less-Suc0 not-less-eq)
have 4: i>0 ⟶ inth (nfilter P xs (Suc 0)) (i-1) > 0
using a0 a1 a2 a3 a4 a5 a6 a7 a8 nfilter-lower-bound[of xs P i-1 Suc 0]
by (metis One-nat-def Suc-le-lessD Suc-le-mono Suc-pred less-imp-le-nat nfilter-ilen)
have 5: i>0 ⟶ inth (nfilter P xs 0) (i-1) < x2a
using 2 3 4 by linarith
have 6: i>0 ⟶ x2a ≤ inth (nfilter P xs 0) i
using a0 a1 a2 a3 a4 a5 a6 a7 a8
by (metis One-nat-def add-diff-cancel-left'
less-Suc-eq-le nfilter-ilen nfilter-inth-n-zero plus-1-eq-Suc)
have 7: i>0 ⟶ i - 1 < ilen (ifilter P xs)
using a2 by linarith
have 8: i>0 ⟶ ifilter P (suffix (inth (nfilter P xs 0) i) xs) = ifilter P (suffix x2a xs)
using a0 5 6 7 by (metis Suc-pred')
have 9: i>0 ⟶ (inth (nfilter P xs 0) i) = x2

```

```

using 7 a0 a1 a2 a3 a4 a5 a6 a7 a8
by (metis One-nat-def Suc-leI Suc-pred add-diff-cancel-left'
    nfilter-ilen nfilter-inth-n-zero plus-1-eq-Suc)
from 1 2 8 9 show ?thesis by auto
qed
qed
next
case False
then show ?thesis using ICons False ⟨(∃ a ∈ iset xs. P a)⟩
proof (auto split: nat.split)
  fix x
  fix x2::nat
  fix x2a
  assume b0: (∧ j i. j ≤ inth (nfilter P xs 0) (Suc i) ⇒
    i < ilen (ifilter P xs) ⇒
    inth (nfilter P xs 0) i < j ⇒
    ifilter P (suffix (inth (nfilter P xs 0) (Suc i)) xs) = ifilter P (suffix j xs))
  assume b1: x2a ≤ x2
  assume b2: i < ilen (ifilter P xs)
  assume b3: inth (nfilter P xs (Suc 0)) i < Suc x2a
  assume b4: ¬ P x1a
  assume b5: x ∈ iset xs
  assume b6: P x
  assume b7: inth (nfilter P xs (Suc 0)) (Suc i) = Suc x2
  assume b8: j = Suc x2a
  show ifilter P (suffix x2 xs) = ifilter P (suffix x2a xs)
  proof -
    have 1: i=0 ⟶ (inth (nfilter P xs (Suc 0)) (Suc i)) - (Suc 0) =
      inth (nfilter P xs 0) (Suc i)
    using b0 b1 b2 b3 b4 b5 b6 b7 b8
    by (metis Suc-leI nfilter-ilen nfilter-inth-n-zero)
    have 2: i=0 ⟶ (inth (nfilter P xs 0) (Suc i)) = x2
    using 1 b7 by linarith
    have 3: i=0 ⟶ inth (nfilter P xs 0) i < x2a
    using b0 b1 b2 b3 b4 b5 b6 b7 b8 nfilter-lower-bound[of xs P i Suc 0]
      nfilter-inth-n-zero[of xs P i Suc 0]
    by (metis One-nat-def Suc-le-lessD le0 less-Suc-eq-le
      ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc)
    have 4: i=0 ⟶ i < ilen (ifilter P xs)
    using b2 by blast
    have 5: i=0 ⟶ x2a ≤ inth (nfilter P xs 0) (Suc i)
    using 1 2 b1 by blast
    have 6: i=0 ⟶ ifilter P (suffix x2 xs) = ifilter P (suffix x2a xs)
    using 2 3 5 b0 b2 by blast
    have 7: i>0 ⟶ (inth (nfilter P xs (Suc 0)) (i)) - (Suc 0) = inth (nfilter P xs 0) (i)
    using b0 b1 b2 b3 b4 b5 b6 b7 b8
    by (metis less-imp-le-nat nfilter-ilen nfilter-inth-n-zero)
    have 8: i>0 ⟶ inth (nfilter P xs (Suc 0)) i < Suc x2a
    using b3 by auto
    have 9: i>0 ⟶ inth (nfilter P xs (Suc 0)) (i) > 0

```

```

using b0 b1 b2 b3 b4 b5 b6 b7 b8
by (metis gr-zeroI idx-nfilter-gr less-imp-le-nat nfilter-ilen not-less0)
have 10:  $i > 0 \longrightarrow \text{inth} (\text{nfilter } P \text{ } xs \ 0) \ (i) < x2a$ 
using 7 9 b3 by linarith
have 11:  $i > 0 \longrightarrow x2a \leq \text{inth} (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } i)$ 
using b0 b1 b2 b3 b4 b5 b6 b7 b8
by (metis One-nat-def Suc-leI add-diff-cancel-left' nfilter-ilen
    nfilter-inth-n-zero plus-1-eq-Suc)
have 12:  $i > 0 \longrightarrow i < \text{ilen} (\text{ifilter } P \text{ } xs)$ 
using b2 by simp
have 13:  $i > 0 \longrightarrow \text{ifilter } P \text{ } (\text{suffix} (\text{inth} (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } i)) \ xs) =$ 
     $\text{ifilter } P \text{ } (\text{suffix } x2a \ xs)$ 
using 10 11 b0 b1 b2 b3 b4 b5 b6 b7 b8 by blast
have 14:  $i > 0 \longrightarrow (\text{inth} (\text{nfilter } P \text{ } xs \ 0) \ (\text{Suc } i)) = x2$ 
using b0 b1 b2 b3 b4 b5 b6 b7 b8
by (metis One-nat-def Suc-leI add-diff-cancel-left' nfilter-ilen
    nfilter-inth-n-zero plus-1-eq-Suc)
show ?thesis using 13 14 6 by blast
qed
qed
qed
next
case False
then show ?thesis using ICons False by auto
qed
qed

```

**lemma** *ifilter-suffixes-imap-help-j-aa:*

```

assumes  $j \leq \text{inth}(\text{nfilter } P \text{ } (\text{suffixes } xs) \ 0) \ (\text{Suc } i)$ 
     $i < \text{ilen}(\text{ifilter } P \text{ } (\text{suffixes } xs))$ 
     $\text{inth}(\text{nfilter } P \text{ } (\text{suffixes } xs) \ 0) \ i < j$ 
     $\exists x \in \text{iset} (\text{suffixes } xs). \ P \ x$ 
shows  $(\text{ifilter } P \text{ } (\text{suffixes} (\text{suffix} (\text{inth}(\text{nfilter } P \text{ } (\text{suffixes } xs) \ 0) \ (\text{Suc } i)) \ xs))) =$ 
     $(\text{ifilter } P \text{ } (\text{suffixes} (\text{suffix } j \ xs)))$ 

```

**proof** –

```

have 1:  $\text{inth}(\text{nfilter } P \text{ } (\text{suffixes } xs) \ 0) \ (\text{Suc } i) \leq \text{ilen} (\text{suffixes } xs)$ 
by (metis Suc-leI add-cancel-right-left assms(2) assms(4) nfilter-ilen nfilter-upper-bound)
have 2:  $(\text{suffixes} (\text{suffix} (\text{inth}(\text{nfilter } P \text{ } (\text{suffixes } xs) \ 0) \ (\text{Suc } i)) \ xs)) =$ 
     $(\text{suffix} (\text{inth}(\text{nfilter } P \text{ } (\text{suffixes } xs) \ 0) \ (\text{Suc } i)) (\text{suffixes } xs))$ 
using 1 suffix-suffixes by fastforce
have 3:  $(\text{suffixes} (\text{suffix } j \ xs)) = (\text{suffix } j (\text{suffixes } xs))$ 
using 1 assms le-trans suffix-suffixes[of j xs] by fastforce
show ?thesis
by (simp add: 2 3 assms(1) assms(2) assms(3) assms(4) ifilter-suffixes-imap-help-j)
qed

```

**lemma** *ifilter-suffixes-imap:*

```

assumes  $(\text{Suc } i) \leq \text{ilen} (\text{ifilter } f \text{ } (\text{suffixes } \sigma))$ 
     $\exists x \in \text{iset}(\text{suffixes } \sigma). \ f \ x$ 
shows  $(\text{suffix} (\text{Suc } i) (\text{imap } (\lambda s. \ \text{inth } s \ 0) (\text{ifilter } f \text{ } (\text{suffixes } \sigma)))) =$ 

```

```

      (imap (λs. inth s 0)
        (ifilter f (suffixes (suffix (Suc (inth (nfilter f (suffixes σ) 0) i)) σ))))
proof –
  have 1: (suffix (Suc i) (imap (λs. inth s 0) (ifilter f (suffixes σ)))) =
    (imap (λs. inth s 0) (suffix (Suc i) (ifilter f (suffixes σ))))
  by (simp add: assms(1) imap-suffix)
  have 2: (Suc (inth (nfilter f (suffixes σ) 0) i)) ≤ ilen(suffixes σ)
    by (metis ilen-ifilter-le Suc-leD assms(1) assms(2) diff-is-0-eq diff-zero
      ifilter-nfilter-suffix-1 suffix-ilen not-less-eq-eq)
  have 3: (ifilter f (suffixes (suffix (Suc (inth (nfilter f (suffixes σ) 0) i)) σ))) =
    (ifilter f (suffix (Suc (inth (nfilter f (suffixes σ) 0) i)) (suffixes σ)))

    using 2 suffix-suffixes by fastforce
  have 4: (suffix (Suc i) (ifilter f (suffixes σ))) =
    (ifilter f (suffix (inth (nfilter f (suffixes σ) 0) (Suc i)) (suffixes σ)))
  by (simp add: assms(1) assms(2) ifilter-nfilter-suffix-1)
  have 5: (Suc (inth (nfilter f (suffixes σ) 0) i)) ≤ inth(nfilter f (suffixes σ) 0) (Suc i)
    by (simp add: Suc-leI Suc-le-lessD assms(1) assms(2) idx-nfilter-mono nfilter-ilen)
  have 6: i < ilen(ifilter f (suffixes σ))
    using Suc-le-eq assms(1) by blast
  have 7: inth (nfilter f (suffixes σ) 0) i < (Suc (inth (nfilter f (suffixes σ) 0) i))
    by simp
  have 8: (ifilter f (suffix (inth (nfilter f (suffixes σ) 0) (Suc i)) (suffixes σ))) =
    (ifilter f (suffix (Suc (inth (nfilter f (suffixes σ) 0) i)) (suffixes σ)))
    using 5 6 7
    ifilter-suffixes-imap-help-j[of (Suc (inth (nfilter f (suffixes σ) 0) i)) f suffixes σ i]
    assms(2) by blast
  show ?thesis
  by (simp add: 1 3 4 8)
qed

```

**lemma** *sfxfilter-suffix-ilen*:

```

assumes k ≤ ilen σ
      ∃ x ∈ iset (suffixes(suffix k σ)). f x
shows ilen (ifilter f (suffixes (suffix k σ))) ≤ ilen (ifilter f (suffixes σ))
using assms
proof (induct k arbitrary: σ)
case 0
then show ?case by simp
next
case (Suc k)
then show ?case
  proof (cases σ)
  case (INil x1)
  then show ?thesis
  by simp
  next
  case (ICons x21 x22)
  then show ?thesis
  proof auto

```

```

fix y
assume a0:  $\sigma = x21 \odot x22$ 
assume a1:  $f (x21 \odot x22)$ 
assume a2:  $osfx\ y\ x22$ 
assume a3:  $f\ y$ 
show  $ilen\ (ifilter\ f\ (suffixes\ (suffix\ k\ x22))) \leq Suc\ (ilen\ (ifilter\ f\ (suffixes\ x22)))$ 
  using Suc a0
  by (metis ilen-ifilter-le Suc.hyps Suc.prem1(2) suffix-ilen-code
      suffix-suc ilen-suffixes le-0-eq le-SucI less-imp-le-nat zero-less-Suc)
next
assume b0:  $\sigma = x21 \odot x22$ 
assume b1:  $f (x21 \odot x22)$ 
assume b2:  $\forall x \in iset\ (suffixes\ x22). \neg f\ x$ 
show  $ilen\ (ifilter\ f\ (suffixes\ (suffix\ k\ x22))) = 0$ 
  using b0 b1 b2 Suc.prem1 osfx-suffix
  by (simp add: osfx-order.order.trans)
next
fix y
assume c0:  $\sigma = x21 \odot x22$ 
assume c1:  $\neg f (x21 \odot x22)$ 
assume c2:  $osfx\ y\ x22$ 
assume c3:  $f\ y$ 
show  $ilen\ (ifilter\ f\ (suffixes\ (suffix\ k\ x22))) \leq ilen\ (ifilter\ f\ (suffixes\ x22))$ 
  using c0 c1
  by (metis ilen-ifilter-le Suc.hyps Suc.prem1(2) ilen-gr-zero
      suffix-ilen-code suffix-suc ilen-suffixes le-0-eq)
next
assume d0:  $\sigma = x21 \odot x22$ 
assume d1:  $\neg f (x21 \odot x22)$ 
assume d2:  $\forall x \in iset\ (suffixes\ x22). \neg f\ x$ 
show  $ilen\ (ifilter\ f\ (suffixes\ (suffix\ k\ x22))) = 0$ 
  using d0 d1 d2 Suc.prem1 osfx-suffix
  by (simp add: osfx-order.order.trans)
qed
qed
qed

lemma sfxfilter-suffix-inth:
  assumes  $k \leq ilen\ \sigma$ 
     $\exists x \in iset\ (suffixes\ (suffix\ k\ \sigma)). f\ x$ 
     $j \leq ilen\ (ifilter\ f\ (suffixes\ (suffix\ k\ \sigma)))$ 
  shows  $(inth\ (ifilter\ f\ (suffixes\ (suffix\ k\ \sigma)))\ j) =$ 
     $(inth\ (suffix\ (ilen\ (ifilter\ f\ (suffixes\ \sigma))) - ilen\ ((ifilter\ f\ (suffixes\ (suffix\ k\ \sigma))))\ j)$ 
  using assms
  proof (induct  $k$  arbitrary:  $\sigma$ )
  case 0
  then show ?case by simp
  next
  case (Suc k)

```

```

then show ?case
  proof (cases  $\sigma$ )
  case (INil  $x1$ )
  then show ?thesis
  by simp
next
case (ICons  $x21$   $x22$ )
then show ?thesis
  proof auto
    fix  $y$ 
    assume  $a0$ :  $\sigma = x21 \odot x22$ 
    assume  $a1$ :  $f (x21 \odot x22)$ 
    assume  $a2$ :  $osfx\ y\ x22$ 
    assume  $a3$ :  $f\ y$ 
    show
       $inth (ifilter\ f\ (suffixes\ (suffix\ k\ x22)))\ j =$ 
       $inth$ 
       $(case\ Suc\ (ilen\ (ifilter\ f\ (suffixes\ x22))) - ilen\ (ifilter\ f\ (suffixes\ (suffix\ k\ x22))))\ of$ 
       $0 \Rightarrow (x21 \odot x22) \odot ifilter\ f\ (suffixes\ x22) \mid Suc\ m \Rightarrow suffix\ m\ (ifilter\ f\ (suffixes\ x22)))$ 
       $j$ 
    using  $a0\ Suc$ 
    by (simp add: Suc-diff-le sfxfilter-suffix-ilen)
  next
    assume  $b0$ :  $\sigma = x21 \odot x22$ 
    assume  $b1$ :  $f (x21 \odot x22)$ 
    assume  $b2$ :  $\forall x \in iset\ (suffixes\ x22). \neg f\ x$ 
    show  $inth (ifilter\ f\ (suffixes\ (suffix\ k\ x22)))\ j = x21 \odot x22$ 
    using Suc  $b0\ b1\ b2$  using osfx-suffix
    using osfx-order.order-trans by fastforce
  next
    fix  $y$ 
    assume  $c0$ :  $\sigma = x21 \odot x22$ 
    assume  $c1$ :  $\neg f (x21 \odot x22)$ 
    assume  $c2$ :  $osfx\ y\ x22$ 
    assume  $c4$ :  $f\ y$ 
    show  $inth (ifilter\ f\ (suffixes\ (suffix\ k\ x22)))\ j =$ 
     $inth$ 
     $(suffix\ (ilen\ (ifilter\ f\ (suffixes\ x22)) - ilen\ (ifilter\ f\ (suffixes\ (suffix\ k\ x22))))$ 
     $(ifilter\ f\ (suffixes\ x22)))$ 
     $j$ 
    using Suc  $c0$  by simp
  next
    assume  $d0$ :  $\sigma = x21 \odot x22$ 
    assume  $d1$ :  $\neg f (x21 \odot x22)$ 
    assume  $d2$ :  $\forall x \in iset\ (suffixes\ x22). \neg f\ x$ 
    show  $inth (ifilter\ f\ (suffixes\ (suffix\ k\ x22)))\ j = x21 \odot x22$ 
    using Suc  $d0\ d1\ d2$  osfx-suffix
    using osfx-order.order-trans by fastforce
  qed
qed

```



qed

**lemma** *sfxfilter-suffix-suffix*:

**assumes**  $k \leq \text{ilen } \sigma$

$\exists x \in \text{iset } (\text{suffixes}(\text{suffix } k \ \sigma)). f \ x$

**shows**  $(\text{ifilter } f \ (\text{suffixes} \ (\text{suffix } k \ \sigma))) =$   
 $(\text{suffix} \ (\text{ilen}(\text{ifilter } f \ (\text{suffixes } \sigma)) - \text{ilen}((\text{ifilter } f \ (\text{suffixes} \ (\text{suffix } k \ \sigma)))))$   
 $(\text{ifilter } f \ (\text{suffixes } \sigma))$

**by** (*simp add: assms(1) assms(2) interval-eq-inth-eq sfxfilter-suffix-ilen sfxfilter-suffix-inth*)

**lemma** *sfxfilter-suffix-suffix-a*:

**assumes**  $jj \leq \text{ilen} \ (\text{ifilter } f \ (\text{suffixes } xs))$

$\exists x \in \text{iset } (\text{suffixes } xs). f \ x$

**shows**  $(\text{suffix } jj \ (\text{ifilter } f \ (\text{suffixes } xs))) =$   
 $\text{ifilter } f \ (\text{suffixes} \ (\text{suffix} \ ((\text{ilen } xs) - \text{ilen} \ (\text{inth} \ (\text{ifilter } f \ (\text{suffixes } xs)) \ jj)) \ xs))$

**proof** –

**have** 1:  $(\text{suffix } jj \ (\text{ifilter } f \ (\text{suffixes } xs))) =$   
 $\text{ifilter } f \ (\text{suffix} \ (\text{inth} \ (\text{nfilter } f \ (\text{suffixes } xs) \ 0) \ jj) \ (\text{suffixes } xs))$

**using** *ifilter-nfilter-suffix-1*

**by** (*simp add: ifilter-nfilter-suffix-1 assms(1) assms(2)*)

**have** 2:  $(\text{inth} \ (\text{nfilter } f \ (\text{suffixes } xs) \ 0) \ jj) \leq \text{ilen} \ (\text{suffixes } xs)$

**by** (*metis add-cancel-right-left assms(1) assms(2) nfilter-ilen nfilter-upper-bound*)

**have** 3:  $(\text{suffix} \ (\text{inth} \ (\text{nfilter } f \ (\text{suffixes } xs) \ 0) \ jj) \ (\text{suffixes } xs)) =$   
 $(\text{suffixes} \ (\text{suffix} \ (\text{inth} \ (\text{nfilter } f \ (\text{suffixes } xs) \ 0) \ jj) \ xs))$

**using** *suffix-suffixes[of (inth (nfilter f (suffixes xs) 0) jj) xs ] 2 by blast*

**have** 4:  $(\text{inth} \ (\text{ifilter } f \ (\text{suffixes } xs)) \ jj) = \text{inth} \ (\text{suffixes } xs) \ (\text{inth} \ (\text{nfilter } f \ (\text{suffixes } xs) \ 0) \ jj)$

**by** (*metis nfilter-imap-ifilter assms(2) inth-imap*)

**have** 5:  $\text{ilen}(\text{suffixes } xs) \leq \text{ilen} \ (xs)$

**by** *simp*

**have** 6:  $(\text{inth} \ (\text{nfilter } f \ (\text{suffixes } xs) \ 0) \ jj) \leq \text{ilen} \ (xs)$

**using** 2 5 *le-trans by blast*

**have** 7:  $\text{inth} \ (\text{suffixes } xs) \ (\text{inth} \ (\text{nfilter } f \ (\text{suffixes } xs) \ 0) \ jj) =$   
 $(\text{suffix} \ (\text{inth} \ (\text{nfilter } f \ (\text{suffixes } xs) \ 0) \ jj) \ xs)$

**by** (*simp add: 6 inth-suffixes*)

**have** 8:  $(\text{inth} \ (\text{nfilter } f \ (\text{suffixes } xs) \ 0) \ jj) =$   
 $((\text{ilen } xs) - \text{ilen} \ (\text{inth} \ (\text{ifilter } f \ (\text{suffixes } xs)) \ jj))$

**by** (*simp add: 4 6 7*)

**show** *?thesis*

**using** 1 3 8 **by** *auto*

qed

**lemma** *sfx-suffix-upperbound*:

$(\forall jj < (\text{ilen}(\text{ifilter } f \ (\text{suffixes } \sigma)) - \text{ilen} \ (\text{ifilter } f \ (\text{suffixes} \ (\text{suffix } k \ \sigma)))).$   
 $((\text{ilen } \sigma) - \text{ilen} \ (\text{inth} \ (\text{ifilter } f \ (\text{suffixes } \sigma)) \ jj)) < k$   
 $)$

**proof**

**fix** *jj*

**show**  $jj < \text{ilen} \ (\text{ifilter } f \ (\text{suffixes } \sigma)) - \text{ilen} \ (\text{ifilter } f \ (\text{suffixes} \ (\text{suffix } k \ \sigma))) \longrightarrow$   
 $\text{ilen } \sigma - \text{ilen} \ (\text{inth} \ (\text{ifilter } f \ (\text{suffixes } \sigma)) \ jj) < k$

```

proof (induct k arbitrary:  $\sigma$  jj)
case 0
then show ?case by simp
next
case (Suc k)
then show ?case
  proof (cases  $\sigma$ )
  case (INil x1)
  then show ?thesis using Suc by simp
  next
  case (ICons x21 x22)
  then show ?thesis
    proof (cases jj)
    case 0
    then show ?thesis using Suc ICons using Suc-diff-le less-Suc-eq-0-disj by auto
    next
    case (Suc nat)
    then show ?thesis
      using Suc.hyps ICons Suc-diff-le less-Suc-eq-0-disj diff-is-0-eq gr-implies-not-zero by auto
    qed
  qed
qed
qed
qed

end

```

## 14 Until and Since operator

```

theory UntilSince
imports Semantics Fuse Theorems TimeReversal
begin

```

This theory introduces the weak and strong versions of the until and since operators. The theorems from [11] are proven in a mostly deductive style.

### 14.1 Definitions

```

definition until-d :: ('a :: world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula
where until-d F G  $\equiv \lambda s. ( (\exists k \leq \text{ilen } s. ((\text{suffix } k \ s) \models G) \wedge$ 
   $(\forall j. j < k \longrightarrow ((\text{suffix } j \ s) \models F) )) )$ 

```

```

definition since-d :: ('a :: world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula
where since-d F G  $\equiv \lambda s. ( (\exists k \leq \text{ilen } s. ( (\text{prefix } k \ s) \models G) \wedge$ 
   $(\forall j. k < j \wedge j \leq \text{ilen } s \longrightarrow ((\text{prefix } j \ s) \models F) )) )$ 

```

**syntax**

```

-until-d :: [lift, lift]  $\Rightarrow$  lift      ((-  $\mathcal{U}$  -) [84,84] 83)
-since-d :: [lift, lift]  $\Rightarrow$  lift      ((-  $\mathcal{S}$  -) [84,84] 83)

```

### **syntax** (ASCII)

-until-d :: [lift, lift]  $\Rightarrow$  lift      ((- until -) [84,84] 83)  
-since-d :: [lift, lift]  $\Rightarrow$  lift      ((- since -) [84,84] 83)

### **translations**

-until-d  $\Rightarrow$  CONST until-d  
-since-d  $\Rightarrow$  CONST since-d

**definition** *suntil-d* :: ('a :: world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula  
**where** *suntil-d* F G  $\equiv$  LIFT( $\circ$ (F  $\mathcal{U}$  G))

**definition** *ssince-d* :: ('a :: world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula  
**where** *ssince-d* F G  $\equiv$  LIFT(prev(F  $\mathcal{S}$  G))

**definition** *wait-d* :: ('a :: world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula  
**where** *wait-d* F G  $\equiv$  LIFT( $\square$  F  $\vee$  F  $\mathcal{U}$  G)

**definition** *pwait-d* :: ('a :: world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula  
**where** *pwait-d* F G  $\equiv$  LIFT(*bi* F  $\vee$  F  $\mathcal{S}$  G)

**definition** *release-d* :: ('a :: world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula  
**where** *release-d* F G  $\equiv$  LIFT( $\neg$ (( $\neg$  F)  $\mathcal{U}$  ( $\neg$  G)))

**definition** *prelease-d* :: ('a :: world) formula  $\Rightarrow$  'a formula  $\Rightarrow$  'a formula  
**where** *prelease-d* F G  $\equiv$  LIFT( $\neg$ (( $\neg$  F)  $\mathcal{S}$  ( $\neg$  G)))

### **syntax**

-wait-d :: [lift, lift]  $\Rightarrow$  lift      ((- W -) [84,84] 83)  
-pwait-d :: [lift, lift]  $\Rightarrow$  lift      ((- PW -) [84,84] 83)  
-release-d :: [lift, lift]  $\Rightarrow$  lift      ((- R -) [84,84] 83)  
-prelease-d :: [lift, lift]  $\Rightarrow$  lift      ((- PR -) [84,84] 83)  
-suntil-d :: [lift, lift]  $\Rightarrow$  lift      ((-  $\mathcal{U}^s$  -) [84,84] 83)  
-ssince-d :: [lift, lift]  $\Rightarrow$  lift      ((-  $\mathcal{S}^s$  -) [84,84] 83)

### **syntax** (ASCII)

-wait-d :: [lift, lift]  $\Rightarrow$  lift      ((- wait -) [84,84] 83)  
-pwait-d :: [lift, lift]  $\Rightarrow$  lift      ((- pwait -) [84,84] 83)  
-release-d :: [lift, lift]  $\Rightarrow$  lift      ((- release -) [84,84] 83)  
-prelease-d :: [lift, lift]  $\Rightarrow$  lift      ((- prelease -) [84,84] 83)  
-suntil-d :: [lift, lift]  $\Rightarrow$  lift      ((- suntil -) [84,84] 83)  
-ssince-d :: [lift, lift]  $\Rightarrow$  lift      ((- ssince -) [84,84] 83)

### **translations**

-wait-d  $\Rightarrow$  CONST wait-d  
-pwait-d  $\Rightarrow$  CONST pwait-d  
-release-d  $\Rightarrow$  CONST release-d  
-prelease-d  $\Rightarrow$  CONST prelease-d  
-suntil-d  $\Rightarrow$  CONST suntil-d

$-ssince-d \quad \Rightarrow \text{CONST } ssince-d$

**definition**  $srelease-d :: ('a :: world) formula \Rightarrow 'a formula \Rightarrow 'a formula$   
**where**  $srelease-d F G \equiv LIFT(\neg((\neg F) \mathcal{W} (\neg G)))$

**definition**  $psrelease-d :: ('a :: world) formula \Rightarrow 'a formula \Rightarrow 'a formula$   
**where**  $psrelease-d F G \equiv LIFT(\neg((\neg F) \mathcal{PW} (\neg G)))$

**syntax**

$-srelease-d :: [lift, lift] \Rightarrow lift \quad ((- \mathcal{M} -) [84, 84] 83)$   
 $-psrelease-d :: [lift, lift] \Rightarrow lift \quad ((- \mathcal{PM} -) [84, 84] 83)$

**syntax** (*ASCII*)

$-srelease-d :: [lift, lift] \Rightarrow lift \quad ((- \text{ srelease } -) [84, 84] 83)$   
 $-psrelease-d :: [lift, lift] \Rightarrow lift \quad ((- \text{ psrelease } -) [84, 84] 83)$

**translations**

$-srelease-d \quad \Rightarrow \text{CONST } srelease-d$   
 $-psrelease-d \quad \Rightarrow \text{CONST } psrelease-d$

## 14.2 Axioms

### 14.2.1 NextUntil

**lemma** *NextUntilsema*:

**assumes**  $(\sigma \models \bigcirc(f \mathcal{U} g))$   
**shows**  $(\sigma \models (\bigcirc f) \mathcal{U} (\bigcirc g))$

**proof** –

**have** 0:  $0 < \text{ilen } \sigma \wedge$   
 $(\exists k \leq \text{ilen } (\text{suffix } (\text{Suc } 0) \sigma). g (\text{suffix } (\text{Suc } k) \sigma) \wedge (\forall j < k. f (\text{suffix } (\text{Suc } j) \sigma)))$   
**using** *assms* **by** (*auto simp add: itl-defs until-d-def*)  
**have** 1:  $0 < \text{ilen } \sigma$   
**using** 0 **by** *auto*  
**have** 2:  $(\exists k \leq \text{ilen } (\text{suffix } (\text{Suc } 0) \sigma). g (\text{suffix } (\text{Suc } k) \sigma) \wedge (\forall j < k. f (\text{suffix } (\text{Suc } j) \sigma)))$   
**using** 0 **by** *auto*  
**obtain**  $k$  **where** 3:  $k \leq \text{ilen } (\text{suffix } (\text{Suc } 0) \sigma) \wedge g (\text{suffix } (\text{Suc } k) \sigma) \wedge$   
 $(\forall j < k. f (\text{suffix } (\text{Suc } j) \sigma))$   
**using** 2 **by** *auto*  
**have** 4:  $g (\text{suffix } (\text{Suc } k) \sigma)$   
**using** 3 **by** *auto*  
**have** 5:  $k \leq \text{ilen } \sigma$   
**using** 3 0 **by** *auto*  
**have** 6:  $0 < \text{ilen } (\text{suffix } k \sigma)$   
**using** 1 3 **by** *auto*  
**have** 7:  $(\forall j < k. 0 < \text{ilen } (\text{suffix } j \sigma) \wedge f (\text{suffix } (\text{Suc } j) \sigma))$   
**using** 3 5 **by** *force*  
**have** 8:  $\exists k \leq \text{ilen } \sigma.$   
 $0 < \text{ilen } (\text{suffix } k \sigma) \wedge$   
 $g (\text{suffix } (\text{Suc } k) \sigma) \wedge (\forall j < k. 0 < \text{ilen } (\text{suffix } j \sigma) \wedge f (\text{suffix } (\text{Suc } j) \sigma))$

using 3 5 6 7 by blast  
 from 8 show ?thesis by (simp add: itl-defs until-d-def)  
 qed

**lemma** *NextUntilsemb*:

**assumes**  $(\sigma \models (\bigcirc f) \mathcal{U} (\bigcirc g))$

**shows**  $(\sigma \models \bigcirc(f \mathcal{U} g))$

**proof** –

**have** 1:  $\exists k \leq \text{ilen } \sigma$ .

$0 < \text{ilen } (\text{suffix } k \sigma) \wedge$

$g (\text{suffix } (\text{Suc } k) \sigma) \wedge (\forall j < k. 0 < \text{ilen } (\text{suffix } j \sigma) \wedge f (\text{suffix } (\text{Suc } j) \sigma))$

**using** *assms* **by** (auto simp add: itl-defs until-d-def)

**obtain**  $k$  **where** 2:  $k \leq \text{ilen } \sigma \wedge 0 < \text{ilen } (\text{suffix } k \sigma) \wedge$

$g (\text{suffix } (\text{Suc } k) \sigma) \wedge (\forall j < k. 0 < \text{ilen } (\text{suffix } j \sigma) \wedge f (\text{suffix } (\text{Suc } j) \sigma))$

**using** 1 **by** auto

**have** 3:  $0 < \text{ilen } \sigma$

**using** 2 **by** auto

**have** 4:  $k \leq \text{ilen } (\text{suffix } (\text{Suc } 0) \sigma)$

**using** 2 *suffix-ilen-good* **by** auto

**have** 5:  $g (\text{suffix } (\text{Suc } k) \sigma)$

**using** 2 **by** auto

**have** 6:  $(\forall j < k. f (\text{suffix } (\text{Suc } j) \sigma))$

**using** 2 **by** blast

**have** 7:  $0 < \text{ilen } \sigma \wedge$

$(\exists k \leq \text{ilen } (\text{suffix } (\text{Suc } 0) \sigma). g (\text{suffix } (\text{Suc } k) \sigma) \wedge (\forall j < k. f (\text{suffix } (\text{Suc } j) \sigma)))$

**using** 2 3 4 **by** blast

**from** 7 **show** ?thesis **by** (auto simp: itl-defs until-d-def)

qed

**lemma** *NextUntilsem*:

$\sigma \models \bigcirc(f \mathcal{U} g) = (\bigcirc f) \mathcal{U} (\bigcirc g)$

**using** *NextUntilsema NextUntilsemb* **using** *unl-lift2* **by** blast

**lemma** *NextUntil*:

$\vdash \bigcirc(f \mathcal{U} g) = (\bigcirc f) \mathcal{U} (\bigcirc g)$

**using** *NextUntilsem Valid-def* **by** blast

## 14.2.2 UntilNextUntil

**lemma** *UntilNextUntilsema*:

**assumes**  $0 < \text{ilen } \sigma \wedge$

$(\exists k > 0. k \leq \text{ilen } \sigma \wedge g (\text{suffix } k \sigma) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f (\text{suffix } j \sigma)))$

**shows**  $\sigma \models \bigcirc (f \mathcal{U} g)$

**proof** –

**have** 1:  $0 < \text{ilen } \sigma \wedge$

$(\exists k > 0. k \leq \text{ilen } \sigma \wedge g (\text{suffix } k \sigma) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f (\text{suffix } j \sigma)))$

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

**have** 3:  $(\exists k > 0. k \leq \text{ilen } \sigma \wedge g (\text{suffix } k \sigma) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f (\text{suffix } j \sigma)))$

**using** 1 **by** auto

**obtain**  $k$  **where** 4:  $(\text{Suc } k) > 0 \wedge (\text{Suc } k) \leq \text{ilen } \sigma \wedge g (\text{suffix } (\text{Suc } k) \sigma) \wedge$

$(\forall j. 0 < j \wedge j < (Suc\ k) \longrightarrow f\ (suffix\ j\ \sigma))$   
**using** 3 **by** (metis Suc-pred)  
**have** 5:  $k \leq ilen\ (suffix\ (Suc\ 0)\ \sigma)$   
**using** 4 **by** auto  
**have** 6:  $g\ (suffix\ (Suc\ k)\ \sigma)$   
**using** 4 **by** auto  
**have** 7:  $(\forall j < k. f\ (suffix\ (Suc\ j)\ \sigma))$   
**using** 4 **by** blast  
**have** 8:  $(\exists k \leq ilen\ (suffix\ (Suc\ 0)\ \sigma). g\ (suffix\ (Suc\ k)\ \sigma) \wedge (\forall j < k. f\ (suffix\ (Suc\ j)\ \sigma)))$   
**using** 4 5 **by** blast  
**have** 9:  $0 < ilen\ \sigma \wedge$   
 $(\exists k \leq ilen\ (suffix\ (Suc\ 0)\ \sigma). g\ (suffix\ (Suc\ k)\ \sigma) \wedge (\forall j < k. f\ (suffix\ (Suc\ j)\ \sigma)))$   
**using** 1 8 **by** blast  
**from** 9 **show** ?thesis **by** (auto simp add: itl-defs until-d-def)  
**qed**

**lemma** *UntilNextUntilsemb:*

**assumes**  $\sigma \models \bigcirc (f\ \mathcal{U}\ g)$   
**shows**  $0 < ilen\ \sigma \wedge$   
 $(\exists k > 0. k \leq ilen\ \sigma \wedge g\ (suffix\ k\ \sigma) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f\ (suffix\ j\ \sigma)))$   
**proof** –  
**have** 1:  $0 < ilen\ \sigma \wedge$   
 $(\exists k \leq ilen\ (suffix\ (Suc\ 0)\ \sigma). g\ (suffix\ (Suc\ k)\ \sigma) \wedge (\forall j < k. f\ (suffix\ (Suc\ j)\ \sigma)))$   
**using** assms **by** (auto simp add: itl-defs until-d-def)  
**have** 2:  $(\exists k \leq ilen\ (suffix\ (Suc\ 0)\ \sigma). g\ (suffix\ (Suc\ k)\ \sigma) \wedge (\forall j < k. f\ (suffix\ (Suc\ j)\ \sigma)))$   
**using** 1 **by** auto  
**obtain** k **where** 3:  $k \leq ilen\ (suffix\ (Suc\ 0)\ \sigma) \wedge g\ (suffix\ (Suc\ k)\ \sigma) \wedge$   
 $(\forall j < k. f\ (suffix\ (Suc\ j)\ \sigma))$   
**using** 2 **by** auto  
**have** 4:  $(Suc\ k) > 0$   
**by** simp  
**have** 5:  $g\ (suffix\ (Suc\ k)\ \sigma)$   
**using** 3 **by** auto  
**have** 6:  $(Suc\ k) \leq ilen\ \sigma$   
**using** 1 3 **by** auto  
**have** 7:  $(\forall j. 0 < j \wedge j < (Suc\ k) \longrightarrow f\ (suffix\ j\ \sigma))$   
**using** 3 less-Suc-eq-0-disj **by** auto  
**have** 8:  $(\exists k > 0. k \leq ilen\ \sigma \wedge g\ (suffix\ k\ \sigma) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f\ (suffix\ j\ \sigma)))$   
**using** 3 6 7 **by** blast  
**show** ?thesis **using** 1 8 **by** blast  
**qed**

**lemma** *UntilNextUntilsem:*

$(\sigma \models \bigcirc (f\ \mathcal{U}\ g)) = (0 < ilen\ \sigma \wedge$   
 $(\exists k > 0. k \leq ilen\ \sigma \wedge g\ (suffix\ k\ \sigma) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f\ (suffix\ j\ \sigma))))$   
**using** UntilNextUntilsema[of  $\sigma\ g$ ] UntilNextUntilsemb[of  $f\ g\ \sigma$ ] **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**  $\sigma \models g \vee f \wedge (\lambda s. 0 < \text{ilen } s \wedge (\exists k > 0. k \leq \text{ilen } s \wedge g (\text{suffix } k s) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f (\text{suffix } j s))))$

**using** *a* **unfolding** *until-d-def* **by** *simp force*

**next**

**assume** *b*:  $\sigma \models g \vee f \wedge (\lambda s. 0 < \text{ilen } s \wedge (\exists k > 0. k \leq \text{ilen } s \wedge g (\text{suffix } k s) \wedge (\forall j. 0 < j \wedge j < k \longrightarrow f (\text{suffix } j s))))$

**show**  $(\sigma \models f \mathcal{U} g)$

**using** *b* **unfolding** *until-d-def* **by** *simp*

*(metis gr-implies-not-zero ilen-gr-zero suffix-zero linorder-cases)*

**qed**

**lemma** *UntilNextUntil*:

$\vdash f \mathcal{U} g = (g \vee (f \wedge \bigcirc(f \mathcal{U} g)))$

**by** *(simp add: UntilNextUntilsem1 Valid-def)*

### 14.2.3 NotUntilFalse

**lemma** *NotUntilFalse*:

$\vdash \neg (f \mathcal{U} \#False)$

**by** *(simp add: intI until-d-def)*

### 14.2.4 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*

### 14.2.5 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)*

### 14.2.6 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)*

### 14.2.7 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 dual-order.strict-trans linorder-cases)

**lemma** *UntilAndDist*:

$\vdash (f \wedge g) \mathcal{U} h = ((f \mathcal{U} h) \wedge (g \mathcal{U} h))$

using *UntilAndDistsem Valid-def* by blast

### 14.2.8 untilNotImp

**lemma** *UntilNotImp*:

$\vdash f \mathcal{U} g \wedge (\neg g) \mathcal{U} h \longrightarrow f \mathcal{U} h$

by (auto simp add: Valid-def until-d-def)  
 (metis less-trans linorder-cases)

### 14.2.9 UntilUntil

**lemma** *UntilUntilsem*:

$(\sigma \models f \mathcal{U} g) = (\sigma \models f \mathcal{U} (f \mathcal{U} g))$

unfolding *until-d-def*

**proof**

**assume** *a*:  $\exists k \leq \text{ilen } \sigma. g (\text{suffix } k \sigma) \wedge (\forall j < k. f (\text{suffix } j \sigma))$

**show**  $\exists k \leq \text{ilen } \sigma. (\exists ka \leq \text{ilen } (\text{suffix } k \sigma). g (\text{suffix } ka (\text{suffix } k \sigma)) \wedge$   
 $(\forall j < ka. f (\text{suffix } j (\text{suffix } k \sigma)))) \wedge (\forall j < k. f (\text{suffix } j \sigma))$

**using** *a* **by** *simp-all force*

**next**

**assume** *b*:  $\exists k \leq \text{ilen } \sigma. (\exists ka \leq \text{ilen } (\text{suffix } k \sigma). g (\text{suffix } ka (\text{suffix } k \sigma)) \wedge$   
 $(\forall j < ka. f (\text{suffix } j (\text{suffix } k \sigma)))) \wedge (\forall j < k. f (\text{suffix } j \sigma))$

**show**  $\exists k \leq \text{ilen } \sigma. g (\text{suffix } k \sigma) \wedge (\forall j < k. f (\text{suffix } j \sigma))$

**proof** –

**obtain** *k* **where** *1*:  $k \leq \text{ilen } \sigma \wedge (\exists ka \leq \text{ilen } (\text{suffix } k \sigma). g (\text{suffix } ka (\text{suffix } k \sigma)) \wedge$   
 $(\forall j < ka. f (\text{suffix } j (\text{suffix } k \sigma)))) \wedge (\forall j < k. f (\text{suffix } j \sigma))$

**using** *b* **by** *auto*

**have** *2*:  $(\exists ka \leq \text{ilen } (\text{suffix } k \sigma). g (\text{suffix } ka (\text{suffix } k \sigma)) \wedge$   
 $(\forall j < ka. f (\text{suffix } j (\text{suffix } k \sigma))))$

**using** *1* **by** *auto*

**obtain** *ka* **where** *3*:  $ka \leq \text{ilen } (\text{suffix } k \sigma) \wedge g (\text{suffix } ka (\text{suffix } k \sigma)) \wedge$   
 $(\forall j < ka. f (\text{suffix } j (\text{suffix } k \sigma)))$

**using** *2* **by** *auto*

**have** *4*:  $ka + k \leq \text{ilen } \sigma$

**using** *1 3* **by** *auto*

**have** *5*:  $g (\text{suffix } (ka+k) \sigma)$

**using** *3* **by** *auto*

**have** *6*:  $(\forall j < ka+k. f (\text{suffix } j \sigma))$

**by** (metis *1 3 add.commute add-diff-inverse-nat suffix-suffix nat-add-left-cancel-less*)

**show** *?thesis*

**using** *4 5 6* **by** *blast*

**qed**

**qed**

**lemma** *UntilUntil*:

$\vdash f \mathcal{U} g = f \mathcal{U} (f \mathcal{U} g)$



using *UntilUntilsem* by *fastforce*

#### 14.2.10 UntilRightor

lemma *UntilRightOr*:

$\vdash f \mathcal{U} (g \mathcal{U} h) \longrightarrow (f \vee g) \mathcal{U} h$

by (auto simp add: *Valid-def until-d-def*)

(metis *Nat.le-diff-conv2 leI le-add-diff-inverse2 less-diff-conv2*)

#### 14.2.11 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:  $\exists k \leq \text{ilen } \sigma. g (\text{suffix } k \sigma) \wedge h (\text{suffix } k \sigma) \wedge (\forall j < k. f (\text{suffix } j \sigma))$

using *assms* by (simp add: *until-d-def*)

obtain *k* where 2:  $k \leq \text{ilen } \sigma \wedge g (\text{suffix } k \sigma) \wedge h (\text{suffix } k \sigma) \wedge (\forall j < k. f (\text{suffix } j \sigma))$

using 1 by auto

have 3:  $h (\text{suffix } k \sigma)$

using 2 by auto

have 4:  $k \leq \text{ilen } \sigma$

using 2 by auto

have 5:  $(\forall j < k.$

$\exists ka \leq \text{ilen } (\text{suffix } j \sigma). g (\text{suffix } (ka + j) \sigma) \wedge (\forall ja < ka. f (\text{suffix } (ja + j) \sigma)))$

proof

fix *j*

show  $j < k \longrightarrow$

$(\exists ka \leq \text{ilen } (\text{suffix } j \sigma). g (\text{suffix } (ka + j) \sigma) \wedge (\forall ja < ka. f (\text{suffix } (ja + j) \sigma)))$

proof

assume *a0*:  $j < k$

show  $(\exists ka \leq \text{ilen } (\text{suffix } j \sigma). g (\text{suffix } (ka + j) \sigma) \wedge (\forall ja < ka. f (\text{suffix } (ja + j) \sigma)))$

proof –

have 51:  $k - j \leq \text{ilen } (\text{suffix } j \sigma)$

using 4 *a0* by auto

have 52:  $g (\text{suffix } ((k - j) + j) \sigma)$

by (simp add: 2 *a0 less-imp-le-nat*)

have 53:  $(\forall ja < (k - j). f (\text{suffix } (ja + j) \sigma))$

using 2 *less-diff-conv* by blast

show ?thesis

using 51 52 53 by blast

qed

qed

qed

have 6:  $\exists k \leq \text{ilen } \sigma.$

$h (\text{suffix } k \sigma) \wedge$

$(\forall j < k. \exists ka \leq \text{ilen } (\text{suffix } j \sigma). g (\text{suffix } (ka + j) \sigma) \wedge (\forall ja < ka. f (\text{suffix } (ja + j) \sigma)))$

using 2 5 by blast

from 6 show ?thesis by (simp add: *until-d-def*)

qed

**lemma** *UntilRightAnd:*

$\vdash f \mathcal{U} (g \wedge h) \longrightarrow (f \mathcal{U} g) \mathcal{U} h$

**using** *UntilRightAndsem Valid-def* **by** *auto*

### 14.2.12 SincePrevSince

**lemma** *SincePrevSincesema:*

**assumes**  $0 < \text{ilen } \sigma \wedge$

$(\exists k < \text{ilen } \sigma. g (\text{prefix } k \sigma) \wedge (\forall j. k < j \wedge j < \text{ilen } \sigma \longrightarrow f (\text{prefix } j \sigma)))$

**shows**  $\sigma \models \text{prev} (f \mathcal{S} g)$

**proof** –

**have** 1:  $0 < \text{ilen } \sigma \wedge$

$(\exists k < \text{ilen } \sigma. g (\text{prefix } k \sigma) \wedge (\forall j. k < j \wedge j < \text{ilen } \sigma \longrightarrow f (\text{prefix } j \sigma)))$

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

**have** 2:  $(\exists k < \text{ilen } \sigma. g (\text{prefix } k \sigma) \wedge (\forall j. k < j \wedge j < \text{ilen } \sigma \longrightarrow f (\text{prefix } j \sigma)))$

**using** 1 **by** *auto*

**obtain**  $k$  **where** 3:  $k < \text{ilen } \sigma \wedge g (\text{prefix } k \sigma) \wedge (\forall j. k < j \wedge j < \text{ilen } \sigma \longrightarrow f (\text{prefix } j \sigma))$

**using** 2 **by** *auto*

**have** 4:  $k \leq \text{ilen } \sigma - \text{Suc } 0$

**using** 3 **by** *linarith*

**have** 5:  $g (\text{prefix } k (\text{prefix } (\text{ilen } \sigma - \text{Suc } 0) \sigma))$

**by** (*simp add: 1 3 pref-pref-help*)

**have** 6:  $(\forall j. k < j \wedge j \leq \text{ilen } \sigma - \text{Suc } 0 \longrightarrow f (\text{prefix } j (\text{prefix } (\text{ilen } \sigma - \text{Suc } 0) \sigma)))$

**using** 3 *pref-pref-help* **by** *fastforce*

**have** 7:  $(\exists k \leq \text{ilen } \sigma - \text{Suc } 0.$

$g (\text{prefix } k (\text{prefix } (\text{ilen } \sigma - \text{Suc } 0) \sigma)) \wedge$

$(\forall j. k < j \wedge j \leq \text{ilen } \sigma - \text{Suc } 0 \longrightarrow f (\text{prefix } j (\text{prefix } (\text{ilen } \sigma - \text{Suc } 0) \sigma))))$

**using** 4 5 6 **by** *blast*

**have** 8:  $0 < \text{ilen } \sigma \wedge$

$(\exists k \leq \text{ilen } \sigma - \text{Suc } 0.$

$g (\text{prefix } k (\text{prefix } (\text{ilen } \sigma - \text{Suc } 0) \sigma)) \wedge$

$(\forall j. k < j \wedge j \leq \text{ilen } \sigma - \text{Suc } 0 \longrightarrow f (\text{prefix } j (\text{prefix } (\text{ilen } \sigma - \text{Suc } 0) \sigma))))$

**using** 1 7 **by** *blast*

**from** 8 **show** *?thesis* **by** (*simp add: min.absorb1 itl-defs since-d-def*)

**qed**

**lemma** *SincePrevSincesemb:*

**assumes**  $\sigma \models \text{prev} (f \mathcal{S} g)$

**shows**  $0 < \text{ilen } \sigma \wedge$

$(\exists k < \text{ilen } \sigma. g (\text{prefix } k \sigma) \wedge (\forall j. k < j \wedge j < \text{ilen } \sigma \longrightarrow f (\text{prefix } j \sigma)))$

**proof** –

**have** 1:  $0 < \text{ilen } \sigma \wedge$

$(\exists k \leq \text{ilen } \sigma - \text{Suc } 0.$

$g (\text{prefix } k (\text{prefix } (\text{ilen } \sigma - \text{Suc } 0) \sigma)) \wedge$

$(\forall j. k < j \wedge j \leq \text{ilen } \sigma - \text{Suc } 0 \longrightarrow f (\text{prefix } j (\text{prefix } (\text{ilen } \sigma - \text{Suc } 0) \sigma))))$

**using** *assms* **by** (*simp add: min.absorb1 itl-defs since-d-def*)

**have** 2:  $(\exists k \leq \text{ilen } \sigma - \text{Suc } 0.$

$g (\text{prefix } k (\text{prefix } (\text{ilen } \sigma - \text{Suc } 0) \sigma)) \wedge$

$(\forall j. k < j \wedge j \leq \text{ilen } \sigma - \text{Suc } 0 \longrightarrow f (\text{prefix } j (\text{prefix } (\text{ilen } \sigma - \text{Suc } 0) \sigma))))$

**using** 1 **by** *auto*

**obtain**  $k$  **where**  $3$ :  $k \leq \text{ilen } \sigma - \text{Suc } 0 \wedge g (\text{prefix } k (\text{prefix } (\text{ilen } \sigma - \text{Suc } 0) \sigma)) \wedge$   
 $(\forall j. k < j \wedge j \leq \text{ilen } \sigma - \text{Suc } 0 \longrightarrow f (\text{prefix } j (\text{prefix } (\text{ilen } \sigma - \text{Suc } 0) \sigma)))$   
**using**  $2$  **by** *auto*  
**have**  $4$ :  $k < \text{ilen } \sigma$   
**using**  $1$   $3$  **by** *linarith*  
**have**  $5$ :  $g (\text{prefix } k \sigma)$   
**using**  $3$   $4$  *pref-pref-help*[*of*  $\sigma$ ] **by** *force*  
**have**  $6$ :  $(\forall j. k < j \wedge j < \text{ilen } \sigma \longrightarrow f (\text{prefix } j \sigma))$   
**using**  $3$  *pref-pref-help*[*of*  $\sigma$ ] **by** *fastforce*  
**have**  $7$ :  $(\exists k < \text{ilen } \sigma. g (\text{prefix } k \sigma) \wedge (\forall j. k < j \wedge j < \text{ilen } \sigma \longrightarrow f (\text{prefix } j \sigma)))$   
**using**  $4$   $5$   $6$  **by** *blast*  
**show** *?thesis* **using**  $1$   $7$  **by** *blast*  
**qed**

**lemma** *SincePrevSincesem*:

$(\sigma \models \text{prev } (f \ \mathcal{S} \ g)) =$   
 $(0 < \text{ilen } \sigma \wedge$   
 $(\exists k < \text{ilen } \sigma. g (\text{prefix } k \sigma) \wedge (\forall j. k < j \wedge j < \text{ilen } \sigma \longrightarrow f (\text{prefix } j \sigma))))$   
**using** *SincePrevSincesema*[*of*  $\sigma$   $g$   $f$ ] *SincePrevSincesemb*[*of*  $f$   $g$   $\sigma$ ] *unl-lift2* **by** *blast*

**lemma** *SincePrevSincesem1*:

$(\sigma \models f \ \mathcal{S} \ g) = (\sigma \models (g \vee (f \wedge \text{prev}(f \ \mathcal{S} \ g))))$

**unfolding** *SincePrevSincesem*

**proof**

**assume**  $a$ :  $(\sigma \models f \ \mathcal{S} \ g)$   
**show**  $\sigma \models g \vee f \wedge (\lambda s. 0 < \text{ilen } s \wedge (\exists k < \text{ilen } s. g (\text{prefix } k s) \wedge$   
 $(\forall j. k < j \wedge j < \text{ilen } s \longrightarrow f (\text{prefix } j s))))$   
**using**  $a$  **unfolding** *since-d-def* **by** *simp*  
 $(\text{metis prefix-ilen-bound prefix-ilen le-neq-implies-less le-zero-eq less-imp-le-nat neq0-conv})$

**next**

**assume**  $b$ :  $\sigma \models g \vee f \wedge (\lambda s. 0 < \text{ilen } s \wedge (\exists k < \text{ilen } s. g (\text{prefix } k s) \wedge$   
 $(\forall j. k < j \wedge j < \text{ilen } s \longrightarrow f (\text{prefix } j s))))$   
**show**  $(\sigma \models f \ \mathcal{S} \ g)$   
**using**  $b$  **unfolding** *since-d-def* **by** *simp*  
 $(\text{metis prefix-ilen leD order-le-less})$

**qed**

**lemma** *SincePrevSince*:

$\vdash f \ \mathcal{S} \ g = (g \vee (f \wedge \text{prev}(f \ \mathcal{S} \ g)))$   
**by** (*simp add: SincePrevSincesem1 Valid-def*)

### 14.2.13 RevUntil

**lemma** *RevUntilsema*:

**assumes**  $\sigma \models (f \ \mathcal{U} \ g)^r$   
**shows**  $\sigma \models ((f^r) \ \mathcal{S} \ (g^r))$

**proof** –

**have**  $1$ :  $\sigma \models (f \ \mathcal{U} \ g)^r$   
**using** *assms* **by** *auto*  
**have**  $2$ :  $\exists k \leq \text{ilen } \sigma.$

$g (\text{suffix } k (\text{irev } \sigma)) \wedge (\forall j < k. f (\text{suffix } j (\text{irev } \sigma)))$   
**using 1 by** (*simp add: until-d-def reverse-d-def*)  
**obtain**  $k$  **where**  $\exists k \leq \text{ilen } \sigma \wedge g (\text{suffix } k (\text{irev } \sigma)) \wedge$   
 $(\forall j < k. f (\text{suffix } j (\text{irev } \sigma)))$   
**using 2 by** *auto*  
**have 4:**  $g (\text{irev } (\text{prefix } (\text{ilen } \sigma - k) \sigma))$   
**by** (*simp add: 3 irev-prefix*)  
**have 5:**  $(\forall j < k. f (\text{irev } (\text{prefix } (\text{ilen } \sigma - j) \sigma)))$   
**by** (*simp add: 3 irev-prefix*)  
**have 6:**  $(\text{ilen } \sigma - k) \leq \text{ilen } \sigma$   
**using** *diff-le-self* **by** *blast*  
**have 7:**  $(\forall j. (\text{ilen } \sigma - k) < j \wedge j \leq \text{ilen } \sigma \longrightarrow f (\text{irev } (\text{prefix } j \sigma)))$   
**by** (*simp add: 3 irev-prefix less-diff-conv2*)  
**have 8:**  $\exists k \leq \text{ilen } \sigma. g (\text{irev } (\text{prefix } k \sigma)) \wedge$   
 $(\forall j. k < j \wedge j \leq \text{ilen } \sigma \longrightarrow f (\text{irev } (\text{prefix } j \sigma)))$   
**using 4 6 7 by** *blast*  
**have 10:**  $\sigma \models (f^r) \mathcal{S} (g^r)$   
**using 8 by** (*simp add: since-d-def reverse-d-def*)  
**show** *?thesis* **by** (*simp add: 10*)  
**qed**

**lemma** *RevUntilsemb:*

**assumes**  $\sigma \models (f^r) \mathcal{S} (g^r)$

**shows**  $\sigma \models (f \mathcal{U} g)^r$

**proof** –

**have 1:**  $\sigma \models (f^r) \mathcal{S} (g^r)$   
**using** *assms* **by** *auto*  
**have 2:**  $\exists k \leq \text{ilen } \sigma. g (\text{irev } (\text{prefix } k \sigma)) \wedge$   
 $(\forall j. k < j \wedge j \leq \text{ilen } \sigma \longrightarrow f (\text{irev } (\text{prefix } j \sigma)))$   
**using 1 by** (*simp add: since-d-def reverse-d-def*)  
**obtain**  $k$  **where**  $\exists k \leq \text{ilen } \sigma \wedge g (\text{irev } (\text{prefix } k \sigma)) \wedge$   
 $(\forall j. k < j \wedge j \leq \text{ilen } \sigma \longrightarrow f (\text{irev } (\text{prefix } j \sigma)))$   
**using 2 by** *auto*  
**have 4:**  $g (\text{suffix } (\text{ilen } \sigma - k) (\text{irev } \sigma))$   
**using** *3 irev-prefix* **by** *fastforce*  
**have 5:**  $(\forall j. j < \text{ilen } \sigma - k \longrightarrow f (\text{suffix } j (\text{irev } \sigma)))$   
**using** *3 irev-prefix[of -  $\sigma$ ]*  
**by** (*metis (mono-tags, lifting) diff-diff-cancel diff-le-self diff-less-mono2 less-imp-le-nat less-le-trans*)  
**have 6:**  $\exists k \leq \text{ilen } \sigma. g (\text{suffix } k (\text{irev } \sigma)) \wedge$   
 $(\forall j < k. f (\text{suffix } j (\text{irev } \sigma)))$   
**using 4 5** *diff-le-self* **by** *blast*  
**have 7:**  $\sigma \models (f \mathcal{U} g)^r$   
**using 6 by** (*simp add: until-d-def reverse-d-def*)  
**show** *?thesis*  
**using 7 by** *blast*  
**qed**

**lemma** *RevUntilsem:*

$\sigma \models (f \mathcal{U} g)^r = (f^r) \mathcal{S} (g^r)$

**using** *RevUntilsema RevUntilsemb* **using** *unl-lift2* **by** *blast*

**lemma** *RevUntil*:

$\vdash (f \mathcal{U} g)^r = (f^r) \mathcal{S} (g^r)$

**using** *RevUntilsem Valid-def* **by** *blast*

#### 14.2.14 DiamondEqvTrueUntil

**lemma** *DiamondEqvTrueUntil*:

$\vdash \Diamond f = \#True \mathcal{U} f$

**by** (*simp add: Valid-def itl-defs until-d-def*)

#### 14.2.15 TrueUntillImpNotUntil

**lemma** *interval-suf-first*:

**assumes**  $(\exists i \leq \text{ilen } xs. f (\text{suffix } i \text{ } xs))$

**shows**  $(\exists i \leq \text{ilen } xs. f (\text{suffix } i \text{ } xs) \wedge$   
 $(\forall j. j < i \longrightarrow \neg (f (\text{suffix } j \text{ } xs))))$

**using** *assms suf-first-upto[of ilen xs + 1 f xs]* **by** (*simp add: discrete*)

**lemma** *NotSuffixFirst*:

**assumes**  $(\exists n \leq \text{ilen } xs. \neg f (\text{suffix } n \text{ } xs))$

**shows**  $(\exists n \leq \text{ilen } xs. \neg f (\text{suffix } n \text{ } xs) \wedge (\forall k. k < n \longrightarrow f (\text{suffix } k \text{ } xs)))$

**using** *assms interval-suf-first[of xs LIFT( $\neg f$ )]* **by** *auto*

**lemma** *NotSuffixFirst-upto*:

**assumes**  $(\exists i < k. \neg f (\text{suffix } i \text{ } xs))$

$k \leq \text{ilen } xs + 1$

**shows**  $(\exists i < k. \neg f (\text{suffix } i \text{ } xs) \wedge$   
 $(\forall j < i. f (\text{suffix } j \text{ } xs)))$

**using** *assms suf-first-upto[of k LIFT( $\neg f$ ) xs]* **by** *auto*

**lemma** *TrueUntillImpNotUntil*:

$\vdash \#True \mathcal{U} g \longrightarrow (\neg g) \mathcal{U} g$

**by** (*simp add: intI interval-suf-first until-d-def*)

#### 14.2.16 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:  $(\forall k. g (\text{suffix } k \text{ } \sigma) \longrightarrow k \leq \text{ilen } \sigma \longrightarrow (\exists j < k. \neg f (\text{suffix } j \text{ } \sigma))) \wedge$   
 $(\exists n \leq \text{ilen } \sigma. \neg f (\text{suffix } n \text{ } \sigma))$

**using** *assms* **by** (*simp add: wait-d-def until-d-def itl-defs*)

**have** 2:  $(\forall k. k \leq \text{ilen } \sigma \longrightarrow \neg g (\text{suffix } k \text{ } \sigma) \vee (\exists j < k. \neg f (\text{suffix } j \text{ } \sigma)))$

**using** 1 **by** *auto*

**have** 3:  $(\exists n \leq \text{ilen } \sigma. \neg f (\text{suffix } n \text{ } \sigma))$

**using** 1 **by** *auto*

**obtain** *n* **where** 4:  $n \leq \text{ilen } \sigma \wedge \neg f (\text{suffix } n \text{ } \sigma) \wedge$   
 $(\forall k < n. f (\text{suffix } k \text{ } \sigma))$

```

  using 3 using NotSuffixFirst by blast
have 16:  $n \leq \text{ilen } \sigma$ 
  by (simp add: 4)
have 17:  $\neg g (\text{suffix } n \sigma)$ 
  using 1 4 by blast
have 18:  $(\forall j < n. \neg g (\text{suffix } j \sigma))$ 
  by (meson 2 4 le-eq-less-or-eq less-le-trans)
have 19:  $\exists k \leq \text{ilen } \sigma. \neg f (\text{suffix } k \sigma) \wedge \neg g (\text{suffix } k \sigma) \wedge (\forall j < k. \neg g (\text{suffix } j \sigma))$ 
  using 16 17 18 4 by blast
have 20:  $(\sigma \models ((\neg g) \mathcal{U} (\neg f \wedge \neg g)))$ 
  using 19 by (simp add: until-d-def)
show ?thesis using 20 by auto
qed

```

```

lemma WaitNotDistUntilsem2:
assumes  $(\sigma \models ((\neg g) \mathcal{U} ((\neg f) \wedge (\neg g))))$ 
shows  $(\sigma \models \neg(f \mathcal{W} g))$ 
using assms not-less-iff-gr-or-eq by (auto simp add: itl-defs wait-d-def until-d-def)

```

```

lemma WaitNotDistUntilsem:
 $(\sigma \models (\neg(f \mathcal{W} g)) = ((\neg g) \mathcal{U} ((\neg f) \wedge (\neg g))))$ 
using WaitNotDistUntilsem1 WaitNotDistUntilsem2 unl-lift2 by blast

```

```

lemma WaitNotDistUntil:
 $\vdash (\neg(f \mathcal{W} g)) = ((\neg g) \mathcal{U} (\neg f \wedge \neg g))$ 
using WaitNotDistUntilsem Valid-def by blast

```

### 14.2.17 UntilInduction

```

lemma LFPUntilsem1:
assumes  $\forall n \leq \text{ilen } \sigma.$ 
   $(g (\text{suffix } n \sigma) \longrightarrow h (\text{suffix } n \sigma)) \wedge$ 
   $(f (\text{suffix } n \sigma) \wedge n < \text{ilen } \sigma \wedge h (\text{suffix } (\text{Suc } n) \sigma) \longrightarrow$ 
     $h (\text{suffix } n \sigma))$ 
   $k \leq \text{ilen } \sigma$ 
   $g (\text{suffix } k \sigma)$ 
   $\forall j < k. f (\text{suffix } j \sigma)$ 
shows  $h \sigma$ 
using assms
proof (induct k arbitrary:  $\sigma$ )
case 0
then show ?case by auto
next
case (Suc k)
then show ?case
  proof (cases  $\sigma$ )
  case (INil x1)
  then show ?thesis using Suc.prem1 Suc.hyps by (metis suffix.simps(1))
  next
  case (ICons x21 x22)

```

**then show** *?thesis*  
**using** *Suc.premis Suc.hyps*  
**by** (*metis Suc-less-eq suffix-suc suffix-zero*  
*ilen.simps(2) le-eq-less-or-eq not-less-eq-eq plus-1-eq-Suc zero-less-Suc*)  
**qed**  
**qed**

**lemma** *LFPUntilsem*:  
 $\sigma \models \Box((g \vee (f \wedge \bigcirc h)) \longrightarrow h) \longrightarrow (f \mathcal{U} g \longrightarrow h)$   
**using** *LFPUntilsem1* **by** (*simp add: itl-defs until-d-def, blast*)

**lemma** *LFPUntil*:  
 $\vdash \Box((g \vee (f \wedge \bigcirc h)) \longrightarrow h) \longrightarrow (f \mathcal{U} g \longrightarrow h)$   
**using** *LFPUntilsem Valid-def* **by** *blast*

**lemma** *UntilInduction-a*:  
 $\vdash \Box(f \longrightarrow ((\bigcirc f) \wedge g) \vee h) \longrightarrow (f \longrightarrow \Box g \vee g \mathcal{U} h)$   
**proof** –  
**have** 1:  $\vdash (\Box g \vee g \mathcal{U} h) = g \mathcal{W} h$   
**by** (*auto simp add: wait-d-def*)  
**have** 2:  $\vdash (f \longrightarrow \Box g \vee g \mathcal{U} h) = ( (\neg h) \mathcal{U} (\neg g \wedge \neg h) \longrightarrow \neg f )$   
**using** 1 *WaitNotDistUntil* **by** *fastforce*  
**have** 3:  $\vdash \Box( ((\neg g \wedge \neg h) \vee (\neg h \wedge \bigcirc(\neg f))) \longrightarrow \neg f ) \longrightarrow ( (\neg h) \mathcal{U} (\neg g \wedge \neg h) \longrightarrow \neg f )$   
**using** *LFPUntil* **by** *blast*  
**have** 4:  $\vdash (f \longrightarrow ((\bigcirc f) \wedge g) \vee h) \longrightarrow ( ((\neg g \wedge \neg h) \vee (\neg h \wedge \bigcirc(\neg f))) \longrightarrow \neg f )$   
**using** *NextImpNotNextNot[of f]* **by** *auto*  
**have** 5:  $\vdash \Box(f \longrightarrow ((\bigcirc f) \wedge g) \vee h) \longrightarrow \Box( ((\neg g \wedge \neg h) \vee (\neg h \wedge \bigcirc(\neg f))) \longrightarrow \neg f )$   
**using** 4 **by** (*rule ImpBoxRule*)  
**show** *?thesis*  
**using** 2 3 5 **by** *fastforce*  
**qed**

**lemma** *UntilInduction-b*:  
 $\vdash \Box(f \longrightarrow (\bigcirc f) \vee g) \longrightarrow (f \longrightarrow \Box f \vee f \mathcal{U} g)$   
**proof** –  
**have** 1:  $\vdash (\Box f \vee f \mathcal{U} g) = f \mathcal{W} g$   
**by** (*auto simp add: wait-d-def*)  
**have** 2:  $\vdash (f \longrightarrow \Box f \vee f \mathcal{U} g) = ( (\neg g) \mathcal{U} (\neg f \wedge \neg g) \longrightarrow \neg f )$   
**using** 1 *WaitNotDistUntil* **by** *fastforce*  
**have** 3:  $\vdash \Box( ((\neg f \wedge \neg g) \vee (\neg g \wedge \bigcirc(\neg f))) \longrightarrow \neg f ) \longrightarrow ( (\neg g) \mathcal{U} (\neg f \wedge \neg g) \longrightarrow \neg f )$   
**using** *LFPUntil* **by** *blast*  
**have** 4:  $\vdash (f \longrightarrow (\bigcirc f) \vee g) \longrightarrow ( ((\neg f \wedge \neg g) \vee (\neg g \wedge \bigcirc(\neg f))) \longrightarrow \neg f )$   
**using** *NextImpNotNextNot[of f]* **by** *auto*  
**have** 5:  $\vdash \Box(f \longrightarrow (\bigcirc f) \vee g) \longrightarrow \Box( ((\neg f \wedge \neg g) \vee (\neg g \wedge \bigcirc(\neg f))) \longrightarrow \neg f )$   
**using** 4 *BoxImpBoxRule* **by** *blast*  
**show** *?thesis*  
**using** 2 3 5 **by** *fastforce*  
**qed**

### 14.3 Theorems

**lemma** *NextFalseSUntil*:

$\vdash \bigcirc g = \#False \mathcal{U}^s g$

**proof** –

**have** 1:  $\vdash \#False \mathcal{U} g = g$

**using** *UntilNextUntil*[of *LIFT*( $\#False$ ) *g*] **by** *auto*

**show** *?thesis unfolding suntil-d-def using 1 inteq-reflection by force*

**qed**

**lemma** *PrevFalseSSince*:

$\vdash prev\ g = \#False \mathcal{S}^s g$

**proof** –

**have** 1:  $\vdash \#False \mathcal{S} g = g$

**using** *SincePrevSince*[of *LIFT*( $\#False$ ) *g*] **by** *auto*

**show** *?thesis unfolding ssince-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*:

$\vdash \Box f = \#False \mathcal{R} f$

**by** (*metis DiamondEqvTrueUntil EqvReverseReverse always-d-def int-simps(3) inteq-reflection release-d-def*)

**lemma** *RevSince*:

$\vdash (f \mathcal{S} g)^r = (f^r) \mathcal{U} (g^r)$

**proof** –

**have** 1:  $\vdash (f^r \mathcal{U} g^r)^r = (f^r)^r \mathcal{S} (g^r)^r$

**by** (*simp add: RevUntil*)

**show** *?thesis*

**by** (*metis 1 EqvReverseReverse inteq-reflection*)

**qed**

**lemma** *LFPSince*:

$\vdash bi((g \vee (f \wedge prev\ h)) \longrightarrow h) \longrightarrow (f \mathcal{S} g \longrightarrow h)$

**by** (*metis (no-types, lifting) LFPUntil RBiEqvBox RPrevEqvNext RevSince ReverseEqv all-rev-eq(3) inteq-reflection*)

**lemma** *UntilTrue*:

$\vdash f \mathcal{U} \#True$

**using** *UntilNextUntil* **by** *fastforce*

**lemma** *UntilIdempotent*:

$\vdash f \mathcal{U} f = f$

**using** *UntilNextUntil* **by** *fastforce*

**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 SinceImpSince:
  assumes  $\vdash f0 \longrightarrow f1$ 
     $\vdash g0 \longrightarrow g1$ 
  shows  $\vdash f0 \mathcal{S} g0 \longrightarrow f1 \mathcal{S} g1$ 
  using assms
  by (metis RevSince ReverseEqv UntilImpUntil all-rev-eq(3) inteq-reflection)

```

```

lemma SinceEqvSince:
  assumes  $\vdash f0 = f1$ 
     $\vdash g0 = g1$ 
  shows  $\vdash f0 \mathcal{S} g0 = f1 \mathcal{S} g1$ 
  proof -
    have 1:  $\vdash f0 \longrightarrow f1$ 
      using assms by auto
    have 2:  $\vdash g0 \longrightarrow g1$ 
      using assms by auto
    have 3:  $\vdash f0 \mathcal{S} g0 \longrightarrow f1 \mathcal{S} g1$ 
      using 1 2 SinceImpSince[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:  $\vdash f1 \mathcal{S} g1 \longrightarrow f0 \mathcal{S} g0$ 
      using 4 5 SinceImpSince[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)
  qed

```

**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 UntilAbsorp-c UntilRightAnd int-simps(17) 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* **by** *fastforce*

**have** 3:  $\vdash \Box f \longrightarrow \Box(h \vee g \wedge \bigcirc((f \wedge g) \mathcal{U} (f \wedge h)) \longrightarrow (f \wedge g) \mathcal{U} (f \wedge h))$

**using** 2 *BoxImpBoxRule* **by** *blast*

**have** 4:  $\vdash \Box f \longrightarrow (g \mathcal{U} h \longrightarrow (f \wedge g) \mathcal{U} (f \wedge h))$

**using** 1 3 *lift-imp-trans* **by** *blast*

**show** *?thesis* **using** 4 **by** *fastforce*

**qed**

**lemma** *UntilAndImp*:

$\vdash \Box f \wedge \Diamond g \longrightarrow f \mathcal{U} g$

**proof** –

**have** 1:  $\vdash \Diamond g = \#True \mathcal{U} g$

**by** (*simp add: DiamondEqvTrueUntil*)

**have** 2:  $\vdash \Box f \wedge \#True \mathcal{U} g \longrightarrow (f \wedge \#True) \mathcal{U} (f \wedge g)$

**using** *UntilAlwaysAndDist* **by** *blast*

**have** 3:  $\vdash (f \wedge \#True) \mathcal{U} (f \wedge g) = f \mathcal{U} (f \wedge g)$

**by** *simp*

**have** 4:  $\vdash f \mathcal{U} (f \wedge g) \longrightarrow (f \mathcal{U} f) \mathcal{U} g$

**by** (*simp add: UntilRightAnd*)

**have** 5:  $\vdash (f \mathcal{U} f) = f$

**by** (*simp add: UntilIdempotent*)

**have** 6:  $\vdash (f \mathcal{U} f) \mathcal{U} g = f \mathcal{U} g$

**by** (*simp add: 5 UntilEqvUntil*)

**show** *?thesis*

**by** (*metis 1 2 3 4 5 inteq-reflection lift-imp-trans*)

**qed**

**lemma** *UntilRightMono*:

$\vdash \Box(f \longrightarrow g) \longrightarrow (h \mathcal{U} f \longrightarrow h \mathcal{U} g)$

**proof** –

**have** 1:  $\vdash \Box(f \longrightarrow g) \wedge h \mathcal{U} f \longrightarrow ((f \longrightarrow g) \wedge h) \mathcal{U} ((f \longrightarrow g) \wedge f)$

**using** *UntilAlwaysAndDist* **by** *blast*

**have** 2:  $\vdash ((f \longrightarrow g) \wedge h) \mathcal{U} ((f \longrightarrow g) \wedge f) \longrightarrow h \mathcal{U} ((f \longrightarrow g) \wedge f)$

**by** (*meson Prop12 UntilImpUntil int-iffD2 lift-and-com*)

**have** 3:  $\vdash ((f \longrightarrow g) \wedge f) \longrightarrow g$

**by** *auto*

**have** 4:  $\vdash h \mathcal{U} ((f \longrightarrow g) \wedge f) \longrightarrow h \mathcal{U} g$

**by** (*simp add: 3 UntilImpUntil*)

**show** *?thesis*

by (meson 1 2 4 Prop09 lift-imp-trans)  
qed

**lemma** *UntilLeftMono*:

$\vdash \Box (f \longrightarrow g) \longrightarrow (f \mathcal{U} h \longrightarrow g \mathcal{U} h)$   
**proof** –  
**have** 1:  $\vdash \Box (f \longrightarrow g) \wedge f \mathcal{U} h \longrightarrow ((f \longrightarrow g) \wedge f) \mathcal{U} ((f \longrightarrow g) \wedge h)$   
 by (simp add: UntilAlwaysAndDist)  
**have** 2:  $\vdash ((f \longrightarrow g) \wedge f) \mathcal{U} ((f \longrightarrow g) \wedge h) \longrightarrow ((f \longrightarrow g) \wedge f) \mathcal{U} h$   
 by (meson Prop12 UntilLeftDistAnd)  
**have** 3:  $\vdash ((f \longrightarrow g) \wedge f) \longrightarrow g$   
 by auto  
**have** 4:  $\vdash ((f \longrightarrow g) \wedge f) \mathcal{U} h \longrightarrow g \mathcal{U} h$   
 by (simp add: 3 UntilImpUntil)  
**show** ?thesis  
 by (meson 1 2 4 Prop09 lift-imp-trans)  
 qed

**lemma** *UntilCatRule*:

$\vdash \Box ( (f \longrightarrow g \mathcal{U} h) \wedge (h \longrightarrow g \mathcal{U} i) ) \longrightarrow (f \longrightarrow (g \mathcal{U} i))$   
**proof** –  
**have** 1:  $\vdash \Box ( (f \longrightarrow g \mathcal{U} h) \wedge (h \longrightarrow g \mathcal{U} i) ) \longrightarrow \Box (f \longrightarrow g \mathcal{U} h)$   
 by (metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection)  
**have** 2:  $\vdash \Box ( (f \longrightarrow g \mathcal{U} h) \wedge (h \longrightarrow g \mathcal{U} i) ) \longrightarrow \Box (h \longrightarrow g \mathcal{U} i)$   
 by (metis BoxElim BoxEqvBoxBox BoxImpBoxRule Prop12 inteq-reflection)  
**have** 3:  $\vdash \Box (h \longrightarrow g \mathcal{U} i) \longrightarrow \Box (g \mathcal{U} h \longrightarrow g \mathcal{U} (g \mathcal{U} i))$   
 by (metis BoxEqvBoxBox BoxImpBoxRule UntilRightMono inteq-reflection)  
**have** 4:  $\vdash \Box (g \mathcal{U} h \longrightarrow g \mathcal{U} (g \mathcal{U} i)) \longrightarrow \Box (g \mathcal{U} h \longrightarrow g \mathcal{U} i)$   
 by (metis BoxEqvBoxBox UntilUntil int-iffD1 inteq-reflection)  
**have** 5:  $\vdash \Box (f \longrightarrow g \mathcal{U} h) \longrightarrow (f \longrightarrow g \mathcal{U} h)$   
 by (simp add: BoxElim)  
**have** 6:  $\vdash \Box (g \mathcal{U} h \longrightarrow g \mathcal{U} i) \longrightarrow (g \mathcal{U} h \longrightarrow g \mathcal{U} i)$   
 by (simp add: BoxElim)  
**have** 7:  $\vdash (f \longrightarrow g \mathcal{U} h) \wedge (g \mathcal{U} h \longrightarrow g \mathcal{U} i) \longrightarrow (f \longrightarrow g \mathcal{U} i)$   
 by auto  
**have** 8:  $\vdash \Box ( (f \longrightarrow g \mathcal{U} h) \wedge (h \longrightarrow g \mathcal{U} i) ) \longrightarrow$   
 $(f \longrightarrow g \mathcal{U} h) \wedge (g \mathcal{U} h \longrightarrow g \mathcal{U} i)$   
 using 1 2 3 4 5 6 by fastforce  
 from 7 8 **show** ?thesis by auto  
 qed

**lemma** *UntilStrengthen*:

$\vdash \Box ( (f \longrightarrow h) \wedge (g \longrightarrow i) ) \longrightarrow (f \mathcal{U} g \longrightarrow h \mathcal{U} i)$   
**proof** –  
**have** 11:  $\vdash \Box ( (f \longrightarrow h) \wedge (g \longrightarrow i) ) \longrightarrow \Box (f \longrightarrow h)$   
 by (meson BoxImpBoxRule Prop12 int-iffD2 lift-and-com)  
**have** 1:  $\vdash \Box ( (f \longrightarrow h) \wedge (g \longrightarrow i) ) \longrightarrow (f \mathcal{U} g \longrightarrow h \mathcal{U} g)$   
 using 11 UntilLeftMono[of f h g] by fastforce  
**have** 21:  $\vdash \Box ( (f \longrightarrow h) \wedge (g \longrightarrow i) ) \longrightarrow \Box (g \longrightarrow i)$

by (simp add: BoxImpBoxRule Prop01 Prop05)  
 have 2:  $\vdash \Box ((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (h \mathcal{U} g \longrightarrow h \mathcal{U} i)$   
 using 21 UntilRightMono[of g i h] by fastforce  
 have 3:  $\vdash \Box ((f \longrightarrow h) \wedge (g \longrightarrow i)) \longrightarrow (f \mathcal{U} g \longrightarrow h \mathcal{U} g) \wedge (h \mathcal{U} g \longrightarrow h \mathcal{U} i)$   
 using 1 2 by fastforce  
 have 4:  $\vdash (f \mathcal{U} g \longrightarrow h \mathcal{U} g) \wedge (h \mathcal{U} g \longrightarrow h \mathcal{U} i) \longrightarrow (f \mathcal{U} g \longrightarrow h \mathcal{U} i)$   
 by auto  
 from 3 4 show ?thesis by auto  
 qed

**lemma** *UntilInduction:*

$\vdash \Box (f \longrightarrow \neg g \wedge \bigcirc f) \longrightarrow (f \longrightarrow \neg(h \mathcal{U} g))$   
**proof** –  
 have 1:  $\vdash \Box (\neg g) \longrightarrow \neg(h \mathcal{U} g)$   
 by (simp add: UntilImpDiamond always-d-def)  
 have 15:  $\vdash (f \longrightarrow \neg g \wedge \bigcirc f) \longrightarrow (g \vee \bigcirc (\neg f) \longrightarrow \neg f)$   
 using NextImpNotNextNot[of f] by fastforce  
 have 16:  $\vdash (f \longrightarrow \neg g \wedge \bigcirc f) \longrightarrow (g \vee \#True \wedge \bigcirc (\neg f) \longrightarrow \neg f)$   
 using 15 by auto  
 have 2:  $\vdash \Box (f \longrightarrow \neg g \wedge \bigcirc f) \longrightarrow \Box (g \vee \#True \wedge \bigcirc (\neg f) \longrightarrow \neg f)$   
 using 16 BoxImpBoxRule by blast  
 have 3:  $\vdash \Box (f \longrightarrow \neg g \wedge \bigcirc f) \longrightarrow (\#True \mathcal{U} g \longrightarrow \neg f)$   
 using 2 LFPUntil[of g LIFT(#True) LIFT(\neg f)]  
 by fastforce  
 have 4:  $\vdash (\#True \mathcal{U} g \longrightarrow \neg f) \longrightarrow (f \longrightarrow \neg(\#True \mathcal{U} g))$   
 by auto  
 have 5:  $\vdash \neg(\#True \mathcal{U} g) = \Box (\neg g)$   
 using BoxEqvFalseRelease NotUntilRelease inteq-reflection by fastforce  
 from 5 4 3 1 show ?thesis by fastforce  
 qed

**lemma** *UntilBoxImp:*

$\vdash f \mathcal{U} (\Box g) \longrightarrow \Box (f \mathcal{U} g)$   
**proof** –  
 have 1:  $\vdash f \mathcal{U} \Box g \longrightarrow f \mathcal{U} g$   
 by (meson BoxElim BoxGen MP UntilRightMono)  
 have 2:  $\vdash \text{wnext } (f \mathcal{U} \Box g) = (\text{empty} \vee \bigcirc (f \mathcal{U} \Box g))$   
 by (meson WnextEqvEmptyOrNext)  
 have 3:  $\vdash \Box g = (g \wedge \text{wnext } (\Box g))$   
 by (metis (no-types) BoxEqvAndWnextBox)  
 have 4:  $\vdash f \mathcal{U} \Box g = (\Box g \vee f \wedge \bigcirc (f \mathcal{U} \Box g))$   
 by (meson UntilNextUntil)  
 have 5:  $\vdash g \wedge \text{wnext } (\Box g) \longrightarrow \text{empty} \vee \bigcirc (f \mathcal{U} \Box g)$   
 by (metis NextUntil Prop01 Prop05 Prop08 UntilIntro WnextEqvEmptyOrNext int-iffD1  
 inteq-reflection)  
 have 6:  $\vdash \text{more} \wedge f \mathcal{U} \Box g \longrightarrow \text{empty} \vee \bigcirc (f \mathcal{U} \Box g)$   
 using 5 3 4 by fastforce  
 have 7:  $\vdash \text{more} \wedge f \mathcal{U} \Box g \longrightarrow \bigcirc (f \mathcal{U} \Box g)$   
 using 2 6 WnextAndMoreEqvNext by fastforce  
 from 1 7 show ?thesis using BoxIntro[of LIFT(f \mathcal{U} (\Box g)) LIFT(f \mathcal{U} g)]



by auto  
qed

**lemma** *UntilBoxEqvBox*:

$\vdash f \mathcal{U} (\Box f) = \Box f$

**proof** –

**have** 1:  $\vdash f \mathcal{U} (\Box f) \longrightarrow \Box(f \mathcal{U} f)$

using *UntilBoxImp*[of f f] **by** auto

**have** 2:  $\vdash \Box(f \mathcal{U} f) = \Box f$

**by** (*simp add: BoxEqvBox UntilIdempotent*)

**have** 3:  $\vdash \Box f \longrightarrow f \mathcal{U} (\Box f)$

**by** (*simp add: UntilIntro*)

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

qed

**lemma** *UntilRightStrengthen*:

$\vdash f \mathcal{U} (g \wedge h) \longrightarrow f \mathcal{U} (g \mathcal{U} h)$

**by** (*meson BoxGen MP OrImpUntil UntilRightMono*)

**lemma** *UntilLeftStrengthen*:

$\vdash (f \wedge g) \mathcal{U} h \longrightarrow (f \mathcal{U} g) \mathcal{U} h$

**by** (*simp add: OrImpUntil UntilImpUntil*)

**lemma** *UntilLeftAndOrder*:

$\vdash (f \wedge g) \mathcal{U} h \longrightarrow f \mathcal{U} (g \mathcal{U} h)$

**by** (*metis Prop12 UntilIdempotent UntilImpUntil UntilIntro inteq-reflection*)

**lemma** *UntilFrameNext*:

$\vdash \Box f \longrightarrow (\bigcirc g \longrightarrow \bigcirc (f \mathcal{U} g))$

**by** (*simp add: NextImpNext Prop01 Prop05 Prop09 UntilIntro*)

**lemma** *UntilFrameDiamond*:

$\vdash \Box f \longrightarrow (\Diamond g \longrightarrow \Diamond (f \mathcal{U} g))$

**by** (*meson NowImpDiamond Prop09 UntilAndImp lift-imp-trans*)

**lemma** *UntilFrameBox*:

$\vdash \Box f \longrightarrow (\Box g \longrightarrow \Box (f \mathcal{U} g))$

**by** (*simp add: BoxAndBoxImpBoxRule OrImpUntil Prop09*)

**lemma** *UntilImpNot*:

$\vdash f \mathcal{U} g \longrightarrow (f \wedge \neg g) \mathcal{U} g$

**proof** –

**have** 1:  $\vdash f \mathcal{U} g \longrightarrow \Diamond g$

**by** (*simp add: UntilImpDiamond*)

**have** 2:  $\vdash \Diamond g = \# \text{True} \mathcal{U} g$

**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:  $\vdash ((\Box f \wedge \Box g) \wedge h) \longrightarrow ((\Box f \vee f \mathcal{U} h) \wedge (\Box g \vee g \mathcal{U} h))$

**by** (*simp add: intI*)

**have** 4:  $\vdash (f \mathcal{U} h \wedge g \mathcal{U} h) \longrightarrow ((\Box f \vee f \mathcal{U} h) \wedge (\Box g \vee g \mathcal{U} h))$   
**by** *auto*  
**have** 5:  $\vdash ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h)) \longrightarrow ((\Box f \vee f \mathcal{U} h) \wedge (\Box g \vee g \mathcal{U} h))$   
**using** 3 4 **by** *fastforce*  
**have** 6:  $\vdash \Box f \wedge \Box g \longrightarrow ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$   
**by** *auto*  
**have** 7:  $\vdash \Box f \wedge g \mathcal{U} h \longrightarrow ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$   
**by** (*metis* 2 *Prop05 Prop12 UntilAlwaysAndDist UntilLeftDistAnd inteq-reflection lift-imp-trans*)  
**have** 8:  $\vdash f \mathcal{U} h \wedge \Box g \longrightarrow ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$   
**by** (*metis Prop05 Prop08 Prop12 UntilAlwaysAndDist UntilAndDist UntilLeftDistAnd inteq-reflection lift-and-com*)  
**have** 9:  $\vdash f \mathcal{U} h \wedge g \mathcal{U} h \longrightarrow ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$   
**by** *auto*  
**have** 10:  $\vdash ((\Box f \vee f \mathcal{U} h) \wedge (\Box g \vee g \mathcal{U} h)) \longrightarrow ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$   
**using** 6 7 8 9 **by** *fastforce*  
**have** 11:  $\vdash ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h)) = ((\Box f \vee f \mathcal{U} h) \wedge (\Box g \vee g \mathcal{U} h))$   
**using** 5 10 **by** *auto*  
**have** 12:  $\vdash (\Box(f \wedge g) \vee (f \wedge g) \mathcal{U} h) = ((\Box f \wedge \Box g) \vee (f \mathcal{U} h \wedge g \mathcal{U} h))$   
**using** 1 2 **by** *fastforce*  
**show** *?thesis* **unfolding** *wait-d-def* **using** 11 12  
**by** (*meson Prop04 UntilIdempotent*)  
**qed**

**lemma** *WaitAndRule*:

$\vdash f \mathcal{W} g = (f \wedge \neg g) \mathcal{W} g$

**proof** –

**have** 1:  $\vdash (f \mathcal{W} g \wedge (\neg g) \mathcal{W} g) = (f \wedge \neg g) \mathcal{W} g$

**by** (*meson Prop11 WaitRightDistAnd*)

**have** 2:  $\vdash (\neg g) \mathcal{W} g$

**by** (*metis* (*no-types, 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** *WnextAlwaysEqvAlwaysWnext*:

$\vdash wnext (\Box f) = \Box (wnext f)$   
**by** (*metis (no-types, lifting) NextDiamondEqvDiamondNext always-d-def int-eq int-simps(4) wnext-d-def*)

**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* **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** *WaitExclMid*:

$\vdash f \mathcal{W} g \vee f \mathcal{W} (\neg g)$

**using** *WaitExpand*

**proof** —

**have** 1:  $\vdash f \mathcal{W} g = (g \vee f \wedge \text{wnext}(f \mathcal{W} g))$

**by** (*simp add: WaitExpand*)

**have** 2:  $\vdash f \mathcal{W} (\neg g) = ((\neg g) \vee f \wedge \text{wnext}(f \mathcal{W} (\neg g)))$

**by** (*simp add: WaitExpand*)

**have** 3:  $\vdash (f \mathcal{W} g \vee f \mathcal{W} (\neg g)) =$

$(g \vee f \wedge \text{wnext}(f \mathcal{W} g)) \vee ((\neg g) \vee f \wedge \text{wnext}(f \mathcal{W} (\neg g)))$

**using** 1 2 **by** *fastforce*

**from** 3 **show** *?thesis* **by** *fastforce*

**qed**

**lemma** *WaitleftZero*:

$\vdash \# \text{True} \mathcal{W} g$

**by** (*meson BoxGen BoxImpWait MP TrueW*)

**lemma** *WaitLeftDistOr*:

$\vdash f \mathcal{W} (g \vee h) = (f \mathcal{W} g \vee f \mathcal{W} h)$   
**proof** –  
**have** 1:  $\vdash f \mathcal{W} (g \vee h) = (\Box f \vee f \mathcal{U} (g \vee h))$   
**by** (*simp add: wait-d-def*)  
**have** 2:  $\vdash (f \mathcal{W} g \vee f \mathcal{W} h) = ((\Box f \vee f \mathcal{U} g) \vee (\Box f \vee f \mathcal{U} h))$   
**by** (*simp add: wait-d-def*)  
**have** 3:  $\vdash f \mathcal{U} (g \vee h) = (f \mathcal{U} g \vee f \mathcal{U} h)$   
**by** (*simp add: UntilOrDist*)  
**from** 1 2 3 **show** ?thesis **by** fastforce  
**qed**

**lemma** *WaitRightDistOr*:  
 $\vdash f \mathcal{W} h \vee g \mathcal{W} h \longrightarrow (f \vee g) \mathcal{W} h$   
**proof** –  
**have** 0:  $\vdash \Box g \longrightarrow \Box (f \vee g)$   
**by** (*simp add: BoxImpBoxRule intI*)  
**have** 1:  $\vdash \Box f \longrightarrow \Box (f \vee g)$   
**by** (*simp add: BoxImpBoxRule intI*)  
**have** 11:  $\vdash \Box f \vee \Box g \longrightarrow \Box (f \vee g)$   
**using** 0 1 *Prop02* **by** blast  
**have** 2:  $\vdash f \mathcal{W} h = (\Box f \vee f \mathcal{U} h)$   
**by** (*simp add: wait-d-def*)  
**have** 3:  $\vdash (f \vee g) \mathcal{W} h = (\Box (f \vee g) \vee (f \vee g) \mathcal{U} h)$   
**by** (*simp add: wait-d-def*)  
**have** 4:  $\vdash g \mathcal{W} h = (\Box g \vee g \mathcal{U} h)$   
**by** (*simp add: wait-d-def*)  
**have** 5:  $\vdash f \mathcal{U} h \vee g \mathcal{U} h \longrightarrow (f \vee g) \mathcal{U} h$   
**using** *UntilRightDistOr* **by** simp  
**have** 6:  $\vdash (f \mathcal{W} h \vee g \mathcal{W} h) = ((\Box f \vee \Box g) \vee (f \mathcal{U} h \vee g \mathcal{U} h))$   
**using** 2 4 **by** fastforce  
**from** 11 5 6 3 **show** ?thesis  
**by** (*meson BoxImpWait Prop02 Prop11 UntilImpWait lift-imp-trans*)  
**qed**

**lemma** *WaitOrRule*:  
 $\vdash f \mathcal{W} g = (f \vee g) \mathcal{W} g$   
**proof** –  
**have** 1:  $\vdash f \mathcal{W} g \longrightarrow (f \vee g) \mathcal{W} g$   
**by** (*metis (no-types, lifting) Prop03 Prop10 UntilAbsorp-a WaitNotDistUntil int-iffD1 int-simps(14) int-simps(32) int-simps(33) inteq-reflection*)  
**have** 2:  $\vdash (f \vee g) \mathcal{W} g \longrightarrow f \mathcal{W} g$   
**by** (*metis (no-types, lifting) Prop03 Prop10 WaitNotDistUntil int-iffD2 int-simps(14) int-simps(32) int-simps(33) inteq-reflection*)  
**from** 1 2 **show** ?thesis **by** fastforce  
**qed**

**lemma** *UntilOrRule*:  
 $\vdash f \mathcal{U} g = (f \vee g) \mathcal{U} g$   
**by** (*metis UntilWait WaitOrRule inteq-reflection*)

**lemma** *WaitRule*:

$\vdash (\neg f) \mathcal{W} f$

**by** (*metis* *BoxGen* *BoxImpWait* *MP* *WaitOrRule* *int-eq-true* *int-simps*(29) *inteq-reflection*)

**lemma** *UntilRule*:

$\vdash (\neg f) \mathcal{U} f = \Diamond f$

**using** *DiamondEqvTrueUntil* *UntilOrRule* *inteq-reflection* **by** *fastforce*

**lemma** *WaitImpRule*:

$\vdash (f \longrightarrow g) \mathcal{W} f$

**proof** –

**have** 1:  $\vdash (f \longrightarrow g) \mathcal{W} f = ((f \longrightarrow g) \vee f) \mathcal{W} f$

**by** (*simp* *add*: *WaitOrRule*)

**have** 2:  $\vdash (f \longrightarrow g) \vee f$

**by** *auto*

**have** 3:  $\vdash ((f \longrightarrow g) \vee f) \mathcal{W} f = \#True \mathcal{W} f$

**by** (*metis* 1 2 *int-eq-true* *inteq-reflection*)

**show** *?thesis*

**using** 1 3 *WaitleftZero* **by** *fastforce*

**qed**

**lemma** *DiamondUntilImpRule*:

$\vdash \Diamond f \longrightarrow (f \longrightarrow g) \mathcal{U} f$

**using** *UntilWait* *WaitImpRule* **by** *fastforce*

**lemma** *WaitNotDist*:

$\vdash (\neg (f \mathcal{W} g)) = (f \wedge \neg g) \mathcal{U} (\neg f \wedge \neg g)$

**proof** –

**have** 1:  $\vdash (\neg (f \mathcal{W} g)) = (\neg g) \mathcal{U} (\neg f \wedge \neg g)$

**using** *WaitNotDistUntil* **by** *blast*

**have** 2:  $\vdash (\neg g) \mathcal{U} (\neg f \wedge \neg g) = (\neg g \wedge \neg(\neg f \wedge \neg g)) \mathcal{U} (\neg f \wedge \neg g)$

**using** *UntilAndRule* **by** *blast*

**have** 3:  $\vdash (\neg g \wedge \neg(\neg f \wedge \neg g)) = (f \wedge \neg g)$

**by** *auto*

**have** 4:  $\vdash (\neg g \wedge \neg(\neg f \wedge \neg g)) \mathcal{U} (\neg f \wedge \neg g) = (f \wedge \neg g) \mathcal{U} (\neg f \wedge \neg g)$

**using** 3 *inteq-reflection* **by** *force*

**show** *?thesis* **using** 1 2 4 **by** *fastforce*

**qed**

**lemma** *UntilNotDist*:

$\vdash (\neg (f \mathcal{U} g)) = (f \wedge \neg g) \mathcal{W} (\neg f \wedge \neg g)$

**proof** –

**have** 1:  $\vdash (\neg (f \mathcal{U} g)) = (\neg g) \mathcal{W} (\neg f \wedge \neg g)$

**using** *UntilNotDistWait* **by** *blast*

**have** 2:  $\vdash (\neg g) \mathcal{W} (\neg f \wedge \neg g) = (\neg g \wedge \neg(\neg f \wedge \neg g)) \mathcal{W} (\neg f \wedge \neg g)$

**by** (*simp* *add*: *WaitAndRule*)

**have** 3:  $\vdash (\neg g \wedge \neg(\neg f \wedge \neg g)) = (f \wedge \neg g)$

**by** *auto*



**have** 4:  $\vdash (\neg g \wedge \neg(\neg f \wedge \neg g)) \mathcal{W} (\neg f \wedge \neg g) = (f \wedge \neg g) \mathcal{W} (\neg f \wedge \neg g)$   
**using** 3 *inteq-reflection* **by** *force*  
**show** ?thesis **using** 1 2 4 **by** *fastforce*  
**qed**

**lemma** *UntilDuala*:

$\vdash (\neg (\neg f) \mathcal{U} (\neg g))) = g \mathcal{W} (f \wedge g)$

**proof** –

**have** 1:  $\vdash (\neg (\neg f) \mathcal{U} (\neg g))) = (\neg f \wedge \neg (\neg g)) \mathcal{W} (\neg(\neg f) \wedge \neg (\neg g))$

**using** *UntilNotDist* **by** *blast*

**have** 2:  $\vdash (\neg f \wedge g) \mathcal{W} (f \wedge g) = g \mathcal{W} (f \wedge g)$

**using** 1 *UntilNotDistWait int-eq* **by** *fastforce*

**show** ?thesis

**using** 1 2 **by** *fastforce*

**qed**

**lemma** *UntilDualb*:

$\vdash (\neg (\neg f) \mathcal{U} (\neg g))) = (\neg f \wedge g) \mathcal{W} (f \wedge g)$

**proof** –

**have** 1:  $\vdash (\neg (\neg f) \mathcal{U} (\neg g))) = (\neg f \wedge \neg (\neg g)) \mathcal{W} (\neg(\neg f) \wedge \neg (\neg g))$

**using** *UntilNotDist* **by** *blast*

**show** ?thesis

**using** 1 **by** *auto*

**qed**

**lemma** *WaitDuala*:

$\vdash (\neg (\neg f) \mathcal{W} (\neg g))) = g \mathcal{U} (f \wedge g)$

**proof** –

**have** 1:  $\vdash (\neg (\neg f) \mathcal{W} (\neg g))) = (\neg f \wedge \neg (\neg g)) \mathcal{U} (\neg(\neg f) \wedge \neg (\neg g))$

**using** *WaitNotDist* **by** *blast*

**have** 2:  $\vdash (\neg f \wedge g) \mathcal{U} (f \wedge g) = g \mathcal{U} (f \wedge g)$

**using** 1 *WaitNotDistUntil int-eq* **by** *fastforce*

**show** ?thesis

**using** 1 2 **by** *fastforce*

**qed**

**lemma** *WaitDualb*:

$\vdash (\neg (\neg f) \mathcal{W} (\neg g))) = (\neg f \wedge g) \mathcal{U} (f \wedge g)$

**proof** –

**have** 1:  $\vdash (\neg (\neg f) \mathcal{W} (\neg g))) = (\neg f \wedge \neg (\neg g)) \mathcal{U} (\neg(\neg f) \wedge \neg (\neg g))$

**using** *WaitNotDist* **by** *blast*

**show** ?thesis **using** 1 **by** *auto*

**qed**

**lemma** *WaitIdempotent*:

$\vdash f \mathcal{W} f = f$

**by** (*meson BoxElim Prop02 Prop12 UntilIdempotent UntilImpWait UntilIntro WaitUntilb int-iffD1 int-iffI lift-imp-trans*)

**lemma** *WaitRightZero*:

$\vdash f \mathcal{W} \# \text{True}$

**by** (*meson MP TrueW UntilImpWait UntilIntro*)

**lemma** *WaitLeftIdentity*:

$\vdash \# \text{False} \mathcal{W} g = g$

**by** (*metis (no-types, lifting) UntilAbsorp-c UntilNotDistWait WaitDuala WaitIdempotent WaitSRelease int-eq int-simps(17) int-simps(3) srelease-d-def*)

**lemma** *WaitImpOr*:

$\vdash f \mathcal{W} g \longrightarrow f \vee g$

**by** (*metis Prop03 WaitIdempotent WaitLeftDistOr WaitOrRule inteq-reflection*)

**lemma** *BoxOrImpWait*:

$\vdash \Box(f \vee g) \longrightarrow f \mathcal{W} g$

**using** *BoxImpWait WaitOrRule* **by** *fastforce*

**lemma** *BoxImpImpWait*:

$\vdash \Box(\neg g \longrightarrow f) \longrightarrow f \mathcal{W} g$

**proof** –

**have** 1:  $\vdash (\neg g \longrightarrow f) = (f \vee g)$

**by** *auto*

**have** 2:  $\vdash \Box(\neg g \longrightarrow f) = \Box(f \vee g)$

**using** 1 *BoxEqvBox* **by** *blast*

**show** *?thesis* **using** 2 *BoxOrImpWait* **by** *fastforce*

**qed**

**lemma** *WaitInsertion*:

$\vdash g \longrightarrow f \mathcal{W} g$

**by** (*simp add: Prop05 UntilIntro wait-d-def*)

**lemma** *WaitFrameNext*:

$\vdash \Box f \longrightarrow (\bigcirc g \longrightarrow \bigcirc (f \mathcal{W} g))$

**by** (*simp add: NextImpNext Prop01 Prop05 Prop09 WaitInsertion*)

**lemma** *WaitFrameDiamond*:

$\vdash \Box f \longrightarrow (\Diamond g \longrightarrow \Diamond (f \mathcal{W} g))$

**by** (*simp add: DiamondImpDiamond Prop01 Prop05 Prop09 WaitInsertion*)

**lemma** *WaitFrameBox*:

$\vdash \Box f \longrightarrow (\Box g \longrightarrow \Box (f \mathcal{W} g))$

**by** (*meson BoxAndBoxImpBoxRule OrImpUntil Prop09 UntilImpWait lift-imp-trans*)

**lemma** *WaitInductiona*:

$\vdash \Box (f \longrightarrow (\bigcirc f \wedge g) \vee h) \longrightarrow (f \longrightarrow g \mathcal{W} h)$

**by** (*simp add: UntilInduction-a wait-d-def*)

**lemma** *WaitInductionb*:

$\vdash \Box (f \longrightarrow \bigcirc f \vee g) \longrightarrow (f \longrightarrow f \mathcal{W} g)$

**by** (*simp add: UntilInduction-b wait-d-def*)

**lemma** *WaitInductionc:*

$\vdash \Box(f \longrightarrow \bigcirc f) \longrightarrow (f \longrightarrow f \mathcal{W} g)$

**proof** –

**have** 1:  $\vdash (f \longrightarrow \bigcirc f) \longrightarrow (f \longrightarrow \text{wnext } f)$

**unfolding** *wnext-d-def* **using** *NextImpNotNextNot[of f]* **by** *auto*

**have** 2:  $\vdash \Box(f \longrightarrow \bigcirc f) \longrightarrow \Box(f \longrightarrow \text{wnext } f)$

**using** 1 *BoxImpBoxRule* **by** *blast*

**show** *?thesis* **by** (*meson* 2 *BoxImpWait BoxInduct Prop09 lift-imp-trans*)

**qed**

**lemma** *WaitInductiond:*

$\vdash \Box(f \longrightarrow g \wedge \bigcirc f) \longrightarrow (f \longrightarrow f \mathcal{W} g)$

**proof** –

**have** 1:  $\vdash (f \longrightarrow g \wedge \bigcirc f) \longrightarrow (f \longrightarrow \text{wnext } f)$

**unfolding** *wnext-d-def* **using** *NextImpNotNextNot[of f]* **by** *auto*

**have** 2:  $\vdash \Box(f \longrightarrow g \wedge \bigcirc f) \longrightarrow \Box(f \longrightarrow \text{wnext } f)$

**using** 1 *BoxImpBoxRule* **by** *blast*

**show** *?thesis* **by** (*meson* 2 *BoxImpWait BoxInduct Prop09 lift-imp-trans*)

**qed**

**lemma** *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:*

$\vdash (f \mathcal{W} g \vee (f \wedge g)) = f \mathcal{W} g$

**by** (*metis* (*no-types, lifting*) *BoxEqvBoxBox UntilAbsorp-a UntilAbsorp-d WaitAbsorptiona WaitLeftDistOr WaitOrRule inteq-reflection wait-d-def*)

**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** 1:  $\vdash \Box(f \longrightarrow g) \wedge \Box f \longrightarrow \Box g \vee g \mathcal{U} h$

**by** (*simp add: itl-defs intI*)

**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 inteq-reflection*)

**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

## 15 Pi operator

```
theory Pi
imports IFilter UntilSince
begin
```

This theory introduces the Pi operator [10, 7]. The Pi operator is defined in terms of the ifilter operator introduced in IFilter.thy. We prove the soundness of the rules and axiom system. The until operator from UntilSince.thy is used as there is a striking similarity of the expressiveness of the until and the Pi operator [7].

### 15.1 Definitions

**definition** *sfxfilt* :: *'a interval*  $\Rightarrow$  (*'a:: world*) *formula*  $\Rightarrow$  *'a interval interval*  
**where** *sfxfilt* *xs f* = (*ifilter* ( $\lambda$  *ys. ys*  $\models$  *f*) (*suffixes xs*))

**definition** *pfxfilt* :: *'a interval*  $\Rightarrow$  (*'a:: world*) *formula*  $\Rightarrow$  *'a interval interval*  
**where** *pfxfilt* *xs f* = (*ifilter* ( $\lambda$  *ys. ys*  $\models$  *f*) (*prefixes xs*))

**definition** *pifilt* :: *'a interval*  $\Rightarrow$  (*'a:: world*) *formula*  $\Rightarrow$  *'a interval*  
**where** *pifilt* *xs f* = (*imap* ( $\lambda$  *s. inth s 0*) (*sfxfilt xs f*))

**definition** *rpifilt* :: *'a interval*  $\Rightarrow$  (*'a:: world*) *formula*  $\Rightarrow$  *'a interval*  
**where** *rpifilt* *xs f* = (*imap* ( $\lambda$  *s. ilast s*) (*pfxfilt xs f*))

**definition** *pi-d* :: (*'a:: world*) *formula*  $\Rightarrow$  *'a formula*  $\Rightarrow$  *'a formula*  
**where** *pi-d* *F G*  $\equiv \lambda s. ( (\exists i \leq \text{ilen } s. (\text{suffix } i s) \models F) \wedge ( (\text{pifilt } s F) \models G ) )$

**definition** *rpi-d* :: (*'a:: world*) *formula*  $\Rightarrow$  *'a formula*  $\Rightarrow$  *'a formula*  
**where** *rpi-d* *F G*  $\equiv \lambda s. ( (\exists i \leq \text{ilen } s. (\text{prefix } i s) \models F) \wedge ( (\text{rpifilt } s F) \models G ) )$

**syntax**

*-pi-d* :: [*lift, lift*]  $\Rightarrow$  *lift*      ((*-*  $\Pi$  *-*) [84,84] 83)  
*-rpi-d* :: [*lift, lift*]  $\Rightarrow$  *lift*      ((*-*  $\Pi^p$  *-*) [84,84] 83)

**syntax** (*ASCII*)

*-pi-d* :: [*lift, lift*]  $\Rightarrow$  *lift*      ((*-*  $\Pi$  *-*) [84,84] 83)  
*-rpi-d* :: [*lift, lift*]  $\Rightarrow$  *lift*      ((*-*  $\Pi^p$  *-*) [84,84] 83)

**translations**

*-pi-d*  $\equiv$  *CONST pi-d*  
*-rpi-d*  $\equiv$  *CONST rpi-d*

**definition** *upi-d* :: (*'a:: world*) *formula*  $\Rightarrow$  *'a formula*  $\Rightarrow$  *'a formula*  
**where** *upi-d* *F G*  $\equiv \text{LIFT}(\neg(F \Pi (\neg G)))$

**syntax**

*-upi-d* :: [*lift, lift*]  $\Rightarrow$  *lift*      ((*-*  $\Pi^u$  *-*) [84,84] 83)

**syntax** (*ASCII*)

*-upi-d* :: [*lift*,*lift*]  $\Rightarrow$  *lift* ((- *UPI* -) [84,84] 83)

**translations**

*-upi-d*  $\Rightarrow$  *CONST upi-d*

## 15.2 Time reversal

**lemma** *pfilt-irev-exists*:

$(\exists k \leq \text{ilen } (\text{irev } \sigma). (\text{suffix } k (\text{irev } \sigma)) \models f) =$   
 $(\exists k \leq \text{ilen } \sigma. (\text{prefix } k \sigma) \models f^r)$

**by** (*simp add: reverse-d-def*)

(*metis diff-diff-cancel irev-prefix suffix-ilen-bound suffix-ilen*)

**lemma** *PiRevsem*:

$(\sigma \models (f \Pi g)^r) = (\sigma \models f^r \Pi^p g^r)$

**proof** –

**have** 1:  $(\sigma \models (f \Pi g)^r) =$   
 $((\exists i \leq \text{ilen } \sigma. f (\text{suffix } i (\text{irev } \sigma))) \wedge g (\text{pfilt } (\text{irev } \sigma) f))$

**by** (*simp add: pi-d-def reverse-d-def*)

**have** 2:  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i (\text{irev } \sigma))) =$   
 $(\exists k \leq \text{ilen } \sigma. (\text{prefix } k \sigma) \models f^r)$

**using** *pfilt-irev-exists* **by** *auto*

**have** 3:  $(\text{pfilt } (\text{irev } \sigma) f) = (\text{imap } (\lambda s. \text{inth } s 0) (\text{sfxfilt } (\text{irev } \sigma) f))$

**by** (*simp add: pfilt-def*)

**have** 4:  $(\text{sfxfilt } (\text{irev } \sigma) f) = (\text{ifilter } (\lambda ys. ys \models f) (\text{suffixes } (\text{irev } \sigma)))$

**by** (*simp add: sfxfilt-def*)

**have** 5:  $(\text{suffixes } (\text{irev } \sigma)) = \text{irev}(\text{imap } \text{irev } (\text{prefixes } \sigma))$

**using** *suffixes-irev* **by** *blast*

**have** 6:  $(\text{ifilter } (\lambda ys. ys \models f) (\text{suffixes } (\text{irev } \sigma))) =$   
 $(\text{ifilter } (\lambda ys. ys \models f) (\text{irev}(\text{imap } \text{irev } (\text{prefixes } \sigma))))$

**using** 5 **by** *auto*

**have** 7:  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i (\text{irev } \sigma))) \Rightarrow$   
 $(\text{ifilter } (\lambda ys. ys \models f) (\text{irev}(\text{imap } \text{irev } (\text{prefixes } \sigma)))) =$   
 $\text{irev } (\text{ifilter } (\lambda ys. ys \models f) (\text{imap } \text{irev } (\text{prefixes } \sigma)))$

**by** (*metis 5 ifilter-irev ilen-imap ilen-prefixes inth-iset inth-suffixes irev-ilen iset-irev*)

**have** 8:  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i (\text{irev } \sigma))) \Rightarrow$   
 $(\text{ifilter } (\lambda ys. ys \models f) (\text{imap } \text{irev } (\text{prefixes } \sigma))) =$   
 $\text{imap } \text{irev } (\text{ifilter } ((\lambda ys. ys \models f) \circ \text{irev}) ((\text{prefixes } \sigma)))$

**by** (*metis 5 ifilter-imap in-iset-suffixes iset-irev osfx-suffix*)

**have** 9:  $((\lambda ys. ys \models f) \circ \text{irev}) = (\lambda ys. ys \models f^r)$

**by** (*auto simp add: reverse-d-def*)

**have** 10:  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i (\text{irev } \sigma))) \Rightarrow$   
 $\text{irev } (\text{ifilter } (\lambda ys. ys \models f) (\text{imap } \text{irev } (\text{prefixes } \sigma))) =$   
 $\text{irev } (\text{imap } \text{irev } (\text{ifilter } (\lambda ys. ys \models f^r) ((\text{prefixes } \sigma))))$

**using** 8 9 **by** *auto*

**have** 11:  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i (\text{irev } \sigma))) \Rightarrow$   
 $(\text{pfilt } (\text{irev } \sigma) f) =$   
 $(\text{imap } (\lambda s. \text{inth } s 0) (\text{irev } (\text{imap } \text{irev } (\text{ifilter } (\lambda ys. ys \models f^r) ((\text{prefixes } \sigma))))))$

**by** (*simp add: 10 3 4 6 7*)  
**have 12:**  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i (\text{irev } \sigma))) \implies$   
 $(\text{imap } (\lambda s. \text{inth } s 0) (\text{irev } (\text{imap } \text{irev } (\text{ifilter } (\lambda ys. ys \models f^r) (\text{prefixes } \sigma)))) =$   
 $\text{irev } (\text{imap } (\lambda s. \text{inth } s 0) (\text{imap } \text{irev } (\text{ifilter } (\lambda ys. ys \models f^r) (\text{prefixes } \sigma))))$   
**by** (*simp add: irev-imap*)  
**have 13:**  $(\text{imap } (\lambda s. \text{inth } s 0) (\text{imap } \text{irev } (\text{ifilter } (\lambda ys. ys \models f^r) (\text{prefixes } \sigma)))) =$   
 $(\text{imap } (\lambda s. \text{ilast } s) (\text{ifilter } (\lambda ys. ys \models f^r) (\text{prefixes } \sigma)))$   
**by** (*auto simp add: irev-inth*)  
**have 14:**  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i (\text{irev } \sigma))) \implies$   
 $(\text{pfilt } (\text{irev } \sigma) f) =$   
 $\text{irev } (\text{imap } (\lambda s. \text{ilast } s) (\text{ifilter } (\lambda ys. ys \models f^r) (\text{prefixes } \sigma)))$   
**using 11 12 13 by auto**  
**have 15:**  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i (\text{irev } \sigma))) \implies$   
 $\text{irev } (\text{imap } (\lambda s. \text{ilast } s) (\text{ifilter } (\lambda ys. ys \models f^r) (\text{prefixes } \sigma))) =$   
 $\text{irev } (\text{rpfilt } \sigma (\text{LIFT}(f^r)))$   
**by** (*simp add: rpfilt-def pfxfilt-def*)  
**have 16:**  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i (\text{irev } \sigma))) \implies$   
 $((\text{pfilt } (\text{irev } \sigma) f) \models g) =$   
 $((\text{rpfilt } \sigma (\text{LIFT}(f^r))) \models g^r)$   
**by** (*metis (mono-tags, lifting) 14 15 reverse-d-def*)  
**have 17:**  $(\sigma \models f^r \Pi^p g^r) =$   
 $((\exists k \leq \text{ilen } \sigma. (\text{prefix } k \sigma) \models f^r) \wedge (\text{irev}(\text{rpfilt } \sigma (\text{LIFT}(f^r))) \models g))$   
**by** (*simp add: rpi-d-def reverse-d-def*)  
**show ?thesis**  
**using 1 14 15 17 2 by force**  
**qed**

**lemma PiRev:**

$\vdash (f \Pi g)^r = (f^r \Pi^p g^r)$

**by** (*simp add: PiRevsem Valid-def*)

### 15.3 Semantic Lemmas

**lemma sfxfilter-help:**

$(\exists ys \in \text{iset } (\text{suffixes } xs) . f ys) = (\exists i \leq \text{ilen } xs. f (\text{suffix } i xs))$

**using iset-suffixes-sfx by auto**

**lemma pfiltinit-help:**

$(\exists y \in \text{iset } (xs). w \langle y \rangle) = (\exists i \leq \text{ilen } xs. w \langle \text{inth } xs i \rangle)$

**by** (*metis inth-and-iset*)

**lemma sfxfilt-INil:**

$\text{sfxfilt } \langle x \rangle f = \langle \langle x \rangle \rangle$

**by** (*auto simp: sfxfilt-def*)

**lemma pfilt-INil:**

$\text{pfilt } \langle x \rangle f = \langle x \rangle$

**by** (*auto simp: pfilt-def sfxfilt-INil*)

**lemma** *sfxfilt-ICons*:

**shows**  $(sfxfilt (x \odot xs) f) =$   
 $(if (\exists i \leq len\ xs. f (suffix\ i\ xs))\ then$   
 $\quad (if\ f\ (x \odot xs)\ then$   
 $\quad\quad (x \odot xs) \odot (sfxfilt\ (xs)\ f)$   
 $\quad\quad else\ (sfxfilt\ xs\ f)\ )$   
 $else\ \langle x \odot xs \rangle)$

**proof** –

**have** 1:  $(sfxfilt (x \odot xs) f) =$   
 $ifilter\ (\lambda ys. f\ ys)\ (suffixes\ (x \odot xs))$

**by** (*simp add: sfxfilt-def*)

**have** 2:  $suffixes\ (x \odot xs) = (x \odot xs) \odot suffixes\ xs$

**by** *simp*

**have** 3:  $ifilter\ (\lambda ys. f\ ys)\ (suffixes\ (x \odot xs)) =$   
 $(if\ (\exists\ ys \in iset\ (suffixes\ xs). f\ ys)\ then$   
 $\quad (if\ f\ (x \odot xs)\ then\ (x \odot xs) \odot (ifilter\ (\lambda ys. f\ ys)\ (suffixes\ (xs)))$   
 $\quad\quad else\ (ifilter\ (\lambda ys. f\ ys)\ (suffixes\ (xs)))\ )$   
 $else\ \langle x \odot xs \rangle)$

**by** *auto*

**have** 4:  $(if\ (\exists\ ys \in iset\ (suffixes\ xs). f\ ys)\ then$   
 $\quad (if\ f\ (x \odot xs)\ then\ (x \odot xs) \odot (ifilter\ (\lambda ys. f\ ys)\ (suffixes\ (xs)))$   
 $\quad\quad else\ (ifilter\ (\lambda ys. f\ ys)\ (suffixes\ (xs)))\ )$   
 $else\ \langle x \odot xs \rangle) =$   
 $(if\ (\exists i \leq len\ xs. f (suffix\ i\ xs))\ then$   
 $\quad (if\ f\ (x \odot xs)\ then$   
 $\quad\quad (x \odot xs) \odot (sfxfilt\ (xs)\ f)$   
 $\quad\quad else\ (sfxfilt\ xs\ f)\ )$   
 $else\ \langle x \odot xs \rangle)$

**by** (*auto simp add: sfxfilt-def iset-suffixes-sfx*)

**show** ?thesis **by** (*simp add: 1 4*)

**qed**

**lemma** *pfilt-ICons*:

**shows**  $(pfilt (x \odot xs) f) =$   
 $(if\ (\exists i \leq len\ xs. f (suffix\ i\ xs))\ then$   
 $\quad (if\ f\ (x \odot xs)\ then$   
 $\quad\quad (x) \odot (pfilt\ (xs)\ f)$   
 $\quad\quad else\ (pfilt\ xs\ f)\ )$   
 $else\ \langle x \rangle)$

**proof** –

**have** 1:  $(pfilt (x \odot xs) f) = imap\ (\lambda s. (inth\ s\ 0))\ (sfxfilt\ (x \odot xs) f)$

**by** (*simp add: pfilt-def*)

**have** 2:  $imap\ (\lambda s. (inth\ s\ 0))\ (sfxfilt\ (x \odot xs) f) =$   
 $(if\ (\exists i \leq len\ xs. f (suffix\ i\ xs))\ then$   
 $\quad (if\ f\ (x \odot xs)\ then$   
 $\quad\quad imap\ (\lambda s. (inth\ s\ 0))((x \odot xs) \odot (sfxfilt\ (xs)\ f))$   
 $\quad\quad else\ imap\ (\lambda s. (inth\ s\ 0))\ (sfxfilt\ xs\ f)\ )$

*else imap (λs. (inth s 0)) ⟨x⊙xs⟩*)

**using** 1

**by** (*metis sfxfilt-ICons*)

**have** 3: *imap (λs. (inth s 0))((x⊙xs)⊙ (sfxfilt (xs) f)) =*  
*(x)⊙ (pifilt (xs) f)*

**by** (*simp add: pifilt-def*)

**have** 4: *imap (λs. (inth s 0)) ⟨x⊙xs⟩ = ⟨x⟩*

**by** *simp*

**have** 5: *imap (λs. (inth s 0)) (sfxfilt xs f) = (pifilt (xs) f)*

**by** (*simp add: pifilt-def*)

**have** 6: (*if (∃ i ≤ ilen xs. f (suffix i xs)) then*  
*(if f (x⊙xs) then*  
*imap (λs. (inth s 0))((x⊙xs)⊙ (sfxfilt (xs) f))*  
*else imap (λs. (inth s 0)) (sfxfilt xs f) )*  
*else imap (λs. (inth s 0)) ⟨x⊙xs⟩ =*  
*(if (∃ i ≤ ilen xs. f (suffix i xs)) then*  
*(if f (x⊙xs) then*  
*(x)⊙ (pifilt (xs) f)*  
*else (pifilt xs f) )*  
*else ⟨x⟩)*)

**using** 5 **by** *auto*

**show** ?thesis **by** (*simp add: 1 2 6*)

**qed**

**lemma** *sfxfilt-inth-ICons*:

*inth (sfxfilt (x⊙xs) f) j =*  
*(if (∃ i ≤ ilen xs. f (suffix i xs)) then*  
*(if f (x⊙xs) then*  
*(if j = 0 then (x⊙xs) else inth (sfxfilt (xs) f) (j-1))*  
*else inth (sfxfilt (xs) f) j)*  
*else (x⊙xs))*

**by** *simp*

(*metis inth.simps(1) One-nat-def inth-ICons-a inth-zero sfxfilt-ICons*)

**lemma** *pifilt-inth-ICons*:

*inth (pifilt (x ⊙ xs) f) i =*  
*(if (∃ i ≤ ilen xs. f (suffix i xs)) then*  
*(if f (x⊙xs) then*  
*(if i = 0 then x else inth (pifilt (xs) f) (i-1))*  
*else inth (pifilt (xs) f) i)*  
*else x)*

**by** (*metis inth.simps(1) One-nat-def Suc-pred inth-Suc inth-zero not-gr0 pifilt-ICons*)

**lemma** *sfxfilt-inth*:

**assumes** (∃ i ≤ ilen σ. (suffix i σ) ⊨ f)

*i ≤ ilen (sfxfilt σ f)*

**shows** (inth (sfxfilt σ f) i) ⊨ f

**using** *assms in-iset-suffixes osfx-suffix sfxfilter-inth*

**by** (*simp add: sfxfilt-def ifilter-inth-aa sfxfilter-help*)

**lemma** *pfilt-exists*:

**assumes**  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i \sigma))$

**shows**  $(\exists i \leq \text{ilen}(sfxfilt \sigma f). (\text{inth } (sfxfilt \sigma f) i) \models f)$

**using** *assms*

**using** *sfxfilt-inth* **by** *blast*

**lemma** *sfxfilt-pfilt-ilen*:

**assumes**  $(\exists i \leq \text{ilen } \sigma. (\text{suffix } i \sigma) \models f)$

**shows**  $\text{ilen } (pfilt \sigma f) = \text{ilen } (\text{imap } (\lambda s. \text{inth } s 0) (sfxfilt \sigma f))$

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

**lemma** *sfxfilt-pfilt-inth*:

**assumes**  $(\exists i \leq \text{ilen } \sigma. (\text{suffix } i \sigma) \models f)$

$j \leq \text{ilen } (pfilt \sigma f)$

**shows**  $\text{inth } (pfilt \sigma f) j = \text{inth } (\text{imap } (\lambda s. \text{inth } s 0) (sfxfilt \sigma f)) j$

**using** *assms*

**by** (*simp add: pfilt-def*)

**lemma** *sfxfilt-pfilt*:

**assumes**  $(\exists i \leq \text{ilen } \sigma. (\text{suffix } i \sigma) \models f)$

**shows**  $(\text{imap } (\lambda s. \text{inth } s 0) (sfxfilt \sigma f)) = pfilt \sigma f$

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

**lemma** *sfxfilt-ilen-bound*:

**assumes**  $(\exists i \leq \text{ilen } xs. f (\text{suffix } i xs))$

**shows**  $\text{ilen } (sfxfilt xs f) \leq \text{ilen } xs$

**using** *assms*

**by** (*simp add: sfxfilt-def*)

(*metis ilen-ifilter-le ilen-suffixes*)

**lemma** *pfilt-ilen-bound*:

**assumes**  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i \sigma))$

**shows**  $\text{ilen } (pfilt \sigma f) \leq \text{ilen } \sigma$

**using** *assms* **by** (*simp add: pfilt-def sfxfilt-ilen-bound*)

**lemma** *sfxfilt-ilen-inth-bound*:

**assumes**  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i \sigma))$

$j \leq \text{ilen } (sfxfilt \sigma f)$

**shows**  $\text{ilen } (\text{inth } (sfxfilt \sigma f) j) \leq \text{ilen } \sigma$

**using** *assms*

**by** (*simp add: sfxfilt-def sfxfilter-help sfxfilter-inth-bound*)

**lemma** *sfxfilt-pfilt-inth-suffix*:

**assumes**  $(\exists i \leq \text{ilen } \sigma. (\text{suffix } i \sigma) \models f)$

$j \leq \text{ilen } (pfilt \sigma f)$

**shows**  $\text{inth } (sfxfilt \sigma f) j = \text{suffix } (\text{ilen } \sigma - (\text{ilen } (\text{inth } (sfxfilt \sigma f) j))) \sigma$

**proof** –

**have 1:**  $\text{inth} (\text{sfxfilt } \sigma f) j = \text{inth} (\text{ifilter } f (\text{suffixes } \sigma)) j$   
**by** (*simp add: sfxfilt-def*)  
**have 2:**  $\text{suffix} (\text{ilen } \sigma - \text{ilen} (\text{inth} (\text{ifilter } f (\text{suffixes } \sigma)) j)) \sigma =$   
 $\text{suffix} (\text{ilen } \sigma - (\text{ilen} (\text{inth} (\text{sfxfilt } \sigma f) j))) \sigma$   
**by** (*simp add: sfxfilt-def*)  
**have 3:**  $j \leq \text{ilen} (\text{ifilter } f (\text{suffixes } \sigma))$   
**using** *assms* **by** (*simp add: pifilt-def sfxfilt-def*)  
**have 4:**  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i \sigma))$   
**using** *assms* **by** *auto*  
**have 5:**  $(\exists ys \in \text{iset} (\text{suffixes } \sigma). f ys)$   
**using** 4  
**using** *in-iset-suffixes osfx-suffix* **by** *blast*  
**have 6:**  $\text{inth} (\text{ifilter } f (\text{suffixes } \sigma)) j =$   
 $\text{suffix} (\text{ilen } \sigma - \text{ilen} (\text{inth} (\text{ifilter } f (\text{suffixes } \sigma)) j)) \sigma$   
  
**using** 3 5 *sfxfilter-inth-suffix*[*of*  $\sigma f j$ ] **by** *blast*  
**show** ?thesis **using** 1 6 **by** *auto*  
**qed**

**lemma** *pifilt-inth*:  
**assumes**  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i \sigma))$   
 $i \leq \text{ilen} (\text{pifilt } \sigma f)$   
**shows**  $(\exists k \leq \text{ilen } \sigma. \text{inth} (\text{pifilt } \sigma f) i = \text{inth } \sigma k)$   
**using** *assms sfxfilt-pifilt-inth-suffix*[*of*  $\sigma f i$ ]  
**by** (*simp add: pifilt-def*)  
 $(\text{metis diff-le-self ifirst-suffix inth-imap})$

**lemma** *sfxfilt-ilen-imp*:  
**assumes**  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i \sigma) \wedge g (\text{suffix } i \sigma))$   
**shows**  $\text{ilen} (\text{sfxfilt } \sigma (\text{LIFT}(f \wedge g))) \leq \text{ilen} (\text{sfxfilt } \sigma f)$   
**proof** –  
**have 1:**  $\text{ilen} (\text{sfxfilt } \sigma (\text{LIFT}(f \wedge g))) =$   
 $\text{ilen} (\text{ifilter } (\lambda ys. f ys \wedge g ys) (\text{suffixes } \sigma))$   
**by** (*simp add: sfxfilt-def*)  
**have 2:**  $\text{ilen} (\text{ifilter } f (\text{suffixes } \sigma)) = \text{ilen} (\text{sfxfilt } \sigma f)$   
**by** (*simp add: sfxfilt-def*)  
**have 3:**  $\exists x \in \text{iset} (\text{suffixes } \sigma). f x \wedge g x$   
**using** *assms* **by** *auto*  
**have 4:**  $\text{ilen} (\text{ifilter } (\lambda x. f x \wedge g x) (\text{suffixes } \sigma)) \leq \text{ilen} (\text{ifilter } f (\text{suffixes } \sigma))$   
**using** 3 *ifilter-ilen-imp*[*of*  $\text{suffixes } \sigma f g$ ] **by** *auto*  
**show** ?thesis **using** 1 2 4 **by** *auto*  
**qed**

**lemma** *pifilt-ilen-imp*:  
**assumes**  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i \sigma) \wedge g (\text{suffix } i \sigma))$   
**shows**  $\text{ilen} (\text{pifilt } \sigma (\text{LIFT}(f \wedge g))) \leq \text{ilen} (\text{pifilt } \sigma f)$   
**using** *assms*  
**by** (*simp add: sfxfilt-ilen-imp pifilt-def*)

**lemma** *interval-iset-sfxfilt* [*simp*]:



```

assumes ( $\exists i \leq \text{ilen } xs. f (\text{suffix } i \text{ } xs)$ )
shows ( $\text{iset } (\text{sfxfilt } xs \ f) = \{ys. ys \in \text{iset } (\text{suffixes } xs) \wedge f (ys)\}$ )
using assms
proof –
  have 1:  $\text{iset } (\text{sfxfilt } xs \ f) = \text{iset } (\text{ifilter } f (\text{suffixes } xs))$ 
    by (simp add: sfxfilt-def)
  have 2:  $\exists x \in \text{iset } (\text{suffixes } xs). f \ x$ 
    using assms using in-iset-suffixes osfx-suffix by blast
  have 3:  $\text{iset } (\text{ifilter } f (\text{suffixes } xs)) =$ 
     $\{ys. ys \in \text{iset } (\text{suffixes } xs) \wedge f \ ys\}$ 

    using 2 iset-ifilter[of suffixes xs f] by auto
  show ?thesis by (simp add: 1 3)
qed

```

```

lemma interval-subset-sfxfilt [simp]:
assumes ( $\exists i \leq \text{ilen } xs. f (\text{suffix } i \text{ } xs)$ )
shows ( $\text{iset } (\text{sfxfilt } xs \ f) \leq (\text{iset } (\text{sfxfilt } xs \ (\text{LIFT}(f \vee g)))$ )
proof –
  have 1:  $\exists x \in \text{iset } (\text{suffixes } xs). f \ x$ 
    using assms using in-iset-suffixes osfx-suffix by blast
  have 2:  $\text{iset } (\text{sfxfilt } xs \ f) = \text{iset } (\text{ifilter } f (\text{suffixes } xs))$ 
    by (simp add: sfxfilt-def)
  have 3:  $\text{iset } (\text{sfxfilt } xs \ (\text{LIFT}(f \vee g))) = \text{iset } (\text{ifilter } (\lambda x. f \ x \vee g \ x) (\text{suffixes } xs))$ 
    by (simp add: sfxfilt-def)
  have 4:  $\text{iset } (\text{ifilter } f (\text{suffixes } xs)) \leq \text{iset } (\text{ifilter } (\lambda x. f \ x \vee g \ x) (\text{suffixes } xs))$ 
    using 1 subset-ifilter[of suffixes xs f g] by auto
  show ?thesis using 2 3 4 by blast
qed

```

```

lemma interval-iset-pifilt [simp]:
assumes ( $\exists i \leq \text{ilen } xs. f (\text{suffix } i \text{ } xs)$ )
shows ( $\text{iset } (\text{pifilt } xs \ f) = \{(\text{inth } ys \ 0) \mid ys. ys \in \text{iset}(\text{suffixes } xs) \wedge f (ys)\}$ )
proof –
  have 1:  $(\text{iset } (\text{pifilt } xs \ f)) = (\text{iset } (\text{imap } (\lambda s. \text{inth } s \ 0) (\text{sfxfilt } xs \ f)))$ 
    by (simp add: pifilt-def)
  have 2:  $(\text{iset } (\text{imap } (\lambda s. \text{inth } s \ 0) (\text{sfxfilt } xs \ f))) =$ 
     $\{(\text{inth } ys \ 0) \mid ys. ys \in \text{iset } (\text{sfxfilt } xs \ f)\}$ 
    by (induction xs) auto
  have 3:  $\{(\text{inth } ys \ 0) \mid ys. ys \in \text{iset } (\text{sfxfilt } xs \ f)\} =$ 
     $\{(\text{inth } ys \ 0) \mid ys. ys \in \text{iset}(\text{suffixes } xs) \wedge f (ys)\}$ 

    using assms by auto
  show ?thesis using 1 2 3 by auto
qed

```

```

lemma interval-inth-sfxfilt-in-iset:
 $x \in \text{iset } (\text{sfxfilt } \sigma \ (\text{LIFT}(f \vee g))) =$ 

```

$(\exists i \leq \text{ilen } (\text{sfxfilt } \sigma (\text{LIFT}(f \vee g))). x = (\text{inth } (\text{sfxfilt } \sigma (\text{LIFT}(f \vee g))) i))$   
**by** (*metis inth-and-iset*)

**lemma** *sfxfilt-inth-or*:

**assumes**  $(\exists i \leq \text{ilen } \sigma. (\text{suffix } i \sigma) \models f)$

**shows**  $(\exists i \leq \text{ilen } (\text{sfxfilt } \sigma (\text{LIFT}(f \vee g))). (\text{inth } (\text{sfxfilt } \sigma (\text{LIFT}(f \vee g))) i) \models f)$

**proof** –

**have** 1:  $\exists x \in \text{iset } (\text{suffixes } \sigma). f x$

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

**have** 2:  $\text{sfxfilt } \sigma (\text{LIFT}(f \vee g)) = \text{ifilter } (\lambda x. f x \vee g x) (\text{suffixes } \sigma)$

**by** (*simp add: sfxfilt-def*)

**have** 3:  $\exists x \in \text{iset}(\text{ifilter } (\lambda x. f x \vee g x) (\text{suffixes } \sigma)). f x$

**using** 1 *ifilter-inth-or[of suffixes  $\sigma$   $f$   $g$ ]* **by** *auto*

**from** 2 3 **show** ?thesis **by** (*metis interval-inth-sfxfilt-in-iset*)

**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 ifilter-True*)

**lemma** *pifilt-init-INil*:

$(\text{pifilt } \langle x \rangle (\text{LIFT}(\text{init } w))) = \langle x \rangle$

**by** (*auto simp add: pifilt-def sfxfilt-def*)

**lemma** *pifilt-init-ICons*:

$(\text{pifilt } (x \odot xs) (\text{LIFT}(\text{init } w))) =$

$(\text{if } (\exists i \leq \text{ilen } xs. w \langle \text{inth } xs i \rangle) \text{ then}$

$(\text{if } w \langle x \rangle \text{ then } x \odot (\text{pifilt } xs (\text{LIFT}(\text{init } w)))$

$\text{else } (\text{pifilt } xs (\text{LIFT}(\text{init } w))))$

$\text{else } \langle x \rangle)$

**by** (*simp add: pifilt-def sfxfilt-def init-defs*)

(*metis ifirst-suffix in-iset-suffixes interval-sfx-1 osfx-suffix pifiltinit-help*)

**lemma** *PiStatesem*:

$(\sigma \models (\text{init } w) \Pi f) =$

$((\exists i \leq \text{ilen } \sigma. w \langle \text{inth } \sigma i \rangle) \wedge f (\text{pifilt } \sigma (\text{LIFT}(\text{init } w))))$

**by** (*simp add: pi-d-def init-defs*)

## 15.4 Soundness of Axioms

### 15.4.1 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*

#### 15.4.2 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*

#### 15.4.3 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*)

#### 15.4.4 PiTrueEqvDiamond

**lemma** *PiTrueEqvDiamond*:  
 $\vdash f \Pi \#True = \Diamond f$   
**by** (*simp add: Valid-def pi-d-def sometimes-defs*)

#### 15.4.5 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*

#### 15.4.6 UPiFalseEqvBoxNot:

**lemma** *UPiFalseEqvBoxNot*:  
 $\vdash f \Pi^u \#False = \Box (\neg f)$   
**by** (*simp add: Valid-def upi-d-def pi-d-def always-defs*)

#### 15.4.7 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:  $\forall n \leq \text{ilen } \sigma. f1 (\text{suffix } n \sigma) = f2 (\text{suffix } n \sigma)$   
**using** *assms* **by** (*simp add: always-defs*)  
**have** 2:  $(\sigma \models (f1 \Pi g)) = ((\exists i \leq \text{ilen } \sigma. f1 (\text{suffix } i \sigma)) \wedge g (\text{pifilt } \sigma f1))$   
**by** (*simp add: pi-d-def*)

```

have 3: ( $\exists i \leq \text{ilen } \sigma. f1 (\text{suffix } i \sigma)$ ) = ( $\exists i \leq \text{ilen } \sigma. f2 (\text{suffix } i \sigma)$ )
  using 1 by blast
have 4: ( $\text{sfxfilt } \sigma f1$ ) = ( $\text{sfxfilt } \sigma f2$ )
  using 1
proof (induct  $\sigma$ )
case (INil x)
then show ?case by (simp add: sfxfilt-INil)
next
case (ICons x1a  $\sigma$ )
then show ?case
proof -
  have 41: ( $\text{sfxfilt } (x1a \odot \sigma) f1$ ) =
    (if ( $\exists i \leq \text{ilen } \sigma. (\text{suffix } i \sigma) \models f1$ ) then
      (if ( $(x1a \odot \sigma) \models f1$ ) then  $(x1a \odot \sigma) \odot (\text{sfxfilt } \sigma f1)$ 
        else ( $\text{sfxfilt } \sigma f1$ ))
      else  $\langle x1a \odot \sigma \rangle$ )
    using sfxfilt-ICons by blast
  have 42:  $\text{sfxfilt } \sigma f1 = \text{sfxfilt } \sigma f2$ 
    by (metis ICons.hyps ICons.premys Suc-le-mono suffix-suc ilen.simps(2)
      plus-1-eq-Suc)
  have 43: ( $(x1a \odot \sigma) \models f1$ ) = ( $(x1a \odot \sigma) \models f2$ )
    using ICons.premys by auto
  have 44: ( $\exists i \leq \text{ilen } \sigma. (\text{suffix } i \sigma) \models f1$ ) =
    ( $\exists i \leq \text{ilen } \sigma. (\text{suffix } i \sigma) \models f2$ )
    by (metis Suc-le-mono suffix-suc ilen.simps(2) ICons(2) plus-1-eq-Suc)
  have 45: (if ( $\exists i \leq \text{ilen } \sigma. (\text{suffix } i \sigma) \models f1$ ) then
    (if ( $(x1a \odot \sigma) \models f1$ ) then  $(x1a \odot \sigma) \odot (\text{sfxfilt } \sigma f1)$ 
      else ( $\text{sfxfilt } \sigma f1$ ))
    else  $\langle x1a \odot \sigma \rangle$ ) =
    (if ( $\exists i \leq \text{ilen } \sigma. (\text{suffix } i \sigma) \models f2$ ) then
      (if ( $(x1a \odot \sigma) \models f2$ ) then  $(x1a \odot \sigma) \odot (\text{sfxfilt } \sigma f2)$ 
        else ( $\text{sfxfilt } \sigma f2$ ))
      else  $\langle x1a \odot \sigma \rangle$ )
    using 42 43 44 by auto
  have 46: (if ( $\exists i \leq \text{ilen } \sigma. (\text{suffix } i \sigma) \models f2$ ) then
    (if ( $(x1a \odot \sigma) \models f2$ ) then  $(x1a \odot \sigma) \odot (\text{sfxfilt } \sigma f2)$ 
      else ( $\text{sfxfilt } \sigma f2$ ))
    else  $\langle x1a \odot \sigma \rangle$ ) = ( $\text{sfxfilt } (x1a \odot \sigma) f2$ )
    by (simp add: sfxfilt-ICons)
  show ?thesis
  using 41 45 46 by presburger
qed
qed
have 47: ( $\text{pifilt } \sigma f1$ ) = ( $\text{pifilt } \sigma f2$ )
  by (simp add: 4 pifilt-def)
have 5: ( $(\exists i \leq \text{ilen } \sigma. f1 (\text{suffix } i \sigma)) \wedge g (\text{pifilt } \sigma f1)$ ) =
  ( $(\exists i \leq \text{ilen } \sigma. f2 (\text{suffix } i \sigma)) \wedge g (\text{pifilt } \sigma f2)$ )
  by (simp add: 3 47)
show ?thesis by (simp add: 5 pi-d-def)
qed

```

qed

**lemma** *BoxEqvImpPiEqv*:

$\vdash \square (f1 = f2) \longrightarrow (f1 \Pi g = f2 \Pi g)$

**using** *BoxEqvImpPiEqvsem* **by** (*simp add: Valid-def, auto*)

#### 15.4.8 PiDiamondImpDiamond

**lemma** *PiDiamondImpDiamondsem*:

$\sigma \models f \Pi (\Diamond (init\ w)) \longrightarrow \Diamond (init\ w)$

**using** *pifilt-inth* **by** (*simp add: Valid-def pi-d-def sometimes-defs init-defs*) *fastforce*

**lemma** *PiDiamondImp*:

$\vdash f \Pi (\Diamond (init\ w)) \longrightarrow \Diamond (init\ w)$

**using** *PiDiamondImpDiamondsem* *Valid-def* **by** *blast*

#### 15.4.9 PiAssoc

**lemma** *PiAssocsem1*:

**assumes**  $i \leq \text{ilen } \sigma$

$f (\text{suffix } i\ \sigma)$

$ia \leq \text{ilen } (\text{pifilt } \sigma\ f)$

$w \langle \text{inth } (\text{pifilt } \sigma\ f)\ ia \rangle$

**shows**  $\exists i \leq \text{ilen } \sigma. f (\text{suffix } i\ \sigma) \wedge w \langle \text{inth } (\text{suffix } i\ \sigma)\ 0 \rangle$

**proof** –

**have** 1:  $(\text{inth } (\text{pifilt } \sigma\ f)\ ia) = (\text{inth } (\text{imap } (\lambda s. \text{inth } s\ 0)\ (\text{sfxfilt } \sigma\ f))\ ia)$

**using** *assms(1) assms(2) assms(3) sfxfilt-pifilt-inth* **by** *blast*

**have** 2:  $(\text{inth } (\text{imap } (\lambda s. \text{inth } s\ 0)\ (\text{sfxfilt } \sigma\ f))\ ia) =$

$(\lambda s. \text{inth } s\ 0) (\text{inth } (\text{sfxfilt } \sigma\ f)\ ia)$

**using** *inth-imap* **by** *auto*

**have** 3:  $f (\text{inth } (\text{sfxfilt } \sigma\ f)\ ia)$

**using** *sfxfilt-inth*

**by** (*metis assms(1) assms(2) assms(3) ilen-imap pifilt-def*)

**have** 4:  $\text{inth } (\text{sfxfilt } \sigma\ f)\ ia = \text{suffix } (\text{ilen } \sigma - \text{ilen } (\text{inth } (\text{sfxfilt } \sigma\ f)\ ia))\ \sigma$

**using** *sfxfilt-pifilt-inth-suffix assms(1) assms(2) assms(3)* **by** *blast*

**have** 5:  $w \langle \text{inth } (\text{suffix } (\text{ilen } \sigma - \text{ilen } (\text{inth } (\text{sfxfilt } \sigma\ f)\ ia))\ \sigma)\ 0 \rangle$

**using** 1 2 4 *assms(4)* **by** *auto*

**show** *?thesis*

**by** (*metis 3 4 5 diff-le-self*)

qed

**lemma** *PiAssocsem2*:

**assumes**  $i \leq \text{ilen } \sigma$

$f (\text{suffix } i\ \sigma)$

$w \langle \text{inth } \sigma\ i \rangle$

**shows**  $\exists j \leq \text{ilen } (\text{pifilt } \sigma\ f). w \langle \text{inth } (\text{pifilt } \sigma\ f)\ j \rangle$

**proof** –

**have** 1:  $\exists j \leq \text{ilen } (\text{sfxfilt } \sigma\ f). f (\text{inth } (\text{sfxfilt } \sigma\ f)\ j)$

**using** *assms pifilt-exists* **by** *blast*

**have** 2:  $(\text{LIFT } (\text{init } w)) (\text{suffix } i\ \sigma)$

**by** (*simp add: assms init-defs*)

**have** 3:  $\exists j \leq \text{ilen} \ (sfxfilt \ \sigma \ (LIFT(\text{init } w))). \ (LIFT(\text{init } w)) \ (inth \ (sfxfilt \ \sigma \ (LIFT(\text{init } w))) \ j)$   
**using** *pfilt-exists 2 assms by blast*  
**have** 4:  $(LIFT \ (f \wedge \text{init } w)) \ (suffix \ i \ \sigma)$   
**by** *(simp add: assms init-defs)*  
**have** 5:  $\exists j \leq \text{ilen} \ (sfxfilt \ \sigma \ (LIFT(f \wedge \text{init } w))).$   
 $(LIFT(f \wedge \text{init } w)) \ (inth \ (sfxfilt \ \sigma \ (LIFT(f \wedge \text{init } w))) \ j)$   
**using** *pfilt-exists 4 assms by blast*  
**have** 6:  $\exists i \leq \text{ilen} \ \sigma. \ suffix \ i \ \sigma \models f \wedge \text{init } w$   
**using** *4 assms by blast*  
**have** 7:  $\exists j \leq \text{ilen} \ (sfxfilt \ \sigma \ (LIFT((f \wedge \text{init } w) \vee (f \wedge \neg(\text{init } w))) \ )).$   
 $(LIFT(f \wedge \text{init } w)) \ (inth \ (sfxfilt \ \sigma \ (LIFT((f \wedge \text{init } w) \vee (f \wedge \neg(\text{init } w))) \ ) \ j)$   
**using** *6 sfxfilt-inth-or[of  $\sigma$   $LIFT(f \wedge \text{init } w)$   $LIFT(f \wedge \neg(\text{init } w))$ ]*  
**by** *auto*  
**have** 8:  $\bigwedge \sigma. \ (\sigma \models ((f \wedge \text{init } w) \vee (f \wedge \neg(\text{init } w)))) = (\sigma \models f)$   
**by** *auto*  
**have** 9:  $(sfxfilt \ \sigma \ (LIFT((f \wedge \text{init } w) \vee (f \wedge \neg(\text{init } w))) \ )) =$   
 $(sfxfilt \ \sigma \ f)$   
**using** *8 by (simp add: sfxfilt-def)*  
**have** 10:  $\exists j \leq \text{ilen} \ (sfxfilt \ \sigma \ f).$   
 $(LIFT(f \wedge \text{init } w)) \ (inth \ (sfxfilt \ \sigma \ f) \ j)$   
**using** *7 9 by auto*  
**have** 11:  $\text{ilen} \ (sfxfilt \ \sigma \ f) = \text{ilen} \ (pfilt \ \sigma \ f)$   
**by** *(simp add: pfilt-def)*  
**have** 12:  $\exists j \leq \text{ilen} \ (pfilt \ \sigma \ f).$   
 $(LIFT(\text{init } w)) \ (inth \ (sfxfilt \ \sigma \ f) \ j)$   
**using** *10 11 by auto*  
**from** 12 11 **show** *?thesis*  
**by** *(simp add: init-defs)*  
*(metis inth-imap pfilt-def)*  
**qed**

**lemma** *PiAssocsema:*

$((\exists i \leq \text{ilen} \ \sigma. \ f \ (suffix \ i \ \sigma)) \wedge$   
 $(\exists i \leq \text{ilen} \ (pfilt \ \sigma \ f). \ w \ \langle inth \ (suffix \ i \ (pfilt \ \sigma \ f)) \ 0 \rangle)) =$   
 $(\exists i \leq \text{ilen} \ \sigma. \ f \ (suffix \ i \ \sigma) \wedge w \ \langle inth \ (suffix \ i \ \sigma) \ 0 \rangle)$

**using** *PiAssocsem1 PiAssocsem2 by fastforce*

**lemma** *PiAssocsemb:*

$((\exists i \leq \text{ilen} \ \sigma. \ f \ (suffix \ i \ \sigma)) \wedge$   
 $(\exists i \leq \text{ilen} \ (pfilt \ \sigma \ f). \ (LIFT(\text{init } w)) \ (suffix \ i \ (pfilt \ \sigma \ f)) \ )) =$   
 $(\exists i \leq \text{ilen} \ \sigma. \ (LIFT(f \wedge \text{init } w)) \ (suffix \ i \ \sigma))$

**using** *PiAssocsem1 PiAssocsem2*

**by** *(simp add: init-defs) fastforce*

**lemma** *pfilt-INil-help:*

$(\exists \ x \in \text{iset} \ (suffixes \ xs). \ (LIFT(\text{init } w)) \ x) = (\exists \ x \in \text{iset} \ xs. \ w \ (\langle x \rangle))$

**proof** *(auto simp add: init-defs)*

**show**  $\bigwedge x. \ osfx \ x \ xs \implies w \ \langle inth \ x \ 0 \rangle \implies \exists x \in \text{iset} \ xs. \ w \ \langle x \rangle$

**using** *in-iset-suffixes interval-sfx-1 by blast*

**show**  $\bigwedge x. \ x \in \text{iset} \ xs \implies w \ \langle x \rangle \implies \exists x \in \text{iset} \ (suffixes \ xs). \ w \ \langle inth \ x \ 0 \rangle$

by (metis in-iset-suffixes ifirst-suffix inth-and-iset osfx-suffix)  
qed

**lemma** *pfilt-init*:

**assumes**  $(\exists i \leq \text{ilen } xs. (\text{LIFT}(\text{init } w)) (\text{suffix } i \text{ } xs))$   
**shows**  $(\text{pfilt } xs (\text{LIFT}(\text{init } w))) = \text{ifilter } (\lambda y. w (\langle y \rangle)) \text{ } xs$   
**using** *assms*  
**proof** (induct *xs*)  
**case** (INil *x*)  
**then show** ?case  
**by** (simp add: pfilt-init-INil)  
**next**  
**case** (ICons *x1a xs*)  
**then show** ?case  
**proof** –  
**have** 1:  $\text{pfilt } (x1a \odot xs) (\text{LIFT}(\text{init } w)) =$   
 $\text{imap } (\lambda xs. (\text{inth } xs \text{ } 0)) (\text{sfxfilt } (x1a \odot xs) (\text{LIFT}(\text{init } w)))$   
  
**using** *sfxfilt-pfilt* **by** (simp add: pfilt-def)  
**have** 2:  $\text{sfxfilt } (x1a \odot xs) (\text{LIFT}(\text{init } w)) =$   
 $(\text{ifilter } (\lambda ys. (\text{LIFT}(\text{init } w)) \text{ } ys) (\text{suffixes } (x1a \odot xs)))$   
  
**using** *sfxfilt-def* **by** blast  
**have** 3:  $\text{suffixes } (x1a \odot xs) = (x1a \odot xs) \odot (\text{suffixes } xs)$   
**by** simp  
**have** 4:  $(\text{ifilter } (\lambda ys. (\text{LIFT}(\text{init } w)) \text{ } ys) (\text{suffixes } (x1a \odot xs))) =$   
 $(\text{if } (\exists x \in \text{iset } (\text{suffixes } xs) . (\text{LIFT}(\text{init } w)) \text{ } x) \text{ then}$   
 $(\text{if } (\text{LIFT}(\text{init } w)) (x1a \odot xs) \text{ then } (x1a \odot xs) \odot (\text{ifilter } (\text{LIFT}(\text{init } w)) (\text{suffixes } xs))$   
 $\text{else } (\text{ifilter } (\text{LIFT}(\text{init } w)) (\text{suffixes } xs)))$   
 $\text{else } \langle x1a \odot xs \rangle)$   
  
**by** simp  
**have** 5:  $\text{imap } (\lambda xs. (\text{inth } xs \text{ } 0)) (\text{ifilter } (\lambda ys. (\text{LIFT}(\text{init } w)) \text{ } ys) (\text{suffixes } (x1a \odot xs))) =$   
 $(\text{if } (\exists x \in \text{iset } (\text{suffixes } xs) . (\text{LIFT}(\text{init } w)) \text{ } x) \text{ then}$   
 $(\text{if } (\text{LIFT}(\text{init } w)) (x1a \odot xs)$   
 $\text{then } (x1a) \odot \text{imap } (\lambda xs. (\text{inth } xs \text{ } 0)) (\text{ifilter } (\text{LIFT}(\text{init } w)) (\text{suffixes } xs))$   
 $\text{else } \text{imap } (\lambda xs. (\text{inth } xs \text{ } 0)) (\text{ifilter } (\text{LIFT}(\text{init } w)) (\text{suffixes } xs)))$   
 $\text{else } \langle x1a \rangle)$   
  
**by** auto  
**have** 6:  $\text{ifilter } (\lambda y. w (\langle y \rangle)) (x1a \odot xs) =$   
 $(\text{if } (\exists x \in \text{iset } xs. w (\langle x \rangle)) \text{ then}$   
 $(\text{if } w (\langle x1a \rangle) \text{ then } x1a \odot (\text{ifilter } (\lambda y. w (\langle y \rangle)) \text{ } xs) \text{ else } (\text{ifilter } (\lambda y. w (\langle y \rangle)) \text{ } xs))$   
 $\text{else } \langle x1a \rangle)$   
  
**by** simp  
**have** 61:  $(\exists x \in \text{iset } (\text{suffixes } xs) . (\text{LIFT}(\text{init } w)) \text{ } x) =$   
 $(\exists x \in \text{iset } xs. w (\langle x \rangle))$

```

    by (auto simp: init-defs interval-sfx-1)
      (metis init-defs prefix-zero-ifirst pifilt-INil-help)
  have 62: (LIFT(init w)) (x1a⊙xs) = w (⟨x1a⟩)
    by (simp add: init-defs)
  have 63: (∃ x ∈ iset xs. w (⟨x⟩)) ⟶
    imap (λxs . (inth xs 0)) (ifilter (LIFT(init w)) (suffixes xs)) =
    (ifilter (λy. w ⟨y⟩) xs)
    by (auto simp add: inth-and-iset )
      (metis ICons.hyps interval.distinct(1) pifilt-ICons pifilt-def pifilt-init-ICons sfxfilt-def)
  have 7: (if (∃ x ∈ iset (suffixes xs) . (LIFT(init w)) x) then
    (if (LIFT(init w)) (x1a⊙xs)
      then (x1a) ⊙ imap (λxs . (inth xs 0)) (ifilter (LIFT(init w)) (suffixes xs))
      else imap (λxs . (inth xs 0)) (ifilter (LIFT(init w)) (suffixes xs)))
    else ⟨x1a⟩) =
    (if (∃ x ∈ iset xs. w (⟨x⟩)) then
      (if w (⟨x1a⟩) then x1a⊙ (ifilter (λy. w ⟨y⟩) xs) else (ifilter (λy. w ⟨y⟩) xs))
      else ⟨x1a⟩)

    by (simp add: 61 62 63)
  show ?thesis using 1 2 5 7 by auto
qed
qed

```

**lemma** *pifilt-init-a*:

```

  assumes (∃ i ≤ ilen xs. w ⟨ inth xs i ⟩)
  shows (pifilt xs (λs. w ⟨ inth s 0 ⟩) ) = ifilter (λy. w (⟨y⟩)) xs
  using assms pifilt-init by (auto simp add: init-defs pifilt-def sfxfilt-def)

```

**lemma** *pifilt-pifilt* :

```

  assumes (∃ i ≤ ilen xs. f (suffix i xs))
    (∃ i ≤ ilen (pifilt xs f). w ⟨ inth (suffix i (pifilt xs f)) 0 ⟩)
  shows (pifilt (pifilt xs f) (LIFT(init w))) = pifilt xs (LIFT(f ∧ init w))
  proof -
    have 1: ∃ i ≤ ilen (pifilt xs f). (LIFT(init w)) (suffix i (pifilt xs f))
      using assms by (simp add: init-defs)
    have 2: (pifilt (pifilt xs f) (LIFT(init w))) =
      ifilter (λy. w (⟨y⟩)) (pifilt xs f)

      using 1 pifilt-init[of (pifilt xs f) w ] by auto
    have 3: (pifilt xs f) =
      imap (λs. inth s 0) (sfxfilt xs f)
      by (simp add: assms sfxfilt-pifilt)
    have 4: (sfxfilt xs f) = ifilter (λ ys. f ys) (suffixes xs)
      using sfxfilt-def by blast
    have 5: (pifilt xs f) = imap (λs. inth s 0) (ifilter (λ ys. f ys) (suffixes xs))
      by (simp add: 3 4)
    have 6: ifilter (λy. w (⟨y⟩)) (pifilt xs f) =
      ifilter (λy. w (⟨y⟩)) (imap (λs. inth s 0) (ifilter (λ ys. f ys) (suffixes xs)))

    using 5 by simp
  end

```



**have** 7:  $\text{ifilter } (\lambda y. w (\langle y \rangle)) (\text{imap } (\lambda s. \text{inth } s \ 0) (\text{ifilter } (\lambda ys. f \ ys) (\text{suffixes } xs))) =$   
 $\text{imap } (\lambda s. \text{inth } s \ 0) (\text{ifilter } ((\lambda y. w (\langle y \rangle)) \circ (\lambda s. \text{inth } s \ 0)) (\text{ifilter } (\lambda ys. f \ ys) (\text{suffixes } xs)))$   
**using** *assms* **by** (*metis* 3 4 *ifilter-imap in-iset-suffixes interval-sfx-1 osfx-suffix*)  
**have** 8:  $\exists x \in \text{iset } (\text{ifilter } (\lambda ys. f \ ys) (\text{suffixes } xs)). ((\lambda y. w (\langle y \rangle)) \circ (\lambda s. \text{inth } s \ 0)) \ x$   
**using** *assms* 3 4  
**by** *simp-all*  
 $(\text{metis } \text{ilen-imap inth-imap inth-iset})$   
**have** 9:  $\exists x \in \text{iset } (\text{suffixes } xs). (\lambda ys. f \ ys) \ x$   
**using** *assms in-iset-suffixes osfx-suffix* **by** *blast*  
**have** 10:  $\exists x \in \text{iset } (\text{suffixes } xs). ((\lambda y. w (\langle y \rangle)) \circ (\lambda s. \text{inth } s \ 0)) \ x \wedge (\lambda ys. f \ ys) \ x$   
**using** 8 9 **by** *auto*  
**have** 11:  $\text{ifilter } ((\lambda y. w (\langle y \rangle)) \circ (\lambda s. \text{inth } s \ 0)) (\text{ifilter } (\lambda ys. f \ ys) (\text{suffixes } xs)) =$   
 $\text{ifilter } (\lambda zs. ((\lambda y. w (\langle y \rangle)) \circ (\lambda s. \text{inth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) (\text{suffixes } xs)$   
  
**using** *ifilter-ifilter* [of  $(\lambda ys. f \ ys) (\text{suffixes } xs) ((\lambda y. w (\langle y \rangle)) \circ (\lambda s. \text{inth } s \ 0))$ ]   
8 10 **by** *blast*  
**have** 12:  $\exists i \leq \text{ilen } xs. (\lambda zs. ((\lambda y. w (\langle y \rangle)) \circ (\lambda s. \text{inth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) (\text{suffix } i \ xs)$   
**by** (*metis* 10 *ilen-suffixes inth-and-iset inth-suffixes*)  
**have** 13:  $(\text{ifilter } (\lambda zs. ((\lambda y. w (\langle y \rangle)) \circ (\lambda s. \text{inth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) (\text{suffixes } xs))$   
 $= (\text{sfxfilt } xs (\lambda zs. ((\lambda y. w (\langle y \rangle)) \circ (\lambda s. \text{inth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs))$   
  
**by** (*simp add: sfxfilt-def*)  
**have** 14:  $\exists i \leq \text{ilen } xs. (\lambda zs. ((\lambda y. w (\langle y \rangle)) \circ (\lambda s. \text{inth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) (\text{suffix } i \ xs)$   
**using** 12 **by** *blast*  
**have** 15:  $\text{imap } (\lambda s. \text{inth } s \ 0)$   
 $((\text{sfxfilt } xs (\lambda zs. ((\lambda y. w (\langle y \rangle)) \circ (\lambda s. \text{inth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs)) ) =$   
 $\text{pifilt } xs (\lambda zs. ((\lambda y. w (\langle y \rangle)) \circ (\lambda s. \text{inth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs)$   
  
**using** 14 **by** (*simp add: sfxfilt-pifilt*)  
**have** 16:  $\bigwedge xs. (\lambda zs. ((\lambda y. w (\langle y \rangle)) \circ (\lambda s. \text{inth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) \ xs =$   
 $(\text{LIFT}(f \wedge \text{init } w)) \ xs$   
  
**by** (*auto simp add: init-defs*)  
**have** 17:  $\text{pifilt } xs (\lambda zs. ((\lambda y. w (\langle y \rangle)) \circ (\lambda s. \text{inth } s \ 0)) \ zs \wedge (\lambda ys. f \ ys) \ zs) =$   
 $\text{pifilt } xs (\text{LIFT}(f \wedge \text{init } w))$   
  
**using** 16 **by** *presburger*  
**show** *?thesis*  
**using** 11 13 15 17 2 3 4 7 **by** *auto*  
**qed**

**lemma** *PiAssocsem*:

$\sigma \models f \ \Pi \ ((\text{init } w) \ \Pi \ g) = (f \wedge (\text{init } w)) \ \Pi \ g$

**proof** (*auto simp add: pi-d-def init-defs*)

**fix** *i*

**fix** *ia*

**assume** *a0*:  $g (\text{pifilt } (\text{pifilt } \sigma \ f) (\text{LIFT}(\text{init } w)))$

**assume** *a1*:  $i \leq \text{ilen } \sigma$

**assume** *a2*:  $f (\text{suffix } i \ \sigma)$

```

assume a3:  $ia \leq \text{ilen } (\text{pifilt } \sigma f)$ 
assume a4:  $w \langle \text{inth } (\text{pifilt } \sigma f) ia \rangle$ 
show  $\exists i \leq \text{ilen } \sigma. f (\text{suffix } i \sigma) \wedge w \langle \text{inth } \sigma i \rangle$ 
  using a0 a1 a2 a3 a4 PiAssocsem1 by fastforce
next
  fix  $i$ 
  fix  $ia$ 
  assume a0:  $g (\text{pifilt } (\text{pifilt } \sigma f) (\text{LIFT}(\text{init } w)))$ 
  assume a1:  $i \leq \text{ilen } \sigma$ 
  assume a2:  $f (\text{suffix } i \sigma)$ 
  assume a3:  $ia \leq \text{ilen } (\text{pifilt } \sigma f)$ 
  assume a4:  $w \langle \text{inth } (\text{pifilt } \sigma f) ia \rangle$ 
  show  $g (\text{pifilt } \sigma (\text{LIFT}(f \wedge \text{init } w)))$ 
    using a0 a1 a2 a3 a4 by (metis ifirst-suffix pifilt-pifilt)
next
  fix  $i$ 
  assume a0:  $g (\text{pifilt } \sigma (\text{LIFT}(f \wedge \text{init } w)))$ 
  assume a1:  $i \leq \text{ilen } \sigma$ 
  assume a2:  $f (\text{suffix } i \sigma)$ 
  assume a3:  $w \langle \text{inth } \sigma i \rangle$ 
  show  $\exists i \leq \text{ilen } (\text{pifilt } \sigma f). w \langle \text{inth } (\text{pifilt } \sigma f) i \rangle$ 
    using a0 a1 a2 a3 by (metis PiAssocsem2)
next
  fix  $i$ 
  assume a0:  $g (\text{pifilt } \sigma (\text{LIFT}(f \wedge \text{init } w)))$ 
  assume a1:  $i \leq \text{ilen } \sigma$ 
  assume a2:  $f (\text{suffix } i \sigma)$ 
  assume a3:  $w \langle \text{inth } \sigma i \rangle$ 
  show  $g (\text{pifilt } (\text{pifilt } \sigma f) (\text{LIFT}(\text{init } w)))$ 
    using a0 a1 a2 a3
    by (metis PiAssocsem2 ifirst-suffix pifilt-pifilt)
qed

```

**lemma** *PiAssoc*:

```

 $\vdash f \amalg ((\text{init } w) \amalg g) = (f \wedge (\text{init } w)) \amalg g$ 
using PiAssocsem Valid-def by blast

```

#### 15.4.10 PiNotEqvDiamondAndNotPi

**lemma** *PiNotEqvDiamondAndNotPisem*:

```

 $\sigma \models f \amalg (\neg g) = (\Diamond f \wedge \neg(f \amalg g))$ 
by (simp add: pi-d-def sometimes-defs) blast

```

**lemma** *PiNotEqvDiamondAndNotPi*:

```

 $\vdash f \amalg (\neg g) = (\Diamond f \wedge \neg(f \amalg g))$ 
using PiNotEqvDiamondAndNotPisem Valid-def by blast

```

#### 15.4.11 PiChopDist

**lemma** *iset-fuse*:

```

assumes  $ilast\ xs = ifirst\ ys$ 
shows  $iset\ (fuse\ xs\ ys) = iset\ xs \cup iset\ ys$ 
using assms
proof (induction xs arbitrary: ys)
case (INil x)
then show ?case
by (metis fuse-INil fuse-rightneutral iapp-assoc iset-iapp opfx-code(1) opfx-def sup.idem)
next
case (ICons x1a xs)
then show ?case by simp
qed

```

```

lemma ifilter-chop:
assumes  $ilast\ xs = ifirst\ ys \wedge P\ (ilast\ xs)$ 
 $(\exists\ x \in iset\ (fuse\ xs\ ys). P\ x)$ 
 $(\exists\ x \in iset\ xs. P\ x)$ 
 $(\exists\ x \in iset\ ys. P\ x)$ 
shows  $ifilter\ P\ (fuse\ xs\ ys) = fuse\ (ifilter\ P\ xs)\ (ifilter\ P\ ys)$ 
using assms
proof (induction xs arbitrary: ys)
case (INil x)
then show ?case
by simp
next
case (ICons x1a xs)
then show ?case
  proof (cases  $(\exists\ x \in iset\ xs. P\ x)$ )
  case True
  then show ?thesis
    using ICons.IH ICons.premis(1) ICons.premis(4) iset-fuse by fastforce
  next
  case False
  then show ?thesis
    using ICons.premis(1) inth-iset order-refl by force
  qed
qed

```

```

lemma ifilter-chop1:
assumes  $n \leq ilen\ xs \wedge P\ (ilast\ (prefix\ n\ xs))$ 
 $(\exists\ x \in iset\ xs. P\ x)$ 
 $(\exists\ x \in iset\ (prefix\ n\ xs). P\ x)$ 
 $(\exists\ x \in iset\ (suffix\ n\ xs). P\ x)$ 
shows  $ifilter\ P\ xs = fuse\ (ifilter\ P\ (prefix\ n\ xs))\ (ifilter\ P\ (suffix\ n\ xs))$ 
proof –
  have 1:  $(\exists\ x \in iset\ xs. P\ x) = (\exists\ x \in iset\ (fuse\ (prefix\ n\ xs)\ (suffix\ n\ xs)). P\ x)$ 
    by (simp add: assms(1) fuse-prefix-suffix)
  have 2:  $ilast\ (prefix\ n\ xs) = ifirst\ (suffix\ n\ xs)$ 
    using ilast-ifirst by blast
  have 3:  $xs = fuse\ (prefix\ n\ xs)\ (suffix\ n\ xs)$ 
    by (simp add: assms(1) fuse-prefix-suffix)

```

```

have 4: ifilter P (fuse (prefix n xs) (suffix n xs)) =
    fuse (ifilter P (prefix n xs)) (ifilter P (suffix n xs))
using assms 2 3 ifilter-chop[of (prefix n xs) (suffix n xs) P]
by auto
show ?thesis using 3 4 by auto
qed

```

**lemma** ifilter-chop1-prefix:

```

assumes n ≤ ilen xs
    P (ilast (prefix n xs))
    (∃ x ∈ iset xs. P x)
    (∃ x ∈ iset (prefix n xs). P x)
    (∃ x ∈ iset (suffix n xs). P x)
shows prefix (ilen (ifilter P (prefix n xs))) (ifilter P xs) =
    (ifilter P (prefix n xs))

```

**proof** –

```

have 2: ifilter P xs = fuse (ifilter P (prefix n xs)) (ifilter P (suffix n xs))
using assms ifilter-chop1 by blast
have 3: ilast (ifilter P (prefix n xs)) = ifirst (ifilter P (suffix n xs))
using assms by (metis ilast-ifirst ifilter-ifirst ifilter-ilast)
have 4: prefix (ilen (ifilter P (prefix n xs)))
    (fuse (ifilter P (prefix n xs)) (ifilter P (suffix n xs))) =
    (ifilter P (prefix n xs))

```

**using** prefix-fuse **using** 3 **by** blast

**show** ?thesis **by** (simp add: 2 4)

**qed**

**lemma** ifilter-chop1-suffix:

```

assumes n ≤ ilen xs
    P (ilast (prefix n xs))
    (∃ x ∈ iset xs. P x)
    (∃ x ∈ iset (prefix n xs). P x)
    (∃ x ∈ iset (suffix n xs). P x)
shows suffix (ilen (ifilter P (prefix n xs))) (ifilter P xs) =
    (ifilter P (suffix n xs))

```

**proof** –

```

have 1: ifilter P xs = fuse (ifilter P (prefix n xs)) (ifilter P (suffix n xs))
using assms ifilter-chop1 by blast
have 3: ilast (ifilter P (prefix n xs)) = ifirst (ifilter P (suffix n xs))
using assms by (metis ilast-ifirst ifilter-ifirst ifilter-ilast)
have 4: suffix (ilen (ifilter P (prefix n xs)))
    (fuse (ifilter P (prefix n xs)) (ifilter P (suffix n xs))) =
    (ifilter P (suffix n xs))

```

**using** suffix-fuse **using** 3 **by** blast

**show** ?thesis **by** (simp add: 1 4)

**qed**

**lemma** *PiChopDistsema*:

**assumes**  $(\sigma \models (\text{init } w) \amalg (g;h))$

**shows**  $(\sigma \models ((\text{init } w) \amalg g);((\text{init } w) \wedge ((\text{init } w) \amalg h)))$

**proof** –

**have** 1:  $(\sigma \models (\text{init } w) \amalg (g;h))$

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

**have** 2:  $((\exists i \leq \text{ilen } \sigma. (\text{LIFT}(\text{init } w)) (\text{suffix } i \sigma)) \wedge ((\text{pifilt } \sigma (\text{LIFT}(\text{init } w))) \models g;h))$

**using** 1 **by** (*simp add: pi-d-def*)

**have** 3:  $(\exists i \leq \text{ilen } \sigma. (\text{LIFT}(\text{init } w)) (\text{suffix } i \sigma))$

**using** 2 **by** *auto*

**have** 4:  $((\text{pifilt } \sigma (\text{LIFT}(\text{init } w))) \models g;h)$

**using** 2 **by** *auto*

**have** 5:  $\text{ifilter } (\lambda y. w \langle y \rangle) \sigma \models g;h$

**using** *pifilt-init*

**using** 2 **by** *fastforce*

**have** 6:  $\exists n \leq \text{ilen } (\text{ifilter } (\lambda y. w \langle y \rangle) \sigma).$

$g (\text{prefix } n (\text{ifilter } (\lambda y. w \langle y \rangle) \sigma)) \wedge$

$h (\text{suffix } n (\text{ifilter } (\lambda y. w \langle y \rangle) \sigma))$

**using** 5 **by** (*simp add: chop-defs*)

**obtain** *n* **where** 7:  $n \leq \text{ilen } (\text{ifilter } (\lambda y. w \langle y \rangle) \sigma) \wedge$

$g (\text{prefix } n (\text{ifilter } (\lambda y. w \langle y \rangle) \sigma)) \wedge$

$h (\text{suffix } n (\text{ifilter } (\lambda y. w \langle y \rangle) \sigma))$

**using** 6 **by** *auto*

**have** 8:  $\exists i \leq \text{ilen } \sigma. w \langle \text{inth } \sigma i \rangle$

**using** 3 **by** (*auto simp add: init-defs*)

**have** 9:  $\exists x \in \text{iset } \sigma. w \langle x \rangle$

**using** 8 *inth-iset* **by** *blast*

**have** 10:  $(\text{prefix } n (\text{ifilter } (\lambda y. w \langle y \rangle) \sigma)) =$

$(\text{ifilter } (\lambda y. w \langle y \rangle) (\text{prefix } ((\text{inth } (\text{nfilter } (\lambda y. w \langle y \rangle) \sigma 0) n) ) \sigma) )$

**by** (*simp add: 7 9 ifilter-nfilter-prefix-1*)

**have** 11:  $(\text{suffix } n (\text{ifilter } (\lambda y. w \langle y \rangle) \sigma)) =$

$(\text{ifilter } (\lambda y. w \langle y \rangle) (\text{suffix } ((\text{inth } (\text{nfilter } (\lambda y. w \langle y \rangle) \sigma 0) n) ) \sigma) )$

**by** (*simp add: 7 9 ifilter-nfilter-suffix-1*)

**have** 12:  $g (\text{ifilter } (\lambda y. w \langle y \rangle) (\text{prefix } ((\text{inth } (\text{nfilter } (\lambda y. w \langle y \rangle) \sigma 0) n) ) \sigma) )$

**using** 10 7 **by** *auto*

**have** 13:  $h (\text{ifilter } (\lambda y. w \langle y \rangle) (\text{suffix } ((\text{inth } (\text{nfilter } (\lambda y. w \langle y \rangle) \sigma 0) n) ) \sigma) )$

**using** 11 7 **by** *auto*

**have** 14:  $((\text{inth } (\text{nfilter } (\lambda y. w \langle y \rangle) \sigma 0) n) ) \leq \text{ilen } \sigma$

**by** (*metis 7 9 add-cancel-right-left nfilter-ilen nfilter-upper-bound*)

**have** 15:  $w \langle \text{inth } (\text{suffix } ((\text{inth } (\text{nfilter } (\lambda y. w \langle y \rangle) \sigma 0) n) ) \sigma) 0 \rangle$

**by** (*metis (no-types, lifting) 14 7 9 ifilter-inth-aa ifirst-suffix nfilter-ifilter nfilter-ilen nfilter-inth-n-zero*)

**have** 16:  $(\exists i \leq \text{ilen } (\text{prefix } ((\text{inth } (\text{nfilter } (\lambda y. w \langle y \rangle) \sigma 0) n) ) \sigma).$

$w \langle \text{inth } (\text{suffix } i (\text{prefix } ((\text{inth } (\text{nfilter } (\lambda y. w \langle y \rangle) \sigma 0) n) ) \sigma)) 0 \rangle)$

**using** 14 15 **by** *auto*

**have** 17:  $(\exists i \leq \text{ilen } (\text{suffix } ((\text{inth } (\text{nfilter } (\lambda y. w \langle y \rangle) \sigma 0) n) ) \sigma).$

$w \langle \text{inth } (\text{suffix } (i + ((\text{inth } (\text{nfilter } (\lambda y. w \langle y \rangle) \sigma 0) n) )) \sigma) 0 \rangle)$

**using 15 by auto**  
**have 18:**  $(\exists n \leq \text{ilen } \sigma. (\exists i \leq \text{ilen } (\text{prefix } n \sigma). w \langle \text{inth } (\text{suffix } i (\text{prefix } n \sigma)) 0 \rangle) \wedge$   
 $g (\text{ifilter } (\lambda y. w \langle y \rangle) (\text{prefix } n \sigma)) \wedge$   
 $w \langle \text{inth } (\text{suffix } n \sigma) 0 \rangle \wedge$   
 $(\exists i \leq \text{ilen } (\text{suffix } n \sigma). w \langle \text{inth } (\text{suffix } (i + n) \sigma) 0 \rangle) \wedge$   
 $h (\text{ifilter } (\lambda y. w \langle y \rangle) (\text{suffix } n \sigma)) )$   
**using 12 13 14 15 16 17 by blast**  
**have 181:**  $(\exists n \leq \text{ilen } \sigma. (\exists i \leq \text{ilen } (\text{prefix } n \sigma). w \langle \text{inth } (\text{prefix } n \sigma) i \rangle) )$   
**using 18 by auto**  
**have 182:**  $(\exists n \leq \text{ilen } \sigma. (\exists i \leq \text{ilen } (\text{suffix } n \sigma). w \langle \text{inth } \sigma (i + n) \rangle) )$   
**using 18 by auto**  
**have 19:**  $(\exists n \leq \text{ilen } \sigma. (\exists i \leq \text{ilen } (\text{prefix } n \sigma). w \langle \text{inth } (\text{suffix } i (\text{prefix } n \sigma)) 0 \rangle) \wedge$   
 $g (\text{pifilt } (\text{prefix } n \sigma) (\text{LIFT}(\text{init } w))) \wedge$   
 $w \langle \text{inth } (\text{suffix } n \sigma) 0 \rangle \wedge$   
 $(\exists i \leq \text{ilen } (\text{suffix } n \sigma). w \langle \text{inth } (\text{suffix } (i + n) \sigma) 0 \rangle) \wedge$   
 $h (\text{pifilt } (\text{suffix } n \sigma) (\text{LIFT}(\text{init } w))))$   
**using 18 pifilt-init[*of* - *w*]**  
**proof –**  
**assume a1:**  $\bigwedge xs. \exists i \leq \text{ilen } xs. (\text{suffix } i xs) \models (\text{init } w) \implies$   
 $\text{pifilt } xs (\text{LIFT}(\text{init } w)) = \text{ifilter } (\lambda y. w \langle y \rangle) xs$   
**obtain nn :: nat and nna :: nat and nnb :: nat where**  
**f2:**  $h (\text{ifilter } (\lambda a. w \langle a \rangle) (\text{suffix } nn \sigma)) \wedge$   
 $(w \langle \text{inth } (\text{suffix } (nna + nn) \sigma) 0 \rangle \wedge nna \leq \text{ilen } (\text{suffix } nn \sigma)) \wedge$   
 $w \langle \text{inth } (\text{suffix } nn \sigma) 0 \rangle \wedge$   
 $g (\text{ifilter } (\lambda a. w \langle a \rangle) (\text{prefix } nn \sigma)) \wedge$   
 $(w \langle \text{inth } (\text{suffix } nnb (\text{prefix } nn \sigma)) 0 \rangle \wedge nnb \leq \text{ilen } (\text{prefix } nn \sigma)) \wedge$   
 $nn \leq \text{ilen } \sigma$   
**using**  $(\exists n \leq \text{ilen } \sigma. (\exists i \leq \text{ilen } (\text{prefix } n \sigma). w \langle \text{inth } (\text{suffix } i (\text{prefix } n \sigma)) 0 \rangle) \wedge$   
 $g (\text{ifilter } (\lambda y. w \langle y \rangle) (\text{prefix } n \sigma)) \wedge$   
 $w \langle \text{inth } (\text{suffix } n \sigma) 0 \rangle \wedge$   
 $(\exists i \leq \text{ilen } (\text{suffix } n \sigma). w \langle \text{inth } (\text{suffix } (i + n) \sigma) 0 \rangle) \wedge$   
 $h (\text{ifilter } (\lambda y. w \langle y \rangle) (\text{suffix } n \sigma)))$  **by blast**  
**then have**  $\exists n \text{ na. } w (\text{prefix } 0 (\text{suffix } na (\text{suffix } nn \sigma))) \wedge$   
 $h (\text{pifilt } (\text{suffix } nn \sigma) (\text{LIFT}(\text{init } w))) \wedge$   
 $na \leq \text{ilen } (\text{suffix } nn \sigma) \wedge$   
 $g (\text{pifilt } (\text{prefix } nn \sigma) (\text{LIFT}(\text{init } w))) \wedge$   
 $n \leq \text{ilen } (\text{prefix } nn \sigma) \wedge$   
 $w (\text{prefix } 0 (\text{suffix } n (\text{prefix } nn \sigma)))$   
**using a1 by** (*metis* (*no-types*) *init-defs prefix-zero-ifirst suffix-suffix*)  
**then show** *?thesis*  
**using f2 by blast**  
**qed**  
**have 20:**  $((\exists n \leq \text{ilen } \sigma. (\exists i \leq \text{ilen } (\text{prefix } n \sigma). (\text{LIFT}(\text{init } w)) (\text{suffix } i (\text{prefix } n \sigma))) \wedge$   
 $g (\text{pifilt } (\text{prefix } n \sigma) (\text{LIFT}(\text{init } w))) \wedge$   
 $(\text{LIFT}(\text{init } w)) (\text{suffix } n \sigma) \wedge (\exists i \leq \text{ilen } (\text{suffix } n \sigma). (\text{LIFT}(\text{init } w)) (\text{suffix } (i + n) \sigma))$   
 $\wedge h (\text{pifilt } (\text{suffix } n \sigma) (\text{LIFT}(\text{init } w))))$

```

  by (metis 19 init-defs prefix-zero-ifirst)
  have 21:  $(\sigma \models ((init\ w) \sqcap g); ((init\ w) \wedge ((init\ w) \sqcap h)))$ 
    using 20 by (simp add: chop-defs pi-d-def)
show ?thesis
using 21 by auto
qed

```

**lemma** *PiChopDistsemb*:

```

  assumes  $(\sigma \models ((init\ w) \sqcap g); ((init\ w) \wedge ((init\ w) \sqcap h)))$ 
  shows  $(\sigma \models (init\ w) \sqcap (g;h))$ 
proof -
  have 1:  $(\sigma \models ((init\ w) \sqcap g); ((init\ w) \wedge ((init\ w) \sqcap h)))$ 
    using assms by auto
  have 2:  $\exists n \leq ilen\ \sigma.$ 
     $((prefix\ n\ \sigma) \models ((init\ w) \sqcap g)) \wedge$ 
     $((suffix\ n\ \sigma) \models ((init\ w) \wedge ((init\ w) \sqcap h)))$ 

    using assms chop-defs by blast
  obtain n where 3:  $n \leq ilen\ \sigma \wedge ((prefix\ n\ \sigma) \models ((init\ w) \sqcap g)) \wedge$ 
     $((suffix\ n\ \sigma) \models ((init\ w) \wedge ((init\ w) \sqcap h)))$ 
    using 2 by auto
  have 4:  $((\exists i \leq ilen\ (prefix\ n\ \sigma). (LIFT(init\ w)) (suffix\ i\ (prefix\ n\ \sigma))) \wedge$ 
     $((pifilt\ (prefix\ n\ \sigma)\ (LIFT(init\ w))) \models g)$ 
    )
    by (meson 3 pi-d-def)
  have 5:  $(\exists i \leq ilen\ (prefix\ n\ \sigma). (LIFT(init\ w)) (suffix\ i\ (prefix\ n\ \sigma)))$ 
    using 4 by auto
  have 6:  $g\ (pifilt\ (prefix\ n\ \sigma)\ (LIFT(init\ w)))$ 
    using 4 by auto
  have 7:  $g\ (ifilter\ (\lambda y. w\ (\langle y \rangle))\ (prefix\ n\ \sigma))$ 
    using 5 6 pifilt-init by (metis)
  have 8:  $((\exists i \leq ilen\ (suffix\ n\ \sigma). (LIFT(init\ w)) (suffix\ i\ (suffix\ n\ \sigma))) \wedge$ 
     $((pifilt\ (suffix\ n\ \sigma)\ (LIFT(init\ w))) \models h)$ 
    )
    by (metis 3 intensional-rews(3) pi-d-def)
  have 9:  $(\exists i \leq ilen\ (suffix\ n\ \sigma). (LIFT(init\ w)) (suffix\ i\ (suffix\ n\ \sigma)))$ 
    using 8 by auto
  have 10:  $h\ (pifilt\ (suffix\ n\ \sigma)\ (LIFT(init\ w)))$ 
    using 8 by auto
  have 11:  $h\ (ifilter\ (\lambda y. w\ (\langle y \rangle))\ (suffix\ n\ \sigma))$ 
    using 10 9 pifilt-init by metis
  have 12:  $(\lambda y. w\ (\langle y \rangle))\ (ilast\ (prefix\ n\ \sigma))$ 
    by (metis 3 ilast-ifirst init-defs intensional-rews(3) prefix-zero-ifirst)
  have 13:  $\exists x \in iset\ \sigma. (\lambda y. w\ (\langle y \rangle))\ x$ 
    using 12 3 inth-iset using ilast-prefix by fastforce
  have 14:  $\exists x \in iset\ (prefix\ n\ \sigma) . (\lambda y. w\ (\langle y \rangle))\ x$ 
    using 12 inth-iset by (metis order-refl)
  have 15:  $\exists x \in iset\ (suffix\ n\ \sigma) . (\lambda y. w\ (\langle y \rangle))\ x$ 
    by (metis 12 3 ifirst-suffix ilast-prefix ilen-gr-zero inth-iset)

```

**have** 16:  $(\text{ifilter } (\lambda y. w \langle y \rangle)) (\text{prefix } n \sigma) =$   
 $\text{prefix } (\text{ilen } (\text{ifilter } (\lambda y. w \langle y \rangle)) (\text{prefix } n \sigma)) (\text{ifilter } (\lambda y. w \langle y \rangle)) \sigma$   
  
**using** 12 13 14 15 3  
 $\text{ifilter-chop1-prefix}[of\ n\ \sigma\ (\lambda y. w \langle y \rangle)]$  **by** *auto*  
**have** 17:  $(\text{ifilter } (\lambda y. w \langle y \rangle)) (\text{suffix } n \sigma) =$   
 $\text{suffix } (\text{ilen } (\text{ifilter } (\lambda y. w \langle y \rangle)) (\text{prefix } n \sigma)) (\text{ifilter } (\lambda y. w \langle y \rangle)) \sigma$   
  
**using** 12 13 14 15 3  
 $\text{ifilter-chop1-suffix}[of\ n\ \sigma\ (\lambda y. w \langle y \rangle)]$  **by** *auto*  
**have** 18:  $\exists n \leq \text{ilen } (\text{ifilter } (\lambda y. w \langle y \rangle)) \sigma.$   
 $g (\text{prefix } n (\text{ifilter } (\lambda y. w \langle y \rangle)) \sigma) \wedge$   
 $h (\text{suffix } n (\text{ifilter } (\lambda y. w \langle y \rangle)) \sigma)$   
**by** (*metis* 11 16 17 7 *prefix-ilen-bound*)  
**have** 19:  $\text{ifilter } (\lambda y. w \langle y \rangle)) \sigma \models g;h$   
**by** (*simp add: 18 chop-defs*)  
**have** 20:  $((\text{pifilt } \sigma (\text{LIFT}(\text{init } w))) \models g;h)$   
**by** (*metis* (*mono-tags*, *lifting*) 18 3 *ilast-ifirst intensional-rews*(3)  
*chop-fuse fuse-prefix-suffix pifilt-init*)  
**have** 21:  $(\exists i \leq \text{ilen } \sigma. (\text{LIFT}(\text{init } w)) (\text{suffix } i \sigma))$   
**using** 3 **by** *auto*  
**show** ?thesis  
**by** (*simp add: 20 21 pi-d-def*)  
**qed**

**lemma** *PiChopDistsem*:

$\sigma \models (\text{init } w) \Pi (g;h) = ((\text{init } w) \Pi g);((\text{init } w) \wedge ((\text{init } w) \Pi h))$   
**using** *PiChopDistsema PiChopDistsemb unl-lift2* **by** *blast*

**lemma** *PiChopDist*:

$\vdash (\text{init } w) \Pi (g;h) = ((\text{init } w) \Pi g);((\text{init } w) \wedge ((\text{init } w) \Pi h))$   
**using** *PiChopDistsem Valid-def* **by** *blast*

### 15.4.12 PiProp

**lemma** *Pistate*:

$(\sigma \models (\text{init } w) \Pi f) =$   
 $((\exists x \in \text{iset } \sigma. w \langle x \rangle) \wedge ((\text{ifilter } (\lambda y. w \langle y \rangle)) \sigma \models f))$

**proof** –

**have** 1:  $(\sigma \models (\text{init } w) \Pi f) =$   
 $((\exists i \leq \text{ilen } \sigma. w \langle \text{inth } \sigma\ i \rangle) \wedge ((\text{pifilt } \sigma (\text{LIFT}(\text{init } w))) \models f))$   
**by** (*auto simp add: pi-d-def init-defs*)  
**have** 2:  $(\exists i \leq \text{ilen } \sigma. (\text{LIFT}(\text{init } w)) (\text{suffix } i \sigma)) =$   
 $(\exists i \leq \text{ilen } \sigma. w \langle \text{inth } \sigma\ i \rangle)$   
**by** (*auto simp add: init-defs*)  
**have** 3:  $(\exists i \leq \text{ilen } \sigma. w \langle \text{inth } \sigma\ i \rangle) \longrightarrow$   
 $(\text{pifilt } \sigma (\text{LIFT}(\text{init } w))) = (\text{ifilter } (\lambda y. w \langle y \rangle)) \sigma$   
**using** *pifilt-init* **using** 2 **by** *blast*  
**have** 4:  $(\exists i \leq \text{ilen } \sigma. w \langle \text{inth } \sigma\ i \rangle) = (\exists x \in \text{iset } \sigma. w \langle x \rangle)$   
**using** *inth-and-iset* **by** *force*



**show** ?thesis  
**using** 1 3 4 **by** auto  
**qed**

**lemma** *PiPropsem1a*:

$(\sigma \models f \Pi \$p) =$   
 $((\exists i \leq \text{ilen } \sigma. f (\text{suffix } i \sigma)) \wedge p (\text{inth } \sigma (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) 0) 0)))$

**using** *inth-imap*[of  $(\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } \sigma)) 0]$   
**using** *nfilter-ifilter*[of *suffixes*  $\sigma$   $f$  0 0 ]  
**by** (*simp add: pi-d-def current-val-d-def pifilt-def sxfilt-def* )  
*(metis in-iset-suffixes inth-imap imap-first-suffixes osfx-suffix)*

**lemma** *PiPropsem2a*:

$(\sigma \models (\neg f) \mathcal{U} (f \wedge \$p)) =$   
 $(\exists k \leq \text{ilen } \sigma. f (\text{suffix } k \sigma) \wedge p (\text{inth } \sigma k) \wedge (\forall j < k. \neg f (\text{suffix } j \sigma)))$   
**by** (*simp add: until-d-def current-val-d-def*)

**lemma** *PiPropsem3a*:

**assumes**  $(\sigma \models f \Pi \$p)$   
**shows**  $(\sigma \models (\neg f) \mathcal{U} (f \wedge \$p))$   
**proof** –  
**have** 1:  $((\exists i \leq \text{ilen } \sigma. f (\text{suffix } i \sigma)) \wedge p (\text{inth } \sigma (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) 0) 0)))$   
**using** *assms PiPropsem1a by auto*  
**have** 2:  $\exists x \in \text{iset}(\text{suffixes } \sigma). f x$   
**using** 1 *in-iset-suffixes osfx-suffix by blast*  
**have** 3:  $\forall x \in \text{iset}(\text{nfilter } f (\text{suffixes } \sigma) 0). f (\text{inth } (\text{suffixes } \sigma) x)$   
**by** (*simp add: 2*)  
**have** 4:  $f (\text{suffix } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) 0) 0) \sigma)$   
**by** (*metis 2 3 add.left-neutral ilen-gr-zero nfilter-upper-bound inth-iset inth-suffixes*)  
**have** 5:  $(\forall j < (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) 0) 0). \neg f (\text{suffix } j \sigma))$   
**using** *nfilter-not-before*[of *suffixes*  $\sigma$   $f$  ] 2  
**proof** –  
**have** f1:  $\forall n. \neg n < \text{inth } (\text{nfilter } f (\text{suffixes } \sigma) 0) 0 \vee \neg f (\text{inth } (\text{suffixes } \sigma) n)$   
**using**  $\langle \wedge i. [\exists x \in \text{iset} (\text{suffixes } \sigma). f x; i < \text{inth } (\text{nfilter } f (\text{suffixes } \sigma) 0) 0] \Rightarrow \neg f (\text{inth } (\text{suffixes } \sigma) i) \rangle \langle \exists x \in \text{iset} (\text{suffixes } \sigma). f x \rangle$  **by** *blast*  
**obtain** *ii* :: 'a *interval* **where**  
*f2*:  $ii \in \text{iset} (\text{suffixes } \sigma) \wedge f ii$   
**using**  $\langle \exists x \in \text{iset} (\text{suffixes } \sigma). f x \rangle$  **by** *blast*  
**obtain** *nn* :: 'a *interval*  $\text{interval} \Rightarrow$  'a *interval*  $\Rightarrow$  nat **where**  
 $\forall x0 x1. (\exists v2 \leq \text{ilen } x0. \text{inth } x0 v2 = x1) = (nn x0 x1 \leq \text{ilen } x0 \wedge \text{inth } x0 (nn x0 x1) = x1)$   
**by** *moura*  
**then have**  $nn (\text{suffixes } \sigma) ii \leq \text{ilen } (\text{suffixes } \sigma) \wedge \text{inth } (\text{suffixes } \sigma) (nn (\text{suffixes } \sigma) ii) = ii$   
**using** *f2 by (meson inth-and-iset)*  
**then have**  $\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) 0) 0 \leq \text{ilen } (\text{suffixes } \sigma)$   
**using** *f2 f1 by (metis (no-types) dual-order.trans le-less-linear)*  
**then show** ?thesis  
**using** *f1 by (metis (no-types) dual-order.trans less-or-eq-imp-le inth-suffixes)*

**qed**  
**have** 6:  $(\exists k \leq \text{ilen } \sigma. f (\text{suffix } k \ \sigma) \wedge p (\text{inth } \sigma \ k) \wedge (\forall j < k. \neg f (\text{suffix } j \ \sigma)))$   
**by** (metis 1 4 5 interval-suf-first neqE)  
**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:  $(\exists k \leq \text{ilen } \sigma. f (\text{suffix } k \ \sigma) \wedge p (\text{inth } \sigma \ k) \wedge (\forall j < k. \neg f (\text{suffix } j \ \sigma)))$   
**using** assms PiPropsem2a **by** auto  
**obtain** k **where** 2:  $k \leq \text{ilen } \sigma \wedge f (\text{suffix } k \ \sigma) \wedge p (\text{inth } \sigma \ k) \wedge (\forall j < k. \neg f (\text{suffix } j \ \sigma))$   
**using** 1 **by** auto  
**have** 3:  $(\exists i \leq \text{ilen } \sigma. f (\text{suffix } i \ \sigma))$   
**using** 2 **by** blast  
**have** 31:  $\exists x \in \text{iset}(\text{suffixes } \sigma). f \ x$   
**using** 2 **by** auto  
**have** 32:  $f (\text{suffix } (\text{nth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 0) \ \sigma)$   
**using** nfilter-holds[of suffixes  $\sigma$  f 0] nfilter-not-before[of suffixes  $\sigma$  f]  
**by** (metis 31 diff-zero ilen-gr-zero nfilter-upper-bound inth-iset inth-suffixes  
ordered-cancel-comm-monoid-diff-class.add-diff-inverse)  
**have** 4:  $p (\text{inth } \sigma \ (\text{nth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 0) \ )$   
**by** (metis 1 31 32 ilen-suffixes linorder-neqE-nat nfilter-not-before inth-suffixes)  
**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

### 15.4.13 PiNext

**lemma** PiNextsem1:  
 $(\sigma \models f \ \Pi \ (\bigcirc \ g)) =$   
 $((\exists i \leq \text{ilen } \sigma. f (\text{suffix } i \ \sigma)) \wedge$   
 $0 < \text{ilen } (\text{ifilter } f (\text{suffixes } \sigma)) \wedge$   
 $g (\text{suffix } (\text{Suc } 0) (\text{imap } (\lambda s. \text{inth } s \ 0) (\text{ifilter } f (\text{suffixes } \sigma)))))$   
**by** (simp add: pi-d-def next-defs pifilt-def sxfilt-def)

**lemma** PiNextsem2:  
 $(\sigma \models (\neg f) \ \mathcal{U} \ (f \wedge \bigcirc(f \ \Pi \ g))) =$   
 $(\exists k \leq \text{ilen } \sigma.$   
 $f (\text{suffix } k \ \sigma) \wedge$   
 $k < \text{ilen } \sigma \wedge$

$(\exists i \leq \text{ilen } \sigma - \text{Suc } k. f (\text{suffix } (\text{Suc } (i + k)) \sigma)) \wedge$   
 $g (\text{imap } (\lambda s. \text{inth } s \ 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } (\text{Suc } k) \sigma)))) \wedge (\forall j < k. \neg f (\text{suffix } j \sigma))$

by (simp add: until-d-def next-defs pi-d-def pifilt-def sfxfilt-def)

**lemma** *PiNextsem3*:

**assumes**  $(\sigma \models f \ \Pi \ (\bigcirc g))$

**shows**  $(\sigma \models (\neg f) \ \mathcal{U} \ (f \wedge \bigcirc(f \ \Pi \ g)))$

**proof** –

**have** 1:  $((\exists i \leq \text{ilen } \sigma. f (\text{suffix } i \sigma)) \wedge$

$0 < \text{ilen } (\text{ifilter } f (\text{suffixes } \sigma)) \wedge$

$g (\text{suffix } (\text{Suc } 0) (\text{imap } (\lambda s. \text{inth } s \ 0) (\text{ifilter } f (\text{suffixes } \sigma))))$

**using** *assms PiNextsem1* **by** *auto*

**have** 2:  $\exists x \in \text{iset}(\text{suffixes } \sigma). f \ x$

**using** 1 *in-iset-suffixes osfx-suffix* **by** *blast*

**have** 3:  $\forall x \in \text{iset}(\text{nfilter } f (\text{suffixes } \sigma) \ 0). f \ (\text{inth } (\text{suffixes } \sigma) \ x)$

**by** (simp add: 2)

**have** 4:  $f (\text{suffix } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 0) \ \sigma)$

**by** (metis 2 *add.left-neutral ilen-gr-zero nfilter-ifilter nfilter-inth-n-zero*  
*nfilter-upper-bound inth-suffixes sfxfilter-inth*)

**have** 41:  $(\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 1) \in \text{iset}(\text{nfilter } f (\text{suffixes } \sigma) \ 0)$

**by** (metis 1 2 *One-nat-def Suc-leI nfilter-ilen inth-iset*)

**have** 42:  $(\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 1) \leq \text{ilen } (\text{suffixes } \sigma)$

**by** (metis 1 2 *One-nat-def Suc-leI add-cancel-right-left nfilter-ilen nfilter-upper-bound*)

**have** 5:  $f (\text{suffix } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 1) \ \sigma)$

**using** 3 41 42 *inth-suffixes* **by** *fastforce*

**have** 6:  $(\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 0) < (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 1)$

**by** (simp add: 1 2 *idx-nfilter-mono nfilter-ilen*)

**have** 7:  $\text{Suc } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 0) \leq \text{ilen } \sigma$

**by** (metis 1 3 *ilen-ifilter-le Suc-diff-Suc diff-is-0-eq'*  
*ifilter-nfilter-suffix ilen-gr-zero suffix-ilen ilen-suffixes*  
*not-less-eq-eq inth-iset*)

**have** 8:  $0 < \text{ilen } (\text{nfilter } f (\text{suffixes } \sigma) \ 0)$

**by** (simp add: 1 2 *nfilter-ilen*)

**have** 9:  $(\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 1) \leq \text{ilen } \sigma$

**using** *nfilter-upper-bound[of suffixes σ f 1 0]*

**by** (simp add: 2 8 *Suc-leI*)

**have** 10:  $(\text{Suc } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 0)) \leq (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 1)$

**using** 6 *Suc-leI* **by** *blast*

**have** 11:  $(\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 1) =$

$(\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 1) - (\text{Suc } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 0)) +$   
 $(\text{Suc } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 0))$

**using** 10 **by** *auto*

**have** 12:  $(\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 1) - (\text{Suc } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 0)) \leq$   
 $\text{ilen } \sigma - \text{Suc } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 0)$

**using** 9 *diff-le-mono* **by** *blast*

**have** 13:  $\exists i \leq \text{ilen } \sigma - \text{Suc } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 0).$

$(\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 1) = (\text{Suc } (i + (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ 0)))$

**using** 11 12 **by** *auto*

**have 14:**  $(\exists i \leq \text{ilen } \sigma - \text{Suc } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) 0) 0)).$   
 $f (\text{suffix } (\text{Suc } (i + (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) 0) 0))) \sigma))$   
**using 13 5 by auto**  
**have 15:**  $(\text{suffix } (\text{Suc } 0) (\text{imap } (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } \sigma)))) =$   
 $(\text{imap } (\lambda s. \text{inth } s 0)$   
 $(\text{ifilter } f (\text{suffixes } (\text{suffix } (\text{Suc } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) 0) 0)) \sigma))))$   
**using 2 8 by (simp add: 1 Suc-leI ifilter-suffixes-imap)**  
**have 16:**  $g (\text{imap } (\lambda s. \text{inth } s 0)$   
 $(\text{ifilter } f (\text{suffixes } (\text{suffix } (\text{Suc } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) 0) 0)) \sigma))))$   
**using 1 15 by auto**  
**have 17:**  $\forall j < (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) 0) 0). \neg f (\text{suffix } j \sigma)$   
**by (metis 7 Suc-leD Suc-leI in-iset-suffixes ilen-suffixes le-trans nfilter-not-before**  
 $\text{inth-suffixes osfx-suffix})$   
**have 18:**  $(\exists k \leq \text{ilen } \sigma.$   
 $f (\text{suffix } k \sigma) \wedge$   
 $k < \text{ilen } \sigma \wedge$   
 $(\exists i \leq \text{ilen } \sigma - \text{Suc } k. f (\text{suffix } (\text{Suc } (i + k)) \sigma)) \wedge$   
 $g (\text{imap } (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } (\text{Suc } k) \sigma)))) \wedge (\forall j < k. \neg f (\text{suffix } j \sigma)))$   
**using 14 16 17 4 7 Suc-leD Suc-le-lessD 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:**  $(\exists k \leq \text{ilen } \sigma.$   
 $f (\text{suffix } k \sigma) \wedge$   
 $k < \text{ilen } \sigma \wedge$   
 $(\exists i \leq \text{ilen } \sigma - \text{Suc } k. f (\text{suffix } (\text{Suc } (i + k)) \sigma)) \wedge$   
 $g (\text{imap } (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } (\text{Suc } k) \sigma)))) \wedge (\forall j < k. \neg f (\text{suffix } j \sigma)))$   
**using assms by (simp add: PiNextsem2)**  
**obtain k where 2:**  $k \leq \text{ilen } \sigma \wedge f (\text{suffix } k \sigma) \wedge k < \text{ilen } \sigma \wedge (\exists i \leq \text{ilen } \sigma - \text{Suc } k.$   
 $f (\text{suffix } (\text{Suc } (i + k)) \sigma)) \wedge$   
 $g (\text{imap } (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } (\text{Suc } k) \sigma)))) \wedge (\forall j < k. \neg f (\text{suffix } j \sigma))$   
**using 1 by auto**  
**have 3:**  $\exists x \in \text{iset } (\text{suffixes } \sigma). f x$   
**using 1 in-iset-suffixes osfx-suffix by blast**  
**have 4:**  $\forall x \in \text{iset}(\text{nfilter } f (\text{suffixes } \sigma) 0). f (\text{inth } (\text{suffixes } \sigma) x)$   
**by (simp add: 3)**  
**have 5:**  $f (\text{suffix } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) 0) 0) \sigma)$   
**by (metis 3 4 add.left-neutral ilen-gr-zero nfilter-upper-bound inth-iset**  
 $\text{inth-suffixes})$   
**have 6:**  $0 < \text{ilen } (\text{ifilter } f (\text{suffixes } \sigma))$   
**using ifilter-ilen-zero-conv-a[of suffixes  $\sigma$  f]**  
**by (metis 2 3 Nat.le-diff-conv2 Suc-leI Suc-n-not-le-n add-Suc-right ilen-suffixes le-add2**  
 $\text{neq0-conv inth-suffixes})$   
**have 61:**  $(\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) 0) 1) \in \text{iset}(\text{nfilter } f (\text{suffixes } \sigma) 0)$   
**by (metis 3 6 One-nat-def Suc-leI nfilter-ilen inth-iset)**

**have** 62: ( $\text{inth } (\text{nfilter } f \text{ (suffixes } \sigma) \ 0) \ 1) \leq \text{ilen } (\text{suffixes } \sigma)$   
**by** (*metis* 3 6 *One-nat-def* *Suc-leI* *add-cancel-right-left* *nfilter-ilen* *nfilter-upper-bound*)  
**have** 7:  $f \text{ (suffix (inth (nfilter } f \text{ (suffixes } \sigma) \ 0) \ 1) \ \sigma)$   
**using** 4 61 62 *inth-suffixes* **by** *fastforce*  
**have** 8:  $(\exists i \leq \text{ilen } \sigma. f \text{ (suffix } i \ \sigma))$   
**using** 1 **by** *blast*  
**have** 9:  $g \text{ (suffix (Suc } 0) \text{ (imap } (\lambda s. \text{inth } s \ 0) \text{ (ifilter } f \text{ (suffixes } \sigma))))}$   
**by** (*metis* 2 3 5 6 *Suc-leI* *ilen-suffixes* *linorder-neqE-nat* *nfilter-not-before* *inth-suffixes* *ifilter-suffixes-imap*)  
**have** 10:  $((\exists i \leq \text{ilen } \sigma. f \text{ (suffix } i \ \sigma)) \wedge$   
 $0 < \text{ilen } (\text{ifilter } f \text{ (suffixes } \sigma)) \wedge$   
 $g \text{ (suffix (Suc } 0) \text{ (imap } (\lambda s. \text{inth } s \ 0) \text{ (ifilter } f \text{ (suffixes } \sigma)))))$   
**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*  
**using** *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*

## 15.4.14 PiUntil

**lemma** *PiUntilDistsem1*:

$(\sigma \models f \ \Pi \ (g \ \mathcal{U} \ h)) =$   
 $((\exists i \leq \text{ilen } \sigma. f \text{ (suffix } i \ \sigma)) \wedge$   
 $(\exists k \leq \text{ilen } (\text{ifilter } f \text{ (suffixes } \sigma)).$   
 $h \text{ (suffix } k \text{ (imap } (\lambda s. \text{inth } s \ 0) \text{ (ifilter } f \text{ (suffixes } \sigma)))) \wedge$   
 $(\forall j < k. g \text{ (suffix } j \text{ (imap } (\lambda s. \text{inth } s \ 0) \text{ (ifilter } f \text{ (suffixes } \sigma))))))$   
**by** (*simp* *add: pi-d-def* *pifilt-def* *sxfilt-def* *until-d-def*)

**lemma** *PiUntilDistsem2*:

$(\sigma \models (f \ \Pi \ g) \ \mathcal{U} \ (f \ \Pi \ h)) =$   
 $(\exists k \leq \text{ilen } \sigma.$   
 $(\exists i \leq \text{ilen } \sigma - k. f \text{ (suffix } (i + k) \ \sigma)) \wedge$   
 $h \text{ (imap } (\lambda s. \text{inth } s \ 0) \text{ (ifilter } f \text{ (suffixes (suffix } k \ \sigma)))) \wedge$   
 $(\forall j < k. (\exists i \leq \text{ilen } \sigma - j. f \text{ (suffix } (i + j) \ \sigma)) \wedge$   
 $g \text{ (imap } (\lambda s. \text{inth } s \ 0) \text{ (ifilter } f \text{ (suffixes (suffix } j \ \sigma))))))$   
**by** (*simp* *add: until-d-def* *pi-d-def* *pifilt-def* *sxfilt-def*)

**lemma** *cover*:

```

assumes ( $\exists i < k. \text{ff}(i) < (j::\text{nat}) \wedge j \leq \text{ff}(\text{Suc } i)$ )
          ( $\forall i < k. \text{ff}(i) < \text{ff}(\text{Suc } i)$ )
shows  $\text{ff}(0) < j \wedge j \leq \text{ff}(k)$ 
using assms
proof (induct k arbitrary: j)
case 0
then show ?case by simp
next
case (Suc k)
then show ?case
proof –
  have 1: ( $\exists i < k. \text{ff}(i) < (j::\text{nat}) \wedge j \leq \text{ff}(\text{Suc } i)$ )  $\vee$  ( $\text{ff}(k) < j \wedge j \leq \text{ff}(\text{Suc } k)$ )
    using Suc.premis(1) less-SucE by blast
  have 2: ( $\forall i < k. \text{ff}(i) < \text{ff}(\text{Suc } i)$ )
    using Suc.premis(2) by auto
  have 3:  $\text{ff } k < \text{ff}(\text{Suc } k)$ 
    by (simp add: Suc.premis(2))
  have 4: ( $\text{ff}(0) < j \wedge j \leq \text{ff}(k)$ )  $\vee$  ( $\text{ff}(k) < j \wedge j \leq \text{ff}(\text{Suc } k)$ )
    using 1 2 Suc.hyps by blast
  have 41:  $\text{ff}(0) < j$ 
    using 2 4 Suc.hyps order-refl by force
  have 42:  $j \leq \text{ff}(\text{Suc } k)$ 
    using 3 4 by linarith
  have 5:  $\text{ff}(0) < j \wedge j \leq \text{ff}(\text{Suc } k)$ 
    by (simp add: 41 42)
  show ?thesis
    by (simp add: 5)
qed
qed

```

**lemma** *cover-a:*

```

assumes ( $\forall j.$ 
          ( $j \leq \text{ff } 0$ )  $\vee$  ( $\exists i < k. \text{ff}(i) < (j::\text{nat}) \wedge j \leq \text{ff}(\text{Suc } i)$ )
           $\longrightarrow gg j$ )
          ( $\forall i < k. \text{ff}(i) < \text{ff}(\text{Suc } i)$ )
shows ( $\forall j < \text{ff } k. gg j$ )
proof –
  have 1: ( $\forall j < \text{ff } 0. gg j$ )
    by (simp add: assms(1))
  have 2: ( $\forall j. \text{ff } 0 < j \wedge j \leq \text{ff } k \longrightarrow gg j$ )
    proof
      fix j
      show  $\text{ff } 0 < j \wedge j \leq \text{ff } k \longrightarrow gg j$ 
        using assms
        proof (induct k arbitrary: j)
          case 0
          then show ?case by simp
          next

```

```

case (Suc k)
then show ?case
  proof –
    have 21: (∀ j.
      (j ≤ ff 0) ∨ (∃ i < k. ff(i) < (j::nat) ∧ j ≤ ff(Suc i))
        ∨ (ff k < j ∧ j ≤ ff (Suc k) )
      → gg j)
    using Suc.prem1(1) less-SucI by blast
    have 22: (∀ i < k. ff(i) < ff(Suc i))
    using Suc.prem1(2) by auto
    have 23: ff k < ff (Suc k)
    by (simp add: Suc.prem1(2))
    have 24: (∀ j.
      (j ≤ ff 0) ∨ (ff 0 < j ∧ j ≤ ff k)
        ∨ (ff k < j ∧ j ≤ ff (Suc k) )
      → gg j)
    using 21 22 Suc.hyps by blast
    have 25: ff 0 < j ∧ j ≤ ff (Suc k) → gg j
    using 24 not-less by blast
    show ?thesis using 25 by blast
  qed
qed
qed
show ?thesis
by (metis 2 assms(1) less-or-eq-imp-le linorder-neqE-nat)
qed

```

```

lemma PiUntilDistsem3:
assumes (σ ⊨ f Π (g U h))
shows (σ ⊨ ( f Π g ) U ( f Π h ) )
proof –
  have 1: ((∃ i ≤ ilen σ. f (suffix i σ)) ∧
    (∃ k ≤ ilen (ifilter f (suffixes σ)).
      h (suffix k (imap (λs. inth s 0) (ifilter f (suffixes σ)))))) ∧
    (∀ j < k. g (suffix j (imap (λs. inth s 0) (ifilter f (suffixes σ))))))
  using assms PiUntilDistsem1 by blast
  have 2: ∃ x ∈ iset(suffixes σ). f x
  using 1 in-iset-suffixes osfx-suffix by blast
  have 3: ∀ x ∈ iset(nfilter f (suffixes σ) 0). f (inth (suffixes σ) x)
  by (simp add: 2)
  have 4: (∃ k ≤ ilen (ifilter f (suffixes σ)).
    h (suffix k (imap (λs. inth s 0) (ifilter f (suffixes σ)))) ∧
    (∀ j < k. g (suffix j (imap (λs. inth s 0) (ifilter f (suffixes σ))))))
  using 1 by auto
  obtain k where 5: k ≤ ilen (ifilter f (suffixes σ)) ∧
    h (suffix k (imap (λs. inth s 0) (ifilter f (suffixes σ)))) ∧
    (∀ j < k. g (suffix j (imap (λs. inth s 0) (ifilter f (suffixes σ))))))
  using 4 by auto
  have 6: f (suffix (inth (nfilter f (suffixes σ) 0) k) σ)

```

```

using 2 3 5 nfilter-upper-bound[of suffixes  $\sigma$   $f$   $k$  0]
by (metis (no-types, lifting) 2 3 5 add.left-neutral nfilter-ilen inth-iset inth-suffixes)
have 7:  $k=0 \longrightarrow (\exists i \leq \text{ilen } \sigma - k. f (\text{suffix } (i + k) \sigma)) \wedge$ 
   $h (\text{imap } (\lambda s. \text{inth } s \ 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } k \sigma)))) \wedge$ 
   $(\forall j < k. (\exists i \leq \text{ilen } \sigma - j. f (\text{suffix } (i + j) \sigma)) \wedge$ 
     $g (\text{imap } (\lambda s. \text{inth } s \ 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } j \sigma)))))$ 
using 1 5 by auto
have 71:  $k > 0 \longrightarrow (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) (k - 1)) \in \text{iset}(\text{nfilter } f (\text{suffixes } \sigma) \ 0)$ 
  by (metis 2 5 diff-le-self le-trans nfilter-ilen inth-iset)
have 8:  $k > 0 \longrightarrow f (\text{suffix } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) (k - 1)) \ \sigma)$ 
  by (metis 2 3 71 add.left-neutral inth-and-iset nfilter-upper-bound inth-suffixes)
have 9:  $k > 0 \longrightarrow (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) (k - 1)) < (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ k)$ 
  by (metis 2 5 One-nat-def Suc-diff-Suc Suc-le-lessD diff-zero idx-nfilter-mono
    nfilter-ilen)
have 10:  $k > 0 \longrightarrow (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) (k - 1)) \leq \text{ilen } \sigma$ 
  using nfilter-upper-bound[of suffixes  $\sigma$   $f$   $k-1$  0]
  by (simp add: 2 5 Suc-leD nfilter-ilen)
have 11:  $k > 0 \longrightarrow (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ k) \leq \text{ilen } \sigma$ 
  using nfilter-upper-bound[of suffixes  $\sigma$   $f$   $k$  0]
  by (simp add: 2 5 nfilter-ilen)
have 12:  $k > 0 \longrightarrow$ 
   $h (\text{imap } (\lambda s. \text{inth } s \ 0)$ 
     $(\text{ifilter } f (\text{suffixes } (\text{suffix } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ k) \ \sigma))))$ 
  using ifilter-nfilter-suffix[of  $f$  (suffixes  $\sigma$ )  $k$ ]
  by (metis 11 5 6 ilen-suffixes imap-suffix inth-suffixes
    suffix-suffixes)
have 121:  $k > 0 \longrightarrow$ 
   $h (\text{imap } (\lambda s. \text{inth } s \ 0)$ 
     $(\text{ifilter } f (\text{suffixes } (\text{suffix } (\text{Suc } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) (k-1))) \ \sigma))))$ 
  by (metis 2 5 Suc-diff-1 ifilter-suffixes-imap)
have 13:  $k > 0 \longrightarrow (\exists i \leq \text{ilen } \sigma - (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ k).$ 
   $f (\text{suffix } (i + (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ k)) \ \sigma))$ 
  using 6 by auto
have 131:  $k > 0 \longrightarrow (\exists i \leq \text{ilen } \sigma - (\text{Suc } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) (k-1))).$ 
   $f (\text{suffix } (i + (\text{Suc } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) (k-1)))) \ \sigma))$ 
  using 11 6 9 diff-le-mono by fastforce
have 14:  $k > 0 \longrightarrow (\forall j < (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ k).$ 
   $(\exists i \leq \text{ilen } \sigma - j. f (\text{suffix } (i + j) \ \sigma)))$ 
  using 11 6 diff-le-mono by fastforce
have 141:  $k > 0 \longrightarrow (\forall j < (\text{Suc } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) (k-1))).$ 
   $(\exists i \leq \text{ilen } \sigma - j. f (\text{suffix } (i + j) \ \sigma)))$ 
  using 14 9 by auto
have 15:  $(\forall j < k. g (\text{suffix } j (\text{imap } (\lambda s. \text{inth } s \ 0) (\text{ifilter } f (\text{suffixes } \sigma))))$ 
  using 5 by blast
have 151:  $(\forall j < k. g (\text{imap } (\lambda s. \text{inth } s \ 0) (\text{suffix } j (\text{ifilter } f (\text{suffixes } \sigma))))$ 
  by (simp add: 5 less-le-trans less-or-eq-imp-le imap-suffix)
have 152:  $(\forall j < k. (\text{suffix } j (\text{ifilter } f (\text{suffixes } \sigma))) =$ 
   $(\text{ifilter } f (\text{suffix } (\text{inth } (\text{nfilter } f (\text{suffixes } \sigma) \ 0) \ j) (\text{suffixes } \sigma))))$ 
  by (meson 2 5 ifilter-nfilter-suffix-1 le-trans less-imp-le-nat)
have 16:  $(\forall j < k. g (\text{imap } (\lambda s. \text{inth } s \ 0)$ 

```



```

      (ifilter f (suffix (inth (nfilter f (suffixes σ) 0) j) (suffixes σ))))))
using 151 152 ifilter-nfilter-suffix by simp
have 1610: (inth (nfilter f (suffixes σ) 0) k) ≤ ilen(suffixes σ)
  by (metis 2 5 diff-zero ilen-gr-zero nfilter-ilen
    nfilter-upper-bound ordered-cancel-comm-monoid-diff-class.add-diff-inverse)
have 1611: (∀ j < k. (inth (nfilter f (suffixes σ) 0) j) < (inth (nfilter f (suffixes σ) 0) k))
  by (simp add: 2 5 idx-nfilter-gr nfilter-ilen)
have 1612: (∀ j < k. (inth (nfilter f (suffixes σ) 0) j) ≤ ilen(suffixes σ))
  using 1610 1611 by auto
have 161: (∀ j < k. ( (suffix (inth (nfilter f (suffixes σ) 0) j) (suffixes σ))) =
  ( (suffixes (suffix (inth (nfilter f (suffixes σ) 0) j) σ))))
  using suffix-suffixes using 1612 by blast
have 17: (∀ j < k. g (imap (λs. inth s 0)
  (ifilter f (suffixes (suffix (inth (nfilter f (suffixes σ) 0) j) σ))))))
  using 16 161 by auto
have 18: k > 0 →
  g (imap (λs. inth s 0)
    (ifilter f (suffixes (suffix (inth (nfilter f (suffixes σ) 0) (k-1)) σ))))
  using 17 by simp
have 19: k > 0 →
  g (imap (λs. inth s 0)
    (ifilter f (suffixes (suffix (inth (nfilter f (suffixes σ) 0) 0) σ))))
  using 17 by blast
have 20: k > 0 → (ifilter f (suffixes (suffix (inth (nfilter f (suffixes σ) 0) 0) σ))) =
  (ifilter f (suffix (inth (nfilter f (suffixes σ) 0) 0) (suffixes σ)))
  using 161 by auto
have 21: k > 0 → (∀ j ≤ (inth (nfilter f (suffixes σ) 0) 0).
  ( (suffixes (suffix j σ))) =
  ( (suffix j (suffixes σ))))
  using 1612 le-trans suffix-suffixes by fastforce
have 22: k > 0 → (∀ j ≤ (inth (nfilter f (suffixes σ) 0) 0).
  (ifilter f (suffixes (suffix j σ))) =
  (ifilter f (suffix j (suffixes σ))))
  using 21 by auto
have 23: k > 0 →
  (∀ j ≤ (inth (nfilter f (suffixes σ) 0) 0).
    (imap (λs. inth s 0)
      (ifilter f (suffixes (suffix (inth (nfilter f (suffixes σ) 0) 0) σ)))) =
    (imap (λs. inth s 0) (ifilter f (suffixes (suffix j σ))))
  )
  by (simp add: 2 21 ifilter-suffixes-imap-help-0)
have 24: k > 0 →
  (∀ j ≤ (inth (nfilter f (suffixes σ) 0) 0).
    g (imap (λs. inth s 0)
      (ifilter f (suffixes (suffix (inth (nfilter f (suffixes σ) 0) 0) σ)))) =
    g (imap (λs. inth s 0) (ifilter f (suffixes (suffix j σ))))
  )
  using 23 by auto
have 241: k > 0 → (∀ j ≤ (inth (nfilter f (suffixes σ) 0) 0).
  g (imap (λs. inth s 0) (ifilter f (suffixes (suffix j σ)))) )

```

**using** 19 24 **by** blast  
**have** 25:  $k > 0 \longrightarrow (\forall i < k - 1.$   
 $(\forall l. l \leq \text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i) ) \wedge$   
 $(\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) i) < l \longrightarrow$   
 $(\text{ifilter } f (\text{suffix} (\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i)) (\text{suffixes } \sigma)) =$   
 $(\text{ifilter } f (\text{suffix } l (\text{suffixes } \sigma))) ) )$   
**proof**  
**assume**  $k > 0$   
**show**  $(\forall i < k - 1.$   
 $(\forall l. l \leq \text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i) ) \wedge$   
 $(\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) i) < l \longrightarrow$   
 $(\text{ifilter } f (\text{suffix} (\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i)) (\text{suffixes } \sigma)) =$   
 $(\text{ifilter } f (\text{suffix } l (\text{suffixes } \sigma))) ) )$   
**proof**  
**fix**  $i$   
**show**  $i < k - 1 \longrightarrow$   
 $(\forall l. l \leq \text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i) \wedge$   
 $\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) i < l \longrightarrow$   
 $\text{ifilter } f (\text{suffix} (\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i)) (\text{suffixes } \sigma)) =$   
 $\text{ifilter } f (\text{suffix } l (\text{suffixes } \sigma)))$   
**proof** –  
**have** 251:  $k = 1 \longrightarrow i < k - 1 \longrightarrow$   
 $(\forall l. l \leq \text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i) \wedge$   
 $\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) i < l \longrightarrow$   
 $\text{ifilter } f (\text{suffix} (\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i)) (\text{suffixes } \sigma)) =$   
 $\text{ifilter } f (\text{suffix } l (\text{suffixes } \sigma)))$   
**by** auto  
**have** 252:  $k > 1 \longrightarrow i < k - 1 \longrightarrow$   
 $(\forall l. l \leq \text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i) \wedge$   
 $\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) i < l \longrightarrow$   
 $\text{ifilter } f (\text{suffix} (\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i)) (\text{suffixes } \sigma)) =$   
 $\text{ifilter } f (\text{suffix } l (\text{suffixes } \sigma)))$   
**proof**  
**assume**  $k > 1$   
**show**  $i < k - 1 \longrightarrow$   
 $(\forall l. l \leq \text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i) \wedge$   
 $\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) i < l \longrightarrow$   
 $\text{ifilter } f (\text{suffix} (\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i)) (\text{suffixes } \sigma)) =$   
 $\text{ifilter } f (\text{suffix } l (\text{suffixes } \sigma)))$   
**proof**  
**assume**  $i < k - 1$   
**show**  $(\forall l. l \leq \text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i) \wedge$   
 $\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) i < l \longrightarrow$   
 $\text{ifilter } f (\text{suffix} (\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i)) (\text{suffixes } \sigma)) =$   
 $\text{ifilter } f (\text{suffix } l (\text{suffixes } \sigma)))$   
**proof**  
**fix**  $l$   
**show**  $l \leq \text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i) \wedge$   
 $\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) i < l \longrightarrow$

```

      ifilter f (suffix (inth (nfilter f (suffixes σ) 0) (Suc i)) (suffixes σ)) =
      ifilter f (suffix l (suffixes σ))
    using ifilter-suffixes-imap-help-j[of l f suffixes σ i]
    using 2 5 ⟨i < k - 1⟩ by linarith
  qed
  qed
  qed
  show ?thesis
  by (simp add: 252)
  qed
  qed
  qed
have 261: k > 0 ⟶ (∀ i < k - 1.
  (∀ l. l ≤ (inth (nfilter f (suffixes σ) 0) (Suc i)) ∧
    (inth (nfilter f (suffixes σ) 0) i) < l ⟶
    l ≤ ilen (suffixes σ)))

  by (metis 1612 Suc-diff-1 Suc-mono le-eq-less-or-eq less-le-trans)
have 262: k > 0 ⟶ (∀ i < k - 1.
  (∀ l. l ≤ (inth (nfilter f (suffixes σ) 0) (Suc i)) ∧
    (inth (nfilter f (suffixes σ) 0) i) < l ⟶
    (suffix l (suffixes σ)) = (suffixes (suffix l σ))))

  using 261 suffix-suffixes by blast
have 26: k > 0 ⟶ (∀ i < k - 1.
  (∀ l. l ≤ (inth (nfilter f (suffixes σ) 0) (Suc i)) ∧
    (inth (nfilter f (suffixes σ) 0) i) < l ⟶
    (imap (λs. inth s 0)
      (ifilter f (suffix (nth (nfilter f (suffixes σ) 0) (Suc i)) (suffixes σ)))) =
    (imap (λs. inth s 0) (ifilter f (suffixes (suffix l σ))) ) ) )

  using 25 262 by auto
have 27: k > 0 ⟶ (∀ i < k - 1.
  (∀ l. l ≤ (inth (nfilter f (suffixes σ) 0) (Suc i)) ∧
    (inth (nfilter f (suffixes σ) 0) i) < l ⟶
    g (imap (λs. inth s 0)
      (ifilter f
        (suffix (nth (nfilter f (suffixes σ) 0) (Suc i)) (suffixes σ))))))

  by (simp add: 16)
have 28: k > 0 ⟶ (∀ i < k - 1.
  (∀ l. l ≤ (inth (nfilter f (suffixes σ) 0) (Suc i)) ∧
    (inth (nfilter f (suffixes σ) 0) i) < l ⟶
    g (imap (λs. inth s 0) (ifilter f (suffixes (suffix l σ))) ) ) )

  using 25 262 27 by auto
have 281: k > 0 ⟶
  (∀ j.
    (j ≤ (inth (nfilter f (suffixes σ) 0) 0) ∨
    (∃ i. i < k - 1 ∧
      j ≤ (inth (nfilter f (suffixes σ) 0) (Suc i)) ∧
      (inth (nfilter f (suffixes σ) 0) i) < j) )
    ⟶ g (imap (λs. inth s 0) (ifilter f (suffixes (suffix j σ))) ) )

  using 241 28 by blast

```

**have** 282:  $k > 0 \longrightarrow (\forall i < k - 1.$   
 $\quad \text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) i < \text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (\text{Suc } i))$   
**by** (metis 2 5 Suc-diff-1 Suc-leI idx-nfilter-gr le-SucI le-trans lessI nfilter-ilen)  
**have** 29:  $k > 0 \longrightarrow (\forall j < (\text{Suc} (\text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) (k-1))).$   
 $\quad g (\text{imap} (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } j \sigma))))$   
**using** 281 282 cover-a[of  $\lambda x. \text{inth} (\text{nfilter } f (\text{suffixes } \sigma) 0) x \ k-1$   
 $\quad (\lambda j. g (\text{imap} (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } j \sigma))))]$   
**using** 18 less-antisym **by** blast  
**have** 30:  $k > 0 \longrightarrow (\exists k \leq \text{ilen } \sigma.$   
 $\quad (\exists i \leq \text{ilen } \sigma - k. f (\text{suffix } (i + k) \sigma)) \wedge$   
 $\quad h (\text{imap} (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } k \sigma)))) \wedge$   
 $\quad (\forall j < k. (\exists i \leq \text{ilen } \sigma - j. f (\text{suffix } (i + j) \sigma)) \wedge$   
 $\quad \quad g (\text{imap} (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } j \sigma)))))$   
**using** 29 121 131 141  
**by** (meson 11 9 Suc-leI less-le-trans)  
**have** 31:  $(\exists k \leq \text{ilen } \sigma.$   
 $\quad (\exists i \leq \text{ilen } \sigma - k. f (\text{suffix } (i + k) \sigma)) \wedge$   
 $\quad h (\text{imap} (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } k \sigma)))) \wedge$   
 $\quad (\forall j < k. (\exists i \leq \text{ilen } \sigma - j. f (\text{suffix } (i + j) \sigma)) \wedge$   
 $\quad \quad g (\text{imap} (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } j \sigma)))))$   
**using** 30 7 **by** blast  
**show** ?thesis **using** 31  
**by** (simp add: PiUntilDistsem2)  
**qed**

**lemma** PiUntilDistsem4:

**assumes**  $(\sigma \models (f \ \Pi \ g) \ \mathcal{U} \ (f \ \Pi \ h) )$   
**shows**  $(\sigma \models f \ \Pi \ (g \ \mathcal{U} \ h))$   
**proof** –  
**have** 1:  $(\exists k \leq \text{ilen } \sigma.$   
 $\quad (\exists i \leq \text{ilen } \sigma - k. f (\text{suffix } (i + k) \sigma)) \wedge$   
 $\quad h (\text{imap} (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } k \sigma)))) \wedge$   
 $\quad (\forall j < k. (\exists i \leq \text{ilen } \sigma - j. f (\text{suffix } (i + j) \sigma)) \wedge$   
 $\quad \quad g (\text{imap} (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } j \sigma)))))$   
**using** assms **by** (simp add: PiUntilDistsem2)  
**obtain** k **where** 2:  $k \leq \text{ilen } \sigma \wedge (\exists i \leq \text{ilen } \sigma - k. f (\text{suffix } (i + k) \sigma)) \wedge$   
 $\quad h (\text{imap} (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } k \sigma)))) \wedge$   
 $\quad (\forall j < k. (\exists i \leq \text{ilen } \sigma - j. f (\text{suffix } (i + j) \sigma)) \wedge$   
 $\quad \quad g (\text{imap} (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } (\text{suffix } j \sigma)))))$   
**using** 1 **by** auto  
**have** 3:  $(\exists i \leq \text{ilen } \sigma - k. f (\text{suffix } (i + k) \sigma))$   
**using** 2 **by** auto  
**have** 4:  $k \leq \text{ilen } \sigma$   
**using** 2 **by** auto  
**obtain** i **where** 5:  $i \leq \text{ilen } \sigma - k \wedge f (\text{suffix } (i + k) \sigma)$   
**using** 3 **by** auto  
**have** 6:  $i + k \leq \text{ilen } \sigma$   
**using** 4 5 Nat.le-diff-conv2 **by** blast

**have** 61:  $\exists x \in \text{iset}(\text{suffixes}(\text{suffix } k \ \sigma)). f \ x$   
**by** (*metis 5 in-iset-suffixes suffix-suffix osfx-suffix*)  
**have** 7:  $(\exists i \leq \text{ilen } \sigma. f(\text{suffix } i \ \sigma))$   
**using** 5 6 **by** *blast*  
**have** 71:  $\exists x \in \text{iset}(\text{suffixes } \sigma). f \ x$   
**using** 5 6 *in-iset-suffixes osfx-suffix* **by** *blast*  
**have** 72:  $\exists x \in \text{iset}(\text{nfilter } f(\text{suffixes } \sigma) \ 0). f(\text{inth}(\text{suffixes } \sigma) \ x)$   
**by** (*metis 5 6 cancel-comm-monoid-add-class.diff-cancel ilen-suffixes le-refl*  
*nfilter-holds-not-a inth-iset inth-suffixes*  
*ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)  
**have** 73:  $f(\text{suffix}(\text{inth}(\text{nfilter } f(\text{suffixes}(\text{suffix } k \ \sigma)) \ 0) \ 0) \ (\text{suffix } k \ \sigma))$   
**using** *nfilter-holds[of suffixes (suffix k σ) f 0]*  
**by** (*metis 61 add.left-neutral diff-zero inth-iset inth-suffixes nfilter-upper-bound zero-le*)  
**have** 74:  $f(\text{suffix}((\text{inth}(\text{nfilter } f(\text{suffixes}(\text{suffix } k \ \sigma)) \ 0) \ 0) + k) \ \sigma)$   
**using** 73 **by** *auto*  
**have** 75:  $f(\text{suffix } k(\text{suffix}(\text{inth}(\text{nfilter } f(\text{suffixes}(\text{suffix } k \ \sigma)) \ 0) \ 0) \ \sigma))$   
**by** (*metis 73 add commute suffix-suffix*)  
**have** 8:  $h(\text{imap}(\lambda s. \text{inth } s \ 0)(\text{ifilter } f(\text{suffixes}(\text{suffix } k \ \sigma))))$   
**using** 2 **by** *auto*  
**have** 9:  $(\text{ifilter } f(\text{suffixes}(\text{suffix } k \ \sigma))) = (\text{ifilter } f(\text{suffix } k(\text{suffixes } \sigma)))$   
**by** (*simp add: 4 suffix-suffixes*)  
**have** 10:  $h(\text{imap}(\lambda s. \text{inth } s \ 0)$   
 $(\text{suffix}(\text{ilen}(\text{ifilter } f(\text{suffixes } \sigma)) - \text{ilen}(\text{ifilter } f(\text{suffixes}(\text{suffix } k \ \sigma))))$   
 $(\text{ifilter } f(\text{suffixes } \sigma))))$   
**using** 2 61 *sfxfilter-suffix-suffix* **by** *fastforce*  
**have** 11:  $h(\text{suffix}(\text{ilen}(\text{ifilter } f(\text{suffixes } \sigma)) - \text{ilen}(\text{ifilter } f(\text{suffixes}(\text{suffix } k \ \sigma))))$   
 $(\text{imap}(\lambda s. \text{inth } s \ 0)(\text{ifilter } f(\text{suffixes } \sigma))))$   
**by** (*metis 10 diff-le-self imap-suffix*)  
**have** 12:  $(\forall j < k. (\exists i \leq \text{ilen } \sigma - j. f(\text{suffix}(i + j) \ \sigma)) \wedge$   
 $g(\text{imap}(\lambda s. \text{inth } s \ 0)(\text{ifilter } f(\text{suffixes}(\text{suffix } j \ \sigma))))$   
**using** 2 **by** *blast*  
**have** 13:  $(\forall j < k. (\exists x \in \text{iset}(\text{suffixes}(\text{suffix } j \ \sigma)). f \ x) \wedge$   
 $g(\text{imap}(\lambda s. \text{inth } s \ 0)(\text{ifilter } f(\text{suffixes}(\text{suffix } j \ \sigma))))$   
**by** (*metis 2 suffix-ilen suffix-suffix ilen-suffixes inth-iset inth-suffixes*)  
**have** 14:  $(\forall j < k. (\exists x \in \text{iset}(\text{suffixes}(\text{suffix } j \ \sigma)). f \ x) \wedge$   
 $g(\text{imap}(\lambda s. \text{inth } s \ 0)$   
 $(\text{suffix}(\text{ilen}(\text{ifilter } f(\text{suffixes } \sigma)) - \text{ilen}(\text{ifilter } f(\text{suffixes}(\text{suffix } j \ \sigma))))$   
 $(\text{ifilter } f(\text{suffixes } \sigma))))$   
**using** 13 *sfxfilter-suffix-suffix*  
**by** (*metis (no-types, lifting) 2 dual-order.strict-iff-order less-le-trans*)  
**have** 15:  $(\forall j < k. (\exists x \in \text{iset}(\text{suffixes}(\text{suffix } j \ \sigma)). f \ x) \wedge$   
 $g(\text{suffix}(\text{ilen}(\text{ifilter } f(\text{suffixes } \sigma)) - \text{ilen}(\text{ifilter } f(\text{suffixes}(\text{suffix } j \ \sigma))))$   
 $(\text{imap}(\lambda s. \text{inth } s \ 0)(\text{ifilter } f(\text{suffixes } \sigma))))$   
**by** (*metis 14 diff-le-self imap-suffix*)  
**have** 150:  $k=0 \longrightarrow (\forall jj < (\text{ilen}(\text{ifilter } f(\text{suffixes } \sigma)) - \text{ilen}(\text{ifilter } f(\text{suffixes}(\text{suffix } k \ \sigma))))$   
 $g(\text{suffix } jj(\text{imap}(\lambda s. \text{inth } s \ 0)(\text{ifilter } f(\text{suffixes } \sigma))))$   
**by** *auto*  
**have** 151:  $(\forall jj < (\text{ilen}(\text{ifilter } f(\text{suffixes } \sigma)) - \text{ilen}(\text{ifilter } f(\text{suffixes}(\text{suffix } k \ \sigma))))$   
 $(\text{suffix } jj(\text{ifilter } f(\text{suffixes } \sigma))) =$   
 $\text{ifilter } f(\text{suffixes}(\text{suffix}((\text{ilen } \sigma) - \text{ilen}(\text{inth}(\text{ifilter } f(\text{suffixes } \sigma)) \ jj)) \ \sigma))$

)  
 by (simp add: 71 sfxfilter-suffix-suffix-a)  
 have 153:  $(\forall jj < (\text{ilen}(\text{ifilter } f (\text{suffixes } \sigma)) - \text{ilen}(\text{ifilter } f (\text{suffixes } (\text{suffix } k \sigma))))$   
 $g (\text{imap } (\lambda s. \text{inth } s 0)$   
 $(\text{ifilter } f$   
 $(\text{suffixes } (\text{suffix } ((\text{ilen } \sigma) - \text{ilen}(\text{inth } (\text{ifilter } f (\text{suffixes } \sigma)) jj)) \sigma))))$   
 )  
 using 13 sfx-suffix-upperbound by blast  
 have 1530:  $(\forall jj < (\text{ilen}(\text{ifilter } f (\text{suffixes } \sigma)) - \text{ilen}(\text{ifilter } f (\text{suffixes } (\text{suffix } k \sigma))))$   
 $g (\text{imap } (\lambda s. \text{inth } s 0) (\text{suffix } jj (\text{ifilter } f (\text{suffixes } \sigma))))$   
 by (simp add: 151 153)  
 have 1531:  $(\forall jj < (\text{ilen}(\text{ifilter } f (\text{suffixes } \sigma)) - \text{ilen}(\text{ifilter } f (\text{suffixes } (\text{suffix } k \sigma))))$   
 $g (\text{suffix } jj (\text{imap } (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } \sigma))))$   
 by (metis 1530 diff-le-self suffix-ilen suffix-ilen-code less-le-trans  
 less-numeral-extra(3) imap-suffix zero-less-diff)  
 have 154:  $((\exists i \leq \text{ilen } \sigma. f (\text{suffix } i \sigma)) \wedge$   
 $(\exists k \leq \text{ilen}(\text{ifilter } f (\text{suffixes } \sigma)).$   
 $h (\text{suffix } k (\text{imap } (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } \sigma)))) \wedge$   
 $(\forall j < k. g (\text{suffix } j (\text{imap } (\lambda s. \text{inth } s 0) (\text{ifilter } f (\text{suffixes } \sigma))))))$   
 using 11 1531 7 diff-le-self by blast  
 show ?thesis  
 by (simp add: 154 PiUntilDistsem1)  
 qed

**lemma** *PiUntilDistsem*:

$\sigma \models f \Pi (g \mathcal{U} h) = (f \Pi g) \mathcal{U} (f \Pi h)$

using *PiUntilDistsem3 PiUntilDistsem4* using *unl-lift2* by blast

**lemma** *PiUntilDist*:

$\vdash f \Pi (g \mathcal{U} h) = (f \Pi g) \mathcal{U} (f \Pi h)$

using *PiUntilDistsem Valid-def* by blast

### 15.4.15 PiChopstar

**lemma** *wnextboxnotstatesem*:

**assumes**  $k \leq \text{ilen } \sigma$

**shows**  $(\forall j \leq \text{ilen } \sigma. k < j \longrightarrow \neg (\lambda y. w \langle y \rangle) (\text{inth } \sigma j)) =$   
 $(\text{LIFT}(wnext (\Box (\text{init } (\neg w)))) (\text{suffix } k \sigma))$

using *assms*

**proof** (auto simp add: always-defs init-defs wnext-defs)

**show**  $\bigwedge j. j \leq \text{ilen } \sigma \implies k < j \implies w \langle \text{inth } \sigma j \rangle \implies$   
 $\forall n \leq \text{ilen } \sigma - \text{Suc } k. \neg w \langle \text{inth } \sigma (\text{Suc } (n + k)) \rangle \implies \text{False}$

**proof** (cases k)

**show**  $\bigwedge j. j \leq \text{ilen } \sigma \implies k < j \implies w \langle \text{inth } \sigma j \rangle \implies$   
 $\forall n \leq \text{ilen } \sigma - \text{Suc } k. \neg w \langle \text{inth } \sigma (\text{Suc } (n + k)) \rangle \implies k = 0 \implies \text{False}$

by simp

(metis Suc-le-mono Suc-pred less-le-trans)

**show**  $\bigwedge j \text{ nat}. j \leq \text{ilen } \sigma \implies k < j \implies w \langle \text{inth } \sigma j \rangle \implies$   
 $\forall n \leq \text{ilen } \sigma - \text{Suc } k. \neg w \langle \text{inth } \sigma (\text{Suc } (n + k)) \rangle \implies k = \text{Suc } \text{nat} \implies \text{False}$

```

proof –
fix  $j :: \text{nat}$  and  $\text{nat} :: \text{nat}$ 
assume  $a1: k < j$ 
assume  $a2: \forall n \leq \text{ilen } \sigma. \text{Suc } k. \neg w \langle \text{inth } \sigma (\text{Suc } (n + k)) \rangle$ 
assume  $a3: j \leq \text{ilen } \sigma$ 
assume  $a4: w \langle \text{inth } \sigma j \rangle$ 
have  $\text{Suc } (k + (j - \text{Suc } k)) = j$ 
using  $a1$  by simp
then show False
using  $a4$   $a3$   $a2$  by (metis diff-add-inverse2 diff-le-mono
    linordered-semidom-class.add-diff-inverse not-add-less2)
qed
qed
qed

```

**lemma** *NotStateUntilStateAndsem:*

$(\sigma \models (\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge f)) =$   
 $(\exists k \leq \text{ilen } \sigma. w \langle \text{inth } \sigma k \rangle \wedge f (\text{suffix } k \sigma) \wedge (\forall j < k. \neg w \langle \text{inth } \sigma j \rangle))$

**by** (*auto simp add: until-d-def init-defs*)

**lemma** *StateUntilEqvWPrevChopsem:*

$\sigma \models (\text{init } w) \mathcal{U} f = (w_{\text{prev}} (\Box (\text{init } w))); f$

**by** (*auto simp add: min.absorb1 until-d-def wprev-defs always-defs init-defs chop-defs*)

**lemma** *StateUntilEqvWPrevChop:*

$\vdash (\text{init } w) \mathcal{U} f = (w_{\text{prev}} (\Box (\text{init } w))); f$

**using** *StateUntilEqvWPrevChopsem Valid-def* **by** *blast*

**lemma** *UntilChopDist:*

$\vdash (\text{init } w) \mathcal{U} (g; h) = ((\text{init } w) \mathcal{U} g); h$

**by** (*metis ChopAssoc StateUntilEqvWPrevChop inteq-reflection*)

**lemma** *PiEmptysem:*

$\sigma \models (\text{init } w) \Pi \text{empty} = (\text{init } (\neg w)) \mathcal{U} ((\text{init } w) \wedge w_{\text{next}} (\Box (\text{init } (\neg w))))$

**proof** –

**have**  $1: (\sigma \models (\text{init } w) \Pi \text{empty}) =$

$((\exists x \in \text{iset } \sigma. w \langle x \rangle) \wedge \text{ilen } (\text{ifilter } (\lambda y. w \langle y \rangle) \sigma) = 0)$

**by** (*simp add: init-defs empty-defs Pistate*)

**have**  $2: ((\exists x \in \text{iset } \sigma. w \langle x \rangle) \wedge \text{ilen } (\text{ifilter } (\lambda y. w \langle y \rangle) \sigma) = 0) =$

$(\exists k \leq \text{ilen } \sigma. (\lambda y. w \langle y \rangle) (\text{inth } \sigma k) \wedge$   
 $(\forall j. j < k \longrightarrow \neg (\lambda y. w \langle y \rangle) (\text{inth } \sigma j)) \wedge$   
 $(\forall j \leq \text{ilen } \sigma. k < j \longrightarrow \neg (\lambda y. w \langle y \rangle) (\text{inth } \sigma j)))$

**by** (*simp add: ifilter-ilen-zero-conv-2*)

**have**  $3: (\exists k \leq \text{ilen } \sigma. (\lambda y. w \langle y \rangle) (\text{inth } \sigma k) \wedge$

$(\forall j. j < k \longrightarrow \neg (\lambda y. w \langle y \rangle) (inth \sigma j)) \wedge$   
 $(\forall j \leq ilen \sigma. k < j \longrightarrow \neg (\lambda y. w \langle y \rangle) (inth \sigma j))) =$   
 $(\exists k \leq ilen \sigma.$   
 $w \langle inth \sigma k \rangle \wedge$   
 $(LIFT(wnext (\Box (init (\neg w)))))) (suffix k \sigma) \wedge$   
 $(\forall j < k. \neg w \langle inth \sigma j \rangle))$   
**using** *wnextboxnotstatesem*  
**by** *metis*  
**have** 4:  $(\exists k \leq ilen \sigma.$   
 $w \langle inth \sigma k \rangle \wedge$   
 $(LIFT(wnext (\Box (init (\neg w)))))) (suffix k \sigma) \wedge$   
 $(\forall j < k. \neg w \langle inth \sigma j \rangle)) =$   
 $(\sigma \models (init (\neg w)) \mathcal{U} ((init w) \wedge wnext (\Box (init (\neg w))))))$   
  
**by** (*simp add: NotStateUntilStateAndsem*)  
**from** 1 2 3 4 **show** ?thesis **by** *auto*  
**qed**

**lemma** *PiEmpty*:  
 $\vdash (init w) \Pi empty = (init (\neg w)) \mathcal{U} ((init w) \wedge wnext (\Box (init (\neg w))))$   
**using** *PiEmptysem Valid-def* **by** *blast*

**lemma** *StatePiBoxStatesem*:  
 $\sigma \models (init w) \Pi f = (init w) \Pi (f \wedge \Box (init w))$   
**proof** –  
**have** 1:  $(\sigma \models (init w) \Pi f) =$   
 $((\exists x \in iset \sigma. w \langle x \rangle) \wedge ((ifilter (\lambda y. w \langle y \rangle) \sigma) \models f))$   
  
**by** (*metis Pystate*)  
**have** 2:  $(\sigma \models (init w) \Pi (f \wedge \Box (init w))) =$   
 $((\exists x \in iset \sigma. w \langle x \rangle) \wedge ((ifilter (\lambda y. w \langle y \rangle) \sigma) \models f \wedge \Box (init w)))$   
  
**by** (*metis Pystate*)  
**have** 3:  $((ifilter (\lambda y. w \langle y \rangle) \sigma) \models f \wedge \Box (init w))$   
 $= (f (ifilter (\lambda y. w \langle y \rangle) \sigma) \wedge$   
 $(\forall n \leq ilen (ifilter (\lambda y. w \langle y \rangle) \sigma). w \langle inth (ifilter (\lambda y. w \langle y \rangle) \sigma) n \rangle))$   
**by** (*simp add: always-defs init-defs*)  
**have** 4:  $((\exists x \in iset \sigma. w \langle x \rangle) \wedge (f (ifilter (\lambda y. w \langle y \rangle) \sigma) \wedge$   
 $(\forall n \leq ilen (ifilter (\lambda y. w \langle y \rangle) \sigma). w \langle inth (ifilter (\lambda y. w \langle y \rangle) \sigma) n \rangle))) =$   
 $((\exists x \in iset \sigma. w \langle x \rangle) \wedge f (ifilter (\lambda y. w \langle y \rangle) \sigma))$   
  
**by** (*meson ifilter-inth-aa*)  
**show** ?thesis **using** 1 2 3 4 **by** *auto*  
**qed**

**lemma** *StatePiBoxState*:  
 $\vdash (init w) \Pi f = (init w) \Pi (f \wedge \Box (init w))$   
**using** *StatePiBoxStatesem Valid-def* **by** *blast*



**lemma** *StatePiUntil1*:

$\vdash (init (\neg w)) \mathcal{U} (init w \wedge (init w) \Pi f) =$   
 $(wprev (\Box (init (\neg w)))) ; (init w \wedge (init w) \Pi f)$

**using** *StateUntilEqvWPrevChop* **by** *blast*

**lemma** *StatePiUntilsem2*:

$(\sigma \models (wprev (\Box (init (\neg w)))) ; (init w \wedge (init w) \Pi f)) =$   
 $(\sigma \models ((wprev (\Box (init (\neg w)))) ; ((init w) \wedge empty)) ; ((init w) \wedge (init w) \Pi f))$

**by** (*auto simp add: chop-defs init-defs empty-defs min.absorb1*)  
*fastforce*

**lemma** *StatePiUntil2*:

$\vdash (wprev (\Box (init (\neg w)))) ; (init w \wedge (init w) \Pi f) =$   
 $((wprev (\Box (init (\neg w)))) ; ((init w) \wedge empty)) ; ((init w) \wedge (init w) \Pi f)$

**by** (*simp add: StatePiUntilsem2 Valid-def*)

**lemma** *StatePiUntil3*:

$\vdash ((wprev (\Box (init (\neg w)))) ; ((init w) \wedge empty)) =$   
 $((init (\neg w)) \mathcal{U} ((init w) \wedge wnext (\Box (init (\neg w)))) ; ((init w) \wedge empty))$

**proof** –

**have** 1:  $\vdash (wprev (\Box (init (\neg w)))) ; ((init w) \wedge empty) =$   
 $(init (\neg w)) \mathcal{U} ((init w) \wedge empty)$

**by** (*meson Prop11 StateUntilEqvWPrevChop*)

**have** 2:  $\vdash ((init w) \wedge empty) = ((init w) \wedge wnext (\Box (init (\neg w)))) ; ((init w) \wedge empty)$

**by** (*auto simp add: min.absorb1 Valid-def init-defs empty-defs wnext-defs always-defs chop-defs*)  
*(metis Suc-pred eq-imp-le ilen-gr-zero le-neq-trans)*

**show** *?thesis* **by** (*metis 1 2 UntilChopDist inteq-reflection*)

**qed**

**lemma** *StatePiUntilsem4*:

$(\sigma \models ((init (\neg w)) \mathcal{U} ((init w) \wedge wnext (\Box (init (\neg w)))))) ; ((init w) \wedge empty) =$   
 $(\sigma \models ((init w) \Pi empty)) ; ((init w) \wedge empty)$

**by** (*metis PiEmpty inteq-reflection*)

**lemma** *StatePiUntil4*:

$\vdash ((init (\neg w)) \mathcal{U} ((init w) \wedge wnext (\Box (init (\neg w)))) ; ((init w) \wedge empty) =$   
 $((init w) \Pi empty) ; ((init w) \wedge empty)$

**by** (*simp add: StatePiUntilsem4 Valid-def*)

**lemma** *StatePiUntilsem*:

$\sigma \models (init w) \Pi f = (init (\neg w)) \mathcal{U} (init w \wedge (init w) \Pi f)$

**proof** –

**have** 2:  $(\sigma \models (init (\neg w)) \mathcal{U} (init w \wedge (init w) \Pi f)) =$   
 $(\sigma \models (wprev (\Box (init (\neg w)))) ; (init w \wedge (init w) \Pi f))$

**using** *StateUntilEqvWPrevChopsem*[*of LIFT*( $\neg w$ ) *LIFT*(( $init w$ )  $\wedge$  ( $init w$ )  $\Pi f$ )  $\sigma$ ]

**by** *simp*

**have** 7:  $(\sigma \models (wprev (\Box (init (\neg w)))) ; (init w \wedge (init w) \Pi f)) =$

$(\sigma \models (((init\ w) \Pi\ empty); ((init\ w) \wedge empty)); (init\ w) \wedge (init\ w) \Pi\ f))$   
**by** (*metis PiEmpty StatePiUntil2 StatePiUntil3 inteq-reflection*)  
**have** 8:  $(\sigma \models (((init\ w) \Pi\ empty); ((init\ w) \wedge empty)); (init\ w) \wedge (init\ w) \Pi\ f)) =$   
 $(\sigma \models (((init\ w) \Pi\ empty); (init\ w) \wedge (init\ w) \Pi\ f))$   
**by** (*auto simp add: chop-defs init-defs empty-defs min.absorb1*)  
*fastforce*  
**have** 9:  $(\sigma \models (((init\ w) \Pi\ empty)); (init\ w) \wedge (init\ w) \Pi\ f)) =$   
 $(\sigma \models (init\ w) \Pi\ (empty;f))$   
**using** *PiChopDistsema PiChopDistsemb* **by** *blast*  
**have** 10:  $(\sigma \models (init\ w) \Pi\ (empty;f)) = (\sigma \models (init\ w) \Pi\ f)$   
**by** (*metis chop-defs empty-defs prefix-ilen-good suffix-zero pi-d-def zero-order(1)*)  
**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*:  
 $\vdash ((init\ w) \wedge (init\ w) \Pi\ empty) = (w \wedge empty) ; (wnext\ (\Box\ (init\ (\neg w))))$   
**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*)  
*force*  
**have** 3:  $\vdash ((init\ w) \wedge (wnext\ (\Box\ (init\ (\neg w)))) = (w \wedge empty) ; (wnext\ (\Box\ (init\ (\neg w))))$   
**by** (*metis InitAndEmptyEqvAndEmpty StateAndEmptyChop inteq-reflection*)  
**show** *?thesis*  
**using** 1 2 3 **by** *fastforce*  
**qed**

**lemma** *PiPowerExpandsem*:  
 $(\sigma \models (\exists k. (init\ w) \Pi\ (power\ f\ k))) =$   
 $(\sigma \models (init\ w) \Pi\ empty \vee (\exists k. (init\ w) \Pi\ (f;(power\ f\ (k)))))$   
**proof** –  
**have** 1:  $(\sigma \models (\exists k. (init\ w) \Pi\ (power\ f\ k))) =$   
 $(\exists k. (\sigma \models (init\ w) \Pi\ (power\ f\ k)))$   
**by** *simp*  
**have** 2:  $(\exists k. (\sigma \models (init\ w) \Pi\ (power\ f\ k))) =$   
 $(\sigma \models (init\ w) \Pi\ (power\ f\ 0)) \vee (\exists k. 1 \leq k \wedge (\sigma \models (init\ w) \Pi\ (power\ f\ (k))))$   
**by** (*metis One-nat-def diff-Suc-1 le-SucE le-add1 plus-1-eq-Suc*)

```

have 3: ( $\sigma \models (\text{init } w) \sqcap (\text{power } f \ 0)$ ) = ( $\sigma \models (\text{init } w) \sqcap \text{empty}$ )
  by simp
have 4: ( $\exists k. 1 \leq k \wedge (\sigma \models (\text{init } w) \sqcap (\text{power } f \ (k)))$ ) =
  ( $\exists k. (\sigma \models (\text{init } w) \sqcap (\text{power } f \ (\text{Suc } k)))$ )
  by (metis le-add1 ordered-cancel-comm-monoid-diff-class.add-diff-inverse plus-1-eq-Suc)
have 5: ( $\exists k. (\sigma \models (\text{init } w) \sqcap (\text{power } f \ (\text{Suc } k)))$ ) =
  ( $\sigma \models (\exists k. (\text{init } w) \sqcap (\text{power } f \ (\text{Suc } k)))$ )

  by simp
have 6: (( $\sigma \models (\text{init } w) \sqcap \text{empty}$ )  $\vee$  ( $\sigma \models (\exists k. (\text{init } w) \sqcap (\text{power } f \ (\text{Suc } k)))$ )) =
  ( $\sigma \models (\text{init } w) \sqcap \text{empty} \vee (\exists k. (\text{init } w) \sqcap (f; \text{power } f \ (k)))$ )
  by auto
show ?thesis
using 1 2 3 4 5 6 by blast
qed

```

**lemma** *PiPowerExpandsem1*:

```

 $\forall \sigma. \sigma \models (\exists k. (\text{init } w) \sqcap (\text{power } f \ k)) =$ 
  ( $(\text{init } w) \sqcap \text{empty} \vee (\exists k. (\text{init } w) \sqcap (\text{power } f \ (\text{Suc } k)))$ )
proof
  fix  $\sigma$ 
  show  $\sigma \models (\exists k. (\text{init } w) \sqcap (\text{power } f \ k)) =$ 
    ( $(\text{init } w) \sqcap \text{empty} \vee (\exists k. (\text{init } w) \sqcap (\text{power } f \ (\text{Suc } k)))$ )
  proof –
    have 1: ( $\sigma \models (\exists k. (\text{init } w) \sqcap (\text{power } f \ k))$ ) =
      ( $(\text{init } w) \sqcap \text{empty} \vee (\exists k. (\text{init } w) \sqcap (\text{power } f \ (\text{Suc } k)))$ )
      = ( $\sigma \models (\exists k. (\text{init } w) \sqcap (\text{power } f \ k))$ ) =
      ( $\sigma \models (\text{init } w) \sqcap \text{empty} \vee (\exists k. (\text{init } w) \sqcap (\text{power } f \ (\text{Suc } k)))$ )

    by auto
    have 2: ( $\sigma \models (\exists k. (\text{init } w) \sqcap (\text{power } f \ k))$ ) =
      ( $\sigma \models (\text{init } w) \sqcap \text{empty} \vee (\exists k. (\text{init } w) \sqcap (\text{power } f \ (\text{Suc } k)))$ )
      using PiPowerExpandsem[of w f  $\sigma$ ] by simp
    show ?thesis
    using 1 2 by blast
  qed
qed

```

**lemma** *PiPowerExpand*:

```

 $\vdash (\exists k. (\text{init } w) \sqcap (\text{power } f \ k)) =$ 
  ( $(\text{init } w) \sqcap \text{empty} \vee (\exists k. (\text{init } w) \sqcap (\text{power } f \ (\text{Suc } k)))$ )
using PiPowerExpandsem1[of w f ] by (auto simp add: Valid-def PiPowerExpandsem1)

```

**lemma** *exists-expand-sem*:

```

( $\sigma \models (\exists k. (\text{power } ((\text{init } w) \sqcap f \wedge \text{fin } w) \ k))$ ) =
  (( $\sigma \models (\text{power } ((\text{init } w) \sqcap f \wedge \text{fin } w) \ 0)$ )  $\vee$ 
  ( $\sigma \models (\exists k. (\text{power } ((\text{init } w) \sqcap f \wedge \text{fin } w) \ (\text{Suc } k)))$ ))
by (metis (no-types, lifting) not0-implies-Suc unl-Rex)

```

**lemma** *exists-expand*:

$\vdash (\exists k. (\text{power } ((\text{init } w) \Pi f \wedge \text{fin } w) k)) =$   
 $((\text{power } ((\text{init } w) \Pi f \wedge \text{fin } w) 0) \vee (\exists k. (\text{power } ((\text{init } w) \Pi f \wedge \text{fin } w) (\text{Suc } k))))$

**using** *exists-expand-sem Valid-def* **by** *fastforce*

#### 15.4.16 TruePiEqv

**lemma** *TruePiEqvsem*:

$\sigma \models \# \text{True} \Pi f = f$

**by** (*simp add: pi-d-def pifilt-true*) *auto*

**lemma** *TruePiEqv*:

$\vdash (\# \text{True}) \Pi f = f$

**using** *TruePiEqvsem* **by** (*auto simp add: Valid-def*)

#### 15.4.17 BoxImpEqvPi

**lemma** *BoxImpEqvPi*:

$\vdash \Box f \longrightarrow g = f \Pi g$

**by** (*simp add: Valid-def always-defs pi-d-def pifilt-def sfxfilt-def*)  
*(metis ifilter-True ilen-gr-zero inth-and-iset ilen-suffixes*  
*imap-first-suffixes inth-suffixes)*

#### 15.4.18 PiEqvDiamondUPi

**lemma** *PiEqvDiamondUPi*:

$\vdash f \Pi g = (\Diamond f \wedge f \Pi^u g)$

**by** (*simp add: Valid-def upi-d-def sometimes-defs pi-d-def,blast*)

#### 15.4.19 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*)

#### 15.4.20 UPiEqvBoxOrPi

**lemma** *UPiEqvBoxOrPi*:

$\vdash f \Pi^u g = (\Box (\neg f) \vee f \Pi g)$

**by** (*simp add: Valid-def upi-d-def always-defs pi-d-def,blast*)

### 15.5 Theorems

**lemma** *UPiImpRule*:

**assumes**  $\vdash g1 \longrightarrow g2$

**shows**  $\vdash f \Pi^u g1 \longrightarrow f \Pi^u g2$

**using** *assms*

**by** (*meson MP PiK PiN*)

**lemma** *UPiEqvRule*:

**assumes**  $\vdash g1 = g2$

**shows**  $\vdash f \Pi^u g1 = f \Pi^u g2$   
**proof** –  
**have** 1:  $\vdash g1 \longrightarrow g2$   
**using** *assms* **by** (*simp add: int-iffD1*)  
**have** 2:  $\vdash f \Pi^u g1 \longrightarrow f \Pi^u g2$   
**using** 1 *UPiImpRule* **by** *blast*  
**have** 3:  $\vdash g2 \longrightarrow g1$   
**using** *assms* **by** (*simp add: int-iffD2*)  
**have** 4:  $\vdash f \Pi^u g2 \longrightarrow f \Pi^u g1$   
**using** 3 *UPiImpRule* **by** *blast*  
**from** 3 4 **show** *?thesis*  
**by** (*simp add: 2 int-iffI*)  
**qed**

**lemma** *PiEqvNotUPiNot*:  
 $\vdash f \Pi g = (\neg (f \Pi^u (\neg g)))$   
**by** (*simp add: upi-d-def*)

**lemma** *NotPiEqvNotUPi*:  
 $\vdash f \Pi (\neg g) = (\neg (f \Pi^u g))$   
**by** (*simp add: upi-d-def*)

**lemma** *UPiEqvNotPiNot*:  
 $\vdash f \Pi^u g = (\neg (f \Pi (\neg g)))$   
**by** (*simp add: upi-d-def*)

**lemma** *NotUPiEqvNotPi*:  
 $\vdash f \Pi^u (\neg g) = (\neg (f \Pi g))$   
**by** (*simp add: upi-d-def*)

**lemma** *PiImpRule*:  
**assumes**  $\vdash g1 \longrightarrow g2$   
**shows**  $\vdash f \Pi g1 \longrightarrow f \Pi g2$   
**proof** –  
**have** 1:  $\vdash \neg g2 \longrightarrow \neg g1$   
**by** (*simp add: assms*)  
**have** 2:  $\vdash f \Pi^u (\neg g2) \longrightarrow f \Pi^u (\neg g1)$   
**using** 1 *UPiImpRule* **by** *blast*  
**have** 3:  $\vdash \neg(f \Pi^u (\neg g1)) \longrightarrow \neg(f \Pi^u (\neg g2))$   
**using** 2 **by** *fastforce*  
**from** 3 **show** *?thesis* **using** *PiEqvNotUPiNot* **by** *fastforce*  
**qed**

**lemma** *PiEqvRule*:  
**assumes**  $\vdash g1 = g2$   
**shows**  $\vdash f \Pi g1 = f \Pi g2$   
**proof** –  
**have** 1:  $\vdash g1 \longrightarrow g2$   
**using** *assms* **by** (*simp add: int-iffD1*)  
**have** 2:  $\vdash f \Pi g1 \longrightarrow f \Pi g2$

**using** 1 *PiImpRule* **by** *blast*  
**have** 3:  $\vdash g2 \longrightarrow g1$   
**using** *assms* **by** (*simp add: int-iffD2*)  
**have** 4:  $\vdash f \Pi g2 \longrightarrow f \Pi g1$   
**using** 3 *PiImpRule* **by** *blast*  
**from** 2 4 **show** ?thesis **by** (*simp add: int-iffI*)  
**qed**

**lemma** *UPiAndPiImpPiAnd*:  
 $\vdash f1 \Pi^u f \wedge f1 \Pi (\neg g) \longrightarrow f1 \Pi (f \wedge \neg g)$   
**proof** –  
**have** 1:  $\vdash (\neg(f \longrightarrow g)) = (f \wedge \neg g)$   
**by** *fastforce*  
**have** 2:  $\vdash (\neg(f1 \Pi^u (f \longrightarrow g))) = f1 \Pi (\neg(f \longrightarrow g))$   
**by** (*simp add: NotPiEqvNotUPi int-iffD1 int-iffD2 int-iffI*)  
**have** 3:  $\vdash \neg(f1 \Pi^u f \longrightarrow f1 \Pi^u g) \longrightarrow \neg(f1 \Pi^u (f \longrightarrow g))$   
**by** (*simp add: PiK*)  
**have** 4:  $\vdash (\neg(f1 \Pi^u f \longrightarrow f1 \Pi^u g)) = (f1 \Pi^u f \wedge f1 \Pi (\neg g))$   
**using** *NotPiEqvNotUPi*[of *f1 g*] **by** *fastforce*  
**have** 5:  $\vdash f1 \Pi (\neg(f \longrightarrow g)) = f1 \Pi (f \wedge \neg g)$   
**using** 1 **by** (*simp add: PiEqvRule*)  
**from** 1 2 3 4 5 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *UPiAndPiImpPiAndA*:  
 $\vdash f1 \Pi^u f \wedge f1 \Pi g \longrightarrow f1 \Pi (f \wedge g)$   
**using** *UPiAndPiImpPiAnd*[of *f1 f LIFT*( $\neg g$ )] **by** *fastforce*

**lemma** *PiAndPiImpPiAnd*:  
 $\vdash f1 \Pi f \wedge f1 \Pi g \longrightarrow f1 \Pi (f \wedge g)$   
**proof** –  
**have** 1:  $\vdash f1 \Pi^u f \wedge f1 \Pi g \longrightarrow f1 \Pi (f \wedge g)$   
**using** *UPiAndPiImpPiAndA* **by** *fastforce*  
**have** 2:  $\vdash f1 \Pi f \longrightarrow f1 \Pi^u f$   
**using** *PiDc* **by** *blast*  
**from** 1 2 **show** ?thesis **by** *fastforce*  
**qed**

**lemma** *PiAnd*:  
 $\vdash f \Pi (g1 \wedge g2) = (f \Pi g1 \wedge f \Pi g2)$   
**proof** –  
**have** 1:  $\vdash f \Pi (g1 \wedge g2) \longrightarrow f \Pi g1$   
**by** (*meson PiImpRule Prop12 int-iffD1 lift-and-com*)  
**have** 2:  $\vdash f \Pi (g1 \wedge g2) \longrightarrow f \Pi g2$   
**by** (*meson PiImpRule Prop12 int-iffD1 lift-and-com*)  
**have** 3:  $\vdash f \Pi (g1 \wedge g2) \longrightarrow f \Pi g1 \wedge f \Pi g2$   
**using** 1 2 **by** *fastforce*

**have** 4:  $\vdash f \Pi g1 \wedge f \Pi g2 \longrightarrow f \Pi (g1 \wedge g2)$   
**by** (*simp add: PiAndPiImpPiAnd*)  
**from** 3 4 **show** ?thesis **by** fastforce  
**qed**

**lemma** *UPiAnd*:

$\vdash f \Pi^u (g1 \wedge g2) = (f \Pi^u g1 \wedge f \Pi^u g2)$   
**proof** –  
**have** 1:  $\vdash f \Pi (\neg g1 \vee \neg g2) = (f \Pi (\neg g1) \vee f \Pi (\neg g2))$   
**by** (*simp add: PiOr*)  
**have** 2:  $\vdash (\neg(f \Pi (\neg g1 \vee \neg g2))) = (\neg(f \Pi (\neg g1) \vee f \Pi (\neg g2)))$   
**using** 1 **by** fastforce  
**have** 3:  $\vdash (\neg(f \Pi (\neg g1 \vee \neg g2))) = f \Pi^u (\neg(\neg g1 \vee \neg g2))$   
**by** (*meson NotUPiEqvNotPi Prop11*)  
**have** 4:  $\vdash (\neg(\neg g1 \vee \neg g2)) = (g1 \wedge g2)$   
**by** fastforce  
**have** 5:  $\vdash f \Pi^u (\neg(\neg g1 \vee \neg g2)) = f \Pi^u (g1 \wedge g2)$   
**using** 4 **by** (*simp add: UPiEqvRule*)  
**have** 6:  $\vdash (\neg(f \Pi (\neg g1) \vee f \Pi (\neg g2))) = (\neg(f \Pi (\neg g1)) \wedge \neg(f \Pi (\neg g2)))$   
**by** fastforce  
**have** 7:  $\vdash \neg(f \Pi (\neg g1)) = f \Pi^u g1$   
**by** (*simp add: NotPiEqvNotUPi*)  
**have** 8:  $\vdash \neg(f \Pi (\neg g2)) = f \Pi^u g2$   
**by** (*simp add: NotPiEqvNotUPi*)  
**have** 9:  $\vdash (\neg(f \Pi (\neg g1)) \wedge \neg(f \Pi (\neg g2))) = (f \Pi^u g1 \wedge f \Pi^u g2)$   
**using** 6 7 8 **by** fastforce  
**from** 2 3 5 6 9 **show** ?thesis **by** fastforce  
**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*:

$\vdash (init\ w)\ \Pi\ f = (wprev\ (\Box (init\ (\neg w)))); ((init\ w) \wedge (init\ w)\ \Pi\ f)$

**using** *StatePiUntil StatePiUntil1* **by** *fastforce*

**lemma** *PiChopstarhelp2a*:

$\vdash (w \wedge empty); (power\ (f; (w \wedge empty)) \wedge more)\ k =$   
 $(power\ (((w \wedge empty); f) \wedge more)\ k) ; (w \wedge empty)$

**proof** (*induction k*)

**case** 0

**then show** *?case*

**by** (*metis ChopEmpty EmptyChop inteq-reflection pow-0*)

**next**

**case** (*Suc k*)

**then show** *?case*

**proof** –

**have** 1:  $\vdash (w \wedge empty); power\ (f; (w \wedge empty) \wedge more)\ (Suc\ k) =$   
 $(w \wedge empty); ((f; (w \wedge empty) \wedge more); power\ (f; (w \wedge empty) \wedge more)\ k)$

**by** *simp*

**have** 11:  $\vdash (f \wedge more); (w \wedge empty) = (fin\ w \wedge (f \wedge more))$

**by** (*metis FinEqvTrueChopAndEmpty TrueChopAndEmptyEqvChopAndEmpty inteq-reflection*)

**have** 12:  $\vdash (f; (w \wedge empty)) = (fin\ w \wedge f)$

**by** (*meson AndFinEqvChopAndEmpty Prop04 lift-and-com*)

**have** 13:  $\vdash (f; (w \wedge empty) \wedge more) = ((fin\ w \wedge f) \wedge more)$

**using** 12 **by** *auto*

**have** 14:  $\vdash ((fin\ w \wedge f) \wedge more) = (fin\ w \wedge (f \wedge more))$

**by** *fastforce*

**have** 2:  $\vdash (f; (w \wedge empty) \wedge more) = (f \wedge more); (w \wedge empty)$

**using** 11 13 **by** *fastforce*

**have** 20:  $\vdash ((f; (w \wedge empty) \wedge more); power\ (f; (w \wedge empty) \wedge more)\ k) =$   
 $((f \wedge more); (w \wedge empty)); power\ (f; (w \wedge empty) \wedge more)\ k$



by (simp add: 2 LeftChopEqvChop)  
 have 21:  $\vdash ((f \wedge \text{more}); (w \wedge \text{empty})); \text{power } (f; (w \wedge \text{empty}) \wedge \text{more}) k =$   
 $(f \wedge \text{more}); ((w \wedge \text{empty}); \text{power } (f; (w \wedge \text{empty}) \wedge \text{more}) k)$   
 using ChopAssocB by blast  
 have 22:  $\vdash (f \wedge \text{more}); ((w \wedge \text{empty}); \text{power } (f; (w \wedge \text{empty}) \wedge \text{more}) k) =$   
 $((f \wedge \text{more}); (\text{power } ((w \wedge \text{empty}); f \wedge \text{more}) k; (w \wedge \text{empty})))$   
 by (simp add: RightChopEqvChop Suc.IH)  
 have 3:  $\vdash (w \wedge \text{empty}); ((f; (w \wedge \text{empty}) \wedge \text{more}); \text{power } (f; (w \wedge \text{empty}) \wedge \text{more}) k) =$   
 $(w \wedge \text{empty}); ((f \wedge \text{more}); (\text{power } ((w \wedge \text{empty}); f \wedge \text{more}) k; (w \wedge \text{empty})))$   
 by (metis 1 2 21 22 inteq-reflection)  
 have 4:  $\vdash (w \wedge \text{empty}); ((f \wedge \text{more}); (\text{power } ((w \wedge \text{empty}); f \wedge \text{more}) k; (w \wedge \text{empty}))) =$   
 $((w \wedge \text{empty}); (f \wedge \text{more})); ((\text{power } ((w \wedge \text{empty}); f \wedge \text{more}) k; (w \wedge \text{empty})))$   
 using ChopAssoc by blast  
 have 5:  $\vdash (w \wedge \text{empty}); (f \wedge \text{more}) = ((w \wedge \text{empty}); f \wedge \text{more})$   
 by (auto simp add: Valid-def chop-defs empty-defs more-defs)  
 have 6:  $\vdash ((w \wedge \text{empty}); (f \wedge \text{more})); ((\text{power } ((w \wedge \text{empty}); f \wedge \text{more}) k; (w \wedge \text{empty}))) =$   
 $((w \wedge \text{empty}); f \wedge \text{more}); ((\text{power } ((w \wedge \text{empty}); f \wedge \text{more}) k; (w \wedge \text{empty})))$   
 using 5 LeftChopEqvChop by blast  
 have 7:  $\vdash ((w \wedge \text{empty}); f \wedge \text{more}); ((\text{power } ((w \wedge \text{empty}); f \wedge \text{more}) k; (w \wedge \text{empty}))) =$   
 $((\text{power } ((w \wedge \text{empty}); f \wedge \text{more}) (\text{Suc } k); (w \wedge \text{empty})))$   
 by (simp add: ChopAssoc)  
 show ?thesis  
 by (metis 1 3 4 5 7 int-eq)  
 qed  
 qed

**lemma** PiChopstarhelp2:

$\vdash (w \wedge \text{empty}); (f; (w \wedge \text{empty}))^* = ((w \wedge \text{empty}); f)^*; (w \wedge \text{empty})$

**proof** –

have 1:  $\vdash (w \wedge \text{empty}); (f; (w \wedge \text{empty}))^* = (w \wedge \text{empty}); (\exists k. \text{power } (f; (w \wedge \text{empty})) \wedge \text{more}) k)$

by (simp add: chopstar-d-def powerstar-d-def)

have 2:  $\vdash (w \wedge \text{empty}); (\exists k. \text{power } (f; (w \wedge \text{empty})) \wedge \text{more}) k =$   
 $(\exists k. (w \wedge \text{empty}); (\text{power } (f; (w \wedge \text{empty})) \wedge \text{more}) k))$

using ChopExist by fastforce

have 3:  $\vdash (\exists k. (w \wedge \text{empty}); (\text{power } (f; (w \wedge \text{empty})) \wedge \text{more}) k) =$   
 $(\exists k. (\text{power } ((w \wedge \text{empty}); f \wedge \text{more}) k; (w \wedge \text{empty})))$

by (simp add: ExEqvRule PiChopstarhelp2a)

have 4:  $\vdash (\exists k. (\text{power } ((w \wedge \text{empty}); f \wedge \text{more}) k; (w \wedge \text{empty}))) =$   
 $(\exists k. (\text{power } ((w \wedge \text{empty}); f \wedge \text{more}) k)); (w \wedge \text{empty})$

using ExistChop by fastforce

have 5:  $\vdash (\exists k. (\text{power } ((w \wedge \text{empty}); f \wedge \text{more}) k)); (w \wedge \text{empty}) =$   
 $((w \wedge \text{empty}); f)^*; (w \wedge \text{empty})$

by (simp add: chopstar-d-def powerstar-d-def)

show ?thesis

using 1 2 3 4 5 by fastforce

qed

**lemma** PiPowerSuca:

$\vdash (\text{init } w) \amalg (\text{power } f (\text{Suc } k)) = ((\text{init } w) \amalg f); ((\text{init } w) \wedge (\text{init } w) \amalg (\text{power } f k))$

**by** (*simp add: PiChopDist*)

**lemma** *PiPowerSuchb*:

$\vdash (init\ w) \Pi (power\ f\ (Suc\ k)) = ((init\ w) \Pi f \wedge fin\ w);((init\ w) \wedge (init\ w) \Pi (power\ f\ k))$

**by** (*metis (no-types, lifting) AndFinChopEqvStateAndChop ChopAssoc ChopImpDiamond FinChopEqvDiamond*

*FinEqvTrueChopAndEmpty InitAndEmptyEqvAndEmpty PiPowerSuca Prop10 StateAndEmptyChop inteq-reflection*)

**lemma** *PiPowerSucc*:

$\vdash (init\ w) \Pi (power\ f\ (Suc\ k)) =$   
 $(power\ ((init\ w) \Pi f \wedge fin\ w)\ (Suc\ k));((init\ w) \wedge (init\ w) \Pi empty)$

**proof** (*induction k*)

**case** 0

**then show** ?case

**by** (*metis (no-types, hide-lams) ChopAssoc EmptyChop PiPowerSuchb inteq-reflection pow-0 pow-Suc*)

**next**

**case** (*Suc k*)

**then show** ?case

**by** (*metis (no-types, hide-lams) AndFinEqvChopAndEmpty ChopAssocB InitAndEmptyEqvAndEmpty PiChopDist StateAndEmptyChop TrueW int-simps(1) inteq-reflection lift-and-com pow-Suc*)

**qed**

**lemma** *PiPowerSuccd*:

$\vdash (init\ w) \Pi (power\ f\ (Suc\ k)) = (power\ ((init\ w) \Pi f \wedge fin\ w)\ (Suc\ k));((init\ w) \Pi empty)$

**proof** (*induction k*)

**case** 0

**then show** ?case

**by** (*metis (no-types, lifting) AndFinChopEqvStateAndChop AndFinEqvChopAndEmpty InitAndEmptyEqvAndEmpty PiPowerSuca int-eq pow-0 pow-Suc*)

**next**

**case** (*Suc k*)

**then show** ?case

**by** (*metis (no-types, lifting) ChopAssoc PiPowerSucc inteq-reflection pow-Suc*)

**qed**

**lemma** *PiChopstar*:

$\vdash (init\ w) \Pi (f^*) = (init\ (\neg w)) \mathcal{U} ((init\ w) \wedge (((init\ w) \Pi f) \wedge fin\ w)^*; wnext(\Box (init\ (\neg w))))$

**proof** –

**have** 1:  $\vdash (init\ w) \Pi (f^*) = (init\ w) \Pi (\exists k. power\ f\ k)$

**by** (*metis ChopstarEqvPowerstar PiEqvRule powerstar-d-def*)

**have** 2:  $\vdash (init\ w) \Pi (\exists k. power\ f\ k) = (\exists k. (init\ w) \Pi (power\ f\ k))$

**by** (*simp add: Valid-def pi-d-def*)

**have** 3:  $\vdash (\exists k. (init\ w) \Pi (power\ f\ k)) =$   
 $( (init\ w) \Pi empty \vee (\exists k. (init\ w) \Pi (power\ f\ (Suc\ k))))$

**using** *PiPowerExpand* **by** *auto*

**have** 4:  $\vdash (\exists k. (init\ w) \Pi (power\ f\ (Suc\ k))) =$   
 $(\exists k. (power\ ((init\ w) \Pi f \wedge fin\ w)\ (Suc\ k));((init\ w) \Pi empty))$

**by** (*meson ExEqvRule PiPowerSuccd*)

**have** 5:  $\vdash (init\ w) \Pi\ empty = empty ; ((init\ w) \Pi\ empty)$   
**by** (*simp add: EmptyChop int-iffD1 int-iffD2 int-iffI*)  
**have** 6:  $\vdash (\exists k. (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ (Suc\ k)); ((init\ w) \Pi\ empty)) =$   
 $(\exists k. (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ (Suc\ k))); ((init\ w) \Pi\ empty)$   
**by** (*simp add: Semantics.ExistChop*)  
**have** 7:  $\vdash ( (init\ w) \Pi\ empty \vee (\exists k. (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ (Suc\ k)); ((init\ w) \Pi\ empty))) =$   
 $( empty; ((init\ w) \Pi\ empty) \vee$   
 $(\exists k. (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ (Suc\ k)); ((init\ w) \Pi\ empty)))$   
**using** 5 6 **by** *fastforce*  
**have** 8:  $\vdash ( empty; ((init\ w) \Pi\ empty) \vee$   
 $(\exists k. (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ (Suc\ k)); ((init\ w) \Pi\ empty)) =$   
 $( empty \vee (\exists k. (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ (Suc\ k)))) ; ((init\ w) \Pi\ empty)$   
**by** (*meson OrChopEqv Prop11*)  
**have** 9:  $\vdash empty = (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ 0)$   
**by** *simp*  
**have** 10:  $\vdash ( empty \vee (\exists k. (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ (Suc\ k)))) =$   
 $( (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ 0) \vee (\exists k. (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ (Suc\ k))))$   
**by** *simp*  
**have** 11:  $\vdash ( (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ 0) \vee (\exists k. (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ (Suc\ k)))) =$   
 $(\exists k. (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ k))$   
**using** *exists-expand[of w f]* **by** *fastforce*  
**have** 12:  $\vdash ( (init\ w) \Pi\ empty \vee$   
 $(\exists k. (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ (Suc\ k)); ((init\ w) \Pi\ empty))) =$   
 $(\exists k. (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ k)); ((init\ w) \Pi\ empty)$   
**by** (*metis 11 7 8 9 inteq-reflection*)  
**have** 13:  $\vdash (\exists k. (power\ ((init\ w) \Pi\ f \wedge fin\ w)\ k)); ((init\ w) \Pi\ empty) =$   
 $((init\ w) \Pi\ f \wedge fin\ w)^* ; ((init\ w) \Pi\ empty)$   
**by** (*metis ChopstarEqvPowerstar LeftChopEqvChop inteq-reflection powerstar-d-def*)  
**have** 14:  $\vdash ((init\ w) \Pi\ f \wedge fin\ w)^* ; ((init\ w) \Pi\ empty) =$   
 $( ((init\ w) \Pi\ empty) \vee$   
 $((init\ w) \Pi\ f \wedge fin\ w); ((init\ w) \Pi\ f \wedge fin\ w)^*); ((init\ w) \Pi\ empty))$   
**using** *CSChopEqvOrChopPlusChop* **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:  $\vdash (((init\ w) \Pi\ f \wedge fin\ w); ((init\ w) \Pi\ f \wedge fin\ w)^*); ((init\ w) \Pi\ empty) =$   
 $((init\ w) \Pi\ f \wedge fin\ w); ((init\ w) \Pi\ f \wedge fin\ w)^* ; wnext\ (\Box (init\ (\neg w)))$   
**by** (*metis (no-types, lifting) AndFinEqvChopAndEmpty ChopAssoc InitAndEmptyEqvAndEmpty*  
*PiChopstarhelp2 StateAndEmptyChop StateAndPiEmpty inteq-reflection*)  
**have** 17:  $\vdash ((init\ w) \Pi\ f \wedge fin\ w) = ((init\ w) \Pi\ f); ((init\ w) \wedge empty)$   
**by** (*metis AndFinEqvChopAndEmpty InitAndEmptyEqvAndEmpty inteq-reflection*)  
**have** 18:  $\vdash ((init\ w) \Pi\ f) = wprev(\Box (init\ (\neg w))); ((init\ w) \wedge (init\ w) \Pi\ f)$   
**by** (*simp add: WPrevPi*)  
**have** 19:  $\vdash wprev(\Box (init\ (\neg w))); ((init\ w) \wedge (init\ w) \Pi\ f) =$   
 $wprev(\Box (init\ (\neg w))); (((init\ w) \wedge empty) ; ((init\ w) \Pi\ f))$   
**by** (*meson RightChopImpChop StateAndEmptyChop int-iffD1 int-iffD2 int-iffI*)  
**have** 20:  $\vdash wprev(\Box (init\ (\neg w))); (((init\ w) \wedge empty) ; ((init\ w) \Pi\ f)) =$   
 $(init\ (\neg w)) \mathcal{U} ( (init\ w) \wedge ((init\ w) \Pi\ f))$   
**using** 18 19 *StatePiUntil* **by** *fastforce*  
**have** 21:  $\vdash (((init\ w) \Pi\ f \wedge fin\ w); ((init\ w) \Pi\ f \wedge fin\ w)^*); wnext\ (\Box (init\ (\neg w))) =$   
 $(init\ (\neg w)) \mathcal{U} ($

```

      (init w) ∧
      (((init w) Π f) ∧ fin w);((init w) Π f ∧ fin w)*;wnext (□ (init (¬ w)))
    )
  by (metis (no-types, lifting) 17 18 AndFinChopEqvStateAndChop ChopAssocB
      StateUntilEqvWPrevChop int-eq)
  have 22: ⊢ ((init (¬ w)) U ( (init w) ∧ wnext (□ (init (¬ w))))) ∨
    (init (¬ w)) U (
      (init w) ∧
      (((init w) Π f) ∧ fin w);((init w) Π f ∧ fin w)*;wnext (□ (init (¬ w)))
    ) =
    (init (¬ w)) U (
      ((init w) ∧ wnext (□ (init (¬ w))))) ∨
      ( (init w) ∧ (((init w) Π f) ∧ fin w);((init w) Π f ∧ fin w)*;wnext (□ (init (¬ w)))))
    )
  using UntilOrDist by fastforce
  have 23: ⊢ (
    ((init w) ∧ wnext (□ (init (¬ w))))) ∨
    ( (init w) ∧ (((init w) Π f) ∧ fin w);((init w) Π f ∧ fin w)*;wnext (□ (init (¬ w)))))
  ) =
    ( (init w) ∧ (((init w) Π f) ∧ fin w)*;wnext(□ (init (¬ w)))))
  by (metis (no-types, lifting) CSChopEqvOrChopPlusChop ChopOrEqv StateAndEmptyChop
      inteq-reflection)
  have 24: ⊢ (init w) Π (f*) = ( (init w) Π empty ∨ (∃ k. (init w) Π (power f (Suc k))))
  using 1 2 3 by fastforce
  have 25: ⊢ ( (init w) Π empty ∨ (∃ k. (init w) Π (power f (Suc k)))) =
    ( (init w) Π empty ∨ (∃ k. (power ((init w) Π f ∧ fin w) (Suc k));((init w) Π empty)))
  using 4 by fastforce
  have 26: ⊢ ( (init w) Π empty ∨
    (∃ k. (power ((init w) Π f ∧ fin w) (Suc k));((init w) Π empty))) =
    ((init w) Π f ∧ fin w)* ; ((init w) Π empty)
  using 12 13 by fastforce
  have 27: ⊢ ((init w) Π f ∧ fin w)* ; ((init w) Π empty) =
    (init (¬ w)) U ((init w) ∧ (((init w) Π f) ∧ fin w)*;wnext(□ (init (¬ w))))
  by (metis 14 15 16 21 22 23 inteq-reflection)
  from 24 25 26 27 show ?thesis by fastforce
qed

```

end

## 16 Interval Temporal Algebra

theory *ITA*

imports *Fuse Semantics TimeReversal*

begin

## 16.1 Definition of Set of intervals and Operations on them

**type-synonym**  $'a \text{ intervals} = 'a \text{ interval set}$

**definition**  $\text{lan} :: ('a :: \text{world}) \text{ formula} \Rightarrow 'a \text{ intervals}$

**where**  $\text{lan } f = \{ \sigma \cdot (\sigma \models f) \}$

**definition**  $\text{fusion} :: 'a \text{ intervals} \Rightarrow 'a \text{ intervals} \Rightarrow 'a \text{ intervals} \text{ (infixl } \cdot 70)$

**where**  $X \cdot Y = \{ \text{fuse } \sigma 1 \sigma 2 \mid \sigma 1 \sigma 2. \sigma 1 \in X \wedge \sigma 2 \in Y \wedge \text{ilast } \sigma 1 = \text{ifirst } \sigma 2 \}$

**definition**  $\text{reverse} :: 'a \text{ intervals} \Rightarrow 'a \text{ intervals} \text{ ((SRev -) [85] 85)}$

**where**  $(\text{SRev } X) = \{ \text{irev } \sigma \mid \sigma. \sigma \in X \}$

**definition**  $\text{empty} :: 'a \text{ intervals} \text{ (SEmpty)}$

**where**

$\text{SEmpty} \equiv \text{range } \text{INil}$

**definition**  $\text{smore} :: 'a \text{ intervals} \text{ (SMore)}$

**where**

$\text{SMore} \equiv - \text{SEmpty}$

**definition**  $\text{sskip} :: 'a \text{ intervals} \text{ (SSkip)}$

**where**

$\text{SSkip} \equiv -(\text{SEmpty} \cup (\text{SMore} \cdot \text{SMore}))$

**definition**  $\text{sfalse} :: 'a \text{ intervals} \text{ (SFalse)}$

**where**

$\text{SFalse} \equiv \{ \}$

**definition**  $\text{strue} :: 'a \text{ intervals} \text{ (STrue)}$

**where**

$\text{STrue} \equiv -\{ \}$

**definition**  $\text{sinit} :: 'a \text{ intervals} \Rightarrow 'a \text{ intervals} \text{ ((SInit -) [85] 85)}$

**where**

$\text{SInit } X \equiv (X \cap \text{SEmpty}) \cdot \text{STrue}$

**definition**  $\text{sfin} :: 'a \text{ intervals} \Rightarrow 'a \text{ intervals} \text{ ((SFin -) [85] 85)}$

**where**

$\text{SFin } X \equiv \text{STrue} \cdot (X \cap \text{SEmpty})$

**definition**  $\text{ssometime} :: 'a \text{ intervals} \Rightarrow 'a \text{ intervals} \text{ ((SSometime -) [85] 85)}$

**where**

$\text{SSometime } X \equiv \text{STrue} \cdot X$

**definition**  $\text{salways} :: 'a \text{ intervals} \Rightarrow 'a \text{ intervals} \text{ ((SAlways -) [85] 85)}$

**where**

$\text{SAlways } X \equiv -(\text{SSometime } (-X))$

**definition**  $sdi :: 'a \text{ intervals} \Rightarrow 'a \text{ intervals} ((SDi \ -) [85] \ 85)$

**where**

$$SDi \ X \equiv X \cdot STrue$$

**definition**  $sbi :: 'a \text{ intervals} \Rightarrow 'a \text{ intervals} ((SBi \ -) [85] \ 85)$

**where**

$$SBi \ X \equiv -(SDi \ (-X))$$

**definition**  $sda :: 'a \text{ intervals} \Rightarrow 'a \text{ intervals} ((SDa \ -) [85] \ 85)$

**where**

$$SDa \ X \equiv STrue \cdot X \cdot STrue$$

**definition**  $sba :: 'a \text{ intervals} \Rightarrow 'a \text{ intervals} ((SBa \ -) [85] \ 85)$

**where**

$$SBa \ X \equiv -(SDa \ (-X))$$

**definition**  $snext :: 'a \text{ intervals} \Rightarrow 'a \text{ intervals} ((SNext \ -) [85] \ 85)$

**where**

$$SNext \ X \equiv SSkip \cdot X$$

**definition**  $swnext :: 'a \text{ intervals} \Rightarrow 'a \text{ intervals} ((SWnext \ -) [85] \ 85)$

**where**

$$SWnext \ X \equiv -(SSkip \cdot (-X))$$

**definition**  $sprev :: 'a \text{ intervals} \Rightarrow 'a \text{ intervals} ((SPrev \ -) [85] \ 85)$

**where**

$$SPrev \ X \equiv X \cdot SSkip$$

**definition**  $swprev :: 'a \text{ intervals} \Rightarrow 'a \text{ intervals} ((SWprev \ -) [85] \ 85)$

**where**

$$SWprev \ X \equiv -((-X) \cdot SSkip)$$

**primrec**  $spower :: 'a \text{ intervals} \Rightarrow \text{nat} \Rightarrow 'a \text{ intervals} ((SPower \ - \ -) [88,88] \ 87)$

**where**

$$\begin{aligned} \text{pwr-0} & : SPower \ X \ 0 = SEmpty \\ | \text{pwr-Suc} & : SPower \ X \ (Suc \ n) = ((X \cap SMore) \cdot (SPower \ X \ n)) \end{aligned}$$

**definition**  $sstar :: 'a \text{ intervals} \Rightarrow 'a \text{ intervals} ((SStar \ -) [85] \ 85)$

**where**

$$SStar \ X \equiv (\bigcup n. SPower \ X \ n)$$

## 16.2 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** *empty-elim* :

$$\sigma \in SEmpty \longleftrightarrow \text{ilen } \sigma = 0$$

by (*simp add: image-iff INil-ilen empty-def*)

**lemma** *smore-elim* :

$$\sigma \in SMore \longleftrightarrow \text{ilen } \sigma > 0$$

by (*simp add: empty-elim smore-def*)

**lemma** *fusion-iff*:

$$\sigma \in X \cdot Y \longleftrightarrow (\exists \sigma 1 \sigma 2. \sigma = \text{fuse } \sigma 1 \sigma 2 \wedge \sigma 1 \in X \wedge \sigma 2 \in Y \wedge \text{ilast } \sigma 1 = \text{ifirst } \sigma 2)$$

by (*unfold fusion-def*) *auto*

**lemma** *fusion-iff-1*:

$$\sigma \in X \cdot Y \longleftrightarrow (\exists i \leq \text{ilen } \sigma. (\text{prefix } i \sigma) \in X \wedge (\text{suffix } i \sigma) \in Y)$$

using *fusion-iff[of \sigma X Y]*

by (*metis ilast-ifirst fuse-ilen fuse-prefix-suffix prefix-fuse suffix-fuse le-add1*)

**lemma** *smore-fusion-smore* :

$$\sigma \in (SMore \cdot SMore) \longleftrightarrow \text{ilen } \sigma > 1$$

using *fusion-iff-1*

by (*metis prefix-ilen-good suffix-ilen-good less-one*

*not-less not-less-iff-gr-or-eq smore-elim zero-less-diff*)

**lemma** *sskip-elim* :

$$\sigma \in SSkip \longleftrightarrow \text{ilen } \sigma = 1$$

using *sskip-def smore-fusion-smore*

by (*metis One-nat-def Suc-lessI Un-iff less-numeral-extra(4) empty-elim smore-def smore-elim snot-elim zero-neq-one*)

**lemma** *spower-elim-zero* :

$$\sigma \in SPower X 0 \longleftrightarrow \sigma \in SEmpty$$

by *simp*

**lemma** *spower-elim-suc* :

$$\sigma \in SPower X (\text{Suc } n) \longleftrightarrow \sigma \in (X \cap SMore) \cdot (SPower X n)$$

by *simp*

**lemma** *spower-elim-suc-1* :

$\sigma \in (X \cap SMore) \cdot (SPower X n) \longleftrightarrow$   
 $(\exists \sigma 1 \sigma 2. \sigma = fuse \sigma 1 \sigma 2 \wedge \sigma 1 \in X \wedge ilen \sigma 1 > 0 \wedge \sigma 2 \in (SPower X n) \wedge$   
 $ilast \sigma 1 = ifirst \sigma 2)$

**by** (*meson IntD1 IntD2 IntI smore-elim fusion-iff*)

**lemma** *sstar-elim* :

$\sigma \in SStar X \longleftrightarrow (\exists n. \sigma \in SPower X n)$

**by** (*simp add: sstar-def*)

**lemma** *sstar-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** *spower-suc* :

$(\exists n. \sigma \in SPower X (Suc n)) \longleftrightarrow$   
 $(\exists n. \sigma \in (X \cap SMore) \cdot (SPower X n))$

**by** *simp*

**lemma** *spower-suc-1* :

$(\exists n. \sigma \in (X \cap SMore) \cdot (SPower X n)) \longleftrightarrow$   
 $\sigma \in (X \cap SMore) \cdot (SStar X)$

**by** (*metis fusion-iff sstar-elim*)

**lemma** *sstar-equiv* :

$\sigma \in SStar X \longleftrightarrow$   
 $(\sigma \in SEmpty \vee \sigma \in (X \cap SMore) \cdot (SStar X))$

**by** (*metis spower.simps(1) spower-elim-suc spower-suc-1 sstar-elim sstar-elim-1*)

**lemma** *spower-skip-elim* :

$(\sigma \in SPower SSkip n) \longleftrightarrow ilen \sigma = n$

**proof** (*induct n arbitrary:  $\sigma$* )

**case** 0

**then show** ?case **by** (*auto simp add: empty-elim*)

**next**

**case** (*Suc n*)

**then show** ?case

**proof** *auto*

**show**  $(\bigwedge \sigma. (\sigma \in SPower SSkip n) = (ilen \sigma = n)) \implies$   
 $\sigma \in SSkip \cap SMore \cdot SPower SSkip n \implies ilen \sigma = Suc n$

**by** (*metis Suc.hyps fuse-ilen plus-1-eq-Suc spower-elim-suc-1 skip-elim*)

**show**  $(\bigwedge \sigma. (\sigma \in SPower SSkip n) = (ilen \sigma = n)) \implies$   
 $ilen \sigma = Suc n \implies \sigma \in SSkip \cap SMore \cdot SPower SSkip n$

**by** (*metis Suc.hyps Compl-Un Int-commute add-diff-cancel-left' fusion-iff-1 inf-sup-aci(4)*  
*prefix-ilen-good suffix-ilen-good le-add1 plus-1-eq-Suc smore-def*  
*skip-def skip-elim*)

**qed**

**qed**



**lemma** *srev-elim*:  
 $\sigma \in (SRev\ X) \longleftrightarrow irev\ \sigma \in X$   
**using** *irev-swap* **by** (*auto simp add: reverse-def*)

## 16.3 Algebraic Laws

### 16.3.1 Commutative Additive Monoid

**lemma** *UnionCommute*:  
 $(X::'a\ intervals) \cup Y = Y \cup X$   
**by** (*simp add: Un-commute*)

**lemma** *UnionSFalse*:  
 $X \cup SFalse = X$   
**by** (*simp add: sfalse-def*)

**lemma** *UnionAssoc*:  
 $(X::'a\ intervals) \cup (Y \cup Z) = (X \cup Y) \cup Z$   
**by** (*simp add: sup-assoc*)

### 16.3.2 Boolean algebra

**lemma** *Huntington*:  
 $(X::'a\ intervals) = -(-X \cup -Y) \cup -(-X \cup Y)$   
**by** *auto*

**lemma** *Morgan*:  
 $(X::'a\ intervals) \cap Y = -(-X \cup -Y)$   
**by** *auto*

— identities

**lemma** *STrueTop*:  
 $STrue = X \cup -X$   
**by** (*simp add: strue-def*)

**lemma** *SFalseBottom*:  
 $SFalse = X \cap -X$   
**by** (*simp add: sfalse-def*)

### 16.3.3 multiplicative monoid

**lemma** *FusionSEmptyL* :  
 $SEmpty \cdot X = X$   
**using** *fusion-iff-1 set-eqI*[*of SEmpty.X X*]  
**by** (*metis ilen-gr-zero prefix-ilen-good suffix-zero empty-elim*)

**lemma** *FusionSEmptyR* :

$X \cdot SEmpty = X$   
**using** *fusion-iff-1 set-eqI*[of  $X \cdot SEmpty$   $X$ ]  
**proof** *auto*  
**show**  $\bigwedge x. (\bigwedge \sigma X Y. (\sigma \in X \cdot Y) = (\exists i \leq \text{ilen } \sigma. \text{prefix } i \sigma \in X \wedge \text{suffix } i \sigma \in Y)) \implies$   
 $((\bigwedge x. (x \in X \cdot SEmpty) = (x \in X)) \implies X \cdot SEmpty = X) \implies x \in X \cdot SEmpty \implies x \in X$   
**by** (*metis diff-diff-cancel diff-zero fusion-iff-1 prefix-ilen*  
*suffix-ilen-good empty-elim*)  
**show**  $\bigwedge x. (\bigwedge \sigma X Y. (\sigma \in X \cdot Y) = (\exists i \leq \text{ilen } \sigma. \text{prefix } i \sigma \in X \wedge \text{suffix } i \sigma \in Y)) \implies$   
 $((\bigwedge x. (x \in X \cdot SEmpty) = (x \in X)) \implies X \cdot SEmpty = X) \implies x \in X \implies x \in X \cdot SEmpty$   
**using** *empty-elim fusion-iff-1* **by** *fastforce*  
**qed**

**lemma** *FusionAssocA*:

**assumes**  $x \in X \cdot (Y \cdot Z)$

**shows**  $x \in (X \cdot Y) \cdot Z$

**proof** –

**have** 1:  $(\exists \sigma 1 \sigma 2. x = \text{fuse } \sigma 1 \sigma 2 \wedge \sigma 1 \in X \wedge \sigma 2 \in Y \cdot Z \wedge \text{ilast } \sigma 1 = \text{ifirst } \sigma 2)$   
**using** *assms fusion-iff*[of  $x X Y \cdot Z$ ] **by** *auto*  
**obtain**  $\sigma 1 \sigma 2$  **where** 2:  $x = \text{fuse } \sigma 1 \sigma 2 \wedge \sigma 1 \in X \wedge \sigma 2 \in Y \cdot Z \wedge \text{ilast } \sigma 1 = \text{ifirst } \sigma 2$   
**using** 1 **by** *auto*  
**have** 3:  $(\exists \sigma 3 \sigma 4. \sigma 2 = \text{fuse } \sigma 3 \sigma 4 \wedge \sigma 3 \in Y \wedge \sigma 4 \in Z \wedge \text{ilast } \sigma 3 = \text{ifirst } \sigma 4)$   
**using** 2 *fusion-iff*[of  $\sigma 2 Y Z$ ] **by** *auto*  
**obtain**  $\sigma 3 \sigma 4$  **where** 4:  $\sigma 2 = \text{fuse } \sigma 3 \sigma 4 \wedge \sigma 3 \in Y \wedge \sigma 4 \in Z \wedge \text{ilast } \sigma 3 = \text{ifirst } \sigma 4$   
**using** 3 **by** *auto*  
**have** 5:  $x = \text{fuse } \sigma 1 (\text{fuse } \sigma 3 \sigma 4)$   
**using** 2 4 **by** *auto*  
**have** 6:  $x = \text{fuse } (\text{fuse } \sigma 1 \sigma 3) \sigma 4$   
**using** 5 2 4 *FusionAssoc ifirst-fuse* **by** *fastforce*  
**show** *?thesis*  
**by** (*metis 2 4 6 fusion-iff ifirst-fuse ilast-fuse*)  
**qed**

**lemma** *FusionAssocB*:

**assumes**  $x \in (X \cdot Y) \cdot Z$

**shows**  $x \in X \cdot (Y \cdot Z)$

**proof** –

**have** 1:  $(\exists \sigma 1 \sigma 2. x = \text{fuse } \sigma 1 \sigma 2 \wedge \sigma 1 \in X \cdot Y \wedge \sigma 2 \in Z \wedge \text{ilast } \sigma 1 = \text{ifirst } \sigma 2)$   
**using** *assms fusion-iff*[of  $x X \cdot Y Z$ ] **by** *auto*  
**obtain**  $\sigma 1 \sigma 2$  **where** 2:  $x = \text{fuse } \sigma 1 \sigma 2 \wedge \sigma 1 \in X \cdot Y \wedge \sigma 2 \in Z \wedge \text{ilast } \sigma 1 = \text{ifirst } \sigma 2$   
**using** 1 **by** *auto*  
**have** 3:  $(\exists \sigma 3 \sigma 4. \sigma 1 = \text{fuse } \sigma 3 \sigma 4 \wedge \sigma 3 \in X \wedge \sigma 4 \in Y \wedge \text{ilast } \sigma 3 = \text{ifirst } \sigma 4)$   
**using** 2 *fusion-iff*[of  $\sigma 1 X Y$ ] **by** *auto*  
**obtain**  $\sigma 3 \sigma 4$  **where** 4:  $\sigma 1 = \text{fuse } \sigma 3 \sigma 4 \wedge \sigma 3 \in X \wedge \sigma 4 \in Y \wedge \text{ilast } \sigma 3 = \text{ifirst } \sigma 4$   
**using** 3 **by** *auto*  
**have** 5:  $x = \text{fuse } (\text{fuse } \sigma 3 \sigma 4) \sigma 2$   
**using** 2 4 **by** *auto*  
**have** 6:  $x = \text{fuse } \sigma 3 (\text{fuse } \sigma 4 \sigma 2)$   
**using** 2 4 *FusionAssoc ilast-fuse* **by** *force*  
**show** *?thesis*  
**by** (*metis 2 4 6 fusion-iff ifirst-fuse ilast-fuse*)

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*

— left and right distributivity

**lemma** *FusionUnionDistL*:

$$(X \cup Y) \cdot Z = (X \cdot Z) \cup (Y \cdot Z)$$

**using** *fusion-iff*[*of* -  $X \cup Y \ Z$ ] *fusion-iff*[*of* -  $X \ Z$ ] *fusion-iff*[*of* -  $Y \ Z$ ]

$$\text{set-eqI}[\text{of } (X \cup Y) \cdot Z \ (X \cdot Z) \cup (Y \cdot Z)]$$

**by** *auto*

**lemma** *FusionUnionDistR*:

$$X \cdot (Y \cup Z) = (X \cdot Y) \cup (X \cdot Z)$$

**using** *fusion-iff set-eqI*[*of*  $X \cdot (Y \cup Z) \ (X \cdot Y) \cup (X \cdot Z)$ ]

*fusion-iff*[*of* -  $X \ Y \cup Z$ ] *fusion-iff*[*of* -  $X \ Y$ ] *fusion-iff*[*of* -  $X \ Z$ ]

**by** *auto*

— left and right annihilation

**lemma** *SFalseFusion*:

$$SFalse \cdot X = SFalse$$

**by** (*simp add: fusion-def sfalse-def*)

**lemma** *FusionSFalse*:

$$X \cdot SFalse = SFalse$$

**by** (*simp add: fusion-def sfalse-def*)

— idempotency

**lemma** *UnionIdem*:

$$(X :: 'a \text{ intervals}) \cup X = X$$

**by** *simp*

### 16.3.4 Subsumption order

**lemma** *Subsumption*:

$$((X :: 'a \text{ intervals}) \subseteq Y) = (X \cup Y = Y)$$

**by** *auto*

### 16.3.5 Helper lemmas

**lemma** *FusionRuleR*:

**assumes**  $X \subseteq Y$

**shows**  $Z \cdot X \subseteq Z \cdot Y$

**using** *assms FusionUnionDistR* **by** (*metis Subsumption*)

**lemma** *FusionRuleL*:

**assumes**  $X \subseteq Y$   
**shows**  $X \cdot Z \subseteq Y \cdot Z$   
**using** *assms* **by** (*metis FusionUnionDistL subset-Un-eq*)

**lemma** *spower-commutes*:

$(X \cap SMore) \cdot (SPower X n) = (SPower X n) \cdot (X \cap SMore)$   
**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*:  $X \cdot (SPower X n \cdot Y) \cup X \cdot Z = X \cdot Z$   
**by** (*metis FusionUnionDistR Suc.hyps assms subset-Un-eq*)  
**have**  $X \cdot SPower X n \cdot Y \cup X \cap SMore \cdot SPower X n \cdot Y = X \cdot (SPower X n \cdot Y)$   
**by** (*metis (no-types) FusionAssoc FusionUnionDistL sup-inf-absorb*)  
**then have**  $SPower X (Suc n) \cdot Y \cup Z = X \cdot Z \cup (Y \cup Z)$   
**using** *f1 assms* **by** *auto*  
**then show** ?*thesis*  
**using** *assms* **by** *auto*  
**qed**  
**qed**

**lemma** *fusion-inductr*:

**assumes**  $Y \cup Z \cdot X \subseteq Z$   
**shows**  $Y \cdot (SPower X n) \subseteq Z$   
**using** *assms*  
**proof** (*induct n*)  
**case** 0  
**then show** ?*case* **by** (*simp add: FusionSEmptyR*)  
**next**  
**case** (*Suc n*)  
**then show** ?*case*  
**proof** –  
**have** *f1*:  $Y \cdot SPower X n \cup Z = Z$   
**using** *Suc.hyps assms* **by** *blast*  
**have**  $Y \cdot (X \cap SMore) \cdot SPower X n = Y \cdot (SPower X n \cdot (X \cap SMore))$

by (metis (no-types) FusionAssoc spower-commutes)  
 then have  $Y \cdot (X \cap SMore) \cdot SPower\ X\ n \subseteq Z$   
 using f1 by (metis (no-types) FusionAssoc FusionUnionDistL FusionUnionDistR Un-subset-iff  
 assms sup-inf-absorb)  
 then show ?thesis  
 by (simp add: FusionAssoc)  
 qed  
 qed

**lemma sstar-contlA:**  
 assumes  $x \in Y \cdot (SStar\ X)$   
 shows  $x \in (\bigcup n. Y \cdot (SPower\ X\ n))$   
**proof** –  
 have 1:  $(\exists \sigma 1\ \sigma 2. x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in Y \wedge \sigma 2 \in (SStar\ X) \wedge ilast\ \sigma 1 = ifirst\ \sigma 2)$   
 using assms by (simp add: fusion-iff)  
 obtain  $\sigma 1\ \sigma 2$  where 2:  $x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in Y \wedge \sigma 2 \in (SStar\ X) \wedge ilast\ \sigma 1 = ifirst\ \sigma 2$   
 using 1 by auto  
 have 3:  $(\exists n. \sigma 2 \in SPower\ X\ n)$   
 using 2 sstar-elim by blast  
 obtain  $n$  where 4:  $\sigma 2 \in SPower\ X\ n$   
 using 3 by auto  
 have 5:  $(\exists n. x \in Y \cdot SPower\ X\ n)$   
 using 2 4 fusion-iff by blast  
 from 5 show ?thesis by blast  
 qed

**lemma sstar-contlB:**  
 assumes  $x \in (\bigcup n. Y \cdot (SPower\ X\ n))$   
 shows  $x \in Y \cdot (SStar\ X)$   
**proof** –  
 have 1:  $\exists n. x \in Y \cdot (SPower\ X\ n)$   
 using assms by blast  
 obtain  $n$  where 2:  $x \in Y \cdot (SPower\ X\ n)$   
 using 1 by auto  
 have 3:  $(\exists \sigma 1\ \sigma 2. x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in Y \wedge \sigma 2 \in (SPower\ X\ n) \wedge ilast\ \sigma 1 = ifirst\ \sigma 2)$   
 using 2 by (simp add: fusion-iff)  
 obtain  $\sigma 1\ \sigma 2$  where 4:  $x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in Y \wedge \sigma 2 \in (SPower\ X\ n) \wedge ilast\ \sigma 1 = ifirst\ \sigma 2$   
 using 3 by auto  
 have 5:  $\sigma 2 \in (SStar\ X)$   
 using 4 sstar-elim by auto  
 from 5 4 show ?thesis using fusion-iff by blast  
 qed

**lemma sstar-contl:**  
 $Y \cdot (SStar\ X) = (\bigcup n. Y \cdot (SPower\ X\ n))$   
 using set-eqI[of  $Y \cdot (SStar\ X)$   $(\bigcup n. Y \cdot (SPower\ X\ n))$ ]  
 by (metis sstar-contlA sstar-contlB)

**lemma sstar-contrA:**

**assumes**  $x \in (SStar\ X) \cdot Y$   
**shows**  $x \in (\bigcup n. (SPower\ X\ n) \cdot Y)$   
**proof** –  
**have** 1:  $(\exists \sigma 1\ \sigma 2. x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in (SStar\ X) \wedge \sigma 2 \in Y \wedge ilast\ \sigma 1 = ifirst\ \sigma 2)$   
**using** *assms* **by** (*simp add: fusion-iff*)  
**obtain**  $\sigma 1\ \sigma 2$  **where** 2:  $x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in (SStar\ X) \wedge \sigma 2 \in Y \wedge ilast\ \sigma 1 = ifirst\ \sigma 2$   
**using** 1 **by** *auto*  
**have** 3:  $\exists n. \sigma 1 \in (SPower\ X\ n)$   
**using** 2 *sstar-elim* **by** *blast*  
**obtain**  $n$  **where** 4:  $\sigma 1 \in (SPower\ X\ n)$   
**using** 3 **by** *auto*  
**have** 5:  $(\exists n. x \in (SPower\ X\ n) \cdot Y)$   
**by** (*metis* 2 4 *fusion-iff-1 fuse-ilen prefix-fuse suffix-fuse le-add1*)  
**from** 5 **show** *?thesis* **by** *blast*  
**qed**

**lemma** *sstar-contrB*:

**assumes**  $x \in (\bigcup n. (SPower\ X\ n) \cdot Y)$   
**shows**  $x \in (SStar\ X) \cdot Y$   
**proof** –  
**have** 1:  $\exists n. x \in (SPower\ X\ n) \cdot Y$   
**using** *assms* **by** *blast*  
**obtain**  $n$  **where** 2:  $x \in (SPower\ X\ n) \cdot Y$   
**using** 1 **by** *auto*  
**have** 3:  $(\exists \sigma 1\ \sigma 2. x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in (SPower\ X\ n) \wedge \sigma 2 \in Y \wedge ilast\ \sigma 1 = ifirst\ \sigma 2)$   
**using** 2 **by** (*simp add: fusion-iff*)  
**obtain**  $\sigma 1\ \sigma 2$  **where** 4:  $x = fuse\ \sigma 1\ \sigma 2 \wedge \sigma 1 \in (SPower\ X\ n) \wedge \sigma 2 \in Y \wedge ilast\ \sigma 1 = ifirst\ \sigma 2$   
**using** 3 **by** *auto*  
**have** 5:  $\sigma 1 \in (SStar\ X)$   
**using** 4 *sstar-elim* **by** *auto*  
**from** 5 4 **show** *?thesis* **using** *fusion-iff* **by** *blast*  
**qed**

**lemma** *sstar-contr*:

$(SStar\ X) \cdot Y = (\bigcup n. (SPower\ X\ n) \cdot Y)$   
**using** *set-eqI* [*of*  $(SStar\ X) \cdot Y$   $(\bigcup n. (SPower\ X\ n) \cdot Y)$ ]  
**by** (*metis* *sstar-contrA sstar-contrB*)

### 16.3.6 Kleene Algebra

— left unfold

**lemma** *UnfoldL*:

$SEmpty \cup X \cdot (SStar\ X) = (SStar\ X)$   
**proof** –  
**have** 1:  $(SStar\ X) = SEmpty \cup (X \cap SMore) \cdot (SStar\ X)$   
**by** (*meson Un-iff set-eqI sstar-eqv*)  
**have** 2:  $(X \cap SMore) \cdot (SStar\ X) \subseteq X \cdot (SStar\ X)$   
**by** (*simp add: FusionRuleL*)  
**have** 3:  $(SStar\ X) \subseteq SEmpty \cup X \cdot (SStar\ X)$

using 1 2 by blast  
 have 4:  $SEmpty \subseteq (SStar\ X)$   
 using 1 by auto  
 have 5:  $X \subseteq SEmpty \cup (X \cap SMore)$   
 by (simp add: Un-Int-distrib smore-def)  
 have 6:  $X \cdot (SStar\ X) \subseteq (SStar\ X) \cup (X \cap SMore) \cdot (SStar\ X)$   
 using 5 by (metis FusionRuleL FusionUnionDistL FusionSEmptyL)  
 have 7:  $(SStar\ X) \subseteq SEmpty \cup (X \cap SMore) \cdot (SStar\ X)$   
 using 1 by auto  
 have 8:  $X \cdot (SStar\ X) \subseteq SEmpty \cup (X \cap SMore) \cdot (SStar\ X)$   
 using 6 7 by blast  
 hence 9:  $X \cdot (SStar\ X) \subseteq (SStar\ X)$   
 using 1 by auto  
 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**  $(SStar\ X) \cdot Y \subseteq Z$

**by** (metis UN-least assms fusion-inductl sstar-contr)

— Right induction

**lemma** *SStarInductR*:

**assumes**  $Y \cup Z \cdot X \subseteq Z$

**shows**  $Y \cdot (SStar\ X) \subseteq Z$

**using** sstar-contl assms fusion-inductr **by** blast

### 16.3.7 ITL specific Laws

**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** set-eqI[of  $((((SPower\ SSkip\ n) \cap X) \cdot V) \cap (((SPower\ SSkip\ n) \cap Y) \cdot W))$

$((((SPower\ SSkip\ n) \cap X \cap Y) \cdot (V \cap W)) ]$

**by** (simp add: fusion-iff-1 spower-skip-elim)

(metis min.absorb1)

**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** set-eqI[of  $((V \cdot ((SPower\ SSkip\ n) \cap X)) \cap (W \cdot ((SPower\ SSkip\ n) \cap Y)))$

$((V \cap W) \cdot ((SPower\ SSkip\ n) \cap X \cap Y)) ]$

**by** (simp add: fusion-iff-1 spower-skip-elim)

(metis diff-diff-cancel)

**lemma** *SSkipFusionImpSMore*:

$SSkip \cdot STrue \subseteq SMore$   
**using** *subsetI*[of  $SSkip \cdot STrue$   $SMore$ ]  
**by** (*auto simp add: fusion-iff-1 sskip-elim smore-elim strue-elim*)

**lemma** *SStarSkip*:  
 $(SStar\ SSkip) = STrue$   
**using** *set-eqI*[of  $(SStar\ SSkip)$   $STrue$ ]  
**by** (*simp add: strue-def spower-sskip-elim sstar-elim*)

## 16.4 Derived Laws

### 16.4.1 Helper Lemmas

**lemma** *B01*:  
**assumes**  $(X:: 'a\ intervals) \subseteq Y$   
**shows**  $\neg Y \subseteq \neg X$   
**using** *assms* **by** *auto*

**lemma** *B04*:  
 $((X:: 'a\ intervals) = Y) \longleftrightarrow (X \subseteq Y) \wedge (Y \subseteq X)$   
**by** *auto*

**lemma** *B09*:  
**assumes**  $\neg X \cup Y = STrue$   
**shows**  $(X:: 'a\ intervals) \subseteq Y$   
**using** *assms* **using** *strue-def* **by** *auto*

**lemma** *B20*:  
 $(X:: 'a\ intervals) \subseteq Y \cup Z \longleftrightarrow X \cap \neg Y \subseteq Z$   
**by** *auto*

**lemma** *B28*:  
 $((X:: 'a\ intervals) \cap Y) \cup (X \cap Z) = X \cap (Y \cup Z)$   
**by** *auto*

**lemma** *CH01*:  
 $STrue \cdot STrue = STrue$   
**by** (*metis FusionSEmptyR FusionUnionDistR Int-commute SStarSkip STrueTop UnfoldL 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 inf-commute smore-def sskip-def*)

**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 inf-commute smore-def sskip-def*)

**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*:

$(SStar (X \cap SEmpty)) = SEmpty$   
**by** (*metis* *FusionSEmptyL inf.right-idem inf-le2 UnfoldL*  
*SStarInductR sup.orderE sup-inf-absorb*)

**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*:

$(SInit (-X)) \subseteq -(SInit X)$   
**by** (*metis* *Compl-disjoint ST24 compl-bot-eq disjoint-eq-subset-Compl double-complement*  
*inf.coboundedI2 inf.orderE sfalse-def SFalseFusion sinit-def strue-def*)

**lemma** *ST37*:

$-(SInit\ X) \subseteq (SInit\ (-X))$   
**using** *B09 ST26* **by** *auto*

**lemma** *ST38*:  
 $-(SInit\ X) = (SInit\ (-X))$   
**using** *ST37 ST36* **by** *auto*

**lemma** *ST47*:  
 $X \cup Y \cdot X = (SEmpty \cup Y) \cdot X$   
**by** (*simp add: FusionUnionDistL FusionSEmptyL*)

**lemma** *SStar01*:  
**assumes**  $X \cdot (SStar\ Y) \cup SEmpty \subseteq (SStar\ Y)$   
**shows**  $(SStar\ X) \subseteq (SStar\ Y)$   
**using** *assms*  
**by** (*metis Un-commute FusionSEmptyR SStarInductL*)

**lemma** *SStar03*:  
 $(SStar\ X) \cdot (SStar\ X) \subseteq (SStar\ X)$   
**by** (*metis SStarInductL Un-absorb UnfoldL sup.absorb-iff1 sup.right-idem*)

**lemma** *SStar05*:  
**assumes**  $((SStar\ X) \cdot (SStar\ X)) \cup SEmpty \subseteq (SStar\ X)$   
**shows**  $(SStar\ (SStar\ X)) \subseteq (SStar\ X)$   
**using** *assms*  
**by** (*simp add: SStar01*)

**lemma** *SStar12*:  
 $(SEmpty \cup (X \cdot (SStar\ X))) \subseteq (SStar\ X)$   
**using** *UnfoldL* **by** *blast*

**lemma** *SStar06*:  
 $((SStar\ X) \cdot (SStar\ X)) \cup SEmpty \subseteq (SStar\ X)$   
**using** *SStar03 SStar12* **by** *force*

**lemma** *SStar07*:  
 $(SStar\ X) \subseteq (SStar\ (SStar\ X))$   
**by** (*metis FusionUnionDistR FusionSEmptyR Subsumption Un-commute UnfoldL ST47 sup.right-idem*)

**lemma** *SStar08*:  
 $(SStar\ X) = (SStar\ (SStar\ X))$   
**by** (*meson B04 SStar05 SStar06 SStar07*)

**lemma** *SStar15*:  
 $SEmpty \subseteq (SStar\ SSkip)$   
**by** (*simp add: SStarSkip strue-def*)

**lemma** *SStar16*:  
 $SSkip \subseteq (SStar\ SSkip)$   
**by** (*simp add: SStarSkip strue-def*)

**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 SStar08 SStarSkip*)

**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** –

$$\text{have } (SEmpty \cup SStar\ (X \cup Y) \cdot Y) \cdot SStar\ X \subseteq SStar\ (X \cup Y)$$

**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** –

$$\text{have } 1: (SStar\ Z) = (SStar\ ((SEmpty \cap Z) \cup (SMore \cap Z)))$$

**by** (*metis* *Int-Un-distrib2* *compl-bot-eq* *inf-top.left-neutral* *smore-def* *strue-def* *STrueTop*)

$$\text{have } 2: (SStar\ ((SEmpty \cap Z) \cup (SMore \cap Z))) =$$

$$(SStar\ (SEmpty \cap Z)) \cdot (SStar\ ((SMore \cap Z) \cdot (SStar\ (SEmpty \cap Z))))$$

**by** (*simp* *add*: *SStar36* *SStar46* *subset-antisym*)

$$\text{have } 3: (SStar\ (SEmpty \cap Z)) \cdot (SStar\ ((SMore \cap Z) \cdot (SStar\ (SEmpty \cap Z)))) =$$

$$(SStar\ (Z \cap SMore))$$

**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*:

$$\text{assumes } SSkip \cdot ((-X) \cup ((SStar\ SSkip) \cdot (X \cap (SSkip \cdot (-X))))) \cup (-X)$$

$$\subseteq (-X) \cup (SStar\ SSkip) \cdot (X \cap (SSkip \cdot (-X)))$$

$$\text{shows } ((SStar\ SSkip) \cdot (-X)) \subseteq ((-X) \cup ((SStar\ SSkip) \cdot (X \cap (SSkip \cdot (-X)))))$$

**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** –

$$\text{have } f1: -X \cup (SEmpty \cup SSkip \cdot SStar\ SSkip) \cdot (X \cap (SSkip \cdot -X)) \subseteq$$

$$-X \cup SStar\ SSkip \cdot (X \cap (SSkip \cdot -X))$$

**by** (*simp* *add*:  $\langle SEmpty \cup SSkip \cdot SStar\ SSkip = SStar\ SSkip \rangle$ )

$$\text{have } f2: SSkip \cdot -X \subseteq -X \cup SStar\ SSkip \cdot (X \cap (SSkip \cdot -X))$$

**by** (*metis* *B20* *Morgan* *ST47* *UnionIdem*  $\langle SEmpty \cup SSkip \cdot SStar\ SSkip = SStar\ SSkip \rangle$  *inf-commute* *inf-idem* *sup.cobounded1*)

$$\text{have } SSkip \cdot (SStar\ SSkip \cdot (X \cap (SSkip \cdot -X))) \subseteq -X \cup SStar\ SSkip \cdot (X \cap (SSkip \cdot -X))$$

**using** *f1* **by** (*metis* (*no-types*) *FusionAssoc* *ST47* *Un-subset-iff*)

**then show** *?thesis*

$$\text{using } f2 \text{ by } (simp \text{ add: } \langle \bigwedge Z\ Y. SSkip \cdot (Y \cup Z) = SSkip \cdot Y \cup SSkip \cdot Z \rangle)$$

**qed**

**lemma** *SStar52*:

$(SStar\ X) \subseteq SEmpty \cup (X \cap SMore) \cdot (SStar\ X)$

**by** (*metis B04 SStar47 UnfoldL*)

**lemma** *SStar53*:

$SEmpty \cup (X \cap SMore) \cdot (SStar\ X) \subseteq (SStar\ X)$

**by** (*metis SStar12 SStar47*)

**lemma** *BD45*:

$(SBI\ ((-X) \cup X1)) \cap (X \cdot Y) \subseteq X1 \cdot Y$

**proof** –

**have** 1:  $(SBI\ ((-X) \cup X1)) = -(-((-X) \cup X1) \cdot (Y \cup (-Y)))$

**by** (*metis sbi-def sdi-def STrueTop*)

**have** 2:  $-(-((-X) \cup X1) \cdot (Y \cup (-Y))) \cap (X \cdot Y) \subseteq$   
 $-(-((-X) \cup X1) \cdot (Y)) \cap (X \cdot Y)$

**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:  $(SAlways\ ((-Y) \cup Y1)) = -((X1 \cup (-X1)) \cdot (-((-Y) \cup Y1) \cdot ))$

**by** (*metis salways-def ssometime-def STrueTop*)

**have** 2:  $-((X1 \cup (-X1)) \cdot (-((-Y) \cup Y1) \cdot )) \cap (X1 \cdot Y) \subseteq$   
 $-(X1 \cdot (-((-Y) \cup Y1) \cdot )) \cap (X1 \cdot Y)$

**using** *FusionUnionDistL* **by** *fastforce*

**have** 3:  $-(X1 \cdot (-((-Y) \cup Y1) \cdot )) \cap (X1 \cdot Y) \subseteq X1 \cdot (((-Y) \cup Y1) \cap Y)$

**by** (*metis (no-types, lifting) B20 B04 compl-inf FusionUnionDistR Huntington Morgan Subsumption sup-ge1 UnionCommute*)

**have** 4:  $X1 \cdot (((-Y) \cup Y1) \cap Y) \subseteq (X1 \cdot Y1)$

**by** (*metis B20 double-compl FusionRuleR inf.right-idem inf-le1*)

**from** 1 2 3 4 **show** *?thesis* **by** *blast*

**qed**

## 16.4.2 ITL Axioms derived

**lemma** *SBoxGen*:

**assumes**  $X = STrue$

**shows**  $(SAlways\ X) = STrue$

**using** *assms*

**by** (*metis double-compl FusionSFalse salways-def sfalse-def ssometime-def strue-def*)

**lemma** *SBiGen*:

**assumes**  $X = STrue$   
**shows**  $(S\text{Bi } X) = STrue$   
**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(1) assms(2)*  
**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*:  
 $(S\text{Init } X) \subseteq (S\text{Bi } (S\text{Init } X))$   
**by** (*simp add: ST28 ST38 sbi-def*)

**lemma** *SNextImpNotNextNot*:  
 $(S\text{Next } X) \subseteq \neg(S\text{Next } (\neg X))$

**proof** –

**have** 1:  $((S\text{Next } X) \subseteq \neg(S\text{Next } (\neg X))) = (((S\text{Next } X) \cap (S\text{Next } (\neg X))) \subseteq SFalse)$

**by** (*simp add: disjoint-eq-subset-Compl sfalse-def*)

**have** 2:  $((S\text{Next } X) \cap (S\text{Next } (\neg X))) = S\text{Skip} \cdot (X \cap (\neg X))$

**by** (*metis CH07 SStar16 inf.orderE snext-def*)

**have** 3:  $(S\text{Skip} \cdot (X \cap (\neg X))) = S\text{Skip} \cdot SFalse$

**by** (*simp add: sfalse-def*)

**have** 4:  $S\text{Skip} \cdot SFalse = SFalse$  **by** (*simp add: FusionSFalse*)

**from** 1 2 3 4 **show** *?thesis* **by** *auto*

**qed**

**lemma** *SBiBoxChopImpChop*:

$(S\text{Bi } ((-X) \cup X1)) \cap (S\text{Always } ((-Y) \cup Y1)) \cap (X \cdot Y) \subseteq (X1 \cdot Y1)$   
**using** *BD45 BD46* **by** *blast*

**lemma** *SBoxInduct*:

$(S\text{Always } (-X \cup (S\text{Wnext } X))) \cap X \subseteq (S\text{Always } X)$

**proof** –

**have** 1:  $((S\text{Always } (-X \cup (S\text{Wnext } X))) \cap X \subseteq (S\text{Always } X)) =$   
 $((S\text{Sometime } (-X)) \subseteq ((-X) \cup (S\text{Sometime } (X \cap (S\text{Next } (-X))))))$   
**by** (*simp add: salways-def snext-def swnext-def*)  
*blast*

**have** 2:  $((S\text{Sometime } (-X)) \subseteq ((-X) \cup (S\text{Sometime } (X \cap (S\text{Next } (-X)))))) =$   
 $((S\text{Star } S\text{Skip}) \cdot (-X)) \subseteq ((-X) \cup ((S\text{Star } S\text{Skip}) \cdot (X \cap (S\text{Skip} \cdot (-X)))))$   
**by** (*simp add: SStarSkip snext-def ssometime-def*)

**have** 3:  $((S\text{Star } S\text{Skip}) \cdot (-X)) \subseteq ((-X) \cup ((S\text{Star } S\text{Skip}) \cdot (X \cap (S\text{Skip} \cdot (-X)))))$   
**using** *SStar51 SStar50* **by** *blast*

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

**qed**

**lemma** *SChopstarEqv*:

$(S\text{Star } X) = S\text{Empty} \cup (X \cap S\text{More}) \cdot (S\text{Star } X)$

**using** *SStar52 SStar53* **by** *blast*

## 16.5 Extra Laws

### 16.5.1 Boolean Laws

**lemma** *B02*:

**assumes**  $-Y \subseteq -X$

**shows**  $(X :: 'a \text{ intervals}) \subseteq Y$

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

**lemma** *B03*:

$((X :: 'a \text{ intervals}) = Y) \longleftrightarrow (-X = -Y)$

**by** *auto*

**lemma** *B05*:

**assumes**  $(X :: 'a \text{ intervals}) \cup Y \subseteq Z$

**shows**  $X \subseteq Z \wedge Y \subseteq Z$

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

**lemma** *B06*:

**assumes**  $X \subseteq Z \wedge Y \subseteq Z$

**shows**  $(X :: 'a \text{ intervals}) \cup Y \subseteq Z$

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

**lemma** *B07*:

$(X :: 'a \text{ intervals}) \cup Y \subseteq Z \longleftrightarrow$

$X \subseteq Z \wedge Y \subseteq Z$

**by** *auto*

**lemma** *B08*:

**assumes**  $(X:: 'a \text{ intervals}) \subseteq Y$   
**shows**  $\neg X \cup Y = STrue$   
**using** *assms*  
**using** *strue-def* **by** *auto*

**lemma B10:**  
 $(X:: 'a \text{ intervals}) \subseteq Y \longleftrightarrow \neg X \cup Y = STrue$   
**using** *strue-def* **by** *auto*

**lemma B11:**  
**assumes**  $(X:: 'a \text{ intervals}) \subseteq Y$   
**shows**  $X \cap \neg Y = SFalse$   
**using** *assms sfalse-def* **by** *auto*

**lemma B12:**  
**assumes**  $X \cap \neg Y = SFalse$   
**shows**  $(X:: 'a \text{ intervals}) \subseteq Y$   
**using** *assms sfalse-def* **by** *auto*

**lemma B13:**  
 $(X:: 'a \text{ intervals}) \subseteq Y \longleftrightarrow X \cap \neg Y = SFalse$   
**using** *sfalse-def* **by** *auto*

**lemma B14:**  
**assumes**  $(X:: 'a \text{ intervals}) \subseteq Y$   
**shows**  $X \cap Y = X$   
**using** *assms* **by** *auto*

**lemma B15:**  
**assumes**  $(X:: 'a \text{ intervals}) \subseteq Y \cap Z$   
**shows**  $X \subseteq Y \wedge X \subseteq Z$   
**using** *assms* **by** *auto*

**lemma B16:**  
**assumes**  $X \subseteq Y \wedge X \subseteq Z$   
**shows**  $(X:: 'a \text{ intervals}) \subseteq Y \cap Z$   
**using** *assms* **by** *auto*

**lemma B17:**  
 $(X:: 'a \text{ intervals}) \subseteq Y \cap Z \longleftrightarrow X \subseteq Y \wedge X \subseteq Z$   
**by** *auto*

**lemma B18:**  
**assumes**  $(X:: 'a \text{ intervals}) \subseteq Y \cup Z$   
**shows**  $X \cap \neg Y \subseteq Z$   
**using** *assms* **by** *auto*

**lemma B19:**  
**assumes**  $X \cap \neg Y \subseteq Z$   
**shows**  $(X:: 'a \text{ intervals}) \subseteq Y \cup Z$



using *assms* by *auto*

**lemma B21:**

$$(X:: 'a \text{ intervals}) \cup (Y \cap Z) \subseteq (X \cup Y) \cap (X \cup Z) \longleftrightarrow \\ X \cup (Y \cap Z) \subseteq (X \cup Y) \wedge X \cup (Y \cap Z) \subseteq (X \cup Z)$$

by *auto*

**lemma B22:**

$$(X:: 'a \text{ intervals}) \cup (Y \cap Z) \subseteq X \cup Y$$

by *auto*

**lemma B23:**

$$(X:: 'a \text{ intervals}) \cup (Y \cap Z) \subseteq X \cup Z$$

by *auto*

**lemma B24:**

$$((X:: 'a \text{ intervals}) \cup Y) \cap (X \cup Z) \subseteq X \cup (Y \cap Z) \longleftrightarrow \\ ((X \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y \cap Z$$

by *auto*

**lemma B25:**

$$(((X:: 'a \text{ intervals}) \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y \cap Z \longleftrightarrow \\ ((X \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y \wedge \\ ((X \cup Y) \cap (X \cup Z)) \cap -X \subseteq Z$$

by *auto*

**lemma B26:**

$$(((X:: 'a \text{ intervals}) \cup Y) \cap (X \cup Z)) \cap -X \subseteq Y$$

by *auto*

**lemma B27:**

$$(((X:: 'a \text{ intervals}) \cup Y) \cap (X \cup Z)) \cap -X \subseteq Z$$

by *auto*

**lemma B29:**

$$(X:: 'a \text{ intervals}) \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$$

by *auto*

## 16.5.2 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**

### 16.5.3 Next

**lemma** *N01*:

$$(SNext SEmpty) = SSkip$$

**by** (*simp add: FusionSEmptyR snext-def*)

**lemma** *N02*:

$$(SNext SFalse) = SFalse$$

**by** (*simp add: FusionSFalse 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:**

$SMore \subseteq SSkip \cdot STrue$   
**by** (*simp add: smore-def*)  
*(metis B10 SStarSkip UnfoldL double-complement)*

**lemma N23:**

$(SWprev X) \subseteq (SEmpty \cup (SPrev X))$   
**proof** –  
**have**  $X \cdot SSkip \cup - X \cdot SSkip = SStar SSkip \cdot SSkip$   
**by** (*metis (no-types) Compl-empty-eq FusionUnionDistL SStarSkip strue-def sup-compl-top*)  
**then have**  $- SWprev X \cup (SEmpty \cup SPrev X) = STrue$   
**by** (*metis (no-types) SStar19 SStarSkip UnfoldL UnionAssoc double-compl spreve-def sup-commute supprev-def*)  
**then show** *?thesis*  
**by** (*meson B09*)  
**qed**

**lemma N24:**

$(SEmpty) \subseteq (SWprev X)$   
**by** (*metis B10 B02 FusionRuleL SSkipFusionImpSMore SStar30 SStarSkip UnfoldL compl-bot-eq double-compl smore-def strue-def subset-antisym supprev-def top-greatest*)

**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 spreve-def supprev-def*)  
**have 2:**  $((SPrev X) \cap (SPrev (-X))) = (X \cap (-X)) \cdot SSkip$   
**by** (*metis CH08 SStar16 inf.orderE spreve-def*)  
**have 3:**  $(X \cap (-X)) \cdot SSkip = SFalse \cdot SSkip$   
**by** (*simp add: sfalse-def*)  
**have 4:**  $SFalse \cdot SSkip = SFalse$   
**by** (*simp add: SFalseFusion*)

from 1 2 3 4 show ?thesis by auto  
qed

lemma N26:  
 $(SWprev\ X) = (SEmpty \cup (SPrev\ X))$   
 using N23 N24 N25 by blast

lemma N09:  
 $SSkip \cup SMore \cdot SSkip \subseteq SMore$   
**proof** –  
 have 1:  $SSkip \subseteq SMore$  by (simp add: smore-def sskip-def)  
 have 2:  $SMore \cdot SSkip \subseteq SMore$   
 by (metis N26 UnionCommute compl-le-swap1 smore-def sup-ge2 supprev-def)  
 from 1 2 show ?thesis by auto  
 qed

lemma N10:  
 assumes  $SSkip \cup SMore \cdot SSkip \subseteq SMore$   
 shows  $SSkip \cdot (SStar\ SSkip) \subseteq SMore$   
 using assms  
 using SStarInductR N09 by blast

lemma N11:  
 $SSkip \cdot STrue \subseteq SMore$   
 by (metis SStarSkip N09 N10)

lemma N12:  
 $(SNext\ X) = -(SWnext\ (-X))$   
 by (simp add: snext-def swnext-def)

lemma N13:  
 $SMore \cdot STrue = SMore$   
 by (metis FusionAssoc N11 N08 SStar48 SStarSkip ST47 UnfoldL subset-antisym sup.right-idem)

lemma N14:  
 $STrue \cdot SSkip \subseteq SMore$   
 by (metis N11 SStar19 SStarSkip)

lemma N15:  
 $SMore \subseteq STrue \cdot SSkip$   
 by (metis N08 SStar19 SStarSkip)

lemma N19:  
 $(SWnext\ X) \subseteq (SEmpty \cup (SNext\ X))$   
**proof** –  
 have  $SSkip \cdot X \cup SSkip \cdot (-X) = SSkip \cdot SStar\ SSkip$   
 using FusionUnionDistR[of SSkip X -X] SStarSkip  
 by (metis STrueTop)  
 then have  $SWnext\ X \cup (SEmpty \cup SNext\ X) = STrue$   
 by (metis SStarSkip UnfoldL UnionCommute double-compl inf-sup-aci(7) snext-def swnext-def)

**then show** *?thesis* **by** (*simp add: B09*)  
**qed**

**lemma** *N20*:

$(SEmpty) \subseteq (SWnext\ X)$

**proof** –

**have** 1:  $((SEmpty) \subseteq (SWnext\ X)) = (\neg(SWnext\ X) \subseteq SMore)$

**by** (*simp add: smore-def*)

**have** 2:  $(\neg(SWnext\ X) \subseteq SMore) = ((SNext\ \neg X) \subseteq SMore)$

**by** (*simp add: snext-def swnext-def*)

**have** 3:  $(SNext\ \neg X) \subseteq SSkip \cdot STrue$

**by** (*metis FusionUnionDistR STrueTop snext-def sup.orderI sup.right-idem*)

**hence** 4:  $(SNext\ \neg X) \subseteq SMore$  **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 swnext-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: swnext-def*)

(*metis (no-types, lifting) CH07 SStar16 compl-inf inf.orderE*)

**lemma** *N27*:

$(SNext\ ((\neg X) \cup Y)) \subseteq (\neg(SNext\ X) \cup (SNext\ Y))$

**using** *N04 SNextImpNotNextNot* **by** *blast*

**lemma** *N28*:

$(SPrev\ ((\neg X) \cup Y)) \subseteq (\neg(SPrev\ X) \cup (SPrev\ Y))$

**unfolding** *sprev-def*

**proof** –

**have**  $\bigwedge I. (SEmpty) \cup I \cdot SSkip = \neg(\neg I \cdot SSkip)$

**using** *N26 sprev-def swprev-def* **by** *blast*

**then have**  $(Y \cup \neg X) \cdot SSkip \subseteq Y \cdot SSkip \cup \neg(X \cdot SSkip)$

**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: spredef swprevdef*)

#### 16.5.4 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*:  
 $(X \cap SEmpty) \subseteq (X \cap SEmpty) \cdot (X \cap SEmpty)$   
**by** (*metis (no-types, hide-lams) B04 CH09 FusionSEmptyL FusionSEmptyR inf.idem inf-assoc inf-left-commute*)

**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 FusionAssoc 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*:

$(X \cap \text{SEmpty}) \cdot (Y \cap \text{SEmpty}) \subseteq (Y \cap \text{SEmpty})$

**using** *ST01* **by** *blast*

**lemma** *ST12*:

$(X \cap \text{SEmpty}) \cap (Y \cap \text{SEmpty}) = (X \cap \text{SEmpty}) \cdot \text{SEmpty} \cap (Y \cap \text{SEmpty}) \cdot \text{SEmpty}$

**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:**

$$(SInit\ X) \cap (SDi\ Y) = (SDi\ (SInit\ X) \cap Y)$$

**unfolding** *sdi-def sinit-def strue-def*

**by** (*metis CH09 FusionAssoc FusionSEmptyL compl-bot-eq inf-top.left-neutral*)

**lemma ST31:**

$$(X \cdot (STrue \cap SEmpty)) \cap (STrue \cdot (Y \cap SEmpty)) = X \cdot (Y \cap SEmpty)$$

**by** (*metis Int-commute compl-bot-eq inf-top.right-neutral CH10 strue-def*)

**lemma ST32:**

$$(STrue \cap SEmpty) \cdot SEmpty \cap (SInit\ X) = (X \cap SEmpty)$$

**by** (*metis Compl-empty-eq Int-commute CH09 ST14 inf-top.right-neutral sinit-def strue-def*)

**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*:

$$\neg(STrue \cdot (X \cap SEmpty)) \subseteq STrue \cdot ((\neg X) \cap SEmpty)$$

by (metis B09 FusionSEmptyR FusionUnionDistR ST21 double-compl)

**lemma** *ST49*:

$$STrue \cdot ((\neg X) \cap SEmpty) \subseteq \neg(STrue \cdot (X \cap SEmpty))$$

by (metis CH10 Compl-disjoint2 FusionSFalse disjoint-eq-subset-Compl inf.idem inf-compl-bot-left2 sfalse-def)

**lemma** *ST50*:

$$\neg(STrue \cdot (X \cap SEmpty)) = STrue \cdot ((\neg X) \cap SEmpty)$$

using *ST48 ST49* by blast

### 16.5.5 SStar

**lemma** *SStar02*:

assumes  $X \subseteq Y$

shows  $X \cdot (SStar\ Y) \cup SEmpty \subseteq (SStar\ Y)$

using *assms*

by (metis FusionUnionDistL Int-lower1 SStar15 Un-commute Un-mono UnfoldL inf.orderE sup.orderE sup.orderI)

**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*:

$$(SStar\ SSkip) = STrue$$

**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*:

$X \subseteq (SStar\ X)$

**by** (*metis FusionSEmptyR FusionUnionDistR SStar08 UnCI UnfoldL subsetI subset-iff*)

**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*:

$(SStar\ (X \cdot Y)) \cdot X \subseteq X \cdot (SStar\ (Y \cdot X))$

**using** *SStar31 SStar32* **by** *blast*

**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*)

$assms \text{ sup.absorb-iff1}$ )  
**then show**  $?thesis$   
**by** ( $meson \text{ SStarInductL sup.absorb-iff1}$ )  
**qed**

**lemma**  $\text{SStar38}$ :  
**assumes**  $Z \cdot X \subseteq Y \cdot Z$   
**shows**  $Z \cdot (\text{SStar } X) \subseteq (\text{SStar } Y) \cdot Z$   
**using**  $assms$   
**proof** –  
**have**  $f1$ :  $Z \cup \text{SStar } Y \cdot Y \cdot Z = \text{SStar } Y \cdot Z$   
**by** ( $metis \text{ (no-types) SStar30 ST47 UnfoldL}$ )  
**have**  $\text{SStar } Y \cdot Y \cdot Z = \text{SStar } Y \cdot Z \cdot X \cup \text{SStar } Y \cdot Y \cdot Z$   
**by** ( $metis \text{ FusionAssoc FusionUnionDistR assms subset-Un-eq}$ )  
**then have**  $Z \cup \text{SStar } Y \cdot Z \cdot X \subseteq \text{SStar } Y \cdot Z$   
**using**  $f1$  **by**  $blast$   
**then show**  $?thesis$   
**by** ( $simp \text{ add: SStarInductR}$ )  
**qed**

**lemma**  $\text{SStar39}$ :  
 $Y \cdot (\text{SStar } ((\text{SStar } X) \cdot Y)) \subseteq (\text{SStar } (Y \cdot (\text{SStar } X))) \cdot Y$   
**by** ( $simp \text{ add: SStar38 FusionAssoc}$ )

**lemma**  $\text{SStar40}$ :  
 $(\text{SStar } (Y \cdot (\text{SStar } X))) \cdot Y \subseteq Y \cdot (\text{SStar } ((\text{SStar } X) \cdot Y))$   
**by** ( $simp \text{ add: SStar33}$ )

**lemma**  $\text{SStar41}$ :  
 $Y \cdot (\text{SStar } ((\text{SStar } X) \cdot Y)) = (\text{SStar } (Y \cdot (\text{SStar } X))) \cdot Y$   
**using**  $\text{SStar39 SStar40}$  **by**  $blast$

**lemma**  $\text{SStar42}$ :  
 $Z \cdot (\text{SStar } (Y \cdot Z)) \subseteq (\text{SStar } (Z \cdot Y)) \cdot Z$   
**by** ( $simp \text{ add: SStar38 FusionAssoc}$ )

**lemma**  $\text{SStar43}$ :  
 $(\text{SStar } (Z \cdot Y)) \cdot Z \subseteq Z \cdot (\text{SStar } (Y \cdot Z))$   
**by** ( $simp \text{ add: SStar33}$ )

**lemma**  $\text{SStar44}$ :  
 $Z \cdot (\text{SStar } (Y \cdot Z)) = (\text{SStar } (Z \cdot Y)) \cdot Z$   
**using**  $\text{SStar42 SStar43}$  **by**  $blast$

**lemma**  $\text{SStar49}$ :  
 $(\text{SStar } X) = \text{SEmpty} \cup (\text{SStar } X) \cdot X$   
**using**  $\text{SStar30 UnfoldL}$  **by**  $blast$

### 16.5.6 Box and Diamond

**lemma** *BD01*:

$$(SSometime SEmpty) = STrue$$

**by** (*simp add: ssometime-def FusionSEmptyR*)

**lemma** *BD02*:

$$X \subseteq (SSometime X)$$

**unfolding** *ssometime-def*

**by** (*metis CH01 SStar25 ST47 UnfoldL 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 CH01 FusionAssoc FusionUnionDistL FusionUnionDistR SStar16 SStarSkip snext-def ssometime-def sup.absorb-iff2*)

**lemma** *BD05*:

$$(SSometime X) \cup (SSometime Y) = (SSometime (X \cup Y))$$

**by** (*simp add: FusionUnionDistR ssometime-def*)

**lemma** *BD06*:

$$(SSometime STrue) = STrue$$

**by** (*simp add: CH01 ssometime-def*)

**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 subset-trans*)

**lemma** *BD11*:

$$(SSometime (SSometime X)) = (SSometime X)$$

**by** (*simp add: CH01 ssometime-def FusionAssoc*)

**lemma** *BD12*:

$(SAlways\ X) \subseteq X$   
**by** (*simp add: B02 BD02 BD09*)

**lemma** *BD13*:  
 $(SDi\ STrue) = STrue$   
**by** (*simp add: CH01 sdi-def*)

**lemma** *BD14*:  
 $(SDi\ SEmpty) = STrue$   
**by** (*simp add: sdi-def FusionSEmptyL*)

**lemma** *BD15*:  
 $(SBi\ STrue) = STrue$   
**by** (*simp add: SBiGen*)

**lemma** *BD16*:  
 $(SDi\ (X \cup Y)) = (SDi\ X) \cup (SDi\ Y)$   
**by** (*simp add: FusionUnionDistL sdi-def*)

**lemma** *BD17*:  
**assumes**  $X \subseteq Y$   
**shows**  $(SDi\ X) \subseteq (SDi\ Y)$   
**using** *assms*  
**by** (*metis FusionUnionDistL Subsumption sdi-def*)

**lemma** *BD18*:  
 $(SDi\ (SDi\ X)) = (SDi\ X)$   
**by** (*metis CH01 FusionAssoc sdi-def*)

**lemma** *BD19*:  
 $(SDa\ SEmpty) = STrue$   
**by** (*simp add: CH01 sda-def FusionSEmptyR*)

**lemma** *BD20*:  
 $(SDa\ STrue) = STrue$   
**by** (*simp add: CH01 sda-def*)

**lemma** *BD21*:  
 $(SBa\ STrue) = STrue$   
**by** (*metis BD15 BD08 BD09 sba-def sbi-def sda-def sdi-def ssometime-def*)

**lemma** *BD22*:  
 $(SDa\ (X \cup Y)) = (SDa\ X) \cup (SDa\ Y)$   
**by** (*simp add: FusionUnionDistL FusionUnionDistR sda-def*)

**lemma** *BD23*:  
**assumes**  $X \subseteq Y$   
**shows**  $(SDa\ X) \subseteq (SDa\ Y)$   
**using** *assms*  
**by** (*metis BD22 Subsumption*)

**lemma** *BD24*:

**assumes**  $X \subseteq Y$

**shows**  $(SDa (-Y)) \subseteq (SDa (-X))$

**using** *assms*

**by** (*simp add: BD23*)

**lemma** *BD25*:

$(SDi X) \subseteq (SDa X)$

**by** (*metis BD02 FusionAssoc sda-def sdi-def ssometime-def*)

**lemma** *BD26*:

$(SSometime X) \subseteq (SDa X)$

**by** (*metis BD01 BD02 FusionSEmptyR FusionUnionDistR SStar14 le-iff-sup sda-def*)

**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*:

$(SAlways X) \cap (SAlways Y) = (SAlways (X \cap Y))$

**by** (*metis BD05 BD09 Morgan compl-inf salways-def*)

**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 N13 SStar14 SStar30 SStar48 SStarSkip double-complement salways-def smore-def*)

**lemma** *BD36*:

$(SBi\ SEmpty) = SEmpty$

**using** *N13 sbi-def sdi-def smore-def* **by** *fastforce*

**lemma** *BD37*:

$(SBa\ SEmpty) = SEmpty$

**by** (*metis N13 SStar30 SStar48 double-complement sba-def sda-def smore-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 CH01 FusionAssoc sda-def*)

**lemma** *BD44*:

$(SBa\ (SBa\ X)) = (SBa\ X)$

**by** (*simp add: BD43 sba-def*)

**lemma** *BD47*:

$(SAlways\ (\neg X) \cup Y) \subseteq (\neg (SAlways\ X) \cup (SAlways\ Y))$

**by** (*metis B20 BD12 BD29 BD38 BD42 double-compl*)

**lemma** *BD48*:

$$(SAlways\ X) \subseteq X \cap (SWnext\ (SAlways\ X))$$

**by** (*metis B02 B16 BD03 BD09 BD12 N12 salways-def*)

**lemma** *BD49*:

$$(SBi\ (\neg X) \cup Y) \subseteq (\neg(SBi\ X) \cup (SBi\ Y))$$

**by** (*metis B20 BD45 Un-commute double-complement sbi-def sdi-def*)

**lemma** *BD50*:

$$(SPrev\ (SDi\ X)) \subseteq (SDi\ X)$$

**by** (*metis B04 FusionAssoc FusionUnionDistR N08 SSkipFusionImpSMore SStar19 SStarSkip STrueTop sdi-def smore-def sprev-def sup-ge2*)

**lemma** *BD51*:

$$\neg(SBi\ X) = (SDi\ (\neg X))$$

**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*:

$$(SAlways\ (SMore \cup X)) = STrue \cdot (X \cap SEmpty)$$

**by** (*simp add: SStar14 SStarSkip ST50 UnionCommute salways-def smore-def*)

## 16.6 Time Reversal

### 16.6.1 Time Reversal Axioms

**lemma** *SRevSEmpty*:

$$(SRev\ SEmpty) = SEmpty$$

**using** *set-eqI*[*of* (*SRev SEmpty*) *SEmpty*]

**by** (*simp add: sempty-elim srev-elim*)

**lemma** *SRevSNot*:

$$(SRev\ (\neg X)) = (\neg (SRev\ X))$$

**using** *set-eqI*[*of* (*SRev* ( $\neg X$ )) ( $\neg (SRev\ X)$ )]



**by** (*simp add: srev-elim*)

**lemma** *fuse-irev*:

**assumes** *ilast xs = ifirst ys*

**shows**  $\text{irev} (\text{fuse } xs \text{ } ys) = \text{fuse} (\text{irev } ys) (\text{irev } xs)$

**using** *assms fuse-prefix-suffix*

*prefix-fuse[of xs ys]*

*suffix-fuse[of xs ys]*

**by** (*metis irev-ilen irev-prefix irev-suffix prefix-ilen-bound suffix-ilen*)

**lemma** *SRevFusionsem*:

$x \in (\text{SRev } (X \cdot Y)) = (x \in ((\text{SRev } Y) \cdot (\text{SRev } X)))$

**by** (*auto simp add: srev-elim fusion-iff* )

(*metis fuse-irev ilast-irev irev-irev-ident, metis fuse-irev ifirst-irev ilast-irev*)

**lemma** *SRevFusion*:

$(\text{SRev } (X \cdot Y)) = (\text{SRev } Y) \cdot (\text{SRev } X)$

**using** *set-eqI[of (SRev (X · Y)) (SRev Y) · (SRev X)]*

**using** *SRevFusionsem* **by** *auto*

**lemma** *SRevUnion*:

$(\text{SRev } (X \cup Y)) = (\text{SRev } X) \cup (\text{SRev } Y)$

**using** *set-eqI[of (SRev (X ∪ Y)) (SRev X) ∪ (SRev Y)]*

**using** *srev-elim* **by** *auto*

**lemma** *SRevSPower*:

$(\text{SRev } (\text{SPower } X \text{ } n)) = (\text{SPower } (\text{SRev } X) \text{ } n)$

**proof** (*induct n*)

**case** *0*

**then show** *?case* **by** (*simp add: SRevSEmpty*)

**next**

**case** (*Suc n*)

**then show** *?case*

**proof** –

**have**  $\text{SRev } X \cap \text{SMore} = \text{SRev } (X \cap \text{SMore})$

**by** (*metis (no-types) Morgan SRevSEmpty SRevSNot SRevUnion smore-def*)

**then show** *?thesis*

**by** (*simp add: SRevFusion Suc.hyps spower-commutes*)

**qed**

**qed**

**lemma** *SRevSStar*:

$(\text{SRev } (\text{SStar } X)) = (\text{SStar } (\text{SRev } X))$

**proof** –

**have** *1*:  $(\text{SRev } (\text{SStar } X)) = (\text{SRev } (\bigcup n. \text{SPower } X \text{ } n))$  **by** (*simp add: sstar-def*)

**have** *2*:  $(\text{SRev } (\bigcup n. \text{SPower } X \text{ } n)) = (\bigcup n. \text{SPower } (\text{SRev } X) \text{ } n)$

**using** *set-eqI[of (SRev (∪ n. SPower X n)) (∪ n. SPower (SRev X) n)]*

*SRevSPower[of X]*

**by** (*metis srev-elim sstar-def sstar-elim*)

**have**  $\exists: (\bigcup n. SPower (SRev X) n) = (SStar (SRev X))$  **by** (*simp add: sstar-def*)  
**from** 1 2 3 **show** *?thesis* **by** *auto*  
**qed**

**lemma** *SRevSRev*:  
 $(SRev (SRev X)) = X$   
**using** *set-eqI[of (SRev (SRev X)) X]*  
**by** (*simp add: srev-elim*)

### 16.6.2 Time Reversal Laws

**lemma** *TR01*:  
 $(SRev SMore) = SMore$   
**by** (*simp add: SRevSEmpty SRevSNot smore-def*)

**lemma** *TR02*:  
 $(SRev SSkip) = SSkip$   
**by** (*metis SRevFusion SRevSEmpty SRevSNot SRevUnion TR01 sskip-def*)

**lemma** *TR03*:  
 $(SRev STrue) = STrue$   
**by** (*metis SRevSStar SStarSkip TR02*)

**lemma** *TR04*:  
 $(SRev SFalse) = SFalse$   
**by** (*metis Compl-eq-Compl-iff SRevSNot TR03 sfalse-def strue-def*)

**lemma** *TR05*:  
 $(SRev SSometime X) = (SDi (SRev X))$   
**by** (*simp add: SRevFusion TR03 sdi-def ssometime-def*)

**lemma** *TR06*:  
 $(SRev (SAlways X)) = (SBi (SRev X))$   
**by** (*simp add: SRevSNot TR05 salways-def sbi-def*)

**lemma** *TR07*:  
 $(SRev (SDi X)) = (SSometime (SRev X))$   
**by** (*simp add: SRevFusion TR03 sdi-def ssometime-def*)

**lemma** *TR08*:  
 $(SRev (SBi X)) = (SAlways (SRev X))$   
**by** (*metis SRevSRev TR06*)

**lemma** *TR09*:  
 $(SRev (SNext X)) = (SPrev (SRev X))$   
**by** (*simp add: SRevFusion TR02 snext-def sprev-def*)

**lemma** *TR10*:  
 $(SRev (SWnext X)) = (SWprev (SRev X))$   
**by** (*simp add: SRevFusion SRevSNot TR02 swnext-def swprev-def*)

**lemma** *TR11*:

$(SRev (SPrev X)) = (SNext (SRev X))$

**by** (*simp add: SRevFusion TR02 snext-def sprevedef*)

**lemma** *TR12*:

$(SRev (SWprev X)) = (SWnext (SRev X))$

**by** (*metis SRevSRev TR10*)

**lemma** *TR13*:

$(SRev (SDa X)) = (SDa (SRev X))$

**by** (*simp add: SRevFusion TR03 sda-def FusionAssoc*)

**lemma** *TR14*:

$(SRev (SBa X)) = (SBa (SRev X))$

**by** (*simp add: SRevSNot TR13 sba-def*)

**lemma** *TR15*:

$(SRev (SPower SSkip n)) = (SPower SSkip n)$

**by** (*simp add: SRevSPower TR02*)

**lemma** *TR16*:

**assumes**  $X \subseteq Y$

**shows**  $(SRev X) \subseteq (SRev Y)$

**using** *assms* **by** (*metis SRevUnion le-iff-sup*)

**lemma** *TR17*:

**assumes**  $X = Y$

**shows**  $(SRev X) = (SRev Y)$

**using** *assms TR16* **by** *auto*

## 16.7 Link between Set of Intervals and ITL

**lemma** *interval-lan [simp]*:

$\sigma \in (\text{lan } f) \longleftrightarrow (\sigma \models f)$

**by** (*simp add: lan-def*)

**lemma** *valid-lan-equiv* :

$((\text{lan } f) = (\text{lan } g)) \longleftrightarrow (\vdash f = g)$

**using** *interval-lan lan-def Valid-def* **by** *fastforce*

**lemma** *valid-lan-imp* :

$((\text{lan } f) \subseteq (\text{lan } g)) \longleftrightarrow (\vdash f \longrightarrow g)$

**using** *interval-lan lan-def Valid-def*

**by** (*simp add: Valid-def lan-def Collect-mono-iff*)

**lemma** *valid-strue* :

$((\text{lan } f) = STrue) \longleftrightarrow (\vdash f)$

**using** *strue-def* **by** *fastforce*

**lemma** *strue-true*:

$$\sigma \in STrue \longleftrightarrow (\sigma \models \#True)$$

**by** (*simp add: strue-elim*)

**lemma** *strue-true-1*:

$$STrue = (\text{lan } (LIFT \ \#True))$$

**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*:

$$SFalse = (\text{lan } (LIFT(\#False)))$$

**using** *sfalse-false* **using** *lan-def* **by** *fastforce*

**lemma** *not-negation*:

$$\sigma \in (\neg(\text{lan } f)) \longleftrightarrow (\sigma \models \neg f)$$

**by** *simp*

**lemma** *not-negation-1*:

$$\neg(\text{lan } f) = (\text{lan } (LIFT(\neg f)))$$

**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*:

$$((\text{lan } f) \cap (\text{lan } g)) = (\text{lan } (LIFT(f \wedge g)))$$

**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*:

$$((\text{lan } f) \cup (\text{lan } g)) = (\text{lan } (LIFT(f \vee g)))$$

**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*:

$$(\neg(\text{lan } f) \cup (\text{lan } g)) = (\text{lan } (LIFT(f \longrightarrow g)))$$

**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** (*metis fusion-iff chop-fuse interval-lan*)

**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*:

$\sigma \in \text{SEmpty} \longleftrightarrow (\sigma \models \text{empty})$

**by** (*simp add: empty-defs empty-elim*)

**lemma** *empty-empty-1*:

$\text{SEmpty} = (\text{lan } (\text{LIFT } \text{empty}))$

**using** *empty-empty lan-def* **by** *fastforce*

**lemma** *smore-more*:

$\sigma \in \text{SMore} \longleftrightarrow (\sigma \models \text{more})$

**by** (*simp add: more-defs smore-elim*)

**lemma** *smore-more-1*:

$\text{SMore} = (\text{lan } (\text{LIFT } \text{more}))$

**using** *smore-more lan-def* **by** *fastforce*

**lemma** *sskip-skip*:

$\sigma \in \text{SSkip} = (\sigma \models \text{skip})$

**by** (*simp add: skip-defs sskip-elim*)

**lemma** *sskip-skip-1*:

$\text{SSkip} = (\text{lan } (\text{LIFT } \text{skip}))$

**using** *sskip-skip lan-def* **by** *fastforce*

**lemma** *snext-next*:

$\sigma \in (\text{SNext } (\text{lan } f)) \longleftrightarrow (\sigma \models \bigcirc f)$

**by** (*metis snext-def fusion-chop next-d-def sskip-skip-1*)

**lemma** *snext-next-1*:

$(\text{SNext } (\text{lan } f)) = (\text{lan } (\text{LIFT}(\bigcirc f)))$

**using** *snext-next lan-def* **by** *fastforce*

**lemma** *swnext-wnext*:

$\sigma \in (\text{SWnext } (\text{lan } f)) \longleftrightarrow (\sigma \models \text{wnext } f)$

**by** (*simp add: swnext-def fusion-chop-1 next-d-def not-negation-1 sskip-skip-1 wnext-d-def*)

**lemma** *swnext-wnext-1*:

$(\text{SWnext } (\text{lan } f)) = (\text{lan } (\text{LIFT}(\text{wnext } f)))$

**using** *swnext-wnext lan-def* **by** *fastforce*

**lemma** *sprev-prev*:

$\sigma \in (\text{SPrev } (\text{lan } f)) \longleftrightarrow (\sigma \models \text{prev } f)$

**by** (*metis fusion-chop prev-d-def sprev-def sskip-skip-1*)

**lemma** *sprev-prev-1*:

$(SPrev (lan f)) = (lan (LIFT(prev f)))$

**using** *sprev-prev lan-def* **by** *fastforce*

**lemma** *suprev-wprev*:

$\sigma \in (SWprev (lan f)) \longleftrightarrow (\sigma \models wprev f)$

**by** (*simp add: fusion-chop-1 not-negation-1 prev-d-def sskip-skip-1 suprev-def wprev-d-def*)

**lemma** *suprev-wprev-1*:

$(SWprev (lan f)) = (lan (LIFT(wprev f)))$

**using** *suprev-wprev lan-def* **by** *fastforce*

**lemma** *sinit-init*:

$\sigma \in SInit (lan f) \longleftrightarrow (\sigma \models init f)$

**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*:

$SInit (lan f) = (lan (LIFT(init f)))$

**using** *sinit-init lan-def* **by** *fastforce*

**lemma** *and-inter-more*:

$\sigma \in (((lan f) \cap SMore)) \longleftrightarrow (\sigma \models (f \wedge more) )$

**using** *smore-more inter-and* **by** *auto*

**lemma** *and-inter-more-1*:

$\sigma \in (((lan f) \cap SMore)) \longleftrightarrow (\sigma \in (lan (LIFT(f \wedge more)) ) )$

**using** *and-inter-more lan-def* **by** (*simp add: smore-more-1*)

**lemma** *and-inter-more-2*:

$((lan f) \cap SMore) = (lan (LIFT(f \wedge more)) )$

**using** *and-inter-more-1* **by** *blast*

**lemma** *and-chop*:

$\sigma \in (((lan f) \cap SMore) \cdot (lan g)) \longleftrightarrow (\sigma \models (f \wedge more);g)$

**by** (*metis fusion-chop inter-and-1 smore-more-1*)

**lemma** *and-chop-1*:

$((((lan f) \cap SMore) \cdot (lan g)) = (lan (LIFT((f \wedge more);g)))$

**using** *and-chop lan-def* **by** *blast*

**lemma** *spower-chop-power*:

$(SPower (lan f) n) = (lan (LIFT(power (f \wedge more) n)))$

**proof** (*induct n*)

**case** 0

**then show** ?case **by** (*simp add: empty-empty-1*)

**next**

**case** (*Suc n*)

**then show** ?case **by** (*metis and-chop-1 pow-Suc pwr-Suc*)

**qed**

**lemma** *sstar-power*:

$\sigma \in SStar\ (lan\ f) \longleftrightarrow (\exists\ n.\ \sigma \in SPower\ (lan\ f)\ n)$   
**by** (*simp add: sstar-def*)

**lemma** *sstar-chopstar*:

$\sigma \in (SStar\ (lan\ f)) \longleftrightarrow \sigma \in (lan\ (LIFT(f^*)))$   
**proof** –  
**have** 1:  $\sigma \in (SStar\ (lan\ f)) = (\exists\ n.\ \sigma \in SPower\ (lan\ f)\ n)$   
**using** *sstar-power* **by** *blast*  
**have** 2:  $(\exists\ n.\ \sigma \in SPower\ (lan\ f)\ n) =$   
 $(\exists\ n.\ \sigma \in lan\ (LIFT(power\ (f \wedge more)\ n)))$   
**using** *power-chop-power* **by** *blast*  
**have** 3:  $(\exists\ n.\ \sigma \in lan\ (LIFT(power\ (f \wedge more)\ n))) =$   
 $(\exists\ n.\ (LIFT(power\ (f \wedge more)\ n))\ \sigma)$   
**using** *interval-lan* **by** *simp*  
**have** 4:  $(\exists\ n.\ (LIFT(power\ (f \wedge more)\ n))\ \sigma) =$   
 $(\sigma \in (lan\ (LIFT(f^*))))$   
**by** (*simp add: chopstar-d-def powerstar-d-def*)  
**show** ?thesis **by** (*simp add: 1 2 4*)  
**qed**

**lemma** *chopstar-sstar-1*:

$(SStar\ (lan\ f)) = (lan\ (LIFT(f^*)))$   
**using** *sstar-chopstar lan-def* **by** *blast*

**lemma** *chopstar-seqv*:

$\sigma \in (lan\ (LIFT(f^*))) \longleftrightarrow$   
 $\sigma \in (lan\ (LIFT(empty \vee (f \wedge more); f^*)))$   
**using** *ChopstarEqv* **by** *fastforce*

**lemma** *chopstar-seqv-1*:

$(lan\ (LIFT(f^*))) = (lan\ (LIFT(empty \vee (f \wedge more); f^*)))$   
**using** *chopstar-seqv lan-def* **by** *blast*

**lemma** *srev-irev*:

$\sigma \in (SRev\ (lan\ f)) \longleftrightarrow \sigma \in (lan\ (LIFT(f^r)))$   
**by** (*simp add: reverse-d-def srev-elim*)

**lemma** *srev-irev-1*:

$(SRev\ (lan\ f)) = (lan\ (LIFT(f^r)))$   
**using** *srev-irev lan-def* **by** *blast*

**end**

## 17 Executability of ITL formulae

**theory** *Executability*

**imports** *Semantics TimeReversal Fuse*

**begin**

In this section we will discuss the notion of executability. It is used to determine whether an ITL formula represents a “programming construct”. We first formalise the notion of forward executability of a formula which corresponds to generating a sequence of states in a particular fashion: we first generate the first state and then generate the next until the final state is generated. This sequence of state constitutes the behaviour of the system described by the formula. We then investigate the reflection of forward executable formula and this requires the introduction of the notion of backward executability. This notion corresponds to generating a sequence of states but now we first generate the final state and then generate the previous state until we generate the first state. This sequence corresponds to the reversed behaviour of the system described by the formula. Forward and backward executability are related by the reflection operator.

## 17.1 Forward Executability

### 17.1.1 Common Prefix Value Trace Definitions

**definition**  $\text{sat-d} :: ('a :: \text{world}) \text{ formula} \Rightarrow \text{bool}$   
**where**  $\text{sat-d } F \equiv (\exists s. s \models F)$

The following definition is a constraint on the intervals which satisfy a formula. Only intervals that share a common prefix of the value trace are allowed

**definition**  $\text{commonpfx} :: ('a :: \text{world}) \text{ formula} \Rightarrow ('a, 'b) \text{ stfun} \Rightarrow \text{bool}$   
**where**  $\text{commonpfx } F v \equiv$   
 $(\forall s1 s. (s1 \models F) \wedge (s \models F) \wedge \text{ilen } s \leq \text{ilen } s1 \longrightarrow$   
 $\quad \text{imap } v s = (\text{imap } v (\text{prefix } (\text{ilen } s) s1))$   
 $)$

The following definition introduces the notion of forward executability.

**definition**  $\text{fexec} :: ('a :: \text{world}) \text{ formula} \Rightarrow ('a, 'b) \text{ stfun} \Rightarrow \text{bool}$   
**where**  $\text{fexec } F v \equiv ( (\text{sat-d } F) \wedge (\text{commonpfx } F v) )$

**syntax**

$\text{-sat} \quad :: \text{lift} \Rightarrow \text{bool} \quad ((\text{sat } (-)) \ 5)$   
 $\text{-commonpfx} \quad :: \text{lift} \Rightarrow \text{lift} \Rightarrow \text{bool} \quad (((\dagger[-]')(-)) \ 5)$   
 $\text{-fexec} \quad :: \text{lift} \Rightarrow \text{lift} \Rightarrow \text{bool} \quad (((\dagger[-]')(-)) \ 5)$

**translations**

$\text{-sat} \quad == \text{CONST sat-d}$   
 $\text{-commonpfx} == \text{CONST commonpfx}$   
 $\text{-fexec} \quad == \text{CONST fexec}$

### 17.1.2 Semantic lemmas

**lemma**  $\text{imap-eq-inth-ilen}$ :

$(\text{imap } v s) = (\text{imap } v s1) \longleftrightarrow \text{ilen } s = \text{ilen } s1 \wedge (\forall i. i \leq \text{ilen } s \longrightarrow v (\text{inth } s i) = v (\text{inth } s1 i))$   
**using**  $\text{interval-eq-inth-eq[of } (\text{imap } v s) (\text{imap } v s1)]$   
**by**  $(\text{simp add: inth-imap})$

**lemma**  $\text{interval-prefix-suffix-help}$ :

$n \leq \text{ilen } s1 \implies (\text{suffix } (\text{min } n (\text{ilen } s1)) (\text{prefix } n s1)) = \langle \text{inth } s1 n \rangle$   
**unfolding**  $\text{min-def}$



**by** (*metis* (*full-types*) *ilast-prefix prefix-ilen-good suffix-ilast*)

**lemma** *ExistsilenNhlp*:  $\exists sa. \text{ilen } sa = \text{Suc } (\text{ilen } s)$

**by** (*metis* *Suc-eq-plus1-left ilen-imap ilen.simps(2)*)

**lemma** *ExistsilenN*:

$\exists s. \text{ilen } s = n$

**proof** (*induct*  $n$ )

**case**  $0$

**then show** *?case* **by** (*meson* *ilen.simps(1)*)

**next**

**case** (*Suc*  $n$ )

**then show** *?case* **using** *ExistsilenNhlp* **by** *blast*

**qed**

**lemma** *unithelp1*:

$(v (\text{inth } s1\ 0) = c \vee v (\text{inth } s1\ 0) = d) \wedge \text{ilen } s1 = 0$

$\implies (\text{imap } v\ s1) = (\text{INil } c) \vee (\text{imap } v\ s1) = (\text{INil } d)$

**by** (*simp* *add: INil-ilen ifirst-imap*)

**lemma** *unithelp2*:

$v (\text{inth } (\text{INil } a)\ 0) = c \vee v (\text{inth } (\text{INil } a)\ 0) = d \implies$

$(\text{imap } v\ (\text{INil } a)) = (\text{INil } c) \vee (\text{imap } v\ (\text{INil } a)) = (\text{INil } d)$

**by** *auto*

**lemma** *skiphelp1*:

$v (\text{inth } s1\ 0) = c \wedge \text{ilen } s1 = 1 \wedge v (\text{inth } s1\ (\text{Suc } 0)) = d \implies (\text{imap } v\ s1) = (\text{ICons } c\ (\text{INil } d))$

**by** (*metis* *One-nat-def interval-hd-tail ilen-imap inth-imap suffix-ilast less-numeral-extra(1)*)

**lemma** *skipnoteqnext*:

$v (\text{inth } s1\ 0) = c \wedge \text{ilen } s1 = (\text{Suc } n) \wedge v (\text{inth } s1\ (\text{Suc } 0)) = d \wedge$

$v (\text{inth } s\ 0) = c \wedge \text{ilen } s = (\text{Suc } n) \wedge v (\text{inth } s\ (\text{Suc } 0)) = d1 \wedge d \neq d1 \wedge s \neq s1 \implies$

$(\text{imap } v\ s1) \neq (\text{imap } v\ s)$

**by** (*metis* *inth-imap*)

**lemma** *skipnoteqfirst*:

$v (\text{inth } s1\ 0) = c \wedge \text{ilen } s1 = (\text{Suc } n) \wedge v (\text{inth } s1\ (\text{Suc } 0)) = d \wedge$

$v (\text{inth } s\ 0) = c1 \wedge \text{ilen } s = (\text{Suc } n) \wedge v (\text{inth } s\ (\text{Suc } 0)) = d \wedge c \neq c1 \wedge s \neq s1 \implies$

$(\text{imap } v\ s1) \neq (\text{imap } v\ s)$

**by** (*metis* *inth-imap*)

**lemma** *skipnoteqfirstnext*:

$v (\text{inth } s1\ 0) = c \wedge \text{ilen } s1 = (\text{Suc } n) \wedge v (\text{inth } s1\ (\text{Suc } 0)) = d \wedge$

$v (\text{inth } s\ 0) = c1 \wedge \text{ilen } s = (\text{Suc } n) \wedge v (\text{inth } s\ (\text{Suc } 0)) = d1 \wedge c \neq c1 \wedge d \neq d1 \wedge s \neq s1 \implies$

$(\text{imap } v\ s1) \neq (\text{imap } v\ s)$

**by** (*metis* *inth-imap*)

**lemma** *skiphelp2*:

$v (\text{inth } s1\ 0) = c \wedge \text{ilen } s1 = 1 \implies (\text{imap } v\ s1) = (\text{ICons } c\ (\text{INil } (v (\text{inth } s1\ 1))))$

**by** (*metis* *interval-hd-tail ilen-imap inth-imap suffix-ilast less-numeral-extra(1)*)

**lemma lenkhelp3:**

*surj v*  $\implies \exists s. v \text{ (inth } s \ 0) = c \wedge \text{ilen } s = n$

**by** (*metis ExistsilenN inth.simps(1) Suc-eq-plus1-left inth-zero ilen.simps(1) ilen.simps(2) not0-implies-Suc surjD*)

**lemma lenkhelp4:**

*surj v*  $\implies 0 < n \implies \exists s. v \text{ (inth } s \ 0) = c \wedge v \text{ (inth } s \ 1) = d \wedge \text{ilen } s = n$

**by** (*metis One-nat-def inth-Suc inth-zero ilen.simps(2) lenkhelp3 less-numeral-extra(3) not0-implies-Suc plus-1-eq-Suc*)

**lemma lenkhelp1:**

$v \text{ (inth } s1 \ 0) = c \wedge \text{ilen } s1 = n \wedge v \text{ (inth } s1 \ (\text{Suc } 0)) = d \wedge n = 1 \implies$

$(\text{imap } v \ s1) = (\text{ICons } c \ (\text{INil } d))$

**by** (*metis One-nat-def skiphelp2*)

**lemma len0help:**

**assumes** *surj v*

**shows**  $(\exists s. v \text{ (inth } s \ 0) = c \wedge \text{ilen } s = 0 \wedge (\forall i. i \leq \text{ilen } s \longrightarrow v \text{ (inth } s \ i) = v \text{ (inth } s \ 0)))$

**proof** –

**have** *0*: *surj v*  $\implies$

$(\exists s. v \text{ (inth } s \ 0) = c \wedge \text{ilen } s = 0 \wedge (\forall i. i \leq \text{ilen } s \longrightarrow v \text{ (inth } s \ i) = v \text{ (inth } s \ 0))) =$

$(\exists s. v \text{ (inth } s \ 0) = c \wedge \text{ilen } s = (\text{Suc } 0) \wedge v \text{ (inth } s \ 0) = v \text{ (inth } s \ 0) )$

**using** *lenkhelp3[of v c]* **by** (*metis inth.simps(1) ilen.simps(1)*)

**have** *01*: *surj v*  $\implies (\exists s. v \text{ (inth } s \ 0) = c \wedge \text{ilen } s = (\text{Suc } 0) \wedge v \text{ (inth } s \ 0) = v \text{ (inth } s \ 0) )$

**by** (*simp add: lenkhelp3*)

**show** *?thesis* **using** *0 01 assms* **by** *auto*

**qed**

**lemma getsmorehelp:**

**assumes**  $\forall i < \text{ilen } s. v \text{ (inth } s \ (\text{Suc } i)) = \text{Suc } (v \text{ (inth } s \ i))$

$v \text{ (ifirst } s) = 0$

$j \leq \text{ilen } s$

**shows**  $\text{inth } (\text{imap } v \ s) \ j = j$

**using** *assms*

**proof** (*induct j arbitrary: s*)

**case** *0*

**then show** *?case* **by** (*simp add: ifirst-imap*)

**next**

**case** (*Suc j*)

**then show** *?case* **by** (*simp add: Suc.premis(1) Suc.premis(3) Suc-leD Suc-le-lessD inth-imap*)

**qed**

**lemma initonlyfinhelp:**

**assumes**  $n \leq \text{ilen } s1$

**shows**  $(\forall na \leq \text{ilen } s1 - n. (na = 0) = w \text{ (suffix } na \text{ (prefix } na \text{ (suffix } n \ s1)))) =$

$(w \langle \text{inth } s1 \ n \rangle \wedge (\forall k. n < k \wedge k \leq \text{ilen } s1 \longrightarrow \neg w \langle \text{inth } s1 \ k \rangle))$

**proof** –

**have** *1*:  $(\forall na \leq \text{ilen } s1 - n. (na = 0) = w \text{ (suffix } na \text{ (prefix } na \text{ (suffix } n \ s1)))) =$

$(\forall na . na+n \leq \text{ilen } s1 \longrightarrow (na = 0) = w (\text{suffix } na (\text{prefix } na (\text{suffix } n s1))))$   
**by** (*simp add: Nat.le-diff-conv2 assms*)  
**have** 2:  $(\forall na . na+n \leq \text{ilen } s1 \longrightarrow (na = 0) = w (\text{suffix } na (\text{prefix } na (\text{suffix } n s1)))) =$   
 $(\forall na . na+n \leq \text{ilen } s1 \longrightarrow (na = 0) = w (\text{suffix } na (\text{suffix } n (\text{prefix } (na+n) s1))))$   
**by** (*simp add: suffix-prefix-swap*)  
**have** 3:  $(\forall na . na+n \leq \text{ilen } s1 \longrightarrow (na = 0) = w (\text{suffix } na (\text{suffix } n (\text{prefix } (na+n) s1)))) =$   
 $(\forall na . na+n \leq \text{ilen } s1 \longrightarrow (na = 0) = w (\text{suffix } (na+n) (\text{prefix } (na+n) s1)))$   
**by** *simp*  
**have** 4:  $(\forall na . na+n \leq \text{ilen } s1 \longrightarrow (na = 0) = w (\text{suffix } (na+n) (\text{prefix } (na+n) s1))) =$   
 $(\forall na . na+n \leq \text{ilen } s1 \longrightarrow (na = 0) = w \langle \text{inth } s1 (na+n) \rangle)$   
**by** (*metis interval-prefix-suffix-help min.orderE*)  
**have** 5:  $(\forall na . na+n \leq \text{ilen } s1 \longrightarrow (na = 0) = w \langle \text{inth } s1 (na+n) \rangle) =$   
 $(\forall k. n \leq k \wedge k \leq \text{ilen } s1 \longrightarrow (k-n=0) = w \langle \text{inth } s1 k \rangle)$   
**by** (*metis add-diff-cancel-right' le-add2 le-add-diff-inverse2*)  
**have** 6:  $(\forall k. n \leq k \wedge k \leq \text{ilen } s1 \longrightarrow (k-n=0) = w \langle \text{inth } s1 k \rangle) =$   
 $(w \langle \text{inth } s1 n \rangle \wedge (\forall k. n < k \wedge k \leq \text{ilen } s1 \longrightarrow \neg w \langle \text{inth } s1 k \rangle))$   
**using** *assms le0 le-eq-less-or-eq less-numeral-extra(3)* **by** *auto*  
**show** *?thesis* **using** 1 2 4 5 6 **by** *auto*  
**qed**

### 17.1.3 Common prefix value trace theorems

**lemma** *commonpfx-symmetric:*

$(\forall s1 s. (s1 \models F) \wedge (s \models F) \wedge \text{ilen } s \leq \text{ilen } s1 \longrightarrow$   
 $\text{imap } v s = (\text{imap } v (\text{prefix } (\text{ilen } s) s1))$   
 $) =$   
 $(\forall s1 s. (s1 \models F) \wedge (s \models F) \longrightarrow$   
 $(\text{ilen } s \leq \text{ilen } s1 \longrightarrow \text{imap } v s = (\text{imap } v (\text{prefix } (\text{ilen } s) s1))) \wedge$   
 $(\text{ilen } s1 \leq \text{ilen } s \longrightarrow \text{imap } v s1 = (\text{imap } v (\text{prefix } (\text{ilen } s1) s)))$   
 $)$

**by** *auto*

**lemma** *CommonPfxFixedTrace:*

**assumes** *basevars v*  
 $\text{sat } f \wedge \text{len } k$   
 $\ddagger[f]-v$   
**shows**  $\text{card } \{ (\text{imap } v s) \mid s. (s \models f) \wedge \text{ilen } s = k \} = 1$   
**proof** –  
**obtain** *s0* **where** 1:  $(s0 \models f) \wedge \text{ilen } s0 = k$   
**by** (*metis (mono-tags, lifting) assms(2) len-defs sat-d-def unl-lift2*)  
**have** 3:  $\forall s1. (s1 \models f) \wedge \text{ilen } s1 = k \longrightarrow (\text{imap } v s1) = (\text{imap } v s0)$   
**by** (*metis 1 assms(3) le-cases commonpfx-def*)  
**have** 4:  $\{ (\text{imap } v s) \mid s. (s \models f) \wedge \text{ilen } s = k \} = \{ (\text{imap } v s0) \}$   
**using** 3 1 **by** *blast*  
**show** *?thesis* **by** (*simp add: 4*)  
**qed**

**lemma** *NoTrace:*

**assumes** *basevars v*

```

       $\neg(\text{sat } f \wedge \text{len } k)$ 
shows  $\text{card } \{ ( \text{imap } v \ s ) \mid s. (s \models f) \wedge \text{ilen } s = k \} = 0$ 
proof –
  have 1:  $\forall s. \neg(s \models f) \vee \text{ilen } s \neq k$ 
    using assms unfolding sat-d-def len-defs by auto
  have 2:  $\{ ( \text{imap } v \ s ) \mid s. (s \models f) \wedge \text{ilen } s = k \} = \{ \}$ 
    using 1 by blast
  show ?thesis by (simp add: 2)
qed

```

The following theorem states that the combination of satisfiability with the notion of common prefix value trace can be used to determine whether a formula is executable or not, i.e., satisfiable and deterministic.

**lemma** *CommonPfxTrace*:

```

assumes basevars v
       $\text{sat } f$ 
       $\ddagger[f]\text{-}v$ 
shows  $\text{card } \{ ( \text{imap } v \ s ) \mid s. (s \models f) \wedge \text{ilen } s = k \} \leq 1$ 
proof –
  have 1:  $(\text{sat } f) = (\text{sat } (\exists k. f \wedge \text{len } k))$ 
    by (simp add: len-defs sat-d-def)
  have 2:  $\bigwedge k. (\text{sat } f \wedge \text{len } k) \vee \neg(\text{sat } f \wedge \text{len } k)$ 
    by simp
  have 3:  $\bigwedge k. \neg(\text{sat } f \wedge \text{len } k) \implies \text{card } \{ \text{imap } v \ s \mid s. f \ s \wedge \text{ilen } s = k \} \leq 1$ 
    using assms NoTrace[of v f] by auto
  have 4:  $\bigwedge k. \text{sat } f \wedge \text{len } k \implies \text{card } \{ \text{imap } v \ s \mid s. f \ s \wedge \text{ilen } s = k \} \leq 1$ 
    using assms CommonPfxFixedTrace[of v f] by auto
  show ?thesis using 2 3 4 by auto
qed

```

**lemma** *PairCommonPfx*:

```

assumes basevars (v1, v2)
       $\ddagger[f]\text{-}v1$ 
       $\ddagger[f]\text{-}v2$ 
shows  $\ddagger[f]\text{-(}v1, v2\text{)}$ 
using assms unfolding commonpfx-def
by (simp add: imap-eq-inth-ilen)

```

**lemma** *ProjCommonPfxA*:

```

assumes basevars (v1, v2)
       $\ddagger[f]\text{-(}v1, v2\text{)}$ 
shows  $\ddagger[f]\text{-}v1$ 
using assms unfolding commonpfx-def
by (simp add: imap-eq-inth-ilen)

```

**lemma** *ProjCommonPfxB*:

```

assumes basevars (v1, v2)
       $\ddagger[f]\text{-(}v1, v2\text{)}$ 
shows  $\ddagger[f]\text{-}v2$ 
using assms unfolding commonpfx-def

```

by (simp add: imap-eq-inth-ilen)

**lemma** *PairSplitCommonPfx*:

**assumes** basevars (v1, v2)

**shows**  $(\ddagger[f]-(v1, v2)) = ((\ddagger[f]-v1) \wedge (\ddagger[f]-v2))$

**using** *PairCommonPfx ProjCommonPfxA ProjCommonPfxB assms* **by** blast

#### 17.1.4 Common prefix value trace and len and assignment

**lemma** *SatLenK*:

*sat (len k)*

**by** (simp add: *Valid-def itl-defs sat-d-def ExistsilenN*)

**lemma** *NotCommonPfxLenK*:

**assumes** basevars (v :: state  $\Rightarrow$  nat)

**shows**  $\neg(\ddagger[\text{len}(k)]-v)$

**proof** –

**have** 1:  $\exists s. v (\text{ifirst } s) = 0 \wedge \text{ilen } s = k$

**by** (meson assms basevars-def lenkhelphelp3)

**have** 2:  $\exists s. v (\text{ifirst } s) = 1 \wedge \text{ilen } s = k$

**by** (meson assms basevars-def lenkhelphelp3)

**have** 3:  $\neg(\ddagger[\text{len } k]-v)$

**using** assms 1 2 **unfolding** *commonpfx-def len-defs*

**by** (metis *inth-imap prefix-ilen imap-prefix order.order-iff-strict zero-neq-one*)

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

**qed**

**lemma** *NotCommonPfxUnitAssignOrAndLenK*:

**assumes** basevars (v :: state  $\Rightarrow$  nat)

**shows**  $\neg(\ddagger[(\$v = \#0 \vee \$v = \#1) \wedge \text{len } k]-v)$

**using** assms lenkhelphelp3[of v 0 k] lenkhelphelp3[of v 1 k]

**by** ( simp add: *commonpfx-def itl-defs basevars-def inth-imap* )

(metis *One-nat-def inth-imap prefix-ilen-bound prefix-ilen zero-neq-one*)

**lemma** *CommonPfxUnitAssignAndNextValAndSkip*:

**assumes** basevars (v :: state  $\Rightarrow$  nat)

**shows**  $(\ddagger[\$v = \#c \wedge \text{skip} \wedge v\$ = \#d]-v)$

**proof** –

**have** 2: *commonpfx* (*LIFT* ( $\$v = \#c \wedge \text{skip} \wedge v\$ = \#d$ )) v

**using** assms lenkhelphelp1[of v - c 1 d]

**by** (simp add: *commonpfx-def next-val-d-def itl-defs basevars-def*)

**show** ?thesis **using** 2 **by** auto

**qed**

**lemma** *NotCommonPfxUnitAssignAndLenK*:

**assumes** basevars (v :: state  $\Rightarrow$  nat)

$0 < k$

**shows**  $\neg(\ddagger[\$v = \#0 \wedge \text{len } k]-v)$

**using** assms **using** lenkhelphelp4[of v k 0 0] lenkhelphelp4[of v k 0 1]

**by** (simp add: *commonpfx-def next-val-d-def itl-defs basevars-def*)

(metis One-nat-def inth-imap prefix-ilen prefix-ilen-bound zero-neq-one)

**lemma** *CommonPfxStableAndLenK*:

**assumes** *basevars v*

**shows**  $(\dagger[\$v = \#(c::nat) \wedge len\ k \wedge stable\ v] - v)$

**using** *assms* **by** (*simp add: commonpfx-def itl-defs basevars-def*)  
(metis prefix-ilen imap-eq-inth-ilen)

### 17.1.5 Tempura

**lemma** *HaltAlways*:

$\vdash (halt\ f) = \Box(if_i\ f\ then\ empty\ else\ (\neg empty))$

**proof** –

**have** 1:  $\vdash (empty=f) = (if_i\ f\ then\ empty\ else\ (\neg empty))$

**unfolding** *ifthenelse-d-def* **by** *auto*

**have** 2:  $\vdash \Box(empty=f) = \Box(if_i\ f\ then\ empty\ else\ (\neg empty))$

**using** 1 *BoxEqvBox[of LIFT(empty=f) LIFT(if\_i f then empty else (¬empty))]* **by** *auto*

**show** *?thesis* **unfolding** *halt-d-def* **using** 2 **by** *auto*

**qed**

**lemma** *ChopEqvExistAndLenKChop*:

$\vdash f;g = (\exists k. (f \wedge len\ k);g)$

**by** (*simp add: Valid-def itl-defs*)

**lemma** *ChopEqvExistChopAndLenK*:

$\vdash f;g = (\exists k. f;(g \wedge len\ k))$

**by** (*simp add: Valid-def itl-defs*)

**lemma** *BiSkipImpEqv* :

$\vdash bi\ (skip \longrightarrow f) = (more \longrightarrow (f \wedge skip));\#True$

**by** (*auto simp add: Valid-def itl-defs itl-def min-def*)

**lemma** *GetsAlways*:

$\vdash a\ gets\ b = \Box ( bi\ (skip \longrightarrow a\$ = b) )$

**by** (*simp add: Valid-def itl-def*)

**lemma** *GetsAlwaysBox*:

$\vdash a\ gets\ b = \Box ( more \longrightarrow (a\$ = b \wedge skip);\#True)$

**by** (*metis BiSkipImpEqv GetsAlways int-eq*)

**lemma**  $\vdash bm\ (a\$ = b) = \Box ( more \longrightarrow a\$ = b)$

**by** (*simp add: Valid-def itl-defs itl-def*)

**lemma**  $\vdash (a\ gets\ \$a + \#(1::nat)) = (\Box ( more \longrightarrow a\$ = \$a + \#1))$

**by** (*auto simp add: Valid-def itl-defs*)

### 17.1.6 Basic theorems

**lemma** *CommonPfxEqv*:

**assumes** *basevars v*

$\vdash f = g$   
**shows**  $(\ddagger[f]-v) = (\ddagger[g]-v)$   
**using** *assms inteq-reflection* **by** *blast*

**lemma** *CommonPfxAnd*:  
**assumes** *basevars v*  
 $\ddagger[f]-v$   
 $\ddagger[g]-v$   
**shows**  $\ddagger[f \wedge g]-v$   
**using** *assms unfolding commonpfx-def* **by** *simp*

**lemma** *CommonPfxEmptyOrMore*:  
**assumes** *basevars v*  
**shows**  $(\ddagger[f]-v) = (\ddagger[(f \wedge \text{empty}) \vee (f \wedge \text{more})]-v)$   
**proof** –  
**have** *1*:  $\vdash f = ((f \wedge \text{empty}) \vee (f \wedge \text{more}))$   
**unfolding** *empty-d-def* **by** *fastforce*  
**show** *?thesis* **using** *CommonPfxEqv 1 assms* **by** *blast*  
**qed**

**lemma** *NotCommonPfxEmpty*:  
**assumes** *basevars (v :: state  $\Rightarrow$  nat)*  
**shows**  $\neg(\ddagger[\text{empty}]-v)$   
**proof** –  
**obtain** *s0* **where** *2*:  $v (\text{ifirst } s0) = 0 \wedge \text{ilen } s0 = 0$   
**by** (*metis inth.simps(1) assms baseE ilen.simps(1)*)  
**obtain** *s1* **where** *3*:  $v (\text{ifirst } s1) = 1 \wedge \text{ilen } s1 = 0$   
**by** (*metis inth.simps(1) assms baseE ilen.simps(1)*)  
**show** *?thesis* **unfolding** *commonpfx-def empty-defs*  
**by** (*metis 2 3 One-nat-def inth-imap n-not-Suc-n order.order-iff-strict*)  
**qed**

**lemma** *NotCommonPfxMore*:  
**assumes** *basevars (v :: state  $\Rightarrow$  nat)*  
**shows**  $\neg(\ddagger[\text{more}]-v)$   
**proof** –  
**obtain** *s0* **where** *2*:  $v (\text{ifirst } s0) = 0 \wedge \text{ilen } s0 = 1$   
**by** (*metis One-nat-def assms baseE inth-zero ilen.simps(1) ilen.simps(2) plus-1-eq-Suc*)  
**obtain** *s1* **where** *3*:  $v (\text{ifirst } s1) = 1 \wedge \text{ilen } s1 = 1$   
**by** (*metis One-nat-def assms baseE inth-zero ilen.simps(1) ilen.simps(2) plus-1-eq-Suc*)  
**show** *?thesis* **unfolding** *commonpfx-def more-defs*  
**by** (*metis 2 3 One-nat-def ifirst-imap lessI n-not-Suc-n order-refl*)  
**qed**

**lemma** *NotCommonPfxTrue*:  
**assumes** *basevars (v :: state  $\Rightarrow$  nat)*  
**shows**  $\neg(\ddagger[\# \text{True}]-v)$   
**using** *assms* **by** (*metis NotCommonPfxEmpty commonpfx-def unl-con*)

**lemma** *CommonPfxChoice*:  
**assumes** *basevars v*  
**shows**  $(\ddagger[f]-v) = (\ddagger[(f \wedge g) \vee (f \wedge \neg g)]-v)$   
**proof** –  
**have**  $1: \vdash f = ((f \wedge g) \vee (f \wedge \neg g))$   
**by** *fastforce*  
**show** *?thesis* **using** *CommonPfxEqv 1 assms* **by** *blast*  
**qed**

**lemma** *CommonPfxDetChoice*:  
**assumes** *basevars v*  
 $\ddagger[(f \wedge g) \vee (f \wedge \neg g)]-v$   
**shows**  $(\ddagger[f \wedge g]-v) \wedge (\ddagger[(f \wedge \neg g)]-v)$   
**using** *assms* **unfolding** *commonpfx-def* **by** *simp*

**lemma** *CommonPfxBox*:  
**assumes** *basevars v*  
**shows**  $(\ddagger[\Box f]-v) = (\ddagger[f \wedge \text{wnext } (\Box f)]-v)$   
**proof** –  
**have**  $1: \vdash \Box f = (f \wedge \text{wnext } (\Box f))$   
**by** (*simp add: BoxEqvAndWnextBox*)  
**show** *?thesis* **using** *assms 1 CommonPfxEqv[of v LIFT( $\Box f$ ) LIFT( $f \wedge \text{wnext } (\Box f)$ )]* **by** *simp*  
**qed**

**lemma** *CommonPfxBi*:  
**assumes** *basevars v*  
 $\ddagger[f]-v$   
**shows**  $\ddagger[bi\ f]-v$   
**using** *assms* **by** (*simp add: commonpfx-def bi-defs*) *force*

**lemma** *CommonPfxBox1*:  
**assumes** *basevars v*  
 $\ddagger[f]-v$   
**shows**  $\ddagger[\Box f]-v$   
**using** *assms* **by** (*simp add: commonpfx-def itl-defs*) *force*

**lemma** *CommonPfxAssign*:  
**assumes** *basevars (v :: state  $\Rightarrow$  nat)*  
**shows**  $\ddagger[\$v = \#c \wedge v := \#d \wedge \text{skip}]-v$   
**proof** –  
**have**  $1: \vdash (\$v = \#c \wedge v := \#d \wedge \text{skip}) = (\$v = \#c \wedge \text{skip} \wedge v\$ = \#d)$   
**unfolding** *next-assign-d-def* **by** *fastforce*  
**have**  $2: \ddagger[\$v = \#c \wedge \text{skip} \wedge v\$ = \#d]-v$   
**using** *CommonPfxUnitAssignAndNextValAndSkip[of v c d] assms* **by** *blast*  
**show** *?thesis* **using** *assms 1 2*  
 $\text{CommonPfxEqv}[of\ v\ \text{LIFT}(\$v = \#c \wedge v := \#d \wedge \text{skip})\ \text{LIFT}(\$v = \#c \wedge \text{skip} \wedge v\$ = \#d)]$   
**by** *blast*  
**qed**

**lemma** *SatValid*:



**assumes**  $\vdash f$   
**shows**  $\text{sat } f$   
**by** (*simp add: assms intD sat-d-def*)

**lemma** *SatValidNotA*:  
 $(\vdash f) = (\neg(\text{sat } \neg f))$   
**unfolding** *sat-d-def Valid-def* **by** *auto*

**lemma** *SatValidNotB*:  
 $(\text{sat } f) = (\neg(\vdash \neg f))$   
**unfolding** *sat-d-def Valid-def* **by** *auto*

**lemma** *ValidAnd*:  
 $(\vdash f \wedge g) = ((\vdash f) \wedge (\vdash g))$   
**unfolding** *Valid-def* **by** *auto*

**lemma** *ValidOrA*:  
**assumes**  $\vdash f$   
**shows**  $\vdash f \vee g$   
**using** *assms* **unfolding** *Valid-def* **by** *auto*

**lemma** *ValidOrB*:  
**assumes**  $\vdash g$   
**shows**  $\vdash f \vee g$   
**using** *assms* **unfolding** *Valid-def* **by** *auto*

**lemma** *SatAnd*:  
**assumes**  $\text{sat } f \wedge g$   
**shows**  $(\text{sat } f) \wedge (\text{sat } g)$   
**using** *assms*  
**by** (*metis sat-d-def unl-lift2*)

**lemma** *SatOr*:  
 $(\text{sat } (f \vee g)) = ((\text{sat } f) \vee (\text{sat } g))$   
**unfolding** *sat-d-def* **by** *auto*

**lemma** *SatAndLenK*:  
**assumes**  $\text{sat } (f \wedge \text{len } k)$   
**shows**  $\text{sat } f$   
**using** *assms SatAnd* **by** *blast*

**lemma** *SatChopHaltPrefix*:  
**assumes**  $\text{sat } (f \wedge \text{halt } w)$   
 $\text{sat } g$   
 $\vdash \text{init}(w) \longrightarrow g$   
**shows**  $\text{sat } (f \wedge \text{halt } w);g$   
**proof** –  
**have** *1*:  $\exists s. (s \models f) \wedge (\forall n \leq \text{ilen } s. (\text{ilen } s = n) = w (\text{suffix } n s))$   
**by** (*metis (mono-tags, lifting) assms(1) halt-defs sat-d-def unl-lift2*)  
**obtain** *s* **where** *2*:  $(s \models f) \wedge (\forall n \leq \text{ilen } s. (\text{ilen } s = n) = w (\text{suffix } n s))$

```

  using 1 by auto
  have 3:  $w \langle \text{inth } s \text{ (ilen } s) \rangle$ 
  using 2 by auto
  obtain s1 where 4:  $(s1 \models g) \wedge (\text{inth } s1 \ 0) = \text{inth } s \text{ (ilen } s)$ 
  by (metis 3 inth.simps(1) Prop10 assms(3) init-defs inteq-reflection prefix-zero-ifirst
      unl-lift2)
  have 5:  $(\text{fuse } s \ s1) \models (f \wedge \text{halt } w);g$ 
  by (metis 2 4 halt-defs chop-fuse unl-lift2)
  show ?thesis using 5 sat-d-def by auto
qed

```

**lemma** *SatChopHaltInitonly*:

```

  assumes sat f  $\wedge$  halt w
          sat g
           $\vdash \text{initonly}(w) \longrightarrow g$ 
  shows sat (f  $\wedge$  halt w);(g  $\wedge$  initonly w)
proof -
  have 1:  $\exists s. (s \models f) \wedge (\forall n \leq \text{ilen } s. (\text{ilen } s = n) = w \text{ (suffix } n \ s))$ 
  by (metis (mono-tags, lifting) assms(1) halt-defs sat-d-def unl-lift2)
  obtain s where 2:  $(s \models f) \wedge (\forall n \leq \text{ilen } s. (\text{ilen } s = n) = w \text{ (suffix } n \ s))$ 
  using 1 by auto
  have 3:  $w \langle \text{inth } s \text{ (ilen } s) \rangle$ 
  using 2 by auto
  obtain s1 where 5:  $(s1 \models g) \wedge (\forall n \leq \text{ilen } s1. (n=0) = w \text{ (prefix } n \ s1)) \wedge \text{ifirst } s1 = \text{ilast } s$ 
  using assms by (simp add: itl-defs Valid-def sat-d-def)
  (metis 3 ifirst-suffix ilen.simps(1) le-zero-eq nat-le-linear prefix-ilen suffix-ilen-last)
  have 6:  $w \langle \text{inth } s1 \ 0 \rangle$ 
  using 5 by auto
  have 7:  $(\text{fuse } s \ s1) \models (f \wedge \text{halt } w);(g \wedge \text{initonly } w)$ 
  using 2 5 3 6 by (auto simp add: itl-defs chop-fuse min-def)
  (metis fuse-ilen-a le-add1 prefix-fuse suffix-fuse suffix-ilen)
  show ?thesis using 7 unfolding sat-d-def by auto
qed

```

**lemma** *SatChopL*:

```

  assumes sat f
          sat g
           $\vdash f \longrightarrow \text{fin } w$ 
           $\vdash \text{init}(w) \longrightarrow g$ 
  shows sat f;g
using assms
proof -
  have 1:  $\exists s. (s \models f)$ 
  by (metis (mono-tags, lifting) assms(1) sat-d-def)
  obtain s where 2:  $(s \models f)$ 
  using 1 by auto
  have 3:  $w \langle \text{inth } s \text{ (ilen } s) \rangle$ 
  using 2 assms by (auto simp add: Valid-def fin-defs init-defs)
  obtain s1 where 4:  $(s1 \models g) \wedge (\text{inth } s1 \ 0) = \text{inth } s \text{ (ilen } s)$ 
  by (metis 3 inth.simps(1) Prop10 assms(4) init-defs inteq-reflection prefix-zero-ifirst

```

```

    unl-lift2)
  have 5: (fuse s s1)  $\models$  (f );g
    by (metis 2 4 chop-fuse )
  show ?thesis using 5 sat-d-def by auto
qed

lemma SatChopR:
  assumes sat f
    sat g
     $\vdash$  fin w  $\longrightarrow$  f
     $\vdash$  g  $\longrightarrow$  init(w)
  shows sat f;g
proof -
  have 1:  $\exists$  s. (s  $\models$  g)
    by (metis (mono-tags, lifting) assms(2) sat-d-def )
  obtain s where 2: (s  $\models$  g)
    using 1 by auto
  have 3: w  $\langle$  inth s 0  $\rangle$ 
    using 2 assms by (simp add: Valid-def init-defs)
  obtain s1 where 4: (s1  $\models$  f)  $\wedge$  (inth s1 (ilen s1)) = (inth s 0)
    using 3 assms(3) by (simp add: Valid-def fin-defs) (metis ilast-irev)
  have 5: (fuse s1 s)  $\models$  f;g
    by (meson 2 4 chop-fuse)
  show ?thesis using 5 sat-d-def by auto
qed

```

```

lemma SatChopExistsL:
  (sat f;g) = (sat ( $\exists$  k. (f  $\wedge$  len k);g))
by (metis ChopEqvExistAndLenKChop inteq-reflection)

```

```

lemma SatChopExistsR:
  (sat f;g) = (sat ( $\exists$  k. f;(g  $\wedge$  len k)))
by (metis ChopEqvExistChopAndLenK inteq-reflection)

```

```

lemma SatExistsChopL:
  (sat f;g) = ( $\exists$  k. (sat (f  $\wedge$  len k);g))
by (metis (mono-tags, lifting) SatChopExistsL sat-d-def unl-Rex )

```

```

lemma SatExistsChopR:
  (sat f;g) = ( $\exists$  k. (sat f;(g  $\wedge$  len k)))
by (metis (mono-tags, lifting) SatChopExistsR sat-d-def unl-Rex )

```

```

lemma CommonPfxChopFixed:
  assumes basevars v
     $\ddagger$ [f]-v
     $\ddagger$ [g]-v
     $\vdash$  f  $\longrightarrow$  fin w
     $\vdash$  init(w)  $\longrightarrow$  g
  shows  $\ddagger$ [(f  $\wedge$  len k);g]-v

```

```

proof –
have 4:  $\ddagger[f \wedge \text{fin } w] \cdot v$ 
  using assms
  by ( simp add: itl-defs min-def interval-prefix-suffix-help commonpfx-def split: if-split-asm)
have 5:  $\ddagger[\text{init } w \wedge g] \cdot v$ 
  using assms
  by ( simp add: itl-defs min-def interval-prefix-suffix-help commonpfx-def split: if-split-asm)
have 6:  $\ddagger[(f \wedge \text{len } k); g] \cdot v$ 
  using assms 4 5
proof ( auto simp add: itl-defs min-def Valid-def interval-prefix-suffix-help commonpfx-def)
  fix s1 :: state interval
  fix s :: state interval
  assume a0: basevars v
  assume a1:  $\forall s1 \ s. f \ s1 \wedge f \ s \wedge \text{ilen } s \leq \text{ilen } s1 \longrightarrow \text{imap } v \ s = \text{imap } v \ (\text{prefix } (\text{ilen } s) \ s1)$ 
  assume a2:  $\forall s1 \ s. g \ s1 \wedge g \ s \wedge \text{ilen } s \leq \text{ilen } s1 \longrightarrow \text{imap } v \ s = \text{imap } v \ (\text{prefix } (\text{ilen } s) \ s1)$ 
  assume a3:  $\forall wa. f \ wa \longrightarrow w \ \langle \text{ilast } wa \rangle$ 
  assume a4:  $\forall wa. w \ \langle \text{ifirst } wa \rangle \longrightarrow g \ wa$ 
  assume a5:  $\text{ilen } s \leq \text{ilen } s1$ 
  assume a6:  $k \leq \text{ilen } s$ 
  assume a7:  $f \ (\text{prefix } k \ s1)$ 
  assume a8:  $f \ (\text{prefix } k \ s)$ 
  assume a9:  $g \ (\text{suffix } k \ s1)$ 
  assume a10:  $g \ (\text{suffix } k \ s)$ 
  show  $\text{imap } v \ s = \text{imap } v \ (\text{prefix } (\text{ilen } s) \ s1)$ 
proof –
  have 61:  $\text{ilen } (\text{prefix } k \ s) \leq \text{ilen } (\text{prefix } k \ s1)$ 
    using a5 by auto
  have 62:  $\text{imap } v \ (\text{prefix } k \ s) = \text{imap } v \ (\text{prefix } (\text{ilen } (\text{prefix } k \ s)) \ (\text{prefix } k \ s1))$ 
    using 61 a1 a7 a8 by blast
  have 63:  $\text{ilen } (\text{suffix } k \ s) \leq \text{ilen } (\text{suffix } k \ s1)$ 
    by (simp add: a5 diff-le-mono)
  have 64:  $\text{imap } v \ (\text{suffix } k \ s) = \text{imap } v \ (\text{prefix } (\text{ilen } (\text{suffix } k \ s)) \ (\text{suffix } k \ s1))$ 
    using 63 a10 a2 a9 by blast
  have 65:  $\text{fuse } (\text{prefix } k \ s) \ (\text{suffix } k \ s) = s$ 
    by (simp add: a6 fuse-prefix-suffix)
  have 66:  $\text{fuse } (\text{prefix } k \ s1) \ (\text{suffix } k \ s1) = s1$ 
    using a5 a6 fuse-prefix-suffix order-trans by blast
  have 67:  $\text{imap } v \ (\text{fuse } (\text{prefix } k \ s) \ (\text{suffix } k \ s)) =$ 
     $\text{fuse } (\text{imap } v \ (\text{prefix } k \ s)) \ (\text{imap } v \ (\text{suffix } k \ s))$ 
    by (simp add: a6 fuse-prefix-suffix imap-prefix imap-suffix)
  have 68:  $\text{fuse } (\text{imap } v \ (\text{prefix } k \ s)) \ (\text{imap } v \ (\text{suffix } k \ s)) =$ 
     $\text{fuse } (\text{imap } v \ (\text{prefix } (\text{ilen } (\text{prefix } k \ s)) \ (\text{prefix } k \ s1)))$ 
     $(\text{imap } v \ (\text{prefix } (\text{ilen } (\text{suffix } k \ s)) \ (\text{suffix } k \ s1)))$ 
    by (simp add: 62 64)
  have 69:  $(\text{fuse } (\text{imap } v \ (\text{prefix } (\text{ilen } (\text{prefix } k \ s)) \ (\text{prefix } k \ s1)))$ 
     $(\text{imap } v \ (\text{prefix } (\text{ilen } (\text{suffix } k \ s)) \ (\text{suffix } k \ s1)))) =$ 
     $(\text{fuse } (\text{prefix } (\text{ilen } (\text{prefix } k \ s)) \ (\text{imap } v \ (\text{prefix } k \ s1)))$ 
     $(\text{prefix } (\text{ilen } (\text{suffix } k \ s)) \ (\text{imap } v \ (\text{suffix } k \ s1))))$ 
    by (metis 61 63 imap-prefix)
  have 70:  $(\text{fuse } (\text{prefix } k \ (\text{imap } v \ (\text{prefix } k \ s1))))$ 

```

```

      (prefix (ilen s - k) (imap v (suffix k s1)))) =
      (prefix (ilen (fuse (prefix k s) (suffix k s)))
        (fuse (imap v (prefix k s1)) (imap v (suffix k s1))))
proof -
  have f1:  $\forall n \text{ na. } \neg (n::\text{nat}) \leq \text{na} \vee \text{min } n \text{ na} = n$ 
    by (meson min-def-raw)
  have f2:  $\forall n \text{ i. } \text{ilen} (\text{prefix } n (\text{i}::\text{state interval})) = \text{min } n (\text{ilen } i)$ 
    using prefix-ilen-min by blast
  have f3:  $k + \text{ilen} (\text{suffix } k s) = \text{ilen} (\text{fuse} (\text{prefix } k s) (\text{suffix } k s))$ 
    using f1 by (simp add: a6 fuse-ilen-a)
  have f4:  $k \leq \text{ilen } s1$ 
    by (meson a5 a6 order.trans)
  have  $\forall n \text{ na. } \neg (n::\text{nat}) \leq \text{na} \vee \text{na} - n + n = \text{na}$ 
    using le-add-diff-inverse2 by blast
  then have  $\text{ilen } s - k + k = \text{ilen } s$ 
    by (meson a6)
  then show ?thesis
    using f4 f3 f2 f1
    by (metis a5 a6 fuse-prefix-suffix ilen-imap imap-prefix
      imap-suffix pref-pref-3 prefix-ilen suffix-prefix-swap)
qed
have 71: (prefix (ilen (fuse (prefix k s) (suffix k s)))
  (fuse (imap v (prefix k s1)) (imap v (suffix k s1)))) =
  (imap v (prefix (ilen s) s1))
  by (metis a5 a6 fuse-prefix-suffix ilen-imap imap-prefix imap-suffix le-trans)
show ?thesis using 65 67 68 69 70 71
using a6 prefix-ilen-good by fastforce
qed
qed
show ?thesis by (simp add: 6)
qed

```

**lemma** *CommonPfxChopFixedExists:*

```

assumes basevars v
   $\ddagger[f]-v$ 
   $\ddagger[g]-v$ 
   $\vdash f \longrightarrow \text{fin } w$ 
   $\vdash \text{init}(w) \longrightarrow g$ 
shows  $(\exists k. (\ddagger[(f \wedge \text{len } k);g]-v))$ 
using CommonPfxChopFixed assms by blast

```

**lemma** *CommonPfxChopExists:*

```

   $(\ddagger[(\exists k. (f \wedge \text{len } k);g)]-v) \implies (\exists k. (\ddagger[(f \wedge \text{len } k);g]-v))$ 
unfolding commonpfx-def
by auto

```

**lemma** *CommonPfxExistsChop:*

```

   $(\ddagger[(\exists k. (f \wedge \text{len } k);g)]-v) = (\ddagger[f;g]-v)$ 
by (metis ChopEqvExistAndLenKChop inteq-reflection)

```

### 17.1.7 Forward Executability Theorems

**lemma** *FExecStableInitial:*

```

assumes basevars v
shows  $\dagger[\$v = \#c \wedge \text{stable } v] \cdot v$ 
proof –
  have 1:  $\text{sat } (\$v = \#c \wedge \text{stable } v)$ 
    using assms by (simp add: sat-d-def commonpfx-def itl-defs basevars-def) (meson len0help)
  have 2:  $\dagger[\$v = \#c \wedge \text{stable } v] \cdot v$ 
    using assms by (simp add: min-def sat-d-def commonpfx-def itl-defs basevars-def imap-eq-inth-ilen)
  show ?thesis unfolding fexec-def using 1 2 by auto
qed

```

**lemma** *FExecAlwaysConstant:*

```

assumes basevars v
shows  $\dagger[\Box(\$v = \#c)] \cdot v$ 
proof –
  have 1:  $\text{sat } \Box(\$v = \#c)$ 
    using assms by (simp add: sat-d-def commonpfx-def itl-defs basevars-def) (metis len0help)
  have 2:  $\dagger[\Box(\$v = \#c)] \cdot v$ 
    using assms by (simp add: min-def sat-d-def commonpfx-def itl-defs basevars-def imap-eq-inth-ilen)
  show ?thesis unfolding fexec-def using 1 2 by auto
qed

```

**lemma** *FExecGetsAndMore:*

```

assumes basevars v
shows  $\dagger[\$v = \#(0::\text{nat}) \wedge \text{more} \wedge v \text{ gets } (\$v + \#1)] \cdot v$ 
proof –
  have 10:  $\text{sat } \$v = \#0 \wedge \text{more} \wedge v \text{ gets } (\$v + \#1)$ 
    using assms by (simp add: sat-d-def itl-defs)
    (metis inth.simps(1) basevars inth-Suc inth-zero ilen.simps(1) ilen.simps(2) neq0-conv not-less-eq plus-1-eq-Suc)
  have 20:  $\dagger[\$v = \#0 \wedge \text{more} \wedge v \text{ gets } (\$v + \#1)] \cdot v$ 
    using assms
  proof (auto simp add: commonpfx-def itl-defs)
    show  $\bigwedge s1 \ s.$ 
       $\text{basevars } v \implies$ 
       $v \ (\text{ifirst } s1) = 0 \implies$ 
       $\forall i < \text{ilen } s1. \ v \ (\text{inth } s1 \ (\text{Suc } i)) = \text{Suc } (v \ (\text{inth } s1 \ i)) \implies$ 
       $v \ (\text{ifirst } s) = 0 \implies$ 
       $0 < \text{ilen } s \implies$ 
       $\forall i < \text{ilen } s. \ v \ (\text{inth } s \ (\text{Suc } i)) = \text{Suc } (v \ (\text{inth } s \ i)) \implies$ 
       $\text{ilen } s \leq \text{ilen } s1 \implies \text{imap } v \ s = (\text{imap } v \ (\text{prefix } (\text{ilen } s) \ s1))$ 
  proof –
    fix s1
    fix s
    assume a0: basevars v
    assume a1:  $v \ (\text{ifirst } s1) = 0$ 
    assume a2:  $\forall i < \text{ilen } s1. \ v \ (\text{inth } s1 \ (\text{Suc } i)) = \text{Suc } (v \ (\text{inth } s1 \ i))$ 
    assume a3:  $v \ (\text{ifirst } s) = 0$ 

```

```

assume a4:  $0 < \text{ilen } s$ 
assume a5:  $\forall i < \text{ilen } s. v \text{ ( } \text{inth } s \text{ ( } \text{Suc } i \text{))} = \text{Suc } (v \text{ ( } \text{inth } s \text{ } i \text{))}$ 
assume a6:  $\text{ilen } s \leq \text{ilen } s1$ 
show  $\text{imap } v \text{ } s = (\text{imap } v \text{ (} \text{prefix } (\text{ilen } s) \text{ } s1 \text{))}$ 
proof –
  have 1:  $\text{ilen}(\text{imap } v \text{ } s) = \text{ilen } (\text{imap } v \text{ (} (\text{prefix } (\text{ilen } s) \text{ } s1) \text{))}$ 
    using a6 by auto
  have 2:  $\bigwedge j. j \leq \text{ilen } s \longrightarrow \text{inth } (\text{imap } v \text{ } s) \text{ } j = \text{inth } (\text{imap } v \text{ (} (\text{prefix } (\text{ilen } s) \text{ } s1) \text{)) } j$ 
    by (simp add: a1 a2 a3 a5 a6 getsmorehelp le-trans)
  show ?thesis by (metis 1 2 interval-eq-inth-eq ilen-imap)
qed
qed
qed
show ?thesis unfolding fexec-def using 10 20 by simp
qed

```

```

lemma FExecGetsAndEmpty:
assumes basevars  $v$ 
shows  $\dagger[\$v = \#(0::\text{nat}) \wedge \text{empty} \wedge v \text{ gets } (\$v + \#1)] \text{ } -v$ 
proof –
  have 10:  $\text{sat } \$v = \#0 \wedge \text{empty} \wedge v \text{ gets } (\$v + \#1)$ 
    using assms by (simp add: sat-d-def itl-defs)
    (metis baseE ilen.simps(1) inth.simps(1) not-less-zero)
  have 20:  $\dagger[\$v = \#0 \wedge \text{empty} \wedge v \text{ gets } (\$v + \#1)] \text{ } -v$ 
    using assms
    by (auto simp add: commonpfx-def itl-defs ifirst-imap INil-ilen)
show ?thesis unfolding fexec-def using 10 20 by simp
qed

```

```

lemma FExecChopFixedExists:
assumes basevars  $v$ 
   $\dagger[f] \text{ } -v$ 
   $\dagger[g] \text{ } -v$ 
   $\vdash f \longrightarrow \text{fin } w$ 
   $\vdash \text{init}(w) \longrightarrow g$ 
shows  $(\exists k. (\dagger[(f \wedge \text{len } k); g] \text{ } -v))$ 
using assms SatChopL[of  $f \text{ } g \text{ } w$ ]
unfolding fexec-def
using SatExistsChopL CommonPfxChopFixed by blast

```

```

lemma FExecNextRule:
assumes basevars  $v$ 
   $\dagger[g] \text{ } -v$ 
   $\dagger[\text{init } w \wedge \text{empty}] \text{ } -v$ 
shows  $\dagger[\text{init } w \wedge \bigcirc g] \text{ } -v$ 
proof –
  obtain s0 where 1:  $w \langle \text{ifirst } s0 \rangle \wedge \text{ilen } s0 = 0$ 
    using assms unfolding fexec-def sat-d-def
    by (auto simp add: itl-defs)
  obtain s1 where 2:  $g \text{ } s1$ 

```

```

using assms unfolding fexec-def sat-d-def
  by (auto simp add: itl-defs)
have 3:  $(\langle \text{ifirst } s0 \rangle \ominus s1) \models \text{init } w \wedge \bigcirc g$ 
  using 1 2 by (auto simp add: itl-defs)
have 4: sat init w  $\wedge \bigcirc g$ 
  unfolding sat-d-def using 3 by blast
have 5: basevars v  $\implies$ 
  (sat g)  $\wedge (\forall s1\ s. g\ s1 \wedge g\ s \wedge \text{ilen } s \leq \text{ilen } s1 \longrightarrow \text{imap } v\ s = \text{imap } v\ (\text{prefix } (\text{ilen } s)\ s1)) \implies$ 
  (sat init w  $\wedge \text{empty}$ )  $\wedge$ 
  ( $\forall s1\ s.$ 
     $w\ \langle \text{ifirst } s1 \rangle \wedge \text{ilen } s1 = 0 \wedge w\ \langle \text{ifirst } s \rangle \wedge \text{ilen } s = 0 \wedge \text{ilen } s \leq \text{ilen } s1 \longrightarrow$ 
     $(\exists y. s = \langle y \rangle \wedge v\ y = v\ (\text{ifirst } s1))) \implies$ 
 $\forall s1\ s.$ 
   $w\ \langle \text{ifirst } s1 \rangle \wedge$ 
   $0 < \text{ilen } s1 \wedge g\ (\text{suffix } (\text{Suc } 0)\ s1) \wedge w\ \langle \text{ifirst } s \rangle \wedge 0 < \text{ilen } s \wedge g\ (\text{suffix } (\text{Suc } 0)\ s) \wedge \text{ilen } s \leq \text{ilen}$ 
 $s1 \longrightarrow$ 
   $\text{imap } v\ s = \text{imap } v\ (\text{prefix } (\text{ilen } s)\ s1)$ 
proof auto
fix s1
fix s
assume a0: basevars v
assume a1:  $\forall s1\ s. g\ s1 \wedge g\ s \wedge \text{ilen } s \leq \text{ilen } s1 \longrightarrow \text{imap } v\ s = (\text{imap } v\ (\text{prefix } (\text{ilen } s)\ s1))$ 
assume a3:  $\forall s1\ s. w\ \langle \text{ifirst } s1 \rangle \wedge \text{ilen } s1 = 0 \wedge w\ \langle \text{ifirst } s \rangle \wedge \text{ilen } s = 0 \wedge \text{ilen } s \leq \text{ilen } s1$ 
 $\longrightarrow (\exists y. s = \langle y \rangle \wedge v\ y = v\ (\text{ifirst } s1))$ 
assume a4:  $w\ \langle \text{ifirst } s1 \rangle$ 
assume a5:  $g\ (\text{suffix } (\text{Suc } 0)\ s1)$ 
assume a6:  $w\ \langle \text{ifirst } s \rangle$ 
assume a7:  $0 < \text{ilen } s$ 
assume a8:  $g\ (\text{suffix } (\text{Suc } 0)\ s)$ 
assume a9:  $\text{ilen } s \leq \text{ilen } s1$ 
show  $\text{imap } v\ s = (\text{imap } v\ (\text{prefix } (\text{ilen } s)\ s1))$ 
proof -
have 51:  $\text{imap } v\ \langle \text{ifirst } s \rangle = (\text{imap } v\ (\text{prefix } (\text{ilen } \langle \text{ifirst } s \rangle)\ \langle \text{ifirst } s1 \rangle))$ 
by (metis a3 a4 a6 ilast-prefix ilen.simps(1) ilen-gr-zero interval.simps(9) prefix-zero-ifirst)
have 52:  $\text{imap } v\ (\text{suffix } (\text{Suc } 0)\ s) =$ 
 $(\text{imap } v\ (\text{prefix } (\text{ilen } (\text{suffix } (\text{Suc } 0)\ s))\ (\text{suffix } (\text{Suc } 0)\ s1)))$ 
by (metis a1 a5 a8 a9 diff-le-mono suffix-ilen)
have 53:  $s = (\langle \text{ifirst } s \rangle \ominus (\text{suffix } (\text{Suc } 0)\ s))$ 
using a7 interval-hd-tail by auto
have 54:  $\text{imap } v\ (\langle \text{ifirst } s \rangle \ominus (\text{suffix } (\text{Suc } 0)\ s)) =$ 
 $(\text{imap } v\ \langle \text{ifirst } s \rangle) \ominus (\text{imap } v\ (\text{suffix } (\text{Suc } 0)\ s))$ 
by simp
have 55:  $(\text{imap } v\ \langle \text{ifirst } s \rangle) \ominus (\text{imap } v\ (\text{suffix } (\text{Suc } 0)\ s)) =$ 
 $(\text{imap } v\ ((\text{prefix } (\text{ilen } \langle \text{ifirst } s \rangle)\ \langle \text{ifirst } s1 \rangle))) \ominus$ 
 $(\text{imap } v\ ((\text{prefix } (\text{ilen } (\text{suffix } (\text{Suc } 0)\ s))\ (\text{suffix } (\text{Suc } 0)\ s1))))$ 
using 51 52 by auto
have 56:  $(\text{imap } v\ ((\text{prefix } (\text{ilen } \langle \text{ifirst } s \rangle)\ \langle \text{ifirst } s1 \rangle))) =$ 
 $(\text{imap } v\ ((\text{prefix } (\text{ilen } \langle \text{ifirst } s \rangle)\ (\langle \text{ifirst } s1 \rangle \ominus (\text{suffix } (\text{Suc } 0)\ s1))))$ 
by auto
have 57:  $(\text{imap } v\ (\text{prefix } (\text{ilen } (\langle \text{ifirst } s \rangle \ominus (\text{suffix } (\text{Suc } 0)\ s))))$ 

```



```

      ((⟨ifirst s1⟩ ⊖ (suffix (Suc 0) s1)))) =
      (imap v ((prefix (ilen ⟨ifirst s⟩) ⟨ifirst s1⟩)) ⊖
      (imap v ((prefix (ilen (suffix (Suc 0) s)) (suffix (Suc 0) s1))))))
    by simp
  have 58: imap v (⟨ifirst s⟩ ⊖ (suffix (Suc 0) s)) =
    (imap v (prefix (ilen (⟨ifirst s⟩ ⊖ (suffix (Suc 0) s)))
    (⟨ifirst s1⟩ ⊖ (suffix (Suc 0) s1))))
    using 55 by auto
  have 59: s1 = (⟨ifirst s1⟩ ⊖ (suffix (Suc 0) s1))
    using a7 a9 interval-hd-tail by force
  show ?thesis using 51 52 53 54 57 59 by metis
qed
qed
have 6: ‡[init w ∧ ○ g]-v
  using assms 5 unfolding commonpfx-def itl-defs Valid-def fexec-def by force
show ?thesis unfolding fexec-def using 6 4 by auto
qed

```

The rewriting process of Tempura corresponds to the notion of forward executability.

**lemma** *FExecWNextRule*:

```

assumes basevars v
  ‡[g]-v
  ‡[init w ∧ empty]-v
shows ‡[init w ∧ wnext g]-v
proof –
  have 1: ⊢ (init w ∧ wnext g) = ( (init w ∧ empty) ∨ (init w ∧ ○ g))
    using WnextEqvEmptyOrNext by fastforce
  have 2: sat init w ∧ wnext g
    using 1 SatOr assms FExecNextRule[of v g w] unfolding fexec-def
    by (metis int-eq)
  have 3: ‡[init w ∧ ○ g]-v
    using assms FExecNextRule[of v g w] unfolding fexec-def
    by blast
  have 4: ‡[init w ∧ wnext g]-v
    using 1 3 assms StateAndEmptyChop[of w LIFT(○g)] unfolding commonpfx-def fexec-def
  proof (auto simp add: itl-defs, metis INil-ilen ilen-gr-zero)
  fix s1 :: state interval and s :: state interval
  assume a1: ∀ s1 s. w ⟨ifirst s1⟩ ∧ 0 < ilen s1 ∧ g (suffix (Suc 0) s1) ∧ w ⟨ifirst s⟩
    ∧ 0 < ilen s ∧ g (suffix (Suc 0) s) ∧ ilen s ≤ ilen s1 ⟶
    imap v s = imap v (prefix (ilen s) s1)
  assume a2: ilen s ≤ ilen s1
  assume a3: w ⟨ifirst s1⟩
  assume a4: w ⟨ifirst s⟩
  assume a5: g (suffix (Suc 0) s1)
  assume a6: g (suffix (Suc 0) s)
  assume a7: ∀ s1 s. w ⟨ifirst s1⟩ ∧ ilen s1 = 0 ∧ w ⟨ifirst s⟩ ∧ ilen s = 0 ∧ ilen s ≤ ilen s1 ⟶
    (∃ y. s = ⟨y⟩ ∧ v y = v (ifirst s1))
  then have a8: ∀ i ia. (((imap v ia = imap v ⟨ifirst i⟩) ∨ ilen i ≠ 0) ∨ ilen ia ≠ 0) ∨
    ¬ w ⟨ifirst i⟩) ∨ ¬ w ⟨ifirst ia⟩) ∨ ¬ ilen ia ≤ ilen i
  by auto

```

```

then show  $\text{imap } v \ s = \text{imap } v \ (\text{prefix } (\text{ilen } s) \ s1)$ 
using  $a6 \ a5 \ a4 \ a3 \ a2 \ a1$ 
by ( $\text{metis } (\text{no-types}) \ \text{ifirst-prefix } \text{le0 } \text{le-antisym } \text{neq0-conv } \text{prefix-ilen-good}$ )
qed
show ?thesis unfolding fexec-def using  $2 \ 4$  by auto
qed

```

The following lemma states the conditions required to strengthen a forward executable specification.

```

lemma FExecStrengthen:
assumes  $\text{basevars } v$ 
 $\text{sat } f \wedge g$ 
 $\dagger[f]-v$ 
shows  $\dagger[f \wedge g]-v$ 
using assms unfolding fexec-def commonpfx-def
by simp

```

## 17.2 Backward Executability

### 17.2.1 Common Suffix Value Trace Definitions

Reflection relates the notion of prefix intervals with that of suffix intervals. So we need to introduce the “mirror image” of common prefix value traces, i.e. the notion of common suffix value trace.

```

definition commonsfx :: ( $'a :: \text{world}$ )  $\text{formula} \Rightarrow ('a, 'b) \text{stfun} \Rightarrow \text{bool}$ 
where  $\text{commonsfx } F \ v \equiv$ 
 $(\forall \ s1 \ s. (s1 \models F) \wedge (s \models F) \wedge \text{ilen } s \leq \text{ilen } s1 \longrightarrow$ 
 $\text{imap } v \ s = (\text{imap } v \ (\text{suffix } ((\text{ilen } s1) - (\text{ilen } s)) \ s1))$ 
 $)$ 

```

The notion of common suffix value trace corresponds to notion of generating a satisfying interval for a formula but it “limits” how this is achieved, i.e., one proceeds in a backward manner. The following definition introduces the notion of backward executability.

```

definition bexec :: ( $'a :: \text{world}$ )  $\text{formula} \Rightarrow ('a, 'b) \text{stfun} \Rightarrow \text{bool}$ 
where  $\text{bexec } F \ v \equiv ( (\text{sat-d } F) \wedge (\text{commonsfx } F \ v) )$ 

```

**syntax**

```

 $\text{-commonsfx} :: \text{lift} \Rightarrow \text{lift} \Rightarrow \text{bool}$   $((\dagger[-]'-(-)) \ 5)$ 
 $\text{-bexec} :: \text{lift} \Rightarrow \text{lift} \Rightarrow \text{bool}$   $((\dagger[-]'-(-)) \ 5)$ 

```

**translations**

```

 $\text{-commonsfx} == \text{CONST commonsfx}$ 
 $\text{-bexec} == \text{CONST bexec}$ 

```

### 17.2.2 Common suffix value trace theorems

```

lemma CommonSfxFixedTrace:
assumes  $\text{basevars } v$ 
 $\text{sat } f \wedge \text{len } k$ 
 $\dagger[f]-v$ 
shows  $\text{card } \{ (\text{imap } v \ s) \mid s. (s \models f) \wedge \text{ilen } s = k \} = 1$ 
proof –

```

```

obtain  $s0$  where 1:  $(s0 \models f) \wedge \text{ilen } s0 = k$ 
  by (metis (mono-tags, lifting) assms(2) len-defs sat-d-def unl-lift2)
have 3:  $\forall s1. (s1 \models f) \wedge \text{ilen } s1 = k \longrightarrow (\text{imap } v \ s1) = (\text{imap } v \ s0)$ 
using 1 assms unfolding commonsfx-def by (metis le-less)
have 4:  $\{ (\text{imap } v \ s) \mid s. (s \models f) \wedge \text{ilen } s = k \} = \{ (\text{imap } v \ s0) \}$ 
  using 3 1 by blast
show ?thesis by (simp add: 4)
qed

```

The following theorem states that the combination of satisfiability with the notion of common suffix value trace can be used to determine whether a formula is deterministic or not, i.e., is backward executable or not.

**lemma** *CommonSfxTrace*:

```

assumes basevars v
   $\text{sat } f$ 
   $\mathfrak{h}[f]\text{-}v$ 
shows  $\text{card } \{ (\text{imap } v \ s) \mid s. (s \models f) \wedge \text{ilen } s = k \} \leq 1$ 
proof –
have 1:  $(\text{sat } f) = (\text{sat } (\exists k. f \wedge \text{len } k))$ 
  by (simp add: len-defs sat-d-def)
have 2:  $\bigwedge k. (\text{sat } f \wedge \text{len } k) \vee \neg (\text{sat } f \wedge \text{len } k)$ 
  by simp
have 3:  $\bigwedge k. \neg (\text{sat } f \wedge \text{len } k) \implies \text{card } \{ \text{imap } v \ s \mid s. f \ s \wedge \text{ilen } s = k \} \leq 1$ 
  using assms NoTrace[of v f] by auto
have 4:  $\bigwedge k. \text{sat } f \wedge \text{len } k \implies \text{card } \{ \text{imap } v \ s \mid s. f \ s \wedge \text{ilen } s = k \} \leq 1$ 
  using assms CommonSfxFixedTrace[of v f] by auto
show ?thesis using 2 3 4 by auto
qed

```

**lemma** *PairCommonSfx*:

```

assumes basevars (v1, v2)
   $\mathfrak{h}[f]\text{-}v1$ 
   $\mathfrak{h}[f]\text{-}v2$ 
shows  $\mathfrak{h}[f]\text{-(}v1, v2\text{)}$ 
using assms unfolding commonsfx-def
by (simp add: imap-eq-inth-ilen)

```

**lemma** *ProjCommonSfxA*:

```

assumes basevars (v1, v2)
   $\mathfrak{h}[f]\text{-(}v1, v2\text{)}$ 
shows  $\mathfrak{h}[f]\text{-}v1$ 
using assms unfolding commonsfx-def
by (simp add: imap-eq-inth-ilen)

```

**lemma** *ProjCommonSfxB*:

```

assumes basevars (v1, v2)
   $\mathfrak{h}[f]\text{-(}v1, v2\text{)}$ 
shows  $\mathfrak{h}[f]\text{-}v2$ 
using assms unfolding commonsfx-def
by (simp add: imap-eq-inth-ilen)

```

**lemma** *PairSplitCommonSfx*:  
**assumes** *basevars* (*v1*, *v2*)  
**shows**  $(\mathbb{h}[f]-(v1, v2)) = ((\mathbb{h}[f]-v1) \wedge (\mathbb{h}[f]-v2))$   
**using** *PairCommonSfx ProjCommonSfxA ProjCommonSfxB assms by blast*

### 17.2.3 Time reversal theorems

**lemma** *SatRev*:  
**assumes** *basevars* *v*  
**shows**  $(\text{sat } f) = (\text{sat } (f^r))$   
**using** *assms unfolding sat-d-def*  
**by** (*metis irev-irev-ident reverse-d-def*)

**lemma** *RevCommonPfxSfx*:  
**assumes** *basevars* *v*  
**shows**  $(\mathbb{h}[f]-v) = (\mathbb{h}[f^r]-v)$   
**using** *assms unfolding commonpfx-def commonsfx-def reverse-d-def*  
**by** (*metis (no-types, lifting) irev-ilen irev-prefix irev-irev-ident irev-imap*)

**lemma** *RevCommonSfxPfx*:  
**assumes** *basevars* *v*  
**shows**  $(\mathbb{h}[f]-v) = (\mathbb{h}[f^r]-v)$   
**using** *assms*  
**by** (*simp add: RevCommonPfxSfx reverse-d-def*)

### 17.2.4 Common suffix value trace and len and assignment

**lemma** *CommonSfxUnitAssignAndNextValAndSkip*:  
**assumes** *basevars* (*v* :: *state*  $\Rightarrow$  *nat*)  
**shows**  $(\mathbb{h}[\$v = \#c \wedge \text{skip} \wedge v\$ = \#d]-v)$   
**proof** –  
**have**  $\exists: \text{commonsfx } (LIFT(\$v = \#c \wedge \text{skip} \wedge v\$ = \#d)) \ v$   
**using** *assms using lenkhelp1[of v - c 1 d]*  
**by** (*simp add: commonsfx-def next-val-d-def itl-defs basevars-def*)  
**show** *?thesis* **using**  $\exists$  **by** *auto*  
**qed**

**lemma** *NotCommonSfxUnitAssignAndLenK*:  
**assumes** *basevars* (*v* :: *state*  $\Rightarrow$  *nat*)  
 $0 < k$   
**shows**  $\neg (\mathbb{h}[\$v = \#0 \wedge \text{len } k]-v)$   
**using** *assms using lenkhelp4[of v k 0 0] lenkhelp4[of v k 0 1]*  
**by** (*simp add: commonsfx-def next-val-d-def itl-defs basevars-def*)  
*(metis One-nat-def inth-imap order-refl zero-neq-one)*

**lemma** *CommonSfxStableAndLenK*:  
**assumes** *basevars* *v*  
**shows**  $(\mathbb{h}[\$v = \#(c::\text{nat}) \wedge \text{len } k \wedge \text{stable } v]-v)$

```

proof –
  have 3: commonsfx (LIFT($v = #(c::nat) ∧ len k ∧ stable v)) v
    using assms by (simp add: commonsfx-def itl-defs basevars-def imap-eq-inth-ilen)
  show ?thesis using 3 by simp
qed

```

**lemma** *NotCommonSfxGetsAndMore*:

**assumes** *basevars* *v*

**shows**  $\neg(\text{h}[\$v = \#(0::nat) \wedge \text{more} \wedge v \text{ gets } (\$v + \#1) ] - v)$

**proof** –

**have** 10:  $\bigwedge k. \exists s. v \text{ (ifirst } s) = 0 \wedge \text{ilen } s = k \wedge$   
 $(\forall i < \text{ilen } s. v \text{ ( inth } s \text{ (Suc } i)) = Suc \text{ (} v \text{ ( inth } s \text{ } i))})$

**using** *assms*

**proof** –

**fix** *k*

**show** *basevars* *v*  $\implies$

$\exists s. v \text{ (ifirst } s) = 0 \wedge \text{ilen } s = k \wedge$   
 $(\forall i < \text{ilen } s. v \text{ ( inth } s \text{ (Suc } i)) = Suc \text{ (} v \text{ ( inth } s \text{ } i))})$

**proof** (*induct* *k*)

**case** 0

**then show** ?case

**by** (*metis* *inth.simps*(1) *basevars* *ilen.simps*(1) *less-nat-zero-code*)

**next**

**case** (*Suc* *k*)

**then show** ?case

**proof** –

**have** 11:  $\exists s. v \text{ (ifirst } s) = 0 \wedge \text{ilen } s = k \wedge$   
 $(\forall i < \text{ilen } s. v \text{ ( inth } s \text{ (Suc } i)) = Suc \text{ (} v \text{ ( inth } s \text{ } i))})$

**using** *Suc.hyps* *Suc.prem*s **by** *blast*

**obtain** *ss* **where** 12:  $v \text{ (ifirst } ss) = 0 \wedge \text{ilen } ss = k \wedge$   
 $(\forall i < \text{ilen } ss. v \text{ ( inth } ss \text{ (Suc } i)) = Suc \text{ (} v \text{ ( inth } ss \text{ } i))})$

**using** 11 **by** *auto*

**have** 121:  $v \text{ (inth } ss \text{ } k) = k$

**by** (*metis* 12 *getsmorehelp* *inth-imap* *order-refl*)

**have** 13:  $\exists s. v \text{ (} s) = (\text{Suc } k)$

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

**obtain** *ssl* **where** 141:  $v \text{ } ssl = (\text{Suc } k)$

**using** 13 **by** *auto*

**have** 14:  $v \text{ (ifirst (iapp } ss \text{ (INil } ssl)))} = 0 \wedge \text{ilen (iapp } ss \text{ (INil } ssl))} = \text{Suc } k$

**by** (*simp* *add*: 12)

**have** 15:  $(\forall i < \text{ilen (iapp } ss \text{ (INil } ssl)).$

$v \text{ (inth (iapp } ss \text{ (INil } ssl)) \text{ (Suc } i))} = \text{Suc (} v \text{ (inth (iapp } ss \text{ (INil } ssl)) \text{ } i))}) =$   
 $((\forall i < \text{ilen } ss.$

$v \text{ (inth (iapp } ss \text{ (INil } ssl)) \text{ (Suc } i))} = \text{Suc (} v \text{ (inth (iapp } ss \text{ (INil } ssl)) \text{ } i))}) \wedge$

$v \text{ (inth (iapp } ss \text{ (INil } ssl)) \text{ (Suc } k))} = \text{Suc (} v \text{ (inth (iapp } ss \text{ (INil } ssl)) \text{ } k))})$

**using** 14 *less-Suc-eq* **by** *auto*

**have** 16:  $v \text{ (inth (iapp } ss \text{ (INil } ssl)) \text{ (Suc } k))} = v \text{ } ssl$

**by** (*metis* 14 *ilast-iapp*)

**have** 17:  $v \text{ (inth (iapp } ss \text{ (INil } ssl)) \text{ } k) = v \text{ (inth } ss \text{ } k)$

```

    by (simp add: 12 iapp-inth)
  have 18:  $v \text{ (inth (iapp ss (INil ssl)) (Suc k))} = \text{Suc } (v \text{ (inth (iapp ss (INil ssl)) k))}$ 
    by (simp add: 121 141 16 17)
  have 19:  $(\forall i < \text{ilen ss. } v \text{ (inth (iapp ss (INil ssl)) (Suc i))} = \text{Suc } (v \text{ (inth (iapp ss (INil ssl)) i))})$ 
    by (simp add: 12 iapp-inth)
  show ?thesis
  using 14 15 18 19 by blast
qed
qed
qed
have 20:  $\neg (\text{h}[\$v = \#0 \wedge \text{more} \wedge v \text{ gets } (\$v + \#1)] - v)$ 
  using assms
proof (simp add: commonsfx-def itl-defs)
  obtain s0 where 1:  $v \text{ (ifirst s0)} = 0 \wedge 1 = \text{ilen s0} \wedge$ 
     $(\forall i < \text{ilen s0. } v \text{ (inth s0 (Suc i))} = \text{Suc } (v \text{ (inth s0 i))})$ 
    using 10[of 1]
    by fastforce
  obtain s1 where 2:  $v \text{ (ifirst s1)} = 0 \wedge 2 = \text{ilen s1} \wedge$ 
     $(\forall i < \text{ilen s1. } v \text{ (inth s1 (Suc i))} = \text{Suc } (v \text{ (inth s1 i))})$ 
    using 10[of 2] by fastforce
  have 3:  $\text{ilen s0} \leq \text{ilen s1}$ 
    using 1 2 by auto
  have 4:  $\text{imap } v \text{ s0} \neq (\text{imap } v (\text{suffix } (\text{ilen s1} - \text{ilen s0}) \text{ s1}))$ 
    by (metis 1 2 3 One-nat-def Suc-1 ilast-suffix ilast-imap lessI less-le-trans n-not-Suc-n)
  have 5:  $0 < \text{ilen s0}$ 
    using 1 by linarith
  have 6:  $0 < \text{ilen s1}$ 
    using 3 5 order.strict-trans2 by blast
  show basevars  $v \implies$ 
 $\exists s1. v \text{ (ifirst s1)} = 0 \wedge$ 
 $0 < \text{ilen s1} \wedge$ 
 $(\forall i < \text{ilen s1. } v \text{ (inth s1 (Suc i))} = \text{Suc } (v \text{ (inth s1 i))}) \wedge$ 
 $(\exists s. v \text{ (ifirst s)} = 0 \wedge$ 
 $0 < \text{ilen s} \wedge$ 
 $(\forall i < \text{ilen s. } v \text{ (inth s (Suc i))} = \text{Suc } (v \text{ (inth s i))}) \wedge$ 
 $\text{ilen s} \leq \text{ilen s1} \wedge \text{imap } v \text{ s} \neq (\text{imap } v (\text{suffix } (\text{ilen s1} - \text{ilen s}) \text{ s1}))$ 
    using 1 2 3 4 5 6 by blast
qed
show ?thesis using 20 by auto
qed

```

**lemma** *CommonSfxGetsAndLenK*:

**assumes** *basevars v*

**shows**  $(\text{h}[\$v = \#(0::\text{nat}) \wedge \text{len } k \wedge v \text{ gets } (\$v + \#1)] - v)$

**proof** –

**have** 10:  $\bigwedge k. \exists s. v \text{ (ifirst s)} = 0 \wedge \text{ilen s} = k \wedge$

$(\forall i < \text{ilen s. } v \text{ (inth s (Suc i))} = \text{Suc } (v \text{ (inth s i))})$

**using** *assms*

**proof** –

```

fix k
show basevars v  $\implies$ 
   $\exists s. v \text{ (ifirst } s) = 0 \wedge \text{ilen } s = k \wedge$ 
     $(\forall i < \text{ilen } s. v \text{ (inth } s \text{ (Suc } i)) = \text{Suc } (v \text{ (inth } s \text{ } i)))$ 
proof (induct k)
case 0
then show ?case
by (metis inth.simps(1) basevars ilen.simps(1) less-nat-zero-code)
next
case (Suc k)
then show ?case
proof -
have 11:  $\exists s. v \text{ (ifirst } s) = 0 \wedge \text{ilen } s = k \wedge$ 
   $(\forall i < \text{ilen } s. v \text{ (inth } s \text{ (Suc } i)) = \text{Suc } (v \text{ (inth } s \text{ } i)))$ 
  using Suc.hyps Suc.premis by blast
obtain ss where 12:  $v \text{ (ifirst } ss) = 0 \wedge \text{ilen } ss = k \wedge$ 
   $(\forall i < \text{ilen } ss. v \text{ (inth } ss \text{ (Suc } i)) = \text{Suc } (v \text{ (inth } ss \text{ } i)))$ 
  using 11 by auto
have 121:  $v \text{ (inth } ss \text{ } k) = k$ 
  by (metis 12 getsmorehelp inth-imap order-refl)
have 13:  $\exists s. v \text{ (s } s) = (\text{Suc } k)$ 
  by (simp add: assms basevars)
obtain ssl where 141:  $v \text{ (s } ssl) = (\text{Suc } k)$ 
  using 13 by auto
have 14:  $v \text{ (ifirst (iapp ss (INil ssl)))} = 0 \wedge \text{ilen (iapp ss (INil ssl))} = \text{Suc } k$ 
  by (simp add: 12)
have 15:  $(\forall i < \text{ilen (iapp ss (INil ssl))}. v \text{ (inth (iapp ss (INil ssl)) (Suc } i)) = \text{Suc } (v \text{ (inth (iapp ss (INil ssl)) } i))) =$ 
   $((\forall i < \text{ilen } ss. v \text{ (inth (iapp ss (INil ssl)) (Suc } i)) = \text{Suc } (v \text{ (inth (iapp ss (INil ssl)) } i))) \wedge$ 
   $v \text{ (inth (iapp ss (INil ssl)) (Suc } k)) = \text{Suc } (v \text{ (inth (iapp ss (INil ssl)) } k)))$ 
  using 14 less-Suc-eq by auto
have 16:  $v \text{ (inth (iapp ss (INil ssl)) (Suc } k)) = v \text{ (s } ssl)$ 
  by (metis 14 ilast-iapp)
have 17:  $v \text{ (inth (iapp ss (INil ssl)) } k) = v \text{ (inth } ss \text{ } k)$ 
  by (simp add: 12 iapp-inth)
have 18:  $v \text{ (inth (iapp ss (INil ssl)) (Suc } k)) = \text{Suc } (v \text{ (inth (iapp ss (INil ssl)) } k))$ 
  by (simp add: 121 141 16 17)
have 19:  $(\forall i < \text{ilen } ss. v \text{ (inth (iapp ss (INil ssl)) (Suc } i)) = \text{Suc } (v \text{ (inth (iapp ss (INil ssl)) } i)))$ 
  by (simp add: 12 iapp-inth)
show ?thesis
using 14 15 18 19 by blast
qed
qed
qed
have 20 :  $(\nexists [ \$v = \#0 \wedge \text{len } k \wedge v \text{ gets } (\$v + \#1) ] -v)$ 
  using assms
  by (simp add: commonsfx-def itl-defs imap-eq-inth-ilen inth-imap)
  (metis getsmorehelp inth-imap)

```

**show** *?thesis* **using** 20 **by** *auto*  
**qed**

**lemma** *RevCommonPfxStableAndLenK*:  
**assumes** *basevars v*  
**shows**  $(\ddagger[(\$v = \#(c::nat) \wedge len\ k \wedge stable\ v)^r] -v)$   
**using** *assms*  
**using** *RevCommonPfxSfx CommonSfxStableAndLenK* **by** *blast*

**lemma** *RevCommonSfxStableAndLenK*:  
**assumes** *basevars v*  
**shows**  $(\natural[(\$v = \#(c::nat) \wedge len\ k \wedge stable\ v)^r] -v)$   
**using** *assms*  
**using** *RevCommonSfxPfx CommonPfxStableAndLenK* **by** *blast*

**lemma** *StableAndLenKRev*:  
 $\vdash (\$v = \#(c::nat) \wedge len\ k \wedge stable\ v) = (\$v = \#(c::nat) \wedge len\ k \wedge stable\ v)^r$   
**by** (*simp add: Valid-def itl-defs current-val-d-def reverse-d-def*)  
*(metis diff-le-self ifirst-irev irev-ilen irev-inth*  
*prefix-ilen-bound prefix-ilen irev-irev-ident)*

## 17.2.5 Basic theorems

**lemma** *RevAndWeakNext*:  
 $\vdash (init\ w \wedge wnext\ f \wedge fin\ w1)^r = (fin\ w \wedge wprev\ (f^r) \wedge init\ w1)$   
**by** (*metis (no-types, lifting) RInitEqvFin RRand RRWPrevEqvWNext all-rev-eq(3) inteq-reflection*)

**lemma** *CommonSfxChopFixed*:  
**assumes** *basevars v*  
 $\natural[f] -v$   
 $\natural[g] -v$   
 $\vdash fin\ w \longrightarrow f$   
 $\vdash g \longrightarrow init\ w$   
**shows**  $\natural[f;(g \wedge len\ k)] -v$   
**proof** –  
**have** 1:  $\ddagger[g^r] -v$   
**using** *assms RevCommonPfxSfx* **by** *blast*  
**have** 2:  $\ddagger[f^r] -v$   
**using** *assms RevCommonPfxSfx* **by** *blast*  
**have** 3:  $\vdash g^r \longrightarrow fin\ w$   
**by** (*metis RImpRule RInitEqvFin assms(5) int-eq*)  
**have** 4:  $\vdash init\ w \longrightarrow f^r$   
**by** (*metis RFinEqvInit RImpRule assms(4) inteq-reflection*)  
**have** 5:  $\ddagger[(g^r \wedge len\ k);f^r] -v$   
**using** *assms using CommonPfxChopFixed[of v LIFT(g^r) LIFT(f^r) w k]*  
**using** 1 2 3 4 **by** *blast*  
**have** 6:  $\vdash (g^r \wedge len\ k)^r = (g \wedge len\ k)$   
**by** (*simp add: intI len-defs reverse-d-def*)  
**have** 7:  $\vdash ((g^r \wedge len\ k);f^r)^r = f;(g \wedge len\ k)$   
**by** (*metis 6 EqvReverseReverse all-rev-eq(13) int-eq*)



**have** 8:  $(\dagger[(g^r \wedge \text{len } k); f^r] - v) = (\natural[f; (g \wedge \text{len } k)] - v)$   
**using** 7 *RevCommonSfxPfx* *assms*(1) *int-eq* **by** *force*  
**show** ?thesis **using** 5 8 **by** *auto*  
**qed**

**lemma** *CommonSfxEqv*:  
**assumes** *basevars v*  
 $\vdash f = g$   
**shows**  $(\natural[f] - v) = (\natural[g] - v)$   
**using** *assms inteq-reflection* **by** *blast*

**lemma** *CommonSfxEmptyOrMore*:  
**assumes** *basevars v*  
**shows**  $(\natural[f] - v) = (\natural[(f \wedge \text{empty}) \vee (f \wedge \text{more})] - v)$   
**proof** –  
**have** 1:  $\vdash f = ((f \wedge \text{empty}) \vee (f \wedge \text{more}))$   
**unfolding** *empty-d-def* **by** *fastforce*  
**show** ?thesis **using** *CommonSfxEqv* 1 *assms* **by** *blast*  
**qed**

**lemma** *CommonSfxChoice*:  
**assumes** *basevars v*  
**shows**  $(\natural[f] - v) = (\natural[(f \wedge g) \vee (f \wedge \neg g)] - v)$   
**proof** –  
**have** 1:  $\vdash f = ((f \wedge g) \vee (f \wedge \neg g))$   
**by** *fastforce*  
**show** ?thesis **using** *CommonSfxEqv* 1 *assms* **by** *blast*  
**qed**

**lemma** *CommonSfxDetChoice*:  
**assumes** *basevars v*  
 $\natural[(f \wedge g) \vee (f \wedge \neg g)] - v$   
**shows**  $(\natural[f \wedge g] - v) \wedge (\natural[f \wedge \neg g] - v)$   
**using** *assms unfolding commonsfx-def* **by** *simp*

**lemma** *CommonSfxBox*:  
**assumes** *basevars v*  
**shows**  $(\natural[\Box f] - v) = (\natural[f \wedge \text{wnext } (\Box f)] - v)$   
**proof** –  
**have** 1:  $\vdash \Box f = (f \wedge \text{wnext } (\Box f))$   
**by** (*simp add: BoxEqvAndWnextBox*)  
**show** ?thesis **using** *assms* 1 *CommonSfxEqv*[of *v* *LIFT*( $\Box f$ ) *LIFT*( $f \wedge \text{wnext } (\Box f)$ )] **by** *simp*  
**qed**

**lemma** *CommonSfxBi*:  
**assumes** *basevars v*  
 $\natural[f] - v$

**shows**  $\models [bi\ f]-v$   
**using** *assms* **by** (*simp add: commonsfx-def itl-defs*) *force*

**lemma** *CommonSfxBox1*:  
**assumes** *basevars v*  
 $\models [f]-v$   
**shows**  $\models [\Box\ f]-v$   
**using** *assms* **by** (*simp add: commonsfx-def itl-defs*) *force*

**lemma** *CommonSfxAssign*:  
**assumes** *basevars (v :: state  $\Rightarrow$  nat)*  
**shows**  $\models [\$v = \#c \wedge v := \#d \wedge skip]-v$   
**proof** –  
**have** 1:  $\vdash (\$v = \#c \wedge v := \#d \wedge skip) = (\$v = \#c \wedge skip \wedge v\$ = \#d)$   
**unfolding** *next-assign-d-def* **by** *fastforce*  
**have** 2:  $\models [\$v = \#c \wedge skip \wedge v\$ = \#d]-v$   
**using** *CommonSfxUnitAssignAndNextValAndSkip[of v c d]* *assms* **by** *blast*  
**show** *?thesis* **using** *assms* 1 2  
 $CommonSfxEqv[of\ v\ LIFT(\$v = \#c \wedge v := \#d \wedge skip)\ LIFT(\$v = \#c \wedge skip \wedge v\$ = \#d)]$   
**by** *blast*  
**qed**

## 17.2.6 Backward Executability theorems

**lemma** *RStableEqvStable*:  
 $\vdash (stable\ v)^r = (stable\ v)$   
**proof** –  
**have** 1:  $\vdash (v\ gets\ \$v)^r = ba\ (skip \longrightarrow (\$v = (\$v)^r))$   
**using** *RGetsEqvBaSkipImp* **by** *blast*  
**have** 2:  $\vdash (skip \longrightarrow (\$v = (\$v)^r)) =$   
 $(skip \longrightarrow (v\$ = \$v))$   
**by** (*auto simp add: Valid-def all-rev-eq itl-defs*)  
**have** 3:  $\vdash ba\ (skip \longrightarrow (\$v = (\$v)^r)) =$   
 $ba\ (skip \longrightarrow (v\$ = \$v))$   
**using** 2 **by** (*simp add: BaEqvBa*)  
**have** 4:  $\vdash ba\ (skip \longrightarrow (v\$ = \$v)) = (v\ gets\ \$v)$   
**using** *BaEqvBtBi GetsAlways* **by** *fastforce*  
**show** *?thesis* **unfolding** *stable-d-def* **using** 1 3 4 **by** *fastforce*  
**qed**

**lemma** *RStableInitialEqvStableFin*:  
 $\vdash (\$v = \#c \wedge stable\ v)^r = (!v = \#c \wedge stable\ v)$

**proof** –  
**have** *f1*:  $\vdash (!v = \#c) = (\$v = \#c)^r$   
**by** (*simp add: all-rev-eq(1) all-rev-eq(3) all-rev-eq(8)*)  
**have**  $\vdash (stable\ v)^r = (stable\ v)$   
**by** (*simp add: RStableEqvStable int-eq*)  
**then show** *?thesis*

using f1 RAnd by fastforce  
qed

**lemma** *StableInitialEqvStableFin*:  
 $\vdash (\$v = \#c \wedge \text{stable } v) = (!v = \#c \wedge \text{stable } v)$   
**by** (simp add: Valid-def itl-defs)  
(metis order-refl)

**lemma** *BExecStableInitial*:  
**assumes** basevars v  
**shows**  $\vdash [\$v = \#c \wedge \text{stable } v] \text{-} v$   
**proof** –  
**have** 1:  $\text{sat } (\$v = \#c \wedge \text{stable } v)$   
**using** assms **by** (simp add: sat-d-def commonpfx-def itl-defs basevars-def)  
(meson len0help)  
**have** 2:  $\ddagger [ \$v = \#c \wedge \text{stable } v ] \text{-} v$   
**by** (meson FExecStableInitial assms fexec-def)  
**have** 3:  $\natural [ \$v = \#c \wedge \text{stable } v ] \text{-} v$   
**using** 2  
**by** (metis RStableInitialEqvStableFin RevCommonSfxPfx StableInitialEqvStableFin assms int-eq)  
**show** ?thesis **unfolding** bexec-def **using** 1 3 **by** auto  
qed

**lemma** *RAlwaysConstantEqvAlwaysConstant*:  
 $\vdash (\Box(\$v = \#c))^r = \Box(\$v = \#c)$   
**by** (auto simp add: Valid-def itl-defs reverse-d-def irev-inth)  
(metis diff-diff-cancel diff-le-self )

**lemma** *BExecAlwaysConstant*:  
**assumes** basevars v  
**shows**  $\vdash [\Box(\$v = \#c)] \text{-} v$   
**proof** –  
**have** 1:  $\text{sat } \Box(\$v = \#c)$   
**using** assms **by** (simp add: sat-d-def commonpfx-def itl-defs basevars-def)  
(metis len0help)  
**have** 2:  $\ddagger [\Box(\$v = \#c)] \text{-} v$   
**by** (meson FExecAlwaysConstant assms fexec-def)  
**have** 3:  $\natural [\Box(\$v = \#c)] \text{-} v$   
**by** (meson 2 RAlwaysConstantEqvAlwaysConstant RevCommonSfxPfx CommonSfxEqv assms)  
**show** ?thesis **unfolding** bexec-def **using** 1 3 **by** auto  
qed

**lemma** *NotBExecGetsAndMore*:  
**assumes** basevars v  
**shows**  $\neg (\vdash [\$v = \#(0::\text{nat}) \wedge \text{more} \wedge v \text{ gets } (\$v + \#1)] \text{-} v)$   
**proof** –  
**have** 10:  $\text{sat } \$v = \#0 \wedge \text{more} \wedge v \text{ gets } (\$v + \#1)$

```

using assms by (simp add: sat-d-def itl-defs)
  (metis inth.simps(1) basevars inth-Suc inth-zero ilen.simps(1) ilen.simps(2) neq0-conv
    not-less-eq plus-1-eq-Suc)
have 20:  $\neg(\mathbb{I}[\$v = \#0 \wedge \text{more} \wedge v \text{ gets } (\$v + \#1)] - v)$ 
using assms
using NotCommonSfxGetsAndMore by auto
show ?thesis unfolding bexec-def using 10 20 by simp
qed

```

```

lemma BExecGetsAndEmpty:
assumes basevars v
shows  $\mathbb{I}[\$v = \#(0::nat) \wedge \text{empty} \wedge v \text{ gets } (\$v + \#1)] - v$ 
proof –
  have 10: sat  $\$v = \#0 \wedge \text{empty} \wedge v \text{ gets } (\$v + \#1)$ 
  using assms by (simp add: sat-d-def itl-defs)
    (metis baseE ilen.simps(1) inth.simps(1) not-less-zero)
  have 20:  $\mathbb{I}[\$v = \#0 \wedge \text{empty} \wedge v \text{ gets } (\$v + \#1)] - v$ 
  using assms
  by (auto simp add: commonsfx-def itl-defs imap-eq-inth-ilen )
show ?thesis unfolding bexec-def using 10 20 by simp
qed

```

```

lemma BExecChopFixedExists:
assumes basevars v
   $\mathbb{I}[f] - v$ 
   $\mathbb{I}[g] - v$ 
   $\vdash \text{fin } w \longrightarrow f$ 
   $\vdash g \longrightarrow \text{init}(w)$ 
shows  $(\exists k. (\mathbb{I}[f; (g \wedge \text{len } k)] - v))$ 
using assms SatChopR[of f g w]
unfolding bexec-def
using SatExistsChopR CommonSfxChopFixed
by metis

```

```

lemma BExecPrevRule:
assumes basevars v
   $\mathbb{I}[g] - v$ 
   $\mathbb{I}[\text{fin } w \wedge \text{empty}] - v$ 
shows  $\mathbb{I}[\text{fin } w \wedge \text{prev } g] - v$ 
proof –
  have 1:  $\dagger[g^r] - v$ 
  using RevCommonPfxSfx SatRev assms fexec-def bexec-def by blast
  have 2:  $\dagger[(\text{init } w \wedge \text{empty})^r] - v$ 
  by (metis RAndEmptyEqvAndEmpty REmptyEqvEmpty RInitEqvFin RevCommonPfxSfx all-rev-eq(3)
    assms(1) assms(3) fexec-def bexec-def inteq-reflection)
  have 3:  $\dagger[(\text{init } w) \wedge \text{empty}] - v$ 
  using 2 RAndEmptyEqvAndEmpty SatRev CommonPfxEqv assms(1) fexec-def by blast

```

```

have 4:  $\dagger[(init\ w) \wedge (\bigcirc (g^r))]-v$ 
by (simp add: 1 3 FExecNextRule assms(1))
have 5:  $\dagger[(fin\ w \wedge prev\ g)^r]-v$ 
by (metis 4 RFinEqvInit RPrevEqvNext all-rev-eq(3) inteq-reflection)
show ?thesis
using 5 RevCommonPfxSfx SatRev assms(1) fexec-def bexec-def by blast
qed

```

In Tempura we have unfortunately no rules for backward execution. But we can define a mirror image of lemma FExecWNextRule. So we first generate the last state of the interval and then proceed to determine the previous states if there are any.

**lemma** BExecWPrevRule:

```

assumes basevars v
           $\mathfrak{b}[g]-v$ 
           $\mathfrak{b}[fin\ w \wedge empty]-v$ 
shows  $\mathfrak{b}[fin\ w \wedge wprev\ g]-v$ 
proof –
have 1:  $\vdash (fin\ w \wedge wprev\ g) = ((fin\ w \wedge empty) \vee (fin\ w \wedge prev\ g))$ 
using WprevEqvEmptyOrPrev by fastforce
have 2:  $sat\ fin\ w \wedge wprev\ g$ 
using 1 SatOr assms BExecPrevRule[of v g w] unfolding bexec-def
by (metis int-eq)
have 3:  $\mathfrak{b}[fin\ w \wedge prev\ g]-v$ 
using assms BExecPrevRule[of v g w] unfolding bexec-def
by blast
have 5:  $\mathfrak{b}[fin\ w \wedge wprev\ g]-v$ 
using assms 1 3 AndFinEqvChopAndEmpty[of LIFT(prev g) w] unfolding bexec-def commonsfx-def
by (simp add: itl-defs)
    (metis diff-self-eq-0 gr-implies-not0 le-less neq0-conv suffix-ilen suffix-ilen-last suffix-zero)
show ?thesis unfolding bexec-def using 2 5 by auto
qed

```

**lemma** BExecStrengthen:

```

assumes basevars v
           $sat\ f \wedge g$ 
           $\mathfrak{b}[f]-v$ 
shows  $\mathfrak{b}[f \wedge g]-v$ 
using assms unfolding bexec-def commonsfx-def
by simp

```

**lemma** RevBExecFExec:

```

assumes basevars v
           $\vdash f = f^r$ 
shows  $(\mathfrak{b}[f]-v) = (\dagger[f]-v)$ 
proof –
have 1:  $(sat\ f) = (sat\ f^r)$ 
using SatRev assms(1) by blast
have 2:  $(\mathfrak{b}[f]-v) = (\dagger[f]-v)$ 
using RevCommonPfxSfx assms(1) assms(2) inteq-reflection by force
show ?thesis unfolding bexec-def fexec-def using 1 2 by auto

```

qed

### 17.3 Reversing bad computations

The following lemmas gives the conditions necessary to “undo” a bad computation.

**lemma** *FExecChopLenKRev*:

**assumes** *basevars v*

$\text{sat } f \wedge \text{len } k$

$\ddagger[f]\text{-}v$

$\natural[f]\text{-}v$

**shows**  $\ddagger[(f \wedge \text{len } k); (f^r \wedge \text{len } k)]\text{-}v$

**proof** –

**have** 1:  $\vdash (f \wedge \text{len } k)^r = (f^r \wedge \text{len } k)$

**by** (*simp add: intI len-defs reverse-d-def*)

**have** 2:  $\text{sat } (f^r \wedge \text{len } k)$

**by** (*metis 1 SatRev assms(1) assms(2) int-eq*)

**obtain** *s0* **where** 3:  $s0 \models (f \wedge \text{len } k)$

**using** *assms(2) sat-d-def* **by** *blast*

**obtain** *s1* **where** 4:  $(s1 \models (f^r \wedge \text{len } k)) \wedge \text{ilast } s0 = \text{ifirst } s1$

**using** 2

**by** (*metis 1 3 RRAAnd all-rev-eq(3) ifirst-irev int-eq reverse-d-def*)

**have** 5:  $\text{fuse } s0 \ s1 \models (f \wedge \text{len } k); (f^r \wedge \text{len } k)$

**by** (*meson 3 4 chop-fuse*)

**have** 6:  $\text{sat } (f \wedge \text{len } k); (f^r \wedge \text{len } k)$

**using** 5 *sat-d-def* **by** *auto*

**have** 7:  $\ddagger[f \wedge \text{len } k]\text{-}v$

**using** *FExecStrengthen assms fexec-def* **by** *blast*

**have** 8:  $\ddagger[f^r \wedge \text{len } k]\text{-}v$

**using** 2 *FExecStrengthen RevCommonPfxSfx assms fexec-def* **by** *blast*

**have** 9:  $\ddagger[(f \wedge \text{len } k); (f^r \wedge \text{len } k)]\text{-}v$

**using** 7 8

**proof** (*auto simp add: itl-defs min-def Valid-def reverse-d-def interval-prefix-suffix-help commonpfx-def*)

**fix** *s1* :: *state interval*

**fix** *s* :: *state interval*

**assume** *a0*:  $\forall s1 \ s. f \ s1 \wedge \text{ilen } s1 = k \wedge f \ s \wedge \text{ilen } s = k \wedge \text{ilen } s \leq \text{ilen } s1$

$\longrightarrow \text{imap } v \ s = \text{imap } v \ (\text{prefix } k \ s1)$

**assume** *a1*:  $\forall s1 \ s. f \ (\text{irev } s1) \wedge \text{ilen } s1 = k \wedge f \ (\text{irev } s) \wedge \text{ilen } s = k \wedge \text{ilen } s \leq \text{ilen } s1$

$\longrightarrow \text{imap } v \ s = \text{imap } v \ (\text{prefix } k \ s1)$

**assume** *a2*:  $\text{ilen } s \leq \text{ilen } s1$

**assume** *a3*:  $k \leq \text{ilen } s$

**assume** *a4*:  $f \ (\text{prefix } k \ s1)$

**assume** *a5*:  $f \ (\text{prefix } k \ s)$

**assume** *a6*:  $f \ (\text{irev } (\text{suffix } k \ s1))$

**assume** *a7*:  $\text{ilen } s1 - k = k$

**assume** *a8*:  $f \ (\text{irev } (\text{suffix } k \ s))$

**assume** *a9*:  $\text{ilen } s - k = k$

**show**  $\text{imap } v \ s = \text{imap } v \ (\text{prefix } (\text{ilen } s) \ s1)$

**proof** –

**have** 91:  $\text{imap } v \ (\text{prefix } k \ s) = \text{imap } v \ (\text{prefix } k \ s1)$

```

    using a0 a3 a4 a5 a7 by force
have 92:  $\text{imap } v (\text{suffix } k \ s) = \text{imap } v (\text{prefix } k (\text{suffix } k \ s1))$ 
    by (simp add: a1 a6 a7 a8 a9)
have 93:  $\text{fuse } (\text{prefix } k \ s) (\text{suffix } k \ s) = s$ 
    by (simp add: a3 fuse-prefix-suffix)
have 94:  $\text{fuse } (\text{prefix } k \ s1) (\text{suffix } k \ s1) = s1$ 
    using a2 a3 fuse-prefix-suffix le-trans by blast
have 95:  $\text{imap } v (\text{fuse } (\text{prefix } k \ s) (\text{suffix } k \ s)) =$ 
     $\text{fuse } (\text{imap } v (\text{prefix } k \ s)) (\text{imap } v (\text{suffix } k \ s))$ 
    by (simp add: a3 fuse-prefix-suffix imap-prefix imap-suffix)
have 96:  $\text{fuse } (\text{imap } v (\text{prefix } k \ s)) (\text{imap } v (\text{suffix } k \ s)) =$ 
     $\text{fuse } (\text{imap } v (\text{prefix } (\text{ilen } (\text{prefix } k \ s)) (\text{prefix } k \ s1)))$ 
     $(\text{imap } v (\text{prefix } (\text{ilen } (\text{suffix } k \ s)) (\text{suffix } k \ s1)))$ 
    using 91 92 a3 a9 prefix-ilen-good by fastforce
have 97:  $(\text{fuse } (\text{imap } v (\text{prefix } (\text{ilen } (\text{prefix } k \ s)) (\text{prefix } k \ s1))))$ 
     $(\text{imap } v (\text{prefix } (\text{ilen } (\text{suffix } k \ s)) (\text{suffix } k \ s1)))) =$ 
     $(\text{fuse } (\text{prefix } (\text{ilen } (\text{prefix } k \ s)) (\text{imap } v (\text{prefix } k \ s1))))$ 
     $(\text{prefix } (\text{ilen } (\text{suffix } k \ s)) (\text{imap } v (\text{suffix } k \ s1))))$ 
    by (metis 91 a3 a7 a9 imap-prefix order-refl pref-pref prefix-ilen-good suffix-ilen)
have 98:  $(\text{fuse } (\text{prefix } (\text{ilen } (\text{prefix } k \ s)) (\text{imap } v (\text{prefix } k \ s1))))$ 
     $(\text{prefix } (\text{ilen } (\text{suffix } k \ s)) (\text{imap } v (\text{suffix } k \ s1)))) =$ 
     $(\text{prefix } (\text{ilen } (\text{fuse } (\text{prefix } k \ s) (\text{suffix } k \ s))))$ 
     $(\text{fuse } (\text{imap } v (\text{prefix } k \ s1)) (\text{imap } v (\text{suffix } k \ s1))))$ 
    by (metis 91 92 95 a7 ilen-imap prefix-ilen suffix-ilen)
have 99:  $(\text{prefix } (\text{ilen } (\text{fuse } (\text{prefix } k \ s) (\text{suffix } k \ s))))$ 
     $(\text{fuse } (\text{imap } v (\text{prefix } k \ s1)) (\text{imap } v (\text{suffix } k \ s1)))) =$ 
     $(\text{imap } v (\text{prefix } (\text{ilen } s) s1))$ 
    by (metis a2 a3 fuse-prefix-suffix ilen-imap imap-prefix imap-suffix le-trans)
show ?thesis using 93 95 96 97 98 99 by auto
qed
qed
show ?thesis by (simp add: 6 9 fexec-def)
qed

```

**lemma** *FExecRevChopLenKRev*:

**assumes** *basevars v*

$\text{sat } f \wedge \text{len } k$

$\ddagger[f]-v$

$\natural[f]-v$

**shows**  $\ddagger[(f^r \wedge \text{len } k); (f \wedge \text{len } k)]-v$

**proof** –

**have** 1:  $\vdash (f \wedge \text{len } k)^r = (f^r \wedge \text{len } k)$

**by** (simp add: intI len-defs reverse-d-def)

**have** 2:  $\text{sat } f^r \wedge \text{len } k$

**by** (metis 1 SatRev assms(1) assms(2) int-eq)

**have** 3:  $\ddagger[f^r]-v$

**using** *RevCommonPfxSfx assms* **by** blast

**have** 4:  $\natural[f^r]-v$

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

**show** ?thesis **using** assms 1 2 3 4

by (metis EquReverseReverse FExecChopLenKRev int-eq)  
qed

**lemma** BExecChopLenKRev:

**assumes** basevars  $v$

$\text{sat } f \wedge \text{len } k$

$\ddagger[f]-v$

$\natural[f]-v$

**shows**  $\vdash [(f \wedge \text{len } k); (f^r \wedge \text{len } k)]-v$

**using** *assms*

**proof** –

**have** 1:  $\vdash (f \wedge \text{len } k)^r = (f^r \wedge \text{len } k)$

by (simp add: intI len-defs reverse-d-def)

**have** 2:  $\text{sat } (f^r \wedge \text{len } k)$

by (metis 1 SatRev assms(1) assms(2) int-eq)

**obtain**  $s0$  **where** 3:  $s0 \models (f \wedge \text{len } k)$

**using** *assms(2) sat-d-def* **by** *blast*

**obtain**  $s1$  **where** 4:  $(s1 \models (f^r \wedge \text{len } k)) \wedge \text{ilast } s0 = \text{ifirst } s1$

**using** 2

by (metis 1 3 RRAAnd all-rev-eq(3) ifirst-irev int-eq reverse-d-def)

**have** 5:  $\text{fuse } s0 \ s1 \models (f \wedge \text{len } k); (f^r \wedge \text{len } k)$

by (meson 3 4 chop-fuse)

**have** 6:  $\text{sat } (f \wedge \text{len } k); (f^r \wedge \text{len } k)$

**using** 5 *sat-d-def* **by** *auto*

**have** 7:  $\ddagger[(f \wedge \text{len } k); (f^r \wedge \text{len } k)]-v$

**using** *assms FExecChopLenKRev[of v f k]* **unfolding** *fexec-def* **by** *blast*

**have** 8:  $\vdash ((f \wedge \text{len } k); (f^r \wedge \text{len } k))^r = (f \wedge \text{len } k); (f^r \wedge \text{len } k)$

by (metis 1 LeftChopEqvChop RRAAnd all-rev-eq(13) all-rev-eq(3) inteq-reflection)

**have** 9:  $\natural[(f \wedge \text{len } k); (f^r \wedge \text{len } k)]-v$

**using** 7 8 *RevCommonPfxSfx assms(1) int-eq* **by** *force*

**show** ?thesis **by** (simp add: 6 9 *bexec-def*)

qed

**lemma** BExecRevChopLenKRev:

**assumes** basevars  $v$

$\text{sat } f \wedge \text{len } k$

$\ddagger[f]-v$

$\natural[f]-v$

**shows**  $\vdash (f^r \wedge \text{len } k); (f \wedge \text{len } k)]-v$

**proof** –

**have** 1:  $\vdash (f \wedge \text{len } k)^r = (f^r \wedge \text{len } k)$

by (simp add: intI len-defs reverse-d-def)

**have** 2:  $\text{sat } f^r \wedge \text{len } k$

by (metis 1 SatRev assms(1) assms(2) int-eq)

**have** 3:  $\ddagger[f^r]-v$

**using** *RevCommonPfxSfx assms* **by** *blast*

**have** 4:  $\natural[f^r]-v$

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

**show** ?thesis **using** *assms 1 2 3 4*

**by** (metis BExecChopLenKRev EquReverseReverse inteq-reflection)



qed

end

## References

- [1] A. Cau, B. Moszkowski, and D. Smallwood. The deep embedding of ITL in Isabelle/HOL, 2018. <http://antonio-cau.co.uk/ITL/itlhomepages6.html>.
- [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] S. Merz. An Encoding of TLA in Isabelle. <http://www.pst.informatik.uni-muenchen.de/~merz/isabelle/>. Part of the Isabelle distribution., 1998.
- [4] B. Moszkowski. Compositional reasoning about projected and infinite time. In *Proceedings of the First IEEE Int'l Conf. on Engineering of Complex Computer Systems (ICECCS'95)*, pages 238–245. IEEE Computer Society Press, 1995. [Download pdf](#).
- [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. Compositional reasoning using intervals and time reversal. *Annals of Mathematics and Artificial Intelligence*, 71(1-3):175–250, 2014.
- [7] B. Moszkowski and D. Guelev. An application of temporal projection to interleaving concurrency. *Formal Aspects of Computing*, 29(4):705–750, July 2017.
- [8] B. C. Moszkowski. Imperative reasoning in interval temporal logic. Technical report, University of Newcastle upon Tyne, 1996.
- [9] B. C. Moszkowski. A Complete Axiom System for Propositional Interval Temporal Logic with Infinite Time. *Logical Methods in Computer Science Journal*, 8(3), 2012.
- [10] B. C. Moszkowski and D. P. Guelev. An Application of Temporal Projection to Interleaving Concurrency. In *Dependable Software Engineering: Theories, Tools, and Applications - First International Symposium, SETTA 2015, Nanjing, China, November 4-6, 2015, Proceedings*, pages 153–167, 2015.
- [11] J. S. Warford, D. Vega, and S. M. Staley. A Calculational Deductive System for Linear Temporal Logic. <https://www.cslab.pepperdine.edu/warford/Papers/Vega-Paper.pdf>, 2019.
- [12] M. Wildmoser and T. Nipkow. Certifying Machine Code Safety: Shallow versus Deep Embedding. In K. Slind, A. Bunker, and G. Gopalakrishnan, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2004)*, volume 3223 of *LNCS*, pages 305–320. Springer, 2004.