

An encoding of Interval Temporal Logic in Isabelle/HOL

Antonio Cau

Ben Moszkowski

David Smallwood

July 21, 2019

Abstract

These Isabelle theories introduce the semantics and syntax of Finite and Infinite Interval Temporal Logic (ITL). The ITL proof system, as introduced in [4, 7], 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 and Infinite ITL theorems, taken from [6], 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.

Contents

1	Finite Intervals	4
1.1	Definitions	5
1.2	Lemmas	6
1.2.1	Interval Length	6
1.2.2	Map	7
1.2.3	nth	8
1.2.4	index sequence	9
1.2.5	prefix, suffix and sub	11
1.2.6	intapp	15
1.2.7	Reverse	16
2	Infinite Intervals	18
2.1	Definitions	18
2.2	Lemmas	19
2.2.1	upt	19
2.2.2	isuffix	22
2.2.3	iprefix	22
2.2.4	subinterval	23
2.2.5	Conc	24
2.2.6	Infinite index sequence	25
3	Finite ITL Semantics	27
3.1	Types of Formulas	27
3.2	Semantics of ITL	28
3.3	Abbreviations	29

3.4	Properties of Operators	34
3.5	Soundness of Finite ITL Axioms	38
3.5.1	ChopAssoc	38
3.5.2	OrChopImp	38
3.5.3	ChopOrImp	39
3.5.4	EmptyChop	39
3.5.5	ChopEmpty	39
3.5.6	StatImpBi	39
3.5.7	NextImpNotNextNot	39
3.5.8	BiBoxChopImpChop	39
3.5.9	BoxInduct	39
3.5.10	ChopStarEqv	40
3.6	Quantification over State (Flexible) Variables	41
3.7	Temporal Quantifiers	41
4	Old vs new definition of Chopstar	42
4.1	Definition	42
4.2	Lemmas	42
5	Infinite ITL Semantics	51
5.1	Types of Formulas	51
5.2	Semantics of ITL	51
5.3	Abbreviations	53
5.4	Properties of Operators	62
5.5	Soundness Axioms	70
5.5.1	ChopAssoc	70
5.5.2	OrChopImp	72
5.5.3	ChopOrImp	72
5.5.4	EmptyChop	72
5.5.5	ChopEmpty	72
5.5.6	StatImpBi	73
5.5.7	NextImpNotNextNot	73
5.5.8	BiBoxChopImpChop	73
5.5.9	BoxInduct	73
5.5.10	ChopStarEqv	74
5.5.11	OmegaUnroll	77
5.5.12	Omegalnduct	80
5.6	Quantification over State (Flexible) Variables	86
5.7	Temporal Quantifiers	87
6	Finite ITL: Axioms and Rules	87
6.1	Rules	87
6.2	Axioms	88
6.3	Additional Lemmas	89
6.4	Quantification	89
6.5	Lemmas about <i>current-val</i>	90
6.6	Lemmas about <i>next-val</i>	90
6.7	Lemmas about <i>fin-val</i>	91

6.8	Lemmas about <i>penult-val</i>	91
6.9	Basic temporal variables properties	92
7	Infinite ITL: Axioms and Rules	92
7.1	Rules	93
7.2	Axioms	93
7.3	Additional Lemmas	94
7.4	Quantification	95
7.5	Lemmas about <i>current-val</i>	95
7.6	Lemmas about <i>next-val</i>	96
7.7	Lemmas about <i>fin-val</i>	96
7.8	Lemmas about <i>penult-val</i>	97
7.9	Basic temporal variables properties	98
8	Finite ITL theorems	98
8.1	Propositional reasoning	98
8.2	State formulas	100
8.3	Basic Theorems	100
8.4	Further Properties Di and Bi	113
8.5	Properties of Da and Ba	118
8.6	Properties of Fin	126
8.7	Properties of Chopstar and Chopplus	147
8.8	Properties of While	163
8.9	Properties of Halt	168
8.10	Properties of Groups of chops	173
9	Infinite ITL theorems using Weak Chop	173
9.1	Propositional reasoning	174
9.2	State formulas	175
9.3	finite and inf properties	175
9.4	Basic Theorems	178
9.5	Further Properties Di and Bi	191
9.6	Properties of Da and Ba	197
9.7	Properties of Fin	205
9.8	Properties of Chopstar and Chopplus	231
9.9	Properties of Omega	259
9.10	Properties of While	264
9.11	Properties of Halt	272
9.12	Properties of Groups of chops	278
10	Infinite ITL theorems using strong chop	278
10.1	Strong Chop axioms	278
10.2	ITL operators in terms of SChop	280
10.3	Basic Theorems	281
10.4	Further Properties Df and Bf	287
10.5	Properties of SDa and SBa	295
10.6	Properties of SFin	304
10.7	Properties of SChopstar and SChopplus	329

10.8 Properties of Omega	351
10.9 Properties of SWhile	352
10.10 Properties of Halt	357
10.11 Properties of Groups of strong chops	361
11 First Order Finite ITL theorems	362
12 Time Reversal	368
12.1 Definition	368
12.2 Time reversal Rules	368
12.3 Properties of Time Reversal	371
13 First Order Infinite ITL theorems	380
14 The First Occurrence Operator in finite ITL	386
14.1 Definitions	386
14.1.1 Definitions Strict Initial and Final	386
14.1.2 Definition First and Last Operators	387
14.2 First and Time Reversal	388
14.3 Semantic Theorems	390
14.3.1 Semantics First and Last Operators	390
14.3.2 Various Semantic Lemmas	392
14.4 Theorems	394
14.4.1 Fixed length intervals	394
14.4.2 Additional ITL theorems	400
14.4.3 Strict initial intervals	409
14.4.4 First occurrence	418
15 Monitors	444
15.1 Syntax	444
15.2 Derived Monitors	445
15.3 Monitor Laws	448
16 Finite ITL Examples	469
16.1 Example 1	469
16.2 Example 2	469
16.3 Example 3	471
16.4 Example 4	473
16.5 Example 5	474
17 Monitor Example	474

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: *length* (*intlen*), *prefix*, *suffix*, *sub*, *nth*, *intfirst*, *intlast*, *intapp* and *intrev*.

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.

1.1 Definitions

```
datatype (set: 'a) interval =
  St 'a ([_])
| Cons 'a 'a interval (infixr  $\odot$  65)
for
  map: map
  rel: interval-all2
  pred: interval-all
```

```
type-synonym index = nat interval
```

```
syntax
— interval Enumeration
-interval :: args => 'a interval  ((-))
```

```
translations
⟨x, xs⟩ == x  $\odot$  ⟨xs⟩
⟨x⟩ == [x]
```

```
primrec (nonexhaustive) intlen :: 'a interval  $\Rightarrow$  nat where
  intlen ⟨x⟩ = 0
| intlen (x  $\odot$  xs) = 1 + (intlen xs)
```

```
primrec (nonexhaustive) nth :: 'a interval  $\Rightarrow$  nat  $\Rightarrow$  'a where
  nth ⟨x⟩ n = x
| nth (x  $\odot$  xs) n = (case n of 0  $\Rightarrow$  x | Suc k  $\Rightarrow$  nth xs k)
```

```
primrec prefix :: nat  $\Rightarrow$  'a interval  $\Rightarrow$  'a interval where
  prefix n ⟨x⟩ = ⟨x⟩
| prefix n (x  $\odot$  xs) = (case n of 0  $\Rightarrow$  ⟨x⟩ | Suc m  $\Rightarrow$  x  $\odot$  (prefix m xs))
```

```
primrec suffix :: nat  $\Rightarrow$  'a interval  $\Rightarrow$  'a interval where
  suffix n ⟨x⟩ = ⟨x⟩
| 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* = (if *k* < *n* then *prefix* 0 (*suffix* *n* *xs*)
 else *prefix* (*k* - *n*) (*suffix* *n* *xs*)
)

primrec *intfirst* :: '*a* *interval* \Rightarrow '*a* **where**
intfirst $\langle x \rangle$ = *x*
| *intfirst* (*Cons* *x* -) = *x*

primrec *intl* :: '*a* *interval* \Rightarrow '*a* **where**
intl $\langle x \rangle$ = *x*
| *intl* (*Cons* - *xs*) = *intl* *xs*

primrec *intapp* :: '*a* *interval* \Rightarrow '*a* *interval* \Rightarrow '*a* *interval* (**infixr** \odot 65) **where**
intapp-St: $\langle x \rangle \odot ys = x \odot ys$ |
intapp-Cons: $(x \odot xs) \odot ys = x \odot (xs \odot ys)$

primrec *intrev* :: '*a* *interval* \Rightarrow '*a* *interval* **where**
intrev $\langle x \rangle$ = $\langle x \rangle$
| *intrev* ($x \odot xs$) = (*intrev* *xs*) $\odot \langle x \rangle$

definition *index-sequence* :: *nat* \Rightarrow *index* \Rightarrow *bool* **where**
index-sequence *x* *idx* \equiv (*nth* *idx* 0 = *x*) \wedge (\forall *n*. *n* < *intlen* *idx* \longrightarrow *nth* *idx* *n* < *nth* *idx* (*Suc* *n*))

definition *shift* :: *nat* \Rightarrow *nat* \Rightarrow *nat* **where**
shift *k* = (λ *x*. *x* + *k*)

definition *shifm* :: *nat* \Rightarrow *nat* \Rightarrow *nat* **where**
shifm *k* = (λ *x*. (if *k* > *x* then 0 else (*x* - *k*)))

1.2 Lemmas

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

1.2.1 Interval Length

lemma *interval-intlen-gr-zero* [*simp*]:
intlen *xs* \geq 0
by *auto*

lemma *interval-intlen-cons* [*simp*]:
(*intlen* ($x \odot xs$)) = (*intlen* *xs*) + 1
by *simp*

lemma *interval-intlen-cons-1* :
intlen *I* > 0 \longleftrightarrow (\exists *x* *ls*. *I* = $x \odot ls$)
by (*induct* *I*) *simp-all*

lemma *interval-intlen-map* [*simp*]:

$\text{intlen } (\text{map } f \text{ } xs) = \text{intlen } xs$
by $(\text{induct } xs) \text{ simp-all}$

lemma *interval-intlen-intapp* [simp]:
 $\text{intlen } (xs \ominus ys) = (\text{intlen } xs) + (\text{intlen } ys) + 1$
by $(\text{induct } xs \text{ arbitrary: } ys) \text{ simp-all}$

lemma *interval-intrev-intlen* [simp]:
 $\text{intlen } (\text{intrev } xs) = \text{intlen } xs$
by $(\text{induct } xs, \text{ simp}, \text{ simp})$

1.2.2 Map

lemma *map-ext*:
 $(\bigwedge x. x: \text{set } xs \longrightarrow f \ x = g \ x) \Longrightarrow \text{map } f \ xs = \text{map } g \ xs$
by $(\text{induct } xs) \text{ simp-all}$

lemma *map-ident* [simp]:
 $\text{map } (\lambda x. x) = (\lambda xs. xs)$
by $(\text{rule ext}, \text{induct-tac } xs) \text{ auto}$

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

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

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

lemma *intrev-map*:
 $\text{intrev } (\text{map } f \ xs) = \text{map } f \ (\text{intrev } xs)$
by $(\text{induct } xs) \text{ auto}$

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

lemma *map-cong*:
 $xs = ys \Longrightarrow (\bigwedge x. x \in \text{set } ys \Longrightarrow f \ x = g \ x) \Longrightarrow \text{map } f \ xs = \text{map } g \ ys$
by *simp*

lemma *map-injective*:
 $\text{map } f \ xs = \text{map } f \ ys \Longrightarrow \text{inj } f \Longrightarrow xs = ys$
by $(\text{meson injD interval.inj-map})$

lemma *inj-map-eq-map* [simp]:

inj f \implies (*map f xs* = *map f ys*) = (*xs* = *ys*)
by (*blast dest: map-injective*)

lemma *inj-mapI*:
inj f \implies *inj* (*map f*)
using *interval.inj-map* **by** *blast*

lemma *inj-mapD*:
inj (*map f*) \implies *inj f*
by (*metis inj-def interval.map(1) interval.simps(1)*)

lemma *inj-map[iff]*:
inj (*map f*) = *inj f*
by (*blast dest: inj-mapD intro: inj-mapI*)

lemma *map-idl*: ($\bigwedge x. x \in \text{set } xs \implies f\ x = x$) \implies *map f xs* = *xs*
by (*induct xs, auto*)

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

declare *map.id* [*simp*]

1.2.3 nth

lemma *interval-nth-zero* [*simp*]:
 $\text{nth } (x \odot xs) \ 0 = x$
by *simp*

lemma *interval-nth-Suc* [*simp*]:
 $\text{nth } (x \odot xs) \ (\text{Suc } n) = \text{nth } xs \ n$
by *auto*

lemma *interval-nth-last*:
 $\text{nth } (x \odot xs) \ (\text{intlen } (x \odot xs)) = \text{nth } xs \ (\text{intlen } xs)$
by *simp*

lemma *interval-nth-cons*:
assumes $0 < i \wedge i < 1 + \text{intlen}(xs)$
shows $\text{nth}(x \odot xs) \ i = \text{nth } xs \ (i - 1) \wedge$
 $\text{nth}(x \odot xs) \ (i + 1) = \text{nth } xs \ ((i - 1) + 1)$
by (*metis One-nat-def Suc-lel add commute assms interval-nth-Suc le-add-diff-inverse2 plus-1-eq-Suc*)

lemma *interval-nth-zero-intfirst*:
 $\text{nth } xs \ 0 = \text{intfirst } xs$
by (*induct xs*) *simp-all*

lemma *interval-nth-intlen-intlast*:
 $\text{nth } xs \ (\text{intlen } xs) = \text{intlast } xs$
by (*induct xs*) *simp-all*

lemma *interval-st-intlen* :

$$(xs = \langle x \rangle) \longleftrightarrow \text{intlen } xs = 0 \wedge \text{nth } xs \ 0 = x$$

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

lemma *interval-eq-nth-eq* :

$$(xs = ys) = (\text{intlen } xs = \text{intlen } ys \wedge (\forall i \leq \text{intlen } xs. \text{nth } xs \ i = \text{nth } ys \ i))$$

by (*induct xs arbitrary: ys, metis interval-st-intlen le-numeral-extra(3),*
(*case-tac ys, simp, fastforce*))

lemma *interval-nth-map* :

$$\text{nth } (\text{map } f \ xs) \ i = f \ (\text{nth } xs \ i)$$

by (*induct xs arbitrary: i, simp, (case-tac i, simp, simp)*)

1.2.4 index sequence

lemma *interval-idx-less*:

assumes *iseq: index-sequence x idx*

shows $(n < \text{intlen } idx \wedge n+k < \text{intlen } idx) \longrightarrow \text{nth } idx \ n < \text{nth } idx \ (\text{Suc}(n+k))$

using *index-sequence-def iseq* **by** (*induct k, simp, auto*)

lemma *interval-idx-less-last* :

assumes *index-sequence x idx*

shows $(i < \text{intlen } idx \wedge i + (\text{intlen } idx - (i+1)) < \text{intlen } idx) \longrightarrow \text{nth } idx \ i < \text{nth } idx \ (\text{Suc}(i + (\text{intlen } idx - (i+1))))$

using *assms interval-idx-less* **by** *blast*

lemma *interval-idx-less-last-1*:

assumes *index-sequence x idx*

shows $i < \text{intlen } idx \longrightarrow \text{nth } idx \ i < \text{nth } idx \ (\text{intlen } idx)$

using *assms interval-idx-less-last* **by** *auto*

lemma *interval-idx-greater-first*:

assumes *index-sequence x idx*

shows $(i > 0 \wedge i \leq \text{intlen } idx) \longrightarrow x < \text{nth } idx \ i$

using *assms* **by** (*induct i, simp,*
metis One-nat-def Suc-le-lessD add-Suc index-sequence-def interval-idx-less
less-le-trans plus-1-eq-Suc)

lemma *interval-idx-cons*:

$$\text{index-sequence } 0 \ (x \odot ls) =$$

$$(x=0 \wedge x < \text{nth } ls \ 0 \wedge \text{index-sequence } (\text{nth } ls \ 0) \ ls)$$

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

lemma *interval-idx-shift-mono*:

$$\text{mono } (\text{shift } k)$$

by (*simp add: Interval.shift-def mono-def*)

lemma *interval-idx-expand*:

assumes $\text{index-sequence } 0 \mid \wedge (\text{nth } l (\text{intlen } l)) = (\text{intlen } xs) \wedge 0 \leq i \wedge i < (\text{intlen } l)$
shows $0 \leq (\text{nth } l i) \wedge (\text{nth } l i) \leq (\text{nth } l (i+1)) \wedge (\text{nth } l (i+1)) \leq (\text{intlen } xs)$
using *assms*
by (*simp add: index-sequence-def*,
(induct l, simp, metis Suc-less1 eq-imp-le interval-idx-less-last-1 less-imp-le-nat))

lemma *interval-idx-shift-idx* [*simp*]:
 $(\text{index-sequence } (x+k) (\text{map } (\text{shift } k) \text{ idx})) = (\text{index-sequence } x \text{ idx})$
by (*simp add: Interval.shift-def index-sequence-def interval-nth-map*)

lemma *interval-idx-shiftm* :
assumes $(\text{index-sequence } k (\text{lsk}) \wedge ls = \text{map } (\text{shiftm } k) \text{ lsk})$
shows $\text{index-sequence } 0 (ls) \wedge (\text{intlen } ls) = (\text{intlen } \text{lsk})$
using *assms*
by (*simp add: interval-eq-nth-eq index-sequence-def shift-def shiftm-def interval-nth-map*)
(smt Suc-le1 diff-less-mono index-sequence-def interval-idx-greater-first interval-intlen-map
le-less-trans less-Suc-eq-0-disj not-less order.asym)

lemma *interval-lsk-ls* :
 $(\text{index-sequence } k (\text{lsk}) \wedge \text{lsk} = \text{map } (\text{shift } k) ls \wedge \text{index-sequence } 0 (ls)) =$
 $(\text{index-sequence } k (\text{lsk}) \wedge ls = \text{map } (\text{shiftm } k) \text{ lsk} \wedge \text{index-sequence } 0 (ls))$
by (*simp add: interval-eq-nth-eq index-sequence-def shift-def shiftm-def interval-nth-map*,
rule,
metis (no-types, lifting) add-diff-cancel-right' not-add-less2,
metis (no-types, lifting) Suc-eq-plus1 add commute add-cancel-right-left add-diff-inverse-nat
ex-least-nat-less le-SucE le-zero-eq not-less-zero order-refl)

lemma *interval-idx-link-shiftm*:
 $(\text{index-sequence } k (\text{lsk}) \wedge ls = \text{map } (\text{shiftm } k) \text{ lsk}) =$
 $(\text{index-sequence } k (\text{lsk}) \wedge ls = \text{map } (\text{shiftm } k) \text{ lsk} \wedge$
 $\text{index-sequence } 0 (ls) \wedge (\text{intlen } ls) = (\text{intlen } \text{lsk}))$
using *interval-idx-shiftm by blast*

lemma *interval-idx-link*:
 $(\text{lsk} = \text{map } (\text{shift } k) ls \wedge \text{index-sequence } 0 (ls)) =$
 $(\text{lsk} = \text{map } (\text{shift } k) ls \wedge \text{index-sequence } k (\text{lsk}) \wedge \text{index-sequence } 0 (ls) \wedge$
 $(\text{intlen } ls) = (\text{intlen } \text{lsk}))$
by (*metis add.left-neutral interval-idx-shift-idx interval-intlen-map*)

lemma *interval-idx-bound-0* :
assumes $\text{index-sequence } 0 ls \wedge \text{Interval.nth } ls (\text{intlen } ls) = \text{intlen } (\text{suffix } k xs)$
shows $((i \leq \text{intlen } ls) \longrightarrow ((\text{nth } ls (i)) \leq (\text{intlen } (\text{suffix } k xs))))$
using *assms*
by (*metis eq-iff interval-idx-less-last-1 le-neq-implies-less less-imp-le-nat*)

lemma *interval-idx-bound-1*:
 $(\text{index-sequence } 0 (ls) \wedge (\text{nth } (ls) (\text{intlen } (ls))) = (\text{intlen } (\text{suffix } k xs))) =$
 $(\text{index-sequence } 0 (ls) \wedge (\text{nth } (ls) (\text{intlen } (ls))) = (\text{intlen } (\text{suffix } k xs)) \wedge$
 $(\forall i. (i \leq \text{intlen } ls) \longrightarrow ((\text{nth } ls (i)) \leq (\text{intlen } (\text{suffix } k xs)))))$
using *interval-idx-bound-0 by blast*

1.2.5 prefix, suffix and sub

lemma *interval-prefix-state* [simp]:

$$\text{prefix } m \langle x \rangle = \langle x \rangle$$

by *simp*

lemma *interval-prefix-suc* [simp]:

$$\text{prefix } (\text{Suc } m) (x \odot xs) = x \odot (\text{prefix } m xs)$$

by *auto*

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

$$\text{prefix } 0 (x \odot xs) = \langle x \rangle$$

by *auto*

lemma *interval-prefix-zero-intfirst* [simp]:

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

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

lemma *interval-intfirst-prefix* [simp]:

$$i \leq \text{intlen } xs \implies \text{intfirst } (\text{prefix } i xs) = \text{intfirst } xs$$

by (*induct xs arbitrary: i, auto*) (*case-tac i, auto*)

lemma *interval-prefix-intlen* [simp]:

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

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

lemma *interval-prefix-intlen-gr-1* [simp]:

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

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

lemma *interval-intlen-prefix-cons* [simp]:

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

using *interval-intlen-cons* **by** *auto*

lemma *interval-prefix-length* [code]:

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

by (*induct xs arbitrary: i, simp*) (*case-tac i, auto*)

lemma *interval-prefix-length-good* [simp]:

assumes $i \leq \text{intlen } xs$

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

using *assms* **by** (*simp add: interval-prefix-length*)

lemma *interval-prefix-length-bad* [simp] :

assumes $i > \text{intlen } xs$

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

using *assms* **by** (*simp add: interval-prefix-length*)

lemma *interval-pref-intlen-bound* :

assumes $i \leq (\text{intlen } xs)$

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

using *assms* **by** (*induct* *xs*, *simp*) (*metis* *interval-prefix-length*)

lemma *interval-suffix-length* [*code*]:

intlen (*suffix* *i* *xs*) = (if $i \leq \text{intlen } xs$ then $(\text{intlen } xs) - i$ else 0)
by (*induct* *xs* *arbitrary*: *i*, *simp*) (*case-tac* *i*, *auto*)

lemma *interval-suffix-length-good* [*simp*]:

assumes $i \leq \text{intlen } xs$
shows $\text{intlen } (\text{suffix } i \text{ } xs) = (\text{intlen } xs) - i$
using *assms* **by** (*simp* *add*: *interval-suffix-length*)

lemma *interval-suffix-length-bad* [*simp*]:

assumes $i > \text{intlen } xs$
shows $\text{intlen } (\text{suffix } i \text{ } xs) = 0$
using *assms* **by** (*simp* *add*: *interval-suffix-length*)

lemma *interval-nth-prefix* [*simp*]:

$i \leq \text{intlen } xs \wedge k \leq i \implies \text{nth } (\text{prefix } i \text{ } xs) \text{ } k = \text{nth } xs \text{ } k$
by (*induct* *xs* *arbitrary*: *i* *k*, *auto*, (*case-tac* *i*, *auto*, (*case-tac* *k*, *auto*)))

lemma *interval-nth-suffix* [*simp*]:

assumes $i \leq \text{intlen } xs \wedge k \leq \text{intlen } xs - i$
shows $\text{nth } (\text{suffix } i \text{ } xs) \text{ } k = \text{nth } xs \text{ } (i+k)$
by (*induct* *xs* *arbitrary*: *i* *k*, *auto*) (*case-tac* *i*, *auto*)

lemma *interval-suffix-prefix-help-1*:

assumes $ia+i \leq \text{intlen } xs \wedge k \leq ia$
shows $\text{nth } (\text{prefix } ia \text{ } (\text{suffix } i \text{ } xs)) \text{ } k = \text{nth } (\text{suffix } i \text{ } (\text{prefix } (ia+i) \text{ } xs)) \text{ } k$
proof –
have 1: $\text{nth } (\text{prefix } ia \text{ } (\text{suffix } i \text{ } xs)) \text{ } k = \text{nth } (\text{suffix } i \text{ } xs) \text{ } k$
using *interval-nth-prefix* *assms* **by** (*metis* *interval-prefix-intlen-gr-1* *le-cases* *le-iff-add*)
have 2: $\text{nth } (\text{suffix } i \text{ } xs) \text{ } k = \text{nth } xs \text{ } (i+k)$
using *interval-nth-suffix* *assms* **by** (*simp* *add*: *add-le-imp-le-diff*)
have 3: $\text{nth } xs \text{ } (i+k) = \text{nth } (\text{prefix } (ia+i) \text{ } xs) \text{ } (i+k)$
using *interval-nth-prefix* *assms* **by** *simp*
have 4: $\text{nth } (\text{prefix } (ia+i) \text{ } xs) \text{ } (i+k) = \text{nth } (\text{suffix } i \text{ } (\text{prefix } (ia+i) \text{ } xs)) \text{ } k$
using *interval-nth-suffix* *assms* **by** *simp*
from 1 2 3 4 **show** ?thesis **by** *auto*
qed

lemma *interval-suffix-prefix-help-2*:

assumes $ia+i \leq \text{intlen } xs$
shows $(\forall k \leq ia. \text{nth } (\text{prefix } ia \text{ } (\text{suffix } i \text{ } xs)) \text{ } k = \text{nth } (\text{suffix } i \text{ } (\text{prefix } (ia+i) \text{ } xs)) \text{ } k)$
using *interval-suffix-prefix-help-1* *assms* **by** *fastforce*

lemma *interval-suffix-prefix-help-3*:

assumes $ia+i \leq \text{intlen } xs$
shows $\text{intlen } (\text{prefix } ia \text{ } (\text{suffix } i \text{ } xs)) = \text{intlen } (\text{suffix } i \text{ } (\text{prefix } (ia+i) \text{ } xs))$
using *assms* *interval-prefix-length-good* *interval-suffix-length-good* **by** *auto*

lemma *interval-suffix-prefix-swap*:
assumes $ia+i \leq \text{intlen } xs$
shows $\text{prefix } ia (\text{suffix } i \text{ } xs) = \text{suffix } i (\text{prefix } (ia+i) \text{ } xs)$
by (*simp add: interval-eq-nth-eq interval-suffix-prefix-help-2 interval-suffix-prefix-help-3 assms*)

lemma *interval-prefix-prefix-zero* [*simp*]:
 $\text{prefix } 0 (\text{prefix } 0 \text{ } xs) = \text{prefix } 0 \text{ } xs$
by (*induct xs*) *simp-all*

lemma *interval-pref-pref* [*simp*]:
 $(\text{prefix } i (\text{prefix } i \text{ } xs)) = \text{prefix } i \text{ } xs$
by (*metis interval-prefix-intlen interval-prefix-intlen-gr-1 interval-prefix-length less-imp-add-positive not-less*)

lemma *interval-pref-pref-3* [*simp*]:
 $(\text{prefix } i (\text{prefix } (i+k) \text{ } xs)) = \text{prefix } i \text{ } xs$
by (*induct xs arbitrary: i k, simp, (case-tac i, auto, (simp add: Nitpick.case-nat-unfold))*)

lemma *interval-pref-help*:
assumes $i \leq \text{intlen } (\text{prefix } (\text{intlen } xs - \text{Suc } 0) \text{ } xs)$
shows $(\text{prefix } i (\text{prefix } (\text{intlen } xs - \text{Suc } 0) \text{ } xs)) = (\text{prefix } i \text{ } xs)$
using *assms*
by (*metis diff-le-self interval-pref-pref-3 interval-prefix-length ordered-cancel-comm-monoid-diff-class.add-diff-inverse*)

lemma *interval-pref-pref-help*:
assumes $\text{intlen } xs > 0 \wedge ia < \text{intlen } (xs)$
shows $(\text{prefix } ia (\text{prefix } (\text{intlen } xs - \text{Suc } 0) \text{ } xs)) = (\text{prefix } ia \text{ } xs)$
using *assms*
by (*metis Suc-le1 Suc-le-mono Suc-pred diff-le-self interval-pref-help interval-prefix-length-good*)

lemma *interval-pref-pref-help-1*:
assumes $i > 0 \wedge i \leq \text{intlen } xs$
shows $(\text{prefix } (\text{intlen } (\text{prefix } i \text{ } xs) - \text{Suc } 0) (\text{prefix } i \text{ } xs)) =$
 $(\text{prefix } (\text{intlen } (\text{prefix } i \text{ } xs) - \text{Suc } 0) \text{ } xs)$
using *assms interval-pref-pref-3* **by** (*metis diff-le-self interval-prefix-length-good le-iff-add*)

lemma *interval-suffix-suc* [*simp*]:
 $\text{suffix } (\text{Suc } m) (x \odot xs) = \text{suffix } m \text{ } xs$
by *auto*

lemma *interval-suffix-zero* [*simp*]:
 $\text{suffix } 0 \text{ } xs = xs$
by (*induct xs*) *simp-all*

lemma *interval-suffix-intlen* [*simp*]:
 $\text{suffix } (\text{intlen } xs) \text{ } xs = \langle (\text{nth } xs (\text{intlen } xs)) \rangle$
by (*induct xs*) *simp-all*

lemma *interval-suffix-intlast* [*simp*]:

$\text{suffix } (\text{intlen } xs) \text{ } xs = \langle \text{intlast } xs \rangle$
by (*induct xs*) *simp-all*

lemma *interval-suffix-suffix* [*simp*]:
 $\text{suffix } i \text{ } (\text{suffix } j \text{ } xs) = \text{suffix } (i+j) \text{ } xs$
by (*induct xs arbitrary: i j, simp, (case-tac i, auto, (simp add: Nitpick.case-nat-unfold))*)

lemma *interval-prefix-suffix-intlen* [*code*]:
 $\text{intlen } (\text{prefix } ia \text{ } (\text{suffix } i \text{ } xs)) =$
 $(\text{if } i \leq \text{intlen } xs \text{ then}$
 $(\text{if } ia \leq \text{intlen } xs - i \text{ then } ia \text{ else } (\text{intlen } xs) - i)$
 $\text{else } 0)$
by (*metis interval-prefix-length interval-suffix-length le-zero-eq*)

lemma *interval-prefix-suffix-intlen-good* [*simp*]:
assumes $ia \leq \text{intlen } xs - i \wedge i \leq \text{intlen } xs$
shows $\text{intlen } (\text{prefix } ia \text{ } (\text{suffix } i \text{ } xs)) = ia$
using *assms by (simp add: interval-prefix-suffix-intlen)*

lemma *interval-prefix-suffix-intlen-bad-0* [*simp*]:
assumes $i > \text{intlen } xs$
shows $\text{intlen } (\text{prefix } ia \text{ } (\text{suffix } i \text{ } xs)) = 0$
using *assms by (simp add: interval-prefix-suffix-intlen)*

lemma *interval-prefix-suffix-intlen-bad-1* [*simp*] :
assumes $i \leq \text{intlen } xs \wedge ia > \text{intlen } xs - i$
shows $\text{intlen } (\text{prefix } ia \text{ } (\text{suffix } i \text{ } xs)) = (\text{intlen } xs) - i$
using *assms by (simp add: interval-prefix-suffix-intlen)*

lemma *interval-suffix-suffix-3*:
assumes $i > 0 \wedge ia < i \wedge i \leq \text{intlen } xs$
shows $(\text{suffix } (i-ia) \text{ } (\text{suffix } ((\text{intlen } xs) - i) \text{ } xs)) = (\text{suffix } (((\text{intlen } xs) - ia)) \text{ } xs)$
using *assms by simp*

lemma *interval-sub-zero-prefix* :
 $\text{sub } 0 \text{ } k \text{ } xs = \text{prefix } k \text{ } xs$
by (*simp add: Interval.sub-def*)

lemma *interval-sub-suffix* :
assumes $(i < j \wedge j \leq (\text{intlen } xs) - k)$
shows $(\text{sub } (i+k) \text{ } (j+k) \text{ } xs) = (\text{sub } i \text{ } j \text{ } (\text{suffix } k \text{ } xs))$
using *assms by (simp add: Interval.sub-def)*

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

lemma *interval-sub-prefix-suffix*:
assumes $0 \leq i \wedge i \leq j \wedge j \leq \text{intlen } xs$

shows $(\text{sub } i \ j \ xs) = (\text{prefix } (j-i) \ (\text{suffix } i \ xs))$
using *assms* **by** (*simp add: Interval.sub-def*)

1.2.6 intapp

lemma *interval-prefix-intapp* [*simp*]:
 $\text{prefix } (\text{intlen } xs - k) \ (xs \ominus ys) = \text{prefix } (\text{intlen } xs - k) \ xs$
by (*induct xs arbitrary: k, simp, (case-tac k, simp,*
metis comm-monoid-add-class.add-0 interval-intlen-gr-zero interval-prefix-intlen
le-add-diff-inverse),
smt Suc-diff-Suc diff-Suc-Suc diff-is-0-eq' intapp.simps(2) interval-prefix-suc
interval-prefix-zero intlen.simps(2) not-le plus-1-eq-Suc)

lemma *interval-prefix-intapp2* [*simp*]:
 $\text{prefix } (\text{intlen } xs + k + 1) \ (xs \ominus ys) = xs \ominus \text{prefix } k \ ys$
by (*induct xs arbitrary: k, simp,*
(case-tac k, simp, metis add.right-neutral interval-prefix-zero-intfirst),
simp add: add-eq-if)

lemma *interval-suffix-intapp* [*simp*]:
 $\text{suffix } (\text{intlen } xs + m + 1) \ (xs \ominus ys) = \text{suffix } (m) \ ys$
by (*induct xs arbitrary: m, simp, (case-tac m, simp, metis add.right-neutral interval-suffix-zero),*
simp add: add-eq-if)

lemma *interval-suffix-intapp2* [*simp*]:
 $(\text{suffix } (\text{intlen } xs - k) \ xs) \ominus ys = \text{suffix } (\text{intlen } xs - k) \ (xs \ominus ys)$
by (*induct xs, simp*)
(metis Suc-diff-le diff-is-0-eq' intapp-Cons interval-suffix-suc interval-suffix-zero
intlen.simps(2) not-less-eq-eq plus-1-eq-Suc)

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

lemma *interval-intapp-nth*:
 $\text{nth } (xs \ominus ys) \ k = (\text{if } k \leq \text{intlen } xs$
 $\quad \text{then } (\text{nth } xs \ k)$
 $\quad \text{else } (\text{nth } ys \ (k - (\text{intlen } xs) - 1)))$
by (*induct xs arbitrary: k, (case-tac k, simp, simp),*
(case-tac k, simp, simp))

lemma *interval-rev-intapp* [*simp*]:
 $\text{intrev } (xs \ominus ys) = (\text{intrev } ys) \ominus (\text{intrev } xs)$
by (*induct xs*) *simp-all*

lemma *interval-intlast-intapp* [*simp*]:
 $\text{intlast}(xs \ominus \langle x \rangle) = x$
by (*induct xs, simp, simp*)

lemma *interval-intlast-intapp2* [simp]:
 $\text{intlast } (xs \ominus ys) = \text{intlast } ys$
by (induct xs arbitrary: ys, simp, simp)

lemma *interval-intfirst-intapp* [simp]:
 $\text{intfirst } (\langle x \rangle \ominus xs) = x$
by (induct xs, simp, simp)

lemma *interval-intfirst-intapp2* [simp]:
 $\text{intfirst}(xs \ominus ys) = \text{intfirst } xs$
by (induct xs arbitrary: ys, simp, simp)

1.2.7 Reverse

lemma *interval-rev-rev-ident* [simp]:
 $\text{intrev } (\text{intrev } xs) = xs$
by (induct xs) auto

lemma *interval-rev-swap* :
 $((\text{intrev } xs) = ys) = (xs = \text{intrev } ys)$
by auto

lemma *interval-rev-singleton-conv* [simp]:
 $(\text{intrev } xs = \langle x \rangle) = (xs = \langle x \rangle)$
by (metis interval-rev-rev-ident intrev.simps(1))

lemma *interval-single-rev-conv* [simp]:
 $(\langle x \rangle = \text{intrev } xs) = (\langle x \rangle = xs)$
by (metis interval-rev-rev-ident intrev.simps(1))

lemma *interval-rev-is-rev-conv* [iff]:
 $(\text{intrev } xs = \text{intrev } ys) = (xs = ys)$

proof
(induct xs arbitrary: ys)
case (St x)
then show ?case **by** simp
next
case (Cons x1a xs)
then show ?case
using interval-rev-swap **by** force
qed

lemma *interval-rev-induct* [case-names St snoc]:
 $\llbracket \bigwedge y. P \langle y \rangle ; \bigwedge x xs. P xs \implies P(xs \ominus \langle x \rangle) \rrbracket \implies P xs$
by (simplesubst interval-rev-rev-ident[symmetric],
rule-tac interval = intrev xs **in** interval.induct, simp-all)

lemma *interval-rev-exhaust* [case-names St snoc]:
 $(\bigwedge x. xs = \langle x \rangle \implies P) \implies (\bigwedge ys y. xs = ys \ominus \langle y \rangle \implies P) \implies P$
by (induct xs rule: interval-rev-induct) auto

lemmas *interval-rev-cases* = *interval-rev-exhaust*

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

lemma *interval-same-intapp-eq*[iff]:
 $(xs \ominus ys = xs \ominus zs) = (ys = zs)$
using *interval-suffix-intapp* **by** (*metis interval-suffix-zero*)

lemma *interval-intapp-eq-conv*[iff]:
 $(xs \ominus \langle x \rangle = ys \ominus \langle y \rangle) = (xs = ys \wedge x = y)$
by (*metis interval-intlast-intapp interval-rev-intapp interval-rev-is-rev-conv interval-same-intapp-eq*)

lemma *interval-intapp-same-eq*[iff]:
 $(ys \ominus xs = zs \ominus xs) = (ys = zs)$
by (*metis interval-rev-intapp interval-rev-swap interval-same-intapp-eq*)

lemma *interval-intrev-intapp-cons*:
 $\text{intrev } (xs \ominus \langle x \rangle) = x \odot \text{intrev } xs$
by (*case-tac xs,simp,simp*)

lemma *interval-intlast-intrev*:
 $\text{intlast } (\text{intrev } xs) = \text{intfirst } xs$
by (*case-tac xs,simp,simp*)

lemma *interval-intfirst-intrev*:
 $\text{intfirst } (\text{intrev } xs) = \text{intlast } xs$
by (*induct xs,simp,simp*)

lemma *interval-intrev-nth*:
assumes $k \leq \text{intlen } (\text{intrev } xs)$
shows $(\text{nth } (\text{intrev } xs) k) = (\text{nth } xs ((\text{intlen } xs) - k))$
using *assms*
by (*induct xs, simp,simp,*
(case-tac k,simp add:interval-intapp-nth,
smt Interval.nth.simps(1) Suc-diff-Suc diff-Suc-Suc diff-is-0-eq' interval-intapp-nth
interval-intrev-intlen le-SucE less-Suc-eq-le old.nat.simps(4) old.nat.simps(5))))

lemma *interval-intrev-prefix*:
assumes $k \leq \text{intlen } xs$
shows $\text{intrev } (\text{prefix } k xs) = \text{suffix } ((\text{intlen } xs) - k) (\text{intrev } xs)$
by (*induct xs arbitrary: k, simp,simp,*
(case-tac k, simp,
metis Suc-eq-plus1 add.right-neutral interval-intlast-intapp interval-intlen-intapp
interval-intrev-intlen interval-suffix-intlast intlen.simps(1)),
metis diff-Suc-Suc interval-intrev-intlen interval-suffix-intapp2)

intrev.simps(2) old.nat.simps(5)

lemma *interval-intrev-suffix*:

assumes $k \leq \text{intlen } xs$

shows $\text{intrev}(\text{suffix } k \text{ } xs) = \text{prefix } ((\text{intlen } xs) - k) (\text{intrev } xs)$

using *assms*

by (*induct xs arbitrary: k, simp, simp add: interval-intrev-prefix interval-rev-swap*)

lemma *interval-intrev-sub*:

assumes $0 \leq i \wedge i \leq j \wedge j \leq \text{intlen } xs$

shows $\text{intrev}(\text{sub } i \text{ } j \text{ } xs) = \text{sub } ((\text{intlen } xs) - j) ((\text{intlen } xs) - i) (\text{intrev } xs)$

using *assms*

proof –

have 1: $\text{intrev}(\text{sub } i \text{ } j \text{ } xs) = \text{intrev}(\text{prefix } (j-i) (\text{suffix } i \text{ } xs))$

using *assms interval-sub-prefix-suffix* **by** (*simp add: interval-sub-prefix-suffix*)

have 2: $\text{intrev}(\text{prefix } (j-i) (\text{suffix } i \text{ } xs)) = \text{suffix } ((\text{intlen } xs) - j) (\text{intrev}(\text{suffix } i \text{ } xs))$

using *assms interval-intrev-prefix*[*of j-i suffix i xs*] **by** *auto*

have 3: $\text{suffix } ((\text{intlen } xs) - j) (\text{intrev}(\text{suffix } i \text{ } xs)) =$

$\text{suffix } ((\text{intlen } xs) - j) (\text{prefix } ((\text{intlen } xs) - i) (\text{intrev } xs))$

using *assms interval-intrev-suffix*[*of i xs*] **by** *auto*

have 4: $\text{suffix } ((\text{intlen } xs) - j) (\text{prefix } ((\text{intlen } xs) - i) (\text{intrev } xs)) =$

$\text{sub } ((\text{intlen } xs) - j) ((\text{intlen } xs) - i) (\text{intrev } xs)$

using *assms* **by** (*simp add: diff-le-mono2 interval-sub-prefix-suffix interval-suffix-prefix-swap*)

from 1 2 3 4 **show** *?thesis* **by** *auto*

qed

end

2 Infinite Intervals

theory *InfiniteInterval*

imports

Interval

begin

An infinite interval is a mapping from the natural numbers to a particular type. This is similar as the theory *Omega-Words-Fun* of the Isabelle/HOL distribution. The difference is that our version has no empty (no symbols) word. This is needed as an finite interval has at least one state. So we have to adapt the definition of *conc* and *upt*. We also define the usual *isuffix*, *iprefix* and *subinterval* on infinite intervals.

2.1 Definitions

type-synonym

$'a \text{ infinterval} = \text{nat} \Rightarrow 'a$

type-synonym

infiniteindex = *nat infinterval*

definition

conc :: [*a interval*, *a infinterval*] \Rightarrow *a infinterval*

where *conc* *w* *x* = (λn . if $n \leq \text{intlen } w$ then *nth* *w* *n* else *x* ($n - \text{intlen } w - 1$))

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

where

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

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

definition

isuffix :: [*nat*, *a infinterval*] \Rightarrow *a infinterval*

where *isuffix* *k* *x* = (λn . *x* ($k+n$))

definition

subinterval :: *a infinterval* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *a interval*

where

subinterval *w* *i* *j* = *map* *w* [*i*..*j*]

definition

iprefix :: *nat* \Rightarrow *a infinterval* \Rightarrow *a interval*

where

iprefix *n* *w* \equiv *subinterval* *w* 0 *n*

definition *infinite-index-sequence* :: *nat* \Rightarrow *infiniteindex* \Rightarrow *bool* **where**

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

2.2 Lemmas

2.2.1 upt

lemma *upt-rec*[code]:

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

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

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

assumes $j < i$

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

using *assms*

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

lemma *upt-same*:

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

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

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

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

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

lemma *upt-eq-cons-conv*:

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

using *InfiniteInterval.upt-rec* **by** (*induct j arbitrary: x xs, simp, auto*)

lemma *upt-suc-append*:

assumes $i \leq j$

shows $[i..(Suc\ j)] = [i..j] \oplus (Suc\ j)$

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

lemma *upt-conv-cons*:

assumes $i < j$

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

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

lemma *upt-conv-cons-cons*:

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

proof (*cases m ≤ q*)

case *True*

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

next

case *False*

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

qed

lemma *upt-add-eq-append*:

assumes $i \leq j \wedge k > 0$

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

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-length*:

intlen $[i..j] = j - i$

by (*induct j*) (*auto simp add: Suc-diff-le*)

lemma *test*:

$k \leq \text{intlen } [i..i + k]$

by (*simp add: upt-length*)

lemma *upt-nth-help*:

Interval.nth $[i..i + k] \ k = i + k$

by (*induct k arbitrary: i, simp add: upt-same,*

*metis InfinitelInterval.upt-rec add-Suc-shift interval-nth-Suc less-add-same-cancel1
zero-less-Suc)*

lemma *upt-nth*:
assumes $i + k \leq j$
shows $(nth [i.. \leq j] k) = i + k$
using *assms*
by (*induct j arbitrary: k i, simp,*
metis InfinitelInterval.upt.upt-Suc Nat.le-diff-conv2 add.commute add-leD1
interval-intapp-nth le-SucE upt-length upt-nth-help)

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

lemma *upt-intlast*:
 $intlast [i.. \leq j] = j$
by (*metis InfinitelInterval.upt-rec interval-nth-intlen-intlast intlast.simps(1) less-or-eq-imp-le*
ordered-cancel-comm-monoid-diff-class.add-diff-inverse upt-length upt-nth)

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-intfirst upt-same*)
next
case (*Suc m*)
then show ?case **using** *InfinitelInterval.upt-rec* **by** *auto*
qed

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

lemma *map-suc-upt*:
 $map\ Suc\ [m.. \leq n] = [Suc\ m.. \leq Suc\ n]$
proof

```

(induct n arbitrary: m)
case 0
then show ?case by simp
next
case (Suc n)
then show ?case by simp
qed

```

```

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

```

2.2.2 isuffix

```

lemma isuffix-nth:
  (isuffix k x) n = x (k+n)
by (simp add: isuffix-def)

```

```

lemma isuffix-0:
  isuffix 0 x = x
by (simp add: isuffix-def)

```

```

lemma isuffix-isuffix:
  (isuffix m (isuffix n x)) = isuffix (n+m) x
by (rule ext) (simp add: isuffix-def add.assoc)

```

2.2.3 iprefix

```

lemma iprefix-0:
  (iprefix 0 x) =  $\langle (x\ 0) \rangle$ 
by (simp add: iprefix-def subinterval-def)

```

```

lemma iprefix-nth:
  assumes  $k \leq m$ 
  shows (nth (iprefix m x) k) = (x k)
using assms
by (simp add: interval-nth-map iprefix-def subinterval-def upt-nth)

```

```

lemma iprefix-length:
  intlen (iprefix n x) = n
by (simp add: iprefix-def subinterval-def upt-length)

```

2.2.4 subinterval

lemma *subinterval-length*:

intlen (subinterval x i j) = j - i

by (*simp add: subinterval-def upt-length*)

lemma *subinterval-nth*:

assumes $i+k \leq j$

shows $\text{nth} (\text{subinterval } x \ i \ j) \ k = x \ (i+k)$

unfolding *subinterval-def*

using *assms* **by** (*simp add: interval-nth-map upt-nth*)

lemma *iprefix-isuffix*:

iprefix n (isuffix k x) = subinterval x k (n+k)

proof –

have 0: $\text{iprefix } n \ (\text{isuffix } k \ x) = \text{map } (\lambda n. x \ (k + n)) \ [0..\leq n]$

by (*simp add: iprefix-def isuffix-def subinterval-def*)

have 1: $[k..\leq n+k] = (\text{map } (\lambda i. i+k) \ [0..\leq n])$

using *map-add-upt* **by** *simp*

hence 2: $\text{map } x \ [k..\leq n+k] = \text{map } x \ (\text{map } (\lambda i. i+k) \ [0..\leq n])$

by *simp*

have 3: $\text{map } x \ (\text{map } (\lambda i. i+k) \ [0..\leq n]) = \text{map } (x \circ (\lambda i. i+k)) \ [0..\leq n]$

by *simp*

have 4: $(x \circ (\lambda i. i+k)) = (\lambda n. x \ (k + n))$ **by** (*metis add.commute comp-apply*)

hence 5: $\text{map } (x \circ (\lambda i. i+k)) \ [0..\leq n] = \text{map } (\lambda n. x \ (k + n)) \ [0..\leq n]$

by *simp*

have 6: $\text{subinterval } x \ k \ (n+k) = \text{map } x \ [k..\leq n+k]$

by (*simp add: subinterval-def*)

from 0 2 3 5 6 **show** *?thesis* **by** *auto*

qed

lemma *subinterval-sub-isuffix*:

assumes $i < j$

shows $(\text{subinterval } xs \ (i+k) \ (j+k)) = (\text{subinterval } (\text{isuffix } k \ xs) \ i \ j)$

proof –

have 1: $(\text{subinterval } xs \ (i+k) \ (j+k)) =$

$\text{iprefix } (j-i) \ (\text{isuffix } (i+k) \ xs)$

by (*simp add: iprefix-isuffix assms less-imp-le-nat*)

have 2: $\text{iprefix } (j-i) \ (\text{isuffix } (i+k) \ xs) =$

$\text{iprefix } (j-i) \ (\text{isuffix } (i) \ (\text{isuffix } k \ xs))$

by (*simp add: isuffix-isuffix add.commute*)

have 3: $\text{iprefix } (j-i) \ (\text{isuffix } (i) \ (\text{isuffix } k \ xs)) =$

$(\text{subinterval } (\text{isuffix } k \ xs) \ i \ j)$

by (*simp add: iprefix-isuffix assms less-imp-le-nat*)

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

qed

lemma *subinterval-sub-isuffix-iidx*:

assumes *infinite-index-sequence* $0 \ \text{Isk} \wedge n > 0$

shows $(\text{subinterval } \sigma \ ((\text{Isk } i) + n) \ ((\text{Isk } (\text{Suc } i)) + n)) =$

$(\text{subinterval } (\text{isuffix } n \ \sigma) \ (\text{Isk } i) \ (\text{Isk } (\text{Suc } i)))$

using *assms* **by** (*simp add: infinite-index-sequence-def subinterval-sub-isuffix*)

lemma *interval-pref-ipref-3-intlen*:

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

by (*simp add: iprefix-length*)

lemma *interval-pref-ipref-3-nth*:

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

by (*smt interval-eq-nth-eq interval-nth-prefix interval-pref-ipref-3-intlen iprefix-length iprefix-nth le-add1 le-trans*)

lemma *interval-pref-ipref-3 [simp]*:

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

by (*simp add: interval-eq-nth-eq interval-pref-ipref-3-intlen interval-pref-ipref-3-nth*)

lemma *interval-iprefix-isuffix-swap-intlen*:

$\text{intlen } (\text{iprefix } ia \text{ (isuffix } i \text{ xs})) = \text{intlen } (\text{suffix } i \text{ (iprefix } (ia+i) \text{ xs}))$

by (*simp add: iprefix-length*)

lemma *interval-iprefix-isuffix-swap-nth*:

assumes $m \leq ia$

shows $(\text{nth } (\text{iprefix } ia \text{ (isuffix } i \text{ xs})) \text{ m}) = (\text{nth } (\text{suffix } i \text{ (iprefix } (ia+i) \text{ xs})) \text{ m})$

using *assms* **by** (*simp add: iprefix-length iprefix-nth isuffix-def*)

lemma *interval-iprefix-isuffix-swap*:

$\text{iprefix } ia \text{ (isuffix } i \text{ xs}) = \text{suffix } i \text{ (iprefix } (ia+i) \text{ xs})$

by (*simp add: interval-eq-nth-eq interval-iprefix-isuffix-swap-nth iprefix-length*)

2.2.5 Conc

lemma *conc-empty-zero*:

$(\text{conc } \langle s \rangle \text{ x}) \text{ 0} = s$

unfolding *conc-def* **by** *auto*

lemma *conc-empty-suc*:

$(\text{conc } \langle s \rangle \text{ x}) (\text{Suc } i) = x \text{ i}$

unfolding *conc-def* **by** *auto*

lemma *conc-conc*:

$\text{conc } x \text{ (conc } y \text{ w}) = \text{conc } (x \oplus y) \text{ w}$ (**is** $?lhs = ?rhs$)

proof

fix n

have $x: n \leq \text{intlen } x \implies ?lhs \text{ n} = ?rhs \text{ n}$

by (*simp add: conc-def interval-intapp-nth*)

have $y: n > \text{intlen } x \wedge n \leq (\text{intlen } x) + (\text{intlen } y) \implies ?lhs \text{ n} = ?rhs \text{ n}$

by (*simp add: conc-def interval-intapp-nth, arith*)

have $w: n > (\text{intlen } x) + (\text{intlen } y) \implies ?lhs \text{ n} = ?rhs \text{ n}$

by (*simp add: conc-def interval-intapp-nth, arith*)

from $x \text{ y w}$ **show** $?lhs \text{ n} = ?rhs \text{ n}$ **using** *not-less* **by** *blast*

qed

lemma *conc-iprefix-isuffix*:

$x = \text{conc } (\text{iprefix } n \ x) \ (\text{isuffix } (\text{Suc } n) \ x)$
by (*rule ext*)
 (*induct n, simp add: conc-def iprefix-0 isuffix-def ,*
 simp add: conc-def iprefix-length iprefix-nth ,
 metis isuffix-def nat-le-linear not-less-eq-eq
 ordered-cancel-comm-monoid-diff-class.add-diff-inverse)

2.2.6 Infinite index sequence

lemma *iidx-1*:

$I = (\text{conc } \langle (I \ 0) \rangle (\lambda x. (I \ (x+1))))$
(*is ?lhs = ?rhs*)

proof

fix *n*
have $x: n \leq \text{intlen } \langle (I \ 0) \rangle \implies ?lhs \ n = ?rhs \ n$
by (*simp add: conc-def interval-intapp-nth*)
have $ls: n > \text{intlen } \langle (I \ 0) \rangle \implies ?lhs \ n = ?rhs \ n$
by (*simp add: conc-def interval-intapp-nth*)
from $x \ ls$ **show** $?lhs \ n = ?rhs \ n$ **by** *auto*
qed

lemma *iidx-2*:

infinite-index-sequence $0 \ (\text{conc } \langle (I \ 0) \rangle (\lambda x. (I \ (x+1)))) =$
 $((I \ 0) = 0 \wedge (I \ 0) < (I \ 1) \wedge$
 $\text{infinite-index-sequence } (I \ 1) (\lambda x. (I \ (x+1))))$
by (*simp add: infinite-index-sequence-def conc-def , auto*)
(*metis Suc-diff-Suc add-diff-cancel-left' gr0l plus-1-eq-Suc zero-less-diff*)

lemma *iidx-less-plus*:

assumes *infinite-index-sequence n ls*
shows $(ls \ i) < (ls \ (\text{Suc } (i+k)))$
using *assms*
by (*simp add: infinite-index-sequence-def lift-Suc-mono-less*)

lemma *iidx-greater*:

assumes *infinite-index-sequence n ls*
shows $i > 0 \longrightarrow n < ls \ i$
using *assms*
by (*induct i, simp, metis infinite-index-sequence-def less-imp-Suc-add iidx-less-plus*)

lemma *iidx-3*:

assumes *infinite-index-sequence n ls*
shows *infinite-index-sequence* $0 \ ((\text{shiftn } n) \circ ls)$
using *assms*
by (*simp add: infinite-index-sequence-def shiftn-def*)
(*metis diff-less-mono less-le lift-Suc-mono-less-iff not-less zero-less-Suc*)

lemma *iidx-4*:

$(\text{infinite-index-sequence } (x+k) ((\text{shift } k) \circ ls)) =$
 $\text{infinite-index-sequence } x \text{ } ls$
by (*simp add: shift-def infinite-index-sequence-def*)

lemma *iidx-5*:

assumes $(\text{infinite-index-sequence } k \text{ } (lsk) \wedge ls = (\text{shiftn } k) \circ lsk)$
shows $\text{infinite-index-sequence } 0 \text{ } ls$
using *assms*
by (*simp add: infinite-index-sequence-def shiftn-def*)
 $(\text{metis add-less-same-cancel1 diff-less-mono lift-Suc-mono-less-iff not-add-less1 not-le-imp-less})$

lemma *iidx-ext*:

$((xs :: \text{infiniteindex}) = ys) = (\forall i. xs \ i = ys \ i)$
by *auto*

lemma *iidx-6*:

$(\text{infinite-index-sequence } k \text{ } lsk \wedge lsk = (\text{shift } k) \circ ls \wedge \text{infinite-index-sequence } 0 \text{ } ls) =$
 $(\text{infinite-index-sequence } k \text{ } lsk \wedge ls = (\text{shiftn } k) \circ lsk \wedge \text{infinite-index-sequence } 0 \text{ } ls)$
by (*simp add: iidx-ext infinite-index-sequence-def shift-def shiftn-def, rule, auto,*
metis add commute add-diff-inverse-nat add-less-same-cancel1 lift-Suc-mono-less-iff not-add-less1)

lemma *iidx-7*:

$(\text{infinite-index-sequence } k \text{ } lsk \wedge ls = (\text{shiftn } k) \circ lsk) =$
 $(\text{infinite-index-sequence } k \text{ } lsk \wedge ls = (\text{shiftn } k) \circ lsk \wedge$
 $\text{infinite-index-sequence } 0 \text{ } ls)$
using *iidx-5 by blast*

lemma *iidx-8*:

$(lsk = (\text{shift } k) \circ ls \wedge \text{infinite-index-sequence } 0 \text{ } ls) =$
 $(lsk = (\text{shift } k) \circ ls \wedge \text{infinite-index-sequence } k \text{ } lsk \wedge$
 $\text{infinite-index-sequence } 0 \text{ } ls)$
by (*metis Interval.shift-def add-left-imp-eq diff-is-0-eq' interval-idx-shift-mono*
le-add-diff-inverse2 mono-def iidx-4 rel-simps(46))

lemma *iidx-0-a*:

$\text{infinite-index-sequence } 0 \text{ } l \implies$
 $(l \ 0) = 0 \wedge (l \ 0) < (l \ 1) \wedge \text{infinite-index-sequence } (l \ 1) \ (\lambda x. l(x+1))$
by (*simp add: infinite-index-sequence-def*)
metis

lemma *iidx-0-b*:

$x = 0 \wedge x < (l \ 0) \wedge \text{infinite-index-sequence } (l \ 0) \text{ } l \implies$
 $\text{infinite-index-sequence } 0 \text{ } (\text{conc } \langle x \rangle \ l)$
using *diff-Suc-less infinite-index-sequence-def*
by (*simp add: conc-def lift-Suc-mono-less*)

```

lemma iidx-0:
  ( $\exists l. \text{infinite-index-sequence } 0\ l$ ) =
  ( $\exists ls\ x. x = 0 \wedge x < (ls\ 0) \wedge \text{infinite-index-sequence } (ls\ 0)\ ls$ )
using infinite-index-sequence-def lessl iidx-0-b by metis

```

end

3 Finite ITL Semantics

```

theory Semantics
imports Interval HOL-TLA.Intensional
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., [8] 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.

3.1 Types of Formulas

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

```

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

```

```

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

```

```

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

```

```

instance
  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.

3.2 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 $skip-d \equiv \lambda s. intlen\ s = 1$

definition $chop-d :: ('a :: world) formula \Rightarrow ('a :: world) formula \Rightarrow ('a :: world) formula$
where $chop-d\ F1\ F2 \equiv \lambda s. \exists n. 0 \leq n \wedge n \leq intlen\ s \wedge ((prefix\ n\ s) \models F1) \wedge ((suffix\ n\ s) \models F2)$

definition $current-val-d :: ('a :: world, 'b) stfun \Rightarrow ('a, 'b) formfun$
where $current-val-d\ f \equiv \lambda s. (nth\ s\ 0) \models f$

definition $next-val-d :: ('a :: world, 'b) stfun \Rightarrow ('a, 'b) formfun$
where $next-val-d\ f \equiv \lambda s. \text{if } intlen\ s > 0 \text{ then } ((nth\ s\ 1) \models f) \text{ else } (\epsilon\ (x :: 'b). x = x)$

definition $fin-val-d :: ('a :: world, 'b) stfun \Rightarrow ('a, 'b) formfun$
where $fin-val-d\ f \equiv \lambda s. (nth\ s\ (intlen\ s)) \models f$

definition $penult-val-d :: ('a :: world, 'b) stfun \Rightarrow ('a, 'b) formfun$
where $penult-val-d\ f \equiv \lambda s. \text{if } intlen\ s > 0 \text{ then } (nth\ s\ ((intlen\ s) - 1)) \models f \text{ else } (\epsilon\ (x :: 'b). x = x)$

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

syntax

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

syntax (ASCII)

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

translations

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

3.3 Abbreviations

Some standard temporal abbreviations, with their concrete syntax.

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

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

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

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

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

syntax

-*sometimes-d* :: lift \Rightarrow lift ((\Diamond -) [88] 87)
 -*di-d* :: lift \Rightarrow lift ((*di* -) [88] 87)
 -*da-d* :: lift \Rightarrow lift ((*da* -) [88] 87)
 -*next-d* :: lift \Rightarrow lift ((\circ -) [88] 87)
 -*prev-d* :: lift \Rightarrow lift ((*prev* -) [88] 87)

syntax (ASCII)

-*sometimes-d* :: lift \Rightarrow lift (($\langle \rangle$ -) [88] 87)
 -*di-d* :: lift \Rightarrow lift ((*di* -) [88] 87)
 -*da-d* :: lift \Rightarrow lift ((*da* -) [88] 87)
 -*next-d* :: lift \Rightarrow lift ((*next* -) [88] 87)
 -*prev-d* :: lift \Rightarrow lift ((*prev* -) [88] 87)

translations

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

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

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

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

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

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

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

syntax

-always-d :: lift \Rightarrow lift ((\square -) [88] 87)
 -bi-d :: lift \Rightarrow lift ((bi-) [88] 87)
 -ba-d :: lift \Rightarrow lift ((ba-) [88] 87)
 -wnext-d :: lift \Rightarrow lift ((wnext-) [88] 87)
 -wprev-d :: lift \Rightarrow lift ((wprev-) [88] 87)
 -more-d :: lift ((more))

syntax (ASCII)

-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 $\Rightarrow \text{CONST always-d}$
 -bi-d $\Rightarrow \text{CONST bi-d}$
 -ba-d $\Rightarrow \text{CONST ba-d}$
 -wnext-d $\Rightarrow \text{CONST wnext-d}$
 -wprev-d $\Rightarrow \text{CONST wprev-d}$
 -more-d $\Rightarrow \text{CONST more-d}$

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

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

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)
-yields-d :: [lift, lift] \Rightarrow lift ((- yields -) [88,88] 87)
-ifthenelse-d :: [lift, lift, lift] \Rightarrow lift ((if_i - then - else -) [88,88,88] 87)
-power-d :: [lift, nat] \Rightarrow lift ((power - -) [88,88] 87)

syntax (ASCII)

-bm-d :: lift \Rightarrow lift ((bm -) [88] 87)
-init-d :: lift \Rightarrow lift ((init -) [88] 87)
-fin-d :: lift \Rightarrow lift ((fin -) [88] 87)

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

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

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

syntax (ASCII)

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

translations

$\text{-chopstar-d} \quad \Rightarrow \text{CONST chopstar-d}$

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

where *ifthen-d* $F\ G \equiv \text{LIFT}(if_i\ F\ \text{then}\ G\ \text{else}\ \#True)$

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

where *while-d* $F\ G \equiv \text{LIFT}((F \wedge G)^* \wedge (fin\ ((\neg F))))$

syntax

-*ifthen-d* $:: [lift, lift] \Rightarrow lift\ ((if_i - then -)\ [88,88]\ 87)$

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

syntax (ASCII)

-*ifthen-d* $:: [lift, lift] \Rightarrow lift\ ((if_i - then -)\ [88,88]\ 87)$

-*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* $F\ G \equiv \text{LIFT}(F;while\ (\neg G)\ do\ F)$

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* $v\ e \equiv \text{LIFT}(v\$ = e)$

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

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

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* $v\ e \equiv \lambda s. s \models keep(\ temporal-assign-d\ v\ e)$

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

where *stable-d* $v \equiv \lambda s. s \models gets-d\ v\ (current-val-d\ v)$

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

where *padded-d* $v \equiv \lambda s. s \models (stable-d\ v);skip \vee empty$

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

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 ((- <~ -) [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

3.4 Properties of Operators

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

lemma *skip-defs* :

(w \models skip) = (intlen w = 1)

by (simp add: skip-d-def)

lemma *chop-defs* :

(w \models F1 ; F2) = ($\exists n. 0 \leq n \wedge n \leq \text{intlen } w \wedge ((\text{prefix } n \text{ } w) \models F1) \wedge ((\text{suffix } n \text{ } w) \models F2)$)

by (simp add: chop-d-def)

lemma *sometimes-defs* :

(w $\models \Diamond F$) = ($\exists n. 0 \leq n \wedge n \leq \text{intlen } w \wedge ((\text{suffix } n \text{ } w) \models F)$)

by (simp add: Semantics.sometimes-d-def chop-defs)

lemma *always-defs* :

$(w \models \Box F) = (\forall n. 0 \leq n \wedge n \leq \text{intlen } w \longrightarrow ((\text{suffix } n \ w) \models F))$

by (*simp add: always-d-def sometimes-defs*)

lemma *di-defs* :

$(w \models \text{di } F) = (\exists n. 0 \leq n \wedge n \leq \text{intlen } w \wedge ((\text{prefix } n \ w) \models F))$

by (*simp add: Semantics.di-d-def chop-defs*)

lemma *bi-defs* :

$(w \models \text{bi } F) = (\forall n. 0 \leq n \wedge n \leq \text{intlen } w \longrightarrow ((\text{prefix } n \ w) \models F))$

by (*simp add: Semantics.bi-d-def di-defs*)

lemma *da-defs* :

$(w \models \text{da } F) = (\exists n \ na. 0 \leq n \wedge na+n \leq \text{intlen } w \wedge ((\text{sub } n \ (na+n) \ w) \models F))$

using *interval-prefix-length-good interval-suffix-length-good*

by (*simp add: Semantics.da-d-def chop-defs,*

*smt add.commute add-diff-cancel-left' add-leD2 interval-sub-prefix-suffix-0 le-iff-add
nat-add-left-cancel-le zero-le*)

lemma *ba-defs* :

$(w \models \text{ba } F) = (\forall n \ na. 0 \leq n \wedge na+n \leq \text{intlen } w \longrightarrow ((\text{sub } n \ (na+n) \ w) \models F))$

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

lemma *next-defs* :

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

using *Suc-le-eq* **by** (*simp add: next-d-def chop-defs skip-defs, force*)

lemma *wnext-defs* :

$(w \models \text{wnext } F) = (\text{intlen } w = 0 \vee ((\text{suffix } 1 \ w) \models F))$

by (*simp add: wnext-d-def next-defs*)

lemma *prev-defs* :

$(w \models \text{prev } F) = (\text{intlen } w > 0 \wedge ((\text{prefix } ((\text{intlen } w) - 1) \ w) \models F))$

by (*simp add: prev-d-def chop-defs skip-defs*)

*(metis One-nat-def Suc-le1 diff-diff-cancel diff-is-0-eq' diff-le-self
interval-suffix-length-good neq0-conv zero-neq-one)*

lemma *wprev-defs* :

$(w \models \text{wprev } F) = (\text{intlen } w = 0 \vee ((\text{prefix } ((\text{intlen } w) - 1) \ w) \models F))$

by (*metis (mono-tags, lifting) less-le prev-defs unl-lift wprev-d-def zero-le*)

lemma *more-defs* :

$(w \models \text{more}) = (\text{intlen } w > 0)$

by (*simp add: more-d-def next-defs*)

lemma *empty-defs* :

$(w \models \text{empty}) = (\text{intlen } w = 0)$

by (*simp add: empty-d-def more-defs*)

lemma *init-defs* :

$(w \models \text{init } F) = ((\text{Interval.prefix } 0 \ w) \models F)$
by (simp add: init-d-def empty-defs chop-defs) auto

lemma *initalt-defs* :
 $(w \models \text{bi}(\text{empty} \longrightarrow F)) = ((\text{Interval.prefix } 0 \ w) \models F)$
by (simp add: bi-defs empty-defs)

lemma *fin-defs* :
 $(w \models \text{fin } F) = ((\text{Interval.suffix } (\text{intlen } w) \ w) \models F)$
by (simp add: fin-d-def empty-defs always-defs)

lemma *finalt-defs* :
 $(w \models \# \text{True}; (F \wedge \text{empty})) = ((\text{Interval.suffix } (\text{intlen } w) \ w) \models F)$
by (simp add: chop-defs empty-defs) fastforce

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

lemma *initonly-defs* :
 $(w \models \text{initonly}(F)) = (\forall n \leq \text{intlen } w. (n = 0) = F (\text{prefix } n \ w))$
by (simp add: initonly-d-def bi-defs empty-defs)

lemma *ifthenelse-defs*:
 $(w \models \text{if}_i \ F \ \text{then } G \ \text{else } H) =$
 $(((w \models F) \wedge (w \models G)) \vee ((\neg(w \models F) \wedge (w \models H))))$
by (simp add: ifthenelse-d-def)

lemma *len-defs* :
 $(w \models \text{len } n) = (\text{intlen } w = n)$
by (induct n arbitrary: w, simp add: len-d-def empty-defs,
simp add: len-d-def chop-defs skip-defs) fastforce

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

lemma *nextval-defs* :
 $(s \models v\$) = (\text{if } \text{intlen } s > 0 \ \text{then } (v (\text{nth } s \ 1)) \ \text{else } (\epsilon \ x. x=x))$
by (simp add: next-val-d-def)

lemma *finval-defs* :
 $(s \models !v) = (v (\text{nth } s \ (\text{intlen } s)))$
by (simp add: fin-val-d-def)

lemma *penultval-defs* :
 $(s \models v!) = (\text{if } \text{intlen } s > 0 \ \text{then } (v (\text{nth } s \ ((\text{intlen } s) - 1))) \ \text{else } (\epsilon \ x. x=x))$
by (simp add: penult-val-d-def)

lemma *next-assign-defs* :

$\text{intlen } s > 0 \implies (s \models v := e) = v (\text{Interval.nth } s \ 1) = e \ s$
by (auto simp: next-assign-d-def next-val-d-def)

lemma prev-assign-defs :

$\text{intlen } s > 0 \implies (s \models v := e) = v (\text{Interval.nth } s \ ((\text{intlen } s) - 1)) = e \ s$
by (auto simp: prev-assign-d-def penult-val-d-def)

lemma always-eqv-defs :

$(s \models v \approx e) = (\forall i \leq \text{intlen } s. v (\text{Interval.nth } s \ i) = e (\text{suffix } i \ s))$
by (simp add: always-eq-d-def always-defs current-val-d-def)

lemma temporal-assign-defs :

$(s \models v \leftarrow e) = (v (\text{Interval.nth } s \ (\text{intlen } s)) = e \ s)$
by (simp add: temporal-assign-d-def fin-val-d-def)

lemma gets-defs :

$(s \models v \text{ gets } e) = (\forall i < \text{intlen } s. v (\text{Interval.nth } s \ (\text{Suc } i)) = e (\text{sub } i \ (i+1) \ s))$
using Suc-le-eq
by (simp add: gets-d-def keep-d-def ba-defs skip-defs sub-def temporal-assign-defs)

lemma stable-defs-help:

$(\forall i < \text{intlen } s. v (\text{Interval.nth } s \ (\text{Suc } i)) = v (\text{Interval.nth } s \ i)) =$
 $(\forall i \leq \text{intlen } s. v (\text{Interval.nth } s \ i) = v (\text{Interval.nth } s \ 0))$

proof

(induct s)
case (St x)
then show ?case **by** simp
next
case (Cons x1a s)
then show ?case
by (smt Suc-less1 interval-nth-Suc intlen.simps(2) le-SucE le-neq-implies-less le-simps(1)
less-Suc-eq plus-1-eq-Suc zero-less-Suc)

qed

lemma stable-defs:

$(s \models \text{stable } v) = (\forall i \leq \text{intlen } s. (v (\text{nth } s \ i)) = (v (\text{nth } s \ 0)))$
by (simp add: stable-d-def gets-defs current-val-d-def sub-def stable-defs-help)

lemma padded-defs :

$(s \models \text{padded } v) = ((\forall i < \text{intlen } s. (v (\text{nth } s \ i)) = (v (\text{nth } s \ 0))) \vee \text{intlen } s = 0)$
by (simp add: padded-d-def stable-defs chop-d-def skip-defs empty-defs interval-suffix-length,
smt Suc-le1 Suc-pred diff-diff-cancel interval-intlen-gr-zero le-neq-implies-less le-simps(1)
less-Suc-eq)

lemma padded-temporal-assign-defs :

$(s \models v < \sim e) =$
 $((s \models \text{padded } v) \wedge$
 $(v (\text{Interval.nth } s \ (\text{intlen } s)) = e \ s))$
by (simp add: padded-temp-assign-d-def padded-defs temporal-assign-defs, auto)

lemma *linalw*:

$a \leq b \wedge b \leq \text{intlen } w \wedge ((\text{suffix } a \ w) \models \Box A) \longrightarrow ((\text{suffix } b \ w) \models \Box A)$

by (*simp add: always-defs*,

smt add.assoc add.commute interval-suffix-length-good le-add-diff-inverse le-trans
ordered-cancel-comm-monoid-diff-class.le-diff-conv2)

3.5 Soundness of Finite ITL Axioms

3.5.1 ChopAssoc

lemma *ChopAssocSemHelp*:

$(\exists i \text{ ia} . i \leq \text{intlen } \sigma \wedge \text{ia} \leq \text{intlen } \sigma - i \wedge (\text{prefix } i \ \sigma \models f) \wedge$
 $(\text{prefix } \text{ia} (\text{suffix } i \ \sigma) \models g) \wedge (\text{suffix } (\text{ia} + i) \ \sigma \models h)) =$
 $(\exists j \text{ ja} . j \leq \text{intlen } \sigma \wedge \text{ja} \leq j \wedge (\text{prefix } \text{ja} (\text{prefix } j \ \sigma) \models f) \wedge$
 $(\text{suffix } \text{ja} (\text{prefix } j \ \sigma) \models g) \wedge (\text{suffix } j \ \sigma \models h))$

by (*smt Nat.le-diff-conv2 add-diff-cancel-left' interval-pref-pref-3 interval-suffix-prefix-swap*
le-add1 le-add-diff-inverse2 le-trans)

lemma *ChopAssocSemHelp2*:

$(\sigma \models f ; (g ; h)) = (\sigma \models (f;g);h)$

proof –

have $(\sigma \models f ; (g ; h)) =$
 $((\exists i \leq \text{intlen } \sigma . (\text{prefix } i \ \sigma \models f) \wedge (\exists \text{ia} \leq \text{intlen } (\text{suffix } i \ \sigma) .$
 $(\text{prefix } \text{ia} (\text{suffix } i \ \sigma) \models g) \wedge (\text{suffix } (\text{ia} + i) \ \sigma \models h))))$

by (*simp add: chop-defs*)

also have ... =

$(\exists i \text{ ia} . i \leq \text{intlen } \sigma \wedge \text{ia} \leq \text{intlen } \sigma - i \wedge (\text{prefix } i \ \sigma \models f) \wedge$
 $(\text{prefix } \text{ia} (\text{suffix } i \ \sigma) \models g) \wedge (\text{suffix } (\text{ia} + i) \ \sigma \models h))$

by *fastforce*

also have ... =

$(\exists j \text{ ja} . j \leq \text{intlen } \sigma \wedge \text{ja} \leq j \wedge (\text{prefix } \text{ja} (\text{prefix } j \ \sigma) \models f) \wedge$
 $(\text{suffix } \text{ja} (\text{prefix } j \ \sigma) \models g) \wedge (\text{suffix } j \ \sigma \models h))$

using *ChopAssocSemHelp*[of $\sigma \ f \ g \ h$] **by** *blast*

also have ... =

$(\exists i \leq \text{intlen } \sigma . (\exists \text{ia} \leq \text{intlen } (\text{prefix } i \ \sigma) . (\text{prefix } \text{ia} (\text{prefix } i \ \sigma) \models f) \wedge$
 $(\text{suffix } \text{ia} (\text{prefix } i \ \sigma) \models g)) \wedge (\text{suffix } i \ \sigma \models h))$

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*:

$(\sigma \models f ; (g ; h)) = (\sigma \models (f;g);h)$

using *ChopAssocSemHelp2* **using** *unl-lift2* **by** *blast*

3.5.2 OrChopImp

lemma *OrChopImpSem*:

$(\sigma \models (f \vee g);h \longrightarrow f;h \vee g;h)$

by (*simp add: chop-defs*) *blast*

3.5.3 ChopOrImp

lemma *ChopOrImpSem*:

$(\sigma \models f; (g \vee h) \longrightarrow f; g \vee f; h)$

by (*simp add: chop-defs*) *blast*

3.5.4 EmptyChop

lemma *EmptyChopSem*:

$(\sigma \models \text{empty} ; f = f)$

by (*simp add: empty-defs chop-defs*) *auto*

3.5.5 ChopEmpty

lemma *ChopEmptySem*:

$(\sigma \models f; \text{empty} = f)$

by (*simp add: empty-defs chop-defs*) *auto*

3.5.6 StateImpBi

lemma *StateImpBiSem*:

$(\sigma \models \text{init } f \longrightarrow \text{bi } (\text{init } f))$

by (*simp add: init-defs bi-defs*)

3.5.7 NextImpNotNextNot

lemma *NextImpNotNextNotSem*:

$(\sigma \models \bigcirc f \longrightarrow \neg (\bigcirc (\neg f)))$

by (*simp add: next-defs*)

3.5.8 BiBoxChopImpChop

lemma *BiBoxChopImpChopSem*:

$(\sigma \models \text{bi } (f \longrightarrow f1) \wedge \Box (g \longrightarrow g1) \longrightarrow f; g \longrightarrow f1; g1)$

by (*simp add: bi-defs always-defs chop-defs*) *fastforce*

3.5.9 BoxInduct

lemma *box-induct-help-1* :

$(\sigma \models f) \wedge (\forall i. \text{Suc } 0 \leq \text{intlen } \sigma - i \longrightarrow$
 $i \leq \text{intlen } \sigma \longrightarrow (\text{suffix } i \sigma \models f) \longrightarrow (\text{suffix } (\text{Suc } i) \sigma \models f))$
 $\implies (\forall j. j \leq \text{intlen } \sigma \longrightarrow (\text{suffix } j \sigma \models f))$

proof

fix *j*

show $(\sigma \models f) \wedge (\forall i. \text{Suc } 0 \leq \text{intlen } \sigma - i \longrightarrow$
 $i \leq \text{intlen } \sigma \longrightarrow (\text{suffix } i \sigma \models f) \longrightarrow (\text{suffix } (\text{Suc } i) \sigma \models f))$
 $\implies j \leq \text{intlen } \sigma \longrightarrow (\text{suffix } j \sigma \models f)$

proof

(induct j arbitrary: σ)

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: ( $\sigma \models \Box (f \longrightarrow \text{wnext } f) \wedge f \longrightarrow \Box f$ ) =
    (( $\forall n \leq \text{intlen } \sigma. f (\text{suffix } n \sigma) \longrightarrow \text{intlen } \sigma = n \vee f (\text{suffix } (\text{Suc } n) \sigma)$ )  $\wedge f \sigma \longrightarrow$ 
     ( $\forall n \leq \text{intlen } \sigma. f (\text{suffix } n \sigma)$ ))
    by (simp add: always-defs wnext-defs)
  from 1 show ?thesis using box-induct-help-1
    by (metis One-nat-def diff-self-eq-0 not-one-le-zero)
qed

```

3.5.10 ChopStarEqv

lemma *ChopExist*:

```

 $\vdash (\exists k. f;g \ k) = f;(\exists k. g \ k)$ 
by (simp add: chop-defs Valid-def, auto)

```

lemma *ExistChop*:

```

 $\vdash (\exists k. (g \ k);f) = (\exists k. g \ k);f$ 
by (simp add: chop-defs Valid-def, auto)

```

lemma *powersem1*:

```

( $\sigma \models (\exists k. \text{power } f \ k) = (\text{empty} \vee (\exists k. \text{power } f (\text{Suc } k)))$ )
by (smt not0-implies-Suc pow-0 unl-Rex unl-lift2)

```

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)

```

3.6 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.

typeddecl state

instance state :: world ..

```

type-synonym 'a statefun = (state, 'a) stfun
type-synonym statepred   = bool statefun
type-synonym 'a tempfun  = (state, 'a) formfun
type-synonym temporal    = state formula

```

Similar to [3, 2] we define a state to be an anonymous type whose only purpose is to provide Skolem constants. 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.

3.7 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 $\vee A == Eex \vee. A$

definition forall-state-d :: ('a statefun \Rightarrow temporal) \Rightarrow temporal (**binder** Aall 10)
where forall-state-d F $\equiv \text{LIFT}(\neg(\exists \exists \exists x. \neg(F \ x)))$

syntax

-Aall :: [idts, lift] \Rightarrow lift $((\exists \forall \forall \text{ -./ -}) [0,10] 10)$

translations

$\neg A \text{all } v \ A \equiv A \text{all } v. \ A$

end

4 Old vs new definition of Chopstar

theory *AltChopstarSem*

imports *Semantics*

begin

We show that the old and new definition of chopstar are the same.

4.1 Definition

definition *chopstar-d-old* :: ('a::world) formula \Rightarrow 'a formula

where *chopstar-d-old* $F \equiv \lambda s. (\exists (I::\text{index}). \text{index-sequence } 0 \ I \ \wedge \ (\text{nth } I \ (\text{intlen } I)) = (\text{intlen } s) \ \wedge$
 $(\forall i. (0 \leq i \ \wedge \ i < (\text{intlen } I)) \longrightarrow$
 $((\text{sub } (\text{nth } I \ i) \ (\text{nth } I \ (i+1))) \ s) \models F)$
 $)$
 $)$

syntax

-chopstar-d-old :: lift \Rightarrow lift $((\text{chopstarold } -) \ [85] \ 85)$

syntax (*ASCII*)

-chopstar-d-old :: lift \Rightarrow lift $((\text{chopstarold } -) \ [85] \ 85)$

translations

-chopstar-d-old $\equiv \text{CONST } \text{chopstar-d-old}$

4.2 Lemmas

lemma *chopstar-help-1*:

$(\exists I. I = \langle 0 \rangle \ \wedge \ \text{index-sequence } 0 \ I \ \wedge$
 $\text{Interval.nth } I \ (\text{intlen } I) = (\text{intlen } \sigma) \ \wedge$
 $(\forall i. (0 \leq i \ \wedge \ i < (\text{intlen } I)) \longrightarrow$
 $((\text{sub } (\text{nth } I \ i) \ (\text{nth } I \ (i+1))) \ \sigma) \models f)$
 $) \longleftrightarrow (\text{intlen } \sigma = 0)$

by (*simp add: index-sequence-def*)

lemma *chopstar-help-2*:

$(\forall i. (0 < i \ \wedge \ i < 1 + (\text{intlen } I)) \longrightarrow$
 $((\text{sub } (\text{nth } I \ (i-1)) \ (\text{nth } I \ ((i-1)+1))) \ \sigma) \models f)$
 $) =$
 $(\forall i. (0 \leq i \ \wedge \ i < (\text{intlen } I)) \longrightarrow$
 $((\text{sub } (\text{nth } I \ i) \ (\text{nth } I \ ((i)+1))) \ \sigma) \models f)$
 $)$

by (metis Suc-eq-plus1 Suc-pred add-diff-cancel-right' add-less-cancel-left
add-nonneg-pos le-add2 le-add-same-cancel2 plus-1-eq-Suc zero-less-one)

lemma chop-power-chain:

$(\exists (l::index). (intlen\ l) = (Suc\ n) \wedge index_sequence\ 0\ l \wedge (nth\ l\ (intlen\ l)) = (intlen\ \sigma) \wedge$
 $(\forall i. (0 \leq i \wedge i < (intlen\ l)) \longrightarrow$
 $((sub\ (nth\ l\ i)\ (nth\ l\ (i+1)))\ \sigma) \models f)$
 $)$
 $) =$
 $(\exists\ k. 0 \leq k \wedge k \leq intlen\ \sigma \wedge k > 0 \wedge$
 $(sub\ 0\ k\ \sigma \models f) \wedge$
 $(\exists\ ls. (intlen\ ls) = n \wedge index_sequence\ 0\ (ls) \wedge$
 $(nth\ (ls)\ (intlen\ (ls))) = (intlen\ (suffix\ k\ \sigma))$
 $\wedge (\forall i. (0 \leq i \wedge i < (intlen\ ls)) \longrightarrow$
 $((sub\ (nth\ ls\ i)\ (nth\ ls\ ((i)+1)))\ (suffix\ k\ \sigma)) \models f)$
 $))$
 $)$

proof –

have $(\exists (l::index). (intlen\ l) = (Suc\ n) \wedge index_sequence\ 0\ l \wedge$
 $(nth\ l\ (intlen\ l)) = (intlen\ \sigma) \wedge$
 $(\forall i. (0 \leq i \wedge i < (intlen\ l)) \longrightarrow$
 $((sub\ (nth\ l\ i)\ (nth\ l\ (i+1)))\ \sigma) \models f)$
 $)$
 $)$
 $=$
 $(\exists\ x\ ls\ l. (intlen\ l) = (Suc\ n) \wedge l = x \odot ls \wedge index_sequence\ 0\ l \wedge$
 $(nth\ l\ (intlen\ l)) = (intlen\ \sigma) \wedge$
 $(\forall i. (0 \leq i \wedge i < (intlen\ l)) \longrightarrow$
 $((sub\ (nth\ l\ i)\ (nth\ l\ (i+1)))\ \sigma) \models f)$
 $)$
 $)$

by (metis interval-intlen-cons-1 zero-less-Suc)

also have ... =

$(\exists\ x\ ls\ l. (intlen\ l) = (Suc\ n) \wedge l = x \odot ls \wedge index_sequence\ 0\ (x \odot ls) \wedge$
 $(nth\ (x \odot ls)\ (intlen\ (x \odot ls))) = (intlen\ \sigma) \wedge$
 $(\forall i. (0 \leq i \wedge i < (intlen\ (x \odot ls))) \longrightarrow$
 $((sub\ (nth\ (x \odot ls)\ i)\ (nth\ (x \odot ls)\ (i+1)))\ \sigma) \models f)$
 $)$
 $)$

by auto

also have ... =

$(\exists\ x\ ls. (intlen\ ls) = n \wedge index_sequence\ 0\ (x \odot ls) \wedge$
 $(nth\ (x \odot ls)\ (intlen\ (x \odot ls))) = (intlen\ \sigma) \wedge$
 $(\forall i. (0 \leq i \wedge i < (intlen\ (x \odot ls))) \longrightarrow$
 $((sub\ (nth\ (x \odot ls)\ i)\ (nth\ (x \odot ls)\ (i+1)))\ \sigma) \models f)$
 $)$
 $)$

by auto

also have ... =

$(\exists\ x\ ls. (intlen\ ls) = n \wedge x = 0 \wedge index_sequence\ 0\ (x \odot ls) \wedge$

```

      (nth (ls) (intlen (ls))) = (intlen σ) ∧
      ((∀ i. (0 ≤ i ∧ i < (intlen (x ⊙ ls))) →
        ((sub (nth (x ⊙ ls) i) (nth (x ⊙ ls) (i+1)) σ) ⊨ f))
    )
  )
)
by (simp add: index-sequence-def)
also have ... =
  (∃ x ls . (intlen ls) = n ∧ x = 0 ∧ index-sequence (nth ls 0) (ls) ∧
    (nth (ls) (intlen (ls))) = (intlen σ) ∧
    (x < (nth ls 0) ∧
      ((∀ i. (0 ≤ i ∧ i < (intlen (x ⊙ ls))) →
        ((sub (nth (x ⊙ ls) i) (nth (x ⊙ ls) (i+1)) σ) ⊨ f))
    )
  )
)
)
using interval-idx-cons by auto
also have ... =
  (∃ x ls . (intlen ls) = n ∧ x = 0 ∧ index-sequence (nth ls 0) (ls) ∧
    (nth (ls) (intlen (ls))) = (intlen σ) ∧
    (x < (nth ls 0) ∧
      ((sub (nth (x ⊙ ls) 0) (nth (x ⊙ ls) (1)) σ) ⊨ f)
    )
    ∧
    ((∀ i. (0 < i ∧ i < 1 + (intlen (ls))) →
      ((sub (nth (x ⊙ ls) i) (nth (x ⊙ ls) (i+1)) σ) ⊨ f))
    )
  )
)
)
by (metis (no-types, lifting) One-nat-def add.right-neutral add-Suc add-Suc-right
  add-cancel-right-left interval-intlen-cons not-gr-zero zero-le zero-less-Suc)
also have ... =
  (∃ x ls . (intlen ls) = n ∧ x = 0 ∧ index-sequence (nth ls 0) (ls) ∧
    (nth (ls) (intlen (ls))) = (intlen σ) ∧
    (x < (nth ls 0) ∧ (nth (x ⊙ ls) 0) = x ∧ (nth (x ⊙ ls) (1)) = (nth ls 0) ∧
      ((sub (nth (x ⊙ ls) 0) (nth (x ⊙ ls) (1)) σ) ⊨ f)
    )
    ∧
    ((∀ i. (0 < i ∧ i < 1 + (intlen (ls))) →
      ((sub (nth (x ⊙ ls) i) (nth (x ⊙ ls) (i+1)) σ) ⊨ f))
    )
  )
)
)
by auto
also have ... =
  (∃ x ls . (intlen ls) = n ∧ x = 0 ∧ index-sequence (nth ls 0) (ls) ∧
    (nth (ls) (intlen (ls))) = (intlen σ) ∧
    (x < (nth ls 0) ∧ (nth (x ⊙ ls) 0) = x ∧ (nth (x ⊙ ls) (1)) = (nth ls 0) ∧
      ((sub x (nth ls 0) σ) ⊨ f)
    )
    ∧
    ((∀ i. (0 < i ∧ i < 1 + (intlen (ls))) →
      ((sub (nth (x ⊙ ls) i) (nth (x ⊙ ls) (i+1)) σ) ⊨ f))
    )
  )
)

```

)
)
by auto
also have ... =
 ($\exists x \text{ } ls . (\text{intlen } ls) = n \wedge x = 0 \wedge \text{index-sequence } (nth \text{ } ls \text{ } 0) (ls) \wedge$
 $(nth (ls) (\text{intlen } (ls))) = (\text{intlen } \sigma) \wedge$
 $(x < (nth \text{ } ls \text{ } 0) \wedge$
 $((sub \text{ } x (nth \text{ } ls \text{ } 0) \sigma) \models f)$
 \wedge
 $((\forall i. (0 < i \wedge i < 1 + (\text{intlen } (ls))) \longrightarrow$
 $((sub (nth (x \odot ls) i) (nth (x \odot ls) (i+1)) \sigma) \models f))$
 $)$
 $)$
 $)$

by auto
also have ... =
 ($\exists x \text{ } ls . (\text{intlen } ls) = n \wedge x = 0 \wedge \text{index-sequence } (nth \text{ } ls \text{ } 0) (ls) \wedge$
 $(nth (ls) (\text{intlen } (ls))) = (\text{intlen } \sigma) \wedge$
 $(x < (nth \text{ } ls \text{ } 0) \wedge$
 $((sub \text{ } x (nth \text{ } ls \text{ } 0) \sigma) \models f)$
 \wedge
 $(\forall i. (0 < i \wedge i < 1 + (\text{intlen } ls)) \longrightarrow$
 $((sub (nth \text{ } ls (i-1)) (nth \text{ } ls ((i-1)+1)) \sigma) \models f)$
 $)))$

using interval-nth-cons by metis

also have ... =
 ($\exists x \text{ } ls . (\text{intlen } ls) = n \wedge x = 0 \wedge \text{index-sequence } (nth \text{ } ls \text{ } 0) (ls) \wedge$
 $(nth (ls) (\text{intlen } (ls))) = (\text{intlen } \sigma) \wedge$
 $(x < (nth \text{ } ls \text{ } 0) \wedge$
 $((sub \text{ } x (nth \text{ } ls \text{ } 0) \sigma) \models f))$
 $\wedge (\forall i. (0 \leq i \wedge i < (\text{intlen } ls)) \longrightarrow$
 $((sub (nth \text{ } ls (i)) (nth \text{ } ls ((i)+1)) \sigma) \models f)$
 $)$
 $)$

using chopstar-help-2 by (metis (mono-tags))

also have ... =
 ($\exists \text{ } ls . (\text{intlen } ls) = n \wedge \text{index-sequence } (nth \text{ } ls \text{ } 0) (ls) \wedge$
 $(nth (ls) (\text{intlen } (ls))) = (\text{intlen } \sigma) \wedge$
 $(0 < (nth \text{ } ls \text{ } 0) \wedge$
 $((sub \text{ } 0 (nth \text{ } ls \text{ } 0) \sigma) \models f))$
 $\wedge (\forall i. (0 \leq i \wedge i < (\text{intlen } ls)) \longrightarrow$
 $((sub (nth \text{ } ls (i)) (nth \text{ } ls ((i)+1)) \sigma) \models f)$
 $)$
 $)$

by simp

also have ... =
 ($\exists \text{ } lsk . (\text{intlen } lsk) = n \wedge (nth \text{ } lsk \text{ } 0) \leq \text{intlen } \sigma \wedge (nth \text{ } lsk \text{ } 0) > 0 \wedge$
 $((sub \text{ } 0 (nth \text{ } lsk \text{ } 0) \sigma) \models f) \wedge$
 $\text{index-sequence } (nth \text{ } lsk \text{ } 0) (lsk) \wedge$
 $(nth (lsk) (\text{intlen } (lsk))) = (\text{intlen } \sigma) \wedge$
 $)$

```

      (∀ i. (0 ≤ i ∧ i < (intlen lsk)) →
        ((sub (nth lsk (i)) (nth lsk ((i)+1)) σ) ⊨ f)
      )
    )
  by (metis Suc-eq-plus1 Suc-pred add.left-neutral eq-iff interval-idx-less-last
    interval-intlen-gr-zero le-neq-implies-less lessl less-imp-le-nat)
  also have ... =
    (∃ k lsk. (intlen lsk) = n ∧ (nth lsk 0) ≤ intlen σ ∧
      (nth lsk 0) > 0 ∧ k = (nth lsk 0) ∧
      (sub 0 (nth lsk 0) σ ⊨ f) ∧
      index-sequence (nth lsk 0) (lsk) ∧
      (nth (lsk) (intlen (lsk))) = (intlen (σ)) ∧
      (∀ i. (0 ≤ i ∧ i < (intlen lsk)) →
        ((sub ((nth lsk (i))) ((nth lsk ((i)+1)))) (σ)) ⊨ f)
      )
    )
  by auto
  also have ... =
    (∃ k lsk. (intlen lsk) = n ∧ 0 ≤ k ∧ k ≤ intlen σ ∧ k > 0 ∧ k = (nth lsk 0) ∧
      (sub 0 k σ ⊨ f) ∧
      (index-sequence k (lsk) ∧
        (nth (lsk) (intlen (lsk))) = ((intlen (suffix k σ)) + k) ∧
        (∀ i. (0 ≤ i ∧ i < (intlen lsk)) →
          ((sub ((nth lsk (i))) ((nth lsk ((i)+1)))) (σ)) ⊨ f)
        ))
      )
  by (simp add: interval-prefix-suffix-intlen interval-suffix-length interval-prefix-length, auto)
  also have ... =
    (∃ k lsk. (intlen lsk) = n ∧ 0 ≤ k ∧ k ≤ intlen σ ∧ k > 0 ∧
      (sub 0 k σ ⊨ f) ∧
      (index-sequence k (lsk) ∧
        (nth (lsk) (intlen (lsk))) = ((intlen (suffix k σ)) + k) ∧
        (∀ i. (0 ≤ i ∧ i < (intlen lsk)) →
          ((sub ((nth lsk (i))) ((nth lsk ((i)+1)))) (σ)) ⊨ f)
        ))
      )
  using index-sequence-def by auto
  also have ... =
    (∃ k. 0 ≤ k ∧ k ≤ intlen σ ∧ k > 0 ∧
      (sub 0 k σ ⊨ f) ∧
      (∃ ls lsk. (intlen lsk) = n ∧ index-sequence k (lsk) ∧
        ls = map (shiftm k) lsk ∧
        (nth (lsk) (intlen (lsk))) = ((intlen (suffix k σ)) + k) ∧
        (∀ i. (0 ≤ i ∧ i < (intlen lsk)) →
          ((sub ((nth lsk (i))) ((nth lsk ((i)+1)))) (σ)) ⊨ f)
        ))
      )
  by blast
  also have ... =
    (∃ k. 0 ≤ k ∧ k ≤ intlen σ ∧ k > 0 ∧

```

```

(sub 0 k σ ⊨ f) ∧
(∃ ls lsk. (intlen lsk) = n ∧ index-sequence k (lsk) ∧
  ls = map (shiftm k) lsk ∧
  index-sequence 0 (ls) ∧ (intlen ls) = n ∧
  (nth (lsk) (intlen (lsk))) = ((intlen (suffix k σ))+k) ∧
  (∀ i. (0 ≤ i ∧ i < (intlen lsk)) →
    ((sub ((nth lsk (i))) ((nth lsk ((i)+1))) (σ)) ⊨ f)
  ))
)
using interval-idx-link-shiftm by blast
also have ... =
(∃ k. 0 ≤ k ∧ k ≤ intlen σ ∧ k > 0 ∧
  (sub 0 k σ ⊨ f) ∧
  (∃ ls lsk. (intlen lsk) = n ∧ index-sequence k (lsk) ∧
    lsk = map (shift k) ls ∧
    index-sequence 0 (ls) ∧ (intlen ls) = n ∧
    (nth (lsk) (intlen (lsk))) = ((intlen (suffix k σ))+k) ∧
    (∀ i. (0 ≤ i ∧ i < (intlen lsk)) →
      ((sub ((nth lsk (i))) ((nth lsk ((i)+1))) (σ)) ⊨ f)
    ))
  ))
)
using interval-lsk-ls by blast
also have ... =
(∃ k ls lsk. 0 ≤ k ∧ k ≤ intlen σ ∧ k > 0 ∧
  (sub 0 k σ ⊨ f) ∧
  ((intlen lsk) = n ∧ lsk = map (shift k) ls ∧
    index-sequence 0 (ls) ∧
    index-sequence k (lsk) ∧
    (nth (ls) (intlen (ls))) = (intlen (suffix k σ)) ∧
    (∀ i. (0 ≤ i ∧ i < (intlen ls)) →
      ((sub ((nth ls (i))+k) ((nth ls ((i)+1))+k) (σ)) ⊨ f)
    ))
  ))
)
by (simp add: Interval.shift-def interval-nth-map, blast)
also have ... =
(∃ k ls lsk. 0 ≤ k ∧ k ≤ intlen σ ∧ k > 0 ∧
  (sub 0 k σ ⊨ f) ∧
  ((intlen lsk) = n ∧ lsk = map (shift k) ls ∧
    (intlen ls) = n ∧ index-sequence 0 (ls) ∧
    (nth (ls) (intlen (ls))) = (intlen (suffix k σ)) ∧
    (∀ i. (0 ≤ i ∧ i < (intlen ls)) →
      ((sub ((nth ls (i))+k) ((nth ls ((i)+1))+k) (σ)) ⊨ f)
    ))
  ))
)
using interval-idx-link by blast
also have ... =
(∃ k. 0 ≤ k ∧ k ≤ intlen σ ∧ k > 0 ∧
  (sub 0 k σ ⊨ f) ∧
  (∃ ls. (intlen ls) = n ∧ index-sequence 0 (ls) ∧
    (nth (ls) (intlen (ls))) = (intlen (suffix k σ)) ∧

```

```

    (∀ i. (0 ≤ i ∧ i < (intlen ls)) →
      ((sub ((nth ls (i)) + k) ((nth ls ((i) + 1)) + k) (σ)) ⊨ f)
    ))
  )
by (simp)
also have ... =
  (∃ k. 0 ≤ k ∧ k ≤ intlen σ ∧ k > 0 ∧
    (sub 0 k σ ⊨ f) ∧
    (∃ ls. (intlen ls) = n ∧ index-sequence 0 (ls) ∧
      (nth (ls) (intlen (ls))) = (intlen (suffix k σ)) ∧
      (∀ i ≤ intlen ls. Interval.nth ls i ≤ intlen (suffix k σ)) ∧
      (∀ i. (0 ≤ i ∧ i < (intlen ls)) →
        ((sub ((nth ls (i)) + k) ((nth ls ((i) + 1)) + k) (σ)) ⊨ f)
      )
    )
  )
)
using interval-idx-bound-1 by blast
also have ... =
  (∃ k. 0 ≤ k ∧ k ≤ intlen σ ∧ k > 0 ∧
    (sub 0 k σ ⊨ f) ∧
    (∃ ls. (intlen ls) = n ∧ index-sequence 0 (ls) ∧
      (nth (ls) (intlen (ls))) = (intlen (suffix k σ)) ∧
      (∀ i ≤ intlen ls. Interval.nth ls i ≤ intlen (suffix k σ))
      ∧ (∀ i. (0 ≤ i ∧ i < (intlen ls)) →
        ((sub (nth ls (i)) (nth ls ((i) + 1)) (suffix k σ)) ⊨ f)
      )
    )
  )
)
by (smt add.commute index-sequence-def interval-idx-expand interval-sub-suffix
  interval-suffix-length-good plus-1-eq-Suc)
also have ... =
  (∃ k. 0 ≤ k ∧ k ≤ intlen σ ∧ k > 0 ∧
    (sub 0 k σ ⊨ f) ∧
    (∃ ls. (intlen ls) = n ∧ index-sequence 0 (ls) ∧
      (nth (ls) (intlen (ls))) = (intlen (suffix k σ))
      ∧ (∀ i. (0 ≤ i ∧ i < (intlen ls)) →
        ((sub (nth ls (i)) (nth ls ((i) + 1)) (suffix k σ)) ⊨ f)
      )
    )
  )
)
using interval-idx-bound-1 by blast
finally show (∃ (l::index). (intlen l) = (Suc n) ∧ index-sequence 0 l ∧
  (nth l (intlen l)) = (intlen σ) ∧
  (∀ i. (0 ≤ i ∧ i < (intlen l)) →
    ((sub (nth l i) (nth l (i + 1)) σ) ⊨ f)
  )
) =
  (∃ k. 0 ≤ k ∧ k ≤ intlen σ ∧ k > 0 ∧ (sub 0 k σ ⊨ f) ∧
    (∃ ls. (intlen ls) = n ∧ index-sequence 0 (ls) ∧
      (nth (ls) (intlen (ls))) = (intlen (suffix k σ)) ∧
      (∀ i. (0 ≤ i ∧ i < (intlen ls)) →

```



```

      ((sub (nth ls (i)) (nth ls ((i)+1)) (suffix k σ)) ⊨ f)
    )
  )
)
.

```

qed

lemma chop-power-equiv-sem:

```

  ( (σ ⊨ (∃ n. (power (f ∧ more) n))) =
    ((σ ⊨ empty) ∨ ( (σ ⊨ (f ∧ more); (∃ n. (power (f ∧ more) n)))))
using ChopstarEqvSem powerstar-d-def chopstar-d-def
by (metis (mono-tags, lifting) unl-lift2)

```

lemma chopstar-equiv-power-chop-help:

```

  ( σ ⊨ power (f ∧ more) n ) =
  (∃ (l::index). intlen(l) = n ∧ index-sequence 0 l ∧
    (nth l (intlen l)) = (intlen (σ)) ∧
    (∀ i. (0 ≤ i ∧ i < (intlen l)) →
      ((sub (nth l i) (nth l (i+1)) (σ)) ⊨ f)
    )
  )
)

```

proof

(induct n arbitrary: σ)

case 0

then show ?case **using** index-sequence-def chopstar-help-1 empty-defs

by (metis (mono-tags, lifting) intlen.simps(1) pow-0)

next

case (Suc n)

then show ?case

proof –

have 1: (σ ⊨ power (f ∧ more) (Suc n)) = (σ ⊨ ((f ∧ more); (power (f ∧ more) n)))

by simp

have 2: (σ ⊨ ((f ∧ more); (power (f ∧ more) n))) =

(∃ k. 0 ≤ k ∧ k ≤ intlen (σ) ∧ k > 0 ∧

(prefix k (σ) ⊨ f) ∧

(suffix k (σ) ⊨ power (f ∧ more) n)

)

by (simp add: more-defs chop-defs) auto

have 3: (∃ k. 0 ≤ k ∧ k ≤ intlen (σ) ∧ k > 0 ∧

(prefix k (σ) ⊨ f) ∧

(suffix k (σ) ⊨ power (f ∧ more) n)

) =

(∃ k. 0 ≤ k ∧ k ≤ intlen (σ) ∧ k > 0 ∧

(sub 0 k (σ) ⊨ f) ∧

(suffix k (σ) ⊨ power (f ∧ more) n)

)

by (simp add: interval-sub-zero-prefix)

have 31: $\bigwedge k. ((\text{suffix } k \text{ } \sigma) \models \text{power } (f \wedge \text{more}) \text{ } n) =$

(∃ (l::index). intlen(l) = n ∧ index-sequence 0 l ∧

```

      (nth l (intlen l)) = (intlen (suffix k σ)) ∧
      (∀ i. (0 ≤ i ∧ i < (intlen l)) →
        ((sub (nth l i) (nth l (i+1))) (suffix k σ)) ⊨ f)
    )
  )
  by (simp add: Suc.hyps)
have 4: (∃ k. 0 ≤ k ∧ k ≤ intlen σ ∧ k > 0 ∧
  (sub 0 k σ) ⊨ f) ∧
  (suffix k σ) ⊨ power (f ∧ more) n)
) =
(∃ k. 0 ≤ k ∧ k ≤ intlen σ ∧ k > 0 ∧
  (sub 0 k σ) ⊨ f) ∧
  (∃ (l::index). intlen l = n ∧ index-sequence 0 l ∧
    (nth l (intlen l)) = (intlen (suffix k σ)) ∧
    (∀ i. (0 ≤ i ∧ i < (intlen l)) →
      ((sub (nth l i) (nth l (i+1))) (suffix k σ)) ⊨ f)
    )
  )
)
)
using 31 by simp
have 5:
  (∃ (l::index). (intlen l) = (Suc n) ∧ index-sequence 0 l ∧
    (nth l (intlen l)) = (intlen σ) ∧
    (∀ i. (0 ≤ i ∧ i < (intlen l)) →
      ((sub (nth l i) (nth l (i+1))) σ) ⊨ f)
    )
  ) =
  (∃ k. 0 ≤ k ∧ k ≤ intlen σ ∧ k > 0 ∧
    (sub 0 k σ ⊨ f) ∧
    (∃ ls. (intlen ls) = n ∧ index-sequence 0 (ls) ∧
      (nth (ls) (intlen (ls))) = (intlen (suffix k σ)) ∧
      (∀ i. (0 ≤ i ∧ i < (intlen ls)) →
        ((sub (nth ls (i)) (nth ls ((i)+1))) (suffix k σ)) ⊨ f)
      )
    )
  )
)
)
using chop-power-chain by simp
from 1 2 3 4 5 show ?thesis by blast
qed
qed

lemma chopstar-equiv-power-chop:
  (σ ⊨ chopstarold f) = ( σ ⊨ (∃ k. power (f ∧ more) k)))
by (simp add: chopstar-d-old-def chopstar-equiv-power-chop-help)

lemma OldChopstarEqvSem:
  (σ ⊨ (chopstarold f = (empty ∨ (f ∧ more); (chopstarold f))))
using chopstar-equiv-power-chop chop-power-equiv-sem
by (smt chop-defs unl-lift2)

```

lemma *OldChopstarEqvChopstar*:

$\vdash (\text{chopstarold } f) = f^*$

by (*simp add: Valid-def chopstar-d-def chopstar-eqv-power-chop powerstar-d-def*)

end

5 Infinite ITL Semantics

theory *InfiniteSemantics*

imports *InfiniteInterval HOL-TLA.Intensional*

begin

This theory mechanises a *shallow* embedding of Infinite ITL using the *InfiniteInterval* and *Intensional* theories. A similar embedding as finite ITL has been used with the difference that we use a sum type which is either a finite or infinite interval.

5.1 Types of Formulas

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

type-synonym *'a intervals* = *'a interval* + *'a infinterval*

type-synonym (*'a, 'b*) *formfun* = *'a intervals* \Rightarrow *'b*

type-synonym (*'a, 'b*) *finformfun* = *'a interval* \Rightarrow *'b*

type-synonym (*'a, 'b*) *infformfun* = *'a infinterval* \Rightarrow *'b*

type-synonym *'a formula* = (*'a, bool*) *formfun*

type-synonym *'a finformula* = (*'a, bool*) *finformfun*

type-synonym *'a infformula* = (*'a, bool*) *infformfun*

type-synonym (*'a, 'b*) *stfun* = *'a* \Rightarrow *'b*

type-synonym *'a stpred* = (*'a, bool*) *stfun*

instance

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

instance

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

instance

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

instance

interval :: (*type*) *world* ..

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

5.2 Semantics of ITL

The semantics of ITL is defined.

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

where

skip-d $\equiv (\lambda s. (\text{case } s \text{ of } (\text{Inl } s) \Rightarrow (\text{intlen } s = 1) \mid (\text{Inr } s) \Rightarrow \text{False}))$

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

where

chop-d *F1 F2* \equiv

($\lambda s.$
 (case *s* of (*Inl s*) \Rightarrow
 ($\exists n. n \geq 0 \wedge n \leq \text{intlen } s \wedge ((\text{Inl } (\text{prefix } n \text{ } s)) \models F1) \wedge ((\text{Inl } (\text{suffix } n \text{ } s)) \models F2)$)
 |
 (*Inr s*) \Rightarrow
 ($(\exists n. ((\text{Inl } (\text{iprefix } n \text{ } s)) \models F1) \wedge ((\text{Inr } (\text{isuffix } n \text{ } s)) \models F2))$
 $\vee ((\text{Inr } s) \models F1)$
)
)
)
)

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

where

current-val-d *f* $= (\lambda s. (\text{case } s \text{ of } (\text{Inl } s) \Rightarrow ((\text{nth } s \text{ } 0) \models f) \mid (\text{Inr } s) \Rightarrow ((s \text{ } 0) \models f)))$

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

where *next-val-d* *f* \equiv

($\lambda s. (\text{case } s \text{ of } (\text{Inl } s) \Rightarrow \text{if } \text{intlen } s > 0 \text{ then } ((\text{nth } s \text{ } 1) \models f) \text{ else } (\epsilon (x :: 'b). x = x)$
 $\mid (\text{Inr } s) \Rightarrow ((s \text{ } 1) \models f)$
)
)

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

where *fin-val-d* *f* $\equiv \lambda s. (\text{case } s \text{ of } (\text{Inl } s) \Rightarrow (\text{nth } s (\text{intlen } s)) \models f$
 $\mid (\text{Inr } s) \Rightarrow (\epsilon (x :: 'b). x = x))$

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

where *penult-val-d* *f* \equiv

($\lambda s.$
 (case *s* of (*Inl s*) $\Rightarrow \text{if } \text{intlen } s > 0 \text{ then } (\text{nth } s ((\text{intlen } s) - 1)) \models f \text{ else } (\epsilon (x :: 'b). x = x)$
 $\mid (\text{Inr } s) \Rightarrow (\epsilon (x :: 'b). x = x)$
)
)

syntax

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

$\text{-penult-val-d} :: \text{lift} \Rightarrow \text{lift} \quad ((\text{-!}) [100] 99)$
 $\text{TEMP} :: \text{lift} \Rightarrow 'b \quad ((\text{TEMP -}))$

syntax (ASCII)

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

translations

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

5.3 Abbreviations

Some standard temporal abbreviations, with their concrete syntax.

definition $\text{infinite-d} :: ('a :: \text{world}) \text{ formula}$

where

$\text{infinite-d} \equiv \text{LIFT}(\# \text{True}; \# \text{False})$

syntax

$\text{-infinite-d} :: \text{lift} \quad (\text{inf})$

syntax (ASCII)

$\text{-infinite-d} :: \text{lift} \quad (\text{inf})$

translations

$\text{-infinite-d} \Rightarrow \text{CONST infinite-d}$

definition $\text{finite-d} :: ('a :: \text{world}) \text{ formula}$

where

$\text{finite-d} \equiv \text{LIFT}(\neg(\text{inf}))$

syntax

$\text{-finite-d} :: \text{lift} \quad (\text{finite})$

syntax (ASCII)

$\text{-finite-d} :: \text{lift} \quad (\text{finite})$

translations

$\text{-finite-d} \Rightarrow \text{CONST finite-d}$

definition *schop-d* :: ('a::world) formula \Rightarrow 'a formula
where *schop-d* F1 F2 \equiv LIFT((F1 \wedge finite);F2)

definition *sometimes-d* :: ('a::world) formula \Rightarrow 'a formula
where *sometimes-d* F \equiv LIFT(finite;F)

definition *di-d* :: ('a::world) formula \Rightarrow 'a formula
where *di-d* F \equiv LIFT(F;#True)

definition *da-d* :: ('a::world) formula \Rightarrow 'a formula
where *da-d* F \equiv LIFT(finite;(F;#True))

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

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

syntax

-*schop-d* :: [lift, lift] \Rightarrow lift ((- \frown -) [84,84] 83)
-*sometimes-d* :: lift \Rightarrow lift ((\diamond -) [88] 87)
-*di-d* :: lift \Rightarrow lift ((di -) [88] 87)
-*da-d* :: lift \Rightarrow lift ((da -) [88] 87)
-*next-d* :: lift \Rightarrow lift ((\bigcirc -) [88] 87)
-*prev-d* :: lift \Rightarrow lift ((prev -) [88] 87)

syntax (ASCII)

-*schop-d* :: [lift, lift] \Rightarrow lift ((- *schop* -) [84,84] 83)
-*sometimes-d* :: lift \Rightarrow lift ((<>-) [88] 87)
-*di-d* :: lift \Rightarrow lift ((di -) [88] 87)
-*da-d* :: lift \Rightarrow lift ((da -) [88] 87)
-*next-d* :: lift \Rightarrow lift ((next -) [88] 87)
-*prev-d* :: lift \Rightarrow lift ((prev -) [88] 87)

translations

-*schop-d* \rightleftharpoons CONST *schop-d*
-*sometimes-d* \rightleftharpoons CONST *sometimes-d*
-*di-d* \rightleftharpoons CONST *di-d*
-*da-d* \rightleftharpoons CONST *da-d*
-*next-d* \rightleftharpoons CONST *next-d*
-*prev-d* \rightleftharpoons CONST *prev-d*

definition *df-d* :: ('a::world) formula \Rightarrow 'a formula
where *df-d* F \equiv LIFT(F \frown #True)

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

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

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

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

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

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

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

syntax

-*df-d* :: lift \Rightarrow lift ((*df -*) [88] 87)
-*sda-d* :: lift \Rightarrow lift ((*sda -*) [88] 87)
-*always-d* :: lift \Rightarrow lift ((\square -) [88] 87)
-*bi-d* :: lift \Rightarrow lift ((*bi -*) [88] 87)
-*ba-d* :: lift \Rightarrow lift ((*ba -*) [88] 87)
-*wnext-d* :: lift \Rightarrow lift ((*wnext -*) [88] 87)
-*wprev-d* :: lift \Rightarrow lift ((*wprev -*) [88] 87)
-*more-d* :: lift ((*more*))

syntax (ASCII)

-*df-d* :: lift \Rightarrow lift ((*df -*) [88] 87)
-*sda-d* :: lift \Rightarrow lift ((*sda -*) [88] 87)
-*always-d* :: lift \Rightarrow lift ((\square -) [88] 87)
-*bi-d* :: lift \Rightarrow lift ((*bi -*) [88] 87)
-*ba-d* :: lift \Rightarrow lift ((*ba -*) [88] 87)
-*wnext-d* :: lift \Rightarrow lift ((*wnext -*) [88] 87)
-*wprev-d* :: lift \Rightarrow lift ((*wprev -*) [88] 87)
-*more-d* :: lift ((*more*))

translations

-*df-d* \rightleftharpoons *CONST df-d*
-*sda-d* \rightleftharpoons *CONST sda-d*
-*always-d* \rightleftharpoons *CONST always-d*
-*bi-d* \rightleftharpoons *CONST bi-d*
-*ba-d* \rightleftharpoons *CONST ba-d*

$\text{-wnext-d} \Rightarrow \text{CONST wnext-d}$
 $\text{-wprev-d} \Rightarrow \text{CONST wprev-d}$
 $\text{-more-d} \Rightarrow \text{CONST more-d}$

definition $\text{bf-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{bf-d } F \equiv \text{LIFT}(\neg(\text{df}(\neg F)))$

definition $\text{sba-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{sba-d } F \equiv \text{LIFT}(\neg(\text{sda}(\neg F)))$

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

definition $\text{fmore-d} :: ('a::\text{world}) \text{ formula}$
where $\text{fmore-d} \equiv \text{LIFT}(\text{more} \wedge \text{finite})$

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

syntax

$\text{-bf-d} \quad :: \text{lift} \Rightarrow \text{lift } ((\text{bf } -) [88] 87)$
 $\text{-sba-d} \quad :: \text{lift} \Rightarrow \text{lift } ((\text{sba } -) [88] 87)$
 $\text{-empty-d} \quad :: \text{lift} \quad ((\text{empty}))$
 $\text{-fmore-d} \quad :: \text{lift} \quad ((\text{fmore}))$
 $\text{-dm-d} \quad :: \text{lift} \Rightarrow \text{lift } ((\text{dm } -) [88] 87)$

syntax (ASCII)

$\text{-bf-d} \quad :: \text{lift} \Rightarrow \text{lift } ((\text{bf } -) [88] 87)$
 $\text{-sba-d} \quad :: \text{lift} \Rightarrow \text{lift } ((\text{sba } -) [88] 87)$
 $\text{-empty-d} \quad :: \text{lift} \quad ((\text{empty}))$
 $\text{-fmore-d} \quad :: \text{lift} \quad ((\text{fmore}))$
 $\text{-dm-d} \quad :: \text{lift} \Rightarrow \text{lift } ((\text{dm } -) [88] 87)$

translations

$\text{-bf-d} \quad \Rightarrow \text{CONST bf-d}$
 $\text{-sba-d} \quad \Rightarrow \text{CONST sba-d}$
 $\text{-empty-d} \Rightarrow \text{CONST empty-d}$
 $\text{-fmore-d} \Rightarrow \text{CONST fmore-d}$
 $\text{-dm-d} \quad \Rightarrow \text{CONST dm-d}$

definition $\text{bm-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{bm-d } F \equiv \text{LIFT}(\neg(\text{dm}(\neg F)))$

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

definition $\text{fin-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{fin-d } F \equiv \text{LIFT}(\Box(\text{empty} \longrightarrow F))$

definition *halt-d* :: ('a::world) formula \Rightarrow 'a formula
where *halt-d* F \equiv LIFT(\Box (empty = F))

definition *initonly-d* :: ('a::world) formula \Rightarrow 'a formula
where *initonly-d* F \equiv LIFT(bi(empty = F))

definition *keep-d* :: ('a::world) formula \Rightarrow 'a formula
where *keep-d* F \equiv LIFT(ba(skip \longrightarrow F))

definition *yields-d* :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a formula
where *yields-d* F1 F2 \equiv LIFT(\neg (F1; \neg F2)))

definition *syields-d* :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a formula
where *syields-d* F1 F2 \equiv LIFT(\neg (F1 \frown (\neg F2)))

definition *ifthenelse-d* :: ('a::world) formula \Rightarrow 'a formula \Rightarrow 'a formula \Rightarrow 'a formula
where *ifthenelse-d* F G H \equiv LIFT((F \wedge G) \vee (\neg F \wedge H))

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 \wedge finite);(power-d F n))

primrec *spower-d* :: ('a::world) formula \Rightarrow nat \Rightarrow 'a formula
where *spow-0* : (spower-d F 0) = LIFT(empty)
| *spow-Suc*: (spower-d F (Suc n)) = LIFT(F \frown (spower-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)
-yields-d :: [lift, lift] \Rightarrow lift ((- yields -) [88,88] 87)
-syields-d :: [lift, lift] \Rightarrow lift ((- syields -) [88,88] 87)
-ifthenelse-d :: [lift, lift, lift] \Rightarrow lift ((if; - then - else -) [88,88,88] 87)
-power-d :: [lift, nat] \Rightarrow lift ((power -) [88,88] 87)
-spower-d :: [lift, nat] \Rightarrow lift ((spower -) [88,88] 87)

syntax (ASCII)

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

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

translations

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

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 $\text{fpowerstar-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{fpowerstar-d } F \equiv \text{LIFT}(\exists k. \text{power } F k)$

definition $\text{spowerstar-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{spowerstar-d } F \equiv \text{LIFT}(\exists k. \text{spower } F k)$

definition $\text{omega-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{omega-d } F \equiv (\lambda s.$
 $\quad (\text{case } s \text{ of } (\text{Inl } s) \Rightarrow \text{False}$
 $\quad \quad | (\text{Inr } s) \Rightarrow$
 $\quad \quad (\exists (l::\text{infiniteindex}). \text{infinite-index-sequence } 0 \ l \wedge$
 $\quad \quad \quad (\forall i.$
 $\quad \quad \quad \quad ((\text{Inl } (\text{subinterval } s \ (l \ i) \ (l \ (\text{Suc } i)))) \models F)$
 $\quad \quad \quad)$
 $\quad \quad)$
 $\quad))$

syntax

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

syntax (ASCII)

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

translations

$-len-d \quad \Rightarrow \text{CONST } len-d$
 $-fpowerstar-d \quad \Rightarrow \text{CONST } fpowerstar-d$
 $-spowerstar-d \quad \Rightarrow \text{CONST } spowerstar-d$
 $-omega-d \quad \Rightarrow \text{CONST } omega-d$

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

where $powerstar-d F \equiv \text{LIFT}((\exists k. \text{power } F k); (\text{empty} \vee (F \wedge \text{inf})))$

syntax

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

syntax (ASCII)

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

translations

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

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

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

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

where $schopstar-d F \equiv \text{LIFT}(\text{spowerstar } (F \wedge \text{more}))$

syntax

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

syntax (ASCII)

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

translations

$-chopstar-d \quad \Rightarrow \text{CONST } chopstar-d$
 $-schopstar-d \quad \Rightarrow \text{CONST } schopstar-d$

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

where $\text{sfm-d } F \equiv \text{LIFT}(\neg (\text{fin } (\neg F)))$

definition $\text{ifthen-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{ifthen-d } F G \equiv \text{LIFT}(\text{if}_i F \text{ then } G \text{ else } \# \text{True})$

definition $\text{while-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{while-d } F G \equiv \text{LIFT}((F \wedge G)^* \wedge (\text{fin } (\neg F)))$

syntax

$\text{-ifthen-d} :: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{if}_i - \text{then} -) [88,88] 87)$
 $\text{-while-d} :: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{while} - \text{do} -) [88,88] 87)$
 $\text{-sfm-d} :: \text{lift} \Rightarrow \text{lift } ((\text{sfm} -) [88] 87)$

syntax (ASCII)

$\text{-ifthen-d} :: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{if}_i - \text{then} -) [88,88] 87)$
 $\text{-while-d} :: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{while} - \text{do} -) [88,88] 87)$
 $\text{-sfm-d} :: \text{lift} \Rightarrow \text{lift } ((\text{sfm} -) [88] 87)$

translations

$\text{-ifthen-d} \Rightarrow \text{CONST ifthen-d}$
 $\text{-while-d} \Rightarrow \text{CONST while-d}$
 $\text{-sfm-d} \Rightarrow \text{CONST sfm-d}$

definition $\text{swhile-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{swhile-d } F G \equiv \text{LIFT}(\text{schopstar}(F \wedge G) \wedge (\text{sfm } (\neg F)))$

definition $\text{repeat-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{repeat-d } F G \equiv \text{LIFT}(F; \text{while } (\neg G) \text{ do } F)$

syntax

$\text{-swhile-d} :: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{swhile} - \text{do} -) [88,88] 87)$
 $\text{-repeat-d} :: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{repeat} - \text{until} -) [88,88] 87)$

syntax (ASCII)

$\text{-swhile-d} :: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{swhile} - \text{do} -) [88,88] 87)$
 $\text{-repeat-d} :: [\text{lift}, \text{lift}] \Rightarrow \text{lift } ((\text{repeat} - \text{until} -) [88,88] 87)$

translations

$\text{-swhile-d} \Rightarrow \text{CONST swhile-d}$
 $\text{-repeat-d} \Rightarrow \text{CONST repeat-d}$

definition $\text{srepeat-d} :: ('a::\text{world}) \text{ formula} \Rightarrow 'a \text{ formula} \Rightarrow 'a \text{ formula}$
where $\text{srepeat-d } F G \equiv \text{LIFT}(F \frown \text{swhile } (\neg G) \text{ do } F)$

definition $\text{next-assign-d} :: ('a::\text{world}, 'b) \text{ stfun} \Rightarrow ('a, 'b) \text{ formfun} \Rightarrow 'a \text{ formula}$
where $\text{next-assign-d } v e \equiv \text{LIFT}(v\$ = e)$

definition $\text{prev-assign-d} :: ('a::\text{world}, 'b) \text{ stfun} \Rightarrow ('a, 'b) \text{ formfun} \Rightarrow 'a \text{ formula}$
where $\text{prev-assign-d } v e \equiv \text{LIFT}(\text{finite} \longrightarrow v! = e)$

definition *always-eq-d* :: ('a::world,'b) stfun \Rightarrow ('a,'b) formfun \Rightarrow 'a formula
where *always-eq-d* v e $\equiv \lambda s. s \models \Box(\$v = e)$

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

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

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

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

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

syntax

-srepeat-d :: [lift,lift] \Rightarrow lift ((srepeat - until -) [88,88] 87)
-next-assign-d :: [lift,lift] \Rightarrow lift ((- := -) [50,51] 50)
-prev-assign-d :: [lift,lift] \Rightarrow lift ((- =: -) [50,51] 50)
-always-eq-d :: [lift,lift] \Rightarrow lift ((- \approx -) [50,51] 50)
-temporal-assign-d :: [lift,lift] \Rightarrow lift ((- \leftarrow -) [50,51] 50)
-gets-d :: [lift,lift] \Rightarrow lift ((- gets -) [50,51] 50)
-stable-d :: lift \Rightarrow lift ((stable -) [51] 50)
-padded-d :: lift \Rightarrow lift ((padded -) [51] 50)
-padded-temp-assign-d :: [lift,lift] \Rightarrow lift ((- $<\sim$ -) [50,51] 50)

syntax (ASCII)

-srepeat-d :: [lift,lift] \Rightarrow lift ((srepeat - until -) [88,88] 87)
-next-assign-d :: [lift,lift] \Rightarrow lift ((- := -) [50,51] 50)
-prev-assign-d :: [lift,lift] \Rightarrow lift ((- =: -) [50,51] 50)
-always-eq-d :: [lift,lift] \Rightarrow lift ((- alweqv -) [50,51] 50)
-temporal-assign-d :: [lift,lift] \Rightarrow lift ((- <- -) [50,51] 50)
-gets-d :: [lift,lift] \Rightarrow lift ((- gets -) [50,51] 50)
-stable-d :: lift \Rightarrow lift ((stable -) [51] 50)
-padded-d :: lift \Rightarrow lift ((padded -) [51] 50)
-padded-temp-assign-d :: [lift,lift] \Rightarrow lift ((- <~ -) [50,51] 50)

translations

-srepeat-d \Rightarrow CONST srepeat-d
-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

$\text{-padded-temp-assign-d} \Rightarrow \text{CONST padded-temp-assign-d}$

5.4 Properties of Operators

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

lemma *skip-defs* :

$(w \models \text{skip}) = (\text{case } w \text{ of } (\text{Inl } w) \Rightarrow (\text{intlen } w = 1) \mid (\text{Inr } w) \Rightarrow \text{False})$

by (*simp add: skip-d-def*)

lemma *skip-defs-finite* :

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

by (*simp add: skip-d-def*)

lemma *skip-defs-infinite* :

$\neg ((\text{Inr } w) \models \text{skip})$

by (*simp add: skip-d-def*)

lemma *chop-defs* :

$(w \models F1 ; F2) =$

$(\text{case } w \text{ of } (\text{Inl } w) \Rightarrow$
 $(\exists n. n \geq 0 \wedge n \leq \text{intlen } w \wedge ((\text{Inl } (\text{prefix } n \ w)) \models F1) \wedge ((\text{Inl } (\text{suffix } n \ w)) \models F2))$
 $\mid (\text{Inr } w) \Rightarrow$
 $(\exists n. ((\text{Inl } (\text{iprefix } n \ w)) \models F1) \wedge ((\text{Inr } (\text{isuffix } n \ w)) \models F2))$
 $\vee ((\text{Inr } w) \models F1)$
 $)$
 $)$

by (*simp add: chop-d-def*)

lemma *chop-defs-finite*:

$((\text{Inl } w) \models F1;F2) =$
 $(\exists n. n \geq 0 \wedge n \leq \text{intlen } w \wedge$
 $(\text{Inl } (\text{prefix } n \ w) \models F1) \wedge (\text{Inl } (\text{suffix } n \ w) \models F2))$
 $)$

by (*simp add: chop-d-def*)

lemma *chop-defs-infinite*:

$((\text{Inr } w) \models F1;F2) =$
 $(\exists n. (\text{Inl } (\text{iprefix } n \ w) \models F1) \wedge (\text{Inr } (\text{isuffix } n \ w) \models F2))$
 $\vee (\text{Inr } w \models F1)$
 $)$

by (*simp add: chop-d-def*)

lemma *infinite-defs*:

$(w \models \text{inf}) = (\text{case } w \text{ of } (\text{Inr } w) \Rightarrow \text{True} \mid (\text{Inl } w) \Rightarrow \text{False})$

by (*simp add: infinite-d-def chop-d-def sum.case-eq-if*)

lemma *infinite-defs-1*:

$((\text{Inr } w) \models \text{inf})$

by (*simp add: infinite-d-def chop-d-def sum.case-eq-if*)

lemma *finite-defs* :

($w \models \text{finite}$) = (case w of ($\text{Inl } w \Rightarrow \text{True}$ | ($\text{Inr } w \Rightarrow \text{False}$))

by (*simp add: finite-d-def infinite-defs chop-d-def sum.case-eq-if*)

lemma *finite-defs-1* :

(($\text{Inl } w$) $\models \text{finite}$)

by (*simp add: finite-defs sum.case-eq-if*)

lemma *schop-defs* :

($w \models F1 \frown F2$) =

(case w of ($\text{Inl } w \Rightarrow$

($\exists n. n \geq 0 \wedge n \leq \text{intlen } w \wedge ((\text{Inl } (\text{prefix } n \ w)) \models F1) \wedge ((\text{Inl } (\text{suffix } n \ w)) \models F2)$)

| ($\text{Inr } w \Rightarrow$

($\exists n. ((\text{Inl } (\text{iprefix } n \ w)) \models F1) \wedge ((\text{Inr } (\text{isuffix } n \ w)) \models F2)$))

)

)

by (*simp add: schop-d-def chop-defs finite-defs sum.case-eq-if*)

lemma *schop-defs-finite* :

(($\text{Inl } w$) $\models F1 \frown F2$) =

($\exists n. n \geq 0 \wedge n \leq \text{intlen } w \wedge ((\text{Inl } (\text{prefix } n \ w)) \models F1) \wedge ((\text{Inl } (\text{suffix } n \ w)) \models F2)$)

by (*simp add: schop-defs*)

lemma *schop-defs-infinite* :

(($\text{Inr } w$) $\models F1 \frown F2$) =

($\exists n. ((\text{Inl } (\text{iprefix } n \ w)) \models F1) \wedge ((\text{Inr } (\text{isuffix } n \ w)) \models F2)$)

by (*simp add: schop-defs*)

lemma *sometimes-defs* :

($w \models \Diamond F$) =

(case w of ($\text{Inl } w \Rightarrow (\exists n. 0 \leq n \wedge n \leq \text{intlen } w \wedge ((\text{Inl } (\text{suffix } n \ w)) \models F))$

| ($\text{Inr } w \Rightarrow (\exists n. ((\text{Inr } (\text{isuffix } n \ w)) \models F))$

)

by (*simp add: sometimes-d-def finite-defs chop-defs sum.case-eq-if*)

lemma *always-defs* :

($w \models \Box F$) =

(case w of ($\text{Inl } w \Rightarrow (\forall n. 0 \leq n \wedge n \leq \text{intlen } w \longrightarrow ((\text{Inl } (\text{suffix } n \ w)) \models F))$

| ($\text{Inr } w \Rightarrow (\forall n. ((\text{Inr } (\text{isuffix } n \ w)) \models F))$

)

by (*simp add: always-d-def sometimes-defs sum.case-eq-if*)

lemma *di-defs* :

($w \models \text{di } F$) =

$(\text{case } w \text{ of } (Inl\ w) \Rightarrow (\exists\ n.\ 0 \leq n \wedge n \leq \text{intlen } w \wedge ((Inl\ (\text{prefix } n\ w)) \models F))$
 $\quad | (Inr\ w) \Rightarrow (\exists\ n.\ ((Inl\ (\text{iprefix } n\ w)) \models F)) \vee ((Inr\ w) \models F))$
 $)$
by (simp add: di-d-def chop-d-def sum.case-eq-if)

lemma df-defs :
 $(w \models df\ F) =$
 $(\text{case } w \text{ of } (Inl\ w) \Rightarrow (\exists\ n.\ 0 \leq n \wedge n \leq \text{intlen } w \wedge ((Inl\ (\text{prefix } n\ w)) \models F))$
 $\quad | (Inr\ w) \Rightarrow (\exists\ n.\ ((Inl\ (\text{iprefix } n\ w)) \models F))$
 $)$
by (simp add: df-d-def schop-defs sum.case-eq-if)

lemma bi-defs :
 $(w \models bi\ F) =$
 $(\text{case } w \text{ of } (Inl\ w) \Rightarrow (\forall\ n.\ 0 \leq n \wedge n \leq \text{intlen } w \longrightarrow ((Inl\ (\text{prefix } n\ w)) \models F))$
 $\quad | (Inr\ w) \Rightarrow (\forall\ n.\ ((Inl\ (\text{iprefix } n\ w)) \models F)) \wedge ((Inr\ w) \models F))$
 $)$
by (simp add: bi-d-def di-defs sum.case-eq-if)

lemma bf-defs :
 $(w \models bf\ F) =$
 $(\text{case } w \text{ of } (Inl\ w) \Rightarrow (\forall\ n.\ 0 \leq n \wedge n \leq \text{intlen } w \longrightarrow ((Inl\ (\text{prefix } n\ w)) \models F))$
 $\quad | (Inr\ w) \Rightarrow (\forall\ n.\ ((Inl\ (\text{iprefix } n\ w)) \models F))$
 $)$
by (simp add: bf-d-def df-defs sum.case-eq-if)

lemma da-defs :
 $(w \models da\ F) =$
 $(\text{case } w \text{ of } (Inl\ w) \Rightarrow (\exists\ n\ na.\ 0 \leq n \wedge na+n \leq \text{intlen } w \wedge ((Inl\ (\text{sub } n\ (na+n)\ w)) \models F))$
 $\quad | (Inr\ w) \Rightarrow (\exists\ n\ na.\ 0 \leq n \wedge ((Inl\ (\text{subinterval } w\ n\ (na+n))) \models F)$
 $\quad \vee ((Inr\ (\text{isuffix } n\ w)) \models F))$
 $)$
by (simp add: da-d-def chop-defs finite-d-def infinite-d-def iprefix-isuffix sum.case-eq-if,
 smt Groups.add-ac(2) Nat.le-diff-conv2 add-leD2 interval-sub-prefix-suffix-0
 interval-suffix-length-good zero-order(1))

lemma ba-defs :
 $(w \models ba\ F) =$
 $(\text{case } w \text{ of } (Inl\ w) \Rightarrow (\forall\ n\ na.\ 0 \leq n \wedge na+n \leq \text{intlen } w \longrightarrow ((Inl\ (\text{sub } n\ (na+n)\ w)) \models F))$
 $\quad | (Inr\ w) \Rightarrow (\forall\ n\ na.\ 0 \leq n \wedge ((Inl\ (\text{subinterval } w\ n\ (na+n))) \models F)$
 $\quad \wedge ((Inr\ (\text{isuffix } n\ w)) \models F))$
 $)$
by (simp add: ba-d-def da-defs sum.case-eq-if)

lemma sda-defs :
 $(w \models sda\ F) =$
 $(\text{case } w \text{ of } (Inl\ w) \Rightarrow (\exists\ n\ na.\ 0 \leq n \wedge na+n \leq \text{intlen } w \wedge ((Inl\ (\text{sub } n\ (na+n)\ w)) \models F))$
 $\quad | (Inr\ w) \Rightarrow (\exists\ n\ na.\ 0 \leq n \wedge ((Inl\ (\text{subinterval } w\ n\ (na+n))) \models F))$


```

)
by (simp add: sda-d-def schop-defs iprefix-isuffix sum.case-eq-if,
    smt Groups.add-ac(2) Nat.le-diff-conv2 add-leD2 interval-sub-prefix-suffix-0
    interval-suffix-length-good zero-order(1))
)

```

lemma *sba-defs* :

```

(w ⊨ sba F) =
  (case w of (Inl w) ⇒ (∀ n na. 0 ≤ n ∧ na + n ≤ intlen w ⟶ ((Inl (sub n (na + n) w)) ⊨ F))
    | (Inr w) ⇒ (∀ n na. 0 ≤ n ∧ ((Inl (subinterval w n (na + n))) ⊨ F))
  )
by (simp add: sba-d-def sda-defs sum.case-eq-if)

```

lemma *next-defs* :

```

(w ⊨ ○ F) =
  (case w of (Inl w) ⇒ (intlen w > 0 ∧ ((Inl (suffix 1 w)) ⊨ F))
    | (Inr w) ⇒ ((Inr (isuffix 1 w)) ⊨ F)
  )
using Suc-le-eq
by (simp add: next-d-def chop-defs skip-d-def iprefix-length sum.case-eq-if, force)

```

lemma *wnext-defs* :

```

(w ⊨ wnext F) =
  (case w of (Inl w) ⇒ (intlen w = 0 ∨ ((Inl (suffix 1 w)) ⊨ F))
    | (Inr w) ⇒ ((Inr (isuffix 1 w)) ⊨ F)
  )
by (simp add: wnext-d-def next-defs sum.case-eq-if)

```

lemma *prev-defs* :

```

(w ⊨ prev F) =
  (case w of (Inl w) ⇒ (intlen w > 0 ∧ ((Inl (prefix ((intlen w) - 1) w)) ⊨ F))
    | (Inr w) ⇒ (Inr w) ⊨ F
  )
by (simp add: prev-d-def chop-defs skip-d-def sum.case-eq-if,
    metis One-nat-def Suc-lel diff-diff-cancel diff-le-self interval-suffix-length-good
    le-zero-eq neq0-conv zero-neq-one)
)

```

lemma *wprev-defs* :

```

(w ⊨ wprev F) =
  (case w of (Inl w) ⇒ (intlen w = 0 ∨ ((Inl (prefix ((intlen w) - 1) w)) ⊨ F))
    | (Inr w) ⇒ (Inr w) ⊨ F
  )
by (simp add: wprev-d-def prev-defs sum.case-eq-if)

```

lemma *more-defs* :

```

(w ⊨ more) =
  (case w of (Inl w) ⇒ (intlen w > 0)
    | (Inr w) ⇒ True
  )

```

)
by (*simp add: more-d-def next-defs sum.case-eq-if*)

lemma *fmore-defs* :
 ($w \models \text{fmore}$) =
 (case w of ($\text{Inl } w \Rightarrow (\text{intlen } w > 0)$
 | ($\text{Inr } w \Rightarrow \text{False}$)
)
by (*simp add: fmore-d-def more-defs finite-defs sum.case-eq-if*)

lemma *empty-defs* :
 ($w \models \text{empty}$) =
 (case w of ($\text{Inl } w \Rightarrow (\text{intlen } w = 0)$
 | ($\text{Inr } w \Rightarrow \text{False}$)
)
by (*simp add: empty-d-def more-defs sum.case-eq-if*)

lemma *init-defs* :
 ($w \models \text{init } F$) =
 (case w of ($\text{Inl } w \Rightarrow ((\text{Inl } (\text{prefix } 0 \ w)) \models F)$
 | ($\text{Inr } w \Rightarrow ((\text{Inl } (\text{iprefix } 0 \ w)) \models F)$)
)
by (*simp add: init-d-def chop-defs empty-defs iprefix-length sum.case-eq-if, auto*)

lemma *init-defs-finite*:
 ($(\text{Inl } w) \models \text{init } F$) = ($(\text{Inl } (\text{prefix } 0 \ w)) \models F$)
by (*simp add: init-defs*)

lemma *init-defs-infinite*:
 ($(\text{Inr } w) \models \text{init } F$) = ($(\text{Inl } (\text{iprefix } 0 \ w)) \models F$)
by (*simp add: init-defs*)

lemma *initalt-defs* :
 ($w \models \text{bi}(\text{empty} \longrightarrow F)$) =
 (case w of ($\text{Inl } w \Rightarrow ((\text{Inl } (\text{prefix } 0 \ w)) \models F)$
 | ($\text{Inr } w \Rightarrow ((\text{Inl } (\text{iprefix } 0 \ w)) \models F)$)
)
by (*simp add: bi-defs empty-defs iprefix-length sum.case-eq-if*)

lemma *fin-defs* :
 ($w \models \text{fin } F$) =
 (case w of ($\text{Inl } w \Rightarrow ((\text{Inl } (\text{suffix } (\text{intlen } w) \ w)) \models F)$
 | ($\text{Inr } w \Rightarrow \text{True}$)
)
by (*simp add: fin-d-def empty-defs always-defs sum.case-eq-if*)

lemma *finalt-defs* :
 ($w \models \# \text{True}; (F \wedge \text{empty})$) =
 (case w of ($\text{Inl } w \Rightarrow ((\text{Inl } (\text{suffix } (\text{intlen } w) \ w)) \models F)$
 | ($\text{Inr } w \Rightarrow \text{True}$)

)
by (*simp add: chop-defs empty-defs sum.case-eq-if, fastforce*)

lemma *sfin-defs* :
 ($w \models \text{sfin } F$) =
 (case w of ($\text{Inl } w$) \Rightarrow ($\text{Inl } (\text{suffix } (\text{intlen } w) w)$) $\models F$)
 | ($\text{Inr } w$) \Rightarrow *False*
)

by (*simp add: sfin-d-def fin-defs sum.case-eq-if*)

lemma *halt-defs* :
 ($w \models \text{halt}(F)$) =
 (case w of ($\text{Inl } w$) \Rightarrow ($\forall n \leq \text{intlen } w. (\text{intlen } w = n) \Rightarrow (\text{Inl } (\text{suffix } n w)) \models F$)
 | ($\text{Inr } w$) \Rightarrow ($\forall n. \neg (\text{Inr } (\text{isuffix } n w)) \models F$)
)

by (*simp add: halt-d-def empty-defs always-defs sum.case-eq-if*)

lemma *initonly-defs* :
 ($w \models \text{initonly}(F)$) =
 (case w of ($\text{Inl } w$) \Rightarrow ($\forall n \leq \text{intlen } w. (n = 0) \Rightarrow (\text{Inl } (\text{prefix } n w)) \models F$)
 | ($\text{Inr } w$) \Rightarrow ($\forall n. (n = 0) \Rightarrow (\text{Inl } (\text{iprefix } n w)) \models F$) $\wedge \neg ((\text{Inr } w) \models F)$
)

by (*simp add: initonly-d-def bi-defs empty-defs iprefix-length sum.case-eq-if*)

lemma *ifthenelse-defs*:
 ($w \models \text{if } F \text{ then } G \text{ else } H$) =
 ((($w \models F$) \wedge ($w \models G$)) \vee (($\neg(w \models F)$) \wedge ($w \models H$)))
by (*simp add: ifthenelse-d-def*)

lemma *len-defs* :
 ($w \models \text{len } n$) =
 (case w of ($\text{Inl } w$) \Rightarrow ($\text{intlen } w = n$)
 | ($\text{Inr } w$) \Rightarrow *False*
)
by (*simp add: len-d-def power-d-def sum.case-eq-if,*
induct n arbitrary: w,
simp add: len-d-def empty-defs finite-defs sum.case-eq-if,
simp add: len-d-def chop-defs skip-defs finite-defs sum.case-eq-if,
fastforce)

lemma *currentval-defs* :
 ($s \models \$v$) =
 (case s of ($\text{Inl } s$) \Rightarrow ($v (\text{nth } s 0)$)
 | ($\text{Inr } s$) \Rightarrow ($v (s 0)$)
)

by (*simp add: current-val-d-def*)

lemma *nextval-defs* :

$(s \models v\$) =$
 $(\text{case } s \text{ of } (Inl\ s) \Rightarrow (\text{if } \text{intlen } s > 0 \text{ then } (v\ (\text{nth } s\ 1)) \text{ else } (\epsilon\ x. x=x))$
 $\quad | (Inr\ s) \Rightarrow (v\ (s\ 1)))$
 $)$
by (*simp add: next-val-d-def*)

lemma *finval-defs* :
 $(s \models !v) =$
 $(\text{case } s \text{ of } (Inl\ s) \Rightarrow (v\ (\text{nth } s\ (\text{intlen } s)))$
 $\quad | (Inr\ s) \Rightarrow (\epsilon\ x. x=x))$
 $)$
by (*simp add: fin-val-d-def*)

lemma *penultval-defs* :
 $(s \models v!) =$
 $(\text{case } s \text{ of } (Inl\ s) \Rightarrow (\text{if } \text{intlen } s > 0 \text{ then } (v\ (\text{nth } s\ ((\text{intlen } s) - 1))) \text{ else } (\epsilon\ x. x=x))$
 $\quad | (Inr\ s) \Rightarrow (\epsilon\ x. x=x))$
 $)$
by (*simp add: penult-val-d-def*)

lemma *next-assign-defs* :
 $(s \models v := e) =$
 $(\text{case } s \text{ of } (Inl\ s) \Rightarrow \text{if } 0 < \text{intlen } s \text{ then } v\ (\text{Interval.nth } s\ 1) \text{ else } (\epsilon\ x. x=x)$
 $\quad | Inr\ s \Rightarrow v\ (s\ 1))$
 $) =$
 $e\ s$

by (*auto simp: next-assign-d-def next-val-d-def*)

lemma *prev-assign-defs* :
 $(s \models v =: e) =$
 $(\text{case } s \text{ of } (Inl\ s) \Rightarrow$
 $\quad \text{if } 0 < \text{intlen } s \text{ then } (v\ (\text{Interval.nth } s\ ((\text{intlen } s) - 1)) = e\ (Inl\ s))$
 $\quad \text{else } ((\epsilon\ x. x=x) = e\ (Inl\ s))$
 $\quad | (Inr\ s) \Rightarrow \text{True})$
 $)$
by (*simp add: prev-assign-d-def penult-val-d-def finite-defs sum.case-eq-if*)

lemma *always-eqv-defs* :
 $(s \models v \approx e) =$
 $(\text{case } s \text{ of } (Inl\ s) \Rightarrow (\forall\ i \leq \text{intlen } s. v\ (\text{Interval.nth } s\ i) = e\ (Inl\ (\text{suffix } i\ s)))$
 $\quad | (Inr\ s) \Rightarrow (\forall\ i. v\ (s\ i) = e\ (Inr\ (\text{isuffix } i\ s))))$
 $)$
by (*simp add: always-eq-d-def always-defs current-val-d-def isuffix-def sum.case-eq-if*)

lemma *temporal-assign-defs* :
 $(s \models v \leftarrow e) =$
 $(\text{case } s \text{ of } (Inl\ s) \Rightarrow (v\ (\text{Interval.nth } s\ (\text{intlen } s))) = e\ (Inl\ s)$
 $\quad | (Inr\ s) \Rightarrow \text{True})$
 $)$

by (*simp add: temporal-assign-d-def fin-val-d-def finite-defs sum.case-eq-if*)

lemma *gets-defs* :

($s \models v \text{ gets } e$) =
 (case s of ($Inl\ s$) $\Rightarrow (\forall\ i < \text{intlen } s. v (\text{Interval.nth } s (\text{Suc } i)) = e (\text{Inl } (\text{sub } i (i+1) s))$))
 | ($Inr\ s$) $\Rightarrow (\forall\ i. v (s (\text{Suc } i)) = e (\text{Inl } (\text{subinterval } s\ i (i+1)))$))
)

using *Suc-lel Suc-le-lessD*

by (*simp add: finite-defs gets-d-def keep-d-def ba-defs skip-defs sub-def temporal-assign-d-def fin-val-d-def subinterval-length subinterval-nth sum.case-eq-if, auto*)

lemma *stable-defs-help*:

($\forall i < \text{intlen } s. v (\text{Interval.nth } s (\text{Suc } i)) = v (\text{Interval.nth } s\ i)$) =
 ($\forall i \leq \text{intlen } s. v (\text{Interval.nth } s\ i) = v (\text{Interval.nth } s\ 0)$)

proof

(*induct s*)

case (*St x*)

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

next

case (*Cons x1a s*)

then show ?case

by (*smt Suc-lessl interval-nth-Suc intlen.simps(2) le-SucE le-neq-implies-less le-simps(1) less-Suc-eq plus-1-eq-Suc zero-less-Suc*)

qed

lemma *stable-defs-infinite*:

($\forall i. v (s (\text{Suc } i)) = v (s\ i)$) = ($\forall i. v (s\ i) = v (s\ 0)$)

proof –

have 1: ($\forall i. v (s (\text{Suc } i)) = v (s\ i)$) \Longrightarrow ($\bigwedge j. v (s\ j) = v (s\ 0)$)

proof –

assume A1: ($\forall i. v (s (\text{Suc } i)) = v (s\ i)$)

show ($\bigwedge j. v (s\ j) = v (s\ 0)$)

proof –

fix j

show $v (s\ j) = v (s\ 0)$

using A1 **by** (*induct j, simp, simp*)

qed

qed

have 2: ($\forall i. v (s (\text{Suc } i)) = v (s\ i)$) \Longrightarrow ($\forall j. v (s\ j) = v (s\ 0)$)

using 1 **by** *blast*

have 3: ($\forall j. v (s\ j) = v (s\ 0)$) \Longrightarrow ($\bigwedge j. v (s (\text{Suc } j)) = v (s\ j)$)

proof –

assume A2: ($\forall j. v (s\ j) = v (s\ 0)$)

show ($\bigwedge j. v (s (\text{Suc } j)) = v (s\ j)$)

proof –

fix j

show $v (s (\text{Suc } j)) = v (s\ j)$

using A2 **by** (*induct j, auto, metis*)

qed

qed

have 4: $(\forall j. v(s\ j) = v(s\ 0)) \implies (\forall j. v(s\ (Suc\ j)) = v(s\ j))$
using 3 **by** *blast*
from 2 4 **show** ?thesis **by** *blast*
qed

lemma *stable-defs*:

$(s \models \text{stable } v) =$
 $(\text{case } s \text{ of } (Inl\ s) \Rightarrow (\forall i \leq \text{intlen } s. (v\ (\text{nth } s\ i)) = (v\ (\text{nth } s\ 0)))$
 $\quad | (Inr\ s) \Rightarrow (\forall i. (v\ (s\ i)) = (v\ (s\ 0)))$
 $)$
by (*simp add: stable-d-def gets-defs current-val-d-def sub-def stable-defs-help*
subinterval-def
upt-same stable-defs-infinite sum.case-eq-if)

lemma *padded-defs* :

$(s \models \text{padded } v) =$
 $(\text{case } s \text{ of } (Inl\ s) \Rightarrow ((\forall i < \text{intlen } s. (v\ (\text{nth } s\ i)) = (v\ (\text{nth } s\ 0))) \vee \text{intlen } s = 0)$
 $\quad | (Inr\ s) \Rightarrow ((\forall i. (v\ (s\ i)) = (v\ (s\ 0))))$
 $)$
by (*simp add: padded-d-def stable-defs chop-d-def skip-defs empty-defs interval-suffix-length*
sum.case-eq-if,
smt Suc-lel Suc-pred diff-diff-cancel interval-intlen-gr-zero le-neq-implies-less le-simps(1)
less-Suc-eq
 $)$

lemma *padded-temporal-assign-defs* :

$(s \models v <\sim e) =$
 $((s \models \text{padded } v) \wedge$
 $(\text{case } s \text{ of } (Inl\ s) \Rightarrow (v\ (\text{Interval.nth } s\ (\text{intlen } s))) = e\ (Inl\ s))$
 $\quad | (Inr\ s) \Rightarrow \text{True})$
 $)$
by (*simp add: padded-temp-assign-d-def padded-defs temporal-assign-defs, auto*)

lemma *linalw-finite*:

$a \leq b \wedge b \leq \text{intlen } w \wedge ((Inl\ (\text{suffix } a\ w)) \models \Box A) \longrightarrow ((Inl\ (\text{suffix } b\ w)) \models \Box A)$
by (*simp add: always-defs,*
smt add.assoc add.commute interval-suffix-length-good le-add-diff-inverse le-trans
ordered-cancel-comm-monoid-diff-class.le-diff-conv2)

lemma *linalw-infinite*:

$a \leq b \wedge ((Inr\ (\text{isuffix } a\ w)) \models \Box A) \longrightarrow ((Inr\ (\text{isuffix } b\ w)) \models \Box A)$
by (*simp add: always-defs,*
metis isuffix-isuffix le-add-diff-inverse)

5.5 Soundness Axioms

5.5.1 ChopAssoc

lemma *ChopAssocSemHelp*:

$(\exists i \text{ ia} . i \leq \text{intlen } \sigma \wedge \text{ia} \leq \text{intlen } \sigma - i \wedge (\text{Inl } (\text{prefix } i \sigma) \models f) \wedge$
 $(\text{Inl } (\text{prefix } \text{ia} (\text{suffix } i \sigma)) \models g) \wedge (\text{Inl } (\text{suffix } (\text{ia} + i) \sigma) \models h)) =$
 $(\exists j \text{ ja} . j \leq \text{intlen } \sigma \wedge \text{ja} \leq j \wedge (\text{Inl } (\text{prefix } \text{ja} (\text{prefix } j \sigma)) \models f) \wedge$
 $(\text{Inl } (\text{suffix } \text{ja} (\text{prefix } j \sigma)) \models g) \wedge (\text{Inl } (\text{suffix } j \sigma) \models h))$
by (smt Nat.le-diff-conv2 add-diff-cancel-left' interval-pref-pref-3 interval-suffix-prefix-swap
le-add1 le-add-diff-inverse2 le-trans)

lemma ChopAssocSemHelpFinite:

$((\text{Inl } \sigma) \models f ; (g ; h)) = ((\text{Inl } \sigma) \models (f;g);h)$
proof –
have $((\text{Inl } \sigma) \models f ; (g ; h)) =$
 $((\exists i \leq \text{intlen } \sigma . (\text{Inl } (\text{prefix } i \sigma) \models f) \wedge (\exists \text{ia} \leq \text{intlen } (\text{suffix } i \sigma) .$
 $(\text{Inl } (\text{prefix } \text{ia} (\text{suffix } i \sigma)) \models g) \wedge (\text{Inl } (\text{suffix } (\text{ia} + i) \sigma) \models h))))$
by (simp add: chop-defs)
also have ... =
 $(\exists i \text{ ia} . i \leq \text{intlen } \sigma \wedge \text{ia} \leq \text{intlen } \sigma - i \wedge ((\text{Inl } (\text{prefix } i \sigma)) \models f) \wedge$
 $(\text{Inl } (\text{prefix } \text{ia} (\text{suffix } i \sigma)) \models g) \wedge (\text{Inl } (\text{suffix } (\text{ia} + i) \sigma) \models h))$
by fastforce
also have ... =
 $(\exists j \text{ ja} . j \leq \text{intlen } \sigma \wedge \text{ja} \leq j \wedge (\text{Inl } (\text{prefix } \text{ja} (\text{prefix } j \sigma)) \models f) \wedge$
 $(\text{Inl } (\text{suffix } \text{ja} (\text{prefix } j \sigma)) \models g) \wedge (\text{Inl } (\text{suffix } j \sigma) \models h))$
using ChopAssocSemHelp[of σ f g h] **by** blast
also have ... =
 $(\exists i \leq \text{intlen } \sigma . (\exists \text{ia} \leq \text{intlen } (\text{prefix } i \sigma) . (\text{Inl } (\text{prefix } \text{ia} (\text{prefix } i \sigma)) \models f) \wedge$
 $(\text{Inl } (\text{suffix } \text{ia} (\text{prefix } i \sigma)) \models g)) \wedge (\text{Inl } (\text{suffix } i \sigma) \models h))$
by fastforce
also have ... =
 $(\text{Inl } \sigma \models (f;g);h)$ **by** (simp add: chop-defs)
finally show $(\text{Inl } \sigma \models f ; (g ; h)) = (\text{Inl } \sigma \models (f;g);h)$.
qed

lemma ChopAssocSemHelpInFinite:

$((\text{Inr } \sigma) \models f ; (g ; h)) = ((\text{Inr } \sigma) \models (f;g);h)$
proof –
have $((\text{Inr } \sigma) \models f ; (g ; h)) =$
 $((\exists n .$
 $f (\text{Inl } (\text{iprefix } n \sigma)) \wedge$
 $((\exists \text{na} . g (\text{Inl } (\text{iprefix } \text{na} (\text{isuffix } n \sigma))) \wedge h (\text{Inr } (\text{isuffix } \text{na} (\text{isuffix } n \sigma)))) \vee$
 $g (\text{Inr } (\text{isuffix } n \sigma)))) \vee$
 $f (\text{Inr } \sigma))$
by (metis chop-defs-infinite)
also have ... =
 $((\exists n \text{ na} .$
 $f (\text{Inl } (\text{iprefix } n \sigma)) \wedge$
 $((g (\text{Inl } (\text{iprefix } \text{na} (\text{isuffix } n \sigma))) \wedge h (\text{Inr } (\text{isuffix } \text{na} (\text{isuffix } n \sigma)))) \vee$
 $g (\text{Inr } (\text{isuffix } n \sigma)))) \vee$
 $f (\text{Inr } \sigma))$
by fastforce
also have ... =
 $((\exists n \text{ na} . \text{na} \leq \text{intlen } (\text{iprefix } n \sigma) \wedge$

$$\begin{aligned}
& (f \text{ (Inl (prefix na (iprefix n } \sigma))) \wedge g \text{ (Inl (suffix na (iprefix n } \sigma))) \wedge \\
& h \text{ (Inr (isuffix n } \sigma))) \vee \\
& (\exists n::\text{nat. } f \text{ (Inl (iprefix n } \sigma)) \wedge g \text{ (Inr (isuffix n } \sigma))) \vee f \text{ (Inr } \sigma))
\end{aligned}$$

by (*smt add.commute interval-iprefix-isuffix-swap interval-pref-ipref-3 iprefix-length isuffix-isuffix le-add1 le-add-diff-inverse2*)

also have ... =

$((\text{Inr } \sigma) \models (f;g);h)$

by (*metis chop-defs-finite chop-defs-infinite le-add2 le-add-same-cancel2*)

finally show $((\text{Inr } \sigma) \models f ; (g ; h)) = ((\text{Inr } \sigma) \models (f;g);h)$.

qed

lemma *ChopAssocSem*:

$(\sigma \models f ; (g ; h) = (f;g);h)$

using *ChopAssocSemHelpFinite ChopAssocSemHelpInFinite unl-lift2*

by (*smt old.sum.exhaust*)

5.5.2 OrChopImp

lemma *OrChopImpSem*:

$(\sigma \models (f \vee g);h \longrightarrow f;h \vee g;h)$

using *chop-defs*

by (*smt sum.case-eq-if unl-lift2*)

5.5.3 ChopOrImp

lemma *ChopOrImpSem*:

$(\sigma \models f;(g \vee h) \longrightarrow f;g \vee f;h)$

using *chop-defs*

by (*smt sum.case-eq-if unl-lift2*)

5.5.4 EmptyChop

lemma *EmptyChopSemFinite*:

$((\text{Inl } \sigma) \models \text{empty} ; f = f)$

by (*simp add: empty-defs chop-defs*) *auto*

lemma *EmptyChopSemInfinite*:

$((\text{Inr } \sigma) \models \text{empty} ; f = f)$

by (*simp add: chop-defs empty-defs iprefix-length isuffix-0*)

lemma *EmptyChopSem*:

$(\sigma \models \text{empty} ; f = f)$

using *EmptyChopSemFinite EmptyChopSemInfinite* **by** (*smt old.sum.exhaust*)

5.5.5 ChopEmpty

lemma *ChopEmptySemFinite*:

$((\text{Inl } \sigma) \models f;\text{empty} = f)$

by (*simp add: empty-defs chop-defs*) *auto*

lemma *ChopEmptySemInfinite*:
 $((\text{Inr } \sigma) \models f; \text{empty} = f)$
by (*simp add: chop-defs empty-defs*)

lemma *ChopEmptySem*:
 $(\sigma \models f; \text{empty} = f)$
using *ChopEmptySemFinite ChopEmptySemInfinite*
by (*smt old.sum.exhaust*)

5.5.6 StatImpBi

lemma *StatImpBiSem*:
 $(\sigma \models \text{init } f \longrightarrow \text{bi } (\text{init } f))$
by (*simp add: init-defs bi-defs sum.case-eq-if,metis conc-def conc-iprefix-isuffix interval-intlen-gr-zero interval-nth-zero-intfirst iprefix-0*)

5.5.7 NextImpNotNextNot

lemma *NextImpNotNextNotSem*:
 $(\sigma \models \bigcirc f \longrightarrow \neg (\bigcirc (\neg f)))$
by (*simp add: next-defs sum.case-eq-if*)

5.5.8 BiBoxChopImpChop

lemma *BiBoxChopImpChopSem*:
 $(\sigma \models \text{bi } (f \longrightarrow f1) \wedge \Box (g \longrightarrow g1) \longrightarrow f; g \longrightarrow f1; g1)$
by (*simp add: bi-defs always-defs chop-defs sum.case-eq-if, fastforce*)

5.5.9 BoxInduct

lemma *box-induct-help-1* :
 $((\text{Inl } \sigma) \models f) \wedge (\forall i. \text{Suc } 0 \leq \text{intlen } \sigma - i \longrightarrow$
 $i \leq \text{intlen } \sigma \longrightarrow (\text{Inl } (\text{suffix } i \sigma) \models f) \longrightarrow (\text{Inl } (\text{suffix } (\text{Suc } i) \sigma) \models f))$
 $\implies (\forall j. j \leq \text{intlen } \sigma \longrightarrow (\text{Inl } (\text{suffix } j \sigma) \models f))$
proof
fix *j*
show $((\text{Inl } \sigma) \models f) \wedge (\forall i. \text{Suc } 0 \leq \text{intlen } \sigma - i \longrightarrow$
 $i \leq \text{intlen } \sigma \longrightarrow (\text{Inl } (\text{suffix } i \sigma) \models f) \longrightarrow (\text{Inl } (\text{suffix } (\text{Suc } i) \sigma) \models f))$
 $\implies j \leq \text{intlen } \sigma \longrightarrow (\text{Inl } (\text{suffix } j \sigma) \models f)$
by
 $(\text{induct } j \text{ arbitrary: } \sigma, \text{ simp, metis Nat.le-diff-conv2 One-nat-def Suc-eq-plus1-left Suc-leD})$
qed

lemma *box-induct-help-infinite* :
 $((\text{Inr } \sigma) \models f) \wedge (\forall i.$
 $((\text{Inr } (\text{isuffix } i \sigma) \models f) \longrightarrow (\text{Inr } (\text{isuffix } (\text{Suc } i) \sigma) \models f)))$
 $\implies (\forall j. (\text{Inr } (\text{isuffix } j \sigma) \models f))$
proof
fix *j*
show $((\text{Inr } \sigma) \models f) \wedge (\forall i.$

$(\text{Inr } (\text{isuffix } i \ \sigma) \models f) \longrightarrow (\text{Inr } (\text{isuffix } (\text{Suc } i) \ \sigma) \models f))$
 $\implies (\text{Inr } (\text{isuffix } j \ \sigma) \models f)$
by (*induct j arbitrary: σ , simp add: isuffix-0, blast*)
qed

lemma *BoxInductSem*:

$(\sigma \models \Box (f \longrightarrow \text{wnext } f) \wedge f \longrightarrow \Box f)$
using *box-induct-help-1 box-induct-help-infinite*
by (*simp add: always-defs wnext-defs sum.case-eq-if,*
smt One-nat-def add commute box-induct-help-1 box-induct-help-infinite
cancel-comm-monoid-add-class.diff-cancel isuffix-isuffix not-one-le-zero plus-1-eq-Suc
sum.collapse(1) sum.collapse(2))

5.5.10 ChopStarEqv

lemma *ChopExist*:

$\vdash (\exists k. f;g \ k) = f;(\exists k. g \ k)$
by (*simp add: chop-defs Valid-def sum.case-eq-if, auto*)

lemma *SChopExist*:

$\vdash (\exists k. f \frown g \ k) = f \frown (\exists k. g \ k)$
by (*simp add: schop-defs Valid-def sum.case-eq-if, auto*)

lemma *ExistChop*:

$\vdash (\exists k. (g \ k);f) = (\exists k. g \ k);f$
by (*simp add: chop-defs Valid-def sum.case-eq-if, auto*)

lemma *ExistSChop*:

$\vdash (\exists k. (g \ k) \frown f) = (\exists k. g \ k) \frown f$
by (*simp add: schop-defs Valid-def sum.case-eq-if, auto*)

lemma *powersem1*:

$(\sigma \models (\exists k. \text{power } f \ k) = (\text{empty} \vee (\exists k. \text{power } f \ (\text{Suc } k))))$
by (*smt not0-implies-Suc pow-0 unl-Rex unl-lift2*)

lemma *spowersem1*:

$(\sigma \models (\exists k. \text{spower } f \ k) = (\text{empty} \vee (\exists k. \text{spower } f \ (\text{Suc } k))))$
by (*smt not0-implies-Suc spow-0 unl-Rex unl-lift2*)

lemma *powersem*:

$\vdash (\exists k. \text{power } f \ k) = (\text{empty} \vee (f \wedge \text{finite});(\exists k. (\text{power } f \ k)))$
proof –
have 1: $\vdash (\exists k. \text{power } f \ k) = (\text{empty} \vee (\exists k. \text{power } f \ (\text{Suc } k)))$
using *powersem1 by blast*
have 2: $\vdash (\exists k. \text{power } f \ (\text{Suc } k)) = (\exists k. (f \wedge \text{finite});\text{power } f \ k)$
by *simp*
have 3: $\vdash (\exists k. (f \wedge \text{finite});(\text{power } f \ k)) = (f \wedge \text{finite});(\exists k. (\text{power } f \ k))$
using *ChopExist by blast*
from 1 2 3 **show** ?thesis **by** *fastforce*

qed

lemma *spowersem*:

$\vdash (\exists k. \text{spower } f \ k) = (\text{empty} \vee f \wedge (\exists k. (\text{spower } f \ k)))$

proof –

have 1: $\vdash (\exists k. \text{spower } f \ k) = (\text{empty} \vee (\exists k. \text{spower } f \ (\text{Suc } k)))$

using *spowersem1* **by** *blast*

have 2: $\vdash (\exists k. \text{spower } f \ (\text{Suc } k)) = (\exists k. f \wedge \text{spower } f \ k)$

by *simp*

have 3: $\vdash (\exists k. f \wedge (\text{spower } f \ k)) = f \wedge (\exists k. (\text{spower } f \ k))$

using *SChopExist* **by** *blast*

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

qed

lemma *PowerstarEqvSemhelp1*:

$\vdash \text{empty};(\text{empty} \vee (f \wedge \text{inf})) = (\text{empty} \vee (f \wedge \text{inf}))$

using *EmptyChopSem* **by** *blast*

lemma *PowerstarEqvSemhelp2*:

$\vdash (f \wedge \text{inf}) = (f \wedge \text{inf});g$

by (*simp add: Valid-def infinite-defs chop-defs sum.case-eq-if*)

lemma *PowerstarEqvSemhelp3*:

$\vdash ((f \wedge \text{inf});g \vee (f \wedge \text{finite});g) = (f ;g)$

by (*simp add: Valid-def finite-defs infinite-defs chop-defs sum.case-eq-if,auto*)

lemma *PowerstarEqvSem*:

$(\sigma \models (\text{powerstar } f) = (\text{empty} \vee f;(\text{powerstar } f)))$

proof –

have 1: $(\sigma \models (\text{powerstar } f)) =$
 $(\sigma \models (\exists k. \text{power } f \ k);(\text{empty} \vee f \wedge \text{inf}))$

by (*simp add: powerstar-d-def*)

have 2: $(\sigma \models (\exists k. \text{power } f \ k);(\text{empty} \vee f \wedge \text{inf})) =$
 $(\sigma \models (\text{empty} \vee (f \wedge \text{finite});(\exists k. (\text{power } f \ k)));(\text{empty} \vee f \wedge \text{inf}))$

using *powersem* **by** (*metis inteq-reflection*)

have 3: $(\sigma \models (\text{empty} \vee (f \wedge \text{finite});(\exists k. (\text{power } f \ k)));(\text{empty} \vee f \wedge \text{inf})) =$
 $(\sigma \models \text{empty};(\text{empty} \vee (f \wedge \text{inf})) \vee$

$((f \wedge \text{finite});(\exists k. (\text{power } f \ k)));(\text{empty} \vee f \wedge \text{inf}))$

by (*smt chop-defs intensional-rews(3) sum.case-eq-if*)

have 4: $(\sigma \models \text{empty};(\text{empty} \vee (f \wedge \text{inf})) \vee$
 $((f \wedge \text{finite});(\exists k. (\text{power } f \ k)));(\text{empty} \vee f \wedge \text{inf})) =$
 $(\sigma \models (\text{empty} \vee (f \wedge \text{inf}) \vee ((f \wedge \text{finite});(\exists k. (\text{power } f \ k)));(\text{empty} \vee f \wedge \text{inf})))$

using *PowerstarEqvSemhelp1*

by (*metis (mono-tags, lifting) inteq-reflection unl-lift2*)

have 5: $(\sigma \models (\text{empty} \vee (f \wedge \text{inf}) \vee ((f \wedge \text{finite});(\exists k. (\text{power } f \ k)));(\text{empty} \vee f \wedge \text{inf}))) =$
 $(\sigma \models (\text{empty} \vee (f \wedge \text{inf});((\exists k. (\text{power } f \ k));(\text{empty} \vee f \wedge \text{inf})) \vee$
 $((f \wedge \text{finite});(\exists k. (\text{power } f \ k)));(\text{empty} \vee f \wedge \text{inf})))$

using *PowerstarEqvSemhelp2*

by (*metis (mono-tags, lifting) inteq-reflection*)

have 6: $(\sigma \models (\text{empty} \vee (f \wedge \text{inf});((\exists k. (\text{power } f \ k));(\text{empty} \vee f \wedge \text{inf}))) \vee$

$$((f \wedge \text{finite}); (\exists k. (\text{power } f \ k))); (\text{empty} \vee f \wedge \text{inf})) = \\ (\sigma \models (\text{empty} \vee (f \wedge \text{inf}); (\exists k. (\text{power } f \ k)); (\text{empty} \vee f \wedge \text{inf}))) \vee \\ (f \wedge \text{finite}); (\exists k. (\text{power } f \ k)); (\text{empty} \vee f \wedge \text{inf})))$$

by (smt ChopAssocSemHelpFinite ChopAssocSemHelpInFinite sum.collapse(1) sum.collapse(2) unl-lift2)
have 7 : ($\sigma \models (\text{empty} \vee (f \wedge \text{inf}); (\exists k. (\text{power } f \ k)); (\text{empty} \vee f \wedge \text{inf})) \vee$
 $(f \wedge \text{finite}); (\exists k. (\text{power } f \ k)); (\text{empty} \vee f \wedge \text{inf}))) =$
 $(\sigma \models (\text{empty} \vee f; (\exists k. (\text{power } f \ k)); (\text{empty} \vee f \wedge \text{inf})))$)
using PowerstarEqvSemhelp3 **by** fastforce
have 8: ($\sigma \models (\text{empty} \vee f; (\exists k. (\text{power } f \ k)); (\text{empty} \vee f \wedge \text{inf}))) =$
 $(\sigma \models (\text{empty} \vee f; (\text{powerstar } f)))$)
by (simp add: powerstar-d-def)
from 1 2 3 4 5 6 7 8 **show** ?thesis **by** fastforce
qed

lemma FPowerstarEqvSem:

$(\sigma \models (\text{fpowerstar } f) = (\text{empty} \vee (f \wedge \text{finite}); (\text{fpowerstar } f)))$
proof –
have 1: ($\sigma \models (\text{fpowerstar } f) =$
 $(\sigma \models (\exists k. \text{power } f \ k))$)
by (simp add: fpowerstar-d-def)
have 2: ($\sigma \models (\exists k. \text{power } f \ k) =$
 $(\sigma \models (\text{empty} \vee (f \wedge \text{finite}); (\exists k. (\text{power } f \ k))))$)
using powersem **by** (metis inteq-reflection)
from 1 2 **show** ?thesis **by** (simp add: fpowerstar-d-def)
qed

lemma SPowerstarEqvSem:

$(\sigma \models (\text{spowerstar } f) = (\text{empty} \vee f \frown (\text{spowerstar } f)))$
proof –
have 1: ($\sigma \models (\text{spowerstar } f) =$
 $(\sigma \models (\exists k. \text{spower } f \ k))$)
by (simp add: spowerstar-d-def)
have 2: ($\sigma \models (\exists k. \text{spower } f \ k) =$
 $(\sigma \models (\text{empty} \vee f \frown (\exists k. (\text{spower } f \ k))))$)
using spowersem **by** (metis inteq-reflection)
from 1 2 **show** ?thesis **by** (simp add: spowerstar-d-def)
qed

lemma powerchopsem:

$\vdash (\exists k. \text{power } (f \wedge \text{more}) \ k) =$
 $(\text{empty} \vee ((f \wedge \text{more}) \wedge \text{finite}); (\exists k. (\text{power } (f \wedge \text{more}) \ k)))$
 $)$
using powersem **by** auto

lemma spowerchopsem:

$\vdash (\exists k. \text{spower } (f \wedge \text{more}) \ k) =$
 $(\text{empty} \vee (f \wedge \text{more}) \frown (\exists k. (\text{spower } (f \wedge \text{more}) \ k)))$
 $)$

using spowersem by auto

lemma ChopstarEqvSem:

$(\sigma \models f^* = (\text{empty} \vee (f \wedge \text{more}); f^*))$

by (metis PowerstarEqvSem chopstar-d-def)

lemma SChopstarEqvSem:

$(\sigma \models (\text{schopstar } f) = (\text{empty} \vee (f \wedge \text{more}) \frown (\text{schopstar } f)))$

by (metis SPowerstarEqvSem schopstar-d-def)

5.5.11 OmegaUnroll

lemma omega-unroll-chain:

$(\exists I. \text{infinite-index-sequence } 0 \ I \wedge (\forall i. f \ (Inl \ (\text{subinterval } \sigma \ (I \ i) \ (I \ (\text{Suc } i))))))$
 $=$
 $(\exists n.$
 $\quad f \ (Inl \ (\text{iprefix } n \ \sigma)) \wedge$
 $\quad 0 < n \wedge$
 $\quad (\exists I.$
 $\quad \quad \text{infinite-index-sequence } 0 \ I \wedge$
 $\quad \quad (\forall i. f \ (Inl \ (\text{subinterval } (\text{isuffix } n \ \sigma) \ (I \ i) \ (I \ (\text{Suc } i))))))$
 $\quad)$
 $)$

proof —

have $(\exists I. \text{infinite-index-sequence } 0 \ I \wedge$
 $\quad (\forall i. f \ (Inl \ (\text{subinterval } \sigma \ (I \ i) \ (I \ (\text{Suc } i)))))) =$
 $(\exists I. \text{infinite-index-sequence } 0 \ I \wedge I = \text{conc } \langle (I \ 0) \rangle \ (\lambda x. (I \ (x+1))) \wedge$
 $\quad (\forall i. f \ (Inl \ (\text{subinterval } \sigma \ (I \ i) \ (I \ (\text{Suc } i))))))$

using iidx-1 by blast

also have ... =
 $(\exists I. \text{infinite-index-sequence } 0 \ (\text{conc } \langle (I \ 0) \rangle \ (\lambda x. (I \ (x+1)))) \wedge$
 $\quad I = \text{conc } \langle (I \ 0) \rangle \ (\lambda x. (I \ (x+1))) \wedge$
 $\quad (\forall i. f \ (Inl \ (\text{subinterval } \sigma \ (\text{conc } \langle (I \ 0) \rangle \ (\lambda x. (I \ (x+1)))) \ i)$
 $\quad \quad (\text{conc } \langle (I \ 0) \rangle \ (\lambda x. (I \ (x+1))) \ (\text{Suc } i))))$
 $)$
 $)$

by force

also have ... =
 $(\exists I. (I \ 0) = 0 \wedge (I \ 0) < (I \ 1) \wedge$
 $\quad \text{infinite-index-sequence } (I \ 1) \ (\lambda x. (I \ (x+1))) \wedge$
 $\quad I = \text{conc } \langle (I \ 0) \rangle \ (\lambda x. (I \ (x+1))) \wedge$
 $\quad (\forall i. f \ (Inl \ (\text{subinterval } \sigma \ (\text{conc } \langle (I \ 0) \rangle \ (\lambda x. (I \ (x+1)))) \ i)$
 $\quad \quad (\text{conc } \langle (I \ 0) \rangle \ (\lambda x. (I \ (x+1))) \ (\text{Suc } i))))$
 $)$
 $)$

using iidx-2 by force

also have ... =
 $(\exists I. (I \ 0) = 0 \wedge (I \ 0) < (I \ 1) \wedge$

$$\begin{aligned}
& \text{infinite-index-sequence } (I\ 1) (\lambda x. (I\ (x+1))) \wedge \\
& I = \text{conc } \langle (I\ 0) \rangle (\lambda x. (I\ (x+1))) \wedge \\
& f\ (Inl\ (\text{subinterval } \sigma\ (I\ 0)\ (I\ 1))) \wedge \\
& (\forall i. f\ (Inl\ (\text{subinterval } \sigma\ ((\lambda x. (I\ (x+1)))\ i)\ ((\lambda x. (I\ (x+1)))\ (Suc\ i)))))
\end{aligned}$$

)

by (smt Suc-diff-Suc Suc-eq-plus1 add-cancel-left-right add-diff-cancel-left' le-add1
le-neq-implies-less plus-1-eq-Suc)

also have ... =

$$\begin{aligned}
& (\exists I\ ls. ls = (\lambda x. (I\ (x+1))) \wedge (I\ 0) = 0 \wedge (I\ 0) < (I\ 1) \wedge \\
& \text{infinite-index-sequence } (I\ 1)\ ls \wedge \\
& I = \text{conc } \langle (I\ 0) \rangle\ ls \wedge \\
& f\ (Inl\ (\text{subinterval } \sigma\ (I\ 0)\ (I\ 1))) \wedge \\
& (\forall i. f\ (Inl\ (\text{subinterval } \sigma\ (ls\ i)\ (ls\ (Suc\ i)))))
\end{aligned}$$

)

by force

also have ... =

$$\begin{aligned}
& (\exists I\ ls. ls = (\lambda x. (I\ (x+1))) \wedge (I\ 0) = 0 \wedge (I\ 0) < (I\ 1) \wedge \\
& \text{infinite-index-sequence } (I\ 1)\ ls \wedge \\
& I = \text{conc } \langle (I\ 0) \rangle\ ls \wedge \\
& f\ (Inl\ (\text{iprefix } (I\ 1)\ \sigma)) \wedge \\
& (\forall i. f\ (Inl\ (\text{subinterval } \sigma\ (ls\ i)\ (ls\ (Suc\ i)))))
\end{aligned}$$

)

by (metis iprefix-def)

also have ... =

$$\begin{aligned}
& (\exists I\ ls\ n. n = (ls\ 0) \wedge ls = (\lambda x. (I\ (x+1))) \wedge 0 < n \wedge \\
& \text{infinite-index-sequence } n\ ls \wedge I = \text{conc } \langle 0 \rangle\ ls \wedge \\
& f\ (Inl\ (\text{iprefix } n\ \sigma)) \wedge \\
& (\forall i. f\ (Inl\ (\text{subinterval } \sigma\ (ls\ i)\ (ls\ (Suc\ i)))))
\end{aligned}$$

)

by (metis (no-types, lifting) One-nat-def conc-empty-suc conc-empty-zero)

also have ... =

$$\begin{aligned}
& (\exists ls\ n. n = (ls\ 0) \wedge 0 < n \wedge \\
& \text{infinite-index-sequence } n\ ls \wedge \\
& f\ (Inl\ (\text{iprefix } n\ \sigma)) \wedge \\
& (\forall i. f\ (Inl\ (\text{subinterval } \sigma\ (ls\ i)\ (ls\ (Suc\ i)))))
\end{aligned}$$

)

using iidix-0 by (rule, metis (no-types, lifting) Suc-eq-plus1 conc-empty-suc)

also have ... =

$$\begin{aligned}
& (\exists ls\ n\ lsk. n = (ls\ 0) \wedge 0 < n \wedge lsk = (\text{shiftn } n) \circ ls \wedge \\
& \text{infinite-index-sequence } n\ ls \wedge \\
& f\ (Inl\ (\text{iprefix } n\ \sigma)) \wedge \\
& (\forall i. f\ (Inl\ (\text{subinterval } \sigma\ (ls\ i)\ (ls\ (Suc\ i)))))
\end{aligned}$$

)

by blast

also have ... =

$(\exists \text{ } l s \text{ } n \text{ } l s k. \text{ } n = (l s \text{ } 0) \wedge 0 < n \wedge l s k = (\text{shiftm } n) \circ l s \wedge$
 $\text{infinite-index-sequence } n \text{ } l s \wedge \text{infinite-index-sequence } 0 \text{ } l s k \wedge$
 $f \text{ } (Inl \text{ } (iprefix \text{ } n \text{ } \sigma)) \wedge$
 $(\forall i. f \text{ } (Inl \text{ } (subinterval \text{ } \sigma \text{ } (l s \text{ } i) \text{ } (l s \text{ } (Suc \text{ } i))))))$
 $)$
using *iidx-5* **by** *auto*
also have ... =
 $(\exists \text{ } l s \text{ } n \text{ } l s k. \text{ } n = (l s \text{ } 0) \wedge 0 < n \wedge l s = (\text{shift } n) \circ l s k \wedge$
 $\text{infinite-index-sequence } n \text{ } l s \wedge \text{infinite-index-sequence } 0 \text{ } l s k \wedge$
 $f \text{ } (Inl \text{ } (iprefix \text{ } n \text{ } \sigma)) \wedge$
 $(\forall i. f \text{ } (Inl \text{ } (subinterval \text{ } \sigma \text{ } (l s \text{ } i) \text{ } (l s \text{ } (Suc \text{ } i))))))$
 $)$
using *iidx-6* **by** *blast*
also have ... =
 $(\exists \text{ } l s \text{ } n \text{ } l s k. \text{ } n = (((\text{shift } n) \circ l s k) \text{ } 0) \wedge 0 < n \wedge l s = (\text{shift } n) \circ l s k \wedge$
 $\text{infinite-index-sequence } n \text{ } l s \wedge \text{infinite-index-sequence } 0 \text{ } l s k \wedge$
 $f \text{ } (Inl \text{ } (iprefix \text{ } n \text{ } \sigma)) \wedge$
 $(\forall i. f \text{ } (Inl \text{ } (subinterval \text{ } \sigma \text{ } (((\text{shift } n) \circ l s k) \text{ } i)$
 $(((\text{shift } n) \circ l s k) \text{ } (Suc \text{ } i)))))$
 $)$
 $)$
by *metis*
also have ... =
 $(\exists \text{ } l s \text{ } n \text{ } l s k. \text{ } 0 = (l s k \text{ } 0) \wedge 0 < n \wedge l s = (\text{shift } n) \circ l s k \wedge$
 $\text{infinite-index-sequence } n \text{ } l s \wedge \text{infinite-index-sequence } 0 \text{ } l s k \wedge$
 $f \text{ } (Inl \text{ } (iprefix \text{ } n \text{ } \sigma)) \wedge$
 $(\forall i. f \text{ } (Inl \text{ } (subinterval \text{ } \sigma \text{ } ((l s k \text{ } i)+n) ((l s k \text{ } (Suc \text{ } i))+n))))$
 $)$
using *shift-def* **by** *auto*
also have ... =
 $(\exists \text{ } l s \text{ } n \text{ } l s k. \text{ } 0 = (l s k \text{ } 0) \wedge 0 < n \wedge l s = (\text{shift } n) \circ l s k \wedge$
 $\text{infinite-index-sequence } 0 \text{ } l s k \wedge$
 $f \text{ } (Inl \text{ } (iprefix \text{ } n \text{ } \sigma)) \wedge$
 $(\forall i. f \text{ } (Inl \text{ } (subinterval \text{ } \sigma \text{ } ((l s k \text{ } i)+n) ((l s k \text{ } (Suc \text{ } i))+n))))$
 $)$
using *iidx-8* **by** *blast*
also have ... =
 $(\exists \text{ } n \text{ } l s k. \text{ } 0 = (l s k \text{ } 0) \wedge 0 < n \wedge$
 $\text{infinite-index-sequence } 0 \text{ } l s k \wedge$
 $f \text{ } (Inl \text{ } (iprefix \text{ } n \text{ } \sigma)) \wedge$
 $(\forall i. f \text{ } (Inl \text{ } (subinterval \text{ } \sigma \text{ } ((l s k \text{ } i)+n) ((l s k \text{ } (Suc \text{ } i))+n))))$
 $)$
by *blast*
also have ... =
 $(\exists \text{ } n \text{ } l s k. \text{ } 0 = (l s k \text{ } 0) \wedge 0 < n \wedge$
 $\text{infinite-index-sequence } 0 \text{ } l s k \wedge$
 $f \text{ } (Inl \text{ } (iprefix \text{ } n \text{ } \sigma)) \wedge$
 $(\forall i. f \text{ } (Inl \text{ } (subinterval \text{ } (isuffix \text{ } n \text{ } \sigma) \text{ } ((l s k \text{ } i)) ((l s k \text{ } (Suc \text{ } i))))))$
 $)$

```

using subinterval-sub-isuffix-iidx by (metis calculation calculation )
also have ... =
  (∃ n. f (Inl (iprefix n σ)) ∧ 0 < n ∧
    (∃ l. infinite-index-sequence 0 l ∧
      (∀ i. f (Inl (subinterval (isuffix n σ) (l i) (l (Suc i))))))
  )
)
using infinite-index-sequence-def by auto
finally show (∃ l. infinite-index-sequence 0 l ∧
  (∀ i. f (Inl (subinterval σ (l i) (l (Suc i))))))
)
=
  (∃ n. f (Inl (iprefix n σ)) ∧ 0 < n ∧
    (∃ l. infinite-index-sequence 0 l ∧
      (∀ i. f (Inl (subinterval (isuffix n σ) (l i) (l (Suc i))))))
  )
) .

```

qed

lemma omega-unroll-sem:

((Inr σ) ⊨ (f ∧ fmore);(omega f) = (omega f))

using omega-unroll-chain **by** (simp add: fmore-defs chop-defs omega-d-def iprefix-length, metis)

lemma OmegaUnrollSem:

σ ⊨ (omega f) = (f ∧ fmore);(omega f)

using omega-unroll-sem

by (cases σ, simp add: omega-d-def chop-defs-finite, fastforce)

5.5.12 OmegaInduct

lemma OmegaInductSem-help:

```

(σ ⊨ inf ∧ g ∧ □(g → (f ∧ fmore);g)) =
  ( case σ of (Inl σ) ⇒ False
    | (Inr σ) ⇒
      g (Inr σ) ∧
      (∀ n::nat. g (Inr (isuffix n σ)) →
        (∃ na::nat. f (Inl (subinterval σ n (na+n))) ∧
          (0::nat) < na ∧ g (Inr (isuffix (n + na) σ))))
  )

```

by (simp add: infinite-defs always-defs chop-defs fmore-defs isuffix-isuffix sum.case-eq-if, metis iprefix-isuffix iprefix-length)

primrec cpoint :: ('a::world) formula ⇒ 'a formula ⇒ nat ⇒ 'a interval ⇒ nat

where cpoint f g 0 σ = 0

| cpoint f g (Suc n) σ =

(ε x. (∃ m. (Inl (subinterval σ (cpoint f g n σ) (m+(cpoint f g n σ))) ⊨ f) ∧ m>0 ∧ (Inr(isuffix (m+(cpoint f g n σ)) σ) ⊨ g) ∧

$$x = m + (\text{cpoint } f \ g \ n \ \sigma)$$

lemma *cpoint-order-0a*:

$na \leq x \implies \text{intlen } (\text{suffix } na \ (\text{iprefix } x \ \sigma)) = x - na$

by (*simp add: iprefix-length*)

lemma *cpoint-expand-0*:

$(\text{cpoint } f \ g \ 0 \ \sigma) = 0$

by *simp*

lemma *cpoint-expand-1*:

$(\text{cpoint } f \ g \ 1 \ \sigma) =$
 $(\text{SOME } x. (\exists m. f \ (\text{Inl } (\text{subinterval } \sigma \ 0 \ (m))))$
 $\quad \wedge m > 0 \wedge g \ (\text{Inr } (\text{isuffix } (m) \ \sigma))$
 $\quad \wedge x = m))$

by (*simp add: fmore-defs subinterval-length*)

lemma *cpoint-expand-n*:

$(\text{cpoint } f \ g \ (\text{Suc } n) \ \sigma) =$
 $(\text{SOME } x. (\exists m. f \ (\text{Inl } (\text{subinterval } \sigma \ (\text{cpoint } f \ g \ n \ \sigma) \ (m + (\text{cpoint } f \ g \ n \ \sigma)))))$
 $\quad \wedge m > 0 \wedge g \ (\text{Inr } (\text{isuffix } (m + (\text{cpoint } f \ g \ n \ \sigma)) \ \sigma))$
 $\quad \wedge x = m + (\text{cpoint } f \ g \ n \ \sigma))$
 $)$

by (*simp add: fmore-defs subinterval-length*)

lemma *cpoint-0*:

assumes $g \ (\text{Inr } \sigma) \wedge$
 $(\forall k. g \ (\text{Inr } (\text{isuffix } k \ \sigma)) \longrightarrow$
 $\quad (\exists m. f \ (\text{Inl } (\text{subinterval } \sigma \ k \ (m + k)))) \wedge$
 $\quad 0 < m \wedge g \ (\text{Inr } (\text{isuffix } (m + k) \ \sigma)))$

shows $g \ (\text{Inr } (\text{isuffix } (\text{cpoint } f \ g \ i \ \sigma) \ \sigma))$

proof

(*induct i*)

case 0

then show ?case **by** (*simp add: assms isuffix-0*)

next

case (*Suc i*)

then show ?case

proof –

have 1: $g \ (\text{Inr } (\text{isuffix } (\text{cpoint } f \ g \ i \ \sigma) \ \sigma))$

by (*simp add: Suc.hyps*)

have 2: $g \ (\text{Inr } (\text{isuffix } (\text{cpoint } f \ g \ i \ \sigma) \ \sigma)) \longrightarrow$
 $(\exists m. f \ (\text{Inl } (\text{subinterval } \sigma \ (\text{cpoint } f \ g \ i \ \sigma) \ (m + (\text{cpoint } f \ g \ i \ \sigma))))) \wedge$
 $0 < m \wedge g \ (\text{Inr } (\text{isuffix } (m + (\text{cpoint } f \ g \ i \ \sigma)) \ \sigma))$

using *assms* **by** *blast*

have 3: $(\exists m. f \ (\text{Inl } (\text{subinterval } \sigma \ (\text{cpoint } f \ g \ i \ \sigma) \ (m + (\text{cpoint } f \ g \ i \ \sigma))))) \wedge$
 $0 < m \wedge g \ (\text{Inr } (\text{isuffix } (m + (\text{cpoint } f \ g \ i \ \sigma)) \ \sigma))$

```

using 1 2 by auto
have 4: (cpoint f g (Suc i) σ) =
  (SOME x. (∃ m. f (Inl (subinterval σ (cpoint f g i σ) (m+(cpoint f g i σ))))
    ∧ m>0 ∧ g (Inr(isuffix (m+(cpoint f g i σ)) σ))
    ∧ x=m+(cpoint f g i σ)))
by simp
have 5: g (Inr(isuffix ((cpoint f g (Suc i) σ) σ))
using 3 4 somef-ex by (smt iprefix-def iprefix-isuffix)
from 5 show ?thesis by auto
qed
qed

```

lemma cpoint-1:

```

assumes g (Inr σ) ∧
  (∀ k. g (Inr (isuffix k σ)) →
    (∃ m. f (Inl (subinterval σ k (m+k)))) ∧
    0 < m ∧ g (Inr (isuffix (m+k) σ))))
shows ( g (Inr(isuffix (cpoint f g i σ) σ))
  ⇒ g (Inr(isuffix (cpoint f g (Suc i) σ) σ)))

```

proof —

```

have 1: g (Inr σ) ∧
  (∀ k. g (Inr (isuffix k σ)) →
    (∃ m. f (Inl (subinterval σ k (m+k)))) ∧
    0 < m ∧ g (Inr (isuffix (m+k) σ)))
  using assms by blast
have 2: ( g (Inr(isuffix (cpoint f g i σ) σ))
  ⇒ g (Inr(isuffix (cpoint f g (Suc i) σ) σ)))

```

proof

```

(induct i)
case 0
then show ?case
proof —
  have 01: g (Inr(isuffix (cpoint f g 0 σ) σ)) =
    g (Inr(isuffix 0 σ))
    by auto
  have 02: g (Inr(isuffix 0 σ)) = g (Inr σ)
    by (simp add: isuffix-0)
  have 03: (∃ m. f (Inl (subinterval σ 0 (m)))) ∧
    0 < m ∧ g (Inr (isuffix (m) σ))
  using 02 1 by fastforce
  have 04: (cpoint f g 1 σ) =
    (SOME x. (∃ m. f (Inl (subinterval σ 0 (m) ))
      ∧ m>0 ∧ g (Inr (isuffix (m) σ))
      ∧ x=m)
    )
  using cpoint-expand-1 by blast

```

```

have 05:  $g \text{ (Inr (isuffix (cpoint } f \text{ } g \text{ } 1 \text{ } \sigma) \text{ } \sigma))}$ 
using 03 04 somel-ex by (metis (mono-tags, lifting))
from 01 05 show ?thesis by auto
qed
next
case (Suc i)
then show ?case
proof -
have n1:  $g \text{ (Inr (isuffix (cpoint } f \text{ } g \text{ } i \text{ } \sigma) \text{ } \sigma))} \implies g \text{ (Inr (isuffix (cpoint } f \text{ } g \text{ (Suc } i) \text{ } \sigma) \text{ } \sigma))}$ 
using Suc.hyps by blast
have n2:  $g \text{ (Inr (isuffix (cpoint } f \text{ } g \text{ (Suc } i) \text{ } \sigma) \text{ } \sigma))}$ 
using Suc.premis by blast
have n3:  $(\exists m. f \text{ (Inl (subinterval } \sigma \text{ (cpoint } f \text{ } g \text{ (Suc } i) \text{ } \sigma) (m+(cpoint } f \text{ } g \text{ (Suc } i) \text{ } \sigma)))) \wedge$ 
 $0 < m \wedge g \text{ (Inr (isuffix (m+(cpoint } f \text{ } g \text{ (Suc } i) \text{ } \sigma) \text{ } \sigma))}$ 
using assms n2 by auto
have n4:  $(cpoint \text{ } f \text{ } g \text{ (Suc (Suc } i)) \text{ } \sigma) =$ 
 $(\text{SOME } x. (\exists m. f \text{ (Inl (subinterval } \sigma \text{ (cpoint } f \text{ } g \text{ (Suc } i) \text{ } \sigma) (m+(cpoint } f \text{ } g \text{ (Suc } i) \text{ } \sigma)))) \wedge$ 
 $m > 0 \wedge g \text{ (Inr (isuffix (m+(cpoint } f \text{ } g \text{ (Suc } i) \text{ } \sigma) \text{ } \sigma))$ 
 $\wedge x = m + (cpoint \text{ } f \text{ } g \text{ (Suc } i) \text{ } \sigma))$ 
)
using cpoint-expand-n by blast
have n5:  $g \text{ (Inr (isuffix (cpoint } f \text{ } g \text{ (Suc (Suc } i)) \text{ } \sigma) \text{ } \sigma))}$ 
using n3 n4 somel-ex by (smt iprefix-def iprefix-isuffix)
from n5 show ?thesis by auto
qed
qed
from 2 show ?thesis using assms cpoint-0 by blast
qed

```

lemma *cpoint-2*:

```

assumes  $g \text{ (Inr } \sigma) \wedge$ 
 $(\forall k. g \text{ (Inr (isuffix } k \text{ } \sigma)) \longrightarrow$ 
 $(\exists m. f \text{ (Inl (subinterval } \sigma \text{ } k \text{ (m+k)))} \wedge$ 
 $0 < m \wedge g \text{ (Inr (isuffix (m+k) } \sigma))))$ 

```

shows $f \text{ (Inl (subinterval } \sigma \text{ (cpoint } f \text{ } g \text{ } i \text{ } \sigma) \text{ (cpoint } f \text{ } g \text{ (Suc } i) \text{ } \sigma))}$

proof

(*induct* *i*)

case 0

then show ?case

proof -

have 1: $g \text{ (Inr (isuffix } 0 \text{ } \sigma))$

using assms cpoint-0 cpoint-expand-0 by (simp add: isuffix-0)

have 2: $(\exists m. f \text{ (Inl (subinterval } \sigma \text{ (cpoint } f \text{ } g \text{ } 0 \text{ } \sigma) (m+(cpoint } f \text{ } g \text{ } 0 \text{ } \sigma)))) \wedge$
 $0 < m \wedge g \text{ (Inr (isuffix (m+(cpoint } f \text{ } g \text{ } 0 \text{ } \sigma) \text{ } \sigma))}$

using assms 1 by auto

have 3: $(cpoint \text{ } f \text{ } g \text{ } 1 \text{ } \sigma) =$

$(\text{SOME } x. (\exists m. f \text{ (Inl (subinterval } \sigma \text{ (cpoint } f \text{ } g \text{ } 0 \text{ } \sigma) (m+(cpoint } f \text{ } g \text{ } 0 \text{ } \sigma)) \text{)})$

$\wedge m > 0 \wedge g (\text{Inr} (\text{isuffix} (m + (\text{cpoint } f \ g \ 0 \ \sigma)) \ \sigma))$
 $\wedge x = m + (\text{cpoint } f \ g \ 0 \ \sigma)$
)

by simp
have 4: $f (\text{Inl} (\text{subinterval } \sigma (\text{cpoint } f \ g \ 0 \ \sigma) ((\text{cpoint } f \ g \ 1 \ \sigma))))$
using 2 3 some-ex by smt
from 4 show ?thesis by auto
qed
next
case (Suc i)
then show ?case
proof —
have n1: $g (\text{Inr} (\text{isuffix} (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma) \ \sigma))$
using assms cpoint-0 by blast
have n2: $(\exists m. f (\text{Inl} (\text{subinterval } \sigma (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma) (m + (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma)))) \wedge$
 $0 < m \wedge g (\text{Inr} (\text{isuffix} (m + (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma)) \ \sigma))$
using assms n1 by auto
have n3: $(\text{cpoint } f \ g \ (\text{Suc } (\text{Suc } i)) \ \sigma) =$
 $(\text{SOME } x. (\exists m. f (\text{Inl} (\text{subinterval } \sigma (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma) (m + (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma)))) \wedge$
 $m > 0 \wedge g (\text{Inr} (\text{isuffix} (m + (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma)) \ \sigma))$
 $\wedge x = m + (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma))$
)
using cpoint-expand-n by blast
have n4: $f (\text{Inl} (\text{subinterval } \sigma (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma) ((\text{cpoint } f \ g \ (\text{Suc } (\text{Suc } i)) \ \sigma))))$
using n2 n3 some-ex by (smt iprefix-def iprefix-isuffix)
from n4 show ?thesis by auto
qed
qed

lemma cpoint-3a:

$m > 0 \wedge x = m + (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma) \implies (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma) < x$
by auto

lemma cpoint-3:

assumes $g (\text{Inr } \sigma) \wedge$
 $(\forall k. g (\text{Inr} (\text{isuffix } k \ \sigma)) \longrightarrow$
 $(\exists m. f (\text{Inl} (\text{subinterval } \sigma \ k \ (m+k)))) \wedge$
 $0 < m \wedge g (\text{Inr} (\text{isuffix} (m+k) \ \sigma)))$

shows $(\text{cpoint } f \ g \ i \ \sigma) < (\text{cpoint } f \ g \ (\text{Suc } i) \ \sigma)$

proof

(induct i)

case 0

then show ?case

proof —

have 1: $g (\text{Inr} (\text{isuffix } 0 \ \sigma))$

using assms cpoint-0 cpoint-expand-0 by (simp add: isuffix-0)

```

have 2: (∃ m. f (Inl (subinterval σ (cpoint f g 0 σ) (m+(cpoint f g 0 σ)))) ∧
        0 < m ∧ g (Inr (isuffix (m+(cpoint f g 0 σ)) σ)))
  using assms 1 by auto
have 3: (cpoint f g 1 σ) =
  (SOME x. (∃ m. f (Inl (subinterval σ (cpoint f g 0 σ) (m+(cpoint f g 0 σ)) ))
    ∧ m > 0 ∧ g (Inr (isuffix (m+(cpoint f g 0 σ)) σ))
    ∧ x = m+(cpoint f g 0 σ))
  )

  by simp
have 4: (cpoint f g 0 σ) < (cpoint f g 1 σ)
  using 2 3 some-ex by (smt add.right-neutral cpoint-expand-0 less-numeral-extra(3) neq0-conv)
from 4 show ?thesis by auto
qed
next
case (Suc i)
then show ?case
proof -
  have n1: g (Inr (isuffix (cpoint f g (Suc i) σ) σ))
    using assms cpoint-0 by blast
  have n2: (∃ m. f (Inl (subinterval σ (cpoint f g (Suc i) σ) (m+(cpoint f g (Suc i) σ)))) ∧
    0 < m ∧ g (Inr (isuffix (m+(cpoint f g (Suc i) σ)) σ)))
    using assms n1 by auto
  have n3: (cpoint f g (Suc (Suc i)) σ) =
    (SOME x. (∃ m. f (Inl (subinterval σ (cpoint f g (Suc i) σ) (m+(cpoint f g (Suc i) σ))))
      ∧ m > 0 ∧ g (Inr (isuffix (m+(cpoint f g (Suc i) σ)) σ))
      ∧ x = m+(cpoint f g (Suc i) σ))
    )
    using cpoint-expand-n by blast
  have n4: (∃ m. f (Inl (subinterval σ (cpoint f g (Suc i) σ) (m+(cpoint f g (Suc i) σ))))
    ∧ m > 0 ∧ g (Inr (isuffix (m+(cpoint f g (Suc i) σ)) σ))
    ∧ (cpoint f g (Suc (Suc i)) σ) = m+(cpoint f g (Suc i) σ))
    using n2 n3 some-ex by smt
  have n5: (cpoint f g (Suc i) σ) < (cpoint f g (Suc (Suc i)) σ)
    using n4 using cpoint-3a by blast
  from n5 show ?thesis by auto
qed
qed

```

lemma *OmegaInductSem*:

(w ⊨ (inf ∧ g ∧ □(g → (f ∧ fmore);g)) → omega f)

proof (cases w)

case (Inl a)

then show ?thesis

by (metis (no-types, lifting) finite-d-def finite-defs-1 int-simps(21) inteq-reflection unl-con unl-lift2)

next

case (Inr b)

then show ?thesis

proof -

```

have 1: ((Inr b)  $\models$  (inf  $\wedge$  g  $\wedge$   $\Box$ (g  $\longrightarrow$  (f  $\wedge$  fmore);g))) =
  (g (Inr b)  $\wedge$ 
    ( $\forall$  k. g (Inr (isuffix k b))  $\longrightarrow$ 
      ( $\exists$  m. f (Inl (subinterval b k (m+k)))  $\wedge$ 
        0 < m  $\wedge$  g (Inr (isuffix (m+k) b))))))
by (smt Groups.add-ac(2) OmegaInductSem-help sum.case(2))
have 2: (g (Inr b)  $\wedge$ 
  ( $\forall$  k. g (Inr (isuffix k b))  $\longrightarrow$ 
    ( $\exists$  m. f (Inl (subinterval b k (m+k)))  $\wedge$ 
      0 < m  $\wedge$  g (Inr (isuffix (m+k) b))))))  $\longrightarrow$ 
  infinite-index-sequence 0 ( $\lambda i$ . (cpoint f g i b ))
using 1 infinite-index-sequence-def cpoint-3 by (metis cpoint-expand-0)
have 3: (g (Inr b)  $\wedge$ 
  ( $\forall$  k. g (Inr (isuffix k b))  $\longrightarrow$ 
    ( $\exists$  m. f (Inl (subinterval b k (m+k)))  $\wedge$ 
      0 < m  $\wedge$  g (Inr (isuffix (m+k) b))))))  $\longrightarrow$ 
    ( $\forall$  i. f (Inl (subinterval b (( $\lambda i$ . (cpoint f g i b )) i)
      (( $\lambda i$ . (cpoint f g i b )) (Suc i))))))

using 1 cpoint-2 by (metis )
have 4: (g (Inr b)  $\wedge$ 
  ( $\forall$  k. g (Inr (isuffix k b))  $\longrightarrow$ 
    ( $\exists$  m. f (Inl (subinterval b k (m+k)))  $\wedge$ 
      0 < m  $\wedge$  g (Inr (isuffix (m+k) b))))))  $\longrightarrow$ 
    infinite-index-sequence 0 ( $\lambda i$ . (cpoint f g i b ))  $\wedge$ 
    ( $\forall$  i. f (Inl (subinterval b (( $\lambda i$ . (cpoint f g i b )) i)
      (( $\lambda i$ . (cpoint f g i b )) (Suc i))))))

using 2 3 by auto
have 5: (g (Inr b)  $\wedge$ 
  ( $\forall$  k. g (Inr (isuffix k b))  $\longrightarrow$ 
    ( $\exists$  m. f (Inl (subinterval b k (m+k)))  $\wedge$ 
      0 < m  $\wedge$  g (Inr (isuffix (m+k) b))))))  $\longrightarrow$ 
    ( $\exists$  (Is::infiniteindex). infinite-index-sequence 0 Is  $\wedge$ 
      Is = ( $\lambda i$ . (cpoint f g i b )))

using 4 by auto
have 6: (g (Inr b)  $\wedge$ 
  ( $\forall$  k. g (Inr (isuffix k b))  $\longrightarrow$ 
    ( $\exists$  m. f (Inl (subinterval b k (m+k)))  $\wedge$ 
      0 < m  $\wedge$  g (Inr (isuffix (m+k) b))))))  $\longrightarrow$  ((Inr b)  $\models$  omega f)
using 5 omega-d-def by (smt 3 old.sum.simps(6))
have 7: ((Inr b)  $\models$  (inf  $\wedge$  g  $\wedge$   $\Box$ (g  $\longrightarrow$  (f  $\wedge$  fmore);g))  $\longrightarrow$  (omega f) )
using 6 1 by auto
from 7 show ?thesis using Inr by blast
qed
qed

```

5.6 Quantification over State (Flexible) Variables

Quantification in Infinite ITL is done similarly as in Finite ITL.

typed *decl state*

instance *state* :: *world* ..

type-synonym *'a statefun* = (*state*,*'a*) *stfun*
type-synonym *statepred* = *bool statefun*
type-synonym *'a tempfun* = (*state*,*'a*) *formfun*
type-synonym *temporal* = *state formula*

5.7 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

6 Finite ITL: Axioms and Rules

theory *ITL*

imports

Semantics

begin

The Finite ITL axiom and proof rules are introduced (taken from [4]). The soundness of the rules and axioms are checked using the lemmas of *Semantics.thy*.

6.1 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: always-defs*)

lemma *BiGen*:
assumes $\vdash f$
shows $\vdash bi\ f$
using *assms* **by** (*auto simp: bi-defs*)

6.2 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 *StatImpBi* :
 $\vdash init\ f \longrightarrow bi\ (init\ f)$
using *StatImpBiSem Valid-def* **by** *blast*

lemma *NextImpNotNextNot* :
 $\vdash \bigcirc f \longrightarrow \neg (\bigcirc (\neg f))$
using *NextImpNotNextNotSem Valid-def* **by** *blast*

lemma *BiBoxChopImpChop* :
 $\vdash bi\ (f \longrightarrow f1) \wedge \Box(g \longrightarrow g1) \longrightarrow f;g \longrightarrow f1;g1$
using *BiBoxChopImpChopSem Valid-def* **by** *blast*

lemma *BoxInduct* :
 $\vdash \Box (f \longrightarrow wnext\ f) \wedge f \longrightarrow \Box f$
using *BoxInductSem Valid-def* **by** *blast*

lemma *ChopstarEqv* :
 $\vdash f^* = (empty \vee (f \wedge more); f^*)$

using ChopstarEqvSem Valid-def by blast

6.3 Additional Lemmas

The following is again from [3, 2] but adapted for our need.

lemma *int-eq-true*: $\vdash P \implies \vdash P = \# \text{True}$
by *auto*

lemma *int-eq*: $\vdash X = Y \implies X = Y$
by (*auto simp: inteq-reflection*)

lemma *int-iff1*:
assumes $\vdash F \longrightarrow G$ and $\vdash G \longrightarrow F$
shows $\vdash F = G$
using *assms* by *force*

lemma *int-iffD1*: assumes *h*: $\vdash F = G$ shows $\vdash F \longrightarrow G$
using *h* by *auto*

lemma *int-iffD2*: assumes *h*: $\vdash F = G$ shows $\vdash G \longrightarrow F$
using *h* by *auto*

lemma *lift-imp-trans*:
assumes $\vdash A \longrightarrow B$ and $\vdash B \longrightarrow C$
shows $\vdash A \longrightarrow C$
using *assms* by *force*

lemma *lift-imp-neg*: assumes $\vdash A \longrightarrow B$ shows $\vdash \neg B \longrightarrow \neg A$
using *assms* by *auto*

lemma *lift-and-com*: $\vdash (A \wedge B) = (B \wedge A)$
by *auto*

6.4 Quantification

lemma *EEExI* :
 $\vdash F y \longrightarrow (\exists \exists x. F x)$
by (*simp add: exist-state-d-def Valid-def, auto*)

lemma *EExE*:
 $\llbracket \bigwedge x. \vdash F x \longrightarrow G \rrbracket \implies \vdash (\exists \exists x. F x) \longrightarrow G$
by (*metis (mono-tags, lifting) Valid-def exist-state-d-def unl-lift2*)

lemma *EEExVal*:
 $(w \models (\exists \exists x. F x)) =$
 $(\exists x (val :: 'a \text{ interval}). ((val = (map x w) \wedge (w \models F x))))$
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*

6.5 Lemmas about *current-val*

lemma *current-const*: $\vdash \$(\#c) = \#c$
by (*auto simp: current-val-d-def*)

lemma *current-fun1*: $\vdash \$(f\langle x \rangle) = f \langle \$x \rangle$
by (*auto simp: current-val-d-def*)

lemma *current-fun2*: $\vdash \$(f\langle x, y \rangle) = f \langle \$x, \$y \rangle$
by (*auto simp: current-val-d-def*)

lemma *current-fun3*: $\vdash \$(f\langle x, y, z \rangle) = f \langle \$x, \$y, \$z \rangle$
by (*auto simp: current-val-d-def*)

lemma *current-forall*: $\vdash \$(\forall x. P x) = (\forall x. \$(P x))$
by (*auto simp: current-val-d-def*)

lemma *current-exists*: $\vdash \$(\exists x. P x) = (\exists x. \$(P x))$
by (*auto simp: current-val-d-def*)

lemma *current-exists1*: $\vdash \$(\exists! x. P x) = (\exists! x. \$(P x))$
by (*auto simp: current-val-d-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]*

6.6 Lemmas about *next-val*

lemma *next-const*: $\vdash \text{more} \longrightarrow (\#c)\$ = \#c$
by (*auto simp: next-val-d-def more-defs*)

lemma *next-fun1*: $\vdash \text{more} \longrightarrow f\langle x \rangle\$ = f \langle x\$ \rangle$
by (*auto simp: next-val-d-def more-defs*)

lemma *next-fun2*: $\vdash \text{more} \longrightarrow f\langle x, y \rangle\$ = f \langle x\$, y\$ \rangle$
by (*auto simp: next-val-d-def more-defs*)

lemma *next-fun3*: $\vdash \text{more} \longrightarrow f\langle x, y, z \rangle\$ = f \langle x\$, y\$, z\$ \rangle$
by (*auto simp: next-val-d-def more-defs*)

lemma *next-forall*: $\vdash \text{more} \longrightarrow (\forall x. P\ x)\$ = (\forall x. (P\ x)\$)$
by (*auto simp: next-val-d-def*)

lemma *next-exists*: $\vdash \text{more} \longrightarrow (\exists x. P\ x)\$ = (\exists x. (P\ x)\$)$
by (*auto simp: next-val-d-def*)

lemma *next-exists1*: $\vdash \text{more} \longrightarrow (\exists! x. P\ x)\$ = (\exists! x. (P\ x)\$)$
by (*auto simp: next-val-d-def more-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]*

6.7 Lemmas about *fin-val*

lemma *fin-const*: $\vdash !(\#c) = \#c$
by (*auto simp: fin-val-d-def*)

lemma *fin-fun1*: $\vdash !(f\langle x \rangle) = f\ \langle!x\rangle$
by (*auto simp: fin-val-d-def*)

lemma *fin-fun2*: $\vdash !(f\langle x, y \rangle) = f\ \langle!x, !y\rangle$
by (*auto simp: fin-val-d-def*)

lemma *fin-fun3*: $\vdash !(f\langle x, y, z \rangle) = f\ \langle!x, !y, !z\rangle$
by (*auto simp: fin-val-d-def*)

lemma *fin-forall*: $\vdash !(\forall x. P\ x) = (\forall x. !(P\ x))$
by (*auto simp: fin-val-d-def*)

lemma *fin-exists*: $\vdash !(\exists x. P\ x) = (\exists x. !(P\ x))$
by (*auto simp: fin-val-d-def*)

lemma *fin-exists1*: $\vdash !(\exists! x. P\ x) = (\exists! x. !(P\ x))$
by (*auto simp: fin-val-d-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]*

6.8 Lemmas about *penult-val*

lemma *penult-const*: $\vdash \text{more} \longrightarrow (\#c)! = \#c$
by (*auto simp: penult-val-d-def more-defs*)

lemma *penult-fun1*: $\vdash \text{more} \longrightarrow f\langle x \rangle! = f\langle x! \rangle$

```

by (auto simp: penult-val-d-def more-defs)

lemma penult-fun2:  $\vdash \text{more} \longrightarrow f \langle x, y \rangle! = f \langle x!, y! \rangle$ 
by (auto simp: penult-val-d-def more-defs)

lemma penult-fun3:  $\vdash \text{more} \longrightarrow f \langle x, y, z \rangle! = f \langle x!, y!, z! \rangle$ 
by (auto simp: penult-val-d-def more-defs)

lemma penult-forall:  $\vdash \text{more} \longrightarrow (\forall x. P x)! = (\forall x. (P x)!)$ 
by (auto simp: penult-val-d-def)

lemma penult-exists:  $\vdash \text{more} \longrightarrow (\exists x. P x)! = (\exists x. (P x)!)$ 
by (auto simp: penult-val-d-def)

lemma penult-exists1:  $\vdash \text{more} \longrightarrow (\exists! x. P x)! = (\exists! x. (P x)!)$ 
by (auto simp: penult-val-d-def more-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]

```

6.9 Basic temporal variables properties

```

lemma empty-imp-fin-equiv-curr:
 $\vdash \text{empty} \longrightarrow !v = \$v$ 
by (simp add: Valid-def current-val-d-def empty-defs finval-defs)

lemma skip-imp-fin-equiv-next:
 $\vdash \text{skip} \longrightarrow !v = v\$$ 
by (simp add: Valid-def skip-defs next-val-d-def finval-defs)

lemma skip-imp-penult-equiv-curr:
 $\vdash \text{skip} \longrightarrow v! = \$v$ 
by (simp add: Valid-def skip-defs penultval-defs current-val-d-def)

```

end

7 Infinite ITL: Axioms and Rules

```

theory InfiniteITL
imports
  InfiniteSemantics
begin

```

The Infinite ITL axiom and proof rules are introduced (taken from [7]). The soundness of the rules and

axioms are checked using the lemmas of InfiniteSemantics.thy.

7.1 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 (*smt always-defs intD intl sum.case-eq-if*)

lemma *BiGen*:
 assumes $\vdash f$
 shows $\vdash bi\ f$
using *assms*
by (*smt bi-defs intD intl sum.case-eq-if*)

7.2 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 *StatImpBi* :
 $\vdash init\ f \longrightarrow bi\ (init\ f)$
using *StatImpBiSem Valid-def* **by** *blast*

lemma *NextImpNotNextNot* :
 $\vdash \bigcirc f \longrightarrow \neg (\bigcirc (\neg f))$

using *NextImpNotNextNotSem Valid-def* **by** *blast*

lemma *BiBoxChopImpChop* :

$\vdash bi (f \longrightarrow f1) \wedge \Box(g \longrightarrow g1) \longrightarrow f;g \longrightarrow f1;g1$

using *BiBoxChopImpChopSem Valid-def* **by** *blast*

lemma *BoxInduct* :

$\vdash \Box (f \longrightarrow wnext f) \wedge f \longrightarrow \Box f$

using *BoxInductSem Valid-def* **by** *blast*

lemma *ChopstarEqv* :

$\vdash f^* = (empty \vee (f \wedge more); f^*)$

using *ChopstarEqvSem Valid-def* **by** *blast*

lemma *OmegaUnroll*:

$\vdash f^\omega = (f \wedge fmore); f^\omega$

using *OmegaUnrollSem Valid-def* **by** *blast*

lemma *OmegaInduct*:

$\vdash (inf \wedge g \wedge \Box(g \longrightarrow (f \wedge fmore);g)) \longrightarrow omega f$

using *OmegaInductSem Valid-def* **by** *blast*

7.3 Additional Lemmas

The following is again from [3, 2] but adapted for our need.

lemma *int-eq-true*: $\vdash P \Longrightarrow \vdash P = \# True$

by *auto*

lemma *int-eq*: $\vdash X = Y \Longrightarrow X = Y$

by (*auto simp: inteq-reflection*)

lemma *int-iff1*:

assumes $\vdash F \longrightarrow G$ **and** $\vdash G \longrightarrow F$

shows $\vdash F = G$

using *assms* **by** *force*

lemma *int-iffD1*: **assumes** *h*: $\vdash F = G$ **shows** $\vdash F \longrightarrow G$

using *h* **by** *auto*

lemma *int-iffD2*: **assumes** *h*: $\vdash F = G$ **shows** $\vdash G \longrightarrow F$

using *h* **by** *auto*

lemma *lift-imp-trans*:

assumes $\vdash A \longrightarrow B$ **and** $\vdash B \longrightarrow C$

shows $\vdash A \longrightarrow C$

using *assms* **by** *force*

lemma *lift-imp-neg*: **assumes** $\vdash A \longrightarrow B$ **shows** $\vdash \neg B \longrightarrow \neg A$

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

lemma *lift-and-com*: $\vdash (A \wedge B) = (B \wedge A)$
by *auto*

7.4 Quantification

lemma *EExI* :
 $\vdash F y \longrightarrow (\exists \exists x. F x)$
by (*simp add: exist-state-d-def Valid-def, auto*)

lemma *EExE*:
 $\llbracket \bigwedge x. \vdash F x \longrightarrow G \rrbracket \implies \vdash (\exists \exists x. F x) \longrightarrow G$
by (*metis (mono-tags, lifting) Valid-def exist-state-d-def unl-lift2*)

lemma *EExValFinite*:
 $((Inl\ w) \models (\exists \exists x. F x)) =$
 $(\exists x (val :: 'a\ interval). (\ (val = (map\ x\ w) \wedge ((Inl\ w) \models F\ x))))$
by (*simp add: exist-state-d-def*)

lemma *EExValInfinite*:
 $((Inr\ w) \models (\exists \exists x. F x)) =$
 $(\exists x (val :: 'a\ infinterval). (\ (val = x \circ w \wedge ((Inr\ w) \models F\ x))))$
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*

7.5 Lemmas about *current-val*

lemma *current-const*: $\vdash \$(\#c) = \#c$
by (*simp add: current-val-d-def intl*)

lemma *current-fun1*: $\vdash \$ (f \langle x \rangle) = f \langle \$x \rangle$
by (*simp add: current-val-d-def intl sum.case-eq-if*)

lemma *current-fun2*: $\vdash \$ (f \langle x, y \rangle) = f \langle \$x, \$y \rangle$
by (*auto simp: current-val-d-def intl sum.case-eq-if*)

lemma *current-fun3*: $\vdash \$ (f \langle x, y, z \rangle) = f \langle \$x, \$y, \$z \rangle$
by (*auto simp: current-val-d-def intl sum.case-eq-if*)

lemma *current-forall*: $\vdash \$ (\forall x. P\ x) = (\forall x. \$ (P\ x))$

by (auto simp: current-val-d-def intl sum.case-eq-if)

lemma current-exists: $\vdash \$(\exists x. P x) = (\exists x. \$(P x))$
by (auto simp: current-val-d-def intl sum.case-eq-if)

lemma current-exists1: $\vdash \$(\exists! x. P x) = (\exists! x. \$(P x))$
by (auto simp: current-val-d-def intl sum.case-eq-if)

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]

7.6 Lemmas about next-val

lemma next-const: $\vdash \text{more} \longrightarrow (\#c)\$ = \#c$
by (auto simp: next-val-d-def more-defs intl sum.case-eq-if)

lemma next-fun1: $\vdash \text{more} \longrightarrow f\langle x \rangle\$ = f\langle x\$ \rangle$
by (auto simp: next-val-d-def more-defs intl sum.case-eq-if)

lemma next-fun2: $\vdash \text{more} \longrightarrow f\langle x, y \rangle\$ = f\langle x\$, y\$ \rangle$
by (auto simp: next-val-d-def more-defs intl sum.case-eq-if)

lemma next-fun3: $\vdash \text{more} \longrightarrow f\langle x, y, z \rangle\$ = f\langle x\$, y\$, z\$ \rangle$
by (auto simp: next-val-d-def more-defs intl sum.case-eq-if)

lemma next-forall: $\vdash \text{more} \longrightarrow (\forall x. P x)\$ = (\forall x. (P x)\$)$
by (auto simp: next-val-d-def intl sum.case-eq-if)

lemma next-exists: $\vdash \text{more} \longrightarrow (\exists x. P x)\$ = (\exists x. (P x)\$)$
by (auto simp: next-val-d-def intl sum.case-eq-if)

lemma next-exists1: $\vdash \text{more} \longrightarrow (\exists! x. P x)\$ = (\exists! x. (P x)\$)$
by (auto simp: next-val-d-def more-defs intl sum.case-eq-if)

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]

7.7 Lemmas about fin-val

lemma fin-const: $\vdash \text{finite} \longrightarrow !(\#c) = \#c$
by (auto simp: fin-val-d-def finite-defs intl sum.case-eq-if)

lemma fin-fun1: $\vdash \text{finite} \longrightarrow !(f\langle x \rangle) = f\langle !x \rangle$
by (auto simp: fin-val-d-def finite-defs intl sum.case-eq-if)

lemma *fin-fun2*: $\vdash \text{finite} \longrightarrow !(f \langle x, y \rangle) = f \langle !x, !y \rangle$
by (*auto simp: fin-val-d-def finite-defs intl sum.case-eq-if*)

lemma *fin-fun3*: $\vdash \text{finite} \longrightarrow !(f \langle x, y, z \rangle) = f \langle !x, !y, !z \rangle$
by (*auto simp: fin-val-d-def finite-defs intl sum.case-eq-if*)

lemma *fin-forall*: $\vdash \text{finite} \longrightarrow !(\forall x. P x) = (\forall x. !(P x))$
by (*auto simp: fin-val-d-def finite-defs intl sum.case-eq-if*)

lemma *fin-exists*: $\vdash \text{finite} \longrightarrow !(\exists x. P x) = (\exists x. !(P x))$
by (*auto simp: fin-val-d-def finite-defs intl sum.case-eq-if*)

lemma *fin-exists1*: $\vdash \text{finite} \longrightarrow !(\exists ! x. P x) = (\exists ! x. !(P x))$
by (*auto simp: fin-val-d-def finite-defs intl sum.case-eq-if*)

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]*

7.8 Lemmas about *penult-val*

lemma *penult-const*: $\vdash \text{more} \wedge \text{finite} \longrightarrow (\#c)! = \#c$
by (*auto simp: penult-val-d-def more-defs finite-defs intl sum.case-eq-if*)

lemma *penult-fun1*: $\vdash \text{more} \wedge \text{finite} \longrightarrow f \langle x \rangle! = f \langle x! \rangle$
by (*auto simp: penult-val-d-def more-defs finite-defs intl sum.case-eq-if*)

lemma *penult-fun2*: $\vdash \text{more} \wedge \text{finite} \longrightarrow f \langle x, y \rangle! = f \langle x!, y! \rangle$
by (*auto simp: penult-val-d-def more-defs finite-defs intl sum.case-eq-if*)

lemma *penult-fun3*: $\vdash \text{more} \wedge \text{finite} \longrightarrow f \langle x, y, z \rangle! = f \langle x!, y!, z! \rangle$
by (*auto simp: penult-val-d-def more-defs finite-defs intl sum.case-eq-if*)

lemma *penult-forall*: $\vdash \text{more} \wedge \text{finite} \longrightarrow (\forall x. P x)! = (\forall x. (P x)!)$
by (*auto simp: penult-val-d-def finite-defs intl sum.case-eq-if*)

lemma *penult-exists*: $\vdash \text{more} \wedge \text{finite} \longrightarrow (\exists x. P x)! = (\exists x. (P x)!)$
by (*auto simp: penult-val-d-def finite-defs intl sum.case-eq-if*)

lemma *penult-exists1*: $\vdash \text{more} \wedge \text{finite} \longrightarrow (\exists ! x. P x)! = (\exists ! x. (P x)!)$
by (*auto simp: penult-val-d-def more-defs finite-defs intl sum.case-eq-if*)

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]*

7.9 Basic temporal variables properties

lemma *empty-imp-fin-equiv-curr*:

$\vdash \text{empty} \longrightarrow !v = \v

by (*simp add: Valid-def current-val-d-def empty-defs finval-defs sum.case-eq-if*)

lemma *skip-imp-fin-equiv-next*:

$\vdash \text{skip} \longrightarrow !v = v\$$

by (*simp add: Valid-def skip-defs next-val-d-def finval-defs sum.case-eq-if*)

lemma *skip-imp-penult-equiv-curr*:

$\vdash \text{skip} \longrightarrow v! = \v

by (*simp add: Valid-def skip-defs penultval-defs current-val-d-def sum.case-eq-if*)

end

8 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 [6].

8.1 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: ifthenelse-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*)

8.2 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: init-defs*)

lemma *Finprop* :

$\vdash ((\# \text{True}; (f \wedge \text{empty})) \wedge (\# \text{True}; (g \wedge \text{empty}))) = (\# \text{True}; ((f \wedge g) \wedge \text{empty}))$
 $\vdash ((\# \text{True}; (f \wedge \text{empty})) \vee (\# \text{True}; (g \wedge \text{empty}))) = (\# \text{True}; ((f \vee g) \wedge \text{empty}))$
 $\vdash (\# \text{True}; ((\# \text{True}) \wedge \text{empty}))$
 $\vdash (\neg (\# \text{True}; (f \wedge \text{empty}))) = (\# \text{True}; (\neg f \wedge \text{empty}))$

by (*auto simp: finalt-defs*) (*simp add: chop-defs empty-defs interval-suffix-length, fastforce*)

8.3 Basic Theorems

lemma *BiChopImpChop* :

$\vdash \text{bi } (f \longrightarrow f1) \longrightarrow f;g \longrightarrow f1;g$

proof –

have 1: $\vdash g \longrightarrow g$ **by** *auto*

hence 2: $\vdash \Box (g \longrightarrow g)$ **by** (*rule BoxGen*)

have 3: $\vdash \text{bi } (f \longrightarrow f1) \wedge \Box(g \longrightarrow g) \longrightarrow f;g \longrightarrow f1;g$ **by** (*rule BiBoxChopImpChop*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *AndChopA*:

$\vdash (f \wedge f1);g \longrightarrow f;g$

proof –

have 1: $\vdash f \wedge f1 \longrightarrow f$ **by** *auto*

hence 2: $\vdash \text{bi } (f \wedge f1 \longrightarrow f)$ **by** (*rule BiGen*)

have 3: $\vdash \text{bi } (f \wedge f1 \longrightarrow f) \longrightarrow (f \wedge f1);g \longrightarrow f;g$ **by** (*rule BiChopImpChop*)

from 2 3 **show** *?thesis* **using** *MP* **by** *blast*

qed

lemma *AndChopB*:

$\vdash (f \wedge f1);g \longrightarrow f1;g$

proof –

have 1: $\vdash f \wedge f1 \longrightarrow f1$ **by** *auto*

hence 2: $\vdash \text{bi } (f \wedge f1 \longrightarrow f1)$ **by** (*rule BiGen*)

have 3: $\vdash \text{bi } (f \wedge f1 \longrightarrow f1) \longrightarrow (f \wedge f1);g \longrightarrow f1;g$ **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: $\vdash \text{skip};(f;g) = (\text{skip};f);g$ **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 $\vdash (f;g) = (f;g1)$
proof –
have 1: $\vdash g = g1$ **using** assms **by** auto
have 2: $(\vdash g \longrightarrow g1) \implies (\vdash f;g \longrightarrow f;g1)$ **by** (rule RightChopImpChop)
have 3: $(\vdash g1 \longrightarrow g) \implies (\vdash f;g1 \longrightarrow f;g)$ **by** (rule RightChopImpChop)
from 1 2 3 **show** ?thesis **by** fastforce
qed

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-iff1 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-iff1)

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*:
 $\vdash \Box (f \longrightarrow g) \longrightarrow \Box f \longrightarrow \Box g$
proof –
have 1: $\vdash (\neg (f \longrightarrow g)) = (f \wedge \neg g)$ **by** *auto*
hence 2: $\vdash skip;(\neg (f \longrightarrow g)) = skip;(f \wedge \neg g)$ **by** (rule *RightChopEqvChop*)
have 3: $\vdash f \longrightarrow g \vee (f \wedge \neg g)$ **by** *auto*
hence 4: $\vdash skip;f \longrightarrow (skip;g) \vee (skip;(f \wedge \neg g))$ **by** (rule *ChopOrImpRule*)
hence 5: $\vdash \neg (skip;(f \wedge \neg g)) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **by** *auto*
have 6: $\vdash \neg (skip;(\neg (f \longrightarrow g))) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **using** 2 5 **by** *fastforce*
hence 7: $\vdash \neg (\Box (\neg (f \longrightarrow g))) \longrightarrow (\Box f) \longrightarrow (\Box g)$ **by** (simp add: next-d-def)
have 8: $\vdash \Box (f \longrightarrow g) \longrightarrow \neg (\Box (\neg (f \longrightarrow g)))$ **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: $\vdash empty;f = f$ **by** (rule *EmptyChop*)
have 2: $\vdash empty \longrightarrow \#True$ **by** *auto*
hence 3: $\vdash empty;f \longrightarrow \#True;f$ **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: $\vdash \text{skip};(\# \text{True};f) = ((\text{skip};\# \text{True});f)$ **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: $\vdash \Box((\neg g) \longrightarrow \neg f) \longrightarrow (\#True; (\neg g)) \longrightarrow (\#True; (\neg f))$ **by** (*rule BoxChopImpChop*)
have 4: $\vdash \Box(f \longrightarrow g) \longrightarrow (\#True; (\neg g)) \longrightarrow (\#True; (\neg f))$ **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*:

$\vdash \Diamond \text{empty}$

proof —

have 1: $\vdash \#True$ **by** *auto*
have 2: $\vdash \#True; \text{empty} = \#True$ **by** (*rule ChopEmpty*)
have 3: $\vdash \#True; \text{empty}$ **using** 1 2 **by** *auto*
from 3 **show** ?thesis **by** (*simp add: sometimes-d-def*)

qed

lemma *NextEqvNext*:

assumes $\vdash f = g$

shows $\vdash \bigcirc f = \bigcirc g$

proof —

have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash \text{skip}; f = \text{skip}; g$ **by** (*rule RightChopEqvChop*)
from 1 **show** ?thesis **by** (*metis 2 next-d-def*)

qed

lemma *NextAndNextImpNextRule*:

assumes $\vdash (f \wedge g) \longrightarrow h$

shows $\vdash (\bigcirc f \wedge \bigcirc g) \longrightarrow \bigcirc h$

using *assms* **by** (*auto simp: next-defs*)

lemma *NextAndNextEqvNextRule*:

assumes $\vdash (f \wedge g) = h$

shows $\vdash (\bigcirc f \wedge \bigcirc g) = \bigcirc h$

using *assms* **by** (*metis NextAndNextImpNextRule Prop11 Prop12 int-eq int-simps(20)*)

lemma *WeakNextEqvWeakNext*:

assumes $\vdash f = g$

shows $\vdash \text{wnext } f = \text{wnext } g$

using *assms* **using** *inteq-reflection* **by** *force*

lemma *DiamondImpDiamond*:

assumes $\vdash f \longrightarrow g$

shows $\vdash \Diamond f \longrightarrow \Diamond g$

using *assms* **by** (*simp add: RightChopImpChop sometimes-d-def*)

lemma *DiamondEqvDiamond*:

assumes $\vdash f = g$

shows $\vdash \Diamond f = \Diamond g$

using *assms* **using** *int-eq* **by** *force*

lemma *BoxEqvBox*:

assumes $\vdash f = g$

shows $\vdash \Box f = \Box g$

using *assms* **using** *inteq-reflection* **by** *force*

lemma *BoxAndBoxImpBoxRule*:

assumes $\vdash f \wedge g \longrightarrow h$

shows $\vdash \Box f \wedge \Box g \longrightarrow \Box h$

using *assms* **by** (*auto simp: always-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 $\vdash f \longrightarrow g$

$\vdash \text{more} \wedge f \longrightarrow \bigcirc f$

shows $\vdash f \longrightarrow \Box g$

proof –

have 1: $\vdash \text{more} \wedge f \longrightarrow \bigcirc f$ **using** *assms* **by** *auto*

hence 2: $\vdash f \longrightarrow (\text{empty} \vee \bigcirc f)$ **by** (*auto simp: next-defs empty-defs more-defs*)

hence 3: $\vdash f \longrightarrow \text{wnext } f$ **by** (*auto simp: wnext-defs empty-defs next-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 $\vdash f \longrightarrow \bigcirc f$

shows $\vdash \neg f$

proof –

have 1: $\vdash f \longrightarrow \bigcirc f$ **using** *assms* **by** *auto*

hence 2: $\vdash f \longrightarrow (\text{more} \wedge \text{wnext } f)$ **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: *always-defs more-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: *empty-defs wnext-defs next-defs*)

lemma *NotEmptyAndNext*:
 $\vdash \neg(\text{empty} \wedge \bigcirc f)$
by (auto simp: *empty-defs next-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-iff1 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: *always-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: always-defs more-defs next-defs wnext-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: wnext-defs always-defs next-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: always-defs more-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: next-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: chop-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 $\vdash empty \longrightarrow f$

$\vdash \circ f \longrightarrow f$
shows $\vdash f$
proof –
have 1: $\vdash \text{empty} \longrightarrow f$ **using** *assms* **by** *auto*
have 2: $\vdash \circ f \longrightarrow f$ **using** *assms* **by** *blast*
have 3: $\vdash (\text{empty} \vee \circ f) \longrightarrow f$ **using** 1 2 **by** *fastforce*
have 4: $\vdash \text{wnext } f = (\text{empty} \vee \circ f)$ **by** (rule *WnextEqvEmptyOrNext*)
hence 5: $\vdash \text{wnext } f \longrightarrow f$ **using** 3 **by** *fastforce*
hence 6: $\vdash \neg f \longrightarrow \neg (\text{wnext } f)$ **by** *auto*
hence 7: $\vdash \neg f \longrightarrow \circ(\neg f)$ **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 $\vdash \text{empty} \wedge f \longrightarrow g$
 $\vdash \circ(f \longrightarrow g) \wedge f \longrightarrow g$
shows $\vdash f \longrightarrow g$
proof –
have 1: $\vdash \text{empty} \wedge f \longrightarrow g$ **using** *assms* **by** *auto*
have 2: $\vdash \circ(f \longrightarrow g) \wedge f \longrightarrow g$ **using** *assms* **by** *blast*
have 3: $\vdash (\text{empty} \vee \circ(f \longrightarrow g)) \wedge f \longrightarrow g$ **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: $\vdash \neg (f \longrightarrow g) \longrightarrow \circ(\neg(f \longrightarrow g))$ **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 *TEMP* ($\text{empty} \longrightarrow f$) *TEMP* ($\text{empty} \longrightarrow g$)] *assms fin-d-def*
by (smt *intl intensional-rews*(3) *inteq-reflection Prop10*)

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: $\vdash \square(\text{empty} = f) = \square(\text{empty} = g)$ **by** (*rule BoxEqvBox*)

from 3 **show** *?thesis* **by** (*simp add: halt-d-def*)

qed

lemma *BiImpDiImpDi*:

$\vdash \text{bi } (f \longrightarrow g) \longrightarrow \text{di } f \longrightarrow \text{di } g$

proof –

have 1: $\vdash \text{bi } (f \longrightarrow g) \longrightarrow (f; \# \text{True}) \longrightarrow (g; \# \text{True})$ **by** (*rule BiChopImpChop*)

from 1 **show** *?thesis* **by** (*simp add: di-d-def*)

qed

lemma *DiImpDi*:

assumes $\vdash f \longrightarrow g$

shows $\vdash \text{di } f \longrightarrow \text{di } g$

proof –

have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*

hence 2: $\vdash f; \# \text{True} \longrightarrow g; \# \text{True}$ **by** (*rule LeftChopImpChop*)

from 2 **show** *?thesis* **by** (*simp add: di-d-def*)

qed

lemma *BiImpBiRule*:

assumes $\vdash f \longrightarrow g$

shows $\vdash \text{bi } f \longrightarrow \text{bi } g$

proof –

have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*

hence 2: $\vdash \neg g \longrightarrow \neg f$ **by** *auto*

hence 3: $\vdash \text{di } (\neg g) \longrightarrow \text{di } (\neg f)$ **by** (*rule DiImpDi*)

hence 4: $\vdash \neg (\text{di } (\neg f)) \longrightarrow \neg (\text{di } (\neg g))$ **by** *auto*

from 4 **show** *?thesis* **by** (*simp add: bi-d-def*)

qed

lemma *DiEqvDi*:

assumes $\vdash f = g$

shows $\vdash \text{di } f = \text{di } g$

proof –

have 1: $\vdash f = g$ **using** *assms* **by** *auto*

hence 2: $\vdash f; \# \text{True} = g; \# \text{True}$ **by** (*rule LeftChopEqvChop*)

from 2 **show** *?thesis* **by** (*simp add: di-d-def*)

qed

lemma *BiEqvBi*:

assumes $\vdash f = g$

shows $\vdash bi\ f = bi\ g$

proof –

have 1: $\vdash f = g$ **using** *assms* **by** *auto*

hence 2: $\vdash (\neg f) = (\neg g)$ **by** *auto*

hence 3: $\vdash di\ (\neg f) = di\ (\neg g)$ **by** (*rule DiEqvDi*)

hence 4: $\vdash (\neg (di\ (\neg f))) = (\neg (di\ (\neg g)))$ **by** *auto*

from 4 **show** *?thesis* **by** (*simp add: bi-d-def*)

qed

lemma *LeftChopChopImpChopRule*:

assumes $\vdash (f; g) \longrightarrow g$

shows $\vdash (f; g); h \longrightarrow (g; h)$

proof –

have 1: $\vdash (f; g) \longrightarrow g$ **using** *assms* **by** *blast*

hence 2: $\vdash (f; g); h \longrightarrow g; h$ **by** (*rule LeftChopImpChop*)

have 3: $\vdash f; (g; h) = (f; g); h$ **by** (*rule ChopAssoc*)

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

qed

lemma *AndChopCommute* :

$\vdash (f \wedge f1); g = (f1 \wedge f); g$

proof –

have 1: $\vdash (f \wedge f1) = (f1 \wedge f)$ **by** *auto*

from 1 **show** *?thesis* **by** (*rule LeftChopEqvChop*)

qed

lemma *BiAndChopImport*:

$\vdash bi\ f \wedge (f1; g) \longrightarrow (f \wedge f1); g$

proof –

have 1: $\vdash f \longrightarrow (f1 \longrightarrow f \wedge f1)$ **by** *auto*

hence 2: $\vdash bi\ f \longrightarrow bi\ (f1 \longrightarrow f \wedge f1)$ **by** (*rule BilmpBiRule*)

have 3: $\vdash bi\ (f1 \longrightarrow (f \wedge f1)) \longrightarrow f1; g \longrightarrow (f \wedge f1); g$ **by** (*rule BiChopImpChop*)

from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*

qed

lemma *StateAndChopImport*:

$\vdash (init\ w) \wedge (f; g) \longrightarrow ((init\ w) \wedge f); g$

proof –

have 1: $\vdash (init\ w) \longrightarrow bi\ (init\ w)$ **by** (*rule StatImpBi*)

hence 2: $\vdash (init\ w) \wedge (f; g) \longrightarrow bi\ (init\ w) \wedge (f; g)$ **by** *auto*

have 3: $\vdash bi\ (init\ w) \wedge (f; g) \longrightarrow ((init\ w) \wedge f); g$ **by** (*rule BiAndChopImport*)

from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*

qed

8.4 Further Properties Di and Bi

lemma *ImpDi*:

$\vdash f \longrightarrow di\ f$

proof –

have 1: $\vdash f; \text{empty} = f$ **by** (rule *ChopEmpty*)

have 2: $\vdash \text{empty} \longrightarrow \#True$ **by** *auto*

hence 3: $\vdash f; \text{empty} \longrightarrow f; \#True$ **by** (rule *RightChopImpChop*)

have 4 : $\vdash f \longrightarrow f; \#True$ **using** 1 3 **by** *fastforce*

from 4 **show** *?thesis* **by** (*simp add: di-d-def*)

qed

lemma *DiState*:

$\vdash di\ (\text{init } w) = (\text{init } w)$

proof –

have 0: $\vdash (\text{init } (\neg w)) \longrightarrow bi\ (\text{init } (\neg w))$ **using** *StateImpBi* **by** *fastforce*

hence 1: $\vdash \neg(\text{init } w) \longrightarrow bi\ (\neg(\text{init } w))$ **using** *Initprop(2)* **by** (*metis inteq-reflection*)

hence 2: $\vdash (\neg(\text{init } w)) \longrightarrow \neg(di\ (\neg\neg(\text{init } w)))$ **by** (*simp add: bi-d-def*)

have 3: $\vdash (\neg(\text{init } w) \longrightarrow \neg(di\ (\neg\neg(\text{init } w)))) \longrightarrow (di\ (\neg\neg(\text{init } w)) \longrightarrow (\text{init } w))$ **by** *auto*

have 4: $\vdash di\ (\neg\neg(\text{init } w)) \longrightarrow (\text{init } w)$ **using** 2 3 *MP* **by** *blast*

have 5: $\vdash (\text{init } w) \longrightarrow \neg\neg(\text{init } w)$ **by** *auto*

hence 6: $\vdash di\ (\text{init } w) \longrightarrow di\ (\neg\neg(\text{init } w))$ **by** (rule *DiImpDi*)

have 7: $\vdash di\ (\text{init } w) \longrightarrow (\text{init } w)$ **using** 6 4 **using** *lift-imp-trans* **by** *metis*

have 8: $\vdash (\text{init } w) \longrightarrow di\ (\text{init } w)$ **by** (rule *ImpDi*)

from 7 8 **show** *?thesis* **by** *fastforce*

qed

lemma *StateChop*:

$\vdash (\text{init } w); f \longrightarrow (\text{init } w)$

using *DiState* **by** (*auto simp: di-defs init-defs chop-defs*)

lemma *StateChopExportA*:

$\vdash ((\text{init } w) \wedge f); g \longrightarrow (\text{init } w)$

using *DiState* **by** (*auto simp: init-defs chop-defs*)

lemma *StateAndChop*:

$\vdash ((\text{init } w) \wedge f); g = ((\text{init } w) \wedge (f; g))$

by (*simp add: AndChopB StateAndChopImport StateChopExportA Prop11 Prop12*)

lemma *StateAndChopImpChopRule*:

assumes $\vdash (\text{init } w) \wedge f \longrightarrow f1$

shows $\vdash (\text{init } w) \wedge (f; g) \longrightarrow (f1; g)$

proof –

have 1: $\vdash (\text{init } w) \wedge f \longrightarrow f1$ **using** *assms* **by** *auto*

hence 2: $\vdash ((\text{init } w) \wedge f); g \longrightarrow f1; g$ **by** (rule *LeftChopImpChop*)

have 3: $\vdash ((\text{init } w) \wedge f); g = ((\text{init } w) \wedge (f; g))$ **by** (rule *StateAndChop*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *StateImpChopEqvChop* :

assumes $\vdash (\text{init } w) \longrightarrow (f = f1)$

shows $\vdash (\text{init } w) \longrightarrow ((f; g) = (f1; g))$
proof –
have 1: $\vdash (\text{init } w) \longrightarrow (f = f1)$ **using** *assms* **by** *auto*
hence 2: $\vdash (\text{init } w) \wedge f \longrightarrow f1$ **by** *auto*
hence 3: $\vdash (\text{init } w) \wedge (f; g) \longrightarrow (f1; g)$ **by** (*rule StateAndChopImpChopRule*)
have 4: $\vdash (\text{init } w) \wedge f1 \longrightarrow f$ **using** 1 **by** *auto*
hence 5: $\vdash (\text{init } w) \wedge (f1; g) \longrightarrow (f; g)$ **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 *DilIntro*:
 $\vdash f \longrightarrow di \ f$
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: $\vdash f; \text{empty} \longrightarrow di \ f$ **by** (*simp add: di-d-def*)
from 1 6 **show** *?thesis* **by** *fastforce*
qed

lemma *BiElim*:
 $\vdash bi \ f \longrightarrow f$
proof –
have 1: $\vdash \neg f \longrightarrow di (\neg f)$ **by** (*rule DilIntro*)
have 2: $\vdash (\neg f \longrightarrow di (\neg f)) \longrightarrow (\neg (di (\neg f)) \longrightarrow f)$ **by** *auto*
have 3: $\vdash \neg (di (\neg f)) \longrightarrow f$ **using** 1 2 *MP* **by** *blast*
from 3 **show** *?thesis* **by** (*metis bi-d-def*)
qed

lemma *BiContraPosImpDist*:
 $\vdash bi (\neg g \longrightarrow \neg f) \longrightarrow (bi \ f) \longrightarrow (bi \ g)$
proof –
have 1: $\vdash bi (\neg g \longrightarrow \neg f) \longrightarrow (di (\neg g)) \longrightarrow (di (\neg f))$ **by** (*rule BilmpDilmpDi*)
hence 2: $\vdash bi (\neg g \longrightarrow \neg f) \longrightarrow (\neg (di (\neg f))) \longrightarrow (\neg (di (\neg g)))$ **by** *auto*
from 2 **show** *?thesis* **by** (*metis bi-d-def*)
qed

lemma *BilmpDist*:

$\vdash bi (f \longrightarrow g) \longrightarrow (bi\ f) \longrightarrow (bi\ g)$
proof –
have 1: $\vdash (f \longrightarrow g) \longrightarrow (\neg g \longrightarrow \neg f)$ **by** *auto*
hence 2: $\vdash \neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g)$ **by** *auto*
hence 3: $\vdash bi (\neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g))$ **by** (*rule BiGen*)
have 4: $\vdash bi (\neg (\neg g \longrightarrow \neg f) \longrightarrow \neg (f \longrightarrow g))$
 \longrightarrow
 $bi (f \longrightarrow g) \longrightarrow bi (\neg g \longrightarrow \neg f)$ **by** (*rule BiContraPosImpDist*)
have 5: $\vdash bi (f \longrightarrow g) \longrightarrow bi (\neg g \longrightarrow \neg f)$ **using** 3 4 *MP* **by** *blast*
have 6: $\vdash bi (\neg g \longrightarrow \neg f) \longrightarrow (bi\ f) \longrightarrow (bi\ g)$ **by** (*rule BiContraPosImpDist*)
from 5 6 **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *IfChopEqvRule*:

assumes $\vdash f = if_i (init\ w)$ **then** $f1$ **else** $f2$
shows $\vdash f; g = if_i (init\ w)$ **then** $(f1; g)$ **else** $(f2; g)$
proof –
have 1: $\vdash f = if_i (init\ w)$ **then** $f1$ **else** $f2$
using *assms* **by** *auto*
hence 2: $\vdash f = (((init\ w) \wedge f1) \vee ((init\ (\neg w)) \wedge f2))$
by (*simp add: ifthenelse-d-def init-defs Valid-def*)
hence 3: $\vdash f; g = (((init\ w) \wedge f1); g \vee ((init\ (\neg w)) \wedge f2); g)$
by (*rule OrChopEqvRule*)
have 4: $\vdash ((init\ w) \wedge f1); g = ((init\ w) \wedge (f1; g))$
by (*rule StateAndChop*)
have 5: $\vdash ((init\ (\neg w)) \wedge f2); g = ((init\ (\neg w)) \wedge (f2; g))$
by (*rule StateAndChop*)
have 6: $\vdash f; g = (((init\ w) \wedge f1; g) \vee ((init\ (\neg w)) \wedge f2; g))$
using 3 4 5 **by** *fastforce*
from 6 **show** *?thesis* **by** (*simp add: ifthenelse-d-def init-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*:

$\vdash (empty \vee f); g = (g \vee (f; g))$
proof –
have 1: $\vdash (empty \vee f); g = ((empty; g) \vee (f; g))$ **by** (*rule OrChopEqv*)
have 2: $\vdash empty; g = g$ **by** (*rule EmptyChop*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextChopEqv*:

$\vdash (\text{empty} \vee \circ f); g = (g \vee \circ(f; g))$

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: init-defs empty-defs always-defs Valid-def*)

lemma *BoxEqvAndBox*:
 $\vdash \Box f = (f \wedge \Box f)$
by (*simp add: always-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*:
 $\vdash \Box (\text{init } w); \Box (\text{init } w) = \Box (\text{init } w)$
proof –
have 1: $\vdash (\Box (\text{init } w)) = ((\text{init } w) \wedge (\text{empty} \vee \bigcirc(\Box (\text{init } w))))$
by (*rule BoxEqvAndEmptyOrNextBox*)
hence 2: $\vdash (\Box (\text{init } w); \Box (\text{init } w)) =$
 $((\text{init } w) \wedge (\text{empty} \vee \bigcirc(\Box (\text{init } w))); \Box (\text{init } w))$
by (*metis StateAndChop inteq-reflection*)
have 3: $\vdash ((\text{empty} \vee \bigcirc(\Box (\text{init } w))); \Box (\text{init } w)) =$
 $(\Box (\text{init } w) \vee \bigcirc(\Box (\text{init } w); \Box (\text{init } w)))$
by (*rule EmptyOrNextChopEqv*)
have 4: $\vdash (\Box (\text{init } w); \Box (\text{init } w)) =$
 $((\text{init } w) \wedge (\Box (\text{init } w) \vee \bigcirc(\Box (\text{init } w); \Box (\text{init } w))))$
using 2 3 **by** *fastforce*
have 5: $\vdash \neg(\Box (\text{init } w)) \longrightarrow \neg(\text{init } w) \vee \neg(\bigcirc(\Box (\text{init } w)))$
by (*rule NotBoxImpNotOrNotNextBox*)
have 6: $\vdash (\Box (\text{init } w); \Box (\text{init } w)) \wedge \neg(\Box (\text{init } w)) \longrightarrow$
 $\bigcirc(\Box (\text{init } w); \Box (\text{init } w)) \wedge \neg(\bigcirc(\Box (\text{init } w)))$
using 4 5 **by** *fastforce*
hence 7: $\vdash \Box (\text{init } w); \Box (\text{init } w) \longrightarrow \Box (\text{init } w)$
by (*rule NextContra*)
have 11: $\vdash \Box (\text{init } w) = ((\text{init } w) \wedge \Box (\text{init } w))$
by (*rule BoxEqvAndBox*)
have 12: $\vdash \text{empty}; \Box (\text{init } w) = \Box (\text{init } w)$

by (rule EmptyChop)
 have 13: $\vdash ((\text{init } w) \wedge \text{empty}); \Box (\text{init } w) = ((\text{init } w) \wedge (\text{empty}; \Box (\text{init } w)))$
 by (rule StateAndChop)
 have 14: $\vdash \Box (\text{init } w) = ((\text{init } w) \wedge \text{empty}); \Box (\text{init } w)$
 using 11 12 13 by fastforce
 have 15: $\vdash (\text{init } w) \wedge \text{empty} \longrightarrow \Box (\text{init } w)$
 by (rule StateAndEmptyImpBoxState)
 hence 16: $\vdash ((\text{init } w) \wedge \text{empty}); \Box (\text{init } w) \longrightarrow \Box (\text{init } w); \Box (\text{init } w)$
 by (rule LeftChopImpChop)
 have 17: $\vdash \Box (\text{init } w) \longrightarrow \Box (\text{init } w); \Box (\text{init } w)$
 using 14 16 by fastforce
 from 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)

8.5 Properties of Da and Ba

lemma DaEqvDtDi:
 $\vdash \text{da } f = \Diamond (di \ f)$
proof –
 have 1: $\vdash \# \text{True}; (f; \# \text{True}) = \# \text{True}; (f; \# \text{True})$ by auto
 hence 2: $\vdash \# \text{True}; (f; \# \text{True}) = \# \text{True}; di \ f$ by (simp add: di-d-def)
 have 3: $\vdash \# \text{True}; di \ f = \Diamond (di \ f)$ by (rule TrueChopEqvDiamond)
 have 4: $\vdash \# \text{True}; (f; \# \text{True}) = \Diamond (di \ f)$ using 2 3 by fastforce
 from 4 show ?thesis by (simp add: da-d-def)
 qed

lemma DaEqvDiDt:

$\vdash da\ f = di\ (\Diamond f)$
proof –
have 1: $\vdash \#True; f = \Diamond f$ **by** (rule TrueChopEqvDiamond)
hence 2: $\vdash (\#True; f); \#True = (\Diamond f); \#True$ **by** (rule LeftChopEqvChop)
hence 3: $\vdash (\#True; f); \#True = di\ (\Diamond f)$ **by** (simp add: di-d-def)
have 4: $\vdash \#True; (f; \#True) = (\#True; f); \#True$ **by** (rule ChopAssoc)
have 5: $\vdash \#True; (f; \#True) = di\ (\Diamond f)$ **using** 3 4 **by** fastforce
from 5 **show** ?thesis **by** (simp add: da-d-def)
qed

lemma DtDiEqvDiDt:
 $\vdash \Diamond (di\ f) = di\ (\Diamond f)$
by (metis ChopAssoc di-d-def sometimes-d-def)

lemma DiamondNotEqvNotBox:
 $\vdash \Diamond (\neg f) = (\neg (\Box f))$
by (simp add: always-d-def)

lemma BaEqvBiBt:
 $\vdash ba\ f = bi\ (\Box f)$
proof –
have 1: $\vdash da\ (\neg f) = di\ (\Diamond (\neg f))$ **by** (rule DaEqvDiDt)
have 2: $\vdash \Diamond (\neg f) = (\neg (\Box f))$ **by** (rule DiamondNotEqvNotBox)
hence 3: $\vdash di\ (\Diamond (\neg f)) = di\ (\neg (\Box f))$ **by** (rule DiEqvDi)
have 4: $\vdash da\ (\neg f) = di\ (\neg (\Box f))$ **using** 1 3 **by** fastforce
hence 5: $\vdash (\neg (da\ (\neg f))) = (\neg (di\ (\neg (\Box f))))$ **by** auto
hence 6: $\vdash (\neg (da\ (\neg f))) = bi\ (\Box f)$ **by** (simp add: bi-d-def)
from 6 **show** ?thesis **by** (simp add: ba-d-def)
qed

lemma DiNotEqvNotBi:
 $\vdash di\ (\neg f) = (\neg (bi\ f))$
proof –
have 1: $\vdash bi\ f = (\neg (di\ (\neg f)))$ **by** (simp add: bi-d-def)
from 1 **show** ?thesis **by** auto
qed

lemma NotDiamondNotEqvBox:
 $\vdash (\neg (\Diamond (\neg f))) = \Box f$
by (simp add: always-d-def)

lemma BaEqvBtBi:
 $\vdash ba\ f = \Box (bi\ f)$
proof –
have 1: $\vdash da\ (\neg f) = \Diamond (di\ (\neg f))$ **by** (rule DaEqvDtDi)
have 2: $\vdash di\ (\neg f) = (\neg (bi\ f))$ **by** (rule DiNotEqvNotBi)
hence 3: $\vdash \Diamond (di\ (\neg f)) = \Diamond (\neg (bi\ f))$ **by** (rule DiamondEqvDiamond)
have 4: $\vdash (\neg (\Diamond (\neg (bi\ f)))) = \Box (bi\ f)$ **by** (rule NotDiamondNotEqvBox)
have 5: $\vdash (\neg (da\ (\neg f))) = \Box (bi\ f)$ **using** 1 2 3 4 **by** fastforce
from 5 **show** ?thesis **by** (simp add: ba-d-def)

qed

lemma *BtBiEqvBiBt*:

$\vdash \Box (bi\ f) = bi(\Box\ f)$

proof –

have 1: $\vdash ba\ f = \Box (bi\ f)$ **by** (rule *BaEqvBtBi*)

have 2: $\vdash ba\ f = bi(\Box\ f)$ **by** (rule *BaEqvBiBt*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *BoxStateEqvBaBoxState*:

$\vdash \Box (init\ w) = ba(\Box (init\ w))$

proof –

have 1: $\vdash (init\ w) = bi\ (init\ w)$ **by** (rule *StateEqvBi*)

hence 2: $\vdash \Box (init\ w) = \Box (bi\ (init\ w))$ **by** (rule *BoxEqvBox*)

have 3: $\vdash \Box (bi\ (init\ w)) = bi(\Box (init\ w))$ **by** (rule *BtBiEqvBiBt*)

have 4: $\vdash \Box (init\ w) = \Box(\Box (init\ w))$ **by** (rule *BoxEqvBoxBox*)

hence 5: $\vdash bi(\Box (init\ w)) = bi(\Box(\Box (init\ w)))$ **by** (rule *BiEqvBi*)

have 6: $\vdash ba(\Box (init\ w)) = bi(\Box(\Box (init\ w)))$ **by** (rule *BaEqvBiBt*)

from 2 3 5 6 **show** ?thesis **by** fastforce

qed

lemma *BaImpBi*:

$\vdash ba\ f \longrightarrow bi\ f$

proof –

have 1: $\vdash ba\ f = \Box(bi\ f)$ **by** (rule *BaEqvBtBi*)

have 2: $\vdash \Box(bi\ f) \longrightarrow bi\ f$ **by** (rule *BoxElim*)

from 1 2 **show** ?thesis **using** lift-imp-trans **by** fastforce

qed

lemma *BaImpBt*:

$\vdash ba\ f \longrightarrow \Box\ f$

proof –

have 1: $\vdash ba\ f = bi(\Box\ f)$ **by** (rule *BaEqvBiBt*)

have 2: $\vdash bi(\Box\ f) \longrightarrow \Box\ f$ **by** (rule *BiElim*)

from 1 2 **show** ?thesis **using** lift-imp-trans **by** fastforce

qed

lemma *DiamondImpDa*:

$\vdash \Diamond\ f \longrightarrow da\ f$

by (metis *DiIntro DiamondImpDiamond da-d-def di-d-def sometimes-d-def*)

lemma *DiImpDa*:

$\vdash di\ f \longrightarrow da\ f$

by (metis *NowImpDiamond da-d-def di-d-def sometimes-d-def*)

lemma *BoxAndChopImport*:

$\vdash \Box\ h \wedge f; g \longrightarrow f; (h \wedge g)$

proof –

have 1: $\vdash h \longrightarrow g \longrightarrow (h \wedge g)$ **by** auto

hence 2: $\vdash \Box h \longrightarrow \Box(g \longrightarrow (h \wedge g))$ **by** (rule *ImpBoxRule*)
have 3: $\vdash \Box(g \longrightarrow (h \wedge g)) \longrightarrow f; g \longrightarrow f; (h \wedge g)$ **by** (rule *BoxChopImpChop*)
from 2 3 **show** ?thesis **by** fastforce
qed

lemma *BaAndChopImport*:

$\vdash \text{ba } f \wedge (g; g1) \longrightarrow (f \wedge g); (f \wedge g1)$

proof –

have 1: $\vdash \text{ba } f \longrightarrow \text{bi } f$ **by** (rule *BalmpBi*)
have 2: $\vdash \text{bi } f \wedge (g; g1) \longrightarrow (f \wedge g); g1$ **by** (rule *BiAndChopImport*)
have 3: $\vdash \text{ba } f \longrightarrow \Box f$ **by** (rule *BalmpBt*)
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*:

$\vdash (\Box(\text{init } w) \wedge (f; g)) = ((\Box(\text{init } w) \wedge f); (\Box(\text{init } w) \wedge g))$

proof –

have 1: $\vdash \Box(\text{init } w) = \text{ba}(\Box(\text{init } w))$
by (rule *BoxStateEqvBaBoxState*)
have 2: $\vdash \text{ba}(\Box(\text{init } w)) \wedge (f; g) \longrightarrow (\Box(\text{init } w) \wedge f); (\Box(\text{init } w) \wedge g)$
by (rule *BaAndChopImport*)
have 3: $\vdash \Box(\text{init } w) \wedge (f; g) \longrightarrow (\Box(\text{init } w) \wedge f); (\Box(\text{init } w) \wedge g)$
using 1 2 **by** fastforce
have 11: $\vdash (\Box(\text{init } w) \wedge f); (\Box(\text{init } w) \wedge g) \longrightarrow (\Box(\text{init } w)); (\Box(\text{init } w) \wedge g)$
by (rule *AndChopA*)
have 12: $\vdash (\Box(\text{init } w)); (\Box(\text{init } w) \wedge g) \longrightarrow (\Box(\text{init } w)); (\Box(\text{init } w))$
by (rule *ChopAndA*)
have 13: $\vdash (\Box(\text{init } w)); (\Box(\text{init } w)) = \Box(\text{init } w)$

by (rule *BoxStateChopBoxEqvBox*)
have 14: $\vdash (\Box (\text{init } w) \wedge f); (\Box (\text{init } w) \wedge g) \longrightarrow f; (\Box (\text{init } w) \wedge g)$
by (rule *AndChopB*)
have 15: $\vdash f; (\Box (\text{init } w) \wedge g) \longrightarrow f; g$
by (rule *ChopAndB*)
have 16: $\vdash (\Box (\text{init } w) \wedge f); (\Box (\text{init } w) \wedge g) \longrightarrow \Box (\text{init } w) \wedge (f; g)$
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: bi-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: next-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: $\vdash di(\circ (init\ w)) = \circ (di\ (init\ w))$ **by** (rule DiNext)
have 2: $\vdash di\ (init\ w) = (init\ w)$ **by** (rule DiState)
hence 3: $\vdash \circ (di\ (init\ w)) = \circ (init\ w)$ **by** (rule NextEqvNext)
from 1 3 **show** ?thesis **by** fastforce
qed

lemma StatImpBiGen:
assumes $\vdash (init\ w) \longrightarrow f$
shows $\vdash (init\ w) \longrightarrow bi\ f$
proof –
have 1: $\vdash (init\ w) \longrightarrow f$ **using** assms **by** auto
hence 2: $\vdash \neg f \longrightarrow \neg (init\ w)$ **by** auto
hence 3: $\vdash di\ (\neg f) \longrightarrow di\ (\neg (init\ w))$ **by** (rule DilmpDi)
hence 4: $\vdash di\ (\neg f) \longrightarrow di\ (init\ (\neg w))$ **by** (metis Initprop(2) inteq-reflection)
have 5: $\vdash di\ (init\ (\neg w)) = (init\ (\neg w))$ **by** (rule DiState)
have 6: $\vdash di\ (\neg f) \longrightarrow \neg (init\ w)$ **using** 4 5 **using** Initprop(2) **by** fastforce
hence 7: $\vdash (init\ w) \longrightarrow \neg (di\ (\neg f))$ **by** auto
from 7 **show** ?thesis **by** (simp add: bi-d-def)
qed

lemma ChopAndNotChopImp:
 $\vdash f; g \wedge \neg (f; g1) \longrightarrow f; (g \wedge \neg g1)$
proof –
have 1: $\vdash g \longrightarrow (g \wedge \neg g1) \vee g1$ **by** auto
hence 2: $\vdash f; g \longrightarrow f; ((g \wedge \neg g1) \vee g1)$ **by** (rule RightChopImpChop)
have 3: $\vdash f; ((g \wedge \neg g1) \vee g1) \longrightarrow (f; (g \wedge \neg g1)) \vee (f; g1)$ **by** (rule ChopOrImp)
have 4: $\vdash f; g \longrightarrow f; (g \wedge \neg g1) \vee f; g1$ **using** 2 3 MP **by** fastforce
from 4 **show** ?thesis **by** auto
qed

lemma ChopAndYieldsImp:
 $\vdash f; g \wedge f\ yields\ g1 \longrightarrow f; (g \wedge g1)$
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:
 $\vdash f; g \wedge f\ yields\ (g \longrightarrow g1) \longrightarrow f; g1$
proof –
have 1: $\vdash f; g \wedge f\ yields\ (g \longrightarrow g1) \longrightarrow f; (g \wedge (g \longrightarrow g1))$ **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

8.6 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 (*metis* *BoxImpBoxBox* *fin-d-def*)

lemma *FinAndChopImport*:

$\vdash (\text{fin } w) \wedge (f;g) \longrightarrow f;((\text{fin } w) \wedge g)$

proof —

have 1: $\vdash \text{fin } w \longrightarrow \Box(\text{fin } w)$ **by** (*rule* *FinImpBox*)

hence 2: $\vdash \text{fin } w \wedge f;g \longrightarrow \Box(\text{fin } w) \wedge (f;g)$ **by** *auto*

have 3: $\vdash \Box(\text{fin } w) \wedge (f;g) \longrightarrow f;((\text{fin } w) \wedge g)$ **using** *BoxAndChopImport* **by** *blast*

from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*

qed

lemma *FinAndChop*:

$\vdash (f;(g \wedge \text{fin } w)) = (\text{fin } w \wedge f;g)$

using *FinAndChopImport* *ChopFinExportA* *ChopAndA* *ChopAndCommute* **by** *fastforce*

lemma *ChopAndEmptyEqvEmptyChopEmpty*:

$\vdash ((f;g) \wedge \text{empty}) = (f \wedge \text{empty});(g \wedge \text{empty})$

by (*auto simp: empty-defs chop-defs*)

lemma *FinAndEmpty*:

$\vdash ((\text{fin } w) \wedge \text{empty}) = (w \wedge \text{empty})$

proof —

have 1: $\vdash ((\text{fin } w) \wedge \text{empty}) = (\# \text{True};(w \wedge \text{empty}) \wedge \text{empty})$

using *FinEqvTrueChopAndEmpty* **by** *fastforce*

have 2: $\vdash (\# \text{True};(w \wedge \text{empty}) \wedge \text{empty}) = ((\# \text{True} \wedge \text{empty});(w \wedge \text{empty}))$

using *ChopAndEmptyEqvEmptyChopEmpty*

by (*smt int-eq int-iffD2 lift-and-com Prop10 Prop12*)

have 3: $\vdash (\# \text{True} \wedge \text{empty});(w \wedge \text{empty}) = (\text{empty};(w \wedge \text{empty}))$

using *LeftChopEqvChop* **by** *fastforce*

have 4: $\vdash (\text{empty};(w \wedge \text{empty})) = (w \wedge \text{empty})$

using *EmptyChop* **by** *blast*

from 1 2 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *AndFinEqvChopAndEmpty*:

$\vdash (f \wedge \text{fin } g) = f;(g \wedge \text{empty})$

proof —

have 1: $\vdash (f \wedge \text{fin } g) = (f;\text{empty} \wedge \text{fin } g)$

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* *inteq-reflection* *lift-and-com*)
qed

lemma *AndFinEqvChopStateAndEmpty*:
 $\vdash (f \wedge fin\ (init\ w)) = f; ((init\ w) \wedge empty)$
using *AndFinEqvChopAndEmpty* **by** *blast*

lemma *FinStateEqvStateAndEmptyOrNextFinState*:
 $\vdash fin\ (init\ w) = (((init\ w) \wedge empty) \vee \circ (fin\ (init\ w)))$
proof –
have 1: $\vdash fin\ (init\ w) = \Box (empty \longrightarrow init\ w)$
by (*simp* *add: fin-d-def*)
have 2: $\vdash \Box (empty \longrightarrow init\ w) =$
 $((empty \longrightarrow init\ w) \wedge wnext\ (\Box (empty \longrightarrow init\ w)))$
by (*rule* *BoxEqvAndWnextBox*)
have 3: $\vdash fin\ (init\ w) = ((empty \longrightarrow init\ w) \wedge wnext\ (fin\ (init\ w)))$
using 1 2 **by** (*simp* *add: fin-d-def*)
have 4: $\vdash wnext\ (fin\ (init\ w)) = (empty \vee \circ (fin\ (init\ w)))$
by (*rule* *WnextEqvEmptyOrNext*)
have 5: $\vdash fin\ (init\ w) = ((empty \longrightarrow init\ w) \wedge (empty \vee \circ (fin\ (init\ w))))$
using 3 4 **by** *fastforce*
have 6: $\vdash ((empty \longrightarrow init\ w) \wedge (empty \vee \circ (fin\ (init\ w)))) =$
 $((empty \longrightarrow init\ w) \wedge empty) \vee ((empty \longrightarrow init\ w) \wedge \circ (fin\ (init\ w)))$
by *auto*
have 7: $\vdash ((empty \longrightarrow init\ w) \wedge empty) = ((init\ w) \wedge empty)$
by *auto*
have 8: $\vdash ((empty \longrightarrow init\ w) \wedge \circ (fin\ (init\ w))) = \circ (fin\ (init\ w))$
by (*metis* 1 *BoxElim* *DiamondFin* *NextDiamondImpDiamond* *int-eq* *lift-and-com* *lift-imp-trans* *Prop10*)
have 9: $\vdash (((empty \longrightarrow init\ w) \wedge empty) \vee ((empty \longrightarrow init\ w) \wedge \circ (fin\ (init\ w)))) =$
 $((init\ w) \wedge empty) \vee \circ (fin\ (init\ w))$
using 7 8 **by** *auto*
from 5 6 8 9 **show** *?thesis* **by** *fastforce*
qed

lemma *FinChopEqvOr*:
 $\vdash (fin\ (init\ w)); f = (((init\ w) \wedge f) \vee \circ ((fin\ (init\ w)); f))$
proof –
have 1: $\vdash fin\ (init\ w) = (((init\ w) \wedge empty) \vee \circ (fin\ (init\ w)))$
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: fin-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: sometimes-defs init-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 BilmpFinEqvYieldsState:

$\vdash \text{bi } (f \longrightarrow \text{fin } (\text{init } w)) = f \text{ yields } (\text{init } w)$

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 StatImpYields:

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 BilmpFinEqvYieldsState*)
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: chop-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*:

$\vdash \bigcirc f \longrightarrow \text{skip yields } f$

proof –

have 1: $\vdash \bigcirc f \longrightarrow \text{wnext } f$ **using** *WnextEqvEmptyOrNext* **by** *fastforce*

have 2: $\vdash \text{skip yields } f = \text{wnext } f$ **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: more-defs next-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: more-defs sometimes-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*:

$\vdash \text{more yields } f = \text{wnext}(\Box f)$

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*:

$\vdash di\ f = di\ (di\ f)$

proof —

have 1: $\vdash \#True = \#True; \#True$ **by** (*rule TrueEqvTrueChopTrue*)

hence 2: $\vdash f; \#True = f; (\#True; \#True)$ **by** (*rule RightChopEqvChop*)

have 3: $\vdash f; (\#True; \#True) = (f; \#True); \#True$ **by** (*rule ChopAssoc*)

have 4: $\vdash f; \#True = (f; \#True); \#True$ **using** 2 3 **by** *fastforce*

from 4 **show** *?thesis* **by** (*metis di-d-def*)

qed

lemma *BiEqvBiBi*:

$\vdash bi\ f = bi\ (bi\ f)$

proof —

have 1: $\vdash di\ (\neg f) = di\ (di\ (\neg f))$ **by** (*rule DiEqvDiDi*)

have 2: $\vdash di\ (\neg f) = (\neg (bi\ f))$ **by** (*rule DiNotEqvNotBi*)

hence 3: $\vdash di\ (di\ (\neg f)) = di\ (\neg (bi\ f))$ **by** (*rule DiEqvDi*)

have 4: $\vdash di\ (\neg f) = di\ (\neg (bi\ f))$ **using** 1 3 **by** *fastforce*

hence 5: $\vdash (\neg (di\ (\neg f))) = (\neg (di\ (\neg (bi\ f))))$ **by** *fastforce*

from 5 **show** *?thesis* **by** (*metis bi-d-def*)

qed

lemma *DiOrEqv*:

$\vdash di\ (f \vee g) = (di\ f \vee di\ g)$

proof —

have 1: $\vdash (f \vee g); \#True = (f; \#True \vee g; \#True)$ **by** (*rule OrChopEqv*)

from 1 **show** *?thesis* **by** (*simp add: di-d-def*)

qed

lemma *DiAndA*:

$\vdash di\ (f \wedge g) \longrightarrow di\ f$

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*:

$\vdash di\ (f \wedge g) \longrightarrow di\ g$

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*:

$\vdash di (f \wedge g) \longrightarrow di f \wedge di g$

proof –

have 1: $\vdash di (f \wedge g) \longrightarrow di f$ **by** (rule *DiAndA*)

have 2: $\vdash di (f \wedge g) \longrightarrow di g$ **by** (rule *DiAndB*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *DiSkipEqvMore*:

$\vdash di skip = more$

proof –

have 1: $\vdash skip ; \#True = \circ \#True$ **by** (rule *SkipChopEqvNext*)

have 2: $\vdash \circ \#True = more$ **by** (auto simp: more-d-def)

have 3: $\vdash skip ; \#True = more$ **using** 1 2 **by** fastforce

from 3 **show** ?thesis **by** (simp add: di-d-def)

qed

lemma *DiMoreEqvMore*:

$\vdash di more = more$

proof –

have 1: $\vdash di (\circ \#True) = \circ (di \#True)$ **by** (rule *DiNext*)

have 2: $\vdash \circ (di \#True) \longrightarrow more$ **by** (auto simp: next-defs di-defs more-defs)

have 3: $\vdash di (\circ \#True) \longrightarrow more$ **using** 1 2 **by** fastforce

hence 4: $\vdash di more \longrightarrow more$ **by** (simp add: more-d-def)

have 5: $\vdash more \longrightarrow di more$ **by** (rule *ImpDi*)

from 4 5 **show** ?thesis **by** fastforce

qed

lemma *DiIfEqvRule*:

assumes $\vdash f = if_i (init w) then g else h$

shows $\vdash di f = if_i (init w) then (di g) else (di h)$

proof –

have 1: $\vdash f = if_i (init w) then g else h$ **using** assms **by** auto

hence 2: $\vdash f ; \#True = if_i (init w) then (g ; \#True) else (h ; \#True)$ **by** (rule *IfChopEqvRule*)

from 2 **show** ?thesis **by** (simp add: di-d-def)

qed

lemma *DiEmpty*:

$\vdash di empty$

proof –

have 1: $\vdash \#True$ **by** auto

have 2: $\vdash empty ; \#True = \#True$ **by** (rule *EmptyChop*)

have 3: $\vdash empty ; \#True$ **using** 1 2 **by** auto

from 3 **show** ?thesis **by** (simp add: di-d-def)

qed

lemma *DaNotEqvNotBa*:

$\vdash da (\neg f) = (\neg (ba f))$

proof –
have 1: $\vdash ba\ f = (\neg (da\ (\neg f)))$ **by** (simp add: ba-d-def)
from 1 **show** ?thesis **by** fastforce
qed

lemma DaEqvDa:
assumes $\vdash f = g$
shows $\vdash da\ f = da\ g$
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** BalmpBt 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 BalmpDist:

$\vdash ba\ (f \longrightarrow g) \longrightarrow ba\ f \longrightarrow ba\ g$

proof –

have 1: $\vdash bi\ (f \longrightarrow g) \longrightarrow (bi\ f \longrightarrow bi\ g)$ **by** (rule BilmpDist)
 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: bi-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 inteq-reflection)
 have 5: $\vdash ba\ f = \Box(bi\ f)$
by (rule BaEqvBtBi)
 have 6: $\vdash ba\ g = \Box(bi\ g)$
by (rule BaEqvBtBi)
 from 1 3 4 5 6 show ?thesis **by** fastforce
 qed

lemma BalmpBaEqvBa:

$\vdash ba\ (f = g) \longrightarrow (ba\ f = ba\ g)$

proof –

have 1: $\vdash ba\ (f \longrightarrow g) \longrightarrow ba\ f \longrightarrow ba\ g$ **by** (rule BalmpDist)
 have 2: $\vdash ba\ (g \longrightarrow f) \longrightarrow ba\ g \longrightarrow ba\ f$ **by** (rule BalmpDist)
 have 3: $\vdash ba\ (f = g) = ba\ ((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** (auto simp: ba-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 BalmpBa:

assumes $\vdash f \longrightarrow g$
shows $\vdash ba\ f \longrightarrow ba\ g$
using *BaGen BalmpDist MP assms by metis*

lemma *BaEqvBa*:
assumes $\vdash f = g$
shows $\vdash ba\ f = ba\ g$
using *BaGen BalmpBaEqvBa MP assms by metis*

lemma *DalmpDa*:
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*:

$\vdash ba\ (f \longrightarrow f1) \longrightarrow f; g \longrightarrow f1; g$

proof –

have 1: $\vdash ba\ (f \longrightarrow f1) \longrightarrow bi\ (f \longrightarrow f1)$ **by** (rule *BaImpBi*)

have 2: $\vdash bi\ (f \longrightarrow f1) \longrightarrow f; g \longrightarrow f1; g$ **by** (rule *BiChopImpChop*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *BaRightChopImpChop*:

$\vdash ba\ (g \longrightarrow g1) \longrightarrow f; g \longrightarrow f; g1$

proof –

have 1: $\vdash ba\ (g \longrightarrow g1) \longrightarrow \Box(g \longrightarrow g1)$ **by** (rule *BaImpBt*)

have 2: $\vdash \Box(g \longrightarrow g1) \longrightarrow f; g \longrightarrow f; g1$ **by** (rule *BoxChopImpChop*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *ChopAndBaImport*:

$\vdash (f; f1) \wedge ba\ g \longrightarrow (f \wedge g); (f1 \wedge g)$

proof –

have 1: $\vdash ba\ g \wedge (f; f1) \longrightarrow (g \wedge f); (g \wedge f1)$ **by** (rule *BaAndChopImport*)

have 2: $\vdash (g \wedge f); (g \wedge f1) = (f \wedge g); (f1 \wedge g)$ **by** (rule *AndChopAndCommute*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *BaImpBaImpBaAnd*:

$\vdash ba\ h \longrightarrow ba(g \longrightarrow ba\ h \wedge g)$

proof –

have 1: $\vdash ba\ h \longrightarrow (g \longrightarrow ba\ h \wedge g)$ **by** *fastforce*

hence 2: $\vdash ba(ba\ h) \longrightarrow ba(g \longrightarrow ba\ h \wedge g)$ **by** (rule *BaImpBa*)

have 3: $\vdash ba\ h = ba(ba\ h)$ **by** (rule *BaEqvBaBa*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *BaChopImpChopBa*:

$\vdash ba\ f \longrightarrow g; g1 \longrightarrow g; ((ba\ f) \wedge g1)$

proof –

have 1: $\vdash ba\ f \longrightarrow ba\ (g1 \longrightarrow (ba\ f) \wedge g1)$ **by** (rule *BaImpBaImpBaAnd*)

have 2: $\vdash ba\ (g1 \longrightarrow ba\ f \wedge g1) \longrightarrow g; g1 \longrightarrow g; (ba\ f \wedge g1)$ **by** (rule *BaRightChopImpChop*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *DiNotBaImpNotBa*:

$\vdash di\ (\neg (ba\ f)) \longrightarrow \neg (ba\ f)$

proof –

have 1: $\vdash ba\ f = ba\ (ba\ f)$ **by** (rule *BaEqvBaBa*)

have 2: $\vdash ba\ (ba\ f) \longrightarrow bi\ (ba\ f)$ **by** (rule *BaImpBi*)

have 3: $\vdash ba\ f \longrightarrow bi\ (ba\ f)$ **using** 1 2 **by** fastforce

hence 4: $\vdash ba\ f \longrightarrow \neg (di\ (\neg (ba\ f)))$ **by** (simp add: bi-d-def)

from 4 **show** ?thesis **by** fastforce

qed

lemma *NotBaChopImpNotBa*:

$\vdash (\neg (ba\ f)); g \longrightarrow \neg (ba\ f)$

proof –

have 1: $\vdash (\neg (ba\ f)); g \longrightarrow di\ (\neg (ba\ f))$ **by** (rule *ChopImpDi*)

have 2: $\vdash di\ (\neg (ba\ f)) \longrightarrow \neg (ba\ f)$ **by** (rule *DiNotBaImpNotBa*)

from 1 2 **show** ?thesis **using** lift-imp-trans **by** blast

qed

lemma *DiamondFinImpFin*:

$\vdash \Diamond (fin\ f) \longrightarrow fin\ f$

proof –

have 1: $\vdash fin\ f = \#True; (f \wedge empty)$

by (rule *FinEqvTrueChopAndEmpty*)

hence 2: $\vdash \Diamond (fin\ f) = \#True; (\#True; (f \wedge empty))$

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: $\vdash f; \text{fin } (\text{init } (\neg w)) \longrightarrow \text{fin } (\text{init } (\neg w))$
by (rule ChopFinImpFin)
have 2: $\vdash \text{fin } (\text{init } (\neg w)) = (\neg (\text{fin } (\text{init } w)))$
using FinNotStateEqvNotFinState **by** blast
hence 3: $\vdash f; \text{fin } (\text{init } (\neg w)) = f; (\neg (\text{fin } (\text{init } w)))$
by (rule RightChopEqvChop)
have 4: $\vdash f; (\neg (\text{fin } (\text{init } w))) \longrightarrow \neg (\text{fin } (\text{init } w))$
using 1 2 3 **by** fastforce
hence 5: $\vdash \text{fin } (\text{init } w) \longrightarrow \neg (f; (\neg (\text{fin } (\text{init } w))))$
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*:

$\vdash ((init\ w) \longrightarrow fin\ (init\ w1)); ((init\ w1) \longrightarrow fin\ (init\ w2))$
 $\longrightarrow ((init\ w) \longrightarrow fin\ (init\ w2))$

proof –

have 1: $\vdash (init\ w) \wedge ((init\ w) \longrightarrow fin\ (init\ w1)); ((init\ w1) \longrightarrow fin\ (init\ w2))$
 \longrightarrow

$((init\ w) \wedge ((init\ w) \longrightarrow fin\ (init\ w1))); ((init\ w1) \longrightarrow fin\ (init\ w2))$

by (rule *StateAndChopImp*)

have 2: $\vdash (init\ w) \wedge ((init\ w) \longrightarrow fin\ (init\ w1)) \longrightarrow fin\ (init\ w1)$

by *auto*

have 3: $\vdash ((init\ w) \wedge ((init\ w) \longrightarrow fin\ (init\ w1))); ((init\ w1) \longrightarrow fin\ (init\ w2))$
 \longrightarrow

$(fin\ (init\ w1)); ((init\ w1) \longrightarrow fin\ (init\ w2))$

using 2 **by** (rule *LeftChopImpChop*)

have 4: $\vdash (fin\ (init\ w1)); ((init\ w1) \longrightarrow fin\ (init\ w2)) =$
 $\Diamond((init\ w1) \wedge ((init\ w1) \longrightarrow fin\ (init\ w2)))$

by (rule *FinChopEqvDiamond*)

have 41: $\vdash ((init\ w1) \wedge ((init\ w1) \longrightarrow fin\ (init\ w2))) \longrightarrow fin\ (init\ w2)$

by *auto*

have 42: $\vdash \Diamond((init\ w1) \wedge ((init\ w1) \longrightarrow fin\ (init\ w2))) \longrightarrow \Diamond(fin\ (init\ w2))$

using 41 *DiamondImpDiamond* **by** *blast*

have 5: $\vdash \Diamond(fin\ (init\ w2)) \longrightarrow fin\ (init\ w2)$

using *DiamondFinImpFin* **by** *blast*

have 6: $\vdash (init\ w) \wedge ((init\ w) \longrightarrow fin\ (init\ w1)); ((init\ w1) \longrightarrow fin\ (init\ w2))$
 $\longrightarrow fin\ (init\ w2)$

using 1 3 4 5 42 **by** *fastforce*

from 6 **show** ?thesis **by** *fastforce*

qed

lemma *ChopRule*:

assumes $\vdash (init\ w) \wedge f \longrightarrow fin\ (init\ w1)$

$\vdash (init\ w1) \wedge f1 \longrightarrow fin\ (init\ w2)$

shows $\vdash (init\ w) \wedge (f; f1) \longrightarrow fin\ (init\ w2)$

proof –

have 1: $\vdash (init\ w) \wedge (f; f1) \longrightarrow ((init\ w) \wedge f); f1$ **by** (rule *StateAndChopImp*)

have 2: $\vdash (init\ w) \wedge f \longrightarrow fin\ (init\ w1)$ **using** *assms* **by** *auto*

hence 3: $\vdash ((init\ w) \wedge f); f1 \longrightarrow (fin\ (init\ w1)); f1$ **by** (rule *LeftChopImpChop*)

have 4: $\vdash (fin\ (init\ w1)); f1 = \Diamond((init\ w1) \wedge f1)$ **by** (rule *FinChopEqvDiamond*)

have 5: $\vdash (init\ w1) \wedge f1 \longrightarrow fin\ (init\ w2)$ **using** *assms* **by** *auto*

hence 6: $\vdash \Diamond((init\ w1) \wedge f1) \longrightarrow \Diamond(fin\ (init\ w2))$ **by** (rule *DiamondImpDiamond*)

have 7: $\vdash \Diamond(fin\ (init\ w2)) \longrightarrow fin\ (init\ w2)$ **using** *DiamondFinImpFin* **by** *blast*

from 1 3 4 6 7 **show** ?thesis **by** *fastforce*

qed

lemma *ChopRep*:

assumes $\vdash (init\ w) \wedge f \longrightarrow f1 \wedge fin\ (init\ w1)$

$\vdash (\text{init } w1) \wedge g \longrightarrow g1$
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*)
from 1 2 3 4 5 6 **show** *?thesis* **using** *ChopRep ChopRule* **by** *fastforce*
qed

lemma *TrueChopMoreEqvMore*:

$\vdash \# \text{True} ; \text{more} = \text{more}$
by (*metis ChopMoreImpMore NowImpDiamond TrueChopEqvDiamond int-eq int-iff1*)

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 $\vdash f \wedge \neg g \longrightarrow (more ; (f \wedge \neg g))$
shows $\vdash f \longrightarrow g$
proof –
have 1: $\vdash f \wedge \neg g \longrightarrow (more ; (f \wedge \neg g))$ **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 $\vdash f \longrightarrow g;f$
 $\vdash g \longrightarrow more$
shows $\vdash \neg f$
proof –
have 1: $\vdash f \longrightarrow g;f$ **using** *assms* **by** *auto*
have 2: $\vdash g \longrightarrow more$ **using** *assms* **by** *auto*
hence 3: $\vdash g;f \longrightarrow more ; f$ **by** (rule *LeftChopImpChop*)
have 4: $\vdash f \longrightarrow more ; f$ **using** 1 3 **by** *fastforce*
from 4 **show** *?thesis* **using** *MoreChopLoop* **by** *auto*
qed

lemma *ChopContra*:
assumes $\vdash f \wedge \neg g \longrightarrow h; f \wedge \neg (h; g)$
 $\vdash h \longrightarrow more$
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: $\vdash h \longrightarrow more$ **using** *assms* **by** *auto*
have 3: $\vdash h; f \wedge \neg (h; g) \longrightarrow h; (f \wedge \neg g)$ **by** (rule *ChopAndNotChopImp*)
have 4: $\vdash h; (f \wedge \neg g) \longrightarrow more ; (f \wedge \neg g)$ **using** 2 **by** (rule *LeftChopImpChop*)
have 5: $\vdash f \wedge \neg g \longrightarrow more ; (f \wedge \neg g)$ **using** 1 3 4 **by** *fastforce*
from 5 **show** *?thesis* **using** *MoreChopContra* **by** *auto*
qed

8.7 Properties of Chopstar and Chopplus

lemma *EmptyImpCS*:
 $\vdash empty \longrightarrow f^*$
proof –
have 1: $\vdash f^* = (empty \vee (f \wedge more); f^*)$ **by** (rule *ChopstarEqv*)
have 2: $\vdash empty \longrightarrow empty \vee (f \wedge more); f^*$ **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 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 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 *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 *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 *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 *CSEqvOrChopCS*:
 $\vdash ((f^*) \wedge \text{empty}) = \text{empty}$
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 *ChopCSEqvOrChopCS*:
 $\vdash ((f; f^*) \wedge \text{empty}) = (f \wedge \text{empty})$
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 *CSEqvOrChopCS* **using** *RightChopEqvChop* **by** *blast*
have 3: $\vdash (f \wedge \text{empty}); \text{empty} = (f \wedge \text{empty})$
by (rule *ChopEmpty*)
from 1 2 3 **show** ?thesis **by** *fastforce*
qed

lemma *AndMoreChopAndMoreEqvAndMoreChop*:
 $\vdash ((f \wedge \text{more}); g \wedge \text{more}) = (f \wedge \text{more}); g$
using *ChopImpDi DiAndB DiMoreEqvMore* **by** *fastforce*

lemma *ChopPlusEqv*:
 $\vdash (f; f^*) = (f \vee (f \wedge \text{more}); (f; f^*))$
proof –
have 1: $\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$
by (rule *ChopstarEqv*)
have 2: $\vdash f^* = (\text{empty} \vee f; f^*)$
by (rule *CSEqvOrChopCS*)
hence 3: $\vdash (\text{empty} \vee f; f^*) = (\text{empty} \vee (f \wedge \text{more}); f^*)$
using 1 2 **by** *fastforce*
have 4: $\vdash (f \wedge \text{more}); (f^*) = (f \wedge \text{more}); (\text{empty} \vee f; f^*)$
using 2 **using** *RightChopEqvChop* **by** *blast*
hence 5: $\vdash \text{empty} \vee f; f^* = \text{empty} \vee (f \wedge \text{more}); (\text{empty} \vee f; f^*)$
using 3 4 **by** *fastforce*
have 6: $\vdash (f \wedge \text{more}); (\text{empty} \vee f; f^*) =$
 $((f \wedge \text{more}); \text{empty} \vee (f \wedge \text{more}); (f; f^*))$

```

using ChopOrEqv by blast
have 7:  $\vdash (f \wedge \text{more}); \text{empty} = (f \wedge \text{more})$ 
using ChopEmpty by blast
have 8:  $\vdash (\text{empty} \vee f;f^*) =$ 
     $(\text{empty} \vee (f \wedge \text{more}) \vee (f \wedge \text{more}); (f;f^*))$ 
using 5 6 7 by (metis 2 3 inteq-reflection)
have 9:  $\vdash ((\text{empty} \vee f;f^*) \wedge \text{more}) = (f;f^* \wedge \text{more})$ 
by (auto simp: empty-d-def)
have 10:  $\vdash ((\text{empty} \vee (f \wedge \text{more}) \vee (f \wedge \text{more}); (f;f^*)) \wedge \text{more}) =$ 
     $((f \wedge \text{more}) \vee (f \wedge \text{more}); (f;f^*)) \wedge \text{more}$ 
by (auto simp: empty-d-def)
have 11:  $\vdash (((f \wedge \text{more}) \vee (f \wedge \text{more}); (f;f^*)) \wedge \text{more}) =$ 
     $((f \wedge \text{more}) \vee (f \wedge \text{more}); (f;f^*))$ 
using 10 6 7 int-eq
using AndMoreChopAndMoreEqvAndMoreChop by fastforce
have 12:  $\vdash (f;f^* \wedge \text{more}) = ((f \wedge \text{more}) \vee (f \wedge \text{more}); (f;f^*))$ 
using 8 9 10 11 by fastforce
have 13:  $\vdash (f;f^* \wedge \text{empty}) = (f \wedge \text{empty})$ 
by (rule ChopCSAndEmptyEqvAndEmpty)
have 14:  $\vdash ((f \wedge \text{more}) \vee (f \wedge \text{more}); (f;f^*) \vee (f \wedge \text{empty})) =$ 
     $(f \vee (f \wedge \text{more}); (f;f^*))$ 
by (auto simp: empty-d-def)
have 15:  $\vdash f;f^* = ((f;f^* \wedge \text{empty}) \vee (f;f^* \wedge \text{more}))$ 
by (auto simp: empty-d-def)
from 12 13 14 15 show ?thesis by fastforce
qed

```

lemma ChopPlusImpChopPlus:

```

assumes  $\vdash f \longrightarrow g$ 
shows  $\vdash f;f^* \longrightarrow g;g^*$ 
proof —
have 1:  $\vdash f \longrightarrow g$ 
using assms by auto
have 2:  $\vdash f;f^* = (f \vee (f \wedge \text{more}); (f;f^*))$ 
by (rule ChopPlusEqv)
have 3:  $\vdash g;g^* = (g \vee (g \wedge \text{more}); (g;g^*))$ 
by (rule ChopPlusEqv)
have 4:  $\vdash f;f^* \wedge \neg (g;g^*) \longrightarrow ((f \wedge \text{more}); (f;f^*)) \wedge \neg ((g \wedge \text{more}); (g;g^*))$ 
using 1 2 3 by fastforce
have 5:  $\vdash f \wedge \text{more} \longrightarrow g \wedge \text{more}$  using 1
by auto
have 6:  $\vdash (f \wedge \text{more}); (f;f^*) \longrightarrow (g \wedge \text{more}); (g;g^*)$ 
using 5 by (rule LeftChopImpChop)
have 7:  $\vdash f;f^* \wedge \neg (g;g^*) \longrightarrow$ 
     $((g \wedge \text{more}); (f;f^*)) \wedge \neg ((g \wedge \text{more}); (g;g^*))$ 
using 4 6 by fastforce
have 8:  $\vdash g \wedge \text{more} \longrightarrow \text{more}$ 
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: sometimes-defs empty-defs*)

lemma *InitAndEmptyEqvAndEmpty*:

$\vdash ((\text{init } w) \wedge \text{empty}) = (w \wedge \text{empty})$

proof —

have 1: $\vdash ((init\ w) \wedge empty) = ((w \wedge empty); \#True \wedge empty)$
by (*metis init-d-def int-eq lift-and-com*)
have 2: $\vdash ((w \wedge empty); \#True \wedge empty) = (w \wedge empty); (\#True \wedge empty)$
by (*meson AndChopA ChopAndA ChopAndEmptyEqvEmptyChopEmpty lift-imp-trans Prop11 Prop12*)
have 3: $\vdash (w \wedge empty); (\#True \wedge empty) = (w \wedge empty); empty$
using *RightChopEqvChop* **by** *fastforce*
have 4: $\vdash (w \wedge empty); empty = (w \wedge empty)$
using *ChopEmpty* **by** *blast*
from 1 2 3 4 **show** ?thesis **by** *fastforce*
qed

lemma *InitAndNotBoxInitImpNotEmpty*:

$\vdash init\ w \wedge \neg(\Box (init\ w)) \longrightarrow \neg empty$

proof —

have 1: $\vdash ((init\ w) \wedge empty) = (w \wedge empty)$
by (*rule InitAndEmptyEqvAndEmpty*)
have 2: $\vdash (\neg(\Box (init\ w)) \wedge empty) = (\Diamond(\neg(init\ w)) \wedge empty)$
by (*auto simp: always-d-def*)
have 3: $\vdash (\Diamond(\neg(init\ w)) \wedge empty) = (\neg(init\ w) \wedge empty)$
by (*simp add: DiamondAndEmptyEqvAndEmpty*)
have 4: $\vdash (\neg(init\ w)) = (init\ (\neg w))$ **using** *Initprop(2)* **by** *blast*
have 5: $\vdash (\neg(init\ w) \wedge empty) = (\neg w \wedge empty)$
using 4 *InitAndEmptyEqvAndEmpty* **by** (*metis inteq-reflection*)
have 6: $\vdash (\neg(\Box (init\ w)) \wedge empty) = (\neg w \wedge empty)$
using 2 3 5 **by** *fastforce*
have 7: $\vdash \neg((init\ w \wedge \neg(\Box (init\ w))) \wedge empty)$
using 1 6 **by** *fastforce*
from 7 **show** ?thesis **by** *auto*
qed

lemma *BoxImpTrueChopAndEmpty*:

$\vdash \Box f \longrightarrow \#True; (f \wedge empty)$

using *BoxAndChopImport Finprop(3)* **by** *fastforce*

lemma *BoxInitAndMoreImpBoxInitAndMoreAndFinInit*:

$\vdash \Box (init\ w) \wedge more \longrightarrow (\Box (init\ w) \wedge more) \wedge fin (init\ w)$

proof —

have 1: $\vdash fin (init\ w) = \#True ; (init\ w \wedge empty)$ **using** *FinEqvTrueChopAndEmpty* **by** *blast*
have 2: $\vdash \Box (init\ w) \longrightarrow \#True; (init\ w \wedge empty)$ **by** (*rule BoxImpTrueChopAndEmpty*)
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *CSImpBox*:

assumes $\vdash f \longrightarrow empty \vee (\Box (init\ w) \wedge more) ; f$

shows $\vdash init\ w \wedge f \longrightarrow \Box (init\ w)$

proof —

have 1: $\vdash f \longrightarrow empty \vee (\Box (init\ w) \wedge more) ; f$
using *assms* **by** *auto*
have 2: $\vdash init\ w \wedge \neg(\Box (init\ w)) \longrightarrow \neg empty$

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: $\vdash \Box(\text{init } w) \wedge \text{more} \longrightarrow (\Box(\text{init } w) \wedge \text{more}) \wedge \text{fin}(\text{init } w)$
 by (rule *BoxInitAndMoreImpBoxInitAndMoreAndFinInit*)
 hence 5: $\vdash (\Box(\text{init } w) \wedge \text{more}); f \longrightarrow ((\Box(\text{init } w) \wedge \text{more}) \wedge \text{fin}(\text{init } w)); f$
 by (rule *LeftChopImpChop*)
 have 6: $\vdash ((\Box(\text{init } w) \wedge \text{more}) \wedge \text{fin}(\text{init } w)); f =$
 $(\Box(\text{init } w) \wedge \text{more}); (\text{init } w \wedge f)$
 by (rule *AndFinChopEqvStateAndChop*)
 have 7: $\vdash \neg(\Box(\text{init } w)) \longrightarrow (\Box(\text{init } w)) \text{ yields } (\neg(\Box(\text{init } w)))$
 by (rule *NotBoxStateImpBoxYieldsNotBox*)
 have 8: $\vdash (\Box(\text{init } w)) \text{ yields } (\neg(\Box(\text{init } w))) \longrightarrow$
 $(\Box(\text{init } w) \wedge \text{more}) \text{ yields } (\neg(\Box(\text{init } w)))$
 by (rule *AndYieldsA*)
 have 9: $\vdash (\Box(\text{init } w) \wedge \text{more}); (\text{init } w \wedge f) \wedge (\Box(\text{init } w) \wedge \text{more}) \text{ yields } (\neg(\Box(\text{init } w)))$
 \longrightarrow
 $(\Box(\text{init } w) \wedge \text{more}); ((\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)))$
 by (rule *ChopAndYieldsImp*)
 have 10: $\vdash (\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)) \longrightarrow$
 $(\Box(\text{init } w) \wedge \text{more}); ((\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)))$
 using 3 5 6 7 8 9 by fastforce
 have 11: $\vdash (\Box(\text{init } w) \wedge \text{more}); ((\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w))) \longrightarrow$
 $\text{more}; ((\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)))$
 by (rule *AndChopB*)
 have 12: $\vdash (\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)) \longrightarrow$
 $\text{more}; ((\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)))$
 using 10 11 by fastforce
 from 12 show ?thesis using *MoreChopContra* by blast
 qed

lemma *BoxCSEqvBox*:

$\vdash (\text{init } w \wedge (\Box(\text{init } w))^*) = \Box(\text{init } w)$
proof –
 have 1: $\vdash (\Box(\text{init } w))^* = (\text{empty} \vee (\Box(\text{init } w) \wedge \text{more}); (\Box(\text{init } w))^*)$
 by (rule *ChopstarEqv*)
 hence 2: $\vdash (\Box(\text{init } w))^* \longrightarrow \text{empty} \vee (\Box(\text{init } w) \wedge \text{more}); (\Box(\text{init } w))^*$
 by fastforce
 hence 3: $\vdash \text{init } w \wedge (\Box(\text{init } w))^* \longrightarrow \Box(\text{init } w)$
 by (rule *CSImpBox*)
 have 11: $\vdash \Box(\text{init } w) \longrightarrow (\text{init } w)$
 using *BoxElim* by blast
 have 12: $\vdash \Box(\text{init } w) \longrightarrow (\Box(\text{init } w))^*$
 by (rule *ImpCS*)
 have 13: $\vdash \Box(\text{init } w) \longrightarrow \text{init } w \wedge (\Box(\text{init } w))^*$
 using 11 12 by fastforce
 from 3 13 show ?thesis by fastforce
 qed

lemma *BoxStateAndCSEqvCS*:

$\vdash (\Box(\text{init } w) \wedge f^*) = (\text{init } w \wedge (\Box(\text{init } w) \wedge f)^*)$
proof –
have 1: $\vdash \Box(\text{init } w) \longrightarrow \text{init } w$
using *BoxElim* **by** *blast*
have 2: $\vdash (f^* \wedge \text{more}) = (f \wedge \text{more}); f^*$
by (*rule CSAndMoreEqvAndMoreChop*)
have 3: $\vdash (\Box(\text{init } w) \wedge ((f \wedge \text{more}); f^*)) =$
 $((\Box(\text{init } w) \wedge f \wedge \text{more}); (\Box(\text{init } w) \wedge f^*))$
by (*rule BoxStateAndChopEqvChop*)
have 4: $\vdash \Box(\text{init } w) \wedge f \wedge \text{more} \longrightarrow (\Box(\text{init } w) \wedge f) \wedge \text{more}$
by *auto*
hence 5: $\vdash (\Box(\text{init } w) \wedge f \wedge \text{more}); (\Box(\text{init } w) \wedge f^*) \longrightarrow$
 $((\Box(\text{init } w) \wedge f) \wedge \text{more}); (\Box(\text{init } w) \wedge f^*)$
by (*rule LeftChopImpChop*)
have 6: $\vdash (\Box(\text{init } w) \wedge f^*) \wedge \text{more} \longrightarrow$
 $((\Box(\text{init } w) \wedge f) \wedge \text{more}); (\Box(\text{init } w) \wedge f^*)$
using 2 3 5 **by** *fastforce*
hence 7: $\vdash \Box(\text{init } w) \wedge f^* \longrightarrow (\Box(\text{init } w) \wedge f)^*$
by (*rule CSIntro*)
have 71: $\vdash \text{init } w \wedge \Box(\text{init } w) \wedge f^* \longrightarrow \text{init } w \wedge (\Box(\text{init } w) \wedge f)^*$
using 7 **by** *fastforce*
have 8: $\vdash \Box(\text{init } w) \wedge f^* \longrightarrow \text{init } w \wedge (\Box(\text{init } w) \wedge f)^*$
using 1 71 **by** *fastforce*
have 11: $\vdash (\Box(\text{init } w) \wedge f)^* \longrightarrow (\Box(\text{init } w))^*$
by (*rule AndCSA*)
have 12: $\vdash (\text{init } w \wedge (\Box(\text{init } w))^*) = \Box(\text{init } w)$
by (*rule BoxCSEqvBox*)
have 13: $\vdash (\Box(\text{init } w) \wedge f)^* \longrightarrow f^*$
by (*rule AndCSB*)
have 14: $\vdash \text{init } w \wedge (\Box(\text{init } w) \wedge f)^* \longrightarrow \text{init } w \wedge (\Box(\text{init } w))^* \wedge f^*$
using 11 13 **by** *fastforce*
have 15: $\vdash \text{init } w \wedge (\Box(\text{init } w))^* \wedge f^* \longrightarrow \Box(\text{init } w) \wedge f^*$
using 12 **by** *auto*
have 16: $\vdash \text{init } w \wedge (\Box(\text{init } w) \wedge f)^* \longrightarrow \Box(\text{init } w) \wedge f^*$
using 14 15 *lift-imp-trans* **by** *blast*
from 8 16 **show** *?thesis* **by** *fastforce*
qed

lemma *BaCSImpCS*:

$\vdash \text{ba}(f \longrightarrow g) \longrightarrow f^* \longrightarrow g^*$

proof –

have 1: $\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$
by (*rule ChopstarEqv*)
have 2: $\vdash g^* = (\text{empty} \vee (g \wedge \text{more}); g^*)$
by (*rule ChopstarEqv*)
have 21: $\vdash \neg(g^*) = (\neg \text{empty} \wedge \neg((g \wedge \text{more}); g^*))$
using 2 **by** *fastforce*
hence 22: $\vdash \neg(g^*) = (\text{more} \wedge \neg((g \wedge \text{more}); g^*))$
using *NotEmptyEqvMore* **by** *fastforce*
have 3: $\vdash f^* \wedge \neg(g^*) \longrightarrow$

$(\text{empty} \vee (f \wedge \text{more}); f^*) \wedge \text{more} \wedge \neg ((g \wedge \text{more}); g^*)$
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: ba-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)

8.8 Properties of While

lemma WhileEqvIf:

$\vdash \text{while } (\text{init } w) \text{ do } f = \text{if } (\text{init } w) \text{ then } (f; (\text{while } (\text{init } w) \text{ do } f)) \text{ else empty}$

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: $\vdash (\text{init } w \wedge (f; (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w))) =$
 $(\text{init } w \wedge (f; (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)))$

using Initprop(2) **by** (metis StateAndEmptyChop int-eq)

have 5: $\vdash ((f; ((\text{init } w \wedge f)^*)) \wedge (\text{fin } (\neg (\text{init } w)))) =$
 $(f; ((\text{init } w \wedge f)^* \wedge (\text{fin } (\neg (\text{init } w))))$

by (rule ChopAndFin)

have 51: $\vdash (f; ((\text{init } w \wedge f)^* \wedge (\text{fin } (\neg (\text{init } w)))) =$
 $(f; ((\text{init } w \wedge f)^* \wedge (\text{fin } (\neg (\text{init } w))))$

using Initprop(2) **by** (smt RightChopEqvChop int-eq lift-and-com)

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 21 22 3 4 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: $\vdash \text{if}_i (\text{init } w) \text{ then } (f; (\text{while } (\text{init } w) \text{ do } f)) \text{ else } \text{empty} =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f; \text{while } (\text{init } w) \text{ do } f)))$
 by (auto simp: ifthenelse-d-def)
 from 7 71 show ?thesis by fastforce
 qed

lemma *WhileChopEqvIf*:

$\vdash (\text{while } (\text{init } w) \text{ do } f); g = \text{if}_i (\text{init } w) \text{ then } (f; ((\text{while } (\text{init } w) \text{ do } f); g)) \text{ else } g$
proof –
 have 1: $\vdash \text{while } (\text{init } w) \text{ do } f =$
 $\text{if}_i (\text{init } w) \text{ then } (f; (\text{while } (\text{init } w) \text{ do } f)) \text{ else } \text{empty}$
 by (rule WhileEqvIf)
 hence 2: $\vdash (\text{while } (\text{init } w) \text{ do } f); g =$
 $\text{if}_i (\text{init } w) \text{ then } ((f; \text{while } (\text{init } w) \text{ do } f); g) \text{ else } (\text{empty}; g)$
 by (rule IfChopEqvRule)
 have 3: $\vdash \text{empty}; g = g$
 by (rule EmptyChop)
 have 4: $\vdash \text{if}_i (\text{init } w) \text{ then } ((f; \text{while } (\text{init } w) \text{ do } f); g) \text{ else } (\text{empty}; g) =$
 $\text{if}_i (\text{init } w) \text{ then } ((f; \text{while } (\text{init } w) \text{ do } f); g) \text{ else } g$
 using 3 using inteq-reflection by fastforce
 have 5: $\vdash ((f; \text{while } (\text{init } w) \text{ do } f); g) = (f; (\text{while } (\text{init } w) \text{ do } f; g))$
 by (rule ChopAssocB)
 have 6: $\vdash \text{if}_i (\text{init } w) \text{ then } ((f; \text{while } (\text{init } w) \text{ do } f); g) \text{ else } g =$
 $\text{if}_i (\text{init } w) \text{ then } (f; ((\text{while } (\text{init } w) \text{ do } f); g)) \text{ else } g$
 using 5 using inteq-reflection by fastforce
 from 1 2 4 6 show ?thesis by fastforce
 qed

lemma *WhileChopEqvIfRule*:

assumes $\vdash f = (\text{while } (\text{init } w) \text{ do } g); h$
 shows $\vdash f = \text{if}_i (\text{init } w) \text{ then } (g; f) \text{ else } h$
proof –
 have 1: $\vdash f = (\text{while } (\text{init } w) \text{ do } g); h$
 using assms by auto
 have 2: $\vdash (\text{while } (\text{init } w) \text{ do } g); h =$
 $\text{if}_i (\text{init } w) \text{ then } (g; ((\text{while } (\text{init } w) \text{ do } g); h)) \text{ else } h$
 by (rule WhileChopEqvIf)
 have 3: $\vdash (g; f) = (g; ((\text{while } (\text{init } w) \text{ do } g); h))$
 using 1 by (rule RightChopEqvChop)
 have 4: $\vdash (g; ((\text{while } (\text{init } w) \text{ do } g); h)) = (g; f)$
 using 3 by auto
 have 5: $\vdash \text{if}_i (\text{init } w) \text{ then } (g; ((\text{while } (\text{init } w) \text{ do } g); h)) \text{ else } h =$
 $\text{if}_i (\text{init } w) \text{ then } (g; f) \text{ else } h$
 using 4 using inteq-reflection by fastforce
 from 1 2 5 show ?thesis by fastforce
 qed

lemma *WhileImpFin*:

$\vdash \text{while } (\text{init } w) \text{ do } f \longrightarrow \text{fin } (\neg (\text{init } w))$

proof —

have 1: $\vdash (init\ w \wedge f)^* \wedge fin\ (\neg (init\ w)) \longrightarrow fin\ (\neg (init\ w))$ **by** *auto*

from 1 **show** *?thesis* **by** (*simp add: while-d-def*)

qed

lemma *WhileEqvEmptyOrChopWhile*:

$\vdash while\ (init\ w)\ do\ f = ((\neg (init\ w) \wedge empty) \vee (init\ w \wedge (f \wedge more); while\ (init\ w)\ do\ f))$

proof —

have 1: $\vdash (init\ w \wedge f)^* = (empty \vee ((init\ w \wedge f) \wedge more); (init\ w \wedge f)^*)$

by (*rule ChopstarEqv*)

have 2: $\vdash ((init\ w \wedge f) \wedge more) = (init\ w \wedge (f \wedge more))$

by *auto*

hence 3: $\vdash ((init\ w \wedge f) \wedge more); (init\ w \wedge f)^* = (init\ w \wedge f \wedge more); (init\ w \wedge f)^*$

by (*rule LeftChopEqvChop*)

have 4: $\vdash (init\ w \wedge f)^* = (empty \vee (init\ w \wedge f \wedge more); (init\ w \wedge f)^*)$

using 1 3 **by** *fastforce*

have 5: $\vdash ((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w))) =$

$((empty \wedge fin\ (\neg (init\ w))) \vee$

$((init\ w \wedge f \wedge more); (init\ w \wedge f)^* \wedge fin\ (\neg (init\ w))))$

using 1 4 **by** *fastforce*

have 6: $\vdash (empty \wedge fin\ (\neg (init\ w))) = (\neg (init\ w) \wedge empty)$

using *AndFinEqvChopAndEmpty EmptyChop* **by** (*metis int-eq*)

have 7: $\vdash (init\ w \wedge f \wedge more); (init\ w \wedge f)^* = (init\ w \wedge (f \wedge more); (init\ w \wedge f)^*)$

by (*rule StateAndChop*)

have 8: $\vdash (((f \wedge more); (init\ w \wedge f)^*) \wedge fin\ (init\ (\neg w))) =$

$((f \wedge more); ((init\ w \wedge f)^* \wedge fin\ (init\ (\neg w))))$

by (*rule ChopAndFin*)

have 81: $\vdash fin\ (init\ (\neg w)) = fin\ (\neg (init\ w))$

using *FinEqvFin Initprop(2)* **by** *fastforce*

have 82: $\vdash ((f \wedge more); (init\ w \wedge f)^* \wedge fin\ (\neg (init\ w))) =$

$((f \wedge more); ((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w))))$

using 8 81

by (*metis inteq-reflection*)

have 9: $\vdash ((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w))) =$

$((\neg (init\ w) \wedge empty) \vee$

$(init\ w \wedge (f \wedge more); ((init\ w \wedge f)^* \wedge fin\ (\neg (init\ w))))$

using 5 6 7 82 **by** *fastforce*

from 9 **show** *?thesis* **by** (*simp add: while-d-def*)

qed

lemma *WhileIntro*:

assumes $\vdash \neg (init\ w) \wedge f \longrightarrow empty$

$\vdash init\ w \wedge f \longrightarrow (g \wedge more); f$

shows $\vdash f \longrightarrow while\ (init\ w)\ do\ g$

proof —

have 1: $\vdash \neg (init\ w) \wedge f \longrightarrow empty$

using *assms* **by** *blast*

have 2: $\vdash init\ w \wedge f \longrightarrow (g \wedge more); f$

using *assms* **by** *blast*

have 3: $\vdash while\ (init\ w)\ do\ g =$

$((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g))$
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 35: $\vdash (f \wedge ((\text{init } w) \vee \text{more}) \wedge (\neg (\text{init } w) \vee \neg ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g)))) =$
 $((f \wedge (\text{init } w) \wedge \neg ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g))) \vee$
 $(f \wedge (\text{init } w) \wedge \neg (\text{init } w)) \vee$
 $(f \wedge \text{more} \wedge \neg ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g))) \vee$
 $(f \wedge \text{more} \wedge \neg (\text{init } w)))$
by auto
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 33 34 35 **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: $\vdash f \wedge \neg (\text{while } (\text{init } w) \text{ do } g) \longrightarrow$
 $((g \wedge \text{more}); f \wedge \neg ((g \wedge \text{more}); \text{while } (\text{init } w) \text{ do } g))$
using 36 40 **by** fastforce
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 $\vdash \neg (\text{init } w) \wedge \text{empty} \longrightarrow g$
 $\vdash \text{init } w \wedge (f \wedge \text{more}) \longrightarrow g$

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: $\vdash ((\text{while } (\text{init } w) \text{ do } f) \wedge \neg g) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f \wedge \text{more}); \text{while } (\text{init } w) \text{ do } f)) \wedge \neg g$
by *auto*
have 2: $\vdash \neg (\text{init } w) \wedge \text{empty} \longrightarrow g$
using *assms* **by** *blast*
hence 21: $\vdash \neg g \longrightarrow \neg (\neg (\text{init } w) \wedge \text{empty})$
by *auto*
have 22: $\vdash ((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f \wedge \text{more}); \text{while } (\text{init } w) \text{ do } f)) \wedge \neg g \longrightarrow$
 $(\text{init } w \wedge (f \wedge \text{more}); \text{while } (\text{init } w) \text{ do } f)$
using 21 **by** *auto*
have 23: $\vdash (\text{while } (\text{init } w) \text{ do } f) \wedge \neg g \longrightarrow$
 $(\text{init } w \wedge (f \wedge \text{more}); \text{while } (\text{init } w) \text{ do } f) \wedge \neg g$
using 11 21 **by** *fastforce*
have 3: $\vdash (\text{init } w) \wedge ((f \wedge \text{more}); g) \longrightarrow g$
using *assms* **by** *blast*
hence 31: $\vdash \neg g \longrightarrow \neg ((\text{init } w) \wedge ((f \wedge \text{more}); g))$
by *fastforce*
have 32: $\vdash (\text{init } w \wedge (f \wedge \text{more}); \text{while } (\text{init } w) \text{ do } f) \wedge \neg g \longrightarrow$
 $((f \wedge \text{more}); (\text{while } (\text{init } w) \text{ do } f)) \wedge \neg ((f \wedge \text{more}); g) \wedge \neg g$
using 31 **by** *auto*
have 4: $\vdash (\text{while } (\text{init } w) \text{ do } f) \wedge \neg g \longrightarrow$
 $((f \wedge \text{more}); (\text{while } (\text{init } w) \text{ do } f)) \wedge \neg ((f \wedge \text{more}); g)$
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*:

$\vdash \text{ba } (f \longrightarrow g) \longrightarrow (\text{while } (\text{init } w) \text{ do } f) \longrightarrow (\text{while } (\text{init } w) \text{ do } g)$
proof –
have 1: $\vdash (f \longrightarrow g) \longrightarrow ((\text{init } w \wedge f) \longrightarrow (\text{init } w \wedge g))$
by *auto*
hence 2: $\vdash \text{ba } (f \longrightarrow g) \longrightarrow \text{ba } ((\text{init } w \wedge f) \longrightarrow (\text{init } w \wedge g))$
by (rule *BaImpBa*)
have 3: $\vdash \text{ba } ((\text{init } w \wedge f) \longrightarrow (\text{init } w \wedge g)) \longrightarrow ((\text{init } w \wedge f)^* \longrightarrow (\text{init } w \wedge g)^*)$
by (rule *BaCSImpCS*)
have 4: $\vdash \text{ba } (f \longrightarrow g) \longrightarrow ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \longrightarrow (\text{init } w \wedge g)^* \wedge \text{fin } (\neg (\text{init } w)))$
using 2 3 **by** *fastforce*
from 4 **show** ?thesis **by** (*simp add: while-d-def*)
qed

lemma *WhileImpWhile*:

assumes $\vdash f \longrightarrow g$
shows $\vdash (\text{while } (\text{init } w) \text{ do } f) \longrightarrow (\text{while } (\text{init } w) \text{ do } g)$

```

proof —
  have 1:  $\vdash f \longrightarrow g$ 
    using assms by auto
  hence 2:  $\vdash ba(f \longrightarrow g)$ 
    by (rule BaGen)
  have 3:  $\vdash ba(f \longrightarrow g) \longrightarrow (while\ (init\ w)\ do\ f) \longrightarrow (while\ (init\ w)\ do\ g)$ 
    by (rule BaWhileImpWhile)
  from 2 3 show ?thesis using MP by blast
qed

```

8.9 Properties of Halt

lemma *WnextAndMoreEqvNext*:

```

 $\vdash (wnext\ f \wedge more) = \bigcirc f$ 
by (auto simp: wnext-defs more-defs next-defs)

```

lemma *BoxStateAndEmptyEqvStateAndEmpty*:

```

 $\vdash (\Box(empty = (init\ w)) \wedge empty) = ((init\ w) \wedge empty)$ 
by (auto simp: always-defs init-defs empty-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:  $\vdash (\Box(empty = (init\ w)) \wedge more) =$ 
   $((\Box(empty = (init\ w)) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more))$ 
  by auto
have 5:  $\vdash ((\Box(empty = (init\ w)) \wedge more) = (\neg(init\ w) \wedge more))$ 
  by (auto simp: empty-d-def)
have 6:  $\vdash (wnext(\Box(empty = (init\ w))) \wedge more) = \bigcirc(\Box(empty = (init\ w)))$ 
  using WnextAndMoreEqvNext by metis
have 7:  $\vdash (\Box(empty = (init\ w)) \wedge more) =$ 
   $((\neg(init\ w) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more))$ 
  using 4 5 by fastforce
have 8:  $\vdash ((\neg(init\ w) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)) =$ 
   $((\neg(init\ w)) \wedge (wnext(\Box(empty = (init\ w))) \wedge more))$  by auto
have 9:  $\vdash ((\neg(init\ w)) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)) =$ 
   $((\neg(init\ w)) \wedge \bigcirc(\Box(empty = (init\ w))))$  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
from 7 9 10 show ?thesis by fastforce
qed

```

lemma *HaltStateEqvIfStateThenEmptyElseNext*:

$\vdash \text{halt}(\text{init } w) = \text{if}_i(\text{init } w) \text{ then empty else } (\bigcirc(\text{halt}(\text{init } w)))$
proof –
have 1: $\vdash \text{halt}(\text{init } w) = \Box(\text{empty} = (\text{init } w))$
by (simp add: halt-d-def)
have 2: $\vdash \Box(\text{empty} = (\text{init } w)) =$
 $((\text{empty} \wedge \text{init } w) \vee (\neg(\text{init } w) \wedge \bigcirc(\Box(\text{empty} = (\text{init } w)))))$
by (rule BoxEmptyEqvStateqvEmptyAndStateOrNotStateNext)
have 21: $\vdash ((\text{empty} \wedge \text{init } w) \vee (\neg(\text{init } w) \wedge \bigcirc(\Box(\text{empty} = (\text{init } w))))) =$
 $((\text{init } w \wedge \text{empty}) \vee (\neg(\text{init } w) \wedge \bigcirc(\Box(\text{empty} = (\text{init } w)))))$
by auto
have 22: $\vdash \bigcirc(\text{halt}(\text{init } w)) = \bigcirc(\Box(\text{empty} = (\text{init } w)))$
using NextEqvNext **using** 1 **by** blast
have 3: $\vdash \text{if}_i(\text{init } w) \text{ then empty else } (\bigcirc(\text{halt}(\text{init } w))) =$
 $((\text{init } w \wedge \text{empty}) \vee (\neg(\text{init } w) \wedge \bigcirc(\text{halt}(\text{init } w))))$
by (simp add: ifthenelse-d-def)
from 1 2 21 22 3 **show** ?thesis **by** fastforce
qed

lemma HaltChopEqv:

$\vdash ((\text{halt}(\text{init } w)); f) = (\text{if}_i(\text{init } w) \text{ then } (f) \text{ else } (\bigcirc((\text{halt}(\text{init } w)); f)))$
proof –
have 1: $\vdash \text{halt}(\text{init } w) =$
 $(\text{if}_i(\text{init } w) \text{ then empty else } (\bigcirc(\text{halt}(\text{init } w))))$
by (rule HaltStateEqvIfStateThenEmptyElseNext)
hence 2: $\vdash ((\text{halt}(\text{init } w)); f) =$
 $(\text{if}_i(\text{init } w) \text{ then } (\text{empty}; f) \text{ else } (\bigcirc(\text{halt}(\text{init } w)); f))$
by (rule IfChopEqvRule)
have 3: $\vdash \text{empty}; f = f$
by (rule EmptyChop)
have 4: $\vdash \bigcirc((\text{halt}(\text{init } w)); f) = \bigcirc(\text{halt}(\text{init } w); f)$
by (rule NextChop)
from 2 3 4 **show** ?thesis **by** (metis inteq-reflection)
qed

lemma AndHaltChopImp:

$\vdash \text{init } w \wedge (\text{halt}(\text{init } w); f) \longrightarrow f$
proof –
have 1: $\vdash \text{halt}(\text{init } w); f = \text{if}_i(\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt}(\text{init } w); f))$
by (rule HaltChopEqv)
have 2: $\vdash \text{init } w \wedge \text{if}_i(\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt}(\text{init } w); f)) \longrightarrow f$
by (auto simp: ifthenelse-d-def)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma NotAndHaltChopImpNext:

$\vdash \neg(\text{init } w) \wedge (\text{halt}(\text{init } w); f) \longrightarrow \bigcirc(\text{halt}(\text{init } w); f)$
proof –
have 1: $\vdash \text{halt}(\text{init } w); f = \text{if}_i(\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt}(\text{init } w); f))$
by (rule HaltChopEqv)
have 2: $\vdash \neg(\text{init } w) \wedge \text{if}_i(\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt}(\text{init } w); f)) \longrightarrow$

$\bigcirc(\text{halt } (init\ w); f)$
by (*auto simp: ifthenelse-d-def*)
from 1 2 show ?thesis by fastforce
qed

lemma NotAndHaltChopImpSkipYields:
 $\vdash \neg (init\ w) \wedge (\text{halt } (init\ w); f) \longrightarrow skip\ yields\ (\text{halt } (init\ w); f)$
proof –
have 1: $\vdash \neg (init\ w) \wedge (\text{halt } (init\ w); f) \longrightarrow \bigcirc(\text{halt } (init\ w); f)$
by (*rule NotAndHaltChopImpNext*)
have 2: $\vdash \bigcirc(\text{halt } (init\ w); f) \longrightarrow skip\ yields\ (\text{halt } (init\ w); f)$
by (*rule NextImpSkipYields*)
from 1 2 show ?thesis by fastforce
qed

lemma TrueChopAndEmptyEqvChopAndEmpty:
 $\vdash ((\# True; (f \wedge empty)) \wedge g) = (g; (f \wedge empty))$
using AndFinEqvChopAndEmpty FinEqvTrueChopAndEmpty by (metis int-eq lift-and-com)

lemma WprevEqvEmptyOrPrev:
 $\vdash wprev\ f = (empty \vee prev\ f)$
by (auto simp: wprev-defs empty-defs prev-defs)

lemma NotChopSkipEqvMoreAndNotChopSkip:
 $\vdash (\neg f); skip = (more \wedge \neg(f; skip))$
proof –
have 1: $\vdash wprev\ f = (empty \vee prev\ f)$ **using WprevEqvEmptyOrPrev by auto**
hence 2: $\vdash (\neg(wprev\ f)) = (\neg(empty \vee prev\ f))$ **by auto**
have 3: $\vdash \neg(wprev\ f) = ((\neg f); skip)$ **by (simp add: wprev-d-def prev-d-def)**
have 31: $\vdash (empty \vee prev\ f) = (empty \vee (f; skip))$ **by (simp add: prev-d-def)**
have 32: $\vdash (empty \vee (f; skip)) = (\neg more \vee \neg \neg(f; skip))$ **by (simp add: empty-d-def)**
have 33: $\vdash (\neg more \vee \neg \neg(f; skip)) = (\neg(more \wedge \neg \neg(f; skip)))$ **by fastforce**
have 34: $\vdash (empty \vee prev\ f) = (\neg(more \wedge \neg \neg(f; skip)))$ **using 31 32 33 by (metis int-eq)**
have 4: $\vdash \neg(empty \vee prev\ f) = (more \wedge \neg(f; skip))$ **using 34 by fastforce**
from 2 3 4 show ?thesis by fastforce
qed

lemma HaltChopImpNotHaltChopNot:
 $\vdash \text{halt } (init\ w); f \longrightarrow \neg (\text{halt } (init\ w); (\neg f))$
proof –
have 1: $\vdash \text{halt } (init\ w); f = if_i\ (init\ w)\ then\ f\ else\ (\bigcirc(\text{halt } (init\ w); f))$
by (rule HaltChopEqv)
have 2: $\vdash if_i\ (init\ w)\ then\ f\ else\ (\bigcirc(\text{halt } (init\ w); f)) \longrightarrow$
 $(((init\ w) \longrightarrow f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(\text{halt } (init\ w); f))))$
by (rule IfThenElseImp)
have 3: $\vdash \text{halt } (init\ w); (\neg f) =$
 $if_i\ (init\ w)\ then\ (\neg f)\ else\ (\bigcirc(\text{halt } (init\ w); (\neg f)))$
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:  $\vdash (\text{halt } (\text{init } w)); f \wedge (\Box (\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)) \longrightarrow$ 
     $\bigcirc((\text{halt } (\text{init } w)); f) \wedge \bigcirc((\Box (\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)))$ 
    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:  $\vdash (\text{halt } (\text{init } w)); f \wedge (\Box (\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)) \longrightarrow$ 
     $\bigcirc(((\text{halt } (\text{init } w)); f) \wedge ((\Box (\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f))))$ 
    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

```

8.10 Properties of Groups of chops

```

lemma NestedChopImpChop:
  assumes  $\vdash \text{init } w \wedge f \longrightarrow g; (\text{init } w1 \wedge f1)$ 
     $\vdash \text{init } w1 \wedge f1 \longrightarrow g1; (\text{init } w2 \wedge f2)$ 
  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

9 Infinite ITL theorems using Weak Chop

```

theory InfiniteTheorems
  imports
    InfiniteITL
begin

```

We give the proofs of a list of Infinite ITL theorems. These proofs and theorems are from [6] but adapted for infinite and finite intervals.

9.1 Propositional reasoning

This is a list of propositional logic theorems used in the proofs of the ITL theorems.

lemma *IfThenElseImp*:

$\vdash (if_i \ g \ \text{then} \ f \ \text{else} \ f1) \longrightarrow ((g \longrightarrow f) \wedge (\neg g \longrightarrow f1))$
by (*simp add: ifthenelse-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*)

lemma Prop13:
assumes $\vdash f \longrightarrow g \vee h$
shows $\vdash f \wedge \neg h \longrightarrow g$
using *assms* **by** (*auto simp: Valid-def*)

9.2 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: init-defs sum.case-eq-if*)

lemma Finprop :
 $\vdash ((\# \text{True}; (f \wedge \text{empty})) \wedge (\# \text{True}; (g \wedge \text{empty}))) = (\# \text{True}; ((f \wedge g) \wedge \text{empty}))$
 $\vdash ((\# \text{True}; (f \wedge \text{empty})) \vee (\# \text{True}; (g \wedge \text{empty}))) = (\# \text{True}; ((f \vee g) \wedge \text{empty}))$
 $\vdash (\# \text{True}; ((\# \text{True}) \wedge \text{empty}))$
 $\vdash \text{finite} \longrightarrow (\neg (\# \text{True}; (f \wedge \text{empty}))) = (\# \text{True}; (\neg f \wedge \text{empty}))$
 $\vdash (\neg (\# \text{True}; (f \wedge \text{empty}))) = ((\# \text{True}; (\neg f \wedge \text{empty})) \wedge \text{finite})$
by (*auto simp: finalt-defs finite-defs sum.case-eq-if*,
simp add: chop-defs finite-defs empty-defs interval-suffix-length sum.case-eq-if, auto)

9.3 finite and inf properties

lemma EmptyImpFinite:
 $\vdash \text{empty} \longrightarrow \text{finite}$
by (*simp add: empty-defs finite-defs intl sum.case-eq-if*)

lemma SkipChopFiniteImpFinite:

$\vdash \text{skip}; \text{finite} \longrightarrow \text{finite}$
by (simp add: Valid-def finite-defs chop-defs skip-defs sum.case-eq-if)

lemma FiniteChopSkipImpFinite:
 $\vdash \text{finite}; \text{skip} \longrightarrow \text{finite}$
by (simp add: Valid-def finite-defs chop-defs skip-defs sum.case-eq-if)

lemma FiniteChopSkipEqvFiniteAndMore:
 $\vdash \text{finite}; \text{skip} = (\text{finite} \wedge \text{more})$
by (simp add: Valid-def more-defs finite-defs chop-defs skip-defs sum.case-eq-if ,
 metis Suc-lel add-diff-cancel-left' diff-diff-cancel diff-is-0-eq'
 diff-le-self interval-suffix-length-good less-Suc0 nat-neq-iff plus-1-eq-Suc)

lemma FiniteChopSkipEqvSkipChopFinite:
 $\vdash \text{finite}; \text{skip} = \text{skip}; \text{finite}$
by (simp add: Valid-def finite-defs chop-defs skip-defs sum.case-eq-if ,
 metis diff-diff-cancel diff-le-self interval-prefix-length-good interval-suffix-length-good)

lemma FiniteAndEmptyEqvEmpty:
 $\vdash (\text{finite} \wedge \text{empty}) = \text{empty}$
by (simp add: Valid-def empty-defs finite-defs chop-defs skip-defs sum.case-eq-if)

lemma FiniteChopFiniteEqvFinite:
 $\vdash \text{finite}; \text{finite} = \text{finite}$
by (simp add: Valid-def finite-defs chop-defs sum.case-eq-if , blast)

lemma InfChopInfEqvInf:
 $\vdash \text{inf}; \text{inf} = \text{inf}$
by (simp add: Valid-def infinite-defs chop-defs sum.case-eq-if)

lemma InfChopFiniteEqvInf:
 $\vdash \text{inf}; \text{finite} = \text{inf}$
by (simp add: Valid-def infinite-defs chop-defs sum.case-eq-if)

lemma FiniteChopInfEqvInf:
 $\vdash \text{finite}; \text{inf} = \text{inf}$
by (simp add: Valid-def infinite-defs finite-defs chop-defs sum.case-eq-if)

lemma InfEqvNotFinite:
 $\vdash \text{inf} = (\neg \text{finite})$
by (simp add: Valid-def infinite-defs finite-defs chop-defs sum.case-eq-if)

lemma FiniteEqvNotInf:
 $\vdash \text{finite} = (\neg \text{inf})$
by (simp add: Valid-def infinite-defs finite-defs chop-defs sum.case-eq-if)

lemma ChopTrueAndFiniteEqvAndFiniteChopFinite:
 $\vdash ((f; \# \text{True}) \wedge \text{finite}) = (f \wedge \text{finite}); \text{finite}$
by (simp add: Valid-def infinite-defs finite-defs chop-defs sum.case-eq-if)

lemma *TrueChopAndFiniteEqvAndFiniteChopFinite*:

$\vdash ((\# \text{True}; f) \wedge \text{finite}) = \text{finite}; (f \wedge \text{finite})$

by (*simp add: Valid-def infinite-defs finite-defs chop-defs sum.case-eq-if*)

lemma *FiniteChopMoreEqvMore*:

$\vdash \text{finite}; \text{more} = \text{more}$

by (*simp add: Valid-def more-defs infinite-defs finite-defs chop-defs sum.case-eq-if, auto*)

lemma *ChopAndFiniteDist*:

$\vdash ((f; g) \wedge \text{finite}) = (f \wedge \text{finite}); (g \wedge \text{finite})$

by (*simp add: Valid-def finite-defs chop-defs sum.case-eq-if*)

lemma *FiniteOrInfinite*:

$\vdash \text{finite} \vee \text{inf}$

by (*simp add: Valid-def finite-defs infinite-defs sum.case-eq-if*)

lemma *FinitImpAnd*:

assumes $\vdash \text{finite} \longrightarrow f = g$

shows $\vdash (f \wedge \text{finite}) = (g \wedge \text{finite})$

using *assms by (simp add: Valid-def finite-defs, auto)*

lemma *FmoreEqvSkipChopFinite*:

$\vdash \text{fmore} = \text{skip}; \text{finite}$

by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite
fmore-d-def inteq-reflection lift-and-com*)

lemma *FinitImp*:

$\vdash (f \wedge \text{finite} \longrightarrow g) = (f \wedge \text{finite} \longrightarrow g \wedge \text{finite})$

by (*simp add: finite-defs Valid-def*)

lemma *ChopAndInf*:

$\vdash ((f; g) \wedge \text{inf}) = (f; (g \wedge \text{inf}))$

by (*simp add: Valid-def chop-defs finite-defs infinite-defs sum.case-eq-if*)

lemma *ChopAndInfB*:

$\vdash ((f; g) \wedge \text{inf}) = ((f \wedge \text{inf}) \vee (f \wedge \text{finite}); (g \wedge \text{inf}))$

by (*simp add: Valid-def chop-defs finite-defs infinite-defs sum.case-eq-if, auto*)

lemma *MoreAndInfEqvInf*:

$\vdash (\text{more} \wedge \text{inf}) = \text{inf}$

by (*metis ChopAndInf EmptyImpFinite FiniteChopMoreEqvMore InfEqvNotFinite Prop11 Prop12 empty-d-def
finite-d-def int-simps(32) inteq-reflection*)

lemma *AndInfChopAndInfEqvAndInf*:

$\vdash (f \wedge \text{inf}); (f \wedge \text{inf}) = (f \wedge \text{inf})$

by (*simp add: Valid-def infinite-defs chop-defs sum.case-eq-if*)

lemma *AndInfChopEqvAndInf*:

$\vdash (f \wedge \text{inf});g = (f \wedge \text{inf})$

by (*simp add: Valid-def chop-defs infinite-defs sum.case-eq-if*)

lemma *AndMoreAndInfEqvAndInf*:

$\vdash ((f \wedge \text{more}) \wedge \text{inf}) = (f \wedge \text{inf})$

by (*simp add: Valid-def more-defs infinite-defs sum.case-eq-if*)

lemma *AndMoreAndFiniteEqvAndFmore*:

$\vdash ((f \wedge \text{more}) \wedge \text{finite}) = (f \wedge \text{fmore})$

by (*simp add: Valid-def more-defs fmore-defs finite-defs sum.case-eq-if*)

lemma *NotFmoreAndEmpty*:

$\vdash \neg (\text{empty} \wedge \text{fmore})$

by (*simp add: fmore-d-def empty-d-def, auto*)

lemma *NotFmoreAndInf*:

$\vdash \neg ((f \wedge \text{inf}) \wedge \text{fmore})$

by (*simp add: fmore-d-def finite-d-def infinite-d-def, auto*)

lemma *FmoreChopAnd*:

$\vdash (((f \wedge \text{more});g) \wedge \text{fmore}) = ((f \wedge \text{fmore});(g \wedge \text{finite}))$

by (*simp add: Valid-def more-defs fmore-defs chop-defs finite-defs sum.case-eq-if, auto*)

lemma *NotEmptyAndInf*:

$\vdash \neg (\text{empty} \wedge \text{inf})$

by (*simp add: Valid-def empty-defs infinite-defs sum.case-eq-if*)

lemma *OrFiniteInf*:

$\vdash f = ((f \wedge \text{finite}) \vee (f \wedge \text{inf}))$

by (*simp add: finite-defs Valid-def infinite-defs sum.case-eq-if*)

lemma *AndInfEqvChopFalse*:

$\vdash (f \wedge \text{inf}) = f; \# \text{False}$

by (*simp add: finite-defs Valid-def infinite-defs chop-defs sum.case-eq-if*)

9.4 Basic Theorems

lemma *BiChopImpChop* :

$\vdash \text{bi } (f \longrightarrow f1) \longrightarrow f;g \longrightarrow f1;g$

proof –

have 1: $\vdash g \longrightarrow g$ **by** *auto*

hence 2: $\vdash \Box (g \longrightarrow g)$ **by** (*rule BoxGen*)

have 3: $\vdash \text{bi } (f \longrightarrow f1) \wedge \Box (g \longrightarrow g) \longrightarrow f;g \longrightarrow f1;g$ **by** (*rule BiBoxChopImpChop*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *AndChopA*:

$\vdash (f \wedge f1);g \longrightarrow f;g$

proof –
have 1: $\vdash f \wedge f1 \longrightarrow f$ **by** *auto*
hence 2: $\vdash bi (f \wedge f1 \longrightarrow f)$ **by** (*rule BiGen*)
have 3: $\vdash bi (f \wedge f1 \longrightarrow f) \longrightarrow (f \wedge f1);g \longrightarrow f;g$ **by** (*rule BiChopImpChop*)
from 2 3 **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *AndChopB*:

$\vdash (f \wedge f1);g \longrightarrow f1;g$

proof –
have 1: $\vdash f \wedge f1 \longrightarrow f1$ **by** *auto*
hence 2: $\vdash bi (f \wedge f1 \longrightarrow f1)$ **by** (*rule BiGen*)
have 3: $\vdash bi (f \wedge f1 \longrightarrow f1) \longrightarrow (f \wedge f1);g \longrightarrow f1;g$ **by** (*rule BiChopImpChop*)
from 2 3 **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *NextChop*:

$\vdash (\bigcirc f);g = \bigcirc(f;g)$

proof –
have 1: $\vdash skip;(f;g) = (skip;f);g$ **by** (*rule ChopAssoc*)
show *?thesis* **by** (*metis 1 int-eq next-d-def*)
qed

lemma *BoxChopImpChop* :

$\vdash \Box (g \longrightarrow g1) \longrightarrow f;g \longrightarrow f;g1$

proof –
have 1: $\vdash g \longrightarrow g$ **by** *auto*
hence 2: $\vdash bi (g \longrightarrow g)$ **by** (*rule BiGen*)
have 3: $\vdash bi (f \longrightarrow f) \wedge \Box (g \longrightarrow g1) \longrightarrow f;g \longrightarrow f;g1$ **by** (*rule BiBoxChopImpChop*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *LeftChopImpChop*:

assumes $\vdash f \longrightarrow f1$

shows $\vdash f;g \longrightarrow f1;g$

proof –
have 1: $\vdash f \longrightarrow f1$ **using** *assms* **by** *auto*
hence 2: $\vdash bi (f \longrightarrow f1)$ **by** (*rule BiGen*)
have 3: $\vdash bi (f \longrightarrow f1) \longrightarrow f;g \longrightarrow f1;g$ **by** (*rule BiChopImpChop*)
from 2 3 **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *RightChopImpChop*:

assumes $\vdash g \longrightarrow g1$

shows $\vdash f;g \longrightarrow f;g1$

proof –
have 1: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*
hence 2: $\vdash \Box (g \longrightarrow g1)$ **by** (*rule BoxGen*)
have 3: $\vdash \Box (g \longrightarrow g1) \longrightarrow f;g \longrightarrow f;g1$ **by** (*rule BoxChopImpChop*)
from 2 3 **show** *?thesis* **using** *MP* **by** *blast*

qed

lemma *RightChopEqvChop*:

assumes $\vdash g = g1$

shows $\vdash (f;g) = (f;g1)$

proof –

have 1: $\vdash g = g1$ **using** *assms* **by** *auto*

have 2: $(\vdash g \longrightarrow g1) \implies (\vdash f;g \longrightarrow f;g1)$ **by** (*rule RightChopImpChop*)

have 3: $(\vdash g1 \longrightarrow g) \implies (\vdash f;g1 \longrightarrow f;g)$ **by** (*rule RightChopImpChop*)

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

qed

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-iff1 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-iff1)
qed

lemma *OrChopEqvRule*:
assumes $\vdash f = (f1 \vee f2)$
shows $\vdash f;g = ((f1;g) \vee (f2;g))$
proof –
have 1: $\vdash f = (f1 \vee f2)$ **using** *assms* **by** *auto*
hence 2: $\vdash f;g = ((f1 \vee f2);g)$ **by** (rule *LeftChopEqvChop*)
have 3: $\vdash (f1 \vee f2);g = (f1;g \vee f2;g)$ **by** (rule *OrChopEqv*)
from 2 3 **show** ?thesis **by** *fastforce*
qed

lemma *NextImpNext*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \bigcirc f \longrightarrow \bigcirc g$
proof –
have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence 2: $\vdash \Box (f \longrightarrow g)$ **by** (rule *BoxGen*)
have 3: $\vdash \Box (f \longrightarrow g) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **by** (rule *BoxChopImpChop*)
have 4: $\vdash (skip;f) \longrightarrow (skip;g)$ **by** (metis 2 3 MP)
from 4 **show** ?thesis **by** (metis next-d-def)
qed

lemma *ChopOrImpRule*:
assumes $\vdash g \longrightarrow g1 \vee g2$
shows $\vdash f;g \longrightarrow (f;g1) \vee (f;g2)$
proof –
have 1: $\vdash g \longrightarrow g1 \vee g2$ **using** *assms* **by** *auto*
hence 2: $\vdash f;g \longrightarrow f;(g1 \vee g2)$ **by** (rule *RightChopImpChop*)
have 3: $\vdash f;(g1 \vee g2) = (f;g1 \vee f;g2)$ **by** (rule *ChopOrEqv*)
from 2 3 **show** ?thesis **by** *fastforce*
qed

lemma *NextImpDist*:
 $\vdash \bigcirc (f \longrightarrow g) \longrightarrow \bigcirc f \longrightarrow \bigcirc g$
proof –
have 1: $\vdash (\neg (f \longrightarrow g)) = (f \wedge \neg g)$ **by** *auto*
hence 2: $\vdash skip;(\neg (f \longrightarrow g)) = skip;(f \wedge \neg g)$ **by** (rule *RightChopEqvChop*)
have 3: $\vdash f \longrightarrow g \vee (f \wedge \neg g)$ **by** *auto*
hence 4: $\vdash skip;f \longrightarrow (skip;g) \vee (skip;(f \wedge \neg g))$ **by** (rule *ChopOrImpRule*)
hence 5: $\vdash \neg (skip;(f \wedge \neg g)) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **by** *auto*
have 6: $\vdash \neg (skip;(\neg (f \longrightarrow g))) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **using** 2 5 **by** *fastforce*
hence 7: $\vdash \neg (\bigcirc(\neg (f \longrightarrow g))) \longrightarrow (\bigcirc f) \longrightarrow (\bigcirc g)$ **by** (simp add: next-d-def)
have 8: $\vdash \bigcirc(f \longrightarrow g) \longrightarrow \neg (\bigcirc(\neg (f \longrightarrow g)))$ **by** (rule *NextImpNotNextNot*)
from 7 8 **show** ?thesis **using** *lift-imp-trans* **by** *blast*
qed

lemma *FiniteChopImpDiamond*:

$\vdash (f \wedge \text{finite});g \longrightarrow \Diamond g$

proof –

have 1: $\vdash f \wedge \text{finite} \longrightarrow \text{finite}$ **by** *auto*

hence 2: $\vdash (f \wedge \text{finite});g \longrightarrow \text{finite};g$ **by** (*rule LeftChopImpChop*)

from 2 **show** *?thesis* **by** (*simp add: sometimes-d-def*)

qed

lemma *NowImpDiamond*:

$\vdash f \longrightarrow \Diamond f$

proof –

have 1: $\vdash \text{empty};f = f$ **by** (*rule EmptyChop*)

have 2: $\vdash \text{empty} \longrightarrow \text{finite}$ **by** (*rule EmptyImpFinite*)

hence 3: $\vdash \text{empty};f \longrightarrow \text{finite};f$ **by** (*rule LeftChopImpChop*)

have 4: $\vdash f \longrightarrow \text{finite};f$ **using** 1 3 **by** *fastforce*

from 4 **show** *?thesis* **by** (*simp add: sometimes-d-def*)

qed

lemma *BoxElim*:

$\vdash \Box f \longrightarrow f$

proof –

have 1: $\vdash \neg f \longrightarrow \Diamond (\neg f)$ **by** (*rule NowImpDiamond*)

hence 2: $\vdash \neg (\Diamond (\neg f)) \longrightarrow f$ **by** *auto*

from 2 **show** *?thesis* **by** (*metis always-d-def*)

qed

lemma *NextDiamondImpDiamond*:

$\vdash \bigcirc (\Diamond f) \longrightarrow \Diamond f$

proof –

have 1: $\vdash \text{skip};(\text{finite};f) = ((\text{skip};\text{finite});f)$ **by** (*rule ChopAssoc*)

hence 2: $\vdash (\text{skip};\text{finite});f = \text{skip};(\text{finite};f)$ **by** *auto*

hence 3: $\vdash (\text{skip};\text{finite});f = \bigcirc(\Diamond f)$ **by** (*simp add: next-d-def sometimes-d-def*)

have 4: $\vdash (\text{skip};\text{finite});f \longrightarrow \Diamond f$

by (*simp add: SkipChopFiniteImpFinite LeftChopImpChop sometimes-d-def*)

from 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *BoxImpNowAndWeakNext*:

$\vdash \Box f \longrightarrow (f \wedge \text{wnext } (\Box f))$

proof –

have 1: $\vdash \neg f \longrightarrow \Diamond (\neg f)$ **by** (*rule NowImpDiamond*)

hence 2: $\vdash \neg (\Diamond (\neg f)) \longrightarrow f$ **by** *auto*

hence 3: $\vdash \Box f \longrightarrow f$ **by** (*metis always-d-def*)

have 4: $\vdash \bigcirc (\Diamond (\neg f)) \longrightarrow \Diamond (\neg f)$ **by** (*rule NextDiamondImpDiamond*)

have 5: $\vdash \neg \neg (\Diamond (\neg f)) \longrightarrow \Diamond (\neg f)$ **by** *auto*

hence 6: $\vdash \bigcirc (\neg \neg (\Diamond (\neg f))) \longrightarrow \bigcirc (\Diamond (\neg f))$ **by** (*rule NextImpNext*)

have 7: $\vdash \bigcirc (\neg \neg (\Diamond (\neg f))) \longrightarrow \Diamond (\neg f)$ **using** 4 6 **by** *auto*

hence 8: $\vdash \bigcirc (\neg (\Box f)) \longrightarrow \Diamond (\neg f)$ **by** (*simp add: always-d-def*)

hence 9: $\vdash \neg (\Diamond (\neg f)) \longrightarrow \neg (\bigcirc (\neg (\Box f)))$ **by** *auto*
 hence 10: $\vdash \Box f \longrightarrow \text{wnext } (\Box f)$ **by** (*simp add: always-d-def wnext-d-def*)
 from 3 10 **show** ?thesis **by** *fastforce*
qed

lemma *BoxImpBoxRule*:

assumes $\vdash f \longrightarrow g$

shows $\vdash \Box f \longrightarrow \Box g$

proof —

have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*

hence 2: $\vdash \neg g \longrightarrow \neg f$ **by** *auto*

hence 3: $\vdash \Box (\neg g \longrightarrow \neg f)$ **by** (*rule BoxGen*)

have 4: $\vdash \Box (\neg g \longrightarrow \neg f) \longrightarrow (\text{finite}; (\neg g)) \longrightarrow (\text{finite}; (\neg f))$ **by** (*rule BoxChopImpChop*)

have 5: $\vdash (\text{finite}; (\neg g)) \longrightarrow (\text{finite}; (\neg f))$ **using** 3 4 *MP* **by** *blast*

hence 6: $\vdash \Diamond (\neg g) \longrightarrow \Diamond (\neg f)$ **by** (*simp add: sometimes-d-def*)

hence 7: $\vdash \neg (\Diamond (\neg f)) \longrightarrow \neg (\Diamond (\neg g))$ **by** *auto*

from 7 **show** ?thesis **by** (*simp add: always-d-def*)

qed

lemma *BoxImpDist*:

$\vdash \Box (f \longrightarrow g) \longrightarrow \Box f \longrightarrow \Box g$

proof —

have 1: $\vdash (f \longrightarrow g) \longrightarrow (\neg g \longrightarrow \neg f)$ **by** *auto*

hence 2: $\vdash \Box (f \longrightarrow g) \longrightarrow \Box (\neg g \longrightarrow \neg f)$ **by** (*rule BoxImpBoxRule*)

have 3: $\vdash \Box ((\neg g) \longrightarrow \neg f) \longrightarrow (\text{finite}; (\neg g)) \longrightarrow (\text{finite}; (\neg f))$

by (*rule BoxChopImpChop*)

have 4: $\vdash \Box (f \longrightarrow g) \longrightarrow (\text{finite}; (\neg g)) \longrightarrow (\text{finite}; (\neg f))$

using 2 3 *lift-imp-trans* **by** *blast*

hence 5: $\vdash \Box (f \longrightarrow g) \longrightarrow \Diamond (\neg g) \longrightarrow \Diamond (\neg f)$ **by** (*simp add: sometimes-d-def*)

hence 6: $\vdash \Box (f \longrightarrow g) \longrightarrow \neg (\Diamond (\neg f)) \longrightarrow \neg (\Diamond (\neg g))$ **by** *auto*

from 6 **show** ?thesis **by** (*simp add: always-d-def*)

qed

lemma *DiamondEmptyEqvFinite*:

$\vdash \Diamond \text{empty} = \text{finite}$

proof —

have 1: $\vdash \text{finite}; \text{empty} = \text{finite}$ **by** (*rule ChopEmpty*)

from 1 **show** ?thesis **by** (*simp add: sometimes-d-def*)

qed

lemma *NextEqvNext*:

assumes $\vdash f = g$

shows $\vdash \bigcirc f = \bigcirc g$

proof —

have 1: $\vdash f = g$ **using** *assms* **by** *auto*

hence 2: $\vdash \text{skip}; f = \text{skip}; g$ **by** (*rule RightChopEqvChop*)

from 1 **show** ?thesis **by** (*metis 2 next-d-def*)

qed

lemma *NextAndNextImpNextRule*:

assumes $\vdash (f \wedge g) \longrightarrow h$
shows $\vdash (\bigcirc f \wedge \bigcirc g) \longrightarrow \bigcirc h$
using *assms*
by (*simp add: Valid-def next-defs sum.case-eq-if*)

lemma *NextAndNextEqvNextRule*:
assumes $\vdash (f \wedge g) = h$
shows $\vdash (\bigcirc f \wedge \bigcirc g) = \bigcirc h$
using *assms* **by** (*metis NextAndNextImpNextRule Prop11 Prop12 int-eq int-simps(20)*)

lemma *WeakNextEqvWeakNext*:
assumes $\vdash f = g$
shows $\vdash \text{wnext } f = \text{wnext } g$
using *assms* **using** *inteq-reflection* **by** *force*

lemma *DiamondImpDiamond*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \Diamond f \longrightarrow \Diamond g$
using *assms* **by** (*simp add: RightChopImpChop sometimes-d-def*)

lemma *DiamondEqvDiamond*:
assumes $\vdash f = g$
shows $\vdash \Diamond f = \Diamond g$
using *assms* **using** *int-eq* **by** *force*

lemma *BoxEqvBox*:
assumes $\vdash f = g$
shows $\vdash \Box f = \Box g$
using *assms* **using** *inteq-reflection* **by** *force*

lemma *BoxAndBoxImpBoxRule*:
assumes $\vdash f \wedge g \longrightarrow h$
shows $\vdash \Box f \wedge \Box g \longrightarrow \Box h$
using *assms* **by** (*auto simp: always-defs Valid-def sum.case-eq-if*)

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 $\vdash f \longrightarrow g$
 $\vdash \text{more} \wedge f \longrightarrow \bigcirc f$
shows $\vdash f \longrightarrow \Box g$
proof –


```

have 1:  $\vdash \text{more} \wedge f \longrightarrow \bigcirc f$  using assms by auto
hence 2:  $\vdash f \longrightarrow (\text{empty} \vee \bigcirc f)$  by (auto simp: Valid-def next-defs empty-defs more-defs sum.case-eq-if)
hence 3:  $\vdash f \longrightarrow \text{wnext } f$  by (auto simp: Valid-def wnext-defs empty-defs next-defs sum.case-eq-if)
hence 4:  $\vdash \Box(f \longrightarrow \text{wnext } f)$  by (rule BoxGen)
have 5:  $\vdash (\Box(f \longrightarrow \text{wnext } f)) \wedge f \longrightarrow \Box f$  by (rule BoxInduct)
hence 6:  $\vdash (\Box(f \longrightarrow \text{wnext } f)) \longrightarrow (f \longrightarrow \Box f)$  by fastforce
have 7:  $\vdash f \longrightarrow \Box f$  using 4 6 MP by blast
have 8:  $\vdash \Box f \longrightarrow f$  by (rule BoxElim)
have 9:  $\vdash f = \Box f$  using 7 8 by fastforce
have 10:  $\vdash f \longrightarrow g$  using assms by auto
hence 11:  $\vdash \Box f \longrightarrow \Box g$  by (rule ImpBoxRule)
from 7 9 11 show ?thesis using lift-imp-trans by blast
qed

```

lemma *NextLoop*:

```

assumes  $\vdash f \longrightarrow \bigcirc f$ 
shows  $\vdash \text{finite} \longrightarrow \neg f$ 
proof –
  have 1:  $\vdash f \longrightarrow \bigcirc f$  using assms by auto
  hence 2:  $\vdash f \longrightarrow (\text{more} \wedge \text{wnext } f)$ 
  by (auto simp: Valid-def more-defs wnext-defs next-defs sum.case-eq-if)
  hence 3:  $\vdash f \longrightarrow \text{wnext } f$  by auto
  hence 4:  $\vdash \Box(f \longrightarrow \text{wnext } f)$  by (rule BoxGen)
  have 5:  $\vdash \Box(f \longrightarrow \text{wnext } f) \wedge f \longrightarrow \Box f$  by (rule BoxInduct)
  hence 6:  $\vdash \Box(f \longrightarrow \text{wnext } f) \longrightarrow (f \longrightarrow \Box f)$  by fastforce
  have 7:  $\vdash f \longrightarrow \Box f$  using 4 6 MP by blast
  have 8:  $\vdash \Box f \longrightarrow f$  by (rule BoxElim)
  have 9:  $\vdash f = \Box f$  using 7 8 by fastforce
  have 10:  $\vdash f \longrightarrow \text{more}$  using 2 by auto
  hence 11:  $\vdash \Box f \longrightarrow \Box \text{more}$  by (rule ImpBoxRule)
  have 12:  $\vdash \text{finite} = (\neg(\Box \text{more}))$ 
  by (auto simp: Valid-def finite-defs always-defs more-defs sum.case-eq-if)
  from 7 9 11 12 show ?thesis by fastforce
qed

```

lemma *WnextEqvEmptyOrNext*:

```

 $\vdash \text{wnext } f = (\text{empty} \vee \bigcirc f)$ 
by (auto simp: Valid-def empty-defs wnext-defs next-defs sum.case-eq-if)

```

lemma *NotEmptyAndNext*:

```

 $\vdash \neg(\text{empty} \wedge \bigcirc f)$ 
by (auto simp: Valid-def empty-defs next-defs sum.case-eq-if)

```

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-iff1 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: Valid-def isuffix-isuffix always-defs sum.case-eq-if*)

lemma *DiamondIntroC*:
assumes $\vdash f \longrightarrow \bigcirc g$
shows $\vdash f \longrightarrow \Diamond g$
using *assms*
by (*metis (no-types, lifting) ChopAssoc FiniteChopSkipEqvSkipChopFinite NextChop*
NextDiamondImpDiamond NowImpDiamond inteq-reflection lift-imp-trans next-d-def
sometimes-d-def)

lemma *DiamondIntro*:
assumes $\vdash (f \wedge \neg g) \longrightarrow \bigcirc f$
shows $\vdash f \wedge \text{finite} \longrightarrow \Diamond g$
proof –
have 1: $\vdash f \wedge \neg g \longrightarrow \bigcirc f$
using *assms* **by** *auto*
hence 2: $\vdash f \wedge \neg g \wedge (\Box(\neg g)) \longrightarrow (\bigcirc f) \wedge (\Box(\neg g))$
by *auto*
have 3: $\vdash (\Box(\neg g)) \longrightarrow \neg g$
by (*rule BoxElim*)
hence 4: $\vdash \Box(\neg g) = ((\Box(\neg g)) \wedge \neg g)$
using *BoxImpBoxBox BoxBoxImpBox* **by** *fastforce*
have 5: $\vdash f \wedge (\Box(\neg g)) \longrightarrow \bigcirc f \wedge \Box(\neg g)$
using 2 4 **by** *fastforce*
have 6: $\vdash \Box(\neg g) = ((\neg g) \wedge \text{wnext}(\Box(\neg g)))$
using *BoxEqvAndWnextBox* **by** *metis*

```

have 7:  $\vdash \bigcirc f \wedge \Box (\neg g) \longrightarrow \bigcirc f \wedge \text{wnext}(\Box (\neg g))$ 
  using 6 by auto
have 8:  $\vdash f \wedge (\Box (\neg g)) \longrightarrow \bigcirc f \wedge \text{wnext}(\Box (\neg g))$ 
  using 5 7 using lift-imp-trans by blast
hence 9:  $\vdash f \wedge (\Box (\neg g)) \longrightarrow \text{more} \wedge \text{wnext } f \wedge \text{wnext}(\Box (\neg g))$ 
  by (auto simp: Valid-def always-defs more-defs next-defs wnext-defs sum.case-eq-if)
hence 10:  $\vdash f \wedge (\Box (\neg g)) \longrightarrow \text{wnext } f \wedge \text{wnext}(\Box (\neg g))$ 
  by auto
hence 11:  $\vdash f \wedge (\Box (\neg g)) \longrightarrow \text{wnext } (f \wedge \Box (\neg g))$ 
  by (auto simp: Valid-def wnext-defs always-defs next-defs sum.case-eq-if)
hence 12:  $\vdash \Box (f \wedge (\Box (\neg g))) \longrightarrow \text{wnext } (f \wedge \Box (\neg g))$ 
  by (rule BoxGen)
have 13:  $\vdash \Box (f \wedge (\Box (\neg g))) \longrightarrow \text{wnext } (f \wedge \Box (\neg g)) \wedge f \wedge (\Box (\neg g)) \longrightarrow \Box (f \wedge (\Box (\neg g)))$ 
  by (rule BoxInduct)
hence 14:  $\vdash \Box (f \wedge (\Box (\neg g))) \longrightarrow \text{wnext } (f \wedge \Box (\neg g)) \longrightarrow ((f \wedge (\Box (\neg g))) \longrightarrow \Box (f \wedge (\Box (\neg g))))$ 
  by fastforce
have 15:  $\vdash ((f \wedge (\Box (\neg g))) \longrightarrow \Box (f \wedge (\Box (\neg g))))$ 
  using 12 14 MP by blast
have 16:  $\vdash \Box (f \wedge (\Box (\neg g))) \longrightarrow (f \wedge (\Box (\neg g)))$ 
  by (rule BoxElim)
have 17:  $\vdash \Box (f \wedge (\Box (\neg g))) = (f \wedge (\Box (\neg g)))$ 
  using 16 15 by fastforce
have 18:  $\vdash (f \wedge (\Box (\neg g))) \longrightarrow \text{more}$ 
  using 9 by auto
hence 19:  $\vdash \Box (f \wedge (\Box (\neg g))) \longrightarrow \Box \text{ more}$ 
  by (rule ImpBoxRule)
have 20:  $\vdash \text{finite} = (\neg(\Box \text{ more}))$ 
  by (auto simp: Valid-def finite-defs always-defs more-defs sum.case-eq-if)
have 21:  $\vdash \text{finite} \longrightarrow \neg(f \wedge (\Box (\neg g)))$ 
  using 17 19 20 by fastforce
hence 22:  $\vdash \text{finite} \longrightarrow \neg f \vee \neg(\Box (\neg g))$ 
  by auto
have 23:  $\vdash (\neg(\Box (\neg g))) = \Diamond g$ 
  by (auto simp: always-d-def)
from 22 23 show ?thesis by fastforce
qed

```

lemma *DiamondIntroB*:

```

assumes  $\vdash (f \wedge \neg g) \longrightarrow \bigcirc (f \wedge \neg g)$ 
shows  $\vdash f \wedge \text{finite} \longrightarrow \Diamond g$ 
proof -
  have 1:  $\vdash (f \wedge \neg g) \longrightarrow \bigcirc (f \wedge \neg g)$  using assms by auto
  hence 2:  $\vdash \text{finite} \longrightarrow \neg(f \wedge \neg g)$  by (rule NextLoop)
  hence 3:  $\vdash f \wedge \text{finite} \longrightarrow g$  by auto
  have 4:  $\vdash g \longrightarrow \Diamond g$  by (rule NowImpDiamond)
  from 3 4 show ?thesis using lift-imp-trans by blast
qed

```

lemma *NextContra* :

assumes $\vdash (f \wedge \neg g) \longrightarrow (\bigcirc f \wedge \neg(\bigcirc g))$
shows $\vdash f \wedge \text{finite} \longrightarrow g$
proof –
have 1: $\vdash (f \wedge \neg g) \longrightarrow (\bigcirc f \wedge \neg(\bigcirc g))$ **using** *assms* **by** *auto*
hence 2: $\vdash \neg(f \longrightarrow g) \longrightarrow \bigcirc(\neg(f \longrightarrow g))$ **by** (*auto simp: next-defs Valid-def sum.case-eq-if*)
hence 3: $\vdash \text{finite} \longrightarrow \neg \neg(f \longrightarrow g)$ **by** (*rule NextLoop*)
from 3 **show** *?thesis* **by** *auto*
qed

lemma *DiamondDiamondEqvDiamond*:

$\vdash \Diamond(\Diamond f) = \Diamond f$
proof –
have 1: $\vdash \text{finite}; \text{finite} = \text{finite}$ **by** (*simp add: FiniteChopFiniteEqvFinite*)
hence 2: $\vdash (\text{finite}; \text{finite}); f = \text{finite}; f$ **using** *LeftChopEqvChop* **by** *blast*
have 3: $\vdash (\text{finite}; \text{finite}); f = \text{finite}; (\text{finite}; f)$ **using** *ChopAssoc* **by** *fastforce*
from 2 3 **show** *?thesis* **by** (*metis inteq-reflection sometimes-d-def*)
qed

lemma *WeakNextDiamondInduct*:

assumes $\vdash \text{wnext } (\Diamond f) \longrightarrow f$
shows $\vdash \text{finite} \longrightarrow f$
proof –
have 1: $\vdash \text{wnext } (\Diamond f) \longrightarrow f$ **using** *assms* **by** *blast*
hence 2: $\vdash \neg f \longrightarrow \neg(\text{wnext } (\Diamond f))$ **by** *fastforce*
hence 3: $\vdash \neg f \longrightarrow \bigcirc(\neg(\Diamond f))$ **by** (*simp add: wnext-d-def*)
have 4: $\vdash f \longrightarrow \Diamond f$ **by** (*rule NowImpDiamond*)
hence 5: $\vdash \neg(\Diamond f) \longrightarrow \neg f$ **by** *auto*
have 6: $\vdash \neg f \longrightarrow \bigcirc(\neg f)$ **using** 3 5 **using** *NextImpNext lift-imp-trans* **by** *blast*
hence 7: $\vdash \text{finite} \longrightarrow \neg \neg f$ **by** (*rule NextLoop*)
from 7 **show** *?thesis* **by** *auto*
qed

lemma *EmptyNextInducta*:

assumes $\vdash \text{empty} \longrightarrow f$
 $\vdash \bigcirc f \longrightarrow f$
shows $\vdash \text{finite} \longrightarrow f$
proof –
have 1: $\vdash \text{empty} \longrightarrow f$ **using** *assms* **by** *auto*
have 2: $\vdash \bigcirc f \longrightarrow f$ **using** *assms* **by** *blast*
have 3: $\vdash (\text{empty} \vee \bigcirc f) \longrightarrow f$ **using** 1 2 **by** *fastforce*
have 4: $\vdash \text{wnext } f = (\text{empty} \vee \bigcirc f)$ **by** (*rule WnextEqvEmptyOrNext*)
hence 5: $\vdash \text{wnext } f \longrightarrow f$ **using** 3 **by** *fastforce*
hence 6: $\vdash \neg f \longrightarrow \neg(\text{wnext } f)$ **by** *auto*
hence 7: $\vdash \neg f \longrightarrow \bigcirc(\neg f)$ **by** (*auto simp: wnext-d-def*)
hence 8: $\vdash \text{finite} \longrightarrow \neg \neg f$ **by** (*rule NextLoop*)
from 8 **show** *?thesis* **by** *auto*
qed

lemma *EmptyNextInductb*:

assumes $\vdash \text{empty} \wedge f \longrightarrow g$

$\vdash \bigcirc(f \longrightarrow g) \wedge f \longrightarrow g$
shows $\vdash f \wedge \text{finite} \longrightarrow g$
proof –
have 1: $\vdash \text{empty} \wedge f \longrightarrow g$ **using** *assms* **by** *auto*
have 2: $\vdash \bigcirc(f \longrightarrow g) \wedge f \longrightarrow g$ **using** *assms* **by** *blast*
have 3: $\vdash (\text{empty} \vee \bigcirc(f \longrightarrow g)) \wedge f \longrightarrow g$ **using** 1 2 **by** *fastforce*
hence 4: $\vdash \text{wnext}(f \longrightarrow g) \wedge f \longrightarrow g$ **using** *WnextEqvEmptyOrNext* **by** *fastforce*
hence 5: $\vdash \text{wnext}(f \longrightarrow g) \longrightarrow (f \longrightarrow g)$ **by** *fastforce*
hence 6: $\vdash \neg(f \longrightarrow g) \longrightarrow \neg(\text{wnext}(f \longrightarrow g))$ **by** *fastforce*
hence 7: $\vdash \neg(f \longrightarrow g) \longrightarrow \bigcirc(\neg(f \longrightarrow g))$ **by** (*simp add: wnext-d-def*)
hence 8: $\vdash \text{finite} \longrightarrow \neg \neg(f \longrightarrow g)$ **by** (*rule NextLoop*)
from 8 **show** *?thesis* **by** *auto*
qed

lemma *FinImpFin*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \text{fin } f \longrightarrow \text{fin } g$
using *ImpBoxRule*[of *TEMP* (*empty* $\longrightarrow f$) *TEMP* (*empty* $\longrightarrow g$)] *assms fin-d-def*
by (*smt intl intensional-rews*(3) *inteq-reflection Prop10*)

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 sum.case-eq-if*)
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*:
 $\vdash bi\ f \wedge (f1; g) \longrightarrow (f \wedge f1); g$
proof –
have 1: $\vdash f \longrightarrow (f1 \longrightarrow f \wedge f1)$ **by** *auto*
hence 2: $\vdash bi\ f \longrightarrow bi\ (f1 \longrightarrow f \wedge f1)$ **by** (rule *BiImpBiRule*)
have 3: $\vdash bi\ (f1 \longrightarrow (f \wedge f1)) \longrightarrow f1; g \longrightarrow (f \wedge f1); g$ **by** (rule *BiChopImpChop*)
from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *StateAndChopImport*:
 $\vdash (init\ w) \wedge (f; g) \longrightarrow ((init\ w) \wedge f); g$
proof –
have 1: $\vdash (init\ w) \longrightarrow bi\ (init\ w)$ **by** (rule *StatImpBi*)
hence 2: $\vdash (init\ w) \wedge (f; g) \longrightarrow bi\ (init\ w) \wedge (f; g)$ **by** *auto*
have 3: $\vdash bi\ (init\ w) \wedge (f; g) \longrightarrow ((init\ w) \wedge f); g$ **by** (rule *BiAndChopImport*)
from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*
qed

9.5 Further Properties Di and Bi

lemma *ImpDi*:
 $\vdash f \longrightarrow di\ f$
proof –
have 1: $\vdash f; empty = f$ **by** (rule *ChopEmpty*)
have 2: $\vdash empty \longrightarrow \#True$ **by** *auto*
hence 3: $\vdash f; empty \longrightarrow f; \#True$ **by** (rule *RightChopImpChop*)
have 4: $\vdash f \longrightarrow f; \#True$ **using** 1 3 **by** *fastforce*
from 4 **show** *?thesis* **by** (*simp add: di-d-def*)
qed

lemma *DiState*:
 $\vdash di\ (init\ w) = (init\ w)$
proof –
have 0: $\vdash (init\ (\neg w)) \longrightarrow bi\ (init\ (\neg w))$ **using** *StatImpBi* **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:  $\vdash (\neg (\text{init } w) \longrightarrow \neg (\text{di } (\neg \neg (\text{init } w)))) \longrightarrow$ 
   $(\text{di } (\neg \neg (\text{init } w)) \longrightarrow (\text{init } w))$  by auto
have 4:  $\vdash \text{di } (\neg \neg (\text{init } w)) \longrightarrow (\text{init } w)$  using 2 3 MP by blast
have 5:  $\vdash (\text{init } w) \longrightarrow \neg \neg (\text{init } w)$  by auto
hence 6:  $\vdash \text{di } (\text{init } w) \longrightarrow \text{di } (\neg \neg (\text{init } w))$  by (rule DImpDi)
have 7:  $\vdash \text{di } (\text{init } w) \longrightarrow (\text{init } w)$  using 6 4 using lift-imp-trans by metis
have 8:  $\vdash (\text{init } w) \longrightarrow \text{di } (\text{init } w)$  by (rule ImpDi)
from 7 8 show ?thesis by fastforce
qed

```

```

lemma StateChop:
 $\vdash (\text{init } w); f \longrightarrow (\text{init } w)$ 
using DiState
by (auto simp: di-defs init-defs chop-defs sum.case-eq-if,
  smt conc-def conc-iprefix-isuffix interval-intlen-gr-zero interval-nth-zero-intfirst iprefix-0)

```

```

lemma StateChopExportA:
 $\vdash ((\text{init } w) \wedge f); g \longrightarrow (\text{init } w)$ 
using DiState
by (auto simp: init-defs chop-defs sum.case-eq-if,
  smt conc-def conc-iprefix-isuffix interval-intlen-gr-zero interval-nth-zero-intfirst iprefix-0)

```

```

lemma StateAndChop:
 $\vdash ((\text{init } w) \wedge f); g = ((\text{init } w) \wedge (f; g))$ 
by (simp add: AndChopB StateAndChopImport StateChopExportA Prop11 Prop12)

```

```

lemma StateAndChopImpChopRule:
assumes  $\vdash (\text{init } w) \wedge f \longrightarrow f1$ 
shows  $\vdash (\text{init } w) \wedge (f; g) \longrightarrow (f1; g)$ 
proof –
have 1:  $\vdash (\text{init } w) \wedge f \longrightarrow f1$  using assms by auto
hence 2:  $\vdash ((\text{init } w) \wedge f); g \longrightarrow f1; g$  by (rule LeftChopImpChop)
have 3:  $\vdash ((\text{init } w) \wedge f); g = ((\text{init } w) \wedge (f; g))$  by (rule StateAndChop)
from 2 3 show ?thesis by fastforce
qed

```

```

lemma StateImpChopEqvChop :
assumes  $\vdash (\text{init } w) \longrightarrow (f = f1)$ 
shows  $\vdash (\text{init } w) \longrightarrow ((f; g) = (f1; g))$ 
proof –
have 1:  $\vdash (\text{init } w) \longrightarrow (f = f1)$  using assms by auto
hence 2:  $\vdash (\text{init } w) \wedge f \longrightarrow f1$  by auto
hence 3:  $\vdash (\text{init } w) \wedge (f; g) \longrightarrow (f1; g)$  by (rule StateAndChopImpChopRule)
have 4:  $\vdash (\text{init } w) \wedge f1 \longrightarrow f$  using 1 by auto
hence 5:  $\vdash (\text{init } w) \wedge (f1; g) \longrightarrow (f; g)$  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 *DilIntro*:
 $\vdash f \longrightarrow \text{di } f$
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 DilIntro*)
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 BilmpDilmpDi*)
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 *BilmpDist*:
 $\vdash \text{bi } (f \longrightarrow g) \longrightarrow (\text{bi } f) \longrightarrow (\text{bi } g)$
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: ifthenelse-d-def init-defs Valid-def sum.case-eq-if*)

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: ifthenelse-d-def init-defs Valid-def sum.case-eq-if*)

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: init-defs empty-defs always-defs Valid-def sum.case-eq-if*)

lemma *BoxEqvAndBox*:

$\vdash \Box f = (f \wedge \Box f)$

by (*simp add: always-defs Valid-def sum.case-eq-if*,
metis (no-types, lifting) interval-intlen-gr-zero interval-suffix-zero
isl-def isuffix-0 projl-def sum.case(1) sum.case-eq-if surjective-sum)

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 *BoxStateChopBoxAndInflmpBox*:

$\vdash \Box(\text{init } w); \Box(\text{init } w) \wedge \text{inf} \longrightarrow \Box(\text{init } w)$

by (*simp add: Valid-def always-defs chop-defs init-defs sum.case-eq-if infinite-defs*
iprefix-length iprefix-0,
metis add.right-neutral interval-nth-suffix interval-nth-zero-intfirst
iprefix-length iprefix-nth isuffix-def le0 le-cases le-iff-add)

lemma *BoxStateChopBoxEqvBox*:

$\vdash \Box(\text{init } w); \Box(\text{init } w) = \Box(\text{init } w)$

proof –

have 1: $\vdash (\Box(\text{init } w)) = ((\text{init } w) \wedge (\text{empty} \vee \bigcirc(\Box(\text{init } w))))$

by (*rule BoxEqvAndEmptyOrNextBox*)

hence 2: $\vdash (\Box(\text{init } w); \Box(\text{init } w)) =$
 $((\text{init } w) \wedge (\text{empty} \vee \bigcirc(\Box(\text{init } w))));$ $\Box(\text{init } w))$

by (*metis StateAndChop inteq-reflection*)

have 3: $\vdash ((\text{empty} \vee \bigcirc(\Box(\text{init } w))));$ $\Box(\text{init } w)) =$
 $(\Box(\text{init } w) \vee \bigcirc(\Box(\text{init } w); \Box(\text{init } w)))$

by (*rule EmptyOrNextChopEqv*)

have 4: $\vdash (\Box(\text{init } w); \Box(\text{init } w)) =$
 $((\text{init } w) \wedge (\Box(\text{init } w) \vee \bigcirc(\Box(\text{init } w); \Box(\text{init } w))))$

using 2 3 **by** *fastforce*

have 5: $\vdash \neg(\Box(\text{init } w)) \longrightarrow \neg(\text{init } w) \vee \neg(\bigcirc(\Box(\text{init } w)))$

by (*rule NotBoxImpNotOrNotNextBox*)

have 6: $\vdash (\Box(\text{init } w); \Box(\text{init } w)) \wedge \neg(\Box(\text{init } w)) \longrightarrow$
 $\bigcirc(\Box(\text{init } w); \Box(\text{init } w)) \wedge \neg(\bigcirc(\Box(\text{init } w)))$

using 4 5 **by** *fastforce*

hence 7: $\vdash \Box(\text{init } w); \Box(\text{init } w) \wedge \text{finite} \longrightarrow \Box(\text{init } w)$

by (*rule NextContra*)

have 8: $\vdash \Box(\text{init } w); \Box(\text{init } w) \wedge \text{inf} \longrightarrow \Box(\text{init } w)$

by (*rule BoxStateChopBoxAndInflmpBox*)

have 9: $\vdash \Box(\text{init } w); \Box(\text{init } w) \wedge (\text{finite} \vee \text{inf}) \longrightarrow \Box(\text{init } w)$

using 7 8 by fastforce
 hence 10: $\vdash \Box (init\ w); \Box (init\ w) \longrightarrow \Box (init\ w)$
 using FiniteOrInfinite by fastforce
 have 11: $\vdash \Box (init\ w) = ((init\ w) \wedge \Box (init\ w))$
 by (rule BoxEqvAndBox)
 have 12: $\vdash empty ; \Box (init\ w) = \Box (init\ w)$
 by (rule EmptyChop)
 have 13: $\vdash ((init\ w) \wedge empty) ; \Box (init\ w) = ((init\ w) \wedge (empty ; \Box (init\ w)))$
 by (rule StateAndChop)
 have 14: $\vdash \Box (init\ w) = ((init\ w) \wedge empty) ; \Box (init\ w)$
 using 11 12 13 by fastforce
 have 15: $\vdash (init\ w) \wedge empty \longrightarrow \Box (init\ w)$
 by (rule StateAndEmptyImpBoxState)
 hence 16: $\vdash ((init\ w) \wedge empty) ; \Box (init\ w) \longrightarrow \Box (init\ w); \Box (init\ w)$
 by (rule LeftChopImpChop)
 have 17: $\vdash \Box (init\ w) \longrightarrow \Box (init\ w); \Box (init\ w)$
 using 14 16 by fastforce
 from 10 17 show ?thesis by fastforce
 qed

lemma NotBoxStateImpBoxYieldsNotBox:

$\vdash \neg(\Box (init\ w)) \longrightarrow (\Box (init\ w))\ yields\ (\neg(\Box (init\ w)))$

proof –

have 1: $\vdash \Box (init\ w); \Box (init\ w) = \Box (init\ w)$ by (rule BoxStateChopBoxEqvBox)
 have 2: $\vdash \Box (init\ w) = (\neg \neg(\Box (init\ w)))$ by auto
 hence 3: $\vdash \Box (init\ w); \Box (init\ w) = \Box (init\ w); (\neg \neg(\Box (init\ w)))$ by (rule RightChopEqvChop)
 have 4: $\vdash \neg(\Box (init\ w)) \longrightarrow \neg(\Box (init\ w); (\neg \neg(\Box (init\ w))))$ using 1 3 by auto
 from 4 show ?thesis by (simp add: yields-d-def)
 qed

lemma StateEqvBi:

$\vdash (init\ w) = bi\ (init\ w)$

proof –

have 1: $\vdash (init\ w) \longrightarrow bi\ (init\ w)$ by (rule StateImpBi)
 have 2: $\vdash bi\ (init\ w) \longrightarrow (init\ w)$ by (rule BiElim)
 from 1 2 show ?thesis by fastforce
 qed

lemma FiniteChopEqvDiamond:

$\vdash finite; f = \Diamond f$

by (simp add: sometimes-d-def)

9.6 Properties of Da and Ba

lemma DaEqvDtDi:

$\vdash da\ f = \Diamond (di\ f)$

proof –

have 1: $\vdash finite; (f; \#True) = finite; (f; \#True)$ by auto
 hence 2: $\vdash finite; (f; \#True) = finite; di\ f$ by (simp add: di-d-def)

have 3: $\vdash \text{finite}; di\ f = \Diamond (di\ f)$ **by** (rule FiniteChopEqvDiamond)
have 4: $\vdash \text{finite}; (f; \#True) = \Diamond (di\ f)$ **using** 2 3 **by** fastforce
from 4 **show** ?thesis **by** (simp add: da-d-def)
qed

lemma DaEqvDiDt:

$\vdash da\ f = di\ (\Diamond f)$

proof –

have 1: $\vdash \text{finite}; f = \Diamond f$ **by** (rule FiniteChopEqvDiamond)
hence 2: $\vdash (\text{finite}; f); \#True = (\Diamond f); \#True$ **by** (rule LeftChopEqvChop)
hence 3: $\vdash (\text{finite}; f); \#True = di\ (\Diamond f)$ **by** (simp add: di-d-def)
have 4: $\vdash \text{finite}; (f; \#True) = (\text{finite}; f); \#True$ **by** (rule ChopAssoc)
have 5: $\vdash \text{finite}; (f; \#True) = di\ (\Diamond f)$ **using** 3 4 **by** fastforce
from 5 **show** ?thesis **by** (simp add: da-d-def)

qed

lemma DtDiEqvDiDt:

$\vdash \Diamond (di\ f) = di\ (\Diamond f)$

by (metis ChopAssoc di-d-def sometimes-d-def)

lemma DiamondNotEqvNotBox:

$\vdash \Diamond (\neg f) = (\neg (\Box f))$

by (simp add: always-d-def)

lemma BaEqvBiBt:

$\vdash ba\ f = bi\ (\Box f)$

proof –

have 1: $\vdash da\ (\neg f) = di\ (\Diamond (\neg f))$ **by** (rule DaEqvDiDt)
have 2: $\vdash \Diamond (\neg f) = (\neg (\Box f))$ **by** (rule DiamondNotEqvNotBox)
hence 3: $\vdash di\ (\Diamond (\neg f)) = di\ (\neg (\Box f))$ **by** (rule DiEqvDi)
have 4: $\vdash da\ (\neg f) = di\ (\neg (\Box f))$ **using** 1 3 **by** fastforce
hence 5: $\vdash (\neg (da\ (\neg f))) = (\neg (di\ (\neg (\Box f))))$ **by** auto
hence 6: $\vdash (\neg (da\ (\neg f))) = bi\ (\Box f)$ **by** (simp add: bi-d-def)
from 6 **show** ?thesis **by** (simp add: ba-d-def)

qed

lemma DiNotEqvNotBi:

$\vdash di\ (\neg f) = (\neg (bi\ f))$

proof –

have 1: $\vdash bi\ f = (\neg (di\ (\neg f)))$ **by** (simp add: bi-d-def)
from 1 **show** ?thesis **by** auto

qed

lemma NotDiamondNotEqvBox:

$\vdash (\neg (\Diamond (\neg f))) = \Box f$

by (simp add: always-d-def)

lemma BaEqvBtBi:

$\vdash ba\ f = \Box (bi\ f)$

proof –

have 1: $\vdash da(\neg f) = \Diamond(di(\neg f))$ **by** (rule *DaEqvDtDi*)
have 2: $\vdash di(\neg f) = \neg(bi\ f)$ **by** (rule *DiNotEqvNotBi*)
hence 3: $\vdash \Diamond(di(\neg f)) = \Diamond(\neg(bi\ f))$ **by** (rule *DiamondEqvDiamond*)
have 4: $\vdash (\neg(\Diamond(\neg(bi\ f)))) = \Box(bi\ f)$ **by** (rule *NotDiamondNotEqvBox*)
have 5: $\vdash (\neg(da(\neg f))) = \Box(bi\ f)$ **using** 1 2 3 4 **by** *fastforce*
from 5 **show** ?thesis **by** (simp add: ba-d-def)
qed

lemma *BtBiEqvBiBt*:

$\vdash \Box(bi\ f) = bi(\Box f)$

proof –

have 1: $\vdash ba\ f = \Box(bi\ f)$ **by** (rule *BaEqvBtBi*)

have 2: $\vdash ba\ f = bi(\Box f)$ **by** (rule *BaEqvBiBt*)

from 1 2 **show** ?thesis **by** *fastforce*

qed

lemma *BoxStateEqvBaBoxState*:

$\vdash \Box(\text{init } w) = ba(\Box(\text{init } w))$

proof –

have 1: $\vdash (\text{init } w) = bi(\text{init } w)$ **by** (rule *StateEqvBi*)

hence 2: $\vdash \Box(\text{init } w) = \Box(bi(\text{init } w))$ **by** (rule *BoxEqvBox*)

have 3: $\vdash \Box(bi(\text{init } w)) = bi(\Box(\text{init } w))$ **by** (rule *BtBiEqvBiBt*)

have 4: $\vdash \Box(\text{init } w) = \Box(\Box(\text{init } w))$ **by** (rule *BoxEqvBoxBox*)

hence 5: $\vdash bi(\Box(\text{init } w)) = bi(\Box(\Box(\text{init } w)))$ **by** (rule *BiEqvBi*)

have 6: $\vdash ba(\Box(\text{init } w)) = bi(\Box(\Box(\text{init } w)))$ **by** (rule *BaEqvBiBt*)

from 2 3 5 6 **show** ?thesis **by** *fastforce*

qed

lemma *BalmpBi*:

$\vdash ba\ f \longrightarrow bi\ f$

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 *BalmpBt*:

$\vdash ba\ f \longrightarrow \Box f$

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 *BoxAndChopImp*:

$\vdash \Box h \wedge f; g \longrightarrow f; (h \wedge g)$

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 *BaAndChopImp*:

$\vdash ba\ f \wedge (g; g1) \longrightarrow (f \wedge g); (f \wedge g1)$

proof –

have 1: $\vdash ba\ f \longrightarrow bi\ f$ **by** (*rule BalmpBi*)

have 2: $\vdash bi\ f \wedge (g; g1) \longrightarrow (f \wedge g); g1$ **by** (*rule BiAndChopImp*)

have 3: $\vdash ba\ f \longrightarrow \Box f$ **by** (*rule BalmpBt*)

have 4: $\vdash \Box f \wedge (f \wedge g); g1 \longrightarrow (f \wedge g); (f \wedge g1)$ **by** (*rule BoxAndChopImp*)

from 1 2 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *ChopAndCommute*:

$\vdash f; (g \wedge g1) = f; (g1 \wedge g)$

proof –

have 1: $\vdash (g \wedge g1) = (g1 \wedge g)$ **by** *auto*

from 1 **show** *?thesis* **by** (*rule RightChopEqvChop*)

qed

lemma *ChopAndA*:

$\vdash f; (g \wedge g1) \longrightarrow f; g$

proof –

have 1: $\vdash (g \wedge g1) \longrightarrow g$ **by** *auto*

from 1 **show** *?thesis* **by** (*rule RightChopImpChop*)

qed

lemma *ChopAndB*:

$\vdash f; (g \wedge g1) \longrightarrow f; g1$

proof –

have 1: $\vdash (g \wedge g1) \longrightarrow g1$ **by** *auto*

from 1 **show** *?thesis* **by** (*rule RightChopImpChop*)

qed

lemma *BoxStateAndChopEqvChop*:

$\vdash (\Box (init\ w) \wedge (f; g)) = ((\Box (init\ w) \wedge f); (\Box (init\ w) \wedge g))$

proof –

have 1: $\vdash \Box (init\ w) = ba(\Box (init\ w))$

by (*rule BoxStateEqvBaBoxState*)

have 2: $\vdash ba(\Box (init\ w)) \wedge (f; g) \longrightarrow (\Box (init\ w) \wedge f); (\Box (init\ w) \wedge g)$
by (rule BaAndChopImport)
have 3: $\vdash \Box (init\ w) \wedge (f; g) \longrightarrow (\Box (init\ w) \wedge f); (\Box (init\ w) \wedge g)$
using 1 2 **by** fastforce
have 11: $\vdash (\Box (init\ w) \wedge f); (\Box (init\ w) \wedge g) \longrightarrow (\Box (init\ w)); (\Box (init\ w) \wedge g)$
by (rule AndChopA)
have 12: $\vdash (\Box (init\ w)); (\Box (init\ w) \wedge g) \longrightarrow (\Box (init\ w)); (\Box (init\ w))$
by (rule ChopAndA)
have 13: $\vdash (\Box (init\ w)); (\Box (init\ w)) = \Box (init\ w)$
by (rule BoxStateChopBoxEqvBox)
have 14: $\vdash (\Box (init\ w) \wedge f); (\Box (init\ w) \wedge g) \longrightarrow f; (\Box (init\ w) \wedge g)$
by (rule AndChopB)
have 15: $\vdash f; (\Box (init\ w) \wedge g) \longrightarrow f; g$
by (rule ChopAndB)
have 16: $\vdash (\Box (init\ w) \wedge f); (\Box (init\ w) \wedge g) \longrightarrow \Box (init\ w) \wedge (f; g)$
using 11 12 13 14 15 **by** fastforce
from 3 16 **show** ?thesis **by** fastforce
qed

lemma DiEqvNotBiNot:

$\vdash di\ f = (\neg (bi\ (\neg f)))$

proof –

have 1: $\vdash bi\ (\neg f) = (\neg (di\ (\neg \neg f)))$ **by** (simp add: bi-d-def)

hence 2: $\vdash di\ (\neg \neg f) = (\neg (bi\ (\neg f)))$ **by** auto

have 3: $\vdash f = (\neg \neg f)$ **by** auto

hence 4: $\vdash di\ f = di\ (\neg \neg f)$ **by** (rule DiEqvDi)

from 2 4 **show** ?thesis **by** auto

qed

lemma ChopAndBoxImport:

$\vdash f; g \wedge \Box h \longrightarrow f; (g \wedge h)$

proof –

have 1: $\vdash \Box h \wedge f; g \longrightarrow f; (h \wedge g)$ **by** (rule BoxAndChopImport)

have 2: $\vdash f; (h \wedge g) = f; (g \wedge h)$ **by** (rule ChopAndCommute)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma AndChopAndCommute:

$\vdash (f \wedge g); (f1 \wedge g1) = (g \wedge f); (g1 \wedge f1)$

proof –

have 1: $\vdash (f \wedge g); (f1 \wedge g1) = (g \wedge f); (f1 \wedge g1)$ **by** (rule AndChopCommute)

have 2: $\vdash (g \wedge f); (f1 \wedge g1) = (g \wedge f); (g1 \wedge f1)$ **by** (rule ChopAndCommute)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma ChopImpChop:

assumes $\vdash f \longrightarrow f1$

$\vdash g \longrightarrow g1$

shows $\vdash f; g \longrightarrow f1; g1$

proof –

have 1: $\vdash f \longrightarrow f1$ **using** *assms* **by** *auto*
hence 2: $\vdash f; g \longrightarrow f1; g$ **by** (*rule LeftChopImpChop*)
have 3: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*
hence 4: $\vdash f1; g \longrightarrow f1; g1$ **by** (*rule RightChopImpChop*)
from 2 4 **show** *?thesis* **by** *fastforce*
qed

lemma *ChopEqvChop*:
assumes $\vdash f = f1$
 $\vdash g = g1$
shows $\vdash f; g = f1; g1$
proof –
have 1: $\vdash f = f1$ **using** *assms* **by** *auto*
hence 2: $\vdash f; g = f1; g$ **by** (*rule LeftChopEqvChop*)
have 3: $\vdash g = g1$ **using** *assms* **by** *auto*
hence 4: $\vdash f1; g = f1; g1$ **by** (*rule RightChopEqvChop*)
from 2 4 **show** *?thesis* **by** *fastforce*
qed

lemma *BoxImpBoxImpBox*:
 $\vdash \Box h \longrightarrow \Box(g \longrightarrow \Box h \wedge g)$
proof –
have 1: $\vdash \Box h \longrightarrow (g \longrightarrow \Box h \wedge g)$ **by** *auto*
hence 2: $\vdash \Box(\Box h) \longrightarrow \Box(g \longrightarrow \Box h \wedge g)$ **by** (*rule ImpBoxRule*)
have 3: $\vdash \Box h = \Box(\Box h)$ **by** (*rule BoxEqvBoxBox*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *BoxChopImpChopBox*:
 $\vdash \Box h \longrightarrow f; g \longrightarrow f; (\Box h \wedge g)$
proof –
have 1: $\vdash \Box h \longrightarrow \Box(g \longrightarrow \Box h \wedge g)$ **by** (*rule BoxImpBoxImpBox*)
have 2: $\vdash \Box(g \longrightarrow \Box h \wedge g) \longrightarrow f; g \longrightarrow f; (\Box h \wedge g)$ **by** (*rule BoxChopImpChop*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *NotChopEqvYieldsNot*:
 $\vdash (\neg(f; g)) = f \text{ yields } (\neg g)$
proof –
have 1: $\vdash g = (\neg \neg g)$ **by** *auto*
hence 2: $\vdash f; g = f; (\neg \neg g)$ **by** (*rule RightChopEqvChop*)
hence 3: $\vdash (\neg(f; g)) = (\neg(f; (\neg \neg g)))$ **by** *auto*
from 3 **show** *?thesis* **by** (*simp add: yields-d-def*)
qed

lemma *NotDiFalse*:
 $\vdash \neg(di \# False)$
proof –
have 1: $\vdash (init \# True) \longrightarrow bi (init \# True)$ **by** (*rule StatImpBi*)
hence 2: $\vdash \# True \longrightarrow bi \# True$ **by** (*auto simp: bi-defs sum.case-eq-if*)

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: next-defs sum.case-eq-if*)

lemma *NextStateAndChop*:

$\vdash \bigcirc(((init\ w) \wedge f); g) = (\bigcirc(init\ w) \wedge \bigcirc(f; g))$

proof –

have 1: $\vdash ((init\ w) \wedge f); g = ((init\ w) \wedge f; g)$ **by** (*rule StateAndChop*)

hence 2: $\vdash \bigcirc(((init\ w) \wedge f); g) = \bigcirc((init\ w) \wedge f; g)$ **by** (*rule NextEqvNext*)

have 3: $\vdash \bigcirc((init\ w) \wedge f; g) = (\bigcirc(init\ w) \wedge \bigcirc(f; g))$ **by** (*rule NextAndEqvNextAndNext*)

from 2 3 **show** ?thesis **by** *fastforce*

qed

lemma *StateYieldsEqv*:

$\vdash ((init\ w) \longrightarrow (f\ yields\ g)) = ((init\ w) \wedge f)\ yields\ g$

proof –

have 1: $\vdash ((init\ w) \wedge f); (\neg g) = ((init\ w) \wedge f; (\neg g))$ **by** (*rule StateAndChop*)

hence 2: $\vdash ((init\ w) \longrightarrow \neg(f; (\neg g))) = (\neg(((init\ w) \wedge f); (\neg g)))$ **by** *auto*

from 2 **show** ?thesis **by** (*simp add: yields-d-def*)

qed

lemma *StateAndDi*:

$\vdash ((init\ w) \wedge di\ f) = di\ ((init\ w) \wedge f)$

proof –

have 1: $\vdash ((init\ w) \wedge f); \#True = ((init\ w) \wedge f; \#True)$ **by** (*rule StateAndChop*)

from 1 **show** ?thesis **by** (*metis di-d-def inteq-reflection*)

qed

lemma *DiNext*:

$\vdash di(\circ f) = \circ(di\ f)$

proof –

have 1: $\vdash (\circ f); \#True = \circ(f; \#True)$ by (rule NextChop)

from 1 show ?thesis by (simp add: di-d-def)

qed

lemma *DiNextState*:

$\vdash di(\circ (init\ w)) = \circ (init\ w)$

proof –

have 1: $\vdash di(\circ (init\ w)) = \circ(di\ (init\ w))$ by (rule DiNext)

have 2: $\vdash di\ (init\ w) = (init\ w)$ by (rule DiState)

hence 3: $\vdash \circ(di\ (init\ w)) = \circ (init\ w)$ by (rule NextEqvNext)

from 1 3 show ?thesis by fastforce

qed

lemma *StateImpBiGen*:

assumes $\vdash (init\ w) \longrightarrow f$

shows $\vdash (init\ w) \longrightarrow bi\ f$

proof –

have 1: $\vdash (init\ w) \longrightarrow f$ using assms by auto

hence 2: $\vdash \neg f \longrightarrow \neg (init\ w)$ by auto

hence 3: $\vdash di(\neg f) \longrightarrow di(\neg (init\ w))$ by (rule DilmpDi)

hence 4: $\vdash di(\neg f) \longrightarrow di\ (init\ (\neg w))$ by (metis Initprop(2) inteq-reflection)

have 5: $\vdash di\ (init\ (\neg w)) = (init\ (\neg w))$ by (rule DiState)

have 6: $\vdash di(\neg f) \longrightarrow \neg (init\ w)$ using 4 5 using Initprop(2) by fastforce

hence 7: $\vdash (init\ w) \longrightarrow \neg (di(\neg f))$ by auto

from 7 show ?thesis by (simp add: bi-d-def)

qed

lemma *ChopAndNotChopImp*:

$\vdash f; g \wedge \neg (f; g1) \longrightarrow f; (g \wedge \neg g1)$

proof –

have 1: $\vdash g \longrightarrow (g \wedge \neg g1) \vee g1$ by auto

hence 2: $\vdash f; g \longrightarrow f; ((g \wedge \neg g1) \vee g1)$ by (rule RightChopImpChop)

have 3: $\vdash f; ((g \wedge \neg g1) \vee g1) \longrightarrow (f; (g \wedge \neg g1)) \vee (f; g1)$ by (rule ChopOrImp)

have 4: $\vdash f; g \longrightarrow f; (g \wedge \neg g1) \vee f; g1$ using 2 3 MP by fastforce

from 4 show ?thesis by auto

qed

lemma *ChopAndYieldsImp*:

$\vdash f; g \wedge f\ yields\ g1 \longrightarrow f; (g \wedge g1)$

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

9.7 Properties of Fin

lemma *FinEqvTrueChopAndEmpty*:

$\vdash \text{fin } f = \# \text{True}; (f \wedge \text{empty})$

proof –

have 1: $\vdash \text{fin } f = \Box(\text{empty} \longrightarrow f)$

by (simp add: fin-d-def)

have 2: $\vdash \Box(\text{empty} \longrightarrow f) = (\neg(\Diamond(\neg(\text{empty} \longrightarrow f))))$

by (simp add: always-d-def)

have 3: $\vdash (\neg(\text{empty} \longrightarrow f)) = (\neg f \wedge \text{empty})$

by *auto*
hence 4: $\vdash \Diamond(\neg(\text{empty} \longrightarrow f)) = \Diamond(\neg f \wedge \text{empty})$
using *DiamondEqvDiamond* **by** *blast*
hence 5: $\vdash \neg(\Diamond(\neg(\text{empty} \longrightarrow f))) = (\neg(\Diamond(\neg f \wedge \text{empty})))$
by *auto*
have 6: $\vdash (\neg(\Diamond(\neg f \wedge \text{empty}))) = \# \text{True}; (f \wedge \text{empty})$
using *interval-suffix-intlast* **by** (*auto simp add: Valid-def sometimes-defs empty-defs chop-defs sum.case-eq-if*)
from 1 2 5 6 **show** *?thesis* **by** *fastforce*
qed

lemma *DiamondFin*:

$\vdash \Diamond(\text{fin } w) = \text{fin } w$

by (metis (no-types, lifting) *ChopAssoc ChopOrEqv FinEqvTrueChopAndEmpty FiniteChopFiniteEqvFinite FiniteChopInfEqvInf FiniteOrInfinite int-eq-true inteq-reflection sometimes-d-def*)

lemma *FiniteChopFinExportA*:

$\vdash (f \wedge \text{finite}); (g \wedge \text{fin } w) \longrightarrow \text{fin } w$

using *DiamondFin*

by (metis *ChopAndB FiniteChopImpDiamond inteq-reflection lift-imp-trans*)

lemma *FinImpBox*:

$\vdash \text{fin } w \longrightarrow \Box(\text{fin } w)$

by (metis *BoxImpBoxBox fin-d-def*)

lemma *FinAndChopImport*:

$\vdash (\text{fin } w) \wedge (f;g) \longrightarrow f;((\text{fin } w) \wedge g)$

proof –

have 1: $\vdash \text{fin } w \longrightarrow \Box(\text{fin } w)$ **by** (rule *FinImpBox*)

hence 2: $\vdash \text{fin } w \wedge f;g \longrightarrow \Box(\text{fin } w) \wedge (f;g)$ **by** *auto*

have 3: $\vdash \Box(\text{fin } w) \wedge (f;g) \longrightarrow f;((\text{fin } w) \wedge g)$ **using** *BoxAndChopImport* **by** *blast*

from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*

qed

lemma *FinAndChop*:

$\vdash ((f \wedge \text{finite}); (g \wedge \text{fin } w)) = (\text{fin } w \wedge (f \wedge \text{finite}); g)$

using *FinAndChopImport FiniteChopFinExportA ChopAndA ChopAndCommute*

by *fastforce*

lemma *ChopAndEmptyEqvEmptyChopEmpty*:

$\vdash ((f;g) \wedge \text{empty}) = (f \wedge \text{empty}); (g \wedge \text{empty})$

by (auto simp: *empty-defs chop-defs sum.case-eq-if*)

lemma *FinAndEmpty*:

$\vdash ((\text{fin } w) \wedge \text{empty}) = (w \wedge \text{empty})$

proof –

have 1: $\vdash ((\text{fin } w) \wedge \text{empty}) = (\# \text{True}; (w \wedge \text{empty}) \wedge \text{empty})$

using *FinEqvTrueChopAndEmpty* **by** *fastforce*

have 2: $\vdash (\# \text{True}; (w \wedge \text{empty}) \wedge \text{empty}) = ((\# \text{True} \wedge \text{empty}); (w \wedge \text{empty}))$

```

    using ChopAndEmptyEqvEmptyChopEmpty
    by (smt int-eq int-iffD2 lift-and-com Prop10 Prop12)
have 3:  $\vdash (\#True \wedge \text{empty}); (w \wedge \text{empty}) = (\text{empty}; (w \wedge \text{empty}))$ 
    using LeftChopEqvChop by fastforce
have 4:  $\vdash (\text{empty}; (w \wedge \text{empty})) = (w \wedge \text{empty})$ 
    using EmptyChop by blast
from 1 2 3 4 show ?thesis by fastforce
qed

```

lemma *AndFinEqvChopAndEmpty*:

```

 $\vdash ((f \wedge \text{finite}) \wedge \text{fin } g) = (f \wedge \text{finite}); (g \wedge \text{empty})$ 
proof –
have 1:  $\vdash ((f \wedge \text{finite}) \wedge \text{fin } g) = ((f \wedge \text{finite}); \text{empty} \wedge \text{fin } g)$ 
    using ChopEmpty by (metis inteq-reflection)
have 2:  $\vdash (\text{fin } g \wedge (f \wedge \text{finite}); \text{empty}) = ((f \wedge \text{finite}); (\text{empty} \wedge \text{fin } g))$ 
    using FinAndChop by fastforce
have 3:  $\vdash (\text{empty} \wedge \text{fin } g) = (\text{fin } g \wedge \text{empty})$ 
    by auto
have 4:  $\vdash (\text{fin } g \wedge \text{empty}) = (g \wedge \text{empty})$ 
    using FinAndEmpty by metis
have 5:  $\vdash (\text{empty} \wedge \text{fin } g) = (g \wedge \text{empty})$ 
    using 3 4 by auto
hence 6:  $\vdash (f \wedge \text{finite}); (\text{empty} \wedge \text{fin } g) = (f \wedge \text{finite}); (g \wedge \text{empty})$ 
    using RightChopEqvChop by blast
from 1 2 5 show ?thesis by (metis inteq-reflection lift-and-com)
qed

```

lemma *AndFinEqvChopStateAndEmpty*:

```

 $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)) = (f \wedge \text{finite}); ((\text{init } w) \wedge \text{empty})$ 
using AndFinEqvChopAndEmpty by blast

```

lemma *FinStateEqvStateAndEmptyOrNextFinState*:

```

 $\vdash \text{fin } (\text{init } w) = (((\text{init } w) \wedge \text{empty}) \vee \bigcirc (\text{fin } (\text{init } w)))$ 
proof –
have 1:  $\vdash \text{fin } (\text{init } w) = \Box (\text{empty} \longrightarrow \text{init } w)$ 
    by (simp add: fin-d-def)
have 2 :  $\vdash \Box (\text{empty} \longrightarrow \text{init } w) =$ 
     $((\text{empty} \longrightarrow \text{init } w) \wedge \text{wnext } (\Box (\text{empty} \longrightarrow \text{init } w)))$ 
    by (rule BoxEqvAndWnextBox)
have 3:  $\vdash \text{fin } (\text{init } w) = ((\text{empty} \longrightarrow \text{init } w) \wedge \text{wnext } (\text{fin } (\text{init } w)))$ 
    using 1 2 by (simp add: fin-d-def)
have 4:  $\vdash \text{wnext } (\text{fin } (\text{init } w)) = (\text{empty} \vee \bigcirc (\text{fin } (\text{init } w)))$ 
    by (rule WnextEqvEmptyOrNext)
have 5:  $\vdash \text{fin } (\text{init } w) = ((\text{empty} \longrightarrow \text{init } w) \wedge (\text{empty} \vee \bigcirc (\text{fin } (\text{init } w))))$ 
    using 3 4 by fastforce
have 6:  $\vdash ((\text{empty} \longrightarrow \text{init } w) \wedge (\text{empty} \vee \bigcirc (\text{fin } (\text{init } w)))) =$ 
     $((\text{empty} \longrightarrow \text{init } w) \wedge \text{empty}) \vee ((\text{empty} \longrightarrow \text{init } w) \wedge \bigcirc (\text{fin } (\text{init } w)))$ 
    by auto
have 7:  $\vdash ((\text{empty} \longrightarrow \text{init } w) \wedge \text{empty}) = ((\text{init } w) \wedge \text{empty})$ 
    by auto

```

have 8: $\vdash ((\text{empty} \longrightarrow \text{init } w) \wedge \bigcirc (\text{fin } (\text{init } w))) = \bigcirc (\text{fin } (\text{init } w))$
by (*metis 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*:

$\vdash (\text{fin } (\text{init } w) \wedge \text{finite}); f = \Diamond ((\text{init } w) \wedge f)$
proof –
have 1: $\vdash (\text{fin } (\text{init } w) \wedge \text{finite}) = (\text{finite}; ((\text{init } w) \wedge \text{empty}))$
by (*metis (no-types, lifting) ChopAndB DiamondEmptyEqvFinite DiamondFin FinAndChopImport FinAndEmpty Prop11 Prop12 inteq-reflection lift-and-com sometimes-d-def*)
hence 2: $\vdash (\text{fin } (\text{init } w) \wedge \text{finite}); f = (\text{finite}; ((\text{init } w) \wedge \text{empty}); f$
by (*rule LeftChopEqvChop*)
have 3: $\vdash \text{finite}; ((\text{init } w) \wedge \text{empty}); f = (\text{finite}; ((\text{init } w) \wedge \text{empty}); f$
by (*rule ChopAssoc*)
have 4: $\vdash \text{finite}; ((\text{init } w) \wedge \text{empty}); f = \Diamond ((\text{init } w) \wedge \text{empty}); f$
by (*simp add: sometimes-d-def*)
have 5: $\vdash ((\text{init } w) \wedge \text{empty}); f = ((\text{init } w) \wedge f)$
using *StateAndEmptyChop* **by** *blast*
hence 6: $\vdash \Diamond ((\text{init } w) \wedge \text{empty}); f = \Diamond ((\text{init } w) \wedge f)$
by (*rule DiamondEqvDiamond*)
from 2 3 4 6 **show** ?thesis **by** *fastforce*
qed

lemma *NotDiamondAndNot*:

$\vdash \neg (\Diamond (f \wedge \neg f))$
proof –
have 1: $\vdash (\neg (\Diamond (f \wedge \neg f))) = \Box (\neg (f \wedge \neg f))$ **using** *NotDiamondNotEqvBox* **by** *fastforce*
have 2: $\vdash \neg (f \wedge \neg f)$ **by** *simp*

have 3: $\vdash \Box(\neg(f \wedge \neg f))$ **using** 2 **by** (*simp add: BoxGen*)
from 1 3 **show** ?thesis **by** fastforce
qed

lemma *FinYields*:

$\vdash (fin (init w) \wedge finite) yields (init w)$

proof –

have 1: $\vdash (fin (init w) \wedge finite); (\neg(init w)) = \Diamond((init w) \wedge \neg(init w))$

by (*rule FinChopEqvDiamond*)

have 2: $\vdash \neg(\Diamond((init w) \wedge \neg(init w)))$ **by** (*rule NotDiamondAndNot*)

have 3: $\vdash \neg((fin (init w) \wedge finite); (\neg(init w)))$ **using** 1 2 **by** fastforce

from 3 **show** ?thesis **by** (*simp add: yields-d-def*)

qed

lemma *ImpAndFinStateOrFinNotState*:

$\vdash f \longrightarrow (f \wedge fin (init w)) \vee (f \wedge fin (\neg (init w)))$

by (*simp add: fin-defs Valid-def sum.case-eq-if*)

lemma *AndFinChopEqvStateAndChop*:

$\vdash ((f \wedge finite) \wedge fin (init w)); g = (f \wedge finite); ((init w) \wedge g)$

proof –

have 1: $\vdash (fin (init w) \wedge finite) yields (init w)$

by (*rule FinYields*)

have 2: $\vdash (f \wedge finite) \wedge fin (init w) \longrightarrow fin (init w)$

by auto

hence 3: $\vdash (fin (init w) \wedge finite) yields (init w) \longrightarrow$

$((f \wedge finite) \wedge fin (init w)) yields (init w)$

using LeftYieldsImpYields

by (*metis AndFinEqvChopAndEmpty Prop11 Prop12 inteq-reflection*)

have 4: $\vdash ((f \wedge finite) \wedge fin (init w)) yields (init w)$

using 1 3 **MP** **by** fastforce

have 5: $\vdash ((f \wedge finite) \wedge fin (init w)); g \wedge ((f \wedge finite) \wedge fin (init w)) yields (init w)$

$\longrightarrow ((f \wedge finite) \wedge fin (init w)); (g \wedge (init w))$

by (*rule ChopAndYieldsImp*)

have 6: $\vdash ((f \wedge finite) \wedge fin (init w)); g \longrightarrow$

$((f \wedge finite) \wedge fin (init w)); (g \wedge (init w))$

using 4 5 **by** fastforce

have 7: $\vdash ((f \wedge finite) \wedge fin (init w)); (g \wedge (init w)) \longrightarrow (f \wedge finite); (g \wedge (init w))$

by (*rule AndChopA*)

have 8: $\vdash g \wedge (init w) \longrightarrow (init w) \wedge g$

by auto

hence 9: $\vdash (f \wedge finite); (g \wedge (init w)) \longrightarrow (f \wedge finite); ((init w) \wedge g)$

by (*rule RightChopImpChop*)

have 10: $\vdash ((f \wedge finite) \wedge fin (init w)); g \longrightarrow (f \wedge finite); ((init w) \wedge g)$

using 6 7 9 **by** fastforce

have 11: $\vdash (f \wedge finite) \longrightarrow ((f \wedge finite) \wedge fin (init w)) \vee ((f \wedge finite) \wedge fin (\neg (init w)))$

using ImpAndFinStateOrFinNotState **by** blast

hence 12: $\vdash (f \wedge finite); ((init w) \wedge g) \longrightarrow$

$((f \wedge finite) \wedge fin (init w)) \vee$

$((finite \wedge f) \wedge fin (\neg (init w)))$; $((init w) \wedge g)$

```

using LeftChopImpChop
by (metis inteq-reflection lift-and-com)
have 13:  $\vdash (((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)) \vee ((f \wedge \text{finite}) \wedge \text{fin } (\neg (\text{init } w)))); ((\text{init } w) \wedge g)$ 
=
 $((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w));$ 
 $((\text{init } w) \wedge g) \vee ((f \wedge \text{finite}) \wedge \text{fin } (\neg (\text{init } w))); ((\text{init } w) \wedge g)$ 
by (rule OrChopEqv)
have 14:  $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } (\neg w))); ((\text{init } w) \wedge g) \longrightarrow$ 
 $\Diamond((\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g))$ 
using FinChopEqvDiamond
by (metis ChopImpChop Prop12 int-iffD1 inteq-reflection lift-and-com)
have 141:  $\vdash \neg(\Diamond((\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g))) \longrightarrow$ 
 $\neg((f \wedge \text{finite}) \wedge \text{fin } (\text{init } (\neg w))); ((\text{init } w) \wedge g)$ 
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: sometimes-defs init-defs sum.case-eq-if)
have 151:  $\vdash \neg((f \wedge \text{finite}) \wedge \text{fin } (\text{init } (\neg w))); ((\text{init } w) \wedge g)$ 
using 15 141 by fastforce
have 1511:  $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\neg (\text{init } w))); ((\text{init } w) \wedge g) \longrightarrow \#False$ 
using 151 by (metis Initprop(2) int-simps(14) inteq-reflection)
have 152:  $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w));$ 
 $((\text{init } w) \wedge g) \vee ((f \wedge \text{finite}) \wedge \text{fin } (\neg (\text{init } w))); ((\text{init } w) \wedge g) \longrightarrow$ 
 $((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); ((\text{init } w) \wedge g)$ 
using 1511 by fastforce
have 16:  $\vdash (f \wedge \text{finite}); ((\text{init } w) \wedge g) \longrightarrow ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); ((\text{init } w) \wedge g)$ 
using 12 13 152
by (metis (no-types, lifting) inteq-reflection lift-and-com lift-imp-trans)
have 17:  $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); ((\text{init } w) \wedge g) \longrightarrow ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); g$ 
by (rule ChopAndB)
have 18:  $\vdash (f \wedge \text{finite}); ((\text{init } w) \wedge g) \longrightarrow ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); g$ 
using 16 17 by fastforce
from 10 18 show ?thesis by fastforce
qed

```

lemma *DiAndFinEqvChopState*:

$\vdash \text{di } ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)) = (f \wedge \text{finite}); (\text{init } w)$

proof –

have 1: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); \#True = (f \wedge \text{finite}); ((\text{init } w) \wedge \#True)$

by (rule AndFinChopEqvStateAndChop)

have 2: $\vdash ((\text{init } w) \wedge \#True) = (\text{init } w)$ **by** auto

hence 3: $\vdash ((f \wedge \text{finite}); ((\text{init } w) \wedge \#True)) = ((f \wedge \text{finite}); (\text{init } w))$

by (rule RightChopEqvChop)

have 4: $\vdash ((f \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); \#True = (f \wedge \text{finite}); (\text{init } w)$

using 1 3 **by** auto

from 4 **show** ?thesis **by** (simp add: di-d-def)

qed

lemma *FinNotStateEqvNotFinState*:

$\vdash (\neg(\text{fin } (\text{init } w)) \wedge \text{finite}) = (\text{fin } (\text{init } (\neg w)) \wedge \text{finite})$

using *FinEqvTrueChopAndEmpty Finprop(4) Initprop(2) FinitImpAnd*
by (*metis inteq-reflection*)

lemma *BilmpFinEqvYieldsState*:

$\vdash bi (f \wedge finite \longrightarrow fin (init w)) = (f \wedge finite) yields (init w)$

proof –

have 1: $\vdash di ((f \wedge finite) \wedge fin (init (\neg w))) = (f \wedge finite); (init (\neg w))$
by (*rule DiAndFinEqvChopState*)

have 2: $\vdash ((f \wedge finite) \wedge fin (init (\neg w))) = ((f \wedge finite) \wedge \neg (fin (init w)))$
using *FinNotStateEqvNotFinState* **by** *fastforce*

have 3: $\vdash ((f \wedge finite) \wedge \neg (fin (init w))) = (\neg (f \wedge finite \longrightarrow fin (init w)))$
by *auto*

have 4: $\vdash ((f \wedge finite) \wedge fin (init (\neg w))) = (\neg (f \wedge finite \longrightarrow fin (init w)))$
using 2 3 **by** *fastforce*

hence 5: $\vdash di ((f \wedge finite) \wedge fin (init (\neg w))) = di (\neg (f \wedge finite \longrightarrow fin (init w)))$
by (*rule DiEqvDi*)

have 6: $\vdash di (\neg (f \wedge finite \longrightarrow fin (init w))) = (\neg (bi (f \wedge finite \longrightarrow fin (init w))))$
by (*rule DiNotEqvNotBi*)

have 7: $\vdash \neg (bi (f \wedge finite \longrightarrow fin (init w))) = (f \wedge finite); (init (\neg w))$
using 1 5 6 *Initprop* **by** *fastforce*

hence 8: $\vdash bi (f \wedge finite \longrightarrow fin (init w)) = (\neg ((f \wedge finite); (\neg (init w))))$
by (*metis Initprop(2) int-eq int-simps(7)*)

from 8 **show** *?thesis* **by** (*simp add: yields-d-def*)

qed

lemma *StatImpYields*:

assumes $\vdash (init w) \wedge f \wedge finite \longrightarrow fin (init w1)$

shows $\vdash (init w) \longrightarrow ((f \wedge finite) yields (init w1))$

proof –

have 1: $\vdash (init w) \wedge f \wedge finite \longrightarrow fin (init w1)$ **using** *assms* **by** *auto*

hence 2: $\vdash (init w) \longrightarrow (f \wedge finite \longrightarrow fin (init w1))$ **by** *auto*

hence 3: $\vdash (init w) \longrightarrow bi (f \wedge finite \longrightarrow fin (init w1))$ **by** (*rule StatImpBiGen*)

have 4: $\vdash bi (f \wedge finite \longrightarrow fin (init w1)) = (f \wedge finite) yields (init w1)$
by (*rule BilmpFinEqvYieldsState*)

from 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *StateAndYieldsImpYields*:

assumes $\vdash (init w) \wedge f \longrightarrow f1$

shows $\vdash (init w) \wedge (f1 yields g) \longrightarrow (f yields g)$

proof –

have 1: $\vdash (init w) \wedge f \longrightarrow f1$ **using** *assms* **by** *auto*

hence 2: $\vdash (init w) \wedge (f; (\neg g)) \longrightarrow f1; (\neg g)$ **by** (*rule StateAndChopImpChopRule*)

hence 3: $\vdash (init w) \wedge \neg (f1; (\neg g)) \longrightarrow \neg (f; (\neg g))$ **by** *auto*

from 3 **show** *?thesis* **by** (*simp add: yields-d-def*)

qed

lemma *AndYieldsA*:

$\vdash f yields g \longrightarrow (f \wedge f1) yields g$

proof –
have 1: $\vdash f \wedge f1 \longrightarrow f$ **by** *auto*
from 1 **show** *?thesis* **by** (rule *LeftYieldsImpYields*)
qed

lemma *AndYieldsB*:
 $\vdash f1 \text{ yields } g \longrightarrow (f \wedge f1) \text{ yields } g$

proof –
have 1: $\vdash f \wedge f1 \longrightarrow f1$ **by** *auto*
from 1 **show** *?thesis* **by** (rule *LeftYieldsImpYields*)
qed

lemma *RightYieldsImpYields*:

assumes $\vdash g \longrightarrow g1$
shows $\vdash (f \text{ yields } g) \longrightarrow (f \text{ yields } g1)$
proof –
have 1: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*
hence 2: $\vdash \neg g1 \longrightarrow \neg g$ **by** *auto*
hence 3: $\vdash f; (\neg g1) \longrightarrow f; (\neg g)$ **by** (rule *RightChopImpChop*)
hence 4: $\vdash \neg (f; (\neg g)) \longrightarrow \neg (f; (\neg g1))$ **by** *auto*
from 4 **show** *?thesis* **by** (*simp add: yields-d-def*)
qed

lemma *RightYieldsEqvYields*:

assumes $\vdash g = g1$
shows $\vdash (f \text{ yields } g) = (f \text{ yields } g1)$
proof –
have 1: $\vdash g = g1$ **using** *assms* **by** *auto*
hence 2: $\vdash (\neg g) = (\neg g1)$ **by** *auto*
hence 3: $\vdash f; (\neg g) = f; (\neg g1)$ **by** (rule *RightChopEqvChop*)
hence 4: $\vdash \neg (f; (\neg g)) = \neg (f; (\neg g1))$ **by** *auto*
from 4 **show** *?thesis* **by** (*simp add: yields-d-def*)
qed

lemma *BoxImpYields*:

$\vdash \Box g \longrightarrow (f \wedge \text{finite}) \text{ yields } g$
proof –
have 1: $\vdash (f \wedge \text{finite}); (\neg g) \longrightarrow \Diamond(\neg g)$ **by** (rule *FiniteChopImpDiamond*)
hence 2: $\vdash \neg (\Diamond(\neg g)) \longrightarrow \neg ((f \wedge \text{finite}); (\neg g))$ **by** *auto*
from 2 **show** *?thesis* **by** (*simp add: yields-d-def always-d-def*)
qed

lemma *BoxEqvFiniteYields*:

$\vdash \Box f = \text{finite yields } f$
proof –
have 1: $\vdash \text{finite}; (\neg f) = \Diamond(\neg f)$ **by** (rule *FiniteChopEqvDiamond*)
hence 2: $\vdash \neg (\text{finite}; (\neg f)) = \neg (\Diamond(\neg f))$ **by** *auto*
have 3: $\vdash \Box f = \neg (\Diamond(\neg f))$ **by** (*simp add: always-d-def*)
have 4: $\vdash \Box f = \neg (\text{finite}; (\neg f))$ **using** 2 3 **by** *fastforce*
from 4 **show** *?thesis* **by** (*simp add: yields-d-def*)

qed

lemma *YieldsGen*:

assumes $\vdash g$

shows $\vdash (f \wedge \text{finite}) \text{ yields } g$

proof –

have $1: \vdash g$ **using** *assms* **by** *auto*

hence $2: \vdash \Box g$ **by** (*rule BoxGen*)

have $3: \vdash \Box g \longrightarrow (f \wedge \text{finite}) \text{ yields } g$ **by** (*rule BoxImpYields*)

from $2\ 3$ **show** *?thesis* **using** *MP* **by** *fastforce*

qed

lemma *YieldsAndYieldsEqvYieldsAnd*:

$\vdash ((f \text{ yields } g) \wedge (f \text{ yields } g1)) = f \text{ yields } (g \wedge g1)$

proof –

have $1: \vdash f; (\neg g \vee \neg g1) = ((f; (\neg g)) \vee (f; (\neg g1)))$ **by** (*rule ChopOrEqv*)

hence $2: \vdash ((f; (\neg g)) \vee (f; (\neg g1))) = f; (\neg g \vee \neg g1)$ **by** *auto*

have $3: \vdash (\neg g \vee \neg g1) = (\neg (g \wedge g1))$ **by** *auto*

hence $4: \vdash f; (\neg g \vee \neg g1) = f; (\neg (g \wedge g1))$ **by** (*rule RightChopEqvChop*)

have $5: \vdash (f; (\neg g)) \vee (f; (\neg g1)) = f; (\neg (g \wedge g1))$ **using** $2\ 4$ **by** *fastforce*

hence $6: \vdash (\neg (f; (\neg g)) \wedge \neg (f; (\neg g1))) = (\neg (f; (\neg (g \wedge g1))))$

by (*auto simp: chop-defs sum.case-eq-if*)

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: *more-defs next-defs sum.case-eq-if*)

have 3: $\vdash (\bigcirc \# \text{True}; f) \longrightarrow \text{more}$ **using** 1 2 **by** fastforce

from 3 **show** ?thesis **by** (metis *more-d-def*)

qed

lemma *FmoreChopImpFmore*:

$\vdash \text{fmore} ; (f \wedge \text{finite}) \longrightarrow \text{fmore}$

proof —

have 1: $\vdash \text{fmore}; (f \wedge \text{finite}) = \bigcirc(\text{finite}; (f \wedge \text{finite}))$

using *FmoreEqvSkipChopFinite* **by** (metis *NextChop inteq-reflection next-d-def*)

have 2: $\vdash \bigcirc(\text{finite}; (f \wedge \text{finite})) \longrightarrow \text{fmore}$

by (auto simp: *fmore-defs chop-defs finite-defs more-defs next-defs sum.case-eq-if*)

have 3: $\vdash (\bigcirc \text{finite}; (f \wedge \text{finite})) \longrightarrow \text{fmore}$ **using** 1 2

by (metis *FmoreEqvSkipChopFinite inteq-reflection next-d-def*)

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

qed

lemma *ChopMoreImpMore*:

$\vdash f; \text{more} \longrightarrow \text{more}$

proof —

have 1: $\vdash (f \wedge \text{finite}); \text{more} \longrightarrow \Diamond \text{more}$ **by** (rule *FiniteChopImpDiamond*)

have 11: $\vdash (f \wedge \text{inf}); \text{more} \longrightarrow \text{more}$

by (simp add: *more-defs infinite-defs chop-defs Valid-def sum.case-eq-if*)

have 2: $\vdash \Diamond \text{more} \longrightarrow \text{more}$ **by** (auto simp: *more-defs sometimes-defs sum.case-eq-if*)

have 3: $\vdash (f \wedge \text{finite}); \text{more} \longrightarrow \text{more}$ **using** 1 2 **by** fastforce

have 4: $\vdash f = ((f \wedge \text{finite}) \vee (f \wedge \text{inf}))$

by (simp add: *Valid-def finite-defs infinite-defs sum.case-eq-if*)

hence 5: $\vdash f; \text{more} = ((f \wedge \text{finite}); \text{more} \vee (f \wedge \text{inf}); \text{more})$ **by** (simp add: *OrChopEqvRule*)

from 11 3 5 **show** ?thesis **by** fastforce

qed

lemma *MoreChopEqvNextDiamond*:

$\vdash \text{fmore} ; f = \bigcirc(\Diamond f)$

proof —

have 1: $\vdash \text{fmore} ; f = (\bigcirc \text{finite}); f$

by (simp add: *Valid-def chop-defs fmore-defs next-defs finite-defs sum.case-eq-if*)

have 2: $\vdash (\bigcirc \text{finite}); f = \bigcirc(\text{finite}; f)$ **by** (rule *NextChop*)

have 3: $\vdash \text{fmore} ; f = \bigcirc(\text{finite}; f)$ **using** 1 2 **by** fastforce

from 3 **show** ?thesis **by** (simp add: *sometimes-d-def*)

qed

lemma *WeakNextBoxImpMoreYields*:

$\vdash \text{fmore} \text{ yields } f = \text{wnext}(\Box f)$

proof —

have 1: $\vdash \text{fmore} ; (\neg f) = \bigcirc(\Diamond (\neg f))$ **by** (rule *MoreChopEqvNextDiamond*)

have 2: $\vdash \bigcirc(\Diamond (\neg f)) = \bigcirc(\neg(\Box f))$ **by** (auto simp: *always-d-def*)

have 3: $\vdash \circ(\neg(\Box f)) = (\neg(\text{wnext}(\Box f)))$ **by** (auto simp: wnext-d-def)
have 4: $\vdash f\text{more} ; (\neg f) = (\neg(f\text{more yields } f))$ **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 LeftChopImpFMoreRule:

assumes $\vdash f \longrightarrow f\text{more}$
shows $\vdash f; (g \wedge \text{finite}) \longrightarrow f\text{more}$

proof –

have 1: $\vdash f \longrightarrow f\text{more}$ **using** assms **by** auto
hence 2: $\vdash f; (g \wedge \text{finite}) \longrightarrow f\text{more} ; (g \wedge \text{finite})$ **by** (rule LeftChopImpChop)
have 3: $\vdash f\text{more} ; (g \wedge \text{finite}) \longrightarrow f\text{more}$ **using** FmoreChopImpFmore **by** fastforce
from 2 3 **show** ?thesis **using** lift-imp-trans **by** blast

qed

lemma RightChopImpMoreRule:

assumes $\vdash g \longrightarrow \text{more}$
shows $\vdash f; g \longrightarrow \text{more}$

proof –

have 1: $\vdash g \longrightarrow \text{more}$ **using** assms **by** auto
hence 2: $\vdash f; g \longrightarrow f; \text{more}$ **by** (rule RightChopImpChop)
have 3: $\vdash f; \text{more} \longrightarrow \text{more}$ **by** (rule ChopMoreImpMore)
from 2 3 **show** ?thesis **using** lift-imp-trans **by** blast

qed

lemma NotDiEqvBiNot:

$\vdash (\neg(\neg \text{di } f)) = \text{bi } (\neg f)$

proof –

have 1: $\vdash f = (\neg \neg f)$ **by** *auto*
hence 2: $\vdash di\ f = di\ (\neg \neg f)$ **by** (rule *DiEqvDi*)
hence 3: $\vdash (\neg (di\ f)) = (\neg (di\ (\neg \neg f)))$ **by** *auto*
from 3 **show** ?thesis **by** (simp add: *bi-d-def*)
qed

lemma *ChopImpDi*:
 $\vdash f; g \longrightarrow di\ f$
proof —
have 1: $\vdash g \longrightarrow \#True$ **by** *auto*
hence 2: $\vdash f; g \longrightarrow f; \#True$ **by** (rule *RightChopImpChop*)
from 2 **show** ?thesis **by** (simp add: *di-d-def*)
qed

lemma *TrueEqvTrueChopTrue*:
 $\vdash \#True = \#True; \#True$
proof —
have 1: $\vdash \#True; \#True \longrightarrow \#True$ **by** *auto*
have 2: $\vdash \#True \longrightarrow di\ \#True$ **by** (rule *DiIntro*)
hence 3: $\vdash \#True \longrightarrow \#True; \#True$ **by** (simp add: *di-d-def*)
from 1 3 **show** ?thesis **by** *auto*
qed

lemma *DiEqvDiDi*:
 $\vdash di\ f = di\ (di\ f)$
proof —
have 1: $\vdash \#True = \#True; \#True$ **by** (rule *TrueEqvTrueChopTrue*)
hence 2: $\vdash f; \#True = f; (\#True; \#True)$ **by** (rule *RightChopEqvChop*)
have 3: $\vdash f; (\#True; \#True) = (f; \#True); \#True$ **by** (rule *ChopAssoc*)
have 4: $\vdash f; \#True = (f; \#True); \#True$ **using** 2 3 **by** *fastforce*
from 4 **show** ?thesis **by** (metis *di-d-def*)
qed

lemma *BiEqvBiBi*:
 $\vdash bi\ f = bi\ (bi\ f)$
proof —
have 1: $\vdash di\ (\neg f) = di\ (di\ (\neg f))$ **by** (rule *DiEqvDiDi*)
have 2: $\vdash di\ (\neg f) = (\neg (bi\ f))$ **by** (rule *DiNotEqvNotBi*)
hence 3: $\vdash di\ (di\ (\neg f)) = di\ (\neg (bi\ f))$ **by** (rule *DiEqvDi*)
have 4: $\vdash di\ (\neg f) = di\ (\neg (bi\ f))$ **using** 1 3 **by** *fastforce*
hence 5: $\vdash (\neg (di\ (\neg f))) = (\neg (di\ (\neg (bi\ f))))$ **by** *fastforce*
from 5 **show** ?thesis **by** (metis *bi-d-def*)
qed

lemma *DiOrEqv*:
 $\vdash di\ (f \vee g) = (di\ f \vee di\ g)$
proof —
have 1: $\vdash (f \vee g); \#True = (f; \#True \vee g; \#True)$ **by** (rule *OrChopEqv*)
from 1 **show** ?thesis **by** (simp add: *di-d-def*)
qed

lemma *DiAndA*:

$\vdash \text{di } (f \wedge g) \longrightarrow \text{di } f$

proof –

have 1: $\vdash (f \wedge g); \# \text{True} \longrightarrow f; \# \text{True}$ **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: $\vdash (f \wedge g); \# \text{True} \longrightarrow g; \# \text{True}$ **by** (rule *AndChopB*)

from 1 **show** ?thesis **by** (simp add: di-d-def)

qed

lemma *DiAndImpAnd*:

$\vdash \text{di } (f \wedge g) \longrightarrow \text{di } f \wedge \text{di } g$

proof –

have 1: $\vdash \text{di } (f \wedge g) \longrightarrow \text{di } f$ **by** (rule *DiAndA*)

have 2: $\vdash \text{di } (f \wedge g) \longrightarrow \text{di } g$ **by** (rule *DiAndB*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *DiSkipEqvMore*:

$\vdash \text{di } \text{skip} = \text{more}$

proof –

have 1: $\vdash \text{skip}; \# \text{True} = \bigcirc \# \text{True}$ **by** (rule *SkipChopEqvNext*)

have 2: $\vdash \bigcirc \# \text{True} = \text{more}$ **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*:

$\vdash \text{di } \text{more} = \text{more}$

proof –

have 1: $\vdash \text{di } (\bigcirc \# \text{True}) = \bigcirc (\text{di } \# \text{True})$ **by** (rule *DiNext*)

have 2: $\vdash \bigcirc (\text{di } \# \text{True}) \longrightarrow \text{more}$ **by** (auto simp: next-defs di-defs more-defs sum.case-eq-if)

have 3: $\vdash \text{di } (\bigcirc \# \text{True}) \longrightarrow \text{more}$ **using** 1 2 **by** fastforce

hence 4: $\vdash \text{di } \text{more} \longrightarrow \text{more}$ **by** (simp add: more-d-def)

have 5: $\vdash \text{more} \longrightarrow \text{di } \text{more}$ **by** (rule *ImpDi*)

from 4 5 **show** ?thesis **by** fastforce

qed

lemma *DiIfEqvRule*:

assumes $\vdash f = \text{if}_i (\text{init } w) \text{ then } g \text{ else } h$

shows $\vdash \text{di } f = \text{if}_i (\text{init } w) \text{ then } (\text{di } g) \text{ else } (\text{di } h)$

proof –

have 1: $\vdash f = \text{if}_i (\text{init } w) \text{ then } g \text{ else } h$ **using** assms **by** auto

hence 2: $\vdash f; \# \text{True} = \text{if}_i (\text{init } w) \text{ then } (g; \# \text{True}) \text{ else } (h; \# \text{True})$ **by** (rule *IfChopEqvRule*)

from 2 **show** ?thesis **by** (simp add: di-d-def)

qed

lemma *DiEmpty*:

$\vdash \text{di empty}$

proof –

have 1: $\vdash \# \text{True}$ **by** *auto*

have 2: $\vdash \text{empty} ; \# \text{True} = \# \text{True}$ **by** (*rule EmptyChop*)

have 3: $\vdash \text{empty} ; \# \text{True}$ **using** 1 2 **by** *auto*

from 3 **show** ?thesis **by** (*simp add: di-d-def*)

qed

lemma *DaNotEqvNotBa*:

$\vdash \text{da} (\neg f) = (\neg (\text{ba } f))$

proof –

have 1: $\vdash \text{ba } f = (\neg (\text{da} (\neg f)))$ **by** (*simp add: ba-d-def*)

from 1 **show** ?thesis **by** *fastforce*

qed

lemma *DaEqvDa*:

assumes $\vdash f = g$

shows $\vdash \text{da } f = \text{da } g$

using *assms* **using** *int-eq* **by** *force*

lemma *DaEqvNotBaNot*:

$\vdash \text{da } f = (\neg (\text{ba} (\neg f)))$

proof –

have 1: $\vdash \text{ba} (\neg f) = (\neg (\text{da} (\neg \neg f)))$ **by** (*simp add: ba-d-def*)

hence 2: $\vdash \text{da} (\neg \neg f) = (\neg (\text{ba} (\neg f)))$ **by** *fastforce*

have 3: $\vdash f = (\neg \neg f)$ **by** *simp*

hence 4: $\vdash \text{da } f = \text{da} (\neg \neg f)$ **by** (*rule DaEqvDa*)

from 2 4 **show** ?thesis **by** *simp*

qed

lemma *BaElim*:

$\vdash \text{ba } f \longrightarrow f$

proof –

have 1: $\vdash \text{ba } f = \Box (bi \ f)$ **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** *BalmpBt lift-imp-trans* **by** *metis*

qed

lemma *DaIntro*:

$\vdash f \longrightarrow \text{da } f$

proof –

have 1: $\vdash \text{ba} (\neg f) \longrightarrow (\neg f)$ **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 *BalmpDist*:

$\vdash ba(f \longrightarrow g) \longrightarrow ba f \longrightarrow ba g$

proof —

have 1: $\vdash bi(f \longrightarrow g) \longrightarrow (bi f \longrightarrow bi g)$ **by** (*rule BilmpDist*)
 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: bi-defs sum.case-eq-if*)
 hence 3: $\vdash \Box(bi(f \wedge g)) = \Box(bi f \wedge bi g)$
using *BoxEqvBox* **by** *blast*
 have 4: $\vdash \Box(bi f \wedge bi g) = (\Box(bi f) \wedge \Box(bi g))$
by (*metis 2 BoxAndBoxEqvBoxRule inteq-reflection*)
 have 5: $\vdash ba f = \Box(bi f)$
by (*rule BaEqvBtBi*)
 have 6: $\vdash ba g = \Box(bi g)$
by (*rule BaEqvBtBi*)
 from 1 3 4 5 6 **show** *?thesis* **by** *fastforce*
qed

lemma *BalmpBaEqvBa*:

$\vdash ba (f = g) \longrightarrow (ba f = ba g)$

proof –

have 1: $\vdash ba (f \longrightarrow g) \longrightarrow ba f \longrightarrow ba g$ **by** (*rule BalmpDist*)

have 2: $\vdash ba (g \longrightarrow f) \longrightarrow ba g \longrightarrow ba f$ **by** (*rule BalmpDist*)

have 3: $\vdash ba (f = g) = ba ((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** (*auto simp: ba-defs sum.case-eq-if*)

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 *BalmpBa*:

assumes $\vdash f \longrightarrow g$

shows $\vdash ba f \longrightarrow ba g$

using *BaGen BalmpDist MP assms* **by** *metis*

lemma *BaEqvBa*:

assumes $\vdash f = g$

shows $\vdash ba f = ba g$

using *BaGen BalmpBaEqvBa MP assms* **by** *metis*

lemma *DalmpDa*:

assumes $\vdash f \longrightarrow g$

shows $\vdash da f \longrightarrow da g$

using *assms* **by** (*metis DaEqvDtDi DiAndB DiamondImpDiamond inteq-reflection Prop10*)

lemma *DiamondEqvDiamondDiamond*:

$\vdash \diamond f = \diamond (\diamond f)$

proof –

have 1: $\vdash \diamond (\diamond f) = finite;(finite;f)$

by (*simp add: sometimes-d-def*)

have 2: $\vdash finite;(finite;f) = (finite;finite);f$

by (*rule ChopAssoc*)

have 3: $\vdash (finite;finite);f = finite;f$

by (*simp add: LeftChopEqvChop FiniteChopFiniteEqvFinite*)

have 4: $\vdash finite;f = \diamond f$

by (*simp add: sometimes-d-def*)

from 1 2 3 4 **show** ?thesis **by** *fastforce*

qed

lemma *DaEqvDaDa*:

$\vdash da f = da (da f)$

proof –

have 1: $\vdash da f = \diamond (di f)$

by (*rule DaEqvDtDi*)

have 2: $\vdash di f = (di (di f))$

by (*rule DiEqvDiDi*)

hence 3: $\vdash \diamond (di f) = \diamond (di (di f))$

by (*rule DiamondEqvDiamond*)

have 4: $\vdash \Diamond (di\ f) = \Diamond(\Diamond (di\ (di\ f)))$
using *DiamondEqvDiamondDiamond DiEqvDiDi* **using** 3 **by** *fastforce*
have 5: $\vdash \Diamond (di\ (di\ f)) = di\ (\Diamond (di\ f))$
by (*rule DtDiEqvDiDt*)
hence 6: $\vdash \Diamond(\Diamond (di\ (di\ f))) = \Diamond (di\ (\Diamond (di\ f)))$
by (*rule DiamondEqvDiamond*)
have 7: $\vdash da\ f = \Diamond (di\ (\Diamond (di\ f)))$
using 1 3 4 6 **by** *fastforce*
have 8: $\vdash da\ (\Diamond (di\ f)) = \Diamond (di\ (\Diamond (di\ f)))$
by (*rule DaEqvDtDi*)
have 9: $\vdash da\ (da\ f) = da\ (\Diamond (di\ f))$
using 1 **by** (*rule DaEqvDa*)
from 7 8 9 **show** *?thesis* **by** *fastforce*
qed

lemma *BaEqvBaBa*:

$\vdash ba\ f = ba\ (ba\ f)$
proof –
have 1: $\vdash da\ (\neg f) = da\ (da\ (\neg f))$ **by** (*rule DaEqvDaDa*)
have 2: $\vdash da\ (da\ (\neg f)) = (\neg (ba\ (\neg (da\ (\neg f)))))$ **by** (*rule DaEqvNotBaNot*)
have 3: $\vdash (\neg (da\ (da\ (\neg f)))) = ba\ (\neg (da\ (\neg f)))$ **by** (*auto simp: ba-d-def*)
have 4: $\vdash (\neg (da\ (\neg f))) = ba\ (\neg (da\ (\neg f)))$ **using** 1 2 3 **by** *fastforce*
from 4 **show** *?thesis* **by** (*metis ba-d-def*)
qed

lemma *BaLeftChopImpChop*:

$\vdash ba\ (f \longrightarrow f1) \longrightarrow f; g \longrightarrow f1; g$
proof –
have 1: $\vdash ba\ (f \longrightarrow f1) \longrightarrow bi\ (f \longrightarrow f1)$ **by** (*rule BalmpBi*)
have 2: $\vdash bi\ (f \longrightarrow f1) \longrightarrow f; g \longrightarrow f1; g$ **by** (*rule BiChopImpChop*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *BaRightChopImpChop*:

$\vdash ba\ (g \longrightarrow g1) \longrightarrow f; g \longrightarrow f; g1$
proof –
have 1: $\vdash ba\ (g \longrightarrow g1) \longrightarrow \Box(g \longrightarrow g1)$ **by** (*rule BalmpBt*)
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 *ChopAndBalmpImport*:

$\vdash (f; f1) \wedge ba\ g \longrightarrow (f \wedge g); (f1 \wedge g)$
proof –
have 1: $\vdash ba\ g \wedge (f; f1) \longrightarrow (g \wedge f); (g \wedge f1)$ **by** (*rule BaAndChopImport*)
have 2: $\vdash (g \wedge f); (g \wedge f1) = (f \wedge g); (f1 \wedge g)$ **by** (*rule AndChopAndCommute*)
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndChopImportA*:
 $\vdash ba\ f \wedge g; g1 \longrightarrow (f \wedge g); g1$
by (*meson BaAndChopImport ChopAndB lift-imp-trans*)

lemma *BaAndChopImportB*:
 $\vdash ba\ f \wedge g; g1 \longrightarrow (f \wedge g); (ba\ f \wedge g1)$
proof —
have 1: $\vdash ba\ f = ba\ (ba\ f)$
by (*simp add: BaEqvBaBa*)
have 2: $\vdash ba\ (ba\ f) \wedge g; g1 \longrightarrow g; (ba\ f \wedge g1)$
by (*metis AndChopB BaAndChopImport lift-imp-trans*)
have 3: $\vdash ba\ f \wedge g; (ba\ f \wedge g1) \longrightarrow (f \wedge g); (ba\ f \wedge g1)$
by (*simp add: BaAndChopImportA*)
from 1 2 3 **show** ?thesis **by** *fastforce*
qed

lemma *BalmpBalmpBaAnd*:
 $\vdash ba\ h \longrightarrow ba(g \longrightarrow ba\ h \wedge g)$
proof —
have 1: $\vdash ba\ h \longrightarrow (g \longrightarrow ba\ h \wedge g)$ **by** *fastforce*
hence 2: $\vdash ba(ba\ h) \longrightarrow ba(g \longrightarrow ba\ h \wedge g)$ **by** (*rule BalmpBa*)
have 3: $\vdash ba\ h = ba(ba\ h)$ **by** (*rule BaEqvBaBa*)
from 2 3 **show** ?thesis **by** *fastforce*
qed

lemma *BaChopImpChopBa*:
 $\vdash ba\ f \longrightarrow g; g1 \longrightarrow g; ((ba\ f) \wedge g1)$
proof —
have 1: $\vdash ba\ f \longrightarrow ba\ (g1 \longrightarrow (ba\ f) \wedge g1)$ **by** (*rule BalmpBalmpBaAnd*)
have 2: $\vdash ba\ (g1 \longrightarrow ba\ f \wedge g1) \longrightarrow g; g1 \longrightarrow g; (ba\ f \wedge g1)$ **by** (*rule BaRightChopImpChop*)
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *DiNotBalmpNotBa*:
 $\vdash di\ (\neg (ba\ f)) \longrightarrow \neg (ba\ f)$
proof —
have 1: $\vdash ba\ f = ba\ (ba\ f)$ **by** (*rule BaEqvBaBa*)
have 2: $\vdash ba\ (ba\ f) \longrightarrow bi\ (ba\ f)$ **by** (*rule BalmpBi*)
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 DiNotBalmpNotBa*)

from 1 2 show ?thesis using lift-imp-trans by blast
qed

lemma DiamondFinImpFin:

$\vdash \Diamond (fin\ f) \longrightarrow fin\ f$

proof —

have 1: $\vdash fin\ f = \#True;(f \wedge empty)$

by (rule FinEqvTrueChopAndEmpty)

hence 2: $\vdash \Diamond (fin\ f) = finite;(\#True;(f \wedge empty))$

by (metis FiniteChopFiniteEqvFinite LeftChopEqvChop inteq-reflection sometimes-d-def)

have 3: $\vdash finite;(\#True;(f \wedge empty)) = (finite;\#True);(f \wedge empty)$

by (rule ChopAssoc)

have 4: $\vdash (finite;\#True);(f \wedge empty) \longrightarrow \#True;(f \wedge empty)$

using 1 2 3 DiamondFin by fastforce

from 1 2 3 4 show ?thesis by fastforce

qed

lemma ChopFinImpFin:

$\vdash (f \wedge finite); fin\ (init\ w) \longrightarrow fin\ (init\ w)$

proof —

have 1: $\vdash (f \wedge finite); fin\ (init\ w) \longrightarrow \Diamond (fin\ (init\ w))$ **by** (rule FiniteChopImpDiamond)

have 2: $\vdash \Diamond (fin\ (init\ w)) \longrightarrow fin\ (init\ w)$ **by** (rule DiamondFinImpFin)

from 1 2 show ?thesis using lift-imp-trans by blast

qed

lemma FinImpYieldsFin:

$\vdash fin\ (init\ w) \wedge finite \longrightarrow (f \wedge finite) yields\ (fin\ (init\ w) \wedge finite)$

proof —

have 1: $\vdash (f \wedge finite); (fin\ (init\ (\neg w)) \wedge finite) \longrightarrow (fin\ (init\ (\neg w)) \wedge finite)$

by (metis (no-types, lifting) ChopAndB FiniteChopEqvDiamond FiniteChopFinExportA
FiniteChopFiniteEqvFinite FiniteChopImpDiamond Prop12 inteq-reflection
lift-and-com lift-imp-trans)

have 2: $\vdash (\neg (fin\ (init\ w)) \wedge finite) = (fin\ (init\ (\neg w)) \wedge finite)$

using FinNotStateEqvNotFinState by fastforce

hence 3: $\vdash (f \wedge finite); (\neg (fin\ (init\ w)) \wedge finite) =$

$(f \wedge finite); (fin\ (init\ (\neg w)) \wedge finite)$

using RightChopEqvChop by blast

have 4: $\vdash (f \wedge finite); (\neg (fin\ (init\ w)) \wedge finite) \longrightarrow (\neg (fin\ (init\ w)) \wedge finite)$

using 1 2 3 by fastforce

hence 5: $\vdash fin\ (init\ w) \wedge finite \longrightarrow \neg ((f \wedge finite); (\neg (fin\ (init\ w)) \wedge finite))$

by fastforce

from 5 show ?thesis

by (smt AndYieldsB BoxElim BoxEqvFiniteYields FinImpBox

FiniteChopInfEqvInf NotChopEqvYieldsNot

YieldsAndYieldsEqvYieldsAnd finite-d-def int-iff1 inteq-reflection)

qed

lemma ChopAndFin:

$\vdash ((f; g) \wedge (\text{fin } (\text{init } w) \wedge \text{finite})) = (f \wedge \text{finite}); (g \wedge (\text{fin } (\text{init } w) \wedge \text{finite}))$
proof –
have 1: $\vdash \text{fin } (\text{init } w) \wedge \text{finite} \longrightarrow (f \wedge \text{finite}) \text{ yields } (\text{fin } (\text{init } w) \wedge \text{finite})$
by (*rule FinImpYieldsFin*)
have 10: $\vdash ((f; g) \wedge (\text{fin } (\text{init } w) \wedge \text{finite})) =$
 $((f \wedge \text{finite}); (g \wedge \text{finite})) \wedge \text{fin } (\text{init } w) \wedge \text{finite}$
using *ChopAndFiniteDist*
by (*smt Prop11 Prop12 inteq-reflection lift-and-com*)
have 2: $\vdash (f; g) \wedge (\text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow$
 $((f \wedge \text{finite}); (g \wedge \text{finite})) \wedge (f \wedge \text{finite}) \text{ yields } (\text{fin } (\text{init } w) \wedge \text{finite})$
using 1 10 **by** *fastforce*
have 3: $\vdash ((f \wedge \text{finite}); (g \wedge \text{finite})) \wedge (f \wedge \text{finite}) \text{ yields } (\text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow$
 $(f \wedge \text{finite}); ((g \wedge \text{finite}) \wedge (\text{fin } (\text{init } w) \wedge \text{finite}))$
using *ChopAndYieldsImp* **by** *blast*
have 30: $\vdash ((g \wedge \text{finite}) \wedge (\text{fin } (\text{init } w) \wedge \text{finite})) = (g \wedge \text{fin } (\text{init } w) \wedge \text{finite})$
by *auto*
have 4: $\vdash (f; g) \wedge (\text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow (f \wedge \text{finite}); (g \wedge \text{fin } (\text{init } w) \wedge \text{finite})$
using 2 3 30
by (*metis (mono-tags, lifting) inteq-reflection lift-imp-trans*)
have 11: $\vdash (f \wedge \text{finite}); (g \wedge \text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow (f \wedge \text{finite}); (g \wedge \text{finite})$
using *ChopAndA* **by** (*metis 30 inteq-reflection*)
have 12: $\vdash (f \wedge \text{finite}); (g \wedge \text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow$
 $(f \wedge \text{finite}); (\text{fin } (\text{init } w) \wedge \text{finite})$
by (*rule ChopAndB*)
have 13: $\vdash (f \wedge \text{finite}); (\text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow \Diamond (\text{fin } (\text{init } w) \wedge \text{finite})$
using *FiniteChopImpDiamond* **by** *blast*
have 14: $\vdash \Diamond (\text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow \text{fin } (\text{init } w)$
by (*metis ChopAndA DiamondFin inteq-reflection sometimes-d-def*)
have 15: $\vdash (f \wedge \text{finite}); (g \wedge \text{fin } (\text{init } w) \wedge \text{finite}) \longrightarrow$
 $((f \wedge \text{finite}); (g \wedge \text{finite})) \wedge \text{fin } (\text{init } w)$
using 11 12 13 14 **by** *fastforce*
from 4 15 **show** ?thesis **by** (*metis ChopAndFiniteDist Prop12 int-iff1 inteq-reflection*)
qed

lemma *ChopAndNotFin*:

$\vdash (f; g \wedge \neg (\text{fin } (\text{init } w)) \wedge \text{finite}) = (f \wedge \text{finite}); (g \wedge \neg (\text{fin } (\text{init } w)) \wedge \text{finite})$
proof –
have 1: $\vdash (f; g \wedge \text{fin } (\text{init } (\neg w)) \wedge \text{finite}) =$
 $(f \wedge \text{finite}); (g \wedge \text{fin } (\text{init } (\neg w)) \wedge \text{finite})$
by (*rule ChopAndFin*)
have 2: $\vdash (\text{fin } (\text{init } (\neg w)) \wedge \text{finite}) = (\neg (\text{fin } (\text{init } w))) \wedge \text{finite}$
using *FinNotStateEqvNotFinState* **by** *fastforce*
hence 3: $\vdash (g \wedge \text{fin } (\text{init } (\neg w)) \wedge \text{finite}) = (g \wedge \neg (\text{fin } (\text{init } w)) \wedge \text{finite})$
by *auto*
hence 4: $\vdash (f \wedge \text{finite}); (g \wedge \text{fin } (\text{init } (\neg w)) \wedge \text{finite}) =$
 $(f \wedge \text{finite}); (g \wedge \neg (\text{fin } (\text{init } w)) \wedge \text{finite})$
by (*rule RightChopEqvChop*)
from 1 2 4 **show** ?thesis **by** *fastforce*
qed

lemma *FinChopChain*:

$\vdash (((init\ w) \wedge finite \longrightarrow fin\ (init\ w1)));$
 $((init\ w1) \wedge finite \longrightarrow fin\ (init\ w2)))$
 $\wedge finite$
 $\longrightarrow (((init\ w) \wedge finite \longrightarrow fin\ (init\ w2)))$

proof –

have 1: $\vdash (init\ w) \wedge finite \wedge$
 $((init\ w) \wedge finite \longrightarrow fin\ (init\ w1)); ((init\ w1) \wedge finite \longrightarrow fin\ (init\ w2))$
 \longrightarrow
 $((init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow fin\ (init\ w1)));$
 $((init\ w1) \wedge finite \longrightarrow fin\ (init\ w2)) \wedge finite$
by (*metis* (*no-types*, *lifting*) *ChopAndFiniteDist StateAndChop int-iffD2*
inteq-reflection lift-and-com)

have 2: $\vdash (init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow fin\ (init\ w1)) \longrightarrow fin\ (init\ w1) \wedge finite$
by *auto*

have 3: $\vdash ((init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow fin\ (init\ w1)));$
 $((init\ w1) \wedge finite \longrightarrow fin\ (init\ w2)) \wedge finite$
 \longrightarrow
 $(fin\ (init\ w1) \wedge finite); (((init\ w1) \wedge finite \longrightarrow fin\ (init\ w2)) \wedge finite)$
using 2 *LeftChopImpChop* **by** *blast*

have 4: $\vdash (fin\ (init\ w1) \wedge finite); (((init\ w1) \wedge finite \longrightarrow fin\ (init\ w2)) \wedge finite) =$
 $\Diamond((init\ w1) \wedge ((init\ w1) \wedge finite \longrightarrow fin\ (init\ w2)) \wedge finite)$
using *FinChopEqvDiamond* **by** *blast*

have 41: $\vdash ((init\ w1) \wedge finite \wedge ((init\ w1) \wedge finite \longrightarrow fin\ (init\ w2))) \longrightarrow fin\ (init\ w2)$
by *auto*

have 42: $\vdash \Diamond((init\ w1) \wedge finite \wedge ((init\ w1) \wedge finite \longrightarrow fin\ (init\ w2))) \longrightarrow \Diamond(fin\ (init\ w2))$
using 41 *DiamondImpDiamond* **by** *blast*

have 5: $\vdash \Diamond(fin\ (init\ w2)) \longrightarrow fin\ (init\ w2)$
using *DiamondFinImpFin* **by** *blast*

have 6: $\vdash (init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow fin\ (init\ w1));$
 $((init\ w1) \wedge finite \longrightarrow fin\ (init\ w2))$
 $\longrightarrow fin\ (init\ w2)$
using 1 3 4 5 42 **by** (*smt Prop10 Prop12 inteq-reflection lift-and-com*)

from 6 **show** *?thesis* **by** *fastforce*

qed

lemma *ChopRule*:

assumes $\vdash (init\ w) \wedge f \wedge finite \longrightarrow fin\ (init\ w1)$
 $\vdash (init\ w1) \wedge f1 \wedge finite \longrightarrow fin\ (init\ w2)$
shows $\vdash (init\ w) \wedge (f; f1) \wedge finite \longrightarrow fin\ (init\ w2)$

proof –

have 1: $\vdash (init\ w) \wedge (f; f1) \wedge finite \longrightarrow ((init\ w) \wedge f \wedge finite); (f1 \wedge finite)$
using *StateAndChopImport*
by (*metis* *ChopAndFiniteDist inteq-reflection*)

have 2: $\vdash (init\ w) \wedge f \wedge finite \longrightarrow fin\ (init\ w1) \wedge finite$ **using** *assms* **by** *auto*

hence 3: $\vdash ((init\ w) \wedge f \wedge finite); (f1 \wedge finite) \longrightarrow (fin\ (init\ w1) \wedge finite); (f1 \wedge finite)$
by (*rule LeftChopImpChop*)

have 4: $\vdash (fin\ (init\ w1) \wedge finite); (f1 \wedge finite) = \Diamond((init\ w1) \wedge f1 \wedge finite)$
by (*rule FinChopEqvDiamond*)

have 5: $\vdash (\text{init } w1) \wedge f1 \wedge \text{finite} \longrightarrow \text{fin } (\text{init } w2)$ **using** *assms* **by** *auto*
hence 6: $\vdash \Diamond((\text{init } w1) \wedge f1 \wedge \text{finite}) \longrightarrow \Diamond(\text{fin } (\text{init } w2))$ **by** (*rule DiamondImpDiamond*)
have 7: $\vdash \Diamond(\text{fin } (\text{init } w2)) \longrightarrow \text{fin } (\text{init } w2)$ **using** *DiamondFinImpFin* **by** *blast*
from 1 3 4 6 7 **show** *?thesis* **by** *fastforce*
qed

lemma *ChopRep*:

assumes $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow f1 \wedge \text{fin } (\text{init } w1)$
 $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1$
shows $\vdash (\text{init } w) \wedge (f; g) \wedge \text{finite} \longrightarrow ((f1 \wedge \text{finite}); g1)$
proof –
have 1: $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow ((f1 \wedge \text{finite}) \wedge \text{fin } (\text{init } w1))$ **using** *assms* **by** *auto*
hence 2: $\vdash (\text{init } w) \wedge ((f \wedge \text{finite}); (g \wedge \text{finite})) \longrightarrow$
 $((f1 \wedge \text{finite}) \wedge \text{fin } (\text{init } w1)); (g \wedge \text{finite})$
using *StateAndChopImpChopRule* **by** *blast*
have 3: $\vdash ((f1 \wedge \text{finite}) \wedge \text{fin } (\text{init } w1)); (g \wedge \text{finite}) =$
 $(f1 \wedge \text{finite}); ((\text{init } w1) \wedge (g \wedge \text{finite}))$
using *AndFinChopEqvStateAndChop* **by** *blast*
have 4: $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1$ **using** *assms* **by** *auto*
hence 5: $\vdash (f1 \wedge \text{finite}); ((\text{init } w1) \wedge g \wedge \text{finite}) \longrightarrow (f1 \wedge \text{finite}); g1$
using *RightChopImpChop* **by** *blast*
from 2 3 5 **show** *?thesis* **using** *ChopAndFiniteDist* **by** *fastforce*
qed

lemma *ChopRepAndFin*:

assumes $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow f1 \wedge \text{fin } (\text{init } w1)$
 $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1 \wedge \text{fin } (\text{init } w2)$
shows $\vdash (\text{init } w) \wedge (f; g) \wedge \text{finite} \longrightarrow ((f1 \wedge \text{finite}); g1) \wedge \text{fin } (\text{init } w2)$
proof –
have 1: $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow f1 \wedge \text{fin } (\text{init } w1)$ **using** *assms* **by** *auto*
have 2: $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1 \wedge \text{fin } (\text{init } w2)$ **using** *assms* **by** *auto*
have 3: $\vdash (\text{init } w) \wedge (f; g) \wedge \text{finite} \longrightarrow (f1 \wedge \text{finite}); (g1 \wedge \text{fin } (\text{init } w2))$
using 1 2 **by** (*rule ChopRep*)
have 4: $\vdash (f1 \wedge \text{finite}); (g1 \wedge \text{fin } (\text{init } w2)) \longrightarrow (f1 \wedge \text{finite}); g1$ **by** (*rule ChopAndA*)
have 5: $\vdash (f1 \wedge \text{finite}); (g1 \wedge \text{fin } (\text{init } w2)) \longrightarrow (f1 \wedge \text{finite}); \text{fin } (\text{init } w2)$
by (*rule ChopAndB*)
have 6: $\vdash (f1 \wedge \text{finite}); \text{fin } (\text{init } w2) \longrightarrow \text{fin } (\text{init } w2)$ **by** (*rule ChopFinImpFin*)
from 1 2 3 4 5 6 **show** *?thesis* **using** *ChopRep ChopRule* **by** *fastforce*
qed

lemma *TrueChopMoreEqvMore*:

$\vdash \# \text{True} ; \text{more} = \text{more}$
by (*metis ChopMoreImpMore EmptyImpFinite FiniteAndEmptyEqvEmpty FiniteChopMoreEqvMore*
LeftChopImpChop Prop09 int-eq-true int-iff1 inteq-reflection)

lemma *FiniteChopFmoreEqvFmore*:

$\vdash \text{finite}; \text{fmore} = \text{fmore}$
by (*metis TrueChopAndFiniteEqvAndFiniteChopFinite TrueChopMoreEqvMore fmore-d-def inteq-reflection*)

```

lemma MoreChopLoop:
  assumes  $\vdash f \longrightarrow fmore ; f$ 
  shows  $\vdash finite \longrightarrow \neg f$ 
proof -
  have 1:  $\vdash f \longrightarrow fmore ; f$ 
    using assms by auto
  hence 11:  $\vdash \Diamond (f) \longrightarrow \Diamond (fmore;f)$ 
    using DiamondImpDiamond by blast
  have 12:  $\vdash \Diamond (fmore;f) = finite;(fmore;f)$ 
    by (simp add: sometimes-d-def)
  have 13:  $\vdash finite;(fmore;f) = (finite;fmore);f$ 
    by (rule ChopAssoc)
  have 14:  $\vdash \Diamond (fmore;f) = fmore;f$ 
    using FiniteChopFmoreEqvFmore 12 13 by (metis int-eq)
  have 2:  $\vdash fmore ; f = \bigcirc(\Diamond f)$ 
    using MoreChopEqvNextDiamond by blast
  have 3:  $\vdash \Diamond (f) \longrightarrow \bigcirc(\Diamond f)$ 
    using 11 14 2 by fastforce
  hence 4:  $\vdash finite \longrightarrow \neg(\Diamond f)$ 
    using NextLoop by blast
  have 5:  $\vdash \neg(\Diamond f) \longrightarrow \neg f$ 
    using NowImpDiamond by fastforce
  from 4 5 show ?thesis using lift-imp-trans by blast
qed

lemma MoreChopContra:
  assumes  $\vdash f \wedge \neg g \longrightarrow (fmore ; (f \wedge \neg g))$ 
  shows  $\vdash f \wedge finite \longrightarrow g$ 
proof -
  have 1:  $\vdash f \wedge \neg g \longrightarrow (fmore ; (f \wedge \neg g))$  using assms by auto
  hence 2:  $\vdash finite \longrightarrow \neg(f \wedge \neg g)$  by (rule MoreChopLoop)
  from 2 show ?thesis
    by (simp add: Valid-def finite-defs infinite-defs sum.case-eq-if)
qed

```

```

lemma MoreChopLoopFinite:
  assumes  $\vdash f \wedge finite \longrightarrow fmore ; f$ 
  shows  $\vdash finite \longrightarrow \neg f$ 
proof -
  have 1:  $\vdash f \wedge finite \longrightarrow fmore ; f$ 
    using assms by auto
  hence 11:  $\vdash \Diamond (f \wedge finite) \longrightarrow \Diamond (fmore;f)$ 
    using DiamondImpDiamond by blast
  have 12:  $\vdash \Diamond (fmore;f) = finite;(fmore;f)$ 
    by (simp add: sometimes-d-def)
  have 13:  $\vdash finite;(fmore;f) = (finite;fmore);f$ 
    by (rule ChopAssoc)
  have 14:  $\vdash \Diamond (fmore;f) = fmore;f$ 
    using FiniteChopFmoreEqvFmore 12 13 by (metis int-eq)

```

```

have 2:  $\vdash f_{\text{more}} ; f = \bigcirc(\Diamond f)$ 
  using MoreChopEqvNextDiamond by blast
have 3:  $\vdash \Diamond (f \wedge \text{finite}) \longrightarrow \bigcirc(\Diamond f)$ 
  using 11 14 2 by fastforce
have 31:  $\vdash \Diamond (f \wedge \text{finite}) = ((\Diamond f) \wedge \text{finite})$ 
  by (smt 3 ChopAndB ChopAndNotChopImp FiniteChopFiniteEqvFinite FiniteChopInfEqvInf
      NextDiamondImpDiamond Prop11 Prop12 finite-d-def infinite-d-def inteq-reflection
      lift-imp-trans sometimes-d-def)
have 32:  $\vdash (\Diamond f) \wedge \text{finite} \longrightarrow \bigcirc(\Diamond f)$ 
  using 3 31 by fastforce
hence 4:  $\vdash \text{finite} \longrightarrow \neg (\Diamond f)$ 
  by (metis (no-types, lifting) DiamondIntro FiniteChopInfEqvInf InfEqvNotFinite Prop09
      finite-d-def int-simps(15) int-simps(32) inteq-reflection sometimes-d-def)
have 5:  $\vdash \neg (\Diamond f) \longrightarrow \neg f$ 
  by (simp add: NowImpDiamond)
from 4 5 show ?thesis using lift-imp-trans by fastforce
qed

```

lemma *MoreChopEqvFmoreOrInf*:

$\vdash \text{more} ; f = ((f_{\text{more}};f) \vee \text{inf})$

by (simp add: Valid-def more-defs fmore-defs chop-defs infinite-defs sum.case-eq-if)

lemma *MoreChopLoopFiniteB*:

assumes $\vdash f \longrightarrow \text{more} ; f$

shows $\vdash \text{finite} \longrightarrow \neg f$

proof –

have 1: $\vdash f \longrightarrow \text{more} ; f$

using assms by auto

have 10: $\vdash f \longrightarrow (f_{\text{more}};f) \vee \text{inf}$

using MoreChopEqvFmoreOrInf assms by fastforce

hence 100: $\vdash f \wedge \text{finite} \longrightarrow (f_{\text{more}};f)$

by (simp add: Prop13 finite-d-def)

hence 11: $\vdash \Diamond (f \wedge \text{finite}) \longrightarrow \Diamond (f_{\text{more}};f)$

using DiamondImpDiamond by blast

have 12: $\vdash \Diamond (f_{\text{more}};f) = \text{finite};(f_{\text{more}};f)$

by (simp add: sometimes-d-def)

have 13: $\vdash \text{finite};(f_{\text{more}};f) = (\text{finite};f_{\text{more}});f$

by (rule ChopAssoc)

have 14: $\vdash \Diamond (f_{\text{more}};f) = f_{\text{more}};f$

using FiniteChopFmoreEqvFmore 12 13 by (metis int-eq)

have 2: $\vdash f_{\text{more}} ; f = \bigcirc(\Diamond f)$

using MoreChopEqvNextDiamond by blast

have 3: $\vdash \Diamond (f \wedge \text{finite}) \longrightarrow \bigcirc(\Diamond f)$

using 11 14 2 by fastforce

have 31: $\vdash \Diamond (f \wedge \text{finite}) = ((\Diamond f) \wedge \text{finite})$

by (smt 3 ChopAndB ChopAndNotChopImp FiniteChopFiniteEqvFinite FiniteChopInfEqvInf
 NextDiamondImpDiamond Prop11 Prop12 finite-d-def infinite-d-def inteq-reflection
 lift-imp-trans sometimes-d-def)

have 32: $\vdash (\Diamond f) \wedge \text{finite} \longrightarrow \bigcirc(\Diamond f)$

```

using 3 31 by fastforce
hence 4:  $\vdash \text{finite} \longrightarrow \neg (\Diamond f)$ 
  by (metis (no-types, lifting) DiamondIntro FiniteChopInfEqvInf InfEqvNotFinite Prop09
    finite-d-def int-simps(15) int-simps(32) inteq-reflection sometimes-d-def)
have 5:  $\vdash \neg (\Diamond f) \longrightarrow \neg f$ 
  by (simp add: NowImpDiamond)
from 4 5 show ?thesis using lift-imp-trans by fastforce
qed

```

```

lemma MoreChopContraFinite:
  assumes  $\vdash (f \wedge \neg g) \wedge \text{finite} \longrightarrow (f \text{ more} ; (f \wedge \neg g))$ 
  shows  $\vdash f \wedge \text{finite} \longrightarrow g$ 
proof -
  have 1:  $\vdash (f \wedge \neg g) \wedge \text{finite} \longrightarrow (f \text{ more} ; (f \wedge \neg g))$  using assms by auto
  hence 2:  $\vdash \text{finite} \longrightarrow \neg (f \wedge \neg g)$  using MoreChopLoopFinite by (simp add: MoreChopLoopFinite)
  from 2 show ?thesis by (simp add: Valid-def)
qed

```

```

lemma MoreChopContraFiniteB:
  assumes  $\vdash (f \wedge \neg g) \longrightarrow (f \text{ more} ; (f \wedge \neg g))$ 
  shows  $\vdash f \wedge \text{finite} \longrightarrow g$ 
proof -
  have 1:  $\vdash (f \wedge \neg g) \longrightarrow (f \text{ more} ; (f \wedge \neg g))$  using assms by auto
  hence 2:  $\vdash \text{finite} \longrightarrow \neg (f \wedge \neg g)$  using MoreChopLoopFinite by (simp add: MoreChopLoopFiniteB)
  from 2 show ?thesis by (simp add: Valid-def)
qed

```

```

lemma ChopLoop:
  assumes  $\vdash f \longrightarrow g; f$ 
   $\vdash g \longrightarrow f \text{ more}$ 
  shows  $\vdash \text{finite} \longrightarrow \neg f$ 
proof -
  have 1:  $\vdash f \longrightarrow g; f$  using assms by auto
  have 2:  $\vdash g \longrightarrow f \text{ more}$  using assms by auto
  hence 3:  $\vdash g; f \longrightarrow f \text{ more} ; f$  by (rule LeftChopImpChop)
  have 4:  $\vdash f \longrightarrow f \text{ more} ; f$  using 1 3 by fastforce
  from 4 show ?thesis using MoreChopLoop by auto
qed

```

```

lemma ChopLoopB:
  assumes  $\vdash f \longrightarrow g; f$ 
   $\vdash g \longrightarrow \text{more}$ 
  shows  $\vdash \text{finite} \longrightarrow \neg f$ 
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 MoreChopLoopFiniteB by auto

```

qed

lemma *ChopContra*:

assumes $\vdash f \wedge \neg g \longrightarrow h; f \wedge \neg (h; g)$

$\vdash h \longrightarrow fmore$

shows $\vdash f \wedge finite \longrightarrow g$

proof —

have 1: $\vdash f \wedge \neg g \longrightarrow h; f \wedge \neg (h; g)$ **using** *assms* **by** *auto*

have 2: $\vdash h \longrightarrow fmore$ **using** *assms* **by** *auto*

have 3: $\vdash h; f \wedge \neg (h; g) \longrightarrow h; (f \wedge \neg g)$ **by** (*rule ChopAndNotChopImp*)

have 4: $\vdash h; (f \wedge \neg g) \longrightarrow fmore ; (f \wedge \neg g)$ **using** 2 **by** (*rule LeftChopImpChop*)

have 5: $\vdash f \wedge \neg g \longrightarrow fmore ; (f \wedge \neg g)$ **using** 1 3 4 **by** *fastforce*

from 5 **show** *?thesis* **using** *MoreChopContra* **by** *auto*

qed

lemma *ChopContraB*:

assumes $\vdash f \wedge \neg g \longrightarrow h; f \wedge \neg (h; g)$

$\vdash h \longrightarrow more$

shows $\vdash f \wedge finite \longrightarrow g$

proof —

have 1: $\vdash f \wedge \neg g \longrightarrow h; f \wedge \neg (h; g)$ **using** *assms* **by** *auto*

have 2: $\vdash h \longrightarrow more$ **using** *assms* **by** *auto*

have 3: $\vdash h; f \wedge \neg (h; g) \longrightarrow h; (f \wedge \neg g)$ **by** (*rule ChopAndNotChopImp*)

have 4: $\vdash h; (f \wedge \neg g) \longrightarrow more ; (f \wedge \neg g)$ **using** 2 **by** (*rule LeftChopImpChop*)

have 5: $\vdash f \wedge \neg g \longrightarrow more ; (f \wedge \neg g)$ **using** 1 3 4 **by** *fastforce*

from 5 **show** *?thesis* **using** *MoreChopContraFiniteB* **by** *auto*

qed

9.8 Properties of Chopstar and Chopplus

lemma *FPowerstardef*:

$\vdash fpowerstar f = (\exists n. power f n)$

by (*simp add: fpowerstar-d-def*)

lemma *Powerstardef*:

$\vdash powerstar f = (fpowerstar f);(empty \vee (f \wedge inf))$

by (*simp add: fpowerstar-d-def powerstar-d-def*)

lemma *Chopstardef*:

$\vdash chopstar f = powerstar (f \wedge more)$

by (*simp add: chopstar-d-def*)

lemma *AndEmptyChopAndEmptyEqvAndEmpty*:

$\vdash (f \wedge empty);(f \wedge empty) = (f \wedge empty)$

by (*simp add: Valid-def empty-defs chop-defs sum.case-eq-if,*
smt interval-prefix-intlen interval-prefix-length interval-suffix-zero
le-numeral-extra(3) sum.collapse(1)))

lemma *PowerCommute*:

$\vdash (f \wedge \text{finite}) ; \text{power } f \ n = \text{power } f \ n ; (f \wedge \text{finite})$

proof

(*induct n*)

case 0

then show ?case

by (*metis ChopEmpty EmptyChop inteq-reflection power-d.pow-0*)

next

case (*Suc n*)

then show ?case

by (*metis ChopAssoc inteq-reflection power-d.pow-Suc*)

qed

lemma *ChopInductL*:

assumes $\vdash g \vee f ; h \longrightarrow h$

shows $\vdash (\text{power } f \ n) ; g \longrightarrow h$

proof

(*induct n*)

case 0

then show ?case **using** *EmptyChop assms*

by (*metis MP Prop12 int-eq int-iffD1 int-simps(33) pow-0*)

next

case (*Suc n*)

then show ?case **using** *assms*

by (*smt AndChopA ChopAndA ChopAssoc Prop03 Prop10 inteq-reflection lift-and-com lift-imp-trans pow-Suc*)

qed

lemma *ChopInductFiniteL*:

assumes $\vdash g \vee (f \wedge \text{finite}) ; h \longrightarrow h$

shows $\vdash (\text{power } f \ n) ; g \longrightarrow h$

proof

(*induct n*)

case 0

then show ?case **using** *EmptyChop assms*

by (*metis MP Prop12 int-eq int-iffD1 int-simps(33) pow-0*)

next

case (*Suc n*)

then show ?case **using** *assms*

by (*smt AndChopA ChopAndA ChopAssoc Prop03 Prop10 inteq-reflection lift-and-com lift-imp-trans pow-Suc*)

qed

lemma *ChopInductFiniteMoreL*:

assumes $\vdash g \vee ((f \wedge \text{more}) \wedge \text{finite}) ; h \longrightarrow h$

shows $\vdash (\text{power } f \ n) ; g \longrightarrow h$

proof

(*induct n*)

case 0

then show ?case **using** *assms* **by** (*metis ChopInductFiniteL pow-0*)

next


```

case (Suc n)
then show ?case
proof -
have 1:  $\vdash \text{power } f \text{ (Suc } n); g = ((f \wedge \text{finite}); \text{power } f \text{ } n); g$ 
  by simp
have 2:  $\vdash ((f \wedge \text{finite}); \text{power } f \text{ } n); g = (f \wedge \text{finite}); ((\text{power } f \text{ } n); g)$ 
  by (meson ChopAssoc Prop11)
have 3:  $\vdash (f \wedge \text{finite}); ((\text{power } f \text{ } n); g) \longrightarrow (f \wedge \text{finite}); h$ 
  by (simp add: RightChopImpChop Suc.hyps)
have 4:  $\vdash (f \wedge \text{finite}); h = ((f \wedge \text{more}) \wedge \text{finite}); h \vee ((f \wedge \text{empty}) \wedge \text{finite}); h$ 
  using neq0-conv
  by (simp add: Valid-def finite-defs chop-defs more-defs empty-defs sum.case-eq-if, blast)
have 5:  $\vdash ((f \wedge \text{more}) \wedge \text{finite}); h \longrightarrow h$  using assms by auto
have 6:  $\vdash ((f \wedge \text{empty}) \wedge \text{finite}); h \longrightarrow h$ 
  by (smt AndChopA AndChopB EmptyChop Prop10 Prop12 inteq-reflection)
from 5 6 4 3 2 1 show ?thesis by fastforce
qed
qed

```

```

lemma ChopInductInfl:
assumes  $\vdash g \vee f; h \longrightarrow h$ 
shows  $\vdash ((\text{power } f \text{ } n); (f \wedge \text{inf})); g \longrightarrow h$ 
proof
(induct n)
case 0
then show ?case using assms
by (metis (no-types, lifting) AndInfChopEqvAndInf ChopAssoc ChopInductFiniteL PowerstarEqvSemhelp3
  Prop03 Prop10 Prop12 inteq-reflection)
next
case (Suc n)
then show ?case using assms
by (metis AndChopA ChopAndB ChopAssoc Prop03 Prop10 int-eq lift-imp-trans pow-Suc)
qed

```

```

lemma ChopInductInfMoreL:
assumes  $\vdash g \vee f; h \longrightarrow h$ 
shows  $\vdash ((\text{power } f \text{ } n); ((f \wedge \text{more}) \wedge \text{inf})); g \longrightarrow h$ 
using ChopInductInfl
by (metis AndMoreAndInfEqvAndInf assms inteq-reflection)

```

```

lemma ChopInductR:
assumes  $\vdash g \vee h; f \longrightarrow h$ 
shows  $\vdash g; (\text{power } f \text{ } n) \longrightarrow h$ 
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 (smt AndChopA ChopAndA ChopAssoc PowerCommute Prop03 Prop10 inteq-reflection lift-and-com
    lift-imp-trans pow-Suc)
qed

```

```

lemma ChopInductInfR:
assumes  $\vdash g \vee h; f \longrightarrow h$ 
shows  $\vdash g; ((\text{power } f \ n); (f \wedge \text{inf})) \longrightarrow h$ 
using assms
by (metis (no-types, hide-lams) AndChopA ChopAndB ChopAssoc ChopInductR Prop05 Prop10
    inteq-reflection lift-and-com lift-imp-trans)

```

```

lemma ChopExistPower:
 $\vdash (g; (\exists n. \text{power } f \ n)) = (\exists n. g; \text{power } f \ n)$ 
using ChopExist by fastforce

```

```

lemma ExistChopPower:
 $\vdash (\exists n. (\text{power } f \ n); g) = (\exists n. \text{power } f \ n); g$ 
using ExistChop by fastforce

```

```

lemma PowerStarCommute:
 $\vdash (f \wedge \text{finite}); (\exists n. \text{power } f \ n) = (\exists n. \text{power } f \ n); (f \wedge \text{finite})$ 
proof -
have 1:  $\vdash (f \wedge \text{finite}); (\exists n. \text{power } f \ n) =$ 
 $(\exists n. (f \wedge \text{finite}); \text{power } f \ n)$ 
using ChopExistPower by blast
have 2:  $\vdash (\exists n. (f \wedge \text{finite}); \text{power } f \ n) =$ 
 $(\exists n. (\text{power } f \ n); (f \wedge \text{finite}))$ 
using PowerCommute by fastforce
have 3:  $\vdash (\exists n. (\text{power } f \ n); (f \wedge \text{finite})) =$ 
 $(\exists n. (\text{power } f \ n); (f \wedge \text{finite}))$ 
using ExistChopPower by blast
from 1 2 3 show ?thesis by fastforce
qed

```

```

lemma PowerSucAndEmptyEqvAndEmpty:
 $\vdash (\text{power } (f \wedge \text{empty}) (\text{Suc } n)) = (f \wedge \text{empty})$ 
proof
(induct n)
case 0
then show ?case using ChopEmpty
by (metis FiniteAndEmptyEqvEmpty Prop11 Prop12 inteq-reflection lift-and-com pow-0 pow-Suc)
next
case (Suc n)
then show ?case
by (smt AndEmptyChopAndEmptyEqvAndEmpty ChopAndA ChopImpChop FiniteAndEmptyEqvEmpty)

```

PowerCommute Prop11 Prop12 inteq-reflection pow-Suc)

qed

lemma FiniteOr:
 $\vdash ((f \vee g) \wedge \text{finite}) = ((f \wedge \text{finite}) \vee (g \wedge \text{finite}))$
by *auto*

lemma PowerOr:
 $\vdash (\text{power } (f \vee g) (\text{Suc } n)) = ((f \wedge \text{finite}); \text{power } (f \vee g) n) \vee ((g \wedge \text{finite}); \text{power } (f \vee g) n)$
by (*simp add: FiniteOr OrChopEqvRule*)

lemma PowerEmptyOrMore:
 $\vdash (\text{power } ((f \wedge \text{empty}) \vee (f \wedge \text{more})) (\text{Suc } n)) = ((f \wedge \text{empty}); (\text{power } ((f \wedge \text{empty}) \vee (f \wedge \text{more})) n) \vee (f \wedge \text{fmore}); (\text{power } ((f \wedge \text{empty}) \vee (f \wedge \text{more})) n))$
using *PowerOr*
by (*smt FiniteAndEmptyEqvEmpty AndMoreAndFiniteEqvAndFmore Prop10 Prop12 int-iffD1 inteq-reflection lift-and-com*)

lemma PSeqvEmptyOrChopPS:
 $\vdash \text{powerstar } f = (\text{empty} \vee f; \text{powerstar } f)$
using *PowerstarEqvSem Valid-def* **by** *blast*

lemma EmptyImpCS:
 $\vdash \text{empty} \longrightarrow f^*$
proof —
have 1: $\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$ **by** (*rule ChopstarEqv*)
have 2: $\vdash \text{empty} \longrightarrow \text{empty} \vee (f \wedge \text{more}); f^*$ **by** *auto*
from 1 2 **show** *?thesis* **by** *fastforce*
qed

lemma CSeqvOrChopCS:
 $\vdash f^* = (\text{empty} \vee (f; f^*))$
proof —
have 1: $\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$ **by** (*rule ChopstarEqv*)
have 2: $\vdash (f \wedge \text{more}); f^* \longrightarrow f; f^*$ **by** (*rule AndChopA*)
have 3: $\vdash f^* \longrightarrow \text{empty} \vee f; f^*$ **using** 1 2 **by** (*metis int-iffD1 Prop08*)
have 4: $\vdash \text{empty} \longrightarrow f^*$ **by** (*rule EmptyImpCS*)
have 5: $\vdash f \longrightarrow \text{empty} \vee (f \wedge \text{more})$ **by** (*auto simp: empty-d-def*)
have 6: $\vdash f; f^* \longrightarrow f^* \vee (f \wedge \text{more}); f^*$ **using** 5 **by** (*rule EmptyOrChopImpRule*)
have 7: $\vdash f^* \longrightarrow \text{empty} \vee (f \wedge \text{more}); f^*$ **using** 1 **by** *fastforce*
have 8: $\vdash f; f^* \longrightarrow \text{empty} \vee (f \wedge \text{more}); f^*$ **using** 6 7 **by** *fastforce*
hence 9: $\vdash f; f^* \longrightarrow f^*$ **using** 1 **by** *fastforce*
have 10: $\vdash \text{empty} \vee f; f^* \longrightarrow f^*$ **using** 9 4 **by** *fastforce*
from 3 10 **show** *?thesis* **by** *fastforce*
qed

lemma *PowerChopCommute*:

$\vdash ((f \wedge \text{more}) \wedge \text{finite}); \text{power } (f \wedge \text{more}) \ n = \text{power } (f \wedge \text{more}) \ n; ((f \wedge \text{more}) \wedge \text{finite})$

using *PowerCommute* **by** *auto*

lemma *ChopExist*:

$\vdash (g; (\exists n. \text{power } (f \wedge \text{more}) \ n)) = (\exists n. g; \text{power } (f \wedge \text{more}) \ n)$

using *ChopExistPower* **by** *auto*

lemma *ExistChop*:

$\vdash (\exists n. (\text{power } (f \wedge \text{more}) \ n); g) = (\exists n. \text{power } (f \wedge \text{more}) \ n); g$

using *ExistChopPower* **by** *auto*

lemma *FPowerstarInductL*:

assumes $\vdash g \vee (f \wedge \text{finite}); h \longrightarrow h$

shows $\vdash (f\text{powerstar } f); g \longrightarrow h$

proof —

have 1: $\vdash (f\text{powerstar } f); g = (\exists n. \text{power } f \ n); g$

by (*simp add: fpowerstar-d-def LeftChopEqvChop*)

have 2: $\vdash (\exists n. \text{power } f \ n); g =$

$(\exists n. (\text{power } f \ n); g)$

using *ExistChopPower* **by** *fastforce*

have 3: $\bigwedge n. \vdash (\text{power } f \ n); g \longrightarrow h$

using *ChopInductFiniteL* *assms* **by** *blast*

have 4: $\vdash (\exists n. ((\text{power } f \ n)); g) \longrightarrow h$

using 3 **by** (*simp add: Valid-def*,

smt ChopAssoc ChopOrEqv EmptyOrChopEqv inteq-reflection unl-lift2)

from 1 2 4 **show** *?thesis* **by** (*metis inteq-reflection*)

qed

lemma *FPowerstarInductMoreL*:

assumes $\vdash g \vee ((f \wedge \text{more}) \wedge \text{finite}); h \longrightarrow h$

shows $\vdash (f\text{powerstar } f); g \longrightarrow h$

proof —

have 1: $\vdash (f\text{powerstar } f); g = (\exists n. \text{power } f \ n); g$

by (*simp add: fpowerstar-d-def LeftChopEqvChop*)

have 2: $\vdash (\exists n. \text{power } f \ n); g =$

$(\exists n. (\text{power } f \ n); g)$

using *ExistChopPower* **by** *fastforce*

have 3: $\bigwedge n. \vdash (\text{power } f \ n); g \longrightarrow h$

using *ChopInductFiniteMoreL* *assms* **by** *blast*

have 4: $\vdash (\exists n. ((\text{power } f \ n)); g) \longrightarrow h$

using 3 **by** (*simp add: Valid-def*,

smt ChopAssoc ChopOrEqv EmptyOrChopEqv inteq-reflection unl-lift2)

from 1 2 4 **show** *?thesis* **by** (*metis inteq-reflection*)

qed

lemma *PowerstarInductL*:

assumes $\vdash g \vee f; h \longrightarrow h$

shows $\vdash (\text{powerstar } f); g \longrightarrow h$

proof –
have 1: $\vdash (\text{powerstar } f);g = ((\exists n. \text{power } f \ n);(\text{empty} \vee (f \wedge \text{inf})));g$
by (simp add: powerstar-d-def LeftChopEqvChop)
have 11: $\vdash ((\exists n. \text{power } f \ n);(\text{empty} \vee (f \wedge \text{inf})));g =$
 $(\exists n. \text{power } f \ n);((\text{empty} \vee (f \wedge \text{inf})));g$
by (meson ChopAssoc Prop11)
have 2: $\vdash (\exists n. \text{power } f \ n);((\text{empty} \vee (f \wedge \text{inf})));g =$
 $(\exists n. ((\text{power } f \ n);((\text{empty} \vee (f \wedge \text{inf})));g))$
using ExistChopPower **by** fastforce
have 3: $\bigwedge n. \vdash (\text{power } f \ n);g \longrightarrow h$
using ChopInductL assms **by** blast
have 31: $\bigwedge n. \vdash ((\text{power } f \ n);(f \wedge \text{inf}));g \longrightarrow h$
using ChopInductInfl assms **by** blast
have 4: $\vdash (\exists n. ((\text{power } f \ n);(\text{empty} \vee (f \wedge \text{inf})));g) \longrightarrow h$
using 3 31 **by** (simp add: Valid-def,
smt ChopAssoc ChopOrEqv EmptyOrChopEqv inteq-reflection unl-lift2)
from 1 11 2 4 **show** ?thesis
by (metis InfiniteSemantics.ExistChop inteq-reflection)
qed

lemma ChopstarInductL:

assumes $\vdash g \vee f;h \longrightarrow h$
shows $\vdash (\text{chopstar } f);g \longrightarrow h$

proof –
have 1: $\vdash (\text{chopstar } f);g = ((\exists n. \text{power } (f \wedge \text{more}) \ n);(\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})));g$
by (simp add: chopstar-d-def powerstar-d-def LeftChopEqvChop)
have 11: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) \ n);(\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})));g =$
 $(\exists n. \text{power } (f \wedge \text{more}) \ n);((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})));g$
by (meson ChopAssoc Prop11)
have 2: $\vdash (\exists n. \text{power } (f \wedge \text{more}) \ n);((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})));g =$
 $(\exists n. (\text{power } (f \wedge \text{more}) \ n);(((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})));g))$
using ExistChopPower **by** fastforce
have 3: $\bigwedge n. \vdash (\text{power } (f \wedge \text{more}) \ n);g \longrightarrow h$
using ChopInductL assms
by (smt AndChopA MP Prop02 Prop12 int-iffD2 int-simps(33) inteq-reflection lift-imp-trans)
have 31: $\bigwedge n. \vdash ((\text{power } (f \wedge \text{more}) \ n);((f \wedge \text{more}) \wedge \text{inf}));g \longrightarrow h$
using assms
by (metis (no-types, lifting) AndChopA ChopInductInfl Prop03 Prop10 int-eq-true int-simps(33)
inteq-reflection lift-imp-trans)
have 4: $\vdash (\exists n. ((\text{power } (f \wedge \text{more}) \ n);(\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})));g) \longrightarrow h$
using 3 31 **by** (simp add: Valid-def,
smt ChopAssoc ChopOrEqv EmptyOrChopEqv inteq-reflection unl-lift2)
from 1 11 2 4 **show** ?thesis
by (metis InfiniteSemantics.ExistChop inteq-reflection)
qed

lemma ChopstarInductMoreL:

assumes $\vdash g \vee (f \wedge \text{more});h \longrightarrow h$
shows $\vdash (\text{chopstar } f);g \longrightarrow h$

proof –

```

have 1:  $\vdash (\text{chopstar } f);g = ((\exists n. \text{power } (f \wedge \text{more}) n);(\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})));g$ 
by (simp add: chopstar-d-def powerstar-d-def LeftChopEqvChop)
have 11:  $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n);(\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})));g =$ 
 $(\exists n. \text{power } (f \wedge \text{more}) n);((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})));g$ 
by (meson ChopAssoc Prop11)
have 2:  $\vdash (\exists n. \text{power } (f \wedge \text{more}) n);((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})));g =$ 
 $(\exists n. (\text{power } (f \wedge \text{more}) n);((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})));g)$ 
using ExistChopPower by fastforce
have 3:  $\bigwedge n. \vdash (\text{power } (f \wedge \text{more}) n);g \longrightarrow h$ 
using ChopInductL assms by (metis)
have 31:  $\vdash (\exists n. (\text{power } (f \wedge \text{more}) n);g) \longrightarrow h$ 
using 3 by fastforce
have 32:  $\bigwedge n. \vdash ((\text{power } (f \wedge \text{more}) n);((f \wedge \text{more}) \wedge \text{inf})));g \longrightarrow h$ 
using assms
by (metis (no-types, lifting) AndChopA ChopInductInfl int-eq-true inteq-reflection)
have 33:  $\vdash (\exists n. ((\text{power } (f \wedge \text{more}) n);((f \wedge \text{more}) \wedge \text{inf})));g \longrightarrow h$ 
using 32 by fastforce
have 34:  $\vdash (\exists n. (\text{power } (f \wedge \text{more}) n);g) \vee$ 
 $(\exists n. ((\text{power } (f \wedge \text{more}) n);((f \wedge \text{more}) \wedge \text{inf})));g \longrightarrow h$ 
using 31 33 by fastforce
have 35:  $\vdash (\exists n. (\text{power } (f \wedge \text{more}) n);g \vee ((\text{power } (f \wedge \text{more}) n);((f \wedge \text{more}) \wedge \text{inf})));g$ 
 $\longrightarrow h$ 
using 34 by fastforce
have 36:  $\bigwedge n. \vdash ((\text{power } (f \wedge \text{more}) n);g \vee ((\text{power } (f \wedge \text{more}) n);((f \wedge \text{more}) \wedge \text{inf})));g =$ 
 $((\text{power } (f \wedge \text{more}) n);(\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})));g$ 
by (metis (no-types, lifting) ChopAssoc ChopOrEqv EmptyOrChopEqv inteq-reflection)
have 4:  $\vdash (\exists n. ((\text{power } (f \wedge \text{more}) n);(\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})));g \longrightarrow h$ 
using 36 35 by fastforce
from 1 11 2 4 show ?thesis
by (metis InfiniteSemantics.ExistChop inteq-reflection)
qed

```

lemma *FPowerstarInductR:*

assumes $\vdash g \vee h; f \longrightarrow h$

shows $\vdash g; (\text{fpowerstar } f) \longrightarrow h$

proof —

have 1: $\vdash g; (\text{fpowerstar } f) = g; (\exists n. \text{power } f n)$

by (*simp add: fpowerstar-d-def*)

have 2: $\vdash (g; (\exists n. \text{power } f n)) = (\exists n. g; (\text{power } f n))$

using *ChopExistPower* **by** *blast*

have 3: $\bigwedge n. \vdash g; (\text{power } f n) \longrightarrow h$

using *ChopInductR assms* **by** *blast*

have 4: $\vdash (\exists n. g; (\text{power } f n)) \longrightarrow h$

using 3 **by** (*simp add: Valid-def,*
smt ChopEmpty ChopOrEqv inteq-reflection unl-lift2)

from 1 2 4 **show** ?thesis **by** (*metis inteq-reflection*)

qed

lemma *PowerstarInductR:*

assumes $\vdash g \vee h; f \longrightarrow h$

```

shows  $\vdash g;(\text{powerstar } f) \longrightarrow h$ 
proof –
have 1:  $\vdash g;(\text{powerstar } f) = g;((\exists n. \text{power } f \ n);(\text{empty} \vee (f \wedge \text{inf})))$ 
by (simp add: chopstar-d-def powerstar-d-def)
have 11:  $\vdash g;((\exists n. \text{power } f \ n);(\text{empty} \vee (f \wedge \text{inf}))) =$ 
 $(g;(\exists n. \text{power } f \ n));(\text{empty} \vee (f \wedge \text{inf}))$ 
using ChopAssoc by blast
have 2:  $\vdash (g;(\exists n. \text{power } f \ n)) = (\exists n. g;(\text{power } f \ n))$ 
using ChopExistPower by blast
hence 21:  $\vdash (g;(\exists n. \text{power } f \ n));(\text{empty} \vee (f \wedge \text{inf})) =$ 
 $(\exists n. g;(\text{power } f \ n));(\text{empty} \vee (f \wedge \text{inf}))$ 
using LeftChopEqvChop by blast
have 3:  $\bigwedge n. \vdash g;(\text{power } f \ n) \longrightarrow h$ 
using ChopInductR assms by blast
have 31:  $\bigwedge n. \vdash g;((\text{power } f \ n);(f \wedge \text{inf})) \longrightarrow h$ 
using ChopInductInfR assms by blast
have 4:  $\vdash (\exists n. g;((\text{power } f \ n);(\text{empty} \vee (f \wedge \text{inf})))) \longrightarrow h$ 
using 3 31 by (simp add: Valid-def,
  smt ChopEmpty ChopOrEqv inteq-reflection unl-lift2)
from 1 11 2 21 4 show ?thesis
by (metis InfiniteSemantics.ChopExist InfiniteSemantics.ExistChop inteq-reflection)
qed

```

```

lemma ChopstarInductR:
assumes  $\vdash g \vee h; f \longrightarrow h$ 
shows  $\vdash g;(\text{chopstar } f) \longrightarrow h$ 
proof –
have 1:  $\vdash g;(\text{chopstar } f) =$ 
 $g;((\exists n. \text{power } (f \wedge \text{more}) \ n);(\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})))$ 
by (simp add: chopstar-d-def powerstar-d-def)
have 11:  $\vdash g;((\exists n. \text{power } (f \wedge \text{more}) \ n);(\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))) =$ 
 $(g;(\exists n. \text{power } (f \wedge \text{more}) \ n));(\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))$ 
using ChopAssoc by blast
have 2:  $\vdash (g;(\exists n. \text{power } (f \wedge \text{more}) \ n));(\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) =$ 
 $((\exists n. g;\text{power } (f \wedge \text{more}) \ n));(\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))$ 
using ChopExistPower LeftChopEqvChop by fastforce
have 3:  $\bigwedge n. \vdash g;(\text{power } (f \wedge \text{more}) \ n) \longrightarrow h$ 
using ChopInductR assms
by (smt ChopAndA MP Prop02 Prop12 int-iffD2 int-simps(33) inteq-reflection lift-imp-trans)
have 31:  $\bigwedge n. \vdash g;((\text{power } (f \wedge \text{more}) \ n);(f \wedge \text{inf})) \longrightarrow h$ 
using assms
by (smt 3 AndChopB ChopAndA ChopAssoc Prop03 Prop10 inteq-reflection lift-imp-trans)
have 4:  $\vdash (\exists n. g;((\text{power } (f \wedge \text{more}) \ n);(\text{empty} \vee (f \wedge \text{inf})))) \longrightarrow h$ 
using 3 31 by (simp add: Valid-def,
  smt ChopEmpty ChopOrEqv inteq-reflection unl-lift2)
from 1 11 2 4 show ?thesis
by (metis InfiniteSemantics.ChopExist InfiniteSemantics.ExistChop AndMoreAndInfEqvAndInf
  inteq-reflection)
qed

```

lemma *ChopstarInductMoreR*:
assumes $\vdash g \vee h; (f \wedge \text{more}) \longrightarrow h$
shows $\vdash g; (\text{chopstar } f) \longrightarrow h$
proof –
have 1: $\vdash g; (\text{chopstar } f) = g; ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})))$
by (*simp add: chopstar-d-def powerstar-d-def*)
have 11: $\vdash g; ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))) =$
 $(g; (\exists n. \text{power } (f \wedge \text{more}) n)); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))$
using *ChopAssoc* **by** *blast*
have 2: $\vdash (g; (\exists n. \text{power } (f \wedge \text{more}) n)); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) =$
 $((\exists n. g; \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})))$
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 31: $\bigwedge n. \vdash g; ((\text{power } (f \wedge \text{more}) n); (f \wedge \text{inf})) \longrightarrow h$
using *assms*
by (*metis ChopInductInfR AndMoreAndInfEqvAndInf inteq-reflection*)
have 4: $\vdash (\exists n. g; ((\text{power } (f \wedge \text{more}) n); (\text{empty} \vee (f \wedge \text{inf})))) \longrightarrow h$
using 3 31 **by** (*simp add: Valid-def,*
smt ChopEmpty ChopOrEqv inteq-reflection unl-lift2)
from 1 11 2 4 **show** *?thesis*
by (*metis InfiniteSemantics.ChopExist InfiniteSemantics.ExistChop AndMoreAndInfEqvAndInf*
inteq-reflection)
qed

lemma *FPSEqvEmptyOrFiniteChopFPS*:
 $\vdash \text{fpowerstar } f = (\text{empty} \vee (f \wedge \text{finite}); \text{fpowerstar } f)$
using *FPowerstarEqvSem Valid-def* **by** *blast*

lemma *FPSAndMoreImpFPS*:
 $\vdash \text{fpowerstar } (f \wedge \text{more}) \longrightarrow \text{fpowerstar } f$
proof –
have 2: $\vdash \text{empty} \vee ((f \wedge \text{more}) \wedge \text{finite}); \text{fpowerstar } f \longrightarrow \text{fpowerstar } f$
by (*smt FPSEqvEmptyOrFiniteChopFPS LeftChopImpChop Prop02 Prop05 intl int-eq intensional-rews(3)*)
have 3: $\vdash \text{fpowerstar } (f \wedge \text{more}); \text{empty} \longrightarrow \text{fpowerstar } f$
using 2 *FPowerstarInductL* **by** *blast*
from 2 3 **show** *?thesis* **by** (*metis ChopEmpty int-eq*)
qed

lemma *FPSImpAndMoreFPS*:
 $\vdash \text{fpowerstar } f \longrightarrow \text{fpowerstar } (f \wedge \text{more})$
by (*meson ChopEmpty FPSEqvEmptyOrFiniteChopFPS FPowerstarInductMoreL int-iffD2 lift-imp-trans*)

lemma *FPSAndMoreEqvFPS*:
 $\vdash \text{fpowerstar } (f \wedge \text{more}) = \text{fpowerstar } f$
using *FPSAndMoreImpFPS FPSImpAndMoreFPS* **by** *fastforce*

lemma *ChopstarImpPowerstar*:
 $\vdash f^* \longrightarrow \text{powerstar } f$

by (*metis ChopEmpty ChopstarInductL PSEqvEmptyOrChopPS int-eq int-iffD2*)

lemma *PowerstarImpChopstar*:

$\vdash \text{powerstar } f \longrightarrow f^*$

by (*metis CSEqvOrChopCS ChopEmpty PowerstarInductL int-iffD2 inteq-reflection*)

lemma *ChopstarEqvPowerstar*:

$\vdash f^* = \text{powerstar } f$

using *ChopstarImpPowerstar PowerstarImpChopstar* **by** *fastforce*

lemma *CSAndMoreEqvAndMoreChop*:

$\vdash (f^* \wedge \text{more}) = (f \wedge \text{more}); f^*$

proof –

have 1: $\vdash (\text{empty} \vee (f \wedge \text{more}); f^*) \wedge \text{more} \longrightarrow (f \wedge \text{more}); f^*$

by (*auto simp: empty-d-def*)

have 2: $\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$

by (*rule ChopstarEqv*)

have 3: $\vdash f^* \wedge \text{more} \longrightarrow (f \wedge \text{more}); f^*$

using 1 2 **by** *fastforce*

have 4: $\vdash (f \wedge \text{more}); f^* \longrightarrow f^*$

using 2 **by** *fastforce*

have 5: $\vdash (f \wedge \text{more}) \longrightarrow \text{more}$

by *auto*

hence 6: $\vdash (f \wedge \text{more}); f^* \longrightarrow \text{more}$

by (*rule LeftChopImpMoreRule*)

have 7: $\vdash (f \wedge \text{more}); f^* \longrightarrow f^* \wedge \text{more}$

using 4 6 **by** *fastforce*

from 3 7 **show** *?thesis* **by** *fastforce*

qed

lemma *AndMoreCSEqvAndFmoreOrInf*:

$\vdash (f \wedge \text{more}); f^* = ((f \wedge \text{fmore}); f^* \vee (f \wedge \text{inf}))$

proof –

have 1: $\vdash (f \wedge \text{more}) = ((f \wedge \text{fmore}) \vee (f \wedge \text{inf}))$

by (*simp add: Valid-def more-defs chop-defs fmore-defs infinite-defs sum.case-eq-if*)

hence 2: $\vdash (f \wedge \text{more}); f^* = ((f \wedge \text{fmore}) \vee (f \wedge \text{inf}); f^*)$

by (*simp add: LeftChopEqvChop*)

have 3: $\vdash ((f \wedge \text{fmore}) \vee (f \wedge \text{inf}); f^*) = ((f \wedge \text{fmore}); f^* \vee (f \wedge \text{inf}); f^*)$

by (*simp add: OrChopEqv*)

have 4: $\vdash (f \wedge \text{inf}); f^* = (f \wedge \text{inf})$

using *AndInfChopEqvAndInf* **by** *blast*

have 5: $\vdash ((f \wedge \text{fmore}); f^* \vee (f \wedge \text{inf}); f^*) = ((f \wedge \text{fmore}); f^* \vee (f \wedge \text{inf}))$

using 3 4 **by** *auto*

from 2 3 5 **show** *?thesis* **by** *fastforce*

qed

lemma *PowerAndMoreAndFinite*:

$\vdash ((\text{power } (f \wedge \text{more}) \ n) \wedge \text{finite}) = (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) \ n)$

```

proof
  (induct n)
  case 0
  then show ?case using FiniteAndEmptyEqvEmpty by auto
  next
  case (Suc n)
  then show ?case
  by (smt chop-defs finite-d-def finite-defs-1 infinite-d-def infinite-defs-1 intl
      intensional-rews(1) intensional-rews(3) inteq-reflection power-d.pow-Suc
      sum.case-eq-if unl-lift)
qed

```

```

lemma CSAndFiniteDist:
   $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))) \wedge \text{finite} =$ 
   $((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{finite})$ 
proof –
  have 1:  $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))) \wedge \text{finite} =$ 
     $((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{finite});$ 
     $((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))) \wedge \text{finite})$ 
    using ChopAndFiniteDist by blast
  have 2:  $\vdash ((\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))) \wedge \text{finite} = \text{empty}$ 
    by (simp add: Valid-def empty-defs more-defs infinite-defs finite-defs sum.case-eq-if)
  from 1 2 show ?thesis
  by (metis ChopEmpty inteq-reflection)
qed

```

```

lemma CSAndFinite:
   $\vdash (f^* \wedge \text{finite}) = (f \wedge \text{finite})^*$ 
proof –
  have 1:  $\vdash (f^* \wedge \text{finite}) = ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))) \wedge \text{finite}$ 
    by (simp add: chopstar-d-def powerstar-d-def intl)
  have 11:  $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf}))) \wedge \text{finite} =$ 
     $((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{finite})$ 
    using CSAndFiniteDist by blast
  have 2:  $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{finite}) =$ 
     $(\exists n. \text{power } (f \wedge \text{more}) n \wedge \text{finite})$ 
    by (simp add: Valid-def)
  have 3:  $\vdash (\exists n. \text{power } (f \wedge \text{more}) n \wedge \text{finite}) =$ 
     $(\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n))$ 
    using PowerAndMoreAndFinite by fastforce
  have 31:  $\vdash (\text{empty} \vee ((f \wedge \text{finite}) \wedge \text{inf})) = \text{empty}$ 
    by (simp add: Valid-def empty-defs infinite-defs finite-defs sum.case-eq-if)
  have 4:  $\vdash (\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n)) = (f \wedge \text{finite})^*$ 
    using 31
    by (metis ChopEmpty AndMoreAndInfEqvAndInf chopstar-d-def inteq-reflection powerstar-d-def)
  from 1 11 2 3 4 show ?thesis by fastforce
qed

```

lemma *PowerchopAndFmore:*

$\vdash ((\text{power } (f \wedge \text{more}) (\text{Suc } n)) \wedge \text{fmore}) = (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) (\text{Suc } n))$

proof

(*induct n*)

case 0

then show ?*case*

by (*smt LeftChopImpFMoreRule AndMoreAndFiniteEqvAndFmore PowerAndMoreAndFinite Prop11 Prop12*
inteq-reflection pow-Suc)

next

case (*Suc n*)

then show ?*case*

by (*smt LeftChopImpFMoreRule AndMoreAndFiniteEqvAndFmore PowerAndMoreAndFinite Prop11 Prop12*
inteq-reflection 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 more)]* **by** *auto*

lemma *CSAndFmoreDist:*

$\vdash ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{fmore}) =$
 $((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{fmore})$

proof –

have 1: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{fmore}) =$
 $((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{finite} \wedge \text{more})$

by (*metis fmore-d-def inteq-reflection lift-and-com*)

have 2: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{finite}) =$
 $((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{finite})$

using *CSAndFiniteDist* **by** *blast*

have 3: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{finite} \wedge \text{more}) =$
 $((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{finite} \wedge \text{more})$

using 2 **by** *fastforce*

have 4: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{finite} \wedge \text{more}) =$
 $((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{fmore})$

by (*metis fmore-d-def inteq-reflection lift-and-com*)

from 1 3 4 **show** ?*thesis* **by** *fastforce*

qed

lemma *CSAndMoreEqvAndFMoreChop:*

$\vdash (f^* \wedge \text{fmore}) = (f \wedge \text{fmore}); (f \wedge \text{finite})^*$

proof –

have 1: $\vdash (f^* \wedge \text{fmore}) = ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{fmore})$
by (*simp add: chopstar-d-def powerstar-d-def intI*)

have 11: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee ((f \wedge \text{more}) \wedge \text{inf})) \wedge \text{fmore}) =$
 $((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{fmore})$

using *CSAndFmoreDist* **by** *fastforce*

have 2: $\vdash ((\exists n. \text{power } (f \wedge \text{more}) n) \wedge \text{fmore}) =$
 $(\exists n. \text{power } (f \wedge \text{more}) n \wedge \text{fmore})$

by (*simp add: Valid-def*)

have 3: $\vdash (\exists n. \text{power } (f \wedge \text{more}) n \wedge \text{fmore}) =$
 $((\text{power } (f \wedge \text{more}) 0 \vee (\exists n. (\text{power } (f \wedge \text{more}) (\text{Suc } n)))) \wedge \text{fmore})$
using *ExistPowerAndMoreExpand* **by** *fastforce*
have 4: $\vdash ((\text{power } (f \wedge \text{more}) 0 \vee (\exists n. (\text{power } (f \wedge \text{more}) (\text{Suc } n)))) \wedge \text{fmore}) =$
 $((\text{power } (f \wedge \text{more}) 0 \wedge \text{fmore}) \vee ((\exists n. (\text{power } (f \wedge \text{more}) (\text{Suc } n))) \wedge \text{fmore}))$
by *auto*
have 5: $\vdash (((\text{power } (f \wedge \text{more}) 0 \wedge \text{fmore}) \vee ((\exists n. (\text{power } (f \wedge \text{more}) (\text{Suc } n))) \wedge \text{fmore}))) =$
 $((\exists n. (\text{power } (f \wedge \text{more}) (\text{Suc } n))) \wedge \text{fmore})$
using *NotFmoreAndEmpty* **by** *fastforce*
have 6: $\vdash ((\exists n. (\text{power } (f \wedge \text{more}) (\text{Suc } n))) \wedge \text{fmore}) =$
 $(\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) (\text{Suc } n)))$
using *PowerchopAndFmore* **by** *fastforce*
have 7: $\vdash (\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) (\text{Suc } n))) =$
 $(\exists n. (((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}); (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n)))$
by *(simp)*
have 8: $\vdash (\exists n. (((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}); (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n))) =$
 $((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}; (\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n))$
by *(simp add: Valid-def,*
smt chop-d-def intensional-rews(6) sum.case-eq-if)
have 9: $\vdash (((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}) = (f \wedge \text{fmore})$
by *(simp add: fmore-d-def, auto)*
have 10: $\vdash (((f \wedge \text{finite}) \wedge \text{more}) \wedge \text{finite}); (\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n)) =$
 $(f \wedge \text{fmore}); (\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n))$
using 8 9 **by** *(simp add: LeftChopEqvChop)*
have 101: $\vdash (\text{empty} \vee ((f \wedge \text{finite}) \wedge \text{inf})) = \text{empty}$
by *(simp add: Valid-def empty-defs infinite-defs finite-defs sum.case-eq-if)*
have 11: $\vdash (\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n)) =$
 $(f \wedge \text{finite})^*$
using 101
by *(metis ChopEmpty AndMoreAndInfEqvAndInf chopstar-d-def inteq-reflection powerstar-d-def)*
hence 12: $\vdash (f \wedge \text{fmore}); (\exists n. (\text{power } ((f \wedge \text{finite}) \wedge \text{more}) n)) =$
 $(f \wedge \text{fmore}); (f \wedge \text{finite})^*$
by *(simp add: RightChopEqvChop)*
from 1 11 2 3 4 5 6 7 8 10 12 **show** *?thesis*
by *(metis CSAndFmoreDist inteq-reflection)*
qed

lemma *CSAndMoreImpChopCS*:

$\vdash f^* \wedge \text{more} \longrightarrow f; f^*$

proof —

have 1: $\vdash (f^* \wedge \text{more}) = (f \wedge \text{more}); f^*$ **by** *(rule CSAndMoreEqvAndMoreChop)*

have 2: $\vdash (f \wedge \text{more}); f^* \longrightarrow f; f^*$ **by** *(rule AndChopA)*

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *NotAndMoreEqvEmptyOr*:

$\vdash \neg (f \wedge \text{more}) = (\text{empty} \vee \neg f)$

by *(auto simp: empty-d-def)*

lemma *MoreAndEmptyOrEqvMoreAnd:*

$\vdash (more \wedge (empty \vee \neg f)) = (more \wedge \neg f)$

by (*auto simp: empty-d-def*)

lemma *CSMoreNotImpChopCSAndMore:*

$\vdash f^* \wedge more \wedge \neg f \longrightarrow (f \wedge more); (f^* \wedge more)$

proof –

have 1: $\vdash (f^* \wedge more) = (f \wedge more); f^*$

by (*rule CSAndMoreEqvAndMoreChop*)

have 2: $\vdash empty \vee more$

by (*auto simp: empty-d-def*)

hence 3: $\vdash f^* \longrightarrow empty \vee (f^* \wedge more)$

by *auto*

hence 4: $\vdash (f \wedge more); f^* \longrightarrow (f \wedge more) \vee ((f \wedge more); (f^* \wedge more))$

by (*rule ChopEmptyOrImpRule*)

hence 5: $\vdash (f \wedge more); f^* \wedge \neg(f \wedge more) \longrightarrow ((f \wedge more); (f^* \wedge more))$

by *fastforce*

have 6: $\vdash (f \wedge more); f^* = ((f \wedge more); f^* \wedge more)$ **using** 1

by *auto*

have 7: $\vdash ((f \wedge more); f^* \wedge \neg(f \wedge more)) = ((f \wedge more); f^* \wedge more \wedge \neg(f \wedge more))$

using 6 **by** *auto*

have 8: $\vdash (f \wedge more); f^* \wedge more \wedge \neg f \longrightarrow (f \wedge more); (f^* \wedge more)$

using 5 7 **by** *auto*

have 9: $\vdash (f^* \wedge more \wedge \neg f) = ((f^* \wedge more) \wedge (more \wedge \neg f))$

by *auto*

have 10: $\vdash ((f^* \wedge more) \wedge (more \wedge \neg f)) = ((f \wedge more); f^* \wedge (more \wedge \neg f))$

using 1 **by** *fastforce*

from 1 8 9 10 **show** *?thesis* **by** *fastforce*

qed

lemma *ChopplusCommutelmpA:*

$\vdash f^*;f \longrightarrow f;f^*$

by (*metis CSEqvOrChopCS ChopAndB ChopEmpty ChopstarInductL EmptyImpCS Prop02 Prop03 Prop10 inteq-reflection*)

lemma *ChopplusCommutelmpB:*

$\vdash f;f^* \longrightarrow f^*;f$

by (*smt CSEqvOrChopCS ChopplusCommutelmpA ChopstarInductR EmptyChop EmptyImpCS LeftChopImpChop Prop02 Prop03 inteq-reflection lift-imp-trans*)

lemma *ChopplusCommute:*

$\vdash f;f^* = f^*;f$

using *ChopplusCommutelmpA ChopplusCommutelmpB* **by** *fastforce*

lemma *CSEqvOrChopCSB:*

$\vdash f^* = (empty \vee (f^*;f))$

by (*meson CSEqvOrChopCS ChopplusCommute Prop06*)

lemma *CSAndMoreImpCSChop*:

$\vdash f^* \wedge \text{more} \longrightarrow f^*; f$

using *CSAndMoreEqvAndMoreChop ChopplusCommute CSAndMoreImpChopCS* **by** *fastforce*

lemma *PowerChopPower*:

$\vdash (\text{power } (f \wedge \text{more}) \ n); (\text{power } (f \wedge \text{more}) \ k) = (\text{power } (f \wedge \text{more}) \ (n+k))$

proof

(*induct n arbitrary: k*)

case 0

then show ?*case* **using** *EmptyChopSem* **by** *auto*

next

case (*Suc n*)

then show ?*case*

by (*metis (no-types, lifting) ChopAssoc add-Suc inteq-reflection pow-Suc*)

qed

lemma *CSChopCS*:

$\vdash f^*; f^* = f^*$

by (*smt CSEqvOrChopCS ChopEmpty ChopstarInductL EmptyImpCS Prop02 Prop05 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 *PowerAndInfB*:

$\vdash ((f \wedge \text{more}); (\text{power } (f \wedge \text{more}) \ n)) \wedge \text{inf} =$
 $((f \wedge \text{inf}) \vee (f \wedge \text{fmore}); ((\text{power } (f \wedge \text{more}) \ n) \wedge \text{inf}))$

proof —

have 1: $\vdash ((f \wedge \text{more}); (\text{power } (f \wedge \text{more}) \ n)) \wedge \text{inf} =$
 $((f \wedge \text{more}) \wedge \text{inf}) \vee ((f \wedge \text{more}) \wedge \text{finite}); ((\text{power } (f \wedge \text{more}) \ n) \wedge \text{inf}))$

using *ChopAndInfB* **by** *blast*

have 2: $\vdash ((f \wedge \text{more}) \wedge \text{inf}) \vee ((f \wedge \text{more}) \wedge \text{finite}); ((\text{power } (f \wedge \text{more}) \ n) \wedge \text{inf}) =$
 $((f \wedge \text{inf}) \vee (f \wedge \text{fmore}); ((\text{power } (f \wedge \text{more}) \ n) \wedge \text{inf}))$

using *AndMoreAndInfEqvAndInf AndMoreAndFiniteEqvAndFmore*

by (*metis 1 inteq-reflection*)

from 1 2 **show** ?*thesis* **by** *fastforce*

qed

lemma *CSAndInf*:

$\vdash (f^* \wedge \text{inf}) = f^*; (f \wedge \text{inf})$

by (*meson AndChopA AndInfChopEqvAndInf CSEqvOrChopCSB ChopAndA ChopAndInf Prop03 Prop11 Prop12 lift-imp-trans*)

lemma *CSChopCSImpCS*:

$\vdash (f^*; f^*) \longrightarrow f^*$

by (*simp add: CSChopCS int-iffD1*)

lemma *ImpChopPlus*:

$\vdash f \longrightarrow f; f^*$

proof —

have 1: $\vdash f^* = (\text{empty} \vee f; f^*)$ **by** (*rule CSEqvOrChopCS*)

hence 2: $\vdash f; f^* = (f; \text{empty} \vee f; (f; f^*))$ **using** *ChopOrEqvRule* **by** *blast*

have 3: $\vdash f; \text{empty} = f$ **using** *ChopEmpty* **by** *blast*

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *ImpCS*:

$\vdash f \longrightarrow f^*$

proof —

have 1: $\vdash f \longrightarrow f; f^*$ **by** (*rule ImpChopPlus*)

hence 2: $\vdash f \longrightarrow \text{empty} \vee f; f^*$ **by** *auto*

from 2 **show** *?thesis* **using** *CSEqvOrChopCS* **by** *fastforce*

qed

lemma *CSChopImpCS*:

$\vdash f^*; f \longrightarrow f^*$

proof —

have 1: $\vdash f \longrightarrow f^*$ **by** (*rule ImpCS*)

hence 2: $\vdash f^*; f \longrightarrow f^*; f^*$ **by** (*rule RightChopImpChop*)

hence 3: $\vdash f^*; f \longrightarrow f^*; f^*$ **by** *auto*

have 4: $\vdash f^*; f^* \longrightarrow f^*$ **by** (*rule CSChopCSImpCS*)

from 2 3 4 **show** *?thesis* **using** *lift-imp-trans* **by** *blast*

qed

lemma *ChopPlusImpCS*:

$\vdash f; f^* \longrightarrow f^*$

proof —

have 1: $\vdash f; f^* \longrightarrow \text{empty} \vee f; f^*$ **by** *auto*

from 1 **show** *?thesis* **using** *CSEqvOrChopCS* **by** *fastforce*

qed

lemma *CSChopEqvOrChopPlusChop*:

$\vdash f^*; g = (g \vee (f; f^*); g)$

proof —

have 1: $\vdash f^* = (\text{empty} \vee f; f^*)$ **by** (*rule CSEqvOrChopCS*)

from 1 show ?thesis using EmptyOrChopEqvRule by blast
qed

lemma CSElim:

assumes $\vdash \text{empty} \longrightarrow g$
 $\vdash (f \wedge \text{more}); g \longrightarrow g$

shows $\vdash f^* \longrightarrow g$

proof —

have 1: $\vdash \text{empty} \vee (f \wedge \text{more}); g \longrightarrow g$

using *assms* **using** *Prop02* **by** *blast*

have 2: $\vdash (\text{chopstar } f); \text{empty} \longrightarrow g$

using *ChopstarInductMoreL 1* **by** *blast*

from 2 show ?thesis

by (*metis ChopEmpty inteq-reflection*)

qed

lemma ChopstarImp:

assumes $\vdash f; (\text{chopstar } g) \vee \text{empty} \longrightarrow (\text{chopstar } g)$

shows $\vdash (\text{chopstar } f) \longrightarrow (\text{chopstar } g)$

using *assms ChopstarInductL ChopEmpty*

by (*metis int-eq int-simps(33) lift-and-com*)

lemma CSCSImpCS:

$\vdash (f^*)^* \longrightarrow f^*$

proof —

have 1: $\vdash ((\text{chopstar } f); (\text{chopstar } f)) \vee \text{empty} \longrightarrow (\text{chopstar } f)$

by (*meson CSChopCSImpCS EmptyImpCS Prop02*)

from 1 show ?thesis using ChopstarImp by blast

qed

lemma CSImpCSCS:

$\vdash f^* \longrightarrow (f^*)^*$

using *ImpCS* **by** *auto*

lemma CSCSEqvCS:

$\vdash (f^*)^* = f^*$

by (*simp add: CSCSImpCS CSImpCSCS int-iff1*)

lemma RightEmptyOrChopEqv:

$\vdash g; (\text{empty} \vee f) = (g \vee (g; f))$

proof —

have 1: $\vdash g; (\text{empty} \vee f) = (g; \text{empty} \vee g; f)$ **by** (*rule ChopOrEqv*)

have 2: $\vdash g; \text{empty} = g$ **by** (*rule ChopEmpty*)

from 1 2 show ?thesis by fastforce

qed

lemma RightEmptyOrChopEqvRule:

assumes $\vdash f = (\text{empty} \vee f1)$

shows $\vdash g; f = (g \vee (g; f1))$

proof –
have 1: $\vdash f = (\text{empty} \vee f1)$ **using** *assms* **by** *auto*
hence 2: $\vdash g;f = g;(\text{empty} \vee f1)$ **by** (rule *RightChopEqvChop*)
have 3: $\vdash g;(\text{empty} \vee f1) = (g \vee (g;f1))$ **by** (rule *RightEmptyOrChopEqv*)
from 2 3 **show** ?thesis **by** *fastforce*
qed

lemma *ChopPlusEqvOrChopChopPlus*:
 $\vdash (f;f^*) = (f \vee f; (f;f^*))$

proof –
have 1: $\vdash f^* = (\text{empty} \vee f;f^*)$ **by** (rule *CSEqvOrChopCS*)
from 1 **show** ?thesis **by** (rule *RightEmptyOrChopEqvRule*)
qed

lemma *CSAndEmptyEqvEmpty*:
 $\vdash ((f^*) \wedge \text{empty}) = \text{empty}$
using *EmptyImpCS* **by** *fastforce*

lemma *NotAndMoreChopAndEmpty*:
 $\vdash \neg(((f \wedge \text{more});g) \wedge \text{empty})$
by (metis *AndChopA ChopEmpty LeftChopImpMoreRule Prop01 empty-d-def int-simps(14)*
int-simps(25) int-simps(4) inteq-reflection lift-and-com)

lemma *NotChopAndMoreAndEmpty*:
 $\vdash \neg((f;(g \wedge \text{more})) \wedge \text{empty})$
by (metis *NotEmptyEqvMore Prop01 Prop05 Prop07 RightChopImpMoreRule empty-d-def int-iffD2*
int-simps(15) inteq-reflection lift-imp-neg)

lemma *ChopCSAndEmptyEqvAndEmpty*:
 $\vdash ((f;f^*) \wedge \text{empty}) = (f \wedge \text{empty})$

proof –
have 1: $\vdash ((f;f^*) \wedge \text{empty}) = (f \wedge \text{empty});(f^* \wedge \text{empty})$
using *ChopAndEmptyEqvEmptyChopEmpty* **by** *blast*
have 2: $\vdash (f \wedge \text{empty});(f^* \wedge \text{empty}) = (f \wedge \text{empty});\text{empty}$
using *CSAndEmptyEqvEmpty* **using** *RightChopEqvChop* **by** *blast*
have 3: $\vdash (f \wedge \text{empty});\text{empty} = (f \wedge \text{empty})$
by (rule *ChopEmpty*)
from 1 2 3 **show** ?thesis **by** *fastforce*
qed

lemma *AndMoreChopAndMoreEqvAndMoreChop*:
 $\vdash ((f \wedge \text{more});g \wedge \text{more}) = (f \wedge \text{more});g$
using *ChopImpDi DiAndB DiMoreEqvMore* **by** *fastforce*

lemma *ChopPlusEqv*:
 $\vdash (f;f^*) = (f \vee (f \wedge \text{more}); (f;f^*))$

proof –
have 1: $\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$
by (rule *ChopstarEqv*)
have 2: $\vdash f^* = (\text{empty} \vee f;f^*)$

by (rule CSEqvOrChopCS)
 hence 3: $\vdash (\text{empty} \vee f;f^*) = (\text{empty} \vee (f \wedge \text{more});f^*)$
 using 1 2 by fastforce
 have 4: $\vdash (f \wedge \text{more});(f^*) = (f \wedge \text{more});(\text{empty} \vee f;f^*)$
 using 2 using RightChopEqvChop by blast
 hence 5: $\vdash \text{empty} \vee f;f^* = \text{empty} \vee (f \wedge \text{more});(\text{empty} \vee f;f^*)$
 using 3 4 by fastforce
 have 6: $\vdash (f \wedge \text{more});(\text{empty} \vee f;f^*) =$
 $((f \wedge \text{more}); \text{empty} \vee (f \wedge \text{more});(f;f^*))$
 using ChopOrEqv by blast
 have 7: $\vdash (f \wedge \text{more}); \text{empty} = (f \wedge \text{more})$
 using ChopEmpty by blast
 have 8: $\vdash (\text{empty} \vee f;f^*) =$
 $(\text{empty} \vee (f \wedge \text{more}) \vee (f \wedge \text{more});(f;f^*))$
 using 5 6 7 by (metis 2 3 inteq-reflection)
 have 9: $\vdash ((\text{empty} \vee f;f^*) \wedge \text{more}) = (f;f^* \wedge \text{more})$
 by (auto simp: empty-d-def)
 have 10: $\vdash ((\text{empty} \vee (f \wedge \text{more}) \vee (f \wedge \text{more});(f;f^*)) \wedge \text{more}) =$
 $((f \wedge \text{more}) \vee (f \wedge \text{more});(f;f^*)) \wedge \text{more}$
 by (auto simp: empty-d-def)
 have 11: $\vdash (((f \wedge \text{more}) \vee (f \wedge \text{more});(f;f^*)) \wedge \text{more}) =$
 $((f \wedge \text{more}) \vee (f \wedge \text{more});(f;f^*))$
 using 10 6 7 int-eq
 using AndMoreChopAndMoreEqvAndMoreChop by fastforce
 have 12: $\vdash (f;f^* \wedge \text{more}) = ((f \wedge \text{more}) \vee (f \wedge \text{more});(f;f^*))$
 using 8 9 10 11 by fastforce
 have 13: $\vdash (f;f^* \wedge \text{empty}) = (f \wedge \text{empty})$
 by (rule ChopCSAndEmptyEqvAndEmpty)
 have 14: $\vdash ((f \wedge \text{more}) \vee (f \wedge \text{more});(f;f^*)) \vee (f \wedge \text{empty}) =$
 $(f \vee (f \wedge \text{more});(f;f^*))$
 by (auto simp: empty-d-def)
 have 15: $\vdash f;f^* = ((f;f^* \wedge \text{empty}) \vee (f;f^* \wedge \text{more}))$
 by (auto simp: empty-d-def)
 from 12 13 14 15 show ?thesis by fastforce
 qed

lemma ChopPlusImpChopPlus:

assumes $\vdash f \longrightarrow g$

shows $\vdash f;f^* \longrightarrow g;g^*$

using assms

by (smt CSEqvOrChopCS ChopAssoc ChopImpChop ChopstarInductR ImpCS ImpChopPlus Prop02 int-iffD1
inteq-reflection lift-imp-trans)

lemma ChopChopPlusImpChopPlus:

$\vdash f; (f;f^*) \longrightarrow f;f^*$

proof –

have 1: $\vdash \text{empty} \vee \text{more}$ by (auto simp: empty-d-def)

hence 2: $\vdash f \longrightarrow \text{empty} \vee (f \wedge \text{more})$ by auto

hence 3: $\vdash f; (f;f^*) \longrightarrow (f;f^*) \vee (f \wedge \text{more});(f;f^*)$ by (rule EmptyOrChopImpRule)

have 4: $\vdash f;f^* = (f \vee (f \wedge \text{more}); (f;f^*))$ **by** (rule ChopPlusEqv)
hence 5: $\vdash (f \wedge \text{more}); (f;f^*) \longrightarrow f;f^*$ **by** auto
from 3 5 **show** ?thesis **using** ChopPlusImpCS RightChopImpChop **by** blast
qed

lemma CSImpCS:

assumes $\vdash f \longrightarrow g$
shows $\vdash f^* \longrightarrow g^*$
proof –
have 1: $\vdash f \longrightarrow g$ **using** assms **by** auto
hence 2: $\vdash f;f^* \longrightarrow g;g^*$ **by** (rule ChopPlusImpChopPlus)
hence 3: $\vdash \text{empty} \vee f;f^* \longrightarrow \text{empty} \vee g;g^*$ **by** auto
from 2 3 **show** ?thesis **using** CSEqvOrChopCS **by** (metis inteq-reflection)
qed

lemma ChopPlusIntro:

assumes $\vdash f \longrightarrow g \vee (g \wedge \text{more}); f$
shows $\vdash f \wedge \text{finite} \longrightarrow g;g^*$
proof –
have 1: $\vdash f \wedge \neg g \longrightarrow (g \wedge \text{more}); f$ **using** assms **by** auto
have 2: $\vdash g;g^* = (g \vee (g \wedge \text{more}); (g;g^*))$ **by** (rule ChopPlusEqv)
have 3: $\vdash f \wedge \neg (g;g^*) \longrightarrow$
 $(g \wedge \text{more}); f \wedge \neg ((g \wedge \text{more}); (g;g^*))$ **using** 1 2 **by** fastforce
have 4: $\vdash g \wedge \text{more} \longrightarrow \text{more}$ **by** auto
from 3 4 **show** ?thesis **using** ChopContraB **by** blast
qed

lemma ChopPlusElim:

assumes $\vdash f \longrightarrow g$
 $\vdash (f \wedge \text{more}); g \longrightarrow g$
shows $\vdash f;f^* \longrightarrow g$
proof –
have 1: $\vdash f \vee (f \wedge \text{more}); g \longrightarrow g$
using assms Prop02 **by** blast
have 2: $\vdash f^*;f \longrightarrow g$
using ChopstarInductMoreL 1 **by** blast
from 2 **show** ?thesis
using ChopplusCommute **by** fastforce
qed

lemma ChopPlusElimWithoutMore:

assumes $\vdash f \longrightarrow g$
 $\vdash f; g \longrightarrow g$
shows $\vdash f;f^* \longrightarrow g$
proof –
have 1: $\vdash f \longrightarrow g$ **using** assms **by** blast
have 2: $\vdash (f; g) \longrightarrow g$ **using** assms **by** blast
have 3: $\vdash (f \wedge \text{more}); g \longrightarrow f; g$ **by** (rule AndChopA)

have 4: $\vdash (f \wedge \text{more}); g \longrightarrow g$ **using** 2 3 *lift-imp-trans* **by** *blast*
 from 1 4 **show** *?thesis* **using** *ChopPlusElim* **by** *blast*
qed

lemma *ChopPlusEqvChopPlus*:

assumes $\vdash f = g$
shows $\vdash f;f^* = g;g^*$

proof —

have 1: $\vdash f = g$ **using** *assms* **by** *auto*
 hence 2: $\vdash f \longrightarrow g$ **by** *auto*
 hence 3: $\vdash f;f^* \longrightarrow g;g^*$ **by** (*rule ChopPlusImpChopPlus*)
 have 4: $\vdash g \longrightarrow f$ **using** 1 **by** *auto*
 hence 5: $\vdash g;g^* \longrightarrow f;f^*$ **by** (*rule ChopPlusImpChopPlus*)
 from 3 5 **show** *?thesis* **by** *fastforce*

qed

lemma *CSEqvCS*:

assumes $\vdash f = g$
shows $\vdash f^* = g^*$

proof —

have 1: $\vdash f = g$ **using** *assms* **by** *auto*
 hence 2: $\vdash f;f^* = g;g^*$ **by** (*rule ChopPlusEqvChopPlus*)
 hence 3: $\vdash (\text{empty} \vee f;f^*) = (\text{empty} \vee g;g^*)$ **by** *auto*
 from 3 **show** *?thesis* **using** *CSEqvOrChopCS* **by** (*metis int-eq*)

qed

lemma *AndCSA*:

$\vdash (f \wedge g)^* \longrightarrow f^*$

proof —

have 1: $\vdash f \wedge g \longrightarrow f$ **by** *auto*
 from 1 **show** *?thesis* **using** *CSImpCS* **by** *blast*

qed

lemma *AndCSB*:

$\vdash (f \wedge g)^* \longrightarrow g^*$

proof —

have 1: $\vdash f \wedge g \longrightarrow g$ **by** *auto*
 from 1 **show** *?thesis* **using** *CSImpCS* **by** *blast*

qed

lemma *CSIntro*:

assumes $\vdash f \wedge \text{more} \longrightarrow (g \wedge \text{more}); f$
shows $\vdash f \wedge \text{finite} \longrightarrow g^*$

proof —

have 1: $\vdash f \wedge \text{more} \longrightarrow (g \wedge \text{more}); f$
 using *assms* **by** *auto*
 have 2: $\vdash \text{more} = (\neg \text{empty})$
 by (*auto simp: empty-d-def*)
 have 3: $\vdash f \wedge \neg \text{empty} \longrightarrow (g \wedge \text{more}); f$

using 1 2 by fastforce
 have 4: $\vdash g^* = (\text{empty} \vee (g \wedge \text{more}); g^*)$
 by (rule ChopstarEqv)
 hence 41: $\vdash (\neg(\text{empty} \vee (g \wedge \text{more}); g^*)) = (\neg \text{empty} \wedge \neg((g \wedge \text{more}); g^*))$
 by fastforce
 have 411: $\vdash (\neg \text{empty} \wedge \neg((g \wedge \text{more}); g^*)) = (\text{more} \wedge \neg((g \wedge \text{more}); g^*))$
 using NotEmptyEqvMore by fastforce
 have 42: $\vdash \neg(g^*) = (\text{more} \wedge \neg((g \wedge \text{more}); g^*))$
 using 4 41 411 by fastforce
 have 43: $\vdash f \wedge \neg(g^*) \longrightarrow f \wedge \text{more} \wedge \neg((g \wedge \text{more}); g^*)$
 using 42 by fastforce
 have 44: $\vdash f \wedge \text{more} \wedge \neg((g \wedge \text{more}); g^*) \longrightarrow (g \wedge \text{more}); f \wedge \neg((g \wedge \text{more}); g^*)$
 using 3 43 1 by auto
 have 5: $\vdash f \wedge \neg(g^*) \longrightarrow$
 $(g \wedge \text{more}); f \wedge \neg((g \wedge \text{more}); g^*)$
 using 43 44 lift-imp-trans by fastforce
 have 6: $\vdash g \wedge \text{more} \longrightarrow \text{more}$
 by auto
 from 5 6 show ?thesis using ChopContraB by blast
 qed

lemma CSElimWithoutMore:

assumes $\vdash \text{empty} \longrightarrow g$
 $\vdash f; g \longrightarrow g$
 shows $\vdash f^* \longrightarrow g$
 proof –
 have 1: $\vdash \text{empty} \longrightarrow g$ using assms by blast
 have 2: $\vdash f; g \longrightarrow g$ using assms by blast
 have 3: $\vdash (f \wedge \text{more}); g \longrightarrow f; g$ by (rule AndChopA)
 have 4: $\vdash (f \wedge \text{more}); g \longrightarrow g$ using 2 3 lift-imp-trans by blast
 from 1 4 show ?thesis using CSElim by blast
 qed

lemma ChopAssocB:

$\vdash (f;g);h = f;(g;h)$
 using ChopAssoc by fastforce

lemma CSChopEqvChopOrRule:

assumes $\vdash f = (g^*; h)$
 shows $\vdash f = ((g; f) \vee h)$
 proof –
 have 1: $\vdash f = (g^*; h)$ using assms by auto
 have 2: $\vdash g^* = (\text{empty} \vee (g; g^*))$ by (rule CSEqvOrChopCS)
 hence 3: $\vdash g^*; h = (h \vee ((g; g^*); h))$ by (rule EmptyOrChopEqvRule)
 have 4: $\vdash (g; g^*); h = g; (g^*; h)$ by (rule ChopAssocB)
 hence 41: $\vdash g^*; h = (h \vee g; (g^*; h))$ using 3 by fastforce
 have 5: $\vdash g; f = g; (g^*; h)$ using 1 by (rule RightChopEqvChop)
 hence 6: $\vdash (g^*; h) = (h \vee g; f)$ using 41 by fastforce
 hence 61: $\vdash (g^*; h) = ((g; f) \vee h)$ by auto
 from 1 61 show ?thesis by fastforce

qed

lemma *CChopIntroRule*:

assumes $\vdash f \wedge \neg h \longrightarrow g; f$

$\vdash g \longrightarrow \text{more}$

shows $\vdash f \wedge \text{finite} \longrightarrow g^*; h$

proof —

have 1: $\vdash f \wedge \neg h \longrightarrow g; f$

using *assms* **by** *blast*

have 2: $\vdash g \longrightarrow \text{more}$

using *assms* **by** *blast*

hence 3: $\vdash g \longrightarrow g \wedge \text{more}$

by *auto*

hence 4: $\vdash g; f \longrightarrow (g \wedge \text{more}); f$

by (*rule LeftChopImpChop*)

have 5: $\vdash f \longrightarrow (g \wedge \text{more}); f \vee h$

using 1 4 **by** *fastforce*

have 6: $\vdash g^* = (\text{empty} \vee (g \wedge \text{more}); g^*)$

by (*rule ChopstarEqv*)

hence 7: $\vdash (g^*); h = (h \vee ((g \wedge \text{more}); g^*); h)$

by (*rule EmptyOrChopEqvRule*)

have 8: $\vdash ((g \wedge \text{more}); g^*); h = (g \wedge \text{more}); (g^*; h)$

by (*rule ChopAssocB*)

have 9: $\vdash (g^*); h = (h \vee (g \wedge \text{more}); (g^*; h))$

using 7 8 **by** *fastforce*

have 10: $\vdash f \wedge \neg (g^*; h) \longrightarrow (g \wedge \text{more}); f \wedge \neg ((g \wedge \text{more}); (g^*; h))$

using 5 9 **by** *fastforce*

have 11: $\vdash g \wedge \text{more} \longrightarrow \text{more}$

by *fastforce*

from 10 11 **show** *?thesis* **using** *ChopContraB* **by** *blast*

qed

lemma *DiamondAndEmptyEqvAndEmpty*:

$\vdash (\Diamond f \wedge \text{empty}) = (f \wedge \text{empty})$

by (*auto simp: sometimes-defs empty-defs sum.case-eq-if*)

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 $\vdash f \longrightarrow \text{empty} \vee ((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}); f$

shows $\vdash (\text{init } w \wedge f) \wedge \text{finite} \longrightarrow \Box(\text{init } w)$

proof –

have 1: $\vdash f \longrightarrow \text{empty} \vee ((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}); f$

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

have 2: $\vdash \text{init } w \wedge \neg(\Box(\text{init } w)) \longrightarrow \neg \text{empty}$

by (*rule InitAndNotBoxInitImpNotEmpty*)

have 3: $\vdash \text{init } w \wedge f \wedge \neg(\Box(\text{init } w)) \longrightarrow ((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}); f$

using 1 2 **by** *fastforce*

have 4: $\vdash \Box(\text{init } w) \wedge \text{more} \longrightarrow (\Box(\text{init } w) \wedge \text{more}) \wedge \text{fin }(\text{init } w)$

by (*rule BoxInitAndMoreImpBoxInitAndMoreAndFinInit*)

have 41: $\vdash (\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite} \longrightarrow$

$((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}) \wedge \text{fin }(\text{init } w)$

using 4 **by** *auto*

hence 5: $\vdash ((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}); f \longrightarrow$

$((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}) \wedge \text{fin }(\text{init } w); f$

```

    by (rule LeftChopImpChop)
have 6:  $\vdash (((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}) \wedge \text{fin } (\text{init } w)); f =$ 
     $((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}); (\text{init } w \wedge f)$ 
    using AndFinChopEqvStateAndChop by blast
have 7:  $\vdash \neg(\Box(\text{init } w)) \longrightarrow (\Box(\text{init } w)) \text{ yields } (\neg(\Box(\text{init } w)))$ 
    by (rule NotBoxStateImpBoxYieldsNotBox)
have 8:  $\vdash (\Box(\text{init } w)) \text{ yields } (\neg(\Box(\text{init } w))) \longrightarrow$ 
     $((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}) \text{ yields } (\neg(\Box(\text{init } w)))$ 
    using AndYieldsA
    by (metis AndMoreAndFiniteEqvAndFmore inteq-reflection)
have 9:  $\vdash ((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}); (\text{init } w \wedge f) \wedge$ 
     $((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}) \text{ yields } (\neg(\Box(\text{init } w)))$ 
     $\longrightarrow$ 
     $((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}); ((\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)))$ 
    by (rule ChopAndYieldsImp)
have 10:  $\vdash (\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)) \longrightarrow$ 
     $((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}); ((\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)))$ 
    using 3 5 6 7 8 9 by fastforce
have 11:  $\vdash ((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}); ((\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w))) \longrightarrow$ 
     $\text{more}; ((\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)))$ 
    by (metis 41 LeftChopImpChop Prop12)
have 12:  $\vdash (\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)) \longrightarrow$ 
     $\text{more}; ((\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)))$ 
    using 10 11 by fastforce
from 12 show ?thesis using MoreChopContraFiniteB by blast
qed

```

lemma BoxCSEqvBox:

```

 $\vdash (\text{init } w \wedge (\Box(\text{init } w))^*) = \Box(\text{init } w)$ 
proof –
have 1:  $\vdash \Box(\text{init } w); \Box(\text{init } w) \longrightarrow \Box(\text{init } w)$ 
    by (simp add: BoxStateChopBoxEqvBox int-iffD1)
have 2:  $\vdash (\text{init } w \wedge \text{empty}) \longrightarrow \Box(\text{init } w)$ 
    by (simp add: StateAndEmptyImpBoxState)
have 3:  $\vdash (\text{init } w \wedge \text{empty}) \vee \Box(\text{init } w); \Box(\text{init } w) \longrightarrow \Box(\text{init } w)$ 
    using 1 2 by fastforce
have 4:  $\vdash (\text{init } w \wedge \text{empty}); (\Box(\text{init } w))^* \longrightarrow \Box(\text{init } w)$ 
    using ChopstarInductR 3 by blast
have 5:  $\vdash \text{init } w \wedge (\Box(\text{init } w))^* \longrightarrow \Box(\text{init } w)$ 
    using 4 StateAndEmptyChop by fastforce
have 11:  $\vdash \Box(\text{init } w) \longrightarrow (\text{init } w)$ 
    using BoxElim by blast
have 12:  $\vdash \Box(\text{init } w) \longrightarrow (\Box(\text{init } w))^*$ 
    by (rule ImpCS)
have 13:  $\vdash \Box(\text{init } w) \longrightarrow \text{init } w \wedge (\Box(\text{init } w))^*$ 
    using 11 12 by fastforce
from 5 13 show ?thesis by fastforce
qed

```


lemma *BoxStateAndCSEqvCS*:

$\vdash (\Box(\text{init } w) \wedge f^* \wedge \text{finite}) = (\text{init } w \wedge (\Box(\text{init } w) \wedge f)^* \wedge \text{finite})$

proof –

have 1: $\vdash \Box(\text{init } w) \longrightarrow \text{init } w$

using *BoxElim* **by** *blast*

have 2: $\vdash (f^* \wedge \text{more}) = (f \wedge \text{more}); f^*$

by (*rule CSAndMoreEqvAndMoreChop*)

have 3: $\vdash (\Box(\text{init } w) \wedge ((f \wedge \text{more}); f^*)) =$

$((\Box(\text{init } w) \wedge f \wedge \text{more}); (\Box(\text{init } w) \wedge f^*))$

by (*rule BoxStateAndChopEqvChop*)

have 4: $\vdash \Box(\text{init } w) \wedge f \wedge \text{more} \longrightarrow (\Box(\text{init } w) \wedge f) \wedge \text{more}$

by *auto*

hence 5: $\vdash (\Box(\text{init } w) \wedge f \wedge \text{more}); (\Box(\text{init } w) \wedge f^*) \longrightarrow$

$((\Box(\text{init } w) \wedge f) \wedge \text{more}); (\Box(\text{init } w) \wedge f^*)$

by (*rule LeftChopImpChop*)

have 6: $\vdash (\Box(\text{init } w) \wedge f^*) \wedge \text{more} \longrightarrow$

$((\Box(\text{init } w) \wedge f) \wedge \text{more}); (\Box(\text{init } w) \wedge f^*)$

using 2 3 5 **by** *fastforce*

hence 7: $\vdash (\Box(\text{init } w) \wedge f^*) \wedge \text{finite} \longrightarrow (\Box(\text{init } w) \wedge f)^*$

using *CSIntro* **by** *blast*

have 71: $\vdash \text{init } w \wedge \Box(\text{init } w) \wedge f^* \wedge \text{finite} \longrightarrow \text{init } w \wedge (\Box(\text{init } w) \wedge f)^* \wedge \text{finite}$

using 7 **by** *fastforce*

have 8: $\vdash \Box(\text{init } w) \wedge f^* \wedge \text{finite} \longrightarrow \text{init } w \wedge (\Box(\text{init } w) \wedge f)^* \wedge \text{finite}$

using 1 71 **by** *fastforce*

have 11: $\vdash (\Box(\text{init } w) \wedge f)^* \longrightarrow (\Box(\text{init } w))^*$

by (*rule AndCSA*)

have 12: $\vdash (\text{init } w \wedge (\Box(\text{init } w))^*) = \Box(\text{init } w)$

by (*rule BoxCSEqvBox*)

have 13: $\vdash (\Box(\text{init } w) \wedge f)^* \longrightarrow f^*$

by (*rule AndCSB*)

have 14: $\vdash \text{init } w \wedge (\Box(\text{init } w) \wedge f)^* \longrightarrow \text{init } w \wedge (\Box(\text{init } w))^* \wedge f^*$

using 11 13 **by** *fastforce*

have 15: $\vdash \text{init } w \wedge (\Box(\text{init } w))^* \wedge f^* \longrightarrow \Box(\text{init } w) \wedge f^*$

using 12 **by** *auto*

have 16: $\vdash \text{init } w \wedge (\Box(\text{init } w) \wedge f)^* \longrightarrow \Box(\text{init } w) \wedge f^*$

using 14 15 *lift-imp-trans* **by** *blast*

from 8 16 **show** *?thesis* **by** *fastforce*

qed

lemma *BaCSImpCS*:

$\vdash \text{ba } (f \longrightarrow g) \wedge \text{finite} \longrightarrow f^* \longrightarrow g^*$

proof –

have 1: $\vdash f^* = (\text{empty} \vee (f \wedge \text{more}); f^*)$

by (*rule ChopstarEqv*)

have 2: $\vdash g^* = (\text{empty} \vee (g \wedge \text{more}); g^*)$

by (*rule ChopstarEqv*)

have 21: $\vdash \neg(g^*) = (\neg \text{empty} \wedge \neg((g \wedge \text{more}); g^*))$
using 2 **by** *fastforce*
hence 22: $\vdash \neg(g^*) = (\text{more} \wedge \neg((g \wedge \text{more}); g^*))$
using *NotEmptyEqvMore* **by** *fastforce*
have 3: $\vdash f^* \wedge \neg(g^*) \longrightarrow$
 $(\text{empty} \vee (f \wedge \text{more}); f^*) \wedge \text{more} \wedge \neg((g \wedge \text{more}); g^*)$
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: $\vdash \text{finite} \longrightarrow \neg((\text{ba}(f \longrightarrow g)) \wedge (f^*) \wedge (\neg(g^*)))$
using *MoreChopLoopFiniteB* **by** *blast*
from 12 **show** ?thesis **by** (*simp add: Valid-def*)
qed

lemma *BaCSEqvCS*:

$\vdash \text{ba}(f = g) \wedge \text{finite} \longrightarrow (f^* = g^*)$

proof —

have 1: $\vdash \text{ba}(f = g) = (\text{ba}(f \longrightarrow g) \wedge \text{ba}(g \longrightarrow f))$ **by** (*auto simp: ba-defs sum.case-eq-if*)
have 2: $\vdash \text{ba}(f \longrightarrow g) \wedge \text{finite} \longrightarrow (f^* \longrightarrow g^*)$ **by** (*rule BaCSImpCS*)
have 3: $\vdash \text{ba}(g \longrightarrow f) \wedge \text{finite} \longrightarrow (g^* \longrightarrow f^*)$ **by** (*rule BaCSImpCS*)
have 4: $\vdash \text{ba}(f = g) \wedge \text{finite} \longrightarrow (f^* \longrightarrow g^*) \wedge (g^* \longrightarrow f^*)$ **using** 1 2 3 **by** *fastforce*
have 5: $\vdash ((f^* \longrightarrow g^*) \wedge (g^* \longrightarrow f^*)) = (f^* = g^*)$ **by** *auto*
from 4 5 **show** ?thesis **by** *auto*
qed

lemma *BaAndCSImport*:

$\vdash ba\ f \wedge g^* \wedge finite \longrightarrow (f \wedge g)^*$
proof –
have 1: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ **by** *auto*
hence 2: $\vdash ba\ f \longrightarrow ba\ (g \longrightarrow f \wedge g)$ **by** (*rule BalmpBa*)
have 3: $\vdash ba\ (g \longrightarrow f \wedge g) \wedge finite \longrightarrow g^* \longrightarrow (f \wedge g)^*$ **by** (*rule BaCSImpCS*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *CSSkipImpFinite*:
 $\vdash skip^* \longrightarrow finite$
using *CSElimWithoutMore EmptyImpFinite SkipChopFiniteImpFinite* **by** *blast*

lemma *FinitImpCSSkip*:
 $\vdash finite \longrightarrow skip^*$
using *CSIntro*
by (*metis (no-types, lifting) CSSkipImpFinite ChopAndB FiniteChopSkipEqvFiniteAndMore FmoreEqvSkipChopFinite ImpChopPlus Prop10 Prop12 fmore-d-def int-iffD1 inteq-reflection*)

lemma *CSSkipEqvFinite*:
 $\vdash skip^* = finite$
using *CSSkipImpFinite FinitImpCSSkip* **by** *fastforce*

9.9 Properties of Omega

lemma *NotOmegaEmpty*:
 $\vdash \neg((empty)^\omega)$
proof –
have 1: $\vdash (empty)^\omega = (empty \wedge fmore);(empty)^\omega$
by (*simp add: OmegaUnroll*)
have 2: $\vdash (empty \wedge fmore) = \#False$
using *NotFmoreAndEmpty* **by** *auto*
have 3: $\vdash \#False;(empty)^\omega = \#False$
by (*metis AndInfChopEqvAndInf int-eq int-simps(22)*)
from 1 2 3 **show** *?thesis*
by (*metis TrueW int-simps(3) inteq-reflection*)
qed

lemma *NotOmegaFalse*:
 $\vdash \neg((\#False)^\omega)$
by (*metis ChopImpDi DiIntro NotDiFalse OmegaUnroll int-iffI int-simps(14) int-simps(19) inteq-reflection*)

lemma *NotOmegaInf*:
 $\vdash \neg((inf)^\omega)$
proof –
have 1: $\vdash (inf)^\omega = (inf \wedge fmore);(inf)^\omega$
by (*simp add: OmegaUnroll*)
have 2: $\vdash (inf \wedge fmore) = \#False$
using *FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite FmoreEqvSkipChopFinite InfEqvNotFinite* **by** *fastforce*

have 3: $\vdash \#False; (inf)^\omega = \#False$
by (metis AndInfChopEqvAndInf int-eq int-simps(22))
from 1 2 3 **show** ?thesis
by (metis TrueW int-simps(3) inteq-reflection)
qed

lemma OmegaLenPlusOneImplInf:
 $\vdash (len(Suc\ n))^\omega \longrightarrow inf$
by (simp add: Valid-def infinite-defs omega-d-def len-defs sum.case-eq-if)

lemma InfImpOmegaLenPlusOne:
 $\vdash inf \longrightarrow (len(Suc\ n))^\omega$
proof –
have 1: $\vdash inf \wedge \#True \wedge \Box(\#True \longrightarrow (len(Suc\ n) \wedge fmore); \#True) \longrightarrow (len(Suc\ n))^\omega$
using OmegaInduct **by** blast
have 2: $\vdash \Box(\#True \longrightarrow (len(Suc\ n) \wedge fmore); \#True) = inf$
by (simp add: Valid-def len-defs fmore-defs chop-defs iprefix-length infinite-defs always-defs sum.case-eq-if, auto)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma OmegaLenPlusOneEqvInf:
 $\vdash (len(Suc\ n))^\omega = inf$
using OmegaLenPlusOneImplInf InfImpOmegaLenPlusOne **by** fastforce

lemma OmegaSkipEqvInf:
 $\vdash (skip)^\omega = inf$
proof –
have 1: $\vdash skip = (len\ 1)$
by (simp add: Valid-def skip-defs len-defs sum.case-eq-if)
have 2: $\vdash (skip)^\omega = (len\ 1)^\omega$
using 1 **by** (metis OmegaUnroll inteq-reflection)
from 2 **show** ?thesis **using** OmegaLenPlusOneEqvInf **by** fastforce
qed

lemma OmegaTrueImplInf:
 $\vdash (\#True)^\omega \longrightarrow inf$
by (simp add: Valid-def infinite-defs omega-d-def skip-defs sum.case-eq-if)

lemma InfImpOmegaTrue:
 $\vdash inf \longrightarrow (\#True)^\omega$
proof –
have 1: $\vdash inf \wedge \#True \wedge \Box(\#True \longrightarrow (\#True \wedge fmore); \#True) \longrightarrow \#True^\omega$
using OmegaInduct **by** blast
have 2: $\vdash \Box(\#True \longrightarrow (\#True \wedge fmore); \#True) = inf$
by (simp add: Valid-def skip-defs fmore-defs chop-defs iprefix-length infinite-defs always-defs sum.case-eq-if, auto)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *OmegaTrueEqvInf*:

$\vdash (\# \text{True})^\omega = \text{inf}$

using *OmegaTrueImplInf ImplOmegaTrue* **by** *fastforce*

lemma *OmegaMoreImplInf*:

$\vdash (\text{more})^\omega \longrightarrow \text{inf}$

by (*simp add: Valid-def infinite-defs omega-d-def more-defs sum.case-eq-if*)

lemma *ImplOmegaMore*:

$\vdash \text{inf} \longrightarrow (\text{more})^\omega$

proof –

have 1: $\vdash \text{inf} \wedge \# \text{True} \wedge \Box(\# \text{True} \longrightarrow (\text{more} \wedge \text{fmore}); \# \text{True}) \longrightarrow \text{more}^\omega$

using *OmegaInduct* **by** *blast*

have 2: $\vdash \Box(\# \text{True} \longrightarrow (\text{more} \wedge \text{fmore}); \# \text{True}) = \text{inf}$

by (*simp add: Valid-def skip-defs more-defs fmore-defs chop-defs iprefix-length infinite-defs always-defs sum.case-eq-if, auto*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *OmegaMoreEqvInf*:

$\vdash (\text{more})^\omega = \text{inf}$

using *OmegaMoreImplInf ImplOmegaMore* **by** *fastforce*

lemma *OmegaFiniteImplInf*:

$\vdash (\text{finite})^\omega \longrightarrow \text{inf}$

by (*simp add: Valid-def infinite-defs omega-d-def more-defs sum.case-eq-if*)

lemma *ImplOmegaFinite*:

$\vdash \text{inf} \longrightarrow (\text{finite})^\omega$

proof –

have 1: $\vdash \text{inf} \wedge \# \text{True} \wedge \Box(\# \text{True} \longrightarrow (\text{finite} \wedge \text{fmore}); \# \text{True}) \longrightarrow \text{finite}^\omega$

using *OmegaInduct* **by** *blast*

have 2: $\vdash \Box(\# \text{True} \longrightarrow (\text{finite} \wedge \text{fmore}); \# \text{True}) = \text{inf}$

by (*simp add: Valid-def skip-defs more-defs finite-defs fmore-defs chop-defs iprefix-length infinite-defs always-defs sum.case-eq-if, auto*)

from 1 2 **show** *?thesis* **by** *fastforce*

qed

lemma *OmegaFiniteEqvInf*:

$\vdash (\text{finite})^\omega = \text{inf}$

using *OmegaFiniteImplInf ImplOmegaFinite* **by** *fastforce*

lemma *BoxStateAndImplOmegaBoxState*:

$\vdash \text{inf} \wedge \Box(\text{init } w) \longrightarrow (\Box(\text{init } w))^\omega$

proof –

have 1: $\vdash \text{inf} \wedge (\text{inf} \wedge \Box(\text{init } w)) \wedge$

$\Box((\text{inf} \wedge \Box(\text{init } w)) \longrightarrow (\Box(\text{init } w) \wedge \text{fmore}); (\text{inf} \wedge \Box(\text{init } w))) \longrightarrow (\Box(\text{init } w))^\omega$

using *OmegaInduct* **by** *blast*

have 2: $\vdash (\text{inf} \wedge \Box(\text{init } w)) = (\text{inf} \wedge (\Box(\text{init } w) \wedge \text{fmore}); \Box(\text{init } w))$

by (metis (no-types, lifting) BoxStateAndChopEqvChop ChopAndInf FmoreEqvSkipChopFinite
 OmegaFiniteEqvInf OmegaUnroll Prop10 SkipChopFiniteImpFinite inteq-reflection lift-and-com)
 have 3: $\vdash (inf \wedge (\Box(init\ w) \wedge fmore); \Box(init\ w)) = (\Box(init\ w) \wedge fmore); (inf \wedge \Box(init\ w))$
 by (metis ChopAndInf inteq-reflection lift-and-com)
 have 4: $\vdash inf \wedge \Box(init\ w) \longrightarrow \Box((inf \wedge \Box(init\ w)) \longrightarrow (\Box(init\ w) \wedge fmore); (inf \wedge \Box(init\ w)))$
 using 2 3 by (metis (mono-tags, lifting) BoxGen intD intl inteq-reflection unl-lift2)
 from 1 4 show ?thesis by fastforce
 qed

lemma OmegaBoxStateImpBoxState:

$\vdash (\Box(init\ w))^\omega \wedge inf \longrightarrow \Box(init\ w)$
 proof –
 have 1: $\vdash (\Box(init\ w))^\omega \longrightarrow init\ w$
 by (smt AndChopA BoxElim OmegaUnroll Prop10 Prop12 StateAndChop inteq-reflection lift-and-com)
 have 2: $\vdash (\Box(init\ w))^\omega \longrightarrow (\Box(init\ w) \wedge fmore); ((\Box(init\ w))^\omega)$
 by (simp add: OmegaUnroll int-iffD1)
 have 21: $\vdash (\Box(init\ w) \wedge fmore) \longrightarrow \bigcirc(\Box(init\ w))$
 by (metis AndChopB BoxStateAndChopEqvChop FmoreEqvSkipChopFinite NextAndEqvNextAndNext
 Prop12 inteq-reflection next-d-def)
 have 22: $\vdash finite = (empty \vee fmore)$
 by (simp add: Valid-def finite-defs empty-defs fmore-defs sum.case-eq-if, auto)
 have 23: $\vdash (\Box(init\ w) \wedge finite) = ((\Box(init\ w) \wedge empty) \vee (\Box(init\ w) \wedge fmore))$
 using 22 by fastforce
 have 24: $\vdash (\Box(init\ w) \wedge empty) = (init\ w \wedge empty)$
 using BoxEqvAndBox StateAndEmptyImpBoxState by fastforce
 have 25: $\vdash \bigcirc(\Box(init\ w)) \wedge fmore \longrightarrow \bigcirc((init\ w \wedge empty) \vee (\Box(init\ w) \wedge fmore))$
 using 23 24 by (metis FmoreEqvSkipChopFinite NextAndEqvNextAndNext SkipChopEqvNext int-iffD2
 inteq-reflection)
 have 26: $\bigwedge g. \vdash (\bigcirc((init\ w \wedge empty) \vee (\Box(init\ w) \wedge fmore))); g =$
 $(\bigcirc(init\ w \wedge g) \vee \bigcirc((\Box(init\ w) \wedge fmore); g))$
 by (metis (mono-tags, lifting) ChopOrEqvRule NextChop OrChopEqv StateAndEmptyChop
 inteq-reflection next-d-def)
 have 3: $\vdash (\Box(init\ w) \wedge fmore); ((\Box(init\ w))^\omega) \longrightarrow$
 $(\bigcirc(init\ w \wedge (\Box(init\ w))^\omega) \vee \bigcirc((\Box(init\ w) \wedge fmore); ((\Box(init\ w))^\omega)))$
 using 23 24 26
 by (metis AndChopB BoxStateAndChopEqvChop FmoreEqvSkipChopFinite LeftChopImpChop
 inteq-reflection next-d-def)
 have 4: $\vdash (\bigcirc(init\ w \wedge (\Box(init\ w))^\omega) \vee \bigcirc((\Box(init\ w) \wedge fmore); ((\Box(init\ w))^\omega))) \longrightarrow$
 $\bigcirc((\Box(init\ w))^\omega)$
 by (metis ChopAndB NextImpNext OmegaUnroll Prop02 Prop11 next-d-def)
 have 5: $\vdash (\Box(init\ w))^\omega \longrightarrow \bigcirc((\Box(init\ w))^\omega)$
 using 2 3 4 by fastforce
 from 1 5 show ?thesis using BoxIntro by (metis Prop01 Prop05 inteq-reflection lift-and-com)
 qed

lemma OmegaIntro:

assumes $\vdash h \longrightarrow (f \wedge fmore); h$
 shows $\vdash h \wedge inf \longrightarrow f^\omega$
 proof –

have 1: $\vdash h \longrightarrow (f \wedge fmore);h$ **using** *assms* **by** *auto*
have 2: $\vdash \Box (h \longrightarrow (f \wedge fmore);h)$ **by** (*simp add: BoxGen assms*)
from 1 2 **show** ?thesis **using** *OmegalInduct* **by** *fastforce*
qed

lemma *OmegalImpRule*:

assumes $\vdash f \longrightarrow g$
shows $\vdash f^\omega \wedge inf \longrightarrow g^\omega$
proof –
have 1: $\vdash (f \wedge fmore) \longrightarrow (g \wedge fmore)$
using *assms* **by** *auto*
have 2: $\vdash (f \wedge fmore);f^\omega \longrightarrow (g \wedge fmore);f^\omega$
using 1 **by** (*simp add: LeftChopImpChop*)
have 3: $\vdash \Box(f^\omega \longrightarrow (f \wedge fmore);f^\omega) \longrightarrow \Box(f^\omega \longrightarrow (g \wedge fmore);f^\omega)$
using 2 **by** (*smt BoxImpBoxRule Prop10 intl int-eq unl-lift2*)
have 4: $\vdash (inf \wedge f^\omega \wedge \Box(f^\omega \longrightarrow (f \wedge fmore);f^\omega)) \longrightarrow$
 $inf \wedge f^\omega \wedge \Box(f^\omega \longrightarrow (g \wedge fmore);f^\omega)$
using 3 **by** *fastforce*
have 5: $\vdash inf \wedge f^\omega \wedge \Box(f^\omega \longrightarrow (g \wedge fmore);f^\omega) \longrightarrow g^\omega$
using *OmegalInduct* **by** *blast*
have 6: $\vdash f^\omega \longrightarrow (f \wedge fmore);f^\omega$
by (*simp add: OmegaUnroll int-iffD1*)
have 7: $\vdash \Box(f^\omega \longrightarrow (f \wedge fmore);f^\omega)$
using 6 **by** (*simp add: BoxGen*)
from 3 5 7 **show** ?thesis **by** *fastforce*
qed

lemma *OmegaEqvRule*:

assumes $\vdash f = g$
shows $\vdash f^\omega = g^\omega$
using *assms* **using** *int-eq* **by** *force*

lemma *AndOmegaA*:

$\vdash (f \wedge g)^\omega \wedge inf \longrightarrow f^\omega$
by (*meson OmegalImpRule Prop12 int-iffD2 lift-and-com*)

lemma *AndOmegaB*:

$\vdash (f \wedge g)^\omega \wedge inf \longrightarrow g^\omega$
by (*meson OmegalImpRule Prop12 int-iffD2 lift-and-com*)

lemma *BaOmegalImpOmega*:

$\vdash ba (f \longrightarrow g) \wedge inf \longrightarrow f^\omega \longrightarrow g^\omega$
proof –
have 1: $\vdash ba (f \longrightarrow g) \wedge (f \wedge fmore);f^\omega \longrightarrow ((f \longrightarrow g) \wedge (f \wedge fmore));((f \longrightarrow g) \wedge f^\omega)$
using *BaAndChopImport* **by** *fastforce*
have 2: $\vdash (f \longrightarrow g) \wedge (f \wedge fmore) \longrightarrow (g \wedge fmore)$
by *auto*
have 3: $\vdash (f \longrightarrow g) \wedge f^\omega \longrightarrow f^\omega$

```

  by auto
have 4:  $\vdash ba(f \longrightarrow g) \wedge (f \wedge fmore); f^\omega \longrightarrow (g \wedge fmore); f^\omega$ 
  using 1 2 3
  by (metis (no-types, lifting) AndChopB ChopAndB Prop10 int-eq lift-imp-trans)
have 5:  $\vdash ba(f \longrightarrow g) \wedge (f \wedge fmore); f^\omega \longrightarrow ((f \longrightarrow g) \wedge (f \wedge fmore)); (ba(f \longrightarrow g) \wedge f^\omega)$ 
  using BaAndChopImportB by blast
have 6:  $\vdash ((f \longrightarrow g) \wedge (f \wedge fmore)); (ba(f \longrightarrow g) \wedge f^\omega) \longrightarrow$ 
   $((g \wedge fmore)); (ba(f \longrightarrow g) \wedge f^\omega)$ 
  using 2 LeftChopImpChop by blast
have 7:  $\vdash (ba(f \longrightarrow g) \wedge f^\omega) \longrightarrow (g \wedge fmore); (ba(f \longrightarrow g) \wedge f^\omega)$ 
  using OmegaUnroll 5 6 by fastforce
have 8:  $\vdash (ba(f \longrightarrow g) \wedge f^\omega) \wedge inf \longrightarrow g^\omega$ 
  using 7 OmegaIntro by blast
from 8 show ?thesis by fastforce
qed

```

```

lemma BaOmegaEqvOmega:
 $\vdash ba(f = g) \wedge inf \longrightarrow (f^\omega = g^\omega)$ 
proof -
  have 1:  $\vdash ba(f = g) = (ba(f \longrightarrow g) \wedge ba(g \longrightarrow f))$  by (auto simp: ba-defs sum.case-eq-if)
  have 2:  $\vdash ba(f \longrightarrow g) \wedge inf \longrightarrow (f^\omega \longrightarrow g^\omega)$  using BaOmegaImpOmega by blast
  have 3:  $\vdash ba(g \longrightarrow f) \wedge inf \longrightarrow (g^\omega \longrightarrow f^\omega)$  using BaOmegaImpOmega by blast
  have 4:  $\vdash ba(f = g) \wedge inf \longrightarrow (f^\omega \longrightarrow g^\omega) \wedge (g^\omega \longrightarrow f^\omega)$  using 1 2 3 by fastforce
  have 5:  $\vdash ((f^\omega \longrightarrow g^\omega) \wedge (g^\omega \longrightarrow f^\omega)) = (f^\omega = g^\omega)$  by auto
  from 4 5 show ?thesis by auto
qed

```

```

lemma BaAndOmegaImport:
 $\vdash ba f \wedge g^\omega \wedge inf \longrightarrow (f \wedge g)^\omega$ 
proof -
  have 1:  $\vdash f \longrightarrow (g \longrightarrow (f \wedge g))$  by auto
  hence 2:  $\vdash ba f \longrightarrow ba(g \longrightarrow f \wedge g)$  by (rule BalmpBa)
  have 3:  $\vdash ba(g \longrightarrow f \wedge g) \wedge inf \longrightarrow g^\omega \longrightarrow (f \wedge g)^\omega$  by (rule BaOmegaImpOmega)
  from 2 3 show ?thesis by fastforce
qed

```

9.10 Properties of While

```

lemma WhileEqvIf:
 $\vdash ((while (init w) do f) \wedge finite) =$ 
 $(if; (init w) then (f; (while (init w) do f)) else empty \wedge finite)$ 
proof -
  have 1:  $\vdash (while (init w) do f \wedge finite) =$ 
 $(((((init w) \wedge f)^* \wedge fin (\neg (init w)))) \wedge finite)$ 
  by (simp add: while-d-def)
  have 2:  $\vdash (init w \wedge f)^* = (empty \vee ((init w \wedge f); (init w \wedge f)^*))$ 
  by (rule CSeqvOrChopCS)
  have 21:  $\vdash (((init w) \wedge f)^* \wedge fin (\neg (init w)) \wedge finite) =$ 
 $((empty \vee ((init w \wedge f); (init w \wedge f)^*)) \wedge fin (\neg (init w)) \wedge finite)$ 
  using 2 by fastforce

```


have 22: $\vdash ((\text{empty} \vee ((\text{init } w \wedge f); (\text{init } w \wedge f)^*)) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$
 $((\text{empty} \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) \vee$
 $((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))$
by auto
have 23: $\vdash (((\text{init } w) \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite} =$
 $((\text{empty} \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) \vee$
 $((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))$
using 21 22 by auto
have 3: $\vdash (\text{empty} \wedge \text{fin } (\neg (\text{init } w))) = (\neg (\text{init } w) \wedge \text{empty})$
by (metis FinAndEmpty Prop04 lift-and-com)
hence 31: $\vdash (\text{empty} \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) = (\neg (\text{init } w) \wedge \text{empty} \wedge \text{finite})$
by auto
have 32: $\vdash (\neg (\text{init } w) \wedge \text{empty} \wedge \text{finite}) = (\neg (\text{init } w) \wedge \text{empty})$
using FiniteAndEmptyEqvEmpty by auto
have 33: $\vdash (\text{empty} \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) = (\neg (\text{init } w) \wedge \text{empty})$
using 31 32 by fastforce
have 34: $\vdash ((\text{empty} \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) \vee$
 $((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))$
using 23 33 by fastforce
have 4: $\vdash (\text{init } w \wedge f); (\text{init } w \wedge f)^* = (\text{init } w \wedge (f; (\text{init } w \wedge f)^*))$
by (rule StateAndChop)
have 41: $\vdash (((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$
 $(\text{init } w \wedge (f; (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})$
using 4 by auto
have 42: $\vdash (\text{init } w \wedge ((f; (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})) =$
 $(\text{init } w \wedge ((f; (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))$
using Initprop(2) by (metis StateAndEmptyChop int-eq)
have 5: $\vdash ((f; ((\text{init } w \wedge f)^*)) \wedge (\text{fin } (\neg (\text{init } w))) \wedge \text{finite})$
 $= ((f \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge (\text{fin } (\neg (\text{init } w))) \wedge \text{finite}))$
using ChopAndFin by fastforce
hence 49: $\vdash (\text{init } w \wedge (f; (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$
 $(\text{init } w \wedge (f \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge (\text{fin } (\neg (\text{init } w))) \wedge \text{finite}))$
using 42 by fastforce
have 50: $\vdash (((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$
 $(\text{init } w \wedge (f \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge (\text{fin } (\neg (\text{init } w))) \wedge \text{finite}))$
using 49 41 by fastforce
have 51: $\vdash (\text{init } w \wedge (f \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge (\text{fin } (\neg (\text{init } w))) \wedge \text{finite})) =$
 $(\text{init } w \wedge (f \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge (\text{fin } (\neg (\text{init } w))) \wedge \text{finite}))$
using Initprop(2) by (smt RightChopEqvChop int-eq lift-and-com)
have 52: $\vdash (((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$
 $(\text{init } w \wedge ((f \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})))$
using 50 51 by fastforce
have 53: $\vdash ((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $((\text{init } w \wedge f); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((f \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))))$
using 52 34 by auto
have 6: $\vdash ((f \wedge \text{finite}); (((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite})) =$

$(f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})$
by (*simp add: while-d-def*)
have 61: $\vdash (\text{init } w \wedge ((f \wedge \text{finite}); (((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)))) \wedge \text{finite}))) =$
 $(\text{init } w \wedge ((f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})))$
using 6
by *auto*
have 62: $\vdash ((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((f \wedge \text{finite}); (((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)))) \wedge \text{finite}))) \vee$
 $= ((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((f \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}))))$
using 61 **by** *fastforce*
have 7: $\vdash (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})$
 $= (((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge (f ; \text{while } (\text{init } w) \text{ do } f) \wedge \text{finite})))$
using 1 23 34 53 62
by (*smt 21 22 ChopAndFiniteDist Prop10 Prop12 int-eq int-iffD2*)
have 71: $\vdash ((\text{if}_i (\text{init } w) \text{ then } (f ; (\text{while } (\text{init } w) \text{ do } f)) \text{ else } \text{empty}) \wedge \text{finite}) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f ; \text{while } (\text{init } w) \text{ do } f) \wedge \text{finite}))$
using *FiniteAndEmptyEqvEmpty* **by** (*auto simp: ifthenelse-d-def*)
from 7 71 **show** ?thesis **by** *fastforce*
qed

lemma *IfAndFiniteDist*:

$\vdash (\text{if}_i (\text{init } w) \text{ then } (f ; g) \text{ else } \text{empty} \wedge \text{finite}) =$
 $(\text{if}_i (\text{init } w) \text{ then } ((f \wedge \text{finite}); (g \wedge \text{finite})) \text{ else } \text{empty})$
proof –
have 1: $\vdash (\text{if}_i (\text{init } w) \text{ then } (f ; g) \text{ else } \text{empty} \wedge \text{finite}) =$
 $((\text{init } w \wedge (f ; g)) \vee (\neg (\text{init } w) \wedge \text{empty})) \wedge \text{finite}$
by (*auto simp: ifthenelse-d-def*)
have 2: $\vdash ((\text{init } w \wedge (f ; g)) \vee (\neg (\text{init } w) \wedge \text{empty})) \wedge \text{finite} =$
 $((\text{init } w \wedge (f ; g) \wedge \text{finite}) \vee (\neg (\text{init } w) \wedge \text{empty} \wedge \text{finite}))$
by *auto*
have 3: $\vdash (\text{init } w \wedge (f ; g) \wedge \text{finite}) = (\text{init } w \wedge (f \wedge \text{finite}); (g \wedge \text{finite}))$
using *ChopAndFiniteDist* **by** *fastforce*
have 4: $\vdash (\neg (\text{init } w) \wedge \text{empty} \wedge \text{finite}) = (\neg (\text{init } w) \wedge \text{empty})$
using *FiniteAndEmptyEqvEmpty* **by** *auto*
have 5: $\vdash ((\text{init } w \wedge (f ; g) \wedge \text{finite}) \vee (\neg (\text{init } w) \wedge \text{empty} \wedge \text{finite})) =$
 $((\text{init } w \wedge (f \wedge \text{finite}); (g \wedge \text{finite})) \vee (\neg (\text{init } w) \wedge \text{empty}))$
using 3 4 **by** *fastforce*
have 6: $\vdash ((\text{init } w \wedge (f \wedge \text{finite}); (g \wedge \text{finite})) \vee (\neg (\text{init } w) \wedge \text{empty})) =$
 $(\text{if}_i (\text{init } w) \text{ then } ((f \wedge \text{finite}); (g \wedge \text{finite})) \text{ else } \text{empty})$
by (*auto simp: ifthenelse-d-def*)
from 1 2 5 6 **show** ?thesis **by** (*metis inteq-reflection*)
qed

lemma *WhileChopEqvIf*:

$\vdash ((\text{while } (\text{init } w) \text{ do } f) \wedge \text{finite}); g =$
 $\text{if}_i (\text{init } w) \text{ then } ((f \wedge \text{finite}); ((\text{while } (\text{init } w) \text{ do } f) \wedge \text{finite}); g) \text{ else } g$
proof –
have 1: $\vdash (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) =$

(if_i (init w) then (f; (while (init w) do f)) else empty ∧ finite)
 by (rule WhileEqvIf)
have 11: ⊢ (if_i (init w) then (f; (while (init w) do f)) else empty ∧ finite) =
 (if_i (init w) then ((f ∧ finite); (while (init w) do f ∧ finite)) else empty)
 using IfAndFiniteDist by fastforce
have 12: ⊢ (while (init w) do f ∧ finite) =
 (if_i (init w) then ((f ∧ finite); (while (init w) do f ∧ finite)) else empty)
 using 1 11 by fastforce
hence 2: ⊢ ((while (init w) do f) ∧ finite); g =
 (if_i (init w)
 then (((f ∧ finite); (while (init w) do f ∧ finite));g)
 else (empty;g))
 by (rule IfChopEqvRule)
have 3: ⊢ empty ; g = g
 by (rule EmptyChop)
have 4: ⊢ (if_i (init w)
 then (((f ∧ finite); (while (init w) do f ∧ finite));g)
 else (empty;g)) =
 (if_i (init w)
 then (((f ∧ finite); (while (init w) do f ∧ finite));g)
 else g)
 using 3 using inteq-reflection by fastforce
have 5: ⊢ (((f ∧ finite); (while (init w) do f ∧ finite));g) =
 ((f ∧ finite);((while (init w) do f ∧ finite);g))
 by (rule ChopAssocB)
have 6: ⊢ (if_i (init w)
 then (((f ∧ finite); (while (init w) do f ∧ finite));g)
 else g) =
 (if_i (init w)
 then ((f ∧ finite);((while (init w) do f ∧ finite);g))
 else g)

 using 5 using inteq-reflection by fastforce
from 1 2 4 6 **show** ?thesis by fastforce
qed

lemma WhileChopEqvIfRule:

assumes ⊢ f = (while (init w) do g ∧ finite); h
shows ⊢ f = if_i (init w) then ((g ∧ finite); f) else h

proof –

have 1: ⊢ f = (while (init w) do g ∧ finite); h
 using assms by auto
have 2: ⊢ (while (init w) do g ∧ finite); h =
 if_i (init w) then ((g ∧ finite); ((while (init w) do g ∧ finite); h)) else h
 by (rule WhileChopEqvIf)
have 3: ⊢ ((g ∧ finite); f) = ((g ∧ finite); ((while (init w) do g ∧ finite); h))
 using 1 by (rule RightChopEqvChop)
have 4: ⊢ ((g ∧ finite); ((while (init w) do g ∧ finite); h)) = ((g ∧ finite); f)

using 3 by auto
 have 5: $\vdash \text{if}_i (\text{init } w) \text{ then } ((g \wedge \text{finite}); ((\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}); h)) \text{ else } h =$
 $\text{if}_i (\text{init } w) \text{ then } ((g \wedge \text{finite}); f) \text{ else } h$
 using 4 using inteq-reflection by fastforce
 from 1 2 5 show ?thesis by fastforce
 qed

lemma WhileImpFin:

$\vdash \text{while } (\text{init } w) \text{ do } f \longrightarrow \text{fin } (\neg (\text{init } w))$
proof –
 have 1: $\vdash (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \longrightarrow \text{fin } (\neg (\text{init } w))$ by auto
 from 1 show ?thesis by (simp add: while-d-def)
 qed

lemma WhileEqvEmptyOrChopWhile:

$\vdash (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})))$
proof –
 have 1: $\vdash (\text{init } w \wedge f)^* = (\text{empty} \vee ((\text{init } w \wedge f) \wedge \text{more}); (\text{init } w \wedge f)^*)$
 by (rule ChopstarEqv)
 have 2: $\vdash ((\text{init } w \wedge f) \wedge \text{more}) = (\text{init } w \wedge (f \wedge \text{more}))$
 by auto
 hence 3: $\vdash ((\text{init } w \wedge f) \wedge \text{more}); (\text{init } w \wedge f)^* = (\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^*$
 by (rule LeftChopEqvChop)
 have 4: $\vdash (\text{init } w \wedge f)^* = (\text{empty} \vee (\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^*)$
 using 1 3 by fastforce
 have 5: $\vdash ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite} =$
 $((\text{empty} \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite}) \vee$
 $((\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite})$
 using 1 4 by fastforce
 have 51: $\vdash ((\text{empty} \wedge \text{fin } (\neg (\text{init } w)))) \wedge \text{finite} = ((\neg (\text{init } w) \wedge \text{empty}) \wedge \text{finite})$
 by (metis FinAndEmpty inteq-reflection lift-and-com)
 have 52: $\vdash ((\neg (\text{init } w) \wedge \text{empty}) \wedge \text{finite}) = (\neg (\text{init } w) \wedge \text{empty})$
 using EmptyImpFinite by auto
 have 6: $\vdash ((\text{empty} \wedge \text{fin } (\neg (\text{init } w)))) \wedge \text{finite} = (\neg (\text{init } w) \wedge \text{empty})$
 using 51 52 by fastforce
 have 61: $\vdash ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite} =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $((\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite})$
 using 5 6 by fastforce
 have 70: $\vdash (\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^* = (\text{init } w \wedge (f \wedge \text{more}); (\text{init } w \wedge f)^*)$
 by (rule StateAndChop)
 have 7: $\vdash ((\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite} =$
 $(\text{init } w \wedge (f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite})$
 using 70 by auto
 have 71: $\vdash ((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $((\text{init } w \wedge f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite}) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge (f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w))) \wedge \text{finite})$

```

using 7 by (smt 61 Initprop(2) inteq-reflection)
have 8:  $\vdash ((f \wedge \text{more}); (\text{init } w \wedge f)^*) \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite} =$ 
 $((f \wedge \text{more}) \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))$ 
using ChopAndFin by fastforce
have 81:  $\vdash \text{fin } (\neg (\text{init } w)) = \text{fin } (\neg (\text{init } w))$ 
using FinEqvFin Initprop(2) by fastforce
have 82:  $\vdash ((f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$ 
 $((f \wedge \text{more}) \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))$ 
using 8 81
by (metis inteq-reflection)
have 83:  $\vdash (\text{init } w \wedge (f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$ 
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))$ 
using 82 by fastforce
have 84:  $\vdash ((\neg (\text{init } w) \wedge \text{empty}) \vee$ 
 $(\text{init } w \wedge (f \wedge \text{more}); (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})) =$ 
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$ 
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite}); ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})))$ 
using 83 by (metis 71 81 inteq-reflection)
have 9:  $\vdash ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$ 
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$ 
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite});$ 
 $((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})))$ 
using 84 61 by (metis 71 inteq-reflection)
have 10:  $\vdash ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$ 
 $((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})$ 
by auto
hence 11:  $\vdash ((\neg (\text{init } w) \wedge \text{empty}) \vee$ 
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite});$ 
 $((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}))) =$ 
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$ 
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite});$ 
 $((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})))$ 
by (metis 84 inteq-reflection)
have 12:  $\vdash ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite}) =$ 
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$ 
 $(\text{init } w \wedge ((f \wedge \text{more}) \wedge \text{finite});$ 
 $((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \wedge \text{finite})))$ 
using 11 9 by fastforce
from 12 show ?thesis by (metis 10 inteq-reflection while-d-def)
qed

```

lemma WhileIntro:

```

assumes  $\vdash \neg (\text{init } w) \wedge f \longrightarrow \text{empty}$ 
 $\vdash \text{init } w \wedge f \longrightarrow ((g \wedge \text{more}) \wedge \text{finite}); f$ 
shows  $\vdash f \wedge \text{finite} \longrightarrow \text{while } (\text{init } w) \text{ do } g$ 
proof -
have 1:  $\vdash \neg (\text{init } w) \wedge f \longrightarrow \text{empty}$ 
using assms by blast

```

have 2: $\vdash \text{init } w \wedge f \longrightarrow ((g \wedge \text{more}) \wedge \text{finite}); f$
using *assms by blast*
have 3: $\vdash (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}) =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite})))$
by (*rule WhileEqvEmptyOrChopWhile*)
hence 31: $\vdash \neg (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}) =$
 $(\neg (\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}))))$
by *fastforce*
hence 32: $\vdash (f \wedge \neg (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite})) =$
 $(f \wedge \neg (\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}))))$
by *fastforce*
have 33: $\vdash (f \wedge \neg (\neg (\text{init } w) \wedge \text{empty}) \vee$
 $(\text{init } w \wedge ((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite})))) =$
 $(f \wedge \neg (\neg (\text{init } w) \wedge \text{empty}) \wedge$
 $\neg (\text{init } w \wedge ((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}))))$
by *auto*
have 34: $\vdash (f \wedge \neg (\neg (\text{init } w) \wedge \text{empty}) \wedge$
 $\neg ((\text{init } w) \wedge (((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite})))) =$
 $(f \wedge ((\text{init } w) \vee \text{more}) \wedge$
 $(\neg (\text{init } w) \vee \neg (((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}))))$
by (*auto simp: empty-d-def*)
have 35: $\vdash (f \wedge ((\text{init } w) \vee \text{more}) \wedge$
 $(\neg (\text{init } w) \vee \neg (((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite})))) =$
 $((f \wedge (\text{init } w) \wedge \neg (((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}))) \vee$
 $(f \wedge (\text{init } w) \wedge \neg (\text{init } w)) \vee$
 $(f \wedge \text{more} \wedge \neg (((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}))) \vee$
 $(f \wedge \text{more} \wedge \neg (\text{init } w)))$
by *auto*
have 36: $\vdash (f \wedge \neg (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite})) =$
 $((f \wedge (\text{init } w) \wedge \neg (((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}))) \vee$
 $(f \wedge (\text{init } w) \wedge \neg (\text{init } w)) \vee$
 $(f \wedge \text{more} \wedge \neg (((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}))) \vee$
 $(f \wedge \text{more} \wedge \neg (\text{init } w)))$ **using** 32 33 34 35 **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: $\vdash (f \wedge \text{more} \wedge \neg (((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}))) \longrightarrow$
 $((g \wedge \text{more}) \wedge \text{finite}); f \wedge$
 $\neg (((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite})))$
using 1 2 **by** (*auto simp: empty-d-def Valid-def*)
have 39: $\vdash (f \wedge (\text{init } w) \wedge \neg (((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}))) \longrightarrow$
 $((g \wedge \text{more}) \wedge \text{finite}); f \wedge$
 $\neg (((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite})))$
using 2 **by** *auto*
have 40: $\vdash ((f \wedge (\text{init } w) \wedge \neg (((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}))) \vee$
 $(f \wedge (\text{init } w) \wedge \neg (\text{init } w)) \vee$
 $(f \wedge \text{more} \wedge \neg (((g \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } g \wedge \text{finite}))) \vee$
 $(f \wedge \text{more} \wedge \neg (\text{init } w))) \longrightarrow$

```

      ((g ∧ more) ∧ finite); f ∧
      ¬ (((g ∧ more) ∧ finite); (while (init w) do g ∧ finite))
using 39 38 37 38 by fastforce
have 4: ⊢ f ∧ ¬ (while (init w) do g ∧ finite) →
      ((g ∧ more) ∧ finite); f ∧
      ¬ (((g ∧ more) ∧ finite); (while (init w) do g ∧ finite))
using 36 40 by fastforce
have 50: ⊢ g ∧ more → more
by auto
have 5: ⊢ (g ∧ more) ∧ finite → more
by (simp add: 50 Prop05 Prop07 finite-d-def)
have 6: ⊢ f ∧ finite → (while (init w) do g ∧ finite)
using 4 5 ChopContraB by blast
from 6 show ?thesis by (simp add: Prop12)
qed

```

lemma WhileElim:

```

assumes ⊢ ¬ (init w) ∧ empty → g
          ⊢ init w ∧ ((f ∧ more) ∧ finite); g → g
shows ⊢ while (init w) do f ∧ finite → g
proof —
have 1: ⊢ (while (init w) do f ∧ finite) =
      ((¬ (init w) ∧ empty) ∨
      (init w ∧ ((f ∧ more) ∧ finite); (while (init w) do f ∧ finite)))
by (rule WhileEqvEmptyOrChopWhile)
hence 11: ⊢ ((while (init w) do f ∧ finite) ∧ ¬ g) =
      (((¬ (init w) ∧ empty) ∨
      (init w ∧ ((f ∧ more) ∧ finite); (while (init w) do f ∧ finite))) ∧ ¬ g)
by auto
have 2: ⊢ ¬ (init w) ∧ empty → g
using assms by blast
hence 21: ⊢ ¬ g → ¬ (¬ (init w) ∧ empty)
by auto
have 22: ⊢ ((¬ (init w) ∧ empty) ∨
      (init w ∧ ((f ∧ more) ∧ finite); (while (init w) do f ∧ finite))) ∧ ¬ g →
      (init w ∧ ((f ∧ more) ∧ finite); (while (init w) do f ∧ finite))
using 21 by auto
have 23: ⊢ (while (init w) do f ∧ finite) ∧ ¬ g →
      (init w ∧ ((f ∧ more) ∧ finite); (while (init w) do f ∧ finite)) ∧ ¬ g
using 11 21 by fastforce
have 3: ⊢ (init w) ∧ (((f ∧ more) ∧ finite); g) → g
using assms by blast
hence 31: ⊢ ¬ g → ¬ ((init w) ∧ (((f ∧ more) ∧ finite); g))
by fastforce
have 32: ⊢ (init w ∧ ((f ∧ more) ∧ finite); (while (init w) do f ∧ finite)) ∧ ¬ g →
      (((f ∧ more) ∧ finite); (while (init w) do f ∧ finite)) ∧
      ¬ (((f ∧ more) ∧ finite); g) ∧ ¬ g
using 31 by auto

```

```

have 4:  $\vdash (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) \wedge \neg g \longrightarrow$ 
   $((f \wedge \text{more}) \wedge \text{finite}); (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite})) \wedge$ 
   $\neg (((f \wedge \text{more}) \wedge \text{finite}); g)$ 
using 23 32 by fastforce
have 5:  $\vdash (f \wedge \text{more}) \wedge \text{finite} \longrightarrow \text{more}$ 
by auto
from 4 5 show ?thesis using ChopContraB
by (smt Prop10 Prop12 int-iffD2 inteq-reflection lift-and-com)
qed

```

lemma BaWhileImpWhile:

$\vdash \text{ba } (f \longrightarrow g) \wedge \text{finite} \longrightarrow (\text{while } (\text{init } w) \text{ do } f) \longrightarrow (\text{while } (\text{init } w) \text{ do } g)$

proof –

```

have 1:  $\vdash (f \longrightarrow g) \longrightarrow ((\text{init } w \wedge f) \longrightarrow (\text{init } w \wedge g))$ 
by auto
hence 2:  $\vdash \text{ba } (f \longrightarrow g) \longrightarrow \text{ba } ((\text{init } w \wedge f) \longrightarrow (\text{init } w \wedge g))$ 
using BaImpBa by blast
have 3:  $\vdash \text{ba } ((\text{init } w \wedge f) \longrightarrow (\text{init } w \wedge g)) \wedge \text{finite} \longrightarrow ((\text{init } w \wedge f)^* \longrightarrow (\text{init } w \wedge g)^*)$ 
by (rule BaCSImpCS)
have 4:  $\vdash \text{ba } (f \longrightarrow g) \wedge \text{finite} \longrightarrow ((\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)))$ 
   $\longrightarrow (\text{init } w \wedge g)^* \wedge \text{fin } (\neg (\text{init } w)))$ 
using 2 3 by fastforce
from 4 show ?thesis by (simp add: while-d-def)
qed

```

lemma WhileImpWhile:

assumes $\vdash f \longrightarrow g$
shows $\vdash (\text{while } (\text{init } w) \text{ do } f) \wedge \text{finite} \longrightarrow (\text{while } (\text{init } w) \text{ do } g)$

proof –

```

have 1:  $\vdash f \longrightarrow g$ 
using assms by auto
hence 2:  $\vdash \text{ba } (f \longrightarrow g)$ 
by (rule BaGen)
have 3:  $\vdash \text{ba } (f \longrightarrow g) \wedge \text{finite} \longrightarrow (\text{while } (\text{init } w) \text{ do } f) \longrightarrow (\text{while } (\text{init } w) \text{ do } g)$ 
by (rule BaWhileImpWhile)
have 4:  $\vdash \text{ba } (f \longrightarrow g) \longrightarrow (\text{while } (\text{init } w) \text{ do } f \wedge \text{finite}) \longrightarrow (\text{while } (\text{init } w) \text{ do } g)$ 
using 3 by (auto simp: Valid-def)
from 2 4 show ?thesis using MP by blast
qed

```

9.11 Properties of Halt

lemma WnextAndMoreEqvNext:

$\vdash (\text{wnext } f \wedge \text{more}) = \bigcirc f$

by (auto simp: wnext-defs more-defs next-defs sum.case-eq-if)

lemma *BoxStateAndEmptyEqvStateAndEmpty*:

$\vdash (\Box(\text{empty} = (\text{init } w)) \wedge \text{empty}) = ((\text{init } w) \wedge \text{empty})$

by (*auto simp: always-defs init-defs empty-defs sum.case-eq-if*)

lemma *BoxEmptyEqvIStateqvEmptyAndStateOrNotStateNext*:

$\vdash \Box(\text{empty} = (\text{init } w)) = ((\text{empty} \wedge \text{init } w) \vee (\neg(\text{init } w) \wedge \bigcirc(\Box(\text{empty} = (\text{init } w)))))$

proof –

have 1: $\vdash \Box(\text{empty} = (\text{init } w)) =$

$((\Box(\text{empty} = (\text{init } w)) \wedge \text{empty}) \vee (\Box(\text{empty} = (\text{init } w)) \wedge \text{more}))$

by (*auto simp: empty-d-def*)

have 2: $\vdash (\Box(\text{empty} = (\text{init } w)) \wedge \text{empty}) = ((\text{init } w) \wedge \text{empty})$

using *BoxStateAndEmptyEqvStateAndEmpty* **by** *blast*

have 3: $\vdash \Box(\text{empty} = (\text{init } w)) = ((\text{empty} = (\text{init } w)) \wedge \text{wnext}(\Box(\text{empty} = (\text{init } w))))$

using *BoxEqvAndWnextBox* **by** *blast*

hence 4: $\vdash (\Box(\text{empty} = (\text{init } w)) \wedge \text{more}) =$

$((\text{empty} = (\text{init } w)) \wedge \text{more}) \wedge (\text{wnext}(\Box(\text{empty} = (\text{init } w))) \wedge \text{more}))$

by *auto*

have 5: $\vdash ((\text{empty} = (\text{init } w)) \wedge \text{more}) = (\neg(\text{init } w) \wedge \text{more})$

by (*auto simp: empty-d-def*)

have 6: $\vdash (\text{wnext}(\Box(\text{empty} = (\text{init } w))) \wedge \text{more}) = \bigcirc(\Box(\text{empty} = (\text{init } w)))$

using *WnextAndMoreEqvNext* **by** *metis*

have 7: $\vdash (\Box(\text{empty} = (\text{init } w)) \wedge \text{more}) =$

$((\neg(\text{init } w) \wedge \text{more}) \wedge (\text{wnext}(\Box(\text{empty} = (\text{init } w))) \wedge \text{more}))$

using 4 5 **by** *fastforce*

have 8: $\vdash ((\neg(\text{init } w) \wedge \text{more}) \wedge (\text{wnext}(\Box(\text{empty} = (\text{init } w))) \wedge \text{more})) =$

$((\neg(\text{init } w)) \wedge (\text{wnext}(\Box(\text{empty} = (\text{init } w))) \wedge \text{more}))$ **by** *auto*

have 9: $\vdash ((\neg(\text{init } w)) \wedge (\text{wnext}(\Box(\text{empty} = (\text{init } w))) \wedge \text{more})) =$

$((\neg(\text{init } w)) \wedge \bigcirc(\Box(\text{empty} = (\text{init } w))))$ **using** 8 6 **by** *auto*

have 10: $\vdash \Box(\text{empty} = (\text{init } w)) = (((\text{init } w) \wedge \text{empty}) \vee (\Box(\text{empty} = (\text{init } w)) \wedge \text{more}))$

using 1 2 **by** *fastforce*

from 7 9 10 **show** *?thesis* **by** *fastforce*

qed

lemma *HaltStateEqvIfStateThenEmptyElseNext*:

$\vdash \text{halt}(\text{init } w) = \text{if}_i(\text{init } w) \text{ then } \text{empty} \text{ else } (\bigcirc(\text{halt}(\text{init } w)))$

proof –

have 1: $\vdash \text{halt}(\text{init } w) = \Box(\text{empty} = (\text{init } w))$

by (*simp add: halt-d-def*)

have 2: $\vdash \Box(\text{empty} = (\text{init } w)) =$

$((\text{empty} \wedge \text{init } w) \vee (\neg(\text{init } w) \wedge \bigcirc(\Box(\text{empty} = (\text{init } w)))))$

by (*rule BoxEmptyEqvIStateqvEmptyAndStateOrNotStateNext*)

have 21: $\vdash ((\text{empty} \wedge \text{init } w) \vee (\neg(\text{init } w) \wedge \bigcirc(\Box(\text{empty} = (\text{init } w))))) =$

$((\text{init } w \wedge \text{empty}) \vee (\neg(\text{init } w) \wedge \bigcirc(\Box(\text{empty} = (\text{init } w)))))$

by *auto*

have 22: $\vdash \bigcirc(\text{halt}(\text{init } w)) = \bigcirc(\Box(\text{empty} = (\text{init } w)))$

using *NextEqvNext* **using** 1 **by** *blast*

have 3: $\vdash \text{if}_i(\text{init } w) \text{ then } \text{empty} \text{ else } (\bigcirc(\text{halt}(\text{init } w))) =$

$((\text{init } w \wedge \text{empty}) \vee (\neg(\text{init } w) \wedge \bigcirc(\text{halt}(\text{init } w))))$

by (*simp add: ifthenelse-d-def*)

from 1 2 21 22 3 **show** *?thesis* **by** *fastforce*

qed

lemma *HaltChopEqv*:

$\vdash ((\text{halt } (init\ w)); f) = (\text{if}_i\ (init\ w)\ \text{then } (f)\ \text{else } (\bigcirc(\text{halt } (init\ w)); f)))$

proof –

have 1: $\vdash \text{halt}(init\ w) =$
 $(\text{if}_i\ (init\ w)\ \text{then } \text{empty}\ \text{else } (\bigcirc(\text{halt } (init\ w))))$

by (rule *HaltStateEqvIfStateThenEmptyElseNext*)

hence 2: $\vdash ((\text{halt}(init\ w)); f) =$

$(\text{if}_i\ (init\ w)\ \text{then } (\text{empty}; f)\ \text{else } (\bigcirc(\text{halt } (init\ w)); f)))$

by (rule *IfChopEqvRule*)

have 3: $\vdash \text{empty}; f = f$

by (rule *EmptyChop*)

have 4: $\vdash (\bigcirc(\text{halt } (init\ w))); f = \bigcirc(\text{halt } (init\ w); f)$

by (rule *NextChop*)

from 2 3 4 **show** ?thesis **by** (metis *inteq-reflection*)

qed

lemma *AndHaltChopImp*:

$\vdash \text{init } w \wedge (\text{halt } (init\ w); f) \longrightarrow f$

proof –

have 1: $\vdash \text{halt } (init\ w); f = \text{if}_i\ (init\ w)\ \text{then } f\ \text{else } (\bigcirc(\text{halt } (init\ w); f))$

by (rule *HaltChopEqv*)

have 2: $\vdash \text{init } w \wedge \text{if}_i\ (init\ w)\ \text{then } f\ \text{else } (\bigcirc(\text{halt } (init\ w); f)) \longrightarrow f$

by (auto simp: *ifthenelse-d-def*)

from 1 2 **show** ?thesis **by** *fastforce*

qed

lemma *NotAndHaltChopImpNext*:

$\vdash \neg (\text{init } w) \wedge (\text{halt } (init\ w); f) \longrightarrow \bigcirc(\text{halt } (init\ w); f)$

proof –

have 1: $\vdash \text{halt } (init\ w); f = \text{if}_i\ (init\ w)\ \text{then } f\ \text{else } (\bigcirc(\text{halt } (init\ w); f))$

by (rule *HaltChopEqv*)

have 2: $\vdash \neg (\text{init } w) \wedge \text{if}_i\ (init\ w)\ \text{then } f\ \text{else } (\bigcirc(\text{halt } (init\ w); f)) \longrightarrow$

$\bigcirc(\text{halt } (init\ w); f)$

by (auto simp: *ifthenelse-d-def*)

from 1 2 **show** ?thesis **by** *fastforce*

qed

lemma *NotAndHaltChopImpSkipYields*:

$\vdash \neg (\text{init } w) \wedge (\text{halt } (init\ w); f) \longrightarrow \text{skip yields } (\text{halt } (init\ w); f)$

proof –

have 1: $\vdash \neg (\text{init } w) \wedge (\text{halt } (init\ w); f) \longrightarrow \bigcirc(\text{halt } (init\ w); f)$

by (rule *NotAndHaltChopImpNext*)

have 2: $\vdash \bigcirc(\text{halt } (init\ w); f) \longrightarrow \text{skip yields } (\text{halt } (init\ w); f)$

by (rule *NextImpSkipYields*)

from 1 2 **show** ?thesis **by** *fastforce*

qed

lemma *FiniteChopAndEmptyEqvChopAndEmpty*:

$\vdash ((finite;(f \wedge empty)) \wedge g) = ((g \wedge finite);(f \wedge empty))$
using AndFinEqvChopAndEmpty FinEqvTrueChopAndEmpty
by (smt ChopAndB ChopEmpty Prop10 Prop11 Prop12 inteq-reflection lift-and-com)

lemma WprevEqvEmptyOrPrev:
 $\vdash wprev\ f = (empty \vee prev\ f)$
by (auto simp: wprev-defs empty-defs prev-defs sum.case-eq-if)

lemma NotChopSkipEqvMoreAndNotChopSkip:
 $\vdash (\neg f);skip = (more \wedge \neg(f;skip))$
proof –
have 1: $\vdash wprev\ f = (empty \vee prev\ f)$ **using** WprevEqvEmptyOrPrev **by** auto
hence 2: $\vdash (\neg(wprev\ f)) = (\neg(empty \vee prev\ f))$ **by** auto
have 3: $\vdash \neg(wprev\ f) = ((\neg f);skip)$ **by** (simp add: wprev-d-def prev-d-def)
have 31: $\vdash (empty \vee prev\ f) = (empty \vee (f;skip))$ **by** (simp add: prev-d-def)
have 32: $\vdash (empty \vee (f;skip)) = (\neg more \vee \neg\neg(f;skip))$ **by** (simp add: empty-d-def)
have 33: $\vdash (\neg more \vee \neg\neg(f;skip)) = (\neg(more \wedge \neg(f;skip)))$ **by** fastforce
have 34: $\vdash (empty \vee prev\ f) = (\neg(more \wedge \neg(f;skip)))$ **using** 31 32 33 **by** (metis int-eq)
have 4: $\vdash \neg(empty \vee prev\ f) = (more \wedge \neg(f;skip))$ **using** 34 **by** fastforce
from 2 3 4 **show** ?thesis **by** fastforce
qed

lemma HaltChopImpNotHaltChopNot:
 $\vdash\ halt\ (init\ w); f \wedge finite \longrightarrow \neg(halt\ (init\ w); (\neg f))$
proof –
have 1: $\vdash halt\ (init\ w); f = if_i\ (init\ w)\ then\ f\ else\ (\bigcirc(halt\ (init\ w); f))$
by (rule HaltChopEqv)
have 2: $\vdash if_i\ (init\ w)\ then\ f\ else\ (\bigcirc(halt\ (init\ w); f)) \longrightarrow$
 $(((init\ w) \longrightarrow f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(halt\ (init\ w); f))))$
by (rule IfThenElseImp)
have 3: $\vdash halt\ (init\ w); (\neg f) =$
 $if_i\ (init\ w)\ then\ (\neg f)\ else\ (\bigcirc(halt\ (init\ w); (\neg f)))$
by (rule HaltChopEqv)
have 4: $\vdash if_i\ (init\ w)\ then\ (\neg f)\ else\ (\bigcirc(halt\ (init\ w); (\neg f))) \longrightarrow$
 $((init\ w) \longrightarrow \neg f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(halt\ (init\ w); (\neg f))))$
by (rule IfThenElseImp)
have 5: $\vdash halt\ (init\ w); f \wedge halt\ (init\ w); (\neg f) \longrightarrow$
 $((init\ w) \longrightarrow f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(halt\ (init\ w); f))) \wedge$
 $((init\ w) \longrightarrow \neg f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(halt\ (init\ w); (\neg f))))$
using 1 2 3 4 **by** fastforce
have 6: $\vdash ((init\ w) \longrightarrow f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(halt\ (init\ w); f))) \wedge$
 $((init\ w) \longrightarrow \neg f) \wedge (\neg(init\ w) \longrightarrow (\bigcirc(halt\ (init\ w); (\neg f)))) \longrightarrow$
 $(\bigcirc(halt\ (init\ w); f)) \wedge (\bigcirc(halt\ (init\ w); (\neg f)))$
by auto
have 7: $\vdash halt\ (init\ w); f \wedge halt\ (init\ w); (\neg f) \longrightarrow$
 $(\bigcirc(halt\ (init\ w); f)) \wedge (\bigcirc(halt\ (init\ w); (\neg f)))$
using 5 6 lift-imp-trans **by** blast
have 8: $\vdash ((\bigcirc(halt\ (init\ w); f)) \wedge (\bigcirc(halt\ (init\ w); (\neg f)))) =$
 $\bigcirc(halt\ (init\ w); f \wedge halt\ (init\ w); (\neg f))$

using NextAndEqvNextAndNext by fastforce
 have 9: $\vdash \text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); (\neg f) \longrightarrow$
 $\bigcirc (\text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); (\neg f))$
 using 7 8 by fastforce
 hence 10: $\vdash \text{finite} \longrightarrow \neg(\text{halt } (\text{init } w); f \wedge \text{halt } (\text{init } w); (\neg f))$
 using NextLoop by blast
 from 10 show ?thesis by auto
 qed

lemma HaltChopImpHaltYields:

$\vdash \text{halt } (\text{init } w); f \wedge \text{finite} \longrightarrow (\text{halt } (\text{init } w)) \text{ yields } f$
proof –
 have 1: $\vdash \text{halt } (\text{init } w); f \wedge \text{finite} \longrightarrow \neg (\text{halt } (\text{init } w); (\neg f))$
 by (rule HaltChopImpNotHaltChopNot)
 from 1 show ?thesis by (simp add: yields-d-def)
 qed

lemma HaltChopAnd:

$\vdash (\text{halt } (\text{init } w)); f \wedge (\text{halt } (\text{init } w)); g \wedge \text{finite} \longrightarrow (\text{halt } (\text{init } w)); (f \wedge g)$
proof –
 have 1: $\vdash (\text{halt } (\text{init } w)); g \wedge \text{finite} \longrightarrow (\text{halt } (\text{init } w)) \text{ yields } g$
 by (rule HaltChopImpHaltYields)
 hence 2: $\vdash (\text{halt } (\text{init } w)); f \wedge (\text{halt } (\text{init } w)); g \wedge \text{finite} \longrightarrow$
 $(\text{halt } (\text{init } w)); f \wedge (\text{halt } (\text{init } w)) \text{ yields } g$ by auto
 have 3: $\vdash (\text{halt } (\text{init } w)); f \wedge (\text{halt } (\text{init } w)) \text{ yields } g \longrightarrow$
 $(\text{halt } (\text{init } w)); (f \wedge g)$ by (rule ChopAndYieldsImp)
 from 2 3 show ?thesis by fastforce
 qed

lemma HaltAndChopAndHaltChopImpHaltAndChopAnd:

$\vdash (\text{halt } (\text{init } w) \wedge f); f1 \wedge (\text{halt } (\text{init } w); g) \wedge \text{finite} \longrightarrow (\text{halt } (\text{init } w) \wedge f); (f1 \wedge g)$
proof –
 have 1: $\vdash f1 \longrightarrow \neg g \vee (f1 \wedge g)$
 by auto
 hence 2: $\vdash (\text{halt } (\text{init } w) \wedge f); f1 \longrightarrow$
 $(\text{halt } (\text{init } w) \wedge f); (\neg g) \vee ((\text{halt } (\text{init } w) \wedge f); (f1 \wedge g))$
 by (rule ChopOrImpRule)
 have 3: $\vdash (\text{halt } (\text{init } w) \wedge f); (\neg g) \longrightarrow \text{halt } (\text{init } w); (\neg g)$
 by (rule AndChopA)
 have 31: $\vdash (\text{halt } (\text{init } w) \wedge f); f1 \longrightarrow$
 $\text{halt } (\text{init } w); (\neg g) \vee ((\text{halt } (\text{init } w) \wedge f); (f1 \wedge g))$
 using 23 by fastforce
 have 4: $\vdash \text{halt } (\text{init } w); g \wedge \text{finite} \longrightarrow \neg (\text{halt } (\text{init } w); (\neg g))$
 by (rule HaltChopImpNotHaltChopNot)
 hence 41: $\vdash (\text{halt } (\text{init } w); (\neg g)) \wedge \text{finite} \longrightarrow \neg(\text{halt } (\text{init } w); g)$
 by auto
 have 42: $\vdash (\text{halt } (\text{init } w) \wedge f); f1 \wedge \text{finite} \longrightarrow$
 $\neg(\text{halt } (\text{init } w); g) \vee ((\text{halt } (\text{init } w) \wedge f); (f1 \wedge g))$
 using 31 41 by fastforce
 from 42 show ?thesis by auto

qed

lemma *HaltImpBoxYields*:

$\vdash (\text{halt } (\text{init } w)); f \wedge \text{finite} \longrightarrow (\Box(\neg (\text{init } w))) \text{ yields } ((\text{halt } (\text{init } w)); f)$

proof –

have 1: $\vdash (\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)) \longrightarrow \text{di } (\Box(\neg (\text{init } w)))$
by (rule *ChopImpDi*)

have 2: $\vdash \Box(\neg (\text{init } w)) \longrightarrow \neg (\text{init } w)$
by (rule *BoxElim*)

hence 3: $\vdash \text{di } (\Box(\neg (\text{init } w))) \longrightarrow \text{di } (\neg (\text{init } w))$
by (rule *DImpDi*)

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: $\vdash (\text{halt } (\text{init } w)); f \wedge (\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)) \longrightarrow$
 $\bigcirc((\text{halt } (\text{init } w)); f) \wedge \bigcirc((\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)))$
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: $\vdash (\text{halt } (\text{init } w)); f \wedge (\Box(\neg (\text{init } w))); (\neg (\text{halt } (\text{init } w); f)) \longrightarrow$

```

      ○((( halt (init w)); f) ∧ ((□(¬ (init w))); (¬ ( halt (init w); f))))
    using 11 12 by fastforce
  hence 14: ⊢ finite → ¬ (( halt (init w)); f ∧ (□ (¬ (init w))); (¬ ( halt (init w); f)))
    using NextLoop by blast
  hence 15: ⊢ ( halt (init w)); f ∧ finite → ¬ ((□ (¬ (init w))); (¬ ( halt (init w); f)))
    by auto
  from 15 show ?thesis by (simp add: yields-d-def)
qed

```

9.12 Properties of Groups of chops

```

lemma NestedChopImpChop:
  assumes ⊢ init w ∧ f → g; (init w1 ∧ f1)
    ⊢ init w1 ∧ f1 → g1; (init w2 ∧ f2)
  shows ⊢ init w ∧ f → g; (g1; (init w2 ∧ f2))
proof -
  have 1: ⊢ init w ∧ f → g; (init w1 ∧ f1) using assms(1) by auto
  have 2: ⊢ init w1 ∧ f1 → g1; (init w2 ∧ f2) using assms(2) by auto
  hence 3: ⊢ g; (init w1 ∧ f1) → g; (g1; (init w2 ∧ f2)) by (rule RightChopImpChop)
  from 1 3 show ?thesis by fastforce
qed

```

end

10 Infinite ITL theorems using strong chop

```

theory InfiniteSChopTheorems
  imports
    InfiniteTheorems
begin

```

We give the proofs of a list of Infinite ITL theorems but now using the strong chop.

10.1 Strong Chop axioms

```

lemma SChopAssoc:
  ⊢ f ∧ (g ∧ h) = (f ∧ g) ∧ h
proof -
  have 1: ⊢ f ∧ (g ∧ h) = (f ∧ finite); ((g ∧ finite); h)
    by (simp add: schop-d-def)
  have 2: ⊢ (f ∧ finite); ((g ∧ finite); h) = ((f ∧ finite); (g ∧ finite)); h
    using ChopAssoc by blast
  have 3: ⊢ ((f ∧ finite); (g ∧ finite)); h = (f ∧ (g ∧ finite)); h
    by (simp add: schop-d-def)
  have 4: ⊢ f ∧ (g ∧ finite) = (f ∧ g ∧ finite)
    by (smt ChopAndFiniteDist Prop10 Prop12 int-iffD2 inteq-reflection
      lift-and-com schop-d-def)
  have 5: ⊢ (f ∧ (g ∧ finite)); h = (f ∧ g ∧ finite); h

```

using 4 **by** (*simp add: LeftChopEqvChop*)
have 6: $\vdash (f \frown g \wedge \text{finite}); h = (f \frown g) \frown h$
by (*simp add: schop-d-def*)
from 1 2 3 5 6 **show** ?thesis **by** fastforce
qed

lemma OrSCHopImp :
 $\vdash (f \vee g) \frown h \longrightarrow f \frown h \vee g \frown h$
by (*simp add: schop-defs Valid-def sum.case-eq-if, auto*)

lemma SCHopOrImp :
 $\vdash f \frown (g \vee h) \longrightarrow f \frown g \vee f \frown h$
by (*simp add: schop-defs Valid-def sum.case-eq-if, auto*)

lemma EmptySCHop :
 $\vdash \text{empty} \frown f = f$
by (*metis EmptyChopSem FiniteAndEmptyEqvEmpty intl inteq-reflection lift-and-com schop-d-def*)

lemma SCHopEmpty :
 $\vdash \text{finite} \longrightarrow f \frown \text{empty} = f$
by (*simp add: schop-defs finite-defs empty-defs Valid-def sum.case-eq-if, auto*)

lemma StatImpBf :
 $\vdash \text{init } f \longrightarrow \text{bf } (\text{init } f)$
by (*simp add: Valid-def bf-defs init-defs sum.case-eq-if,*
metis conc-def conc-iprefix-isuffix interval-intlen-gr-zero interval-nth-zero-intfirst
iprefix-0)

lemma BfBoxSCHopImpSCHop :
 $\vdash \text{bf } (f \longrightarrow f1) \wedge \Box(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f1 \frown g1$
by (*simp add: Valid-def schop-defs bf-defs always-defs sum.case-eq-if, auto*)

lemma SCHopstarEqv :
 $\vdash (\text{schopstar } f) = (\text{empty} \vee (f \wedge \text{more}) \frown (\text{schopstar } f))$
using SCHopstarEqvSem Valid-def **by** blast

lemma AndMoreSCHopEqvAndFmoreChop:
 $\vdash (f \wedge \text{more}) \frown g = (f \wedge \text{fmore}); g$
by (*simp add: LeftChopEqvChop AndMoreAndFiniteEqvAndFmore schop-d-def*)

lemma SOmegaUnroll:
 $\vdash f^\omega = (f \wedge \text{more}) \frown f^\omega$
using OmegaUnroll AndMoreSCHopEqvAndFmoreChop **by** fastforce

lemma SOmegaInduct:
 $\vdash (\text{inf } \wedge g \wedge \Box(g \longrightarrow (f \wedge \text{more}) \frown g)) \longrightarrow \text{omega } f$
using OmegaInductSem AndMoreSCHopEqvAndFmoreChop Valid-def
by (*metis inteq-reflection*)

lemma *FiniteBfGen*:
assumes $\vdash \text{finite} \longrightarrow f$
shows $\vdash \text{bf } f$
using *assms*
by (*simp add: Valid-def bf-defs finite-defs sum.case-eq-if*)

lemma *BfGen*:
assumes $\vdash f$
shows $\vdash \text{bf } f$
using *assms*
by (*smt bf-defs intD intl sum.case-eq-if*)

10.2 ITL operators in terms of SChop

lemma *NextSChopdef*:
 $\vdash \bigcirc f = \text{skip} \frown f$
by (*metis FiniteChopSkipEqvSkipChopFinite NowImpDiamond Prop10 SkipChopFiniteImpFinite*
inteq-reflection lift-imp-trans next-d-def schop-d-def sometimes-d-def)

lemma *DiamondSChopdef*:
 $\vdash \Diamond f = \# \text{True} \frown f$
by (*simp add: schop-d-def sometimes-d-def*)

lemma *FiniteSChopdef*:
 $\vdash \text{finite} = \Diamond \text{empty}$
by (*simp add: DiamondEmptyEqvFinite int-iffD1 int-iffD2 int-iffI*)

lemma *ChopSChopdef*:
 $\vdash f;g = ((f \frown g) \vee (f \wedge \text{inf}))$
by (*metis AndInfChopEqvAndInf OrChopEqv OrFiniteInf inteq-reflection schop-d-def*)

lemma *PowerPowerdef*:
 $\vdash \text{power } f \ n = \text{spower } f \ n$
proof
(induct n)
case 0
then show ?case **by** *auto*
next
case (*Suc n*)
then show ?case
by (*metis PowerCommute inteq-reflection pow-Suc schop-d-def spow-Suc*)
qed

lemma *SChopstarFPowerstardef*:
 $\vdash \text{schopstar } f = \text{fpowerstar } f$
proof —
have 1: $\vdash \text{schopstar } f = (\exists k. \text{spower } (f \wedge \text{more}) \ k)$
by (*simp add: schopstar-d-def spowerstar-d-def*)
have 2: $\vdash \text{fpowerstar } f = \text{fpowerstar } (f \wedge \text{more})$

using *FPSAndMoreEqvFPS* **by** *fastforce*
have 3: $\vdash \text{fpowerstar } (f \wedge \text{more}) = (\exists k. \text{power } (f \wedge \text{more}) k)$
by (*simp add: fpowerstar-d-def*)
have 4: $\bigwedge n. \vdash \text{spower } (f \wedge \text{more}) n = \text{power } (f \wedge \text{more}) n$
using *PowerSpowerdef* **by** *fastforce*
from 1 2 3 4 **show** *?thesis* **by** *fastforce*
qed

lemma *SFinprop* :

$\vdash ((\# \text{True} \wedge (f \wedge \text{empty})) \wedge (\# \text{True} \wedge (g \wedge \text{empty}))) = (\# \text{True} \wedge ((f \wedge g) \wedge \text{empty}))$
 $\vdash ((\# \text{True} \wedge (f \wedge \text{empty})) \vee (\# \text{True} \wedge (g \wedge \text{empty}))) = (\# \text{True} \wedge ((f \vee g) \wedge \text{empty}))$
 $\vdash \text{finite} \longrightarrow (\neg (\# \text{True} \wedge (f \wedge \text{empty}))) = (\# \text{True} \wedge (\neg f \wedge \text{empty}))$
 $\vdash (\neg (\# \text{True} \wedge (f \wedge \text{empty}))) = ((\# \text{True} \wedge (\neg f \wedge \text{empty})) \vee \text{inf})$
using *le-neq-implies-less*
by (*auto simp add: Valid-def finite-defs infinite-defs schop-defs empty-defs sum.case-eq-if*,
fastforce, fastforce)

10.3 Basic Theorems

lemma *BfSChopImpSChop* :

$\vdash \text{bf } (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$
proof –
have 1: $\vdash g \longrightarrow g$ **by** *auto*
hence 2: $\vdash \Box (g \longrightarrow g)$ **by** (*rule BoxGen*)
have 3: $\vdash \text{bf } (f \longrightarrow f1) \wedge \Box (g \longrightarrow g) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (*rule BfBoxSChopImpSChop*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *BiImpBf*:

$\vdash \text{bi } f \longrightarrow \text{bf } f$
by (*simp add: bi-defs bf-defs Valid-def sum.case-eq-if*)

lemma *BiSChopImpSChop* :

$\vdash \text{bi } (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$
proof –
have 1: $\vdash g \longrightarrow g$ **by** *auto*
hence 2: $\vdash \Box (g \longrightarrow g)$ **by** (*rule BoxGen*)
have 3: $\vdash \text{bi } (f \longrightarrow f1) \wedge \Box (g \longrightarrow g) \longrightarrow f \frown g \longrightarrow f1 \frown g$
using *BiImpBf BfBoxSChopImpSChop* **using** *BfSChopImpSChop* **by** *fastforce*
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *AndSChopA*:

$\vdash (f \wedge f1) \frown g \longrightarrow f \frown g$
proof –
have 1: $\vdash f \wedge f1 \longrightarrow f$ **by** *auto*
hence 2: $\vdash \text{bf } (f \wedge f1 \longrightarrow f)$ **by** (*rule BfGen*)
have 3: $\vdash \text{bf } (f \wedge f1 \longrightarrow f) \longrightarrow (f \wedge f1) \frown g \longrightarrow f \frown g$ **by** (*rule BfSChopImpSChop*)
from 2 3 **show** *?thesis* **using** *MP* **by** *blast*

qed

lemma *AndSChopB*:

$\vdash (f \wedge f1) \frown g \longrightarrow f1 \frown g$

proof –

have 1: $\vdash f \wedge f1 \longrightarrow f1$ **by** *auto*

hence 2: $\vdash bf (f \wedge f1 \longrightarrow f1)$ **by** (*rule BfGen*)

have 3: $\vdash bf (f \wedge f1 \longrightarrow f1) \longrightarrow (f \wedge f1) \frown g \longrightarrow f1 \frown g$ **by** (*rule BfSChopImpSChop*)

from 2 3 **show** *?thesis* **using** *MP* **by** *blast*

qed

lemma *NextSChop*:

$\vdash (\bigcirc f) \frown g = \bigcirc(f \frown g)$

proof –

have 1: $\vdash skip \frown (f \frown g) = (skip \frown f) \frown g$ **by** (*rule SChopAssoc*)

from 1 **show** *?thesis* **using** *NextSChopdef* **by** (*metis inteq-reflection*)

qed

lemma *BoxSChopImpSChop* :

$\vdash \Box (g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$

proof –

have 1: $\vdash g \longrightarrow g$ **by** *auto*

hence 2: $\vdash bf (g \longrightarrow g)$ **by** (*rule BfGen*)

have 3: $\vdash bf (f \longrightarrow f) \wedge \Box (g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **by** (*rule BfBoxSChopImpSChop*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *LeftSChopImpSChop*:

assumes $\vdash f \longrightarrow f1$

shows $\vdash f \frown g \longrightarrow f1 \frown g$

proof –

have 1: $\vdash f \longrightarrow f1$ **using** *assms* **by** *auto*

hence 2: $\vdash bf (f \longrightarrow f1)$ **by** (*rule BfGen*)

have 3: $\vdash bf (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (*rule BfSChopImpSChop*)

from 2 3 **show** *?thesis* **using** *MP* **by** *blast*

qed

lemma *RightSChopImpSChop*:

assumes $\vdash g \longrightarrow g1$

shows $\vdash f \frown g \longrightarrow f \frown g1$

proof –

have 1: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*

hence 2: $\vdash \Box (g \longrightarrow g1)$ **by** (*rule BoxGen*)

have 3: $\vdash \Box (g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **by** (*rule BoxSChopImpSChop*)

from 2 3 **show** *?thesis* **using** *MP* **by** *blast*

qed

lemma *RightSChopEqvSChop*:

assumes $\vdash g = g1$

shows $\vdash (f \frown g) = (f \frown g1)$

proof –

have 1: $\vdash g = g1$ **using** *assms* **by** *auto*

have 2: $(\vdash g \longrightarrow g1) \implies (\vdash f \frown g \longrightarrow f \frown g1)$ **by** (*rule RightSChopImpSChop*)

have 3: $(\vdash g1 \longrightarrow g) \implies (\vdash f \frown g1 \longrightarrow f \frown g)$ **by** (*rule RightSChopImpSChop*)

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

qed

lemma *BoxRightSChopEqvSChop*:

$\vdash \Box (g = g1) \longrightarrow (f \frown g) = (f \frown g1)$

proof –

have 1: $\vdash \Box (g = g1) = (\Box (g \longrightarrow g1) \wedge \Box (g1 \longrightarrow g))$

by (*simp add: Valid-def always-defs sum.case-eq-if, auto*)

have 2: $\vdash \Box (g \longrightarrow g1) \longrightarrow (f \frown g) \longrightarrow (f \frown g1)$ **by** (*simp add: BoxSChopImpSChop*)

have 3: $\vdash \Box (g1 \longrightarrow g) \longrightarrow (f \frown g1) \longrightarrow (f \frown g)$ **by** (*simp add: BoxSChopImpSChop*)

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

qed

lemma *FiniteRightSChopEqvSChop*:

assumes $\vdash \text{finite} \longrightarrow g = g1$

shows $\vdash \text{finite} \longrightarrow (f \frown g) = (f \frown g1)$

using *assms*

by (*simp add: Valid-def finite-defs schop-defs sum.case-eq-if*)

lemma *SChopOrEqv*:

$\vdash f \frown (g \vee g1) = (f \frown g \vee f \frown g1)$

proof –

have 1: $\vdash g \longrightarrow g \vee g1$ **by** *auto*

hence 2: $\vdash f \frown g \longrightarrow f \frown (g \vee g1)$ **by** (*rule RightSChopImpSChop*)

have 3: $\vdash g1 \longrightarrow g \vee g1$ **by** *auto*

hence 4: $\vdash f \frown g1 \longrightarrow f \frown (g \vee g1)$ **by** (*rule RightSChopImpSChop*)

from 2 4 **show** *?thesis* **by** (*meson SChopOrImp Prop02 Prop11*)

qed

lemma *OrSChopEqv*:

$\vdash (f \vee f1) \frown g = (f \frown g \vee f1 \frown g)$

proof –

have 1: $\vdash f \longrightarrow f \vee f1$ **by** *auto*

hence 2: $\vdash f \frown g \longrightarrow (f \vee f1) \frown g$ **by** (*rule LeftSChopImpSChop*)

have 3: $\vdash f1 \longrightarrow f \vee f1$ **by** *auto*

hence 4: $\vdash f1 \frown g \longrightarrow (f \vee f1) \frown g$ **by** (*rule LeftSChopImpSChop*)

from 2 4 **show** *?thesis*

by (*meson OrSChopImp int-iff1 Prop02*)

qed

lemma *OrSChopImpRule:*

assumes $\vdash f \longrightarrow f1 \vee f2$

shows $\vdash f \wedge g \longrightarrow (f1 \wedge g) \vee (f2 \wedge g)$

proof –

have 1: $\vdash f \longrightarrow f1 \vee f2$ **using** *assms* **by** *auto*

hence 2: $\vdash f \wedge g \longrightarrow (f1 \vee f2) \wedge g$ **by** (*rule LeftSChopImpSChop*)

have 3: $\vdash (f1 \vee f2) \wedge g = (f1 \wedge g \vee f2 \wedge g)$ **by** (*rule OrSChopEqv*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *LeftSChopEqvSChop:*

assumes $\vdash f = f1$

shows $\vdash f \wedge g = (f1 \wedge g)$

proof –

have 1: $\vdash f = f1$ **using** *assms* **by** *auto*

hence 2: $\vdash f \longrightarrow f1$ **by** *auto*

hence 3: $\vdash f \wedge g \longrightarrow f1 \wedge g$ **by** (*rule LeftSChopImpSChop*)

have $\vdash f1 \longrightarrow f$ **using** 1 **by** *auto*

hence 4: $\vdash f1 \wedge g \longrightarrow f \wedge g$ **by** (*rule LeftSChopImpSChop*)

from 3 4 **show** *?thesis* **by** (*simp add: int-iff1*)

qed

lemma *OrSChopEqvRule:*

assumes $\vdash f = (f1 \vee f2)$

shows $\vdash f \wedge g = ((f1 \wedge g) \vee (f2 \wedge g))$

proof –

have 1: $\vdash f = (f1 \vee f2)$ **using** *assms* **by** *auto*

hence 2: $\vdash f \wedge g = ((f1 \vee f2) \wedge g)$ **by** (*rule LeftSChopEqvSChop*)

have 3: $\vdash (f1 \vee f2) \wedge g = (f1 \wedge g \vee f2 \wedge g)$ **by** (*rule OrSChopEqv*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *SChopOrImpRule:*

assumes $\vdash g \longrightarrow g1 \vee g2$

shows $\vdash f \wedge g \longrightarrow (f \wedge g1) \vee (f \wedge g2)$

proof –

have 1: $\vdash g \longrightarrow g1 \vee g2$ **using** *assms* **by** *auto*

hence 2: $\vdash f \wedge g \longrightarrow f \wedge (g1 \vee g2)$ **by** (*rule RightSChopImpSChop*)

have 3: $\vdash f \wedge (g1 \vee g2) = (f \wedge g1 \vee f \wedge g2)$ **by** (*rule SChopOrEqv*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *SChopImpDiamond:*

$\vdash f \wedge g \longrightarrow \Diamond g$

proof –

have 1: $\vdash f \longrightarrow \#True$ **by** *auto*

hence 2: $\vdash f \frown g \longrightarrow \#True \frown g$ **by** (rule LeftSChopImpSChop)
 from 2 show ?thesis **using** DiamondSChopdef **by** fastforce
 qed

lemma BfImpDfImpDf:

$\vdash bf (f \longrightarrow g) \longrightarrow df f \longrightarrow df g$

proof –

have 1: $\vdash bf (f \longrightarrow g) \longrightarrow (f \frown \#True) \longrightarrow (g \frown \#True)$ **by** (rule BfSChopImpSChop)

from 1 show ?thesis **by** (simp add: df-d-def)

qed

lemma DfImpDf:

assumes $\vdash f \longrightarrow g$

shows $\vdash df f \longrightarrow df g$

proof –

have 1: $\vdash f \longrightarrow g$ **using** assms **by** auto

hence 2: $\vdash f \frown \#True \longrightarrow g \frown \#True$ **by** (rule LeftSChopImpSChop)

from 2 show ?thesis **by** (simp add: df-d-def)

qed

lemma BfImpBfRule:

assumes $\vdash f \longrightarrow g$

shows $\vdash bf f \longrightarrow bf g$

proof –

have 1: $\vdash f \longrightarrow g$ **using** assms **by** auto

hence 2: $\vdash \neg g \longrightarrow \neg f$ **by** auto

hence 3: $\vdash df (\neg g) \longrightarrow df (\neg f)$ **by** (rule DfImpDf)

hence 4: $\vdash \neg (df (\neg f)) \longrightarrow \neg (df (\neg g))$ **by** auto

from 4 show ?thesis **by** (simp add: bf-d-def)

qed

lemma DfEqvDf:

assumes $\vdash f = g$

shows $\vdash df f = df g$

proof –

have 1: $\vdash f = g$ **using** assms **by** auto

hence 2: $\vdash f \frown \#True = g \frown \#True$ **by** (rule LeftSChopEqvSChop)

from 2 show ?thesis **by** (simp add: df-d-def)

qed

lemma BfEqvBf:

assumes $\vdash f = g$
shows $\vdash bf\ f = bf\ g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash (\neg f) = (\neg g)$ **by** *auto*
hence 3: $\vdash df\ (\neg f) = df\ (\neg g)$ **by** (*rule DfEqvDf*)
hence 4: $\vdash (\neg (df\ (\neg f))) = (\neg (df\ (\neg g)))$ **by** *auto*
from 4 **show** *?thesis* **by** (*simp add: bf-d-def*)
qed

lemma *LeftSChopSChopImpSChopRule*:
assumes $\vdash (f \frown g) \longrightarrow g$
shows $\vdash (f \frown g) \frown h \longrightarrow (g \frown h)$
proof –
have 1: $\vdash (f \frown g) \longrightarrow g$ **using** *assms* **by** *blast*
hence 2: $\vdash (f \frown g) \frown h \longrightarrow g \frown h$ **by** (*rule LeftSChopImpSChop*)
have 3: $\vdash f \frown (g \frown h) = (f \frown g) \frown h$ **by** (*rule SChopAssoc*)
from 2 3 **show** *?thesis* **by** *auto*
qed

lemma *AndSChopCommute* :
 $\vdash (f \wedge f1) \frown g = (f1 \wedge f) \frown g$
proof –
have 1: $\vdash (f \wedge f1) = (f1 \wedge f)$ **by** *auto*
from 1 **show** *?thesis* **by** (*rule LeftSChopEqvSChop*)
qed

lemma *BfAndSChopImport*:
 $\vdash bf\ f \wedge (f1 \frown g) \longrightarrow (f \wedge f1) \frown g$
proof –
have 1: $\vdash f \longrightarrow (f1 \longrightarrow f \wedge f1)$ **by** *auto*
hence 2: $\vdash bf\ f \longrightarrow bf\ (f1 \longrightarrow f \wedge f1)$ **by** (*rule BfImpBfRule*)
have 3: $\vdash bf\ (f1 \longrightarrow (f \wedge f1)) \longrightarrow f1 \frown g \longrightarrow (f \wedge f1) \frown g$ **by** (*rule BfSChopImpSChop*)
from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *BiAndSChopImport*:
 $\vdash bi\ f \wedge (f1 \frown g) \longrightarrow (f \wedge f1) \frown g$
proof –
have 1: $\vdash f \longrightarrow (f1 \longrightarrow f \wedge f1)$ **by** *auto*
hence 2: $\vdash bi\ f \longrightarrow bi\ (f1 \longrightarrow f \wedge f1)$ **by** (*rule BiImpBiRule*)
have 3: $\vdash bi\ (f1 \longrightarrow (f \wedge f1)) \longrightarrow f1 \frown g \longrightarrow (f \wedge f1) \frown g$ **by** (*rule BiSChopImpSChop*)
from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *StateAndSChopImport*:
 $\vdash (init\ w) \wedge (f \frown g) \longrightarrow ((init\ w) \wedge f) \frown g$
proof –

have 1: $\vdash (\text{init } w) \longrightarrow \text{bf } (\text{init } w)$ **by** (rule StatImpBf)
hence 2: $\vdash (\text{init } w) \wedge (f \frown g) \longrightarrow \text{bf } (\text{init } w) \wedge (f \frown g)$ **by** auto
have 3: $\vdash \text{bf } (\text{init } w) \wedge (f \frown g) \longrightarrow ((\text{init } w) \wedge f) \frown g$ **by** (rule BfAndSChopImport)
from 2 3 **show** ?thesis **using** MP **by** fastforce
qed

10.4 Further Properties Df and Bf

lemma AndFinitImpDf:

$\vdash f \wedge \text{finite} \longrightarrow \text{df } f$

proof –

have 1: $\vdash \text{finite} \longrightarrow f \frown \text{empty} = f$ **by** (rule SChopEmpty)
have 2: $\vdash \text{empty} \longrightarrow \# \text{True}$ **by** auto
hence 3: $\vdash f \frown \text{empty} \longrightarrow f \frown \# \text{True}$ **by** (rule RightSChopImpSChop)
have 4: $\vdash f \wedge \text{finite} \longrightarrow f \frown \# \text{True}$ **using** 1 3 **by** fastforce
from 4 **show** ?thesis **by** (simp add: df-d-def)
qed

lemma DfState:

$\vdash \text{df } (\text{init } w) = (\text{init } w)$

proof –

have 0: $\vdash (\text{init } (\neg w)) \longrightarrow \text{bf } (\text{init } (\neg w))$ **using** StatImpBf **by** fastforce
hence 1: $\vdash \neg(\text{init } w) \longrightarrow \text{bf } (\neg(\text{init } w))$ **using** Initprop(2) **by** (metis inteq-reflection)
hence 2: $\vdash (\neg(\text{init } w)) \longrightarrow \neg(\text{df } (\neg\neg(\text{init } w)))$ **by** (simp add: bf-d-def)
have 3: $\vdash (\neg(\text{init } w) \longrightarrow \neg(\text{df } (\neg\neg(\text{init } w)))) \longrightarrow (\text{df } (\neg\neg(\text{init } w)) \longrightarrow (\text{init } w))$ **by** auto
have 4: $\vdash \text{df } (\neg\neg(\text{init } w)) \longrightarrow (\text{init } w)$ **using** 2 3 MP **by** blast
have 5: $\vdash (\text{init } w) \longrightarrow \neg\neg(\text{init } w)$ **by** auto
hence 6: $\vdash \text{df } (\text{init } w) \longrightarrow \text{df } (\neg\neg(\text{init } w))$ **by** (rule DfImpDf)
have 7: $\vdash \text{df } (\text{init } w) \longrightarrow (\text{init } w)$ **using** 6 4 **using** lift-imp-trans **by** metis
have 8: $\vdash (\text{init } w) \wedge \text{finite} \longrightarrow \text{df } (\text{init } w)$ **by** (rule AndFinitImpDf)
from 7 8 **show** ?thesis
by (metis NowImpDiamond Prop10 StateAndChop df-d-def int-simps(17) inteq-reflection
lift-and-com schop-d-def sometimes-d-def)
qed

lemma StateSChop:

$\vdash (\text{init } w) \frown f \longrightarrow (\text{init } w)$

by (simp add: StateChopExportA schop-d-def)

lemma StateSChopExportA:

$\vdash ((\text{init } w) \wedge f) \frown g \longrightarrow (\text{init } w)$

by (meson AndSChopA StateSChop lift-imp-trans)

lemma StateAndSChop:

$\vdash ((\text{init } w) \wedge f) \frown g = ((\text{init } w) \wedge (f \frown g))$

by (*simp add: AndSChopB StateAndSChopImport StateSChopExportA Prop11 Prop12*)

lemma *StateAndSChopImpSChopRule*:

assumes $\vdash (\text{init } w) \wedge f \longrightarrow f1$

shows $\vdash (\text{init } w) \wedge (f \frown g) \longrightarrow (f1 \frown g)$

proof –

have 1: $\vdash (\text{init } w) \wedge f \longrightarrow f1$ **using** *assms* **by** *auto*

hence 2: $\vdash ((\text{init } w) \wedge f) \frown g \longrightarrow f1 \frown g$ **by** (*rule LeftSChopImpSChop*)

have 3: $\vdash ((\text{init } w) \wedge f) \frown g = ((\text{init } w) \wedge (f \frown g))$ **by** (*rule StateAndSChop*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *StateImpSChopEqvSChop* :

assumes $\vdash (\text{init } w) \longrightarrow (f = f1)$

shows $\vdash (\text{init } w) \longrightarrow ((f \frown g) = (f1 \frown g))$

proof –

have 1: $\vdash (\text{init } w) \longrightarrow (f = f1)$ **using** *assms* **by** *auto*

hence 2: $\vdash (\text{init } w) \wedge f \longrightarrow f1$ **by** *auto*

hence 3: $\vdash (\text{init } w) \wedge (f \frown g) \longrightarrow (f1 \frown g)$ **by** (*rule StateAndSChopImpSChopRule*)

have 4: $\vdash (\text{init } w) \wedge f1 \longrightarrow f$ **using** 1 **by** *auto*

hence 5: $\vdash (\text{init } w) \wedge (f1 \frown g) \longrightarrow (f \frown g)$ **by** (*rule StateAndSChopImpSChopRule*)

from 3 5 **show** *?thesis* **by** *fastforce*

qed

lemma *ChopEqvStateAndSChop*:

assumes $\vdash f = (\text{init } w) \wedge f1$

shows $\vdash (f \frown g) = ((\text{init } w) \wedge (f1 \frown g))$

proof –

have 1: $\vdash f = ((\text{init } w) \wedge f1)$ **using** *assms* **by** *auto*

hence 2: $\vdash f \frown g = (((\text{init } w) \wedge f1) \frown g)$ **by** (*rule LeftSChopEqvSChop*)

have 3: $\vdash ((\text{init } w) \wedge f1) \frown g = ((\text{init } w) \wedge (f1 \frown g))$ **by** (*rule StateAndSChop*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *DfIntro*:

$\vdash f \wedge \text{finite} \longrightarrow \text{df } f$

proof –

have 1: $\vdash \text{finite} \longrightarrow f \frown \text{empty} = f$ **by** (*rule SChopEmpty*)

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: $\vdash f \frown \text{empty} \longrightarrow f \frown \# \text{True}$ **using** 3 4 *MP* **by** (*simp add: RightSChopImpSChop*)

hence 6: $\vdash f \frown \text{empty} \longrightarrow \text{df } f$ **by** (*simp add: df-d-def*)

from 1 6 **show** *?thesis* **using** *AndFinitImpDf* **by** *blast*

qed

lemma *BfElim*:

$\vdash \text{bf } f \wedge \text{finite} \longrightarrow f$

proof –

have 1: $\vdash \neg f \wedge \text{finite} \longrightarrow \text{df } (\neg f)$ **by** (*rule DfIntro*)

have 2: $\vdash (\neg f \wedge \text{finite} \longrightarrow \text{df } (\neg f)) \longrightarrow (\neg(\text{df } (\neg f)) \longrightarrow \neg(\neg f \wedge \text{finite}))$
by *simp*
have 21: $\vdash \neg(\neg f \wedge \text{finite}) = (f \vee \text{inf})$ **by** (*simp add: Valid-def finite-d-def*)
have 3: $\vdash \neg(\text{df } (\neg f)) \longrightarrow f \vee \text{inf}$ **using** 1 2 21 **by** *fastforce*
from 3 **show** ?thesis **by** (*simp add: Prop13 bf-d-def finite-d-def*)
qed

lemma *BfContraPosImpDist*:

$\vdash \text{bf } (\neg g \longrightarrow \neg f) \longrightarrow (\text{bf } f) \longrightarrow (\text{bf } g)$
proof –
have 1: $\vdash \text{bf } (\neg g \longrightarrow \neg f) \longrightarrow (\text{df } (\neg g)) \longrightarrow (\text{df } (\neg f))$ **by** (*rule BfImpDfImpDf*)
hence 2: $\vdash \text{bf } (\neg g \longrightarrow \neg f) \longrightarrow (\neg(\text{df } (\neg f))) \longrightarrow (\neg(\text{df } (\neg g)))$ **by** *auto*
from 2 **show** ?thesis **by** (*metis bf-d-def*)
qed

lemma *BfImpDist*:

$\vdash \text{bf } (f \longrightarrow g) \longrightarrow (\text{bf } f) \longrightarrow (\text{bf } g)$
proof –
have 1: $\vdash (f \longrightarrow g) \longrightarrow (\neg g \longrightarrow \neg f)$ **by** *auto*
hence 2: $\vdash \neg(\neg g \longrightarrow \neg f) \longrightarrow \neg(f \longrightarrow g)$ **by** *auto*
hence 3: $\vdash \text{bf } (\neg(\neg g \longrightarrow \neg f) \longrightarrow \neg(f \longrightarrow g))$ **by** (*rule BfGen*)
have 4: $\vdash \text{bf } (\neg(\neg g \longrightarrow \neg f) \longrightarrow \neg(f \longrightarrow g))$
 \longrightarrow
 $\text{bf } (f \longrightarrow g) \longrightarrow \text{bf } (\neg g \longrightarrow \neg f)$ **by** (*rule BfContraPosImpDist*)
have 5: $\vdash \text{bf } (f \longrightarrow g) \longrightarrow \text{bf } (\neg g \longrightarrow \neg f)$ **using** 3 4 *MP* **by** *blast*
have 6: $\vdash \text{bf } (\neg g \longrightarrow \neg f) \longrightarrow (\text{bf } f) \longrightarrow (\text{bf } g)$ **by** (*rule BfContraPosImpDist*)
from 5 6 **show** ?thesis **using** *lift-imp-trans* **by** *blast*
qed

lemma *FinitelImpBfImpBfRule*:

assumes $\vdash \text{finite} \longrightarrow (f \longrightarrow g)$
shows $\vdash \text{bf } f \longrightarrow \text{bf } g$
proof –
have 1: $\vdash \text{finite} \longrightarrow f \longrightarrow g$ **using** *assms* **by** *auto*
have 2: $\vdash \text{bf}(f \longrightarrow g)$ **using** 1 **by** (*simp add: FiniteBfGen*)
have 3: $\vdash \text{bf}(f \longrightarrow g) \longrightarrow \text{bf } f \longrightarrow \text{bf } g$ **using** *BfImpDist* **by** *blast*
from 2 3 **show** ?thesis **by** *fastforce*
qed

lemma *FinitelImpBfEqvRule*:

assumes $\vdash \text{finite} \longrightarrow (f = g)$
shows $\vdash \text{bf } f = \text{bf } g$
proof –
have 1: $\vdash \text{finite} \longrightarrow (f = g)$ **using** *assms* **by** *blast*
have 2: $\vdash \text{finite} \longrightarrow (f \longrightarrow g)$ **using** 1 **by** *auto*
have 3: $\vdash \text{bf } f \longrightarrow \text{bf } g$ **by** (*simp add: 2 FinitelImpBfImpBfRule*)

have 4: $\vdash \text{finite} \longrightarrow (g \longrightarrow f)$ **using** 1 **by** *auto*
have 5: $\vdash bf\ g \longrightarrow bf\ f$ **by** (*simp add: 4 FinitelmpBfImpBfRule*)
from 3 5 **show** ?thesis **by** *fastforce*
qed

lemma *IfSChopEqvRule*:

assumes $\vdash f = \text{if}_i\ (\text{init}\ w)\ \text{then}\ f1\ \text{else}\ f2$
shows $\vdash f \frown g = \text{if}_i\ (\text{init}\ w)\ \text{then}\ (f1 \frown g)\ \text{else}\ (f2 \frown g)$
proof –
have 1: $\vdash f = \text{if}_i\ (\text{init}\ w)\ \text{then}\ f1\ \text{else}\ f2$
using *assms* **by** *auto*
hence 2: $\vdash f = (((\text{init}\ w) \wedge f1) \vee ((\text{init}\ (\neg w)) \wedge f2))$
by (*simp add: ifthenelse-d-def init-defs Valid-def sum.case-eq-if*)
hence 3: $\vdash f \frown g = (((\text{init}\ w) \wedge f1) \frown g \vee ((\text{init}\ (\neg w)) \wedge f2) \frown g)$
by (*rule OrSChopEqvRule*)
have 4: $\vdash ((\text{init}\ w) \wedge f1) \frown g = ((\text{init}\ w) \wedge (f1 \frown g))$
by (*rule StateAndSChop*)
have 5: $\vdash ((\text{init}\ (\neg w)) \wedge f2) \frown g = ((\text{init}\ (\neg w)) \wedge (f2 \frown g))$
by (*rule StateAndSChop*)
have 6: $\vdash f \frown g = (((\text{init}\ w) \wedge f1 \frown g) \vee ((\text{init}\ (\neg w)) \wedge f2 \frown g))$
using 3 4 5 **by** *fastforce*
from 6 **show** ?thesis **by** (*simp add: ifthenelse-d-def init-defs Valid-def sum.case-eq-if*)
qed

lemma *SChopOrEqvRule*:

assumes $\vdash g = (g1 \vee g2)$
shows $\vdash f \frown g = ((f \frown g1) \vee (f \frown g2))$
proof –
have 1: $\vdash g = (g1 \vee g2)$ **using** *assms* **by** *auto*
hence 2: $\vdash f \frown g = (f \frown (g1 \vee g2))$ **by** (*rule RightSChopEqvSChop*)
have 3: $\vdash f \frown (g1 \vee g2) = (f \frown g1 \vee f \frown g2)$ **by** (*rule SChopOrEqv*)
from 2 3 **show** ?thesis **by** *fastforce*
qed

lemma *EmptyOrSChopEqv*:

$\vdash (\text{empty} \vee f) \frown g = (g \vee (f \frown g))$
proof –
have 1: $\vdash (\text{empty} \vee f) \frown g = ((\text{empty} \frown g) \vee (f \frown g))$ **by** (*rule OrSChopEqv*)
have 2: $\vdash \text{empty} \frown g = g$ **by** (*rule EmptySChop*)
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *EmptyOrNextSChopEqv*:

$\vdash (\text{empty} \vee \circ f) \frown g = (g \vee \circ(f \frown g))$
proof –
have 1: $\vdash (\text{empty} \vee \circ f) \frown g = (g \vee ((\circ f) \frown g))$ **by** (*rule EmptyOrSChopEqv*)
have 2: $\vdash (\circ f) \frown g = \circ(f \frown g)$ **by** (*rule NextSChop*)
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *EmptyOrSChopImpRule:*

assumes $\vdash f \longrightarrow \text{empty} \vee f1$

shows $\vdash f \frown g \longrightarrow g \vee (f1 \frown g)$

proof –

have 1: $\vdash f \longrightarrow \text{empty} \vee f1$ **using** *assms* **by** *auto*

hence 2: $\vdash f \frown g \longrightarrow (\text{empty} \vee f1) \frown g$ **by** (*rule LeftSChopImpSChop*)

have 3: $\vdash (\text{empty} \vee f1) \frown g = (g \vee (f1 \frown g))$ **by** (*rule EmptyOrSChopEqv*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *EmptyOrSChopEqvRule:*

assumes $\vdash f = (\text{empty} \vee f1)$

shows $\vdash f \frown g = (g \vee (f1 \frown g))$

proof –

have 1: $\vdash f = (\text{empty} \vee f1)$ **using** *assms* **by** *auto*

hence 2: $\vdash f \frown g = ((\text{empty} \vee f1) \frown g)$ **by** (*rule LeftSChopEqvSChop*)

have 3: $\vdash (\text{empty} \vee f1) \frown g = (g \vee (f1 \frown g))$ **by** (*rule EmptyOrSChopEqv*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *EmptyOrNextSChopImpRule:*

assumes $\vdash f \longrightarrow \text{empty} \vee \circ f1$

shows $\vdash f \frown g \longrightarrow g \vee \circ(f1 \frown g)$

proof –

have 1: $\vdash f \longrightarrow \text{empty} \vee \circ f1$ **using** *assms* **by** *auto*

hence 2: $\vdash f \frown g \longrightarrow (\text{empty} \vee \circ f1) \frown g$ **by** (*rule LeftSChopImpSChop*)

have 3: $\vdash (\text{empty} \vee \circ f1) \frown g = (g \vee \circ(f1 \frown g))$ **by** (*rule EmptyOrNextSChopEqv*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *EmptyOrNextSChopEqvRule:*

assumes $\vdash f = (\text{empty} \vee \circ f1)$

shows $\vdash f \frown g = (g \vee \circ(f1 \frown g))$

proof –

have 1: $\vdash f = (\text{empty} \vee \circ f1)$ **using** *assms* **by** *auto*

hence 2: $\vdash f \frown g = ((\text{empty} \vee \circ f1) \frown g)$ **by** (*rule LeftSChopEqvSChop*)

have 3: $\vdash (\text{empty} \vee \circ f1) \frown g = (g \vee \circ(f1 \frown g))$ **by** (*rule EmptyOrNextSChopEqv*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *SChopEmptyOrImpRule:*

assumes $\vdash g \longrightarrow \text{empty} \vee g1$

shows $\vdash f \frown g \wedge \text{finite} \longrightarrow f \vee (f \frown g1)$

proof –

have 1: $\vdash g \longrightarrow \text{empty} \vee g1$ **using** *assms* **by** *auto*

hence 2: $\vdash f \frown g \longrightarrow (f \frown \text{empty}) \vee (f \frown g1)$ **by** (*rule SChopOrImpRule*)

have 3: $\vdash \text{finite} \longrightarrow f \frown \text{empty} = f$ **by** (*rule SChopEmpty*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *BoxStateSChopBoxAndInflmpBox*:

$\vdash \Box (init\ w) \frown \Box (init\ w) \wedge inf \longrightarrow \Box (init\ w)$

by (*simp add: Valid-def always-defs schop-defs init-defs sum.case-eq-if infinite-defs*
iprefix-length iprefix-0,
metis add.right-neutral interval-nth-suffix interval-nth-zero-intfirst
iprefix-length iprefix-nth isuffix-def le0 le-cases le-iff-add)

lemma *BoxStateSChopBoxEqvBox*:

$\vdash \Box (init\ w) \frown \Box (init\ w) = \Box (init\ w)$

proof –

have 1: $\vdash (\Box (init\ w)) = ((init\ w) \wedge (\text{empty} \vee \bigcirc(\Box (init\ w))))$
by (*rule BoxEqvAndEmptyOrNextBox*)

hence 2: $\vdash (\Box (init\ w) \frown \Box (init\ w)) =$
 $((init\ w) \wedge ((\text{empty} \vee \bigcirc(\Box (init\ w))) \frown \Box (init\ w)))$
by (*metis StateAndSChop inteq-reflection*)

have 3: $\vdash ((\text{empty} \vee \bigcirc(\Box (init\ w))) \frown \Box (init\ w)) =$
 $(\Box (init\ w) \vee \bigcirc(\Box (init\ w) \frown \Box (init\ w)))$
by (*rule EmptyOrNextSChopEqv*)

have 4: $\vdash (\Box (init\ w) \frown \Box (init\ w)) =$
 $((init\ w) \wedge (\Box (init\ w) \vee \bigcirc(\Box (init\ w) \frown \Box (init\ w))))$
using 2 3 **by** *fastforce*

have 5: $\vdash \neg(\Box (init\ w)) \longrightarrow \neg (init\ w) \vee \neg(\bigcirc(\Box (init\ w)))$
by (*rule NotBoxImpNotOrNotNextBox*)

have 6: $\vdash (\Box (init\ w) \frown \Box (init\ w)) \wedge \neg(\Box (init\ w)) \longrightarrow$
 $\bigcirc(\Box (init\ w) \frown \Box (init\ w)) \wedge \neg(\bigcirc(\Box (init\ w)))$
using 4 5 **by** *fastforce*

hence 7: $\vdash \Box (init\ w) \frown \Box (init\ w) \wedge finite \longrightarrow \Box (init\ w)$
by (*rule NextContra*)

have 8: $\vdash \Box (init\ w) \frown \Box (init\ w) \wedge inf \longrightarrow \Box (init\ w)$
by (*rule BoxStateSChopBoxAndInflmpBox*)

have 9: $\vdash \Box (init\ w) \frown \Box (init\ w) \wedge (finite \vee inf) \longrightarrow \Box (init\ w)$
using 7 8 **by** *fastforce*

hence 10: $\vdash \Box (init\ w) \frown \Box (init\ w) \longrightarrow \Box (init\ w)$
using *FiniteOrInfinite* **by** *fastforce*

have 11: $\vdash \Box (init\ w) = ((init\ w) \wedge \Box (init\ w))$
by (*rule BoxEqvAndBox*)

have 12: $\vdash \text{empty} \frown \Box (init\ w) = \Box (init\ w)$
by (*rule EmptySChop*)

have 13: $\vdash ((init\ w) \wedge \text{empty}) \frown \Box (init\ w) = ((init\ w) \wedge (\text{empty} \frown \Box (init\ w)))$
by (*rule StateAndSChop*)

have 14: $\vdash \Box (init\ w) = ((init\ w) \wedge \text{empty}) \frown \Box (init\ w)$
using 11 12 13 **by** *fastforce*

have 15: $\vdash (init\ w) \wedge \text{empty} \longrightarrow \Box (init\ w)$
by (*rule StateAndEmptyImpBoxState*)

hence 16: $\vdash ((init\ w) \wedge \text{empty}) \frown \Box (init\ w) \longrightarrow \Box (init\ w) \frown \Box (init\ w)$
by (*rule LeftSChopImpSChop*)

have 17: $\vdash \Box (init\ w) \longrightarrow \Box (init\ w) \frown \Box (init\ w)$
using 14 16 **by** *fastforce*

from 10 17 **show** *?thesis* **by** *fastforce*

qed

lemma *NotBoxStateImpBoxSYieldsNotBox*:

$\vdash \neg(\Box(\text{init } w)) \longrightarrow (\Box(\text{init } w) \text{ syields } (\neg(\Box(\text{init } w))))$

proof –

have 1: $\vdash \Box(\text{init } w) \frown \Box(\text{init } w) = \Box(\text{init } w)$ **by** (rule *BoxStateSChopBoxEqvBox*)

have 2: $\vdash \Box(\text{init } w) = (\neg \neg(\Box(\text{init } w)))$ **by** *auto*

hence 3: $\vdash \Box(\text{init } w) \frown \Box(\text{init } w) = \Box(\text{init } w) \frown (\neg \neg(\Box(\text{init } w)))$ **by** (rule *RightSChopEqvSChop*)

have 4: $\vdash \neg(\Box(\text{init } w)) \longrightarrow \neg(\Box(\text{init } w) \frown (\neg \neg(\Box(\text{init } w))))$ **using** 1 3 **by** *auto*

from 4 **show** ?thesis **by** (simp add: syields-d-def)

qed

lemma *StateEqvBf*:

$\vdash (\text{init } w) = \text{bf } (\text{init } w)$

proof –

have 1: $\vdash (\text{init } w) \longrightarrow \text{bf } (\text{init } w)$ **by** (rule *StateImpBf*)

have 2: $\vdash \text{bf } (\text{init } w) \wedge \text{finite} \longrightarrow (\text{init } w)$ **by** (rule *BfElim*)

from 1 2 **show** ?thesis

by (metis *DfState Initprop(2) Prop11 bf-d-def int-simps(4) inteq-reflection*)

qed

lemma *TrueSChopEqvDiamond*:

$\vdash \# \text{True} \frown f = \Diamond f$

using *DiamondSChopdef* **by** *fastforce*

lemma *BfAndEqvBfAndBf*:

$\vdash \text{bf}(f \wedge g) = (\text{bf } f \wedge \text{bf } g)$

proof –

have 1: $\vdash f \wedge g \longrightarrow f$ **by** *auto*

have 2: $\vdash \text{bf}(f \wedge g) \longrightarrow \text{bf } f$ **by** (simp add: 1 *BfImpBfRule*)

have 3: $\vdash f \wedge g \longrightarrow g$ **by** *auto*

have 4: $\vdash \text{bf}(f \wedge g) \longrightarrow \text{bf } g$ **by** (simp add: 3 *BfImpBfRule*)

have 5: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ **by** *auto*

have 6: $\vdash \text{bf } f \longrightarrow \text{bf } (g \longrightarrow f \wedge g)$ **by** (simp add: 5 *BfImpBfRule*)

have 7: $\vdash \text{bf } (g \longrightarrow f \wedge g) \longrightarrow (\text{bf } g \longrightarrow \text{bf}(f \wedge g))$ **by** (simp add: *BfImpDist*)

have 8: $\vdash \text{bf } f \wedge \text{bf } g \longrightarrow \text{bf } (f \wedge g)$ **using** 6 7 **by** *fastforce*

from 2 4 8 **show** ?thesis **by** *fastforce*

qed

lemma *BfEqvBfImpAndBfImp*:

$\vdash \text{bf}(f = g) = (\text{bf } (f \longrightarrow g) \wedge \text{bf } (g \longrightarrow f))$

proof –

have 1: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** *auto*

have 2: $\vdash \text{bf}(f = g) = \text{bf}((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** (simp add: 1 *BfEqvBf*)

have 3: $\vdash \text{bf}((f \longrightarrow g) \wedge (g \longrightarrow f)) = (\text{bf } (f \longrightarrow g) \wedge \text{bf } (g \longrightarrow f))$ **by** (simp add: *BfAndEqvBfAndBf*)

from 2 3 **show** ?thesis **by** *fastforce*

qed

lemma *BfEqvImpSChopEqvSChop*:

$\vdash bf(f = f1) \longrightarrow f \frown g = f1 \frown g$

proof —

have 1: $\vdash bf(f = f1) = (bf(f \longrightarrow f1) \wedge bf(f1 \longrightarrow f))$ **by** (*simp add: BfEqvBfImpAndBfImp*)

have 2: $\vdash bf(f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (*simp add: BfSChopImpSChop*)

have 3: $\vdash bf(f1 \longrightarrow f) \longrightarrow f1 \frown g \longrightarrow f \frown g$ **by** (*simp add: BfSChopImpSChop*)

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

qed

lemma *BfEqvDfEqvDf*:

$\vdash bf(f = g) \longrightarrow (df\ f = df\ g)$

proof —

have 1: $\vdash bf(f = g) \longrightarrow (f \frown \#True) = (g \frown \#True)$

using *BfEqvImpSChopEqvSChop* **by** *fastforce*

from 1 **show** ?thesis **by** (*simp add: df-d-def*)

qed

lemma *FinitelImpEqvDfImpRule*:

assumes $\vdash finite \longrightarrow f = g$

shows $\vdash df\ f = df\ g$

proof —

have 1: $\vdash finite \longrightarrow f = g$ **using** *assms* **by** *auto*

have 2: $\vdash bf(f = g)$ **using** 1 **by** (*simp add: FiniteBfGen*)

have 3: $\vdash bf(f = g) \longrightarrow (df\ f = df\ g)$ **by** (*simp add: BfEqvDfEqvDf*)

from 2 3 **show** ?thesis **by** *fastforce*

qed

lemma *DfEmpty*:

$\vdash df\ empty$

proof —

have 1: $\vdash \#True$ **by** *auto*

have 2: $\vdash empty \frown \#True = \#True$ **by** (*rule EmptySChop*)

have 3: $\vdash empty \frown \#True$ **using** 1 2 **by** *auto*

from 3 **show** ?thesis **by** (*simp add: df-d-def*)

qed

lemma *BfImpDf*:

$\vdash bf\ f \longrightarrow df\ f$

proof —

have 1: $\vdash f \longrightarrow (empty \longrightarrow f)$ **by** *auto*

have 2: $\vdash bf\ f \longrightarrow bf(empty \longrightarrow f)$ **by** (*simp add: 1 BfImpBfRule*)

have 3: $\vdash bf(empty \longrightarrow f) \longrightarrow df\ empty \longrightarrow df\ f$ **by** (*simp add: BfImpDfImpDf*)

have 4: $\vdash bf\ f \longrightarrow df\ empty \longrightarrow df\ f$ **using** 2 3 *lift-imp-trans* **by** *blast*
have 5: $\vdash df\ empty$ **by** (*simp add: DfEmpty*)
from 4 5 **show** ?thesis **by** *fastforce*
qed

10.5 Properties of SDa and SBa

lemma *SDaEqvDtDf*:

$\vdash sda\ f = \Diamond (df\ f)$

proof –

have 1: $\vdash \#True \frown (f \frown \#True) = \#True \frown (f \frown \#True)$ **by** *auto*
hence 2: $\vdash \#True \frown (f \frown \#True) = \#True \frown df\ f$ **by** (*simp add: df-d-def*)
have 3: $\vdash \#True \frown (df\ f) = \Diamond (df\ f)$ **by** (*simp add: TrueSChopEqvDiamond*)
have 4: $\vdash \#True \frown (f \frown \#True) = \Diamond (df\ f)$ **using** 2 3 **by** *fastforce*
from 4 **show** ?thesis **by** (*simp add: sda-d-def*)

qed

lemma *SDaEqvDfDt*:

$\vdash sda\ f = df\ (\Diamond f)$

proof –

have 1: $\vdash \#True \frown f = \Diamond f$ **by** (*rule TrueSChopEqvDiamond*)
hence 2: $\vdash (\#True \frown f) \frown \#True = (\Diamond f) \frown \#True$ **by** (*rule LeftSChopEqvSChop*)
hence 3: $\vdash (\#True \frown f) \frown \#True = df\ (\Diamond f)$ **by** (*simp add: df-d-def*)
have 4: $\vdash \#True \frown (f \frown \#True) = (\#True \frown f) \frown \#True$ **by** (*rule SChopAssoc*)
have 5: $\vdash \#True \frown (f \frown \#True) = df\ (\Diamond f)$ **using** 3 4 **by** *fastforce*
from 5 **show** ?thesis **by** (*simp add: sda-d-def*)

qed

lemma *DtDfEqvDfDt*:

$\vdash \Diamond (df\ f) = df\ (\Diamond f)$

by (*meson Prop04 SDaEqvDfDt SDaEqvDtDf*)

lemma *SBaEqvBfBt*:

$\vdash sba\ f = bf\ (\Box f)$

proof –

have 1: $\vdash sda\ (\neg f) = df\ (\Diamond (\neg f))$ **by** (*rule SDaEqvDfDt*)
have 2: $\vdash \Diamond (\neg f) = (\neg (\Box f))$ **by** (*rule DiamondNotEqvNotBox*)
hence 3: $\vdash df\ (\Diamond (\neg f)) = df\ (\neg (\Box f))$ **by** (*rule DfEqvDf*)
have 4: $\vdash sda\ (\neg f) = df\ (\neg (\Box f))$ **using** 1 3 **by** *fastforce*
hence 5: $\vdash (\neg (sda\ (\neg f))) = (\neg (df\ (\neg (\Box f))))$ **by** *auto*
hence 6: $\vdash (\neg (sda\ (\neg f))) = bf\ (\Box f)$ **by** (*simp add: bf-d-def*)
from 6 **show** ?thesis **by** (*simp add: sba-d-def*)

qed

lemma *DfNotEqvNotBf*:

$\vdash df\ (\neg f) = (\neg (bf\ f))$

proof –

have 1: $\vdash bf\ f = (\neg (df\ (\neg f)))$ **by** (*simp add: bf-d-def*)
from 1 **show** ?thesis **by** *auto*

qed

lemma *DfDfNotEqvNotBfBf*:

$\vdash df(df(\neg f)) = (\neg(bf(bf f)))$

proof –

have 1: $\vdash df(\neg f) = (\neg bf f)$ **by** (*simp add: DfNotEqvNotBf*)

have 2: $\vdash df(df(\neg f)) = df(\neg bf f)$ **by** (*simp add: 1 DfEqvDf*)

have 3: $\vdash df(\neg bf f) = (\neg bf(bf f))$ **by** (*simp add: DfNotEqvNotBf*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *DfDtEqvDtDf*:

$\vdash df(\Diamond f) = \Diamond(df f)$

proof –

have 1: $\vdash (\#True \wedge f) \wedge \#True = \#True \wedge (f \wedge \#True)$

using *SChopAssoc* **by** *fastforce*

have 2: $\vdash (\Diamond f) \wedge \#True = \Diamond(f \wedge \#True)$

using 1 **by** (*metis TrueSChopEqvDiamond int-eq*)

from 1 2 **show** *?thesis* **by** (*simp add: df-d-def*)

qed

lemma *DfDtNotEqvNotBfBt*:

$\vdash df(\Diamond(\neg f)) = (\neg(bf(\Box f)))$

proof –

have 1: $\vdash \Diamond(\neg f) = (\neg(\Box f))$ **by** (*simp add: DiamondNotEqvNotBox*)

have 2: $\vdash df(\Diamond(\neg f)) = df(\neg(\Box f))$ **by** (*simp add: 1 DfEqvDf*)

have 3: $\vdash df(\neg(\Box f)) = (\neg(bf(\Box f)))$ **by** (*simp add: DfNotEqvNotBf*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *DtDfNotEqvNotBtBf*:

$\vdash \Diamond(df(\neg f)) = (\neg(\Box(bf f)))$

proof –

have 1: $\vdash df(\neg f) = (\neg(bf f))$ **using** *DfNotEqvNotBf* **by** *blast*

have 2: $\vdash \Diamond(df(\neg f)) = \Diamond(\neg(bf f))$ **by** (*simp add: 1 DiamondEqvDiamond*)

have 3: $\vdash \Diamond(\neg(bf f)) = (\neg\Box(bf f))$ **by** (*simp add: DiamondNotEqvNotBox*)

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *SBaEqvBtBf*:

$\vdash sba f = \Box(bf f)$

proof –

have 1: $\vdash sda(\neg f) = \Diamond(df(\neg f))$ **by** (*rule SDaEqvDtDf*)

have 2: $\vdash df(\neg f) = (\neg(bf f))$ **by** (*rule DfNotEqvNotBf*)

hence 3: $\vdash \Diamond(df(\neg f)) = \Diamond(\neg(bf f))$ **by** (*rule DiamondEqvDiamond*)

have 4: $\vdash (\neg (\Diamond (\neg (bf\ f)))) = \Box (bf\ f)$ **by** (rule NotDiamondNotEqvBox)
have 5: $\vdash (\neg (\neg (sda\ (\neg\ f)))) = \Box (bf\ f)$ **using** 1 2 3 4 **by** fastforce
from 5 **show** ?thesis **by** (simp add: sba-d-def)
qed

lemma BalmpSBa:
 $\vdash ba\ f \longrightarrow sba\ f$
using BaEqvBiBt BilmpBf SBaEqvBfBt **by** fastforce

lemma SDalmpDa:
 $\vdash sda\ f \longrightarrow da\ f$
proof –
have 1: $\vdash ba\ (\neg\ f) \longrightarrow sba\ (\neg\ f)$
 using BalmpSBa **by** blast
have 2: $\vdash \neg\ sba\ (\neg\ f) \longrightarrow \neg\ ba\ (\neg\ f)$
 using 1 **by** fastforce
from 2 **show** ?thesis **by** (simp add: sba-d-def ba-d-def)
qed

lemma BtBfEqvBfBt:
 $\vdash \Box (bf\ f) = bf(\Box\ f)$
proof –
have 1: $\vdash sba\ f = \Box (bf\ f)$ **by** (rule SBaEqvBtBf)
have 2: $\vdash sba\ f = bf(\Box\ f)$ **by** (rule SBaEqvBfBt)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma BoxStateEqvSBaBoxState:
 $\vdash \Box (init\ w) = sba(\Box (init\ w))$
proof –
have 1: $\vdash (init\ w) = bf\ (init\ w)$ **by** (rule StateEqvBf)
hence 2: $\vdash \Box (init\ w) = \Box (bf\ (init\ w))$ **by** (rule BoxEqvBox)
have 3: $\vdash \Box (bf\ (init\ w)) = bf(\Box (init\ w))$ **by** (rule BtBfEqvBfBt)
have 4: $\vdash \Box (init\ w) = \Box(\Box (init\ w))$ **by** (rule BoxEqvBoxBox)
hence 5: $\vdash bf(\Box (init\ w)) = bf(\Box(\Box (init\ w)))$ **by** (rule BfEqvBf)
have 6: $\vdash sba(\Box (init\ w)) = bf(\Box(\Box (init\ w)))$ **by** (rule SBaEqvBfBt)
from 2 3 5 6 **show** ?thesis **by** fastforce
qed

lemma SBalmpBf:
 $\vdash sba\ f \longrightarrow bf\ f$
proof –
have 1: $\vdash sba\ f = \Box (bf\ f)$ **by** (rule SBaEqvBtBf)
have 2: $\vdash \Box (bf\ f) \longrightarrow bf\ f$ **by** (rule BoxElim)
from 1 2 **show** ?thesis **using** lift-imp-trans **by** fastforce
qed

lemma BalmpBf:
 $\vdash ba\ f \longrightarrow bf\ f$

proof –
have 1: $\vdash ba\ f = \Box(bi\ f)$ **by** (rule BaEqvBtBi)
have 2: $\vdash \Box(bi\ f) \longrightarrow bi\ f$ **by** (rule BoxElim)
have 3: $\vdash bi\ f \longrightarrow bf\ f$ **by** (simp add: BImpBf)
from 1 2 3 **show** ?thesis **using** lift-imp-trans **by** fastforce
qed

lemma SBalmpBt:
 $\vdash sba\ f \wedge finite \longrightarrow \Box\ f$

proof –
have 1: $\vdash sba\ f = bf(\Box\ f)$ **by** (rule SBaEqvBfBt)
have 2: $\vdash bf(\Box\ f) \wedge finite \longrightarrow \Box\ f$ **by** (rule BfElim)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma DiamondImpSDa:
 $\vdash \Diamond\ f \wedge finite \longrightarrow sda\ f$
using AndFinitelmpDf SDaEqvDfDt **by** force

lemma DfImpSDa:
 $\vdash df\ f \longrightarrow sda\ f$
using NowImpDiamond SDaEqvDtDf **by** fastforce

lemma BoxAndSChopImport:
 $\vdash \Box\ h \wedge f \frown g \longrightarrow f \frown (h \wedge g)$
proof –
have 1: $\vdash h \longrightarrow g \longrightarrow (h \wedge g)$ **by** auto
hence 2: $\vdash \Box\ h \longrightarrow \Box(g \longrightarrow (h \wedge g))$ **by** (rule ImpBoxRule)
have 3: $\vdash \Box(g \longrightarrow (h \wedge g)) \longrightarrow f \frown g \longrightarrow f \frown (h \wedge g)$ **by** (rule BoxSChopImpSChop)
from 2 3 **show** ?thesis **by** fastforce
qed

lemma SBaAndSChopImport:
 $\vdash sba\ f \wedge finite \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown (f \wedge g1)$
proof –
have 1: $\vdash sba\ f \longrightarrow bf\ f$ **by** (rule SBalmpBf)
have 2: $\vdash bf\ f \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown g1$ **by** (rule BfAndSChopImport)
have 3: $\vdash sba\ f \wedge finite \longrightarrow \Box\ f$ **by** (rule SBalmpBt)
have 4: $\vdash \Box\ f \wedge (f \wedge g) \frown g1 \longrightarrow (f \wedge g) \frown (f \wedge g1)$ **by** (rule BoxAndSChopImport)
from 1 2 3 4 **show** ?thesis **by** fastforce
qed

lemma BaAndSChopImport:
 $\vdash ba\ f \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown (f \wedge g1)$
proof –
have 1: $\vdash ba\ f \longrightarrow bi\ f$ **by** (rule BalmpBi)
have 2: $\vdash bi\ f \wedge (g \frown g1) \longrightarrow (f \wedge g) \frown g1$ **by** (rule BiAndSChopImport)
have 3: $\vdash ba\ f \longrightarrow \Box\ f$ **by** (rule BalmpBt)
have 4: $\vdash \Box\ f \wedge (f \wedge g) \frown g1 \longrightarrow (f \wedge g) \frown (f \wedge g1)$ **by** (rule BoxAndSChopImport)
from 1 2 3 4 **show** ?thesis **by** fastforce

qed

lemma *SChopAndCommute*:

$\vdash f \frown (g \wedge g1) = f \frown (g1 \wedge g)$

proof –

have 1: $\vdash (g \wedge g1) = (g1 \wedge g)$ **by** *auto*

from 1 **show** ?thesis **by** (rule *RightSChopEqvSChop*)

qed

lemma *SChopAndA*:

$\vdash f \frown (g \wedge g1) \longrightarrow f \frown g$

proof –

have 1: $\vdash (g \wedge g1) \longrightarrow g$ **by** *auto*

from 1 **show** ?thesis **by** (rule *RightSChopImpSChop*)

qed

lemma *SChopAndB*:

$\vdash f \frown (g \wedge g1) \longrightarrow f \frown g1$

proof –

have 1: $\vdash (g \wedge g1) \longrightarrow g1$ **by** *auto*

from 1 **show** ?thesis **by** (rule *RightSChopImpSChop*)

qed

lemma *BoxStateAndSChopEqvSChop*:

$\vdash (\Box (init\ w) \wedge finite \wedge (f \frown g)) = ((\Box (init\ w) \wedge f) \frown (\Box (init\ w) \wedge g) \wedge finite)$

proof –

have 1: $\vdash \Box (init\ w) = sba(\Box (init\ w))$

by (rule *BoxStateEqvSBaBoxState*)

have 2: $\vdash sba(\Box (init\ w)) \wedge finite \wedge (f \frown g) \longrightarrow (\Box (init\ w) \wedge f) \frown (\Box (init\ w) \wedge g)$

by (rule *SBaAndSChopImport*)

have 3: $\vdash \Box (init\ w) \wedge finite \wedge (f \frown g) \longrightarrow (\Box (init\ w) \wedge f) \frown (\Box (init\ w) \wedge g)$

using 1 2 **by** *fastforce*

have 11: $\vdash (\Box (init\ w) \wedge f) \frown (\Box (init\ w) \wedge g) \longrightarrow (\Box (init\ w)) \frown (\Box (init\ w) \wedge g)$

by (rule *AndSChopA*)

have 12: $\vdash (\Box (init\ w)) \frown (\Box (init\ w) \wedge g) \longrightarrow (\Box (init\ w)) \frown (\Box (init\ w))$

by (rule *SChopAndA*)

have 13: $\vdash (\Box (init\ w)) \frown (\Box (init\ w)) = \Box (init\ w)$

by (rule *BoxStateSChopBoxEqvBox*)

have 14: $\vdash (\Box (init\ w) \wedge f) \frown (\Box (init\ w) \wedge g) \longrightarrow f \frown (\Box (init\ w) \wedge g)$

by (rule *AndSChopB*)

have 15: $\vdash f \frown (\Box (init\ w) \wedge g) \longrightarrow f \frown g$

by (rule *SChopAndB*)

have 16: $\vdash (\Box (init\ w) \wedge f) \frown (\Box (init\ w) \wedge g) \longrightarrow \Box (init\ w) \wedge (f \frown g)$

using 11 12 13 14 15 **by** *fastforce*

from 3 16 **show** ?thesis **by** *fastforce*

qed

lemma *DfEqvNotBfNot*:

$\vdash df\ f = (\neg (bf\ (\neg f)))$

proof –
have 1: $\vdash bf (\neg f) = (\neg (df (\neg \neg f)))$ **by** (simp add: bf-d-def)
hence 2: $\vdash df (\neg \neg f) = (\neg (bf (\neg f)))$ **by** auto
have 3: $\vdash f = (\neg \neg f)$ **by** auto
hence 4: $\vdash df f = df (\neg \neg f)$ **by** (rule DfEqvDf)
from 2 4 **show** ?thesis **by** auto
qed

lemma SChopAndBoxImport:
 $\vdash f \frown g \wedge \Box h \longrightarrow f \frown (g \wedge h)$
proof –
have 1: $\vdash \Box h \wedge f \frown g \longrightarrow f \frown (h \wedge g)$ **by** (rule BoxAndSChopImport)
have 2: $\vdash f \frown (h \wedge g) = f \frown (g \wedge h)$ **by** (rule SChopAndCommute)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma AndSChopAndCommute:
 $\vdash (f \wedge g) \frown (f1 \wedge g1) = (g \wedge f) \frown (g1 \wedge f1)$
proof –
have 1: $\vdash (f \wedge g) \frown (f1 \wedge g1) = (g \wedge f) \frown (f1 \wedge g1)$ **by** (rule AndSChopCommute)
have 2: $\vdash (g \wedge f) \frown (f1 \wedge g1) = (g \wedge f) \frown (g1 \wedge f1)$ **by** (rule SChopAndCommute)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma SChopImpSChop:
assumes $\vdash f \longrightarrow f1$
 $\vdash g \longrightarrow g1$
shows $\vdash f \frown g \longrightarrow f1 \frown g1$
proof –
have 1: $\vdash f \longrightarrow f1$ **using** assms **by** auto
hence 2: $\vdash f \frown g \longrightarrow f1 \frown g$ **by** (rule LeftSChopImpSChop)
have 3: $\vdash g \longrightarrow g1$ **using** assms **by** auto
hence 4: $\vdash f1 \frown g \longrightarrow f1 \frown g1$ **by** (rule RightSChopImpSChop)
from 2 4 **show** ?thesis **by** fastforce
qed

lemma SChopEqvSChop:
assumes $\vdash f = f1$
 $\vdash g = g1$
shows $\vdash f \frown g = f1 \frown g1$
proof –
have 1: $\vdash f = f1$ **using** assms **by** auto
hence 2: $\vdash f \frown g = f1 \frown g$ **by** (rule LeftSChopEqvSChop)
have 3: $\vdash g = g1$ **using** assms **by** auto
hence 4: $\vdash f1 \frown g = f1 \frown g1$ **by** (rule RightSChopEqvSChop)
from 2 4 **show** ?thesis **by** fastforce
qed

lemma *BoxSChopImpSChopBox*:

$\vdash \Box h \longrightarrow f \frown g \longrightarrow f \frown (\Box h \wedge g)$

proof –

have 1: $\vdash \Box h \longrightarrow \Box(g \longrightarrow \Box h \wedge g)$ **by** (rule *BoxImpBoxImpBox*)

have 2: $\vdash \Box(g \longrightarrow \Box h \wedge g) \longrightarrow f \frown g \longrightarrow f \frown (\Box h \wedge g)$ **by** (rule *BoxSChopImpSChop*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *NotChopEqvSYieldsNot*:

$\vdash (\neg (f \frown g)) = f \text{ syields } (\neg g)$

proof –

have 1: $\vdash g = (\neg \neg g)$ **by** auto

hence 2: $\vdash f \frown g = f \frown (\neg \neg g)$ **by** (rule *RightSChopEqvSChop*)

hence 3: $\vdash (\neg (f \frown g)) = (\neg (f \frown (\neg \neg g)))$ **by** auto

from 3 **show** ?thesis **by** (simp add: syields-d-def)

qed

lemma *NotDfFalse*:

$\vdash \neg (df \# False)$

proof –

have 1: $\vdash (init \# True) \longrightarrow bf (init \# True)$ **by** (rule *StateImpBf*)

hence 2: $\vdash \# True \longrightarrow bf \# True$ **by** (auto simp: bf-defs sum.case-eq-if)

have 3: $\vdash \# True$ **by** auto

have 4: $\vdash bf \# True$ **using** 2 3 **MP** **by** auto

hence 5: $\vdash \neg (df (\neg \# True))$ **by** (simp add: bf-d-def)

have 6: $\vdash (\neg \# True) = \# False$ **by** auto

hence 7: $\vdash df (\neg \# True) = df \# False$ **by** (rule *DfEqvDf*)

from 5 7 **show** ?thesis **by** auto

qed

lemma *StateAndEmptySChop*:

$\vdash ((init\ w) \wedge empty) \frown f = ((init\ w) \wedge f)$

proof –

have 1: $\vdash ((init\ w) \wedge empty) \frown f = ((init\ w) \wedge empty \frown f)$ **by** (rule *StateAndSChop*)

have 2: $\vdash empty \frown f = f$ **by** (rule *EmptySChop*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *StateAndNextSChop*:

$\vdash ((init\ w) \wedge \bigcirc f) \frown g = ((init\ w) \wedge \bigcirc(f \frown g))$

proof –

have 1: $\vdash ((init\ w) \wedge \bigcirc f) \frown g = ((init\ w) \wedge (\bigcirc f) \frown g)$ **by** (rule *StateAndSChop*)

have 2: $\vdash (\bigcirc f) \frown g = \bigcirc(f \frown g)$ **by** (rule *NextSChop*)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma *NextStateAndSChop*:

$\vdash \bigcirc(((init\ w) \wedge f) \frown g) = \bigcirc((init\ w) \wedge \bigcirc(f \frown g))$

proof –
have 1: $\vdash ((init\ w) \wedge f) \frown g = ((init\ w) \wedge f \frown g)$ **by** (rule StateAndSChop)
hence 2: $\vdash \bigcirc(((init\ w) \wedge f) \frown g) = \bigcirc((init\ w) \wedge f \frown g)$ **by** (rule NextEqvNext)
have 3: $\vdash \bigcirc((init\ w) \wedge f \frown g) = (\bigcirc (init\ w) \wedge \bigcirc(f \frown g))$ **by** (rule NextAndEqvNextAndNext)
from 2 3 **show** ?thesis **by** fastforce
qed

lemma StateSYieldsEqv:

$\vdash ((init\ w) \longrightarrow (f\ syields\ g)) = ((init\ w) \wedge f)\ syields\ g$

proof –
have 1: $\vdash ((init\ w) \wedge f) \frown (\neg g) = ((init\ w) \wedge f \frown (\neg g))$ **by** (rule StateAndSChop)
hence 2: $\vdash ((init\ w) \longrightarrow \neg(f \frown (\neg g))) = (\neg(((init\ w) \wedge f) \frown (\neg g)))$ **by** auto
from 2 **show** ?thesis **by** (simp add: syields-d-def)
qed

lemma StateAndDf:

$\vdash ((init\ w) \wedge df\ f) = df\ ((init\ w) \wedge f)$

proof –
have 1: $\vdash ((init\ w) \wedge f) \frown \#True = ((init\ w) \wedge f \frown \#True)$ **by** (rule StateAndSChop)
from 1 **show** ?thesis **by** (metis df-d-def inteq-reflection)
qed

lemma DfNext:

$\vdash df(\bigcirc f) = \bigcirc(df\ f)$

proof –
have 1: $\vdash (\bigcirc f) \frown \#True = \bigcirc(f \frown \#True)$ **by** (rule NextSChop)
from 1 **show** ?thesis **by** (simp add: df-d-def)
qed

lemma DfNextState:

$\vdash df(\bigcirc (init\ w)) = \bigcirc (init\ w)$

proof –
have 1: $\vdash df(\bigcirc (init\ w)) = \bigcirc(df\ (init\ w))$ **by** (rule DfNext)
have 2: $\vdash df\ (init\ w) = (init\ w)$ **by** (rule DfState)
hence 3: $\vdash \bigcirc(df\ (init\ w)) = \bigcirc (init\ w)$ **by** (rule NextEqvNext)
from 1 3 **show** ?thesis **by** fastforce
qed

lemma DfStateAndNextStateEqvStateAndNextState:

$\vdash df(init\ w \wedge \bigcirc (init\ w1)) = (init\ w \wedge \bigcirc (init\ w1))$

proof –
have 1: $\vdash (init\ w \wedge \bigcirc (init\ w1)) \frown \#True = (init\ w \wedge \bigcirc ((init\ w1) \frown \#True))$
using StateAndNextSChop **by** blast
have 2: $\vdash df(init\ w \wedge \bigcirc (init\ w1)) = (init\ w \wedge \bigcirc ((init\ w1) \frown \#True))$
using 1 **by** (simp add: df-d-def)
have 3: $\vdash df(init\ w1) = init\ w1$
by (simp add: DfState)
have 4: $\vdash skip \frown df(init\ w1) = skip \frown (init\ w1)$
by (simp add: 3 RightSChopEqvSChop)

have 5: $\vdash \bigcirc(df(init\ w1)) = \bigcirc(init\ w1)$
by (*simp add: 3 NextEqvNext*)
from 2 5 **show** ?thesis **by** (*metis df-d-def int-eq*)
qed

lemma *StatImpBfGen*:

assumes $\vdash (init\ w) \longrightarrow f$
shows $\vdash (init\ w) \longrightarrow bf\ f$

proof –

have 1: $\vdash (init\ w) \longrightarrow f$ **using** *assms* **by** *auto*
hence 2: $\vdash \neg f \longrightarrow \neg (init\ w)$ **by** *auto*
hence 3: $\vdash df(\neg f) \longrightarrow df(\neg (init\ w))$ **by** (*rule DfImpDf*)
hence 4: $\vdash df(\neg f) \longrightarrow df(init(\neg w))$ **by** (*metis Initprop(2) inteq-reflection*)
have 5: $\vdash df(init(\neg w)) = (init(\neg w))$ **by** (*rule DfState*)
have 6: $\vdash df(\neg f) \longrightarrow \neg (init\ w)$ **using** 4 5 **using** *Initprop(2)* **by** *fastforce*
hence 7: $\vdash (init\ w) \longrightarrow \neg(df(\neg f))$ **by** *auto*
from 7 **show** ?thesis **by** (*simp add: bf-d-def*)
qed

lemma *SChopAndNotSChopImp*:

$\vdash f \frown g \wedge \neg(f \frown g1) \longrightarrow f \frown (g \wedge \neg g1)$

proof –

have 1: $\vdash g \longrightarrow (g \wedge \neg g1) \vee g1$ **by** *auto*
hence 2: $\vdash f \frown g \longrightarrow f \frown ((g \wedge \neg g1) \vee g1)$ **by** (*rule RightSChopImpSChop*)
have 3: $\vdash f \frown ((g \wedge \neg g1) \vee g1) \longrightarrow (f \frown (g \wedge \neg g1)) \vee (f \frown g1)$ **by** (*rule SChopOrImp*)
have 4: $\vdash f \frown g \longrightarrow f \frown (g \wedge \neg g1) \vee f \frown g1$ **using** 2 3 *MP* **by** *fastforce*
from 4 **show** ?thesis **by** *auto*
qed

lemma *SChopAndSYieldsImp*:

$\vdash f \frown g \wedge f\ syields\ g1 \longrightarrow f \frown (g \wedge g1)$

proof –

have 1: $\vdash g \longrightarrow (g \wedge g1) \vee \neg g1$ **by** *auto*
hence 2: $\vdash f \frown g \longrightarrow f \frown ((g \wedge g1) \vee \neg g1)$ **by** (*rule RightSChopImpSChop*)
have 3: $\vdash f \frown ((g \wedge g1) \vee \neg g1) \longrightarrow (f \frown (g \wedge g1)) \vee (f \frown (\neg g1))$ **by** (*rule SChopOrImp*)
have 4: $\vdash f \frown g \longrightarrow f \frown (g \wedge g1) \vee f \frown (\neg g1)$ **using** 2 3 *MP* **by** *fastforce*
hence 5: $\vdash f \frown g \wedge \neg(f \frown (\neg g1)) \longrightarrow f \frown (g \wedge g1)$ **by** *auto*
from 5 **show** ?thesis **by** (*simp add: syields-d-def*)
qed

lemma *SChopAndSYieldsMP*:

$\vdash f \frown g \wedge f\ syields\ (g \longrightarrow g1) \longrightarrow f \frown g1$

proof –

have 1: $\vdash f \frown g \wedge f\ syields\ (g \longrightarrow g1) \longrightarrow f \frown (g \wedge (g \longrightarrow g1))$ **by** (*rule SChopAndSYieldsImp*)
have 2: $\vdash g \wedge (g \longrightarrow g1) \longrightarrow g1$ **by** *auto*
hence 3: $\vdash f \frown (g \wedge (g \longrightarrow g1)) \longrightarrow f \frown g1$ **by** (*rule RightSChopImpSChop*)
from 1 3 **show** ?thesis **by** *fastforce*
qed

lemma *OrSYieldsImp*:

$\vdash (f \vee f1) \text{ syields } g = ((f \text{ syields } g) \wedge (f1 \text{ syields } g))$
proof –
have 1: $\vdash ((f \vee f1) \frown (\neg g)) = ((f \frown (\neg g)) \vee (f1 \frown (\neg g)))$ **by** (rule OrSChopEqv)
hence 2: $\vdash (\neg ((f \vee f1) \frown (\neg g))) = (\neg (f \frown (\neg g)) \wedge \neg (f1 \frown (\neg g)))$ **by** auto
from 2 **show** ?thesis **by** (simp add: syields-d-def)
qed

lemma LeftSYieldsImpSYields:
assumes $\vdash f \longrightarrow f1$
shows $\vdash (f1 \text{ syields } g) \longrightarrow (f \text{ syields } g)$
proof –
have 1: $\vdash f \longrightarrow f1$ **using** assms **by** auto
hence 2: $\vdash f \frown (\neg g) \longrightarrow f1 \frown (\neg g)$ **by** (rule LeftSChopImpSChop)
hence 3: $\vdash \neg (f1 \frown (\neg g)) \longrightarrow \neg (f \frown (\neg g))$ **by** auto
from 3 **show** ?thesis **by** (simp add: syields-d-def)
qed

lemma LeftSYieldsEqvSYields:
assumes $\vdash f = f1$
shows $\vdash (f \text{ syields } g) = (f1 \text{ syields } g)$
proof –
have 1: $\vdash f = f1$ **using** assms **by** auto
hence 2: $\vdash f \frown (\neg g) = f1 \frown (\neg g)$ **by** (rule LeftSChopEqvSChop)
hence 3: $\vdash (\neg (f \frown (\neg g))) = (\neg (f1 \frown (\neg g)))$ **by** auto
from 3 **show** ?thesis **by** (simp add: syields-d-def)
qed

10.6 Properties of SFin

lemma SFinEqvTrueSChopAndEmpty:
 $\vdash \text{sfin } f = \# \text{True} \frown (f \wedge \text{empty})$
proof –
have 01: $\vdash \text{sfin } f = (\neg \text{fin } (\neg f))$
by (simp add: sfin-d-def)
have 02: $\vdash (\neg \text{fin } (\neg f)) = (\neg (\Box (\text{empty} \longrightarrow \neg f)))$
by (simp add: fin-d-def)
have 03: $\vdash (\neg (\Box (\text{empty} \longrightarrow \neg f))) = \Diamond (\neg (\text{empty} \longrightarrow \neg f))$
by (simp add: always-d-def)
have 04: $\vdash \neg (\text{empty} \longrightarrow \neg f) = (\text{empty} \wedge f)$
by auto
have 05: $\vdash \Diamond (\neg (\text{empty} \longrightarrow \neg f)) = \Diamond (\text{empty} \wedge f)$
using 04
using inteq-reflection **by** fastforce
from 01 02 03 05 **show** ?thesis
by (metis SChopAndCommute TrueSChopEqvDiamond inteq-reflection)
qed

lemma DiamondSFin:
 $\vdash \Diamond (\text{sfin } w) = \text{sfin } w$

by (*metis* (*no-types*, *lifting*) *ChopAssoc FiniteChopFiniteEqvFinite FiniteOr FiniteOrInfinite InfEqvNotFinite OrFiniteInf SFinEqvTrueSCHopAndEmpty finite-d-def int-eq-true int-simps*(21) *inteq-reflection schop-d-def sometimes-d-def*)

lemma *SChopSFinExportA*:

$\vdash f \frown (g \wedge \text{sf}in\ w) \longrightarrow \text{sf}in\ w$

using *DiamondSFin*

by (*metis* *SChopAndB SChopImpDiamond inteq-reflection lift-imp-trans*)

lemma *SFinImpBox*:

$\vdash \text{sf}in\ w \longrightarrow \Box(\text{sf}in\ w)$

by

(*metis* (*mono-tags*, *lifting*) *DiamondFin always-d-def intl int-eq int-simps*(4) *sf}in-d-def unl-lift2*)

lemma *SFinAndSCHopImport*:

$\vdash (\text{sf}in\ w) \wedge (f \frown g) \longrightarrow f \frown ((\text{sf}in\ w) \wedge g)$

proof –

have 1: $\vdash \text{sf}in\ w \longrightarrow \Box(\text{sf}in\ w)$ **by** (*rule SFinImpBox*)

hence 2: $\vdash \text{sf}in\ w \wedge (f \frown g) \longrightarrow \Box(\text{sf}in\ w) \wedge (f \frown g)$ **by** *auto*

have 3: $\vdash \Box(\text{sf}in\ w) \wedge (f \frown g) \longrightarrow f \frown ((\text{sf}in\ w) \wedge g)$ **using** *BoxAndSCHopImport* **by** *blast*

from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*

qed

lemma *SFinAndSCHop*:

$\vdash (f \frown (g \wedge \text{sf}in\ w)) = (\text{sf}in\ w \wedge f \frown g)$

using *SFinAndSCHopImport SChopSFinExportA SChopAndA SChopAndCommute*

by *fastforce*

lemma *SChopAndEmptyEqvEmptySCHopEmpty*:

$\vdash ((f \frown g) \wedge \text{empty}) = (f \wedge \text{empty}) \frown (g \wedge \text{empty})$

by (*auto simp: empty-defs schop-defs sum.case-eq-if*)

lemma *SFinAndEmpty*:

$\vdash ((\text{sf}in\ w) \wedge \text{empty}) = (w \wedge \text{empty})$

proof –

have 1: $\vdash ((\text{sf}in\ w) \wedge \text{empty}) = (\# \text{True} \frown (w \wedge \text{empty}) \wedge \text{empty})$

using *SFinEqvTrueSCHopAndEmpty* **by** *fastforce*

have 2: $\vdash (\# \text{True} \frown (w \wedge \text{empty}) \wedge \text{empty}) = ((\# \text{True} \wedge \text{empty}) \frown (w \wedge \text{empty}))$

using *SChopAndEmptyEqvEmptySCHopEmpty*

by (*smt int-eq int-iffD2 lift-and-com Prop10 Prop12*)

have 3: $\vdash (\# \text{True} \wedge \text{empty}) \frown (w \wedge \text{empty}) = (\text{empty} \frown (w \wedge \text{empty}))$

using *LeftSCHopEqvSCHop* **by** *fastforce*

have 4: $\vdash (\text{empty} \frown (w \wedge \text{empty})) = (w \wedge \text{empty})$

using *EmptySCHop* **by** *blast*

from 1 2 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *AndSFinEqvSCHopAndEmpty*:

$\vdash ((f \wedge \text{finite}) \wedge \text{sf}in\ g) = f \frown (g \wedge \text{empty})$

proof –
have 1: $\vdash ((f \wedge \text{finite}) \wedge \text{sf}in\ g) = (f \frown \text{empty} \wedge \text{sf}in\ g)$
using *SChopEmpty*
by (*metis* (*no-types*, *lifting*) *DiamondEmptyEqvFinite FinitelmpAnd Prop10 SChoplmpDiamond*
inteq-reflection lift-and-com)
have 2: $\vdash (\text{sf}in\ g \wedge f \frown \text{empty}) = (f \frown (\text{empty} \wedge \text{sf}in\ g))$
using *SFinAndSChop* **by** *fastforce*
have 3: $\vdash (\text{empty} \wedge \text{sf}in\ g) = (\text{sf}in\ g \wedge \text{empty})$
by *auto*
have 4: $\vdash (\text{sf}in\ g \wedge \text{empty}) = (g \wedge \text{empty})$
using *SFinAndEmpty* **by** *metis*
have 5: $\vdash (\text{empty} \wedge \text{sf}in\ g) = (g \wedge \text{empty})$
using 3 4 **by** *auto*
hence 6: $\vdash f \frown (\text{empty} \wedge \text{sf}in\ g) = f \frown (g \wedge \text{empty})$
using *RightSChopEqvSChop* **by** *blast*
from 1 2 5 **show** ?thesis **by** (*metis* *inteq-reflection lift-and-com*)
qed

lemma *AndSFinEqvSChopStateAndEmpty*:
 $\vdash ((f \wedge \text{finite}) \wedge \text{sf}in\ (\text{init}\ w)) = f \frown ((\text{init}\ w) \wedge \text{empty})$
using *AndSFinEqvSChopAndEmpty* **by** *blast*

lemma *DiamondEqvEmptyOrNextDiamond*:
 $\vdash \Diamond f = (f \vee \bigcirc(\Diamond f))$

proof –
have 1: $\vdash \Box(\neg f) = ((\neg f) \wedge \text{wnext}(\Box(\neg f)))$
by (*simp* *add*: *BoxEqvAndWnextBox*)
have 2: $\vdash (\neg \Diamond f) = ((\neg f) \wedge \text{wnext}(\Box(\neg f)))$
using 1 **by** (*simp* *add*: *always-d-def*)
have 3: $\vdash \Diamond f = (f \vee \neg(\text{wnext}(\Box(\neg f))))$
using 2 **by** *auto*
have 4: $\vdash (\neg(\text{wnext}(\Box(\neg f)))) = \bigcirc(\neg\Box(\neg f))$
by (*simp* *add*: *wnext-d-def*)
have 5: $\vdash \neg\Box(\neg f) = \Diamond f$
by (*simp* *add*: *always-d-def*)
have 6: $\vdash \bigcirc(\neg\Box(\neg f)) = \bigcirc(\Diamond f)$
using 5 **using** *inteq-reflection* **by** *force*
from 3 4 6 **show** ?thesis **by** *fastforce*
qed

lemma *SFinStateEqvStateAndEmptyOrNextSFinState*:
 $\vdash \text{sf}in\ (\text{init}\ w) = (((\text{init}\ w) \wedge \text{empty}) \vee \bigcirc(\text{sf}in\ (\text{init}\ w)))$

proof –
have 01: $\vdash \text{sf}in\ (\text{init}\ w) = \# \text{True} \frown ((\text{init}\ w) \wedge \text{empty})$
by (*simp* *add*: *SFinEqvTrueSChopAndEmpty*)
have 02: $\vdash \# \text{True} \frown ((\text{init}\ w) \wedge \text{empty}) = \Diamond((\text{init}\ w) \wedge \text{empty})$
by (*simp* *add*: *TrueSChopEqvDiamond*)
have 03: $\vdash \Diamond((\text{init}\ w) \wedge \text{empty}) = (((\text{init}\ w) \wedge \text{empty}) \vee \bigcirc(\text{sf}in\ (\text{init}\ w)))$
using *DiamondEqvEmptyOrNextDiamond* 02 01 **by** (*metis* *inteq-reflection*)
from 01 02 03 **show** ?thesis **by** *fastforce*

qed

lemma *SFinSChopEqvOr*:

$\vdash (sfin\ (init\ w)) \frown f = (((init\ w) \wedge f) \vee \bigcirc((sfin\ (init\ w)) \frown f))$

proof –

have 1: $\vdash sfin\ (init\ w) = (((init\ w) \wedge empty) \vee \bigcirc(sfin\ (init\ w)))$

by (rule *SFinStateEqvStateAndEmptyOrNextSFinState*)

hence 2: $\vdash (sfin\ (init\ w)) \frown f = (((init\ w) \wedge empty) \vee \bigcirc(sfin\ (init\ w))) \frown f$

by (rule *LeftSChopEqvSChop*)

have 3: $\vdash (((init\ w) \wedge empty) \vee \bigcirc(sfin\ (init\ w))) \frown f = (((init\ w) \wedge empty) \frown f \vee (\bigcirc(sfin\ (init\ w))) \frown f)$

by (rule *OrSChopEqv*)

have 4: $\vdash ((init\ w) \wedge empty) \frown f = ((init\ w) \wedge f)$

by (rule *StateAndEmptySChop*)

have 5: $\vdash (\bigcirc(sfin\ (init\ w))) \frown f = \bigcirc((sfin\ (init\ w)) \frown f)$

by (rule *NextSChop*)

from 2 3 4 5 **show** ?thesis **by** fastforce

qed

lemma *SFinSChopEqvDiamond*:

$\vdash (sfin\ (init\ w)) \frown f = \Diamond((init\ w) \wedge f)$

proof –

have 1: $\vdash (sfin\ (init\ w)) = (\#True \frown ((init\ w) \wedge empty))$

by (simp add: *SFinEqvTrueSChopAndEmpty*)

hence 2: $\vdash (sfin\ (init\ w)) \frown f = (\#True \frown ((init\ w) \wedge empty)) \frown f$

by (rule *LeftSChopEqvSChop*)

have 3: $\vdash \#True \frown ((init\ w) \wedge empty) \frown f = (\#True \frown ((init\ w) \wedge empty)) \frown f$

by (rule *SChopAssoc*)

have 4: $\vdash \#True \frown ((init\ w) \wedge empty) \frown f = \Diamond((init\ w) \wedge empty) \frown f$

using *TrueSChopEqvDiamond* **by** blast

have 5: $\vdash ((init\ w) \wedge empty) \frown f = ((init\ w) \wedge f)$

using *StateAndEmptySChop* **by** blast

hence 6: $\vdash \Diamond((init\ w) \wedge empty) \frown f = \Diamond((init\ w) \wedge f)$

by (rule *DiamondEqvDiamond*)

from 2 3 4 6 **show** ?thesis **by** fastforce

qed

lemma *SFinSYields*:

$\vdash (sfin\ (init\ w))\ syields\ (init\ w)$

proof –

have 1: $\vdash (sfin\ (init\ w)) \frown (\neg(init\ w)) = \Diamond((init\ w) \wedge \neg(init\ w))$

by (rule *SFinSChopEqvDiamond*)

have 2: $\vdash \neg(\Diamond((init\ w) \wedge \neg(init\ w)))$ **by** (rule *NotDiamondAndNot*)

have 3: $\vdash \neg((sfin\ (init\ w)) \frown (\neg(init\ w)))$ **using** 1 2 **by** fastforce

from 3 **show** ?thesis **by** (simp add: *syields-d-def*)

qed

lemma *AndFiniteImpAndSFinStateOrSFinNotState*:

$\vdash f \wedge finite \longrightarrow (f \wedge sfin\ (init\ w)) \vee (f \wedge sfin\ (\neg(init\ w)))$

by (simp add: finite-defs sfin-defs Valid-def sum.case-eq-if)

lemma *AndSFinSChopEqvStateAndSChop*:

$\vdash (f \wedge \text{sfin } (\text{init } w)) \frown g = f \frown ((\text{init } w) \wedge g)$

proof –

have 1: $\vdash (\text{sfin } (\text{init } w)) \text{syields } (\text{init } w)$

by (rule *SFinSYields*)

have 2: $\vdash f \wedge \text{sfin } (\text{init } w) \longrightarrow \text{sfin } (\text{init } w)$

by auto

hence 3: $\vdash (\text{sfin } (\text{init } w)) \text{syields } (\text{init } w) \longrightarrow$

$(f \wedge \text{sfin } (\text{init } w)) \text{syields } (\text{init } w)$

using *LeftSYieldsImpSYields* **by** *metis*

have 4: $\vdash (f \wedge \text{sfin } (\text{init } w)) \text{syields } (\text{init } w)$

using 1 3 *MP* **by** *fastforce*

have 5: $\vdash (f \wedge \text{sfin } (\text{init } w)) \frown g \wedge (f \wedge \text{sfin } (\text{init } w)) \text{syields } (\text{init } w)$

$\longrightarrow (f \wedge \text{sfin } (\text{init } w)) \frown (g \wedge (\text{init } w))$

by (rule *SChopAndSYieldsImp*)

have 6: $\vdash (f \wedge \text{sfin } (\text{init } w)) \frown g \longrightarrow (f \wedge \text{sfin } (\text{init } w)) \frown (g \wedge (\text{init } w))$

using 4 5 **by** *fastforce*

have 7: $\vdash (f \wedge \text{sfin } (\text{init } w)) \frown (g \wedge (\text{init } w)) \longrightarrow f \frown (g \wedge (\text{init } w))$

by (rule *AndSChopA*)

have 8: $\vdash g \wedge (\text{init } w) \longrightarrow (\text{init } w) \wedge g$

by auto

hence 9: $\vdash f \frown (g \wedge (\text{init } w)) \longrightarrow f \frown ((\text{init } w) \wedge g)$

by (rule *RightSChopImpSChop*)

have 10: $\vdash (f \wedge \text{sfin } (\text{init } w)) \frown g \longrightarrow f \frown ((\text{init } w) \wedge g)$

using 6 7 9 **by** *fastforce*

have 11: $\vdash (f \wedge \text{finite}) \longrightarrow (f \wedge \text{sfin } (\text{init } w)) \vee (f \wedge \text{sfin } (\neg (\text{init } w)))$

using *AndFinitImpAndSFinStateOrSFinNotState* **by** *blast*

hence 12: $\vdash f \frown ((\text{init } w) \wedge g) \longrightarrow$

$((f \wedge \text{sfin } (\text{init } w)) \vee (f \wedge \text{sfin } (\neg (\text{init } w)))) \frown ((\text{init } w) \wedge g)$

by (*metis* *FinitImp LeftChopImpChop inteq-reflection schop-d-def*)

have 13: $\vdash ((f \wedge \text{sfin } (\text{init } w)) \vee (f \wedge \text{sfin } (\neg (\text{init } w)))) \frown ((\text{init } w) \wedge g)$

$=$

$((f \wedge \text{sfin } (\text{init } w)) \frown ((\text{init } w) \wedge g)) \vee (f \wedge \text{sfin } (\neg (\text{init } w))) \frown ((\text{init } w) \wedge g)$

by (rule *OrSChopEqv*)

have 14: $\vdash (f \wedge \text{sfin } (\text{init } (\neg w))) \frown ((\text{init } w) \wedge g) \longrightarrow \Diamond (\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g)$

using *SFinSChopEqvDiamond*

by (*metis* *SChopImpSChop Prop12 int-iffD1 inteq-reflection lift-and-com*)

have 141: $\vdash \neg (\Diamond (\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g)) \longrightarrow$

$\neg ((f \wedge \text{sfin } (\text{init } (\neg w))) \frown ((\text{init } w) \wedge g))$

using 14 **by** *fastforce*

have 15: $\vdash \neg (\Diamond (\text{init } (\neg w)) \wedge ((\text{init } w) \wedge g))$

using *NotDiamondAndNot Initprop(2)* **by** (*auto simp: sometimes-defs init-defs sum.case-eq-if*)

have 151: $\vdash \neg ((f \wedge \text{sfin } (\text{init } (\neg w))) \frown ((\text{init } w) \wedge g))$

using 15 141 **by** *fastforce*

have 1511: $\vdash (f \wedge \text{sfin } (\neg (\text{init } w))) \frown ((\text{init } w) \wedge g) \longrightarrow \#False$

using 151 **by** (*metis* *Initprop(2) int-simps(14) inteq-reflection*)

have 152: $\vdash (f \wedge \text{sfin } (\text{init } w)) \frown ((\text{init } w) \wedge g) \vee (f \wedge \text{sfin } (\neg (\text{init } w))) \frown ((\text{init } w) \wedge g) \longrightarrow$

$(f \wedge \text{sfin } (\text{init } w)) \frown ((\text{init } w) \wedge g)$

using 1511 by fastforce
 have 16: $\vdash f \frown ((\text{init } w) \wedge g) \longrightarrow (f \wedge \text{sfinit } (\text{init } w)) \frown ((\text{init } w) \wedge g)$
 using 12 13 152
 by (smt inteq-reflection lift-and-com lift-imp-trans)
 have 17: $\vdash (f \wedge \text{sfinit } (\text{init } w)) \frown ((\text{init } w) \wedge g) \longrightarrow (f \wedge \text{sfinit } (\text{init } w)) \frown g$
 by (rule SChopAndB)
 have 18: $\vdash f \frown ((\text{init } w) \wedge g) \longrightarrow (f \wedge \text{sfinit } (\text{init } w)) \frown g$
 using 16 17 by fastforce
 from 10 18 show ?thesis by fastforce
 qed

lemma DfAndSFinEqvSChopState:

$\vdash \text{df } (f \wedge \text{sfinit } (\text{init } w)) = f \frown (\text{init } w)$

proof –

have 1: $\vdash (f \wedge \text{sfinit } (\text{init } w)) \frown \# \text{True} = f \frown ((\text{init } w) \wedge \# \text{True})$
 by (rule AndSFinSChopEqvStateAndSChop)
 have 2: $\vdash ((\text{init } w) \wedge \# \text{True}) = (\text{init } w)$ by auto
 hence 3: $\vdash (f \frown ((\text{init } w) \wedge \# \text{True})) = (f \frown (\text{init } w))$ by (rule RightSChopEqvSChop)
 have 4: $\vdash (f \wedge \text{sfinit } (\text{init } w)) \frown \# \text{True} = f \frown (\text{init } w)$ using 1 3 by auto
 from 4 show ?thesis by (simp add: df-d-def)
 qed

lemma SFinNotStateEqvNotSFinState:

$\vdash \text{finite} \longrightarrow (\neg (\text{sfinit } (\text{init } w))) = (\text{sfinit } (\text{init } (\neg w)))$

using SFinEqvTrueSChopAndEmpty

by (metis InitAndEmptyEqvAndEmpty SFinprop(3) inteq-reflection)

lemma BfImpSFinEqvSYieldsState:

$\vdash \text{bf } (f \longrightarrow \text{sfinit } (\text{init } w)) = f \text{ syields } (\text{init } w)$

proof –

have 1: $\vdash \text{df } (f \wedge \text{sfinit } (\text{init } (\neg w))) = f \frown (\text{init } (\neg w))$
 by (rule DfAndSFinEqvSChopState)
 have 2: $\vdash \text{finite} \longrightarrow (f \wedge \text{sfinit } (\text{init } (\neg w))) = (f \wedge \neg (\text{sfinit } (\text{init } w)))$
 using SFinNotStateEqvNotSFinState by fastforce
 have 3: $\vdash (f \wedge \neg (\text{sfinit } (\text{init } w))) = (\neg (f \longrightarrow \text{sfinit } (\text{init } w)))$
 by auto
 have 4: $\vdash \text{finite} \longrightarrow (f \wedge \text{sfinit } (\text{init } (\neg w))) = (\neg (f \longrightarrow \text{sfinit } (\text{init } w)))$
 using 2 3 by fastforce
 hence 5: $\vdash \text{df } (f \wedge \text{sfinit } (\text{init } (\neg w))) = \text{df } (\neg (f \longrightarrow \text{sfinit } (\text{init } w)))$
 by (metis DfEqvNotBfNot FinitImpAnd df-d-def inteq-reflection schop-d-def)
 have 6: $\vdash \text{df } (\neg (f \longrightarrow \text{sfinit } (\text{init } w))) = (\neg (\text{bf } (f \longrightarrow \text{sfinit } (\text{init } w))))$
 by (rule DfNotEqvNotBf)
 have 7: $\vdash \neg (\text{bf } (f \longrightarrow \text{sfinit } (\text{init } w))) = f \frown (\text{init } (\neg w))$
 using 1 5 6 Initprop by fastforce
 hence 8: $\vdash \text{bf } (f \longrightarrow \text{sfinit } (\text{init } w)) = (\neg (f \frown (\neg (\text{init } w))))$
 by (metis Initprop(2) int-eq int-simps(7))
 from 8 show ?thesis by (simp add: syields-d-def)
 qed

lemma *StateImpSYields*:

assumes $\vdash (\text{init } w) \wedge f \longrightarrow \text{sfm } (\text{init } w1)$

shows $\vdash (\text{init } w) \longrightarrow (f \text{ syields } (\text{init } w1))$

proof –

have 1: $\vdash (\text{init } w) \wedge f \longrightarrow \text{sfm } (\text{init } w1)$ **using** *assms* **by** *auto*

hence 2: $\vdash (\text{init } w) \longrightarrow (f \longrightarrow \text{sfm } (\text{init } w1))$ **by** *auto*

hence 3: $\vdash (\text{init } w) \longrightarrow \text{bf } (f \longrightarrow \text{sfm } (\text{init } w1))$

using *StateImpBfGen* **by** *auto*

have 4: $\vdash \text{bf } (f \longrightarrow \text{sfm } (\text{init } w1)) = f \text{ syields } (\text{init } w1)$

by (*rule BfImpSFmEqvSYieldsState*)

from 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *StateAndSYieldsImpSYields*:

assumes $\vdash (\text{init } w) \wedge f \longrightarrow f1$

shows $\vdash (\text{init } w) \wedge (f1 \text{ syields } g) \longrightarrow (f \text{ syields } g)$

proof –

have 1: $\vdash (\text{init } w) \wedge f \longrightarrow f1$ **using** *assms* **by** *auto*

hence 2: $\vdash (\text{init } w) \wedge (f \frown (\neg g)) \longrightarrow f1 \frown (\neg g)$ **by** (*rule StateAndSChopImpSChopRule*)

hence 3: $\vdash (\text{init } w) \wedge \neg (f1 \frown (\neg g)) \longrightarrow \neg (f \frown (\neg g))$ **by** *auto*

from 3 **show** *?thesis* **by** (*simp add: syields-d-def*)

qed

lemma *AndSYieldsA*:

$\vdash f \text{ syields } g \longrightarrow (f \wedge f1) \text{ syields } g$

proof –

have 1: $\vdash f \wedge f1 \longrightarrow f$ **by** *auto*

from 1 **show** *?thesis* **by** (*rule LeftSYieldsImpSYields*)

qed

lemma *AndSYieldsB*:

$\vdash f1 \text{ syields } g \longrightarrow (f \wedge f1) \text{ syields } g$

proof –

have 1: $\vdash f \wedge f1 \longrightarrow f1$ **by** *auto*

from 1 **show** *?thesis* **by** (*rule LeftSYieldsImpSYields*)

qed

lemma *RightSYieldsImpSYields*:

assumes $\vdash g \longrightarrow g1$

shows $\vdash (f \text{ syields } g) \longrightarrow (f \text{ syields } g1)$

proof –

have 1: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*

hence 2: $\vdash \neg g1 \longrightarrow \neg g$ **by** *auto*

hence 3: $\vdash f \frown (\neg g1) \longrightarrow f \frown (\neg g)$ **by** (*rule RightSChopImpSChop*)

hence 4: $\vdash \neg (f \frown (\neg g)) \longrightarrow \neg (f \frown (\neg g1))$ **by** *auto*

from 4 **show** *?thesis* **by** (*simp add: syields-d-def*)

qed

lemma *RightSYieldsEqvSYields*:

assumes $\vdash g = g1$
shows $\vdash (f \text{ syields } g) = (f \text{ syields } g1)$
proof –
have 1: $\vdash g = g1$ **using** *assms* **by** *auto*
hence 2: $\vdash (\neg g) = (\neg g1)$ **by** *auto*
hence 3: $\vdash f \frown (\neg g) = f \frown (\neg g1)$ **by** (*rule RightSChopEqvSChop*)
hence 4: $\vdash (\neg (f \frown (\neg g))) = (\neg (f \frown (\neg g1)))$ **by** *auto*
from 4 **show** *?thesis* **by** (*simp add: syields-d-def*)
qed

lemma *BoxImpSYields*:
 $\vdash \Box g \longrightarrow f \text{ syields } g$
proof –
have 1: $\vdash f \frown (\neg g) \longrightarrow \Diamond (\neg g)$ **by** (*rule SChopImpDiamond*)
hence 2: $\vdash \neg (\Diamond (\neg g)) \longrightarrow \neg (f \frown (\neg g))$ **by** *auto*
from 2 **show** *?thesis* **by** (*simp add: syields-d-def always-d-def*)
qed

lemma *BoxEqvTrueSYields*:
 $\vdash \Box f = \# \text{True syields } f$
proof –
have 1: $\vdash \# \text{True} \frown (\neg f) = \Diamond (\neg f)$ **by** (*rule TrueSChopEqvDiamond*)
hence 2: $\vdash (\neg (\# \text{True} \frown (\neg f))) = (\neg (\Diamond (\neg f)))$ **by** *auto*
have 3: $\vdash \Box f = (\neg (\Diamond (\neg f)))$ **by** (*simp add: always-d-def*)
have 4: $\vdash \Box f = (\neg (\# \text{True} \frown (\neg f)))$ **using** 2 3 **by** *fastforce*
from 4 **show** *?thesis* **by** (*simp add: syields-d-def*)
qed

lemma *SYieldsGen*:
assumes $\vdash g$
shows $\vdash f \text{ syields } g$
proof –
have 1: $\vdash g$ **using** *assms* **by** *auto*
hence 2: $\vdash \Box g$ **by** (*rule BoxGen*)
have 3: $\vdash \Box g \longrightarrow f \text{ syields } g$ **by** (*rule BoxImpSYields*)
from 2 3 **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *SYieldsAndSYieldsEqvSYieldsAnd*:
 $\vdash ((f \text{ syields } g) \wedge (f \text{ syields } g1)) = f \text{ syields } (g \wedge g1)$
proof –
have 1: $\vdash f \frown (\neg g \vee \neg g1) = ((f \frown (\neg g)) \vee (f \frown (\neg g1)))$ **by** (*rule SChopOrEqv*)
hence 2: $\vdash ((f \frown (\neg g)) \vee (f \frown (\neg g1))) = f \frown (\neg g \vee \neg g1)$ **by** *auto*
have 3: $\vdash (\neg g \vee \neg g1) = (\neg (g \wedge g1))$ **by** *auto*
hence 4: $\vdash f \frown (\neg g \vee \neg g1) = f \frown (\neg (g \wedge g1))$ **by** (*rule RightSChopEqvSChop*)
have 5: $\vdash (f \frown (\neg g)) \vee (f \frown (\neg g1)) = f \frown (\neg (g \wedge g1))$ **using** 2 4 **by** *fastforce*
hence 6: $\vdash (\neg (f \frown (\neg g)) \wedge \neg (f \frown (\neg g1))) = (\neg (f \frown (\neg (g \wedge g1))))$
by (*auto simp: schop-defs sum.case-eq-if*)
from 6 **show** *?thesis* **by** (*simp add: syields-d-def*)
qed

lemma *SYieldsAndSYieldsImpAndSYieldsAnd*:

$\vdash (f \text{ syields } g) \wedge (f1 \text{ syields } g1) \longrightarrow (f \wedge f1) \text{ syields } (g \wedge g1)$

proof —

have 1: $\vdash f \text{ syields } g \longrightarrow (f \wedge f1) \text{ syields } g$

by (rule *AndSYieldsA*)

have 2: $\vdash f1 \text{ syields } g1 \longrightarrow (f \wedge f1) \text{ syields } g1$

by (rule *AndSYieldsB*)

have 3: $\vdash ((f \wedge f1) \text{ syields } g \wedge (f \wedge f1) \text{ syields } g1) = (f \wedge f1) \text{ syields } (g \wedge g1)$

by (rule *SYieldsAndSYieldsEqvSYieldsAnd*)

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

qed

lemma *SYieldsSYieldsEqvSChopSYields*:

$\vdash f \text{ syields } (g \text{ syields } h) = (f \frown g) \text{ syields } h$

proof —

have 1: $\vdash f \frown (g \frown (\neg h)) = (f \frown g) \frown (\neg h)$ **by** (rule *SChopAssoc*)

hence 2: $\vdash f \frown (g \frown (\neg h)) = (f \frown g) \frown (\neg h)$ **by** auto

have 3: $\vdash g \frown (\neg h) = (\neg \neg (g \frown (\neg h)))$ **by** auto

hence 4: $\vdash f \frown (g \frown (\neg h)) = f \frown (\neg \neg (g \frown (\neg h)))$ **by** (rule *RightSChopEqvSChop*)

have 5: $\vdash f \frown (\neg \neg (g \frown (\neg h))) = (f \frown g) \frown (\neg h)$ **using** 2 4 **by** auto

hence 6: $\vdash f \frown (\neg (g \text{ syields } h)) = (f \frown g) \frown (\neg h)$ **by** (simp add: syields-d-def)

hence 7: $\vdash (\neg (f \frown (\neg (g \text{ syields } h)))) = (\neg ((f \frown g) \frown (\neg h)))$ **by** auto

from 7 **show** ?thesis **by** (simp add: syields-d-def)

qed

lemma *EmptyYields*:

$\vdash \text{empty} \text{ syields } f = f$

proof —

have 1: $\vdash \text{empty} \frown (\neg f) = (\neg f)$ **by** (rule *EmptySChop*)

hence 2: $\vdash (\neg (\text{empty} \frown (\neg f))) = f$ **by** auto

from 2 **show** ?thesis **by** (simp add: syields-d-def)

qed

lemma *NextSYields*:

$\vdash (\bigcirc f) \text{ syields } g = \text{wnext } (f \text{ syields } g)$

proof —

have 1: $\vdash (\bigcirc f) \frown (\neg g) = \bigcirc(f \frown (\neg g))$ **by** (rule *NextSChop*)

hence 2: $\vdash (\neg ((\bigcirc f) \frown (\neg g))) = (\neg (\bigcirc(f \frown (\neg g))))$ **by** auto

hence 3: $\vdash (\bigcirc f) \text{ syields } g = (\neg (\bigcirc(f \frown (\neg g))))$ **by** (simp add: syields-d-def)

have 4: $\vdash (\neg (\bigcirc(f \frown (\neg g)))) = \text{wnext } (\neg (f \frown (\neg g)))$ **by** (auto simp: wnext-d-def)

have 5: $\vdash (\bigcirc f) \text{ syields } g = \text{wnext } (\neg (f \frown (\neg g)))$ **using** 3 4 **by** fastforce

from 5 **show** ?thesis **by** (simp add: syields-d-def)

qed

lemma *SkipSChopEqvNext*:

$\vdash \text{skip} \frown f = \bigcirc f$

by (meson *NextSChopdef Prop11*)

lemma *SkipSYieldsEqvWeakNext*:

$\vdash \text{skip} \text{ syields } f = \text{wnext } f$
proof –
have 1: $\vdash \text{skip} \frown (\neg f) = \bigcirc(\neg f)$ **by** (rule SkipSChopEqvNext)
hence 2: $\vdash (\neg (\text{skip} \frown (\neg f))) = (\neg (\bigcirc(\neg f)))$ **by** auto
have 3: $\vdash (\neg (\bigcirc(\neg f))) = \text{wnext } f$ **by** (auto simp: wnext-d-def)
have 4: $\vdash (\neg (\text{skip} \frown (\neg f))) = \text{wnext } f$ **using** 2 3 **by** fastforce
from 4 **show** ?thesis **by** (simp add: syields-d-def)
qed

lemma NextImpSkipSYields:
 $\vdash \bigcirc f \longrightarrow \text{skip} \text{ syields } f$
proof –
have 1: $\vdash \bigcirc f \longrightarrow \text{wnext } f$ **using** WnextEqvEmptyOrNext **by** fastforce
have 2: $\vdash \text{skip} \text{ syields } f = \text{wnext } f$ **by** (rule SkipSYieldsEqvWeakNext)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma MoreEqvSkipSChopTrue:
 $\vdash \text{more} = \text{skip} \frown \# \text{True}$
proof –
have 1: $\vdash \text{skip} \frown \# \text{True} = \bigcirc \# \text{True}$ **by** (rule SkipSChopEqvNext)
hence 2: $\vdash \bigcirc \# \text{True} = \text{skip} \frown \# \text{True}$ **by** auto
from 2 **show** ?thesis **by** (simp add: more-d-def)
qed

lemma MoreSChopImpMore:
 $\vdash \text{more} \frown f \longrightarrow \text{more}$
proof –
have 1: $\vdash (\bigcirc \# \text{True}) \frown f = \bigcirc (\# \text{True} \frown f)$ **by** (rule NextSChop)
have 2: $\vdash \bigcirc (\# \text{True} \frown f) \longrightarrow \text{more}$ **by** (auto simp: more-defs next-defs sum.case-eq-if)
have 3: $\vdash (\bigcirc \# \text{True} \frown f) \longrightarrow \text{more}$ **using** 1 2 **by** fastforce
from 3 **show** ?thesis **by** (metis more-d-def)
qed

lemma MoreSChopImpFmore:
 $\vdash \text{more} \frown (f \wedge \text{finite}) \longrightarrow \text{fmore}$
proof –
have 1: $\vdash \text{more} \frown (f \wedge \text{finite}) = \bigcirc (\# \text{True} \frown (f \wedge \text{finite}))$
by (simp add: NextSChop more-d-def)
have 2: $\vdash \bigcirc (\# \text{True} \frown (f \wedge \text{finite})) \longrightarrow \text{fmore}$
by (auto simp: fmore-defs schop-defs finite-defs more-defs next-defs sum.case-eq-if)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma SChopMoreImpMore:
 $\vdash f \frown \text{more} \longrightarrow \text{more}$
proof –
have 1: $\vdash f \frown \text{more} \longrightarrow \Diamond \text{more}$ **by** (rule SChopImpDiamond)
have 2: $\vdash \Diamond \text{more} \longrightarrow \text{more}$ **by** (auto simp: more-defs sometimes-defs sum.case-eq-if)

from 1 2 show ?thesis by fastforce
qed

lemma MoreSChopEqvNextDiamond:

$\vdash \text{more} \frown f = \bigcirc(\Diamond f)$

proof —

have 1: $\vdash \text{more} \frown f = (\bigcirc \# \text{True}) \frown f$

by (simp add: Valid-def schop-defs more-defs next-defs finite-defs sum.case-eq-if)

have 2: $\vdash (\bigcirc \# \text{True}) \frown f = \bigcirc(\# \text{True} \frown f)$ **by** (rule NextSChop)

have 3: $\vdash \text{more} \frown f = \bigcirc(\# \text{True} \frown f)$ **using 1 2 by fastforce**

from 3 show ?thesis

by (metis TrueSChopEqvDiamond inteq-reflection)

qed

lemma WeakNextBoxImpMoreSYields:

$\vdash \text{more} \text{ syields } f = \text{wnext}(\Box f)$

proof —

have 1: $\vdash \text{more} \frown (\neg f) = \bigcirc(\Diamond (\neg f))$ **by** (rule MoreSChopEqvNextDiamond)

have 2: $\vdash \bigcirc(\Diamond (\neg f)) = \bigcirc(\neg(\Box f))$ **by** (auto simp: always-d-def)

have 3: $\vdash \bigcirc(\neg(\Box f)) = (\neg(\text{wnext}(\Box f)))$ **by** (auto simp: wnext-d-def)

have 4: $\vdash \text{more} \frown (\neg f) = (\neg(\text{more} \text{ syields } f))$ **by** (simp add: syields-d-def)

from 1 2 3 4 show ?thesis by fastforce

qed

lemma NotEqvSYieldsMore:

$\vdash \text{finite} \longrightarrow (\neg f) = f \text{ syields more}$

proof —

have 1: $\vdash \text{finite} \longrightarrow f \frown \text{empty} = f$ **by** (rule SChopEmpty)

hence 2: $\vdash \text{finite} \longrightarrow (\neg(f \frown \text{empty})) = (\neg f)$ **by auto**

have 3: $\vdash \text{empty} = (\neg \text{more})$ **by** (auto simp: empty-d-def)

hence 4: $\vdash f \frown \text{empty} = f \frown (\neg \text{more})$ **by** (rule RightSChopEqvSChop)

hence 5: $\vdash (\neg(f \frown \text{empty})) = (\neg(f \frown (\neg \text{more})))$ **by auto**

have 6: $\vdash \text{finite} \longrightarrow (\neg f) = (\neg(f \frown (\neg \text{more})))$ **using 2 5 by fastforce**

from 6 show ?thesis by (metis syields-d-def)

qed

lemma LeftSChopImpMoreRule:

assumes $\vdash f \longrightarrow \text{more}$

shows $\vdash f \frown g \longrightarrow \text{more}$

proof —

have 1: $\vdash f \longrightarrow \text{more}$ **using assms by auto**

hence 2: $\vdash f \frown g \longrightarrow \text{more} \frown g$ **by** (rule LeftSChopImpSChop)

have 3: $\vdash \text{more} \frown g \longrightarrow \text{more}$ **by** (rule MoreSChopImpMore)

from 2 3 show ?thesis using lift-imp-trans by blast

qed

lemma LeftSChopImpFMoreRule:

assumes $\vdash f \longrightarrow \text{fmore}$

shows $\vdash f \frown (g \wedge \text{finite}) \longrightarrow \text{fmore}$

proof —

have 1: $\vdash f \longrightarrow fmore$ **using** *assms* **by** *auto*
hence 2: $\vdash f \frown (g \wedge finite) \longrightarrow more \frown (g \wedge finite)$
by (*metis FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite*
FmoreEqvSkipChopFinite LeftSCHopImpSCHop Prop12 inteq-reflection)
have 3: $\vdash more \frown (g \wedge finite) \longrightarrow fmore$ **using** *MoreSCHopImpFmore* **by** *fastforce*
from 2 3 **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *RightSCHopImpMoreRule*:

assumes $\vdash g \longrightarrow more$

shows $\vdash f \frown g \longrightarrow more$

proof –

have 1: $\vdash g \longrightarrow more$ **using** *assms* **by** *auto*

hence 2: $\vdash f \frown g \longrightarrow f \frown more$ **by** (*rule RightSCHopImpSCHop*)

have 3: $\vdash f \frown more \longrightarrow more$ **by** (*rule SCHopMoreImpMore*)

from 2 3 **show** *?thesis* **using** *lift-imp-trans* **by** *blast*

qed

lemma *NotDfEqvBfNot*:

$\vdash (\neg (df\ f)) = bf\ (\neg\ f)$

proof –

have 1: $\vdash f = (\neg \neg\ f)$ **by** *auto*

hence 2: $\vdash df\ f = df\ (\neg \neg\ f)$ **by** (*rule DfEqvDf*)

hence 3: $\vdash (\neg (df\ f)) = (\neg (df\ (\neg \neg\ f)))$ **by** *auto*

from 3 **show** *?thesis* **by** (*simp add: bf-d-def*)

qed

lemma *SCHopImpDf*:

$\vdash f \frown g \longrightarrow df\ f$

proof –

have 1: $\vdash g \longrightarrow \#True$ **by** *auto*

hence 2: $\vdash f \frown g \longrightarrow f \frown \#True$ **by** (*rule RightSCHopImpSCHop*)

from 2 **show** *?thesis* **by** (*simp add: df-d-def*)

qed

lemma *TrueEqvTrueSCHopTrue*:

$\vdash \#True = \#True \frown \#True$

proof –

have 1: $\vdash \#True \frown \#True \longrightarrow \#True$ **by** *auto*

have 2: $\vdash \#True \longrightarrow \#True \frown \#True$

by (*metis DfState Initprop(4) df-d-def int-eq-true int-iffD1 inteq-reflection*)

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

qed

lemma *DfEqvDfDf*:

$\vdash df\ f = df\ (df\ f)$

proof –

have 1: $\vdash \#True = \#True \frown \#True$ **by** (*rule TrueEqvTrueSCHopTrue*)

hence 2: $\vdash f \frown \#True = f \frown (\#True \frown \#True)$ **by** (rule RightSChopEqvSChop)
have 3: $\vdash f \frown (\#True \frown \#True) = (f \frown \#True) \frown \#True$ **by** (rule SChopAssoc)
have 4: $\vdash f \frown \#True = (f \frown \#True) \frown \#True$ **using** 2 3 **by** fastforce
from 4 **show** ?thesis **by** (metis df-d-def)
qed

lemma BfEqvBfBf:

$\vdash bf\ f = bf(bf\ f)$

proof –

have 1: $\vdash df(\neg f) = df(df(\neg f))$ **by** (rule DfEqvDfDf)
have 2: $\vdash df(\neg f) = (\neg(bf\ f))$ **by** (rule DfNotEqvNotBf)
hence 3: $\vdash df(df(\neg f)) = df(\neg(bf\ f))$ **by** (rule DfEqvDf)
have 4: $\vdash df(\neg f) = df(\neg(bf\ f))$ **using** 1 3 **by** fastforce
hence 5: $\vdash (\neg(df(\neg f))) = (\neg(df(\neg(bf\ f))))$ **by** fastforce
from 5 **show** ?thesis **by** (metis bf-d-def)
qed

lemma BfImpBfBf:

$\vdash bf\ f \longrightarrow bf(bf\ f)$

proof –

have 1: $\vdash bf(bf\ f) = bf\ f$ **using** BfEqvBfBf **by** fastforce
from 1 **show** ?thesis **by** (simp add: int-iffD2)
qed

lemma DfOrEqv:

$\vdash df(f \vee g) = (df\ f \vee df\ g)$

proof –

have 1: $\vdash (f \vee g) \frown \#True = (f \frown \#True \vee g \frown \#True)$ **by** (rule OrSChopEqv)
from 1 **show** ?thesis **by** (simp add: df-d-def)
qed

lemma DfAndA:

$\vdash df(f \wedge g) \longrightarrow df\ f$

proof –

have 1: $\vdash (f \wedge g) \frown \#True \longrightarrow f \frown \#True$ **by** (rule AndSChopA)
from 1 **show** ?thesis **by** (simp add: df-d-def)
qed

lemma DfAndB:

$\vdash df(f \wedge g) \longrightarrow df\ g$

proof –

have 1: $\vdash (f \wedge g) \frown \#True \longrightarrow g \frown \#True$ **by** (rule AndSChopB)
from 1 **show** ?thesis **by** (simp add: df-d-def)
qed

lemma DfAndImpAnd:

$\vdash df (f \wedge g) \longrightarrow df f \wedge df g$

proof –

have 1: $\vdash df (f \wedge g) \longrightarrow df f$ **by** (rule DfAndA)

have 2: $\vdash df (f \wedge g) \longrightarrow df g$ **by** (rule DfAndB)

from 1 2 **show** ?thesis **by** fastforce

qed

lemma DfSkipEqvMore:

$\vdash df skip = more$

proof –

have 1: $\vdash skip \frown \#True = \circ \#True$ **by** (rule SkipSChopEqvNext)

have 2: $\vdash \circ \#True = more$ **by** (auto simp: more-d-def)

have 3: $\vdash skip \frown \#True = more$ **using** 1 2 **by** fastforce

from 3 **show** ?thesis **by** (simp add: df-d-def)

qed

lemma DfMoreEqvMore:

$\vdash df more = more$

proof –

have 1: $\vdash df (\circ \#True) = \circ(df \#True)$ **by** (rule DfNext)

have 2: $\vdash \circ(df \#True) \longrightarrow more$ **by** (auto simp: next-defs di-defs more-defs sum.case-eq-if)

have 3: $\vdash df (\circ \#True) \longrightarrow more$ **using** 1 2 **by** fastforce

hence 4: $\vdash df more \longrightarrow more$ **by** (simp add: more-d-def)

have 5: $\vdash more \longrightarrow df more$

by (metis 1 4 TrueEqvTrueSChopTrue df-d-def inteq-reflection more-d-def)

from 4 5 **show** ?thesis **by** fastforce

qed

lemma DfIfEqvRule:

assumes $\vdash f = if_i (init w) then g else h$

shows $\vdash df f = if_i (init w) then (df g) else (df h)$

proof –

have 1: $\vdash f = if_i (init w) then g else h$ **using** assms **by** auto

hence 2: $\vdash f \frown \#True = if_i (init w) then (g \frown \#True) else (h \frown \#True)$

by (rule IfSChopEqvRule)

from 2 **show** ?thesis **by** (simp add: df-d-def)

qed

lemma SDaNotEqvNotSBa:

$\vdash sda (\neg f) = (\neg (sba f))$

proof –

have 1: $\vdash sba f = (\neg (sda (\neg f)))$ **by** (simp add: sba-d-def)

from 1 **show** ?thesis **by** fastforce

qed

lemma SDaEqvSDa:

assumes $\vdash f = g$

shows $\vdash sda\ f = sda\ g$
using *assms* **using** *int-eq* **by** *force*

lemma *SDaEqvNotSBaNot*:

$\vdash sda\ f = (\neg (sba\ (\neg f)))$

proof $-$

have 1: $\vdash sba\ (\neg f) = (\neg (sda\ (\neg \neg f)))$ **by** (*simp add: sba-d-def*)

hence 2: $\vdash sda\ (\neg \neg f) = (\neg (sba\ (\neg f)))$ **by** *fastforce*

have 3: $\vdash f = (\neg \neg f)$ **by** *simp*

hence 4: $\vdash sda\ f = sda\ (\neg \neg f)$ **by** (*rule SDaEqvSDa*)

from 2 4 **show** *?thesis* **by** *simp*

qed

lemma *SBaElim*:

$\vdash sba\ f \wedge finite \longrightarrow f$

proof $-$

have 1: $\vdash sba\ f = \Box (bf\ f)$ **by** (*rule SBaEqvBtBf*)

have 2: $\vdash bf\ f \wedge finite \longrightarrow f$ **by** (*rule BfElim*)

hence 3: $\vdash \Box (bf\ f \wedge finite \longrightarrow f)$ **by** (*rule BoxGen*)

have 4: $\vdash \Box (bf\ f \wedge finite \longrightarrow f) \longrightarrow \Box (bf\ f \wedge finite) \longrightarrow \Box f$ **by** (*rule BoxImpDist*)

have 5: $\vdash \Box (bf\ f \wedge finite) \longrightarrow \Box f$ **using** 3 4 *MP* **by** *fastforce*

have 6: $\vdash \Box (bf\ f \wedge finite) = (\Box (bf\ f) \wedge finite)$

by (*metis (no-types, lifting) BoxEqvFiniteYields FiniteChopInfEqvInf NotChopEqvYieldsNot YieldsAndYieldsEqvYieldsAnd finite-d-def inteq-reflection*)

have 7: $\vdash \Box f \longrightarrow f$ **by** (*rule BoxElim*)

from 1 5 6 7 **show** *?thesis* **using** *SBaImpBt lift-imp-trans* **by** *metis*

qed

lemma *SDaIntro*:

$\vdash f \wedge finite \longrightarrow sda\ f$

proof $-$

have 1: $\vdash sba\ (\neg f) \wedge finite \longrightarrow (\neg f)$ **by** (*rule SBaElim*)

hence 2: $\vdash \neg \neg f \longrightarrow \neg (sba\ (\neg f) \wedge finite)$ **by** *fastforce*

have 3: $\vdash f = (\neg \neg f)$ **by** *simp*

have 4: $\vdash sda\ f = (\neg (sba\ (\neg f)))$ **by** (*rule SDaEqvNotSBaNot*)

from 2 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *SBaGen*:

assumes $\vdash f$

shows $\vdash sba\ f$

proof $-$

have 1: $\vdash f$ **using** *assms* **by** *auto*

hence 2: $\vdash \Box f$ **by** (*rule BoxGen*)

hence 3: $\vdash bf\ (\Box f)$ **by** (*rule BfGen*)

have 4: $\vdash sba\ f = bf\ (\Box f)$ **by** (*rule SBaEqvBfBt*)

from 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *SBaImpDist*:

$\vdash sba (f \longrightarrow g) \longrightarrow sba f \longrightarrow sba g$
proof –
have 1: $\vdash bf (f \longrightarrow g) \longrightarrow (bf f \longrightarrow bf g)$ **by** (rule BfImpDist)
hence 2: $\vdash \Box (bf (f \longrightarrow g) \longrightarrow (bf f \longrightarrow bf g))$ **by** (rule BoxGen)
have 3: $\vdash \Box (bf (f \longrightarrow g) \longrightarrow (bf f \longrightarrow bf g))$
 \longrightarrow
 $(\Box (bf (f \longrightarrow g)) \longrightarrow (\Box (bf f) \longrightarrow \Box (bf g)))$
by (meson 2 BoxImpDist MP lift-imp-trans Prop01 Prop05 Prop09)
have 4: $\vdash \Box (bf (f \longrightarrow g)) \longrightarrow (\Box (bf f) \longrightarrow \Box (bf g))$ **using** 2 3 MP **by** fastforce
have 5: $\vdash sba (f \longrightarrow g) = \Box (bf (f \longrightarrow g))$ **by** (rule SBaEqvBtBf)
have 6: $\vdash sba f = \Box (bf f)$ **by** (rule SBaEqvBtBf)
have 7: $\vdash sba g = \Box (bf g)$ **by** (rule SBaEqvBtBf)
from 4 5 6 7 **show** ?thesis **by** fastforce
qed

lemma SBaAndEqv:

$\vdash sba (f \wedge g) = (sba f \wedge sba g)$
proof –
have 1: $\vdash sba (f \wedge g) = \Box (bf (f \wedge g))$
by (rule SBaEqvBtBf)
have 2: $\vdash bf (f \wedge g) = (bf f \wedge bf g)$
by (auto simp: bf-defs sum.case-eq-if)
hence 3: $\vdash \Box (bf (f \wedge g)) = \Box (bf f \wedge bf g)$
using BoxEqvBox **by** blast
have 4: $\vdash \Box (bf f \wedge bf g) = (\Box (bf f) \wedge \Box (bf g))$
by (metis 2 BoxAndBoxEqvBoxRule inteq-reflection)
have 5: $\vdash sba f = \Box (bf f)$
by (rule SBaEqvBtBf)
have 6: $\vdash sba g = \Box (bf g)$
by (rule SBaEqvBtBf)
from 1 3 4 5 6 **show** ?thesis **by** fastforce
qed

lemma SBaImpSBaEqvSBa:

$\vdash sba (f = g) \longrightarrow (sba f = sba g)$
proof –
have 1: $\vdash sba (f \longrightarrow g) \longrightarrow sba f \longrightarrow sba g$ **by** (rule SBaImpDist)
have 2: $\vdash sba (g \longrightarrow f) \longrightarrow sba g \longrightarrow sba f$ **by** (rule SBaImpDist)
have 3: $\vdash (f = g) = ((f \longrightarrow g) \wedge (g \longrightarrow f))$
by auto
hence 31: $\vdash sba (f = g) = sba ((f \longrightarrow g) \wedge (g \longrightarrow f))$
using inteq-reflection **by** force
have 4: $\vdash sba ((f \longrightarrow g) \wedge (g \longrightarrow f)) = (sba ((f \longrightarrow g)) \wedge sba ((g \longrightarrow f)))$
by (rule SBaAndEqv)
have 5: $\vdash ((sba f \longrightarrow sba g) \wedge (sba g \longrightarrow sba f)) = (sba f = sba g)$ **by** auto
from 1 2 31 4 5 **show** ?thesis **by** fastforce
qed

lemma SBaImpSBa:

assumes $\vdash f \longrightarrow g$

shows $\vdash sba\ f \longrightarrow sba\ g$
using *SBaGen SBaImpDist MP assms by metis*

lemma *SBaEqvSBa*:
assumes $\vdash f = g$
shows $\vdash sba\ f = sba\ g$
using *SBaGen SBaImpSBaEqvSBa MP assms by metis*

lemma *SDaImpSDa*:
assumes $\vdash f \longrightarrow g$
shows $\vdash sda\ f \longrightarrow sda\ g$
using *assms by (metis SDaEqvDtDf DfAndB DiamondImpDiamond inteq-reflection Prop10)*

lemma *SDaEqvSDaSDa*:
 $\vdash sda\ f = sda\ (sda\ f)$
proof –
have 1: $\vdash sda\ f = \Diamond (df\ f)$
by (*rule SDaEqvDtDf*)
have 2: $\vdash df\ f = (df\ (df\ f))$
by (*rule DfEqvDfDf*)
hence 3: $\vdash \Diamond (df\ f) = \Diamond (df\ (df\ f))$
by (*rule DiamondEqvDiamond*)
have 4: $\vdash \Diamond (df\ f) = \Diamond (\Diamond (df\ (df\ f)))$
using *DiamondEqvDiamondDiamond DfEqvDfDf using 3 by fastforce*
have 5: $\vdash \Diamond (df\ (df\ f)) = df\ (\Diamond (df\ f))$
by (*rule DtDfEqvDfDt*)
hence 6: $\vdash \Diamond (\Diamond (df\ (df\ f))) = \Diamond (df\ (\Diamond (df\ f)))$
by (*rule DiamondEqvDiamond*)
have 7: $\vdash sda\ f = \Diamond (df\ (\Diamond (df\ f)))$
using 1 3 4 6 **by** *fastforce*
have 8: $\vdash sda\ (\Diamond (df\ f)) = \Diamond (df\ (\Diamond (df\ f)))$
by (*rule SDaEqvDtDf*)
have 9: $\vdash sda\ (sda\ f) = sda\ (\Diamond (df\ f))$
using 1 **by** (*rule SDaEqvSDa*)
from 7 8 9 **show** *?thesis* **by** *fastforce*
qed

lemma *SBaEqvSBaSBa*:
 $\vdash sba\ f = sba\ (sba\ f)$
proof –
have 1: $\vdash sda\ (\neg f) = sda\ (sda\ (\neg f))$ **by** (*rule SDaEqvSDaSDa*)
have 2: $\vdash sda\ (sda\ (\neg f)) = (\neg (sba\ (\neg (sda\ (\neg f)))))$ **by** (*rule SDaEqvNotSBaNot*)
have 3: $\vdash (\neg (sda\ (sda\ (\neg f)))) = sba\ (\neg (sda\ (\neg f)))$ **by** (*auto simp: sba-d-def*)
have 4: $\vdash (\neg (sda\ (\neg f))) = sba\ (\neg (sda\ (\neg f)))$ **using** 1 2 3 **by** *fastforce*
from 4 **show** *?thesis* **by** (*metis sba-d-def*)
qed

lemma *SBaLeftSChopImpSChop*:

$\vdash \text{ sba } (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$
proof –
have 1: $\vdash \text{ sba } (f \longrightarrow f1) \longrightarrow \text{ bf } (f \longrightarrow f1)$ **by** (rule SBalmpBf)
have 2: $\vdash \text{ bf } (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (rule BfSCHopImpSCHop)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma BaLeftSCHopImpSCHop:
 $\vdash \text{ ba } (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$
proof –
have 1: $\vdash \text{ ba } (f \longrightarrow f1) \longrightarrow \text{ bf } (f \longrightarrow f1)$ **by** (rule BalmpBf)
have 2: $\vdash \text{ bf } (f \longrightarrow f1) \longrightarrow f \frown g \longrightarrow f1 \frown g$ **by** (rule BfSCHopImpSCHop)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma SBaRightSCHopImpSCHop:
 $\vdash \text{ sba } (g \longrightarrow g1) \wedge \text{ finite} \longrightarrow f \frown g \longrightarrow f \frown g1$
proof –
have 1: $\vdash \text{ sba } (g \longrightarrow g1) \wedge \text{ finite} \longrightarrow \Box(g \longrightarrow g1)$ **by** (rule SBalmpBt)
have 2: $\vdash \Box(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **by** (rule BoxSCHopImpSCHop)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma BaRightSCHopImpSCHop:
 $\vdash \text{ ba } (g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$
proof –
have 1: $\vdash \text{ ba } (g \longrightarrow g1) \longrightarrow \Box(g \longrightarrow g1)$ **by** (rule BalmpBt)
have 2: $\vdash \Box(g \longrightarrow g1) \longrightarrow f \frown g \longrightarrow f \frown g1$ **by** (rule BoxSCHopImpSCHop)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma SCHopAndSBalmpImport:
 $\vdash (f \frown f1) \wedge \text{ sba } g \wedge \text{ finite} \longrightarrow (f \wedge g) \frown (f1 \wedge g)$
proof –
have 1: $\vdash \text{ sba } g \wedge \text{ finite} \wedge (f \frown f1) \longrightarrow (g \wedge f) \frown (g \wedge f1)$ **by** (rule SBaAndSCHopImpImport)
have 2: $\vdash (g \wedge f) \frown (g \wedge f1) = (f \wedge g) \frown (f1 \wedge g)$ **by** (rule AndSCHopAndCommute)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma SCHopAndBalmpImport:
 $\vdash (f \frown f1) \wedge \text{ ba } g \longrightarrow (f \wedge g) \frown (f1 \wedge g)$
proof –
have 1: $\vdash \text{ ba } g \wedge (f \frown f1) \longrightarrow (g \wedge f) \frown (g \wedge f1)$ **by** (rule BaAndSCHopImpImport)
have 2: $\vdash (g \wedge f) \frown (g \wedge f1) = (f \wedge g) \frown (f1 \wedge g)$ **by** (rule AndSCHopAndCommute)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma BaAndSCHopImportA:
 $\vdash \text{ ba } f \wedge g \frown g1 \longrightarrow (f \wedge g) \frown g1$
by (meson BaAndSCHopImport SCHopAndB lift-imp-trans)

lemma *BaAndSChopImportB*:
 $\vdash ba\ f \wedge g \frown g1 \longrightarrow (f \wedge g) \frown (ba\ f \wedge g1)$
proof –
have 1: $\vdash ba\ f = ba\ (ba\ f)$
by (*simp add: BaEqvBaBa*)
have 2: $\vdash ba\ (ba\ f) \wedge g \frown g1 \longrightarrow g \frown (ba\ f \wedge g1)$
by (*metis AndSChopB BaAndSChopImport lift-imp-trans*)
have 3: $\vdash ba\ f \wedge g \frown (ba\ f \wedge g1) \longrightarrow (f \wedge g) \frown (ba\ f \wedge g1)$
by (*simp add: BaAndSChopImportA*)
from 1 2 3 **show** ?thesis **by** fastforce
qed

lemma *SBalmpSBalmpSBaAnd*:
 $\vdash sba\ h \longrightarrow sba(g \longrightarrow sba\ h \wedge g)$
proof –
have 1: $\vdash sba\ h \longrightarrow (g \longrightarrow sba\ h \wedge g)$ **by** fastforce
hence 2: $\vdash sba(sba\ h) \longrightarrow sba(g \longrightarrow sba\ h \wedge g)$ **by** (*rule SBalmpSBa*)
have 3: $\vdash sba\ h = sba(sba\ h)$ **by** (*rule SBaEqvSBaSBa*)
from 2 3 **show** ?thesis **by** fastforce
qed

lemma *SBaSChopImpSChopSBa*:
 $\vdash sba\ f \wedge finite \longrightarrow g \frown g1 \longrightarrow g \frown ((sba\ f) \wedge g1)$
proof –
have 1: $\vdash sba\ f \longrightarrow sba(g1 \longrightarrow (sba\ f) \wedge g1)$ **by** (*rule SBalmpSBalmpSBaAnd*)
have 2: $\vdash sba(g1 \longrightarrow sba\ f \wedge g1) \wedge finite \longrightarrow g \frown g1 \longrightarrow g \frown (sba\ f \wedge g1)$
by (*rule SBaRightSChopImpSChop*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *BaSChopImpSChopBa*:
 $\vdash ba\ f \longrightarrow g \frown g1 \longrightarrow g \frown ((ba\ f) \wedge g1)$
proof –
have 1: $\vdash ba\ f \longrightarrow ba(g1 \longrightarrow (ba\ f) \wedge g1)$ **by** (*rule BalmpBalmpBaAnd*)
have 2: $\vdash ba(g1 \longrightarrow ba\ f \wedge g1) \longrightarrow g \frown g1 \longrightarrow g \frown (ba\ f \wedge g1)$
by (*rule BaRightSChopImpSChop*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *DfNotSBalmpNotSBa*:
 $\vdash df(\neg(sba\ f)) \longrightarrow \neg(sba\ f)$
proof –
have 1: $\vdash sba\ f = sba(sba\ f)$ **by** (*rule SBaEqvSBaSBa*)
have 2: $\vdash sba(sba\ f) \longrightarrow bf(sba\ f)$ **by** (*rule SBalmpBf*)
have 3: $\vdash sba\ f \longrightarrow bf(sba\ f)$ **using** 1 2 **by** fastforce
hence 4: $\vdash sba\ f \longrightarrow \neg(df(\neg(sba\ f)))$ **by** (*simp add: bf-d-def*)
from 4 **show** ?thesis **by** fastforce

qed

lemma *DfNotBaImpNotBa*:

$\vdash df(\neg(ba\ f)) \longrightarrow \neg(ba\ f)$

proof —

have 1: $\vdash ba\ f = ba(ba\ f)$ **by** (rule *BaEqvBaBa*)

have 2: $\vdash ba(ba\ f) \longrightarrow bf(ba\ f)$ **by** (rule *BaImpBf*)

have 3: $\vdash ba\ f \longrightarrow bf(ba\ f)$ **using** 1 2 **by** *fastforce*

hence 4: $\vdash ba\ f \longrightarrow \neg(df(\neg(ba\ f)))$ **by** (simp add: *bf-d-def*)

from 4 **show** ?thesis **by** *fastforce*

qed

lemma *NotSBaSChopImpNotSBa*:

$\vdash (\neg(sba\ f)) \frown g \longrightarrow \neg(sba\ f)$

proof —

have 1: $\vdash (\neg(sba\ f)) \frown g \longrightarrow df(\neg(sba\ f))$ **by** (rule *SChopImpDf*)

have 2: $\vdash df(\neg(sba\ f)) \longrightarrow \neg(sba\ f)$ **by** (rule *DfNotSBaImpNotSBa*)

from 1 2 **show** ?thesis **using** *lift-imp-trans* **by** *blast*

qed

lemma *NotBaSChopImpNotSBa*:

$\vdash (\neg(ba\ f)) \frown g \longrightarrow \neg(ba\ f)$

proof —

have 1: $\vdash (\neg(ba\ f)) \frown g \longrightarrow df(\neg(ba\ f))$ **by** (rule *SChopImpDf*)

have 2: $\vdash df(\neg(ba\ f)) \longrightarrow \neg(ba\ f)$ **by** (rule *DfNotBaImpNotBa*)

from 1 2 **show** ?thesis **using** *lift-imp-trans* **by** *blast*

qed

lemma *DiamondSFinImpSFin*:

$\vdash \Diamond(sfin\ f) \longrightarrow sfin\ f$

proof —

have 1: $\vdash sfin\ f = \#True \frown (f \wedge empty)$

by (rule *SFinEqvTrueSChopAndEmpty*)

hence 2: $\vdash \Diamond(sfin\ f) = \#True \frown (\#True \frown (f \wedge empty))$

using *DiamondSChopdef* *inteq-reflection* **by** *force*

have 3: $\vdash \#True \frown (\#True \frown (f \wedge empty)) = (\#True \frown \#True) \frown (f \wedge empty)$

by (rule *SChopAssoc*)

have 4: $\vdash (\#True \frown \#True) \frown (f \wedge empty) \longrightarrow \#True \frown (f \wedge empty)$

using 1 2 3

by (metis *SChopImpDiamond* *TrueEqvTrueSChopTrue* *inteq-reflection*)

from 1 2 3 4 **show** ?thesis **by** *fastforce*

qed

lemma *SChopSFinImpSFin*:

$\vdash f \frown sfin\ (init\ w) \longrightarrow sfin\ (init\ w)$

proof —

have 1: $\vdash f \frown sfin\ (init\ w) \longrightarrow \Diamond(sfin\ (init\ w))$ **by** (rule *SChopImpDiamond*)

have 2: $\vdash \Diamond(sfin\ (init\ w)) \longrightarrow sfin\ (init\ w)$ **by** (rule *DiamondSFinImpSFin*)

from 1 2 **show** ?thesis **using** *lift-imp-trans* **by** *blast*

qed

lemma *SFinImpSYieldsSFin*:

$\vdash \text{sf}in \text{ (init } w) \longrightarrow f \text{ syields (sf}in \text{ (init } w))$

proof –

have 1: $\vdash f \frown (\text{sf}in \text{ (init } (\neg w))) \longrightarrow (\text{sf}in \text{ (init } (\neg w)))$

by (*simp add: SChopSFinImpSFin*)

have 2: $\vdash \text{finite} \longrightarrow (\neg (\text{sf}in \text{ (init } w))) = (\text{sf}in \text{ (init } (\neg w)))$

using *SFinNotStateEqvNotSFinState* **by** *fastforce*

hence 3: $\vdash \text{finite} \longrightarrow f \frown (\neg (\text{sf}in \text{ (init } w))) = f \frown (\text{sf}in \text{ (init } (\neg w)))$

using *FiniteRightSChopEqvSChop* **by** *blast*

have 4: $\vdash f \frown (\neg (\text{sf}in \text{ (init } w))) \wedge \text{finite} \longrightarrow (\neg (\text{sf}in \text{ (init } w)))$

using 1 2 3 **by** *fastforce*

hence 5: $\vdash \text{sf}in \text{ (init } w) \longrightarrow \neg (f \frown (\neg (\text{sf}in \text{ (init } w))))$

by (*metis SChopImpDiamond SFinImpBox always-d-def int-simps(32) inteq-reflection lift-imp-trans*)

from 5 **show** ?thesis

by (*simp add: syields-d-def*)

qed

lemma *SChopAndSFin*:

$\vdash ((f \frown g) \wedge (\text{sf}in \text{ (init } w))) = f \frown (g \wedge (\text{sf}in \text{ (init } w)))$

proof –

have 1: $\vdash \text{sf}in \text{ (init } w) \longrightarrow f \text{ syields (sf}in \text{ (init } w))$

by (*rule SFinImpSYieldsSFin*)

have 2: $\vdash (f \frown g) \wedge (\text{sf}in \text{ (init } w)) \longrightarrow (f \frown g) \wedge f \text{ syields (sf}in \text{ (init } w))$

using 1 **by** *fastforce*

have 3: $\vdash f \frown g \wedge f \text{ syields (sf}in \text{ (init } w)) \longrightarrow$

$f \frown (g \wedge (\text{sf}in \text{ (init } w)))$

using *SChopAndSYieldsImp* **by** *blast*

have 4: $\vdash (f \frown g) \wedge (\text{sf}in \text{ (init } w)) \longrightarrow f \frown (g \wedge \text{sf}in \text{ (init } w))$

using 2 3 **by** (*metis (mono-tags, lifting) lift-imp-trans*)

from 4 **show** ?thesis

by (*simp add: Prop12 SChopAndA SChopSFinExportA int-iff1*)

qed

lemma *SChopAndNotSFin*:

$\vdash (f \frown g \wedge \neg (\text{sf}in \text{ (init } w)) \wedge \text{finite}) = f \frown (g \wedge \neg (\text{sf}in \text{ (init } w)) \wedge \text{finite})$

proof –

have 1: $\vdash (f \frown g \wedge \text{sf}in \text{ (init } (\neg w))) = f \frown (g \wedge \text{sf}in \text{ (init } (\neg w)))$

by (*rule SChopAndSFin*)

have 2: $\vdash (\text{sf}in \text{ (init } (\neg w)) \wedge \text{finite}) = (\neg (\text{sf}in \text{ (init } w))) \wedge \text{finite}$

using *SFinNotStateEqvNotSFinState* **by** *fastforce*

hence 3: $\vdash (g \wedge \text{sf}in \text{ (init } (\neg w))) = (g \wedge \neg (\text{sf}in \text{ (init } w)) \wedge \text{finite})$

using *DiamondEmptyEqvFinite SChopAndB SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond*

by *fastforce*

hence 4: $\vdash f \frown (g \wedge \text{sf}in \text{ (init } (\neg w))) = f \frown (g \wedge \neg (\text{sf}in \text{ (init } w)) \wedge \text{finite})$

using *RightSChopEqvSChop* **by** *blast*

from 1 2 4 **show** ?thesis

using *DiamondEmptyEqvFinite SChopAndB SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond* **by** *fastforce*

qed

lemma *SFinSChopChain*:

$\vdash (((init\ w) \wedge finite \longrightarrow sfin\ (init\ w1))) \frown$
 $((((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2)))$
 $\wedge finite$
 $\longrightarrow (((init\ w) \wedge finite \longrightarrow sfin\ (init\ w2))))$

proof –

have 1: $\vdash (init\ w) \wedge finite \wedge$
 $((init\ w) \wedge finite \longrightarrow sfin\ (init\ w1)) \frown$
 $((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2))$
 \longrightarrow
 $((init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow sfin\ (init\ w1))) \frown$
 $((((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite)$
by (*smt ChopAndFiniteDist StateAndSChop int-iffD2 inteq-reflection*
lift-and-com schop-d-def)

have 2: $\vdash (init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow sfin\ (init\ w1)) \longrightarrow$
 $sfin\ (init\ w1)$

by *auto*

have 3: $\vdash ((init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow sfin\ (init\ w1))) \frown$
 $((((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite)$
 \longrightarrow
 $(sfin\ (init\ w1)) \frown (((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2)) \wedge finite)$

using 2 *LeftSChopImpSChop* **by** *blast*

have 4: $\vdash (sfin\ (init\ w1)) \frown (((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2))) =$
 $\Diamond((init\ w1) \wedge ((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2)))$

using *SFinSChopEqvDiamond* **by** *blast*

have 41: $\vdash ((init\ w1) \wedge finite \wedge ((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2))) \longrightarrow sfin\ (init\ w2)$
by *auto*

have 42: $\vdash \Diamond((init\ w1) \wedge finite \wedge ((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2))) \longrightarrow \Diamond(sfin\ (init\ w2))$
using 41 *DiamondImpDiamond* **by** *blast*

have 5: $\vdash \Diamond(sfin\ (init\ w2)) \longrightarrow sfin\ (init\ w2)$

using *DiamondSFinImpSFin* **by** *blast*

have 6: $\vdash (init\ w) \wedge finite \wedge ((init\ w) \wedge finite \longrightarrow sfin\ (init\ w1)) \frown$
 $((init\ w1) \wedge finite \longrightarrow sfin\ (init\ w2))$
 $\longrightarrow sfin\ (init\ w2)$

using 1 3 4 5 42

by (*smt SFinSChopEqvDiamond inteq-reflection lift-and-com lift-imp-trans*)

from 6 **show** *?thesis* **by** *fastforce*

qed

lemma *SChopRule*:

assumes $\vdash (init\ w) \wedge f \wedge finite \longrightarrow sfin\ (init\ w1)$

$\vdash (init\ w1) \wedge f1 \wedge finite \longrightarrow sfin\ (init\ w2)$

shows $\vdash (init\ w) \wedge (f \frown f1) \wedge finite \longrightarrow sfin\ (init\ w2)$

proof –

have 1: $\vdash (init\ w) \wedge (f \frown f1) \wedge finite \longrightarrow ((init\ w) \wedge f) \frown (f1 \wedge finite)$

using *StateAndSChopImport*

by (*smt ChopAndFiniteDist SChopAndB StateAndEmptySChop StateAndSChopImpSChopRule*
inteq-reflection lift-and-com schop-d-def)

have 2: $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w1)$ **using** *assms* **by** *auto*
hence 3: $\vdash ((\text{init } w) \wedge f) \frown (f1 \wedge \text{finite}) \longrightarrow (\text{sfm } (\text{init } w1)) \frown (f1 \wedge \text{finite})$
by (*smt DiamondEmptyEqvFinite Prop10 SChopAndB SFinEqvTrueSChopAndEmpty*
StateAndChopImpChopRule StateAndSChop TrueSChopEqvDiamond inteq-reflection schop-d-def)
have 4: $\vdash (\text{sfm } (\text{init } w1)) \frown (f1 \wedge \text{finite}) = \Diamond((\text{init } w1) \wedge f1 \wedge \text{finite})$
by (*rule SFinSChopEqvDiamond*)
have 5: $\vdash (\text{init } w1) \wedge f1 \wedge \text{finite} \longrightarrow \text{sfm } (\text{init } w2)$ **using** *assms* **by** *auto*
hence 6: $\vdash \Diamond((\text{init } w1) \wedge f1 \wedge \text{finite}) \longrightarrow \Diamond(\text{sfm } (\text{init } w2))$ **by** (*rule DiamondImpDiamond*)
have 7: $\vdash \Diamond(\text{sfm } (\text{init } w2)) \longrightarrow \text{sfm } (\text{init } w2)$ **using** *DiamondSFinImpSFin* **by** *blast*
from 1 3 4 6 7 **show** *?thesis* **by** *fastforce*
qed

lemma *SChopRep*:

assumes $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow f1 \wedge \text{sfm } (\text{init } w1)$
 $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1$
shows $\vdash (\text{init } w) \wedge (f \frown g) \wedge \text{finite} \longrightarrow (f1 \frown g1)$
proof –
have 1: $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow (f1 \wedge \text{sfm } (\text{init } w1))$ **using** *assms* **by** *auto*
hence 2: $\vdash (\text{init } w) \wedge (f \frown (g \wedge \text{finite})) \longrightarrow (f1 \wedge \text{sfm } (\text{init } w1)) \frown (g \wedge \text{finite})$
by (*metis DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty Prop12 SChopSFinExportA*
SFinEqvTrueSChopAndEmpty StateAndChopImpChopRule StateAndEmptySChop
TrueSChopEqvDiamond inteq-reflection schop-d-def)
have 3: $\vdash (f1 \wedge \text{sfm } (\text{init } w1)) \frown (g \wedge \text{finite}) = f1 \frown ((\text{init } w1) \wedge (g \wedge \text{finite}))$
using *AndSFinSChopEqvStateAndSChop* **by** *blast*
have 4: $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1$ **using** *assms* **by** *auto*
hence 5: $\vdash f1 \frown ((\text{init } w1) \wedge g \wedge \text{finite}) \longrightarrow f1 \frown g1$
using *RightSChopImpSChop* **by** *blast*
from 2 3 5 **show** *?thesis*
by (*smt DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty SFinAndSChop SFinEqvTrueSChopAndEmpty*
TrueSChopEqvDiamond inteq-reflection lift-and-com lift-imp-trans)
qed

lemma *SChopRepAndSFin*:

assumes $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow f1 \wedge \text{sfm } (\text{init } w1)$
 $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1 \wedge \text{sfm } (\text{init } w2)$
shows $\vdash (\text{init } w) \wedge (f \frown g) \wedge \text{finite} \longrightarrow (f1 \frown g1) \wedge \text{sfm } (\text{init } w2)$
proof –
have 1: $\vdash (\text{init } w) \wedge f \wedge \text{finite} \longrightarrow f1 \wedge \text{sfm } (\text{init } w1)$ **using** *assms* **by** *auto*
have 2: $\vdash (\text{init } w1) \wedge g \wedge \text{finite} \longrightarrow g1 \wedge \text{sfm } (\text{init } w2)$ **using** *assms* **by** *auto*
have 3: $\vdash (\text{init } w) \wedge (f \frown g) \wedge \text{finite} \longrightarrow f1 \frown (g1 \wedge \text{sfm } (\text{init } w2))$
using 1 2 **by** (*rule SChopRep*)
have 4: $\vdash f1 \frown (g1 \wedge \text{sfm } (\text{init } w2)) \longrightarrow f1 \frown g1$ **by** (*rule SChopAndA*)
have 5: $\vdash f1 \frown (g1 \wedge \text{sfm } (\text{init } w2)) \longrightarrow f1 \frown \text{sfm } (\text{init } w2)$ **by** (*rule SChopAndB*)
have 6: $\vdash f1 \frown \text{sfm } (\text{init } w2) \longrightarrow \text{sfm } (\text{init } w2)$
by (*rule SChopSFinImpSFin*)
from 1 2 3 4 5 6 **show** *?thesis* **using** *SChopRep SChopRule* **by** *fastforce*
qed

lemma *TrueSChopMoreEqvMore*:

$\vdash \# \text{True} \frown \text{more} = \text{more}$
by (metis ChopAssoc TrueChopMoreEqvMore TrueEqvTrueSChopTrue inteq-reflection schop-d-def)

lemma SChopFmoreEqvFmore:

$\vdash \# \text{True} \frown \text{fmore} = \text{fmore}$
by (metis ChopAndFiniteDist TrueChopMoreEqvMore fmore-d-def inteq-reflection schop-d-def)

lemma MoreSChopLoop:

assumes $\vdash f \longrightarrow \text{more} \frown f$
shows $\vdash \text{finite} \longrightarrow \neg f$
proof –
have 1: $\vdash f \longrightarrow \text{more} \frown f$
 using *assms* **by** *auto*
hence 11: $\vdash \Diamond(f) \longrightarrow \Diamond(\text{more} \frown f)$
 using *DiamondImpDiamond* **by** *blast*
have 12: $\vdash \Diamond(\text{more} \frown f) = \# \text{True} \frown (\text{more} \frown f)$
 by (*simp add: DiamondSChopdef*)
have 13: $\vdash \# \text{True} \frown (\text{more} \frown f) = (\# \text{True} \frown \text{more}) \frown f$
 by (*rule SChopAssoc*)
have 14: $\vdash \Diamond(\text{more} \frown f) = \text{more} \frown f$
 using 12 13 **by** (metis TrueSChopMoreEqvMore inteq-reflection)
have 2: $\vdash \text{more} \frown f = \bigcirc(\Diamond f)$
 using *MoreSChopEqvNextDiamond* **by** *blast*
have 3: $\vdash \Diamond(f) \longrightarrow \bigcirc(\Diamond f)$
 using 11 14 2 **by** *fastforce*
hence 4: $\vdash \text{finite} \longrightarrow \neg(\Diamond f)$
 using *NextLoop* **by** *blast*
have 5: $\vdash \neg(\Diamond f) \longrightarrow \neg f$
 using *NowImpDiamond* **by** *fastforce*
from 4 5 **show** ?thesis **using** *lift-imp-trans* **by** *blast*
qed

lemma MoreSChopContra:

assumes $\vdash f \wedge \neg g \longrightarrow (\text{more} \frown (f \wedge \neg g))$
shows $\vdash f \wedge \text{finite} \longrightarrow g$
proof –
have 1: $\vdash f \wedge \neg g \longrightarrow (\text{more} \frown (f \wedge \neg g))$ **using** *assms* **by** *auto*
hence 2: $\vdash \text{finite} \longrightarrow \neg(f \wedge \neg g)$ **by** (*rule MoreSChopLoop*)
from 2 **show** ?thesis
by (*simp add: Valid-def finite-defs infinite-defs sum.case-eq-if*)
qed

lemma MoreSChopLoopFinite:

assumes $\vdash f \wedge \text{finite} \longrightarrow \text{more} \frown f$
shows $\vdash \text{finite} \longrightarrow \neg f$
proof –
have 1: $\vdash f \wedge \text{finite} \longrightarrow \text{more} \frown f$
 using *assms* **by** *auto*
hence 11: $\vdash \Diamond(f \wedge \text{finite}) \longrightarrow \Diamond(\text{more} \frown f)$
 using *DiamondImpDiamond* **by** *blast*

```

have 12:  $\vdash \Diamond (more \frown f) = \#True \frown (more \frown f)$ 
  by (simp add: DiamondSChopdef)
have 13:  $\vdash \#True \frown (more \frown f) = (\#True \frown more) \frown f$ 
  by (rule SChopAssoc)
have 14:  $\vdash \Diamond (more \frown f) = more \frown f$ 
  using 12 13 by (metis TrueSChopMoreEqvMore inteq-reflection)
have 2:  $\vdash more \frown f = \bigcirc(\Diamond f)$ 
  using MoreSChopEqvNextDiamond by blast
have 3:  $\vdash \Diamond (f \wedge finite) \longrightarrow \bigcirc(\Diamond f)$ 
  using 11 14 2 by fastforce
have 31:  $\vdash \Diamond (f \wedge finite) = ((\Diamond f) \wedge finite)$ 
  by (smt 3 ChopAndB ChopAndNotChopImp FiniteChopFiniteEqvFinite FiniteChopInfEqvInf
    NextDiamondImpDiamond Prop11 Prop12 finite-d-def infinite-d-def inteq-reflection
    lift-imp-trans sometimes-d-def)
have 32:  $\vdash (\Diamond f) \wedge finite \longrightarrow \bigcirc(\Diamond f)$ 
  using 3 31 by fastforce
hence 4:  $\vdash finite \longrightarrow \neg(\Diamond f)$ 
  by (metis (no-types, lifting) DiamondIntro FiniteChopInfEqvInf InfEqvNotFinite Prop09
    finite-d-def int-simps(15) int-simps(32) inteq-reflection sometimes-d-def)
have 5:  $\vdash \neg(\Diamond f) \longrightarrow \neg f$ 
  by (simp add: NowImpDiamond)
from 4 5 show ?thesis using lift-imp-trans by fastforce
qed

```

lemma *MoreSChopContraFinite*:

```

assumes  $\vdash (f \wedge \neg g) \wedge finite \longrightarrow (more \frown (f \wedge \neg g))$ 
shows  $\vdash f \wedge finite \longrightarrow g$ 

```

proof —

```

have 1:  $\vdash (f \wedge \neg g) \wedge finite \longrightarrow (more \frown (f \wedge \neg g))$  using assms by auto
hence 2:  $\vdash finite \longrightarrow \neg(f \wedge \neg g)$  by (simp add: MoreSChopLoopFinite)
from 2 show ?thesis by (simp add: Valid-def)

```

qed

lemma *SChopLoop*:

```

assumes  $\vdash f \longrightarrow g \frown f$ 
   $\vdash g \longrightarrow fmore$ 
shows  $\vdash finite \longrightarrow \neg f$ 

```

proof —

```

have 1:  $\vdash f \longrightarrow g \frown f$  using assms by auto
have 2:  $\vdash g \longrightarrow more$  using assms by (simp add: Prop12 fmore-d-def)
hence 3:  $\vdash g \frown f \longrightarrow more \frown f$  by (rule LeftSChopImpSChop)
have 4:  $\vdash f \longrightarrow more \frown f$  using 1 3 by fastforce
from 4 show ?thesis using MoreSChopLoop by auto

```

qed

lemma *SChopLoopB*:

```

assumes  $\vdash f \longrightarrow g \frown f$ 
   $\vdash g \longrightarrow more$ 
shows  $\vdash finite \longrightarrow \neg f$ 

```

proof —

have 1: $\vdash f \longrightarrow g \frown f$ **using** *assms* **by** *auto*
have 2: $\vdash g \longrightarrow \text{more}$ **using** *assms* **by** *auto*
hence 3: $\vdash g \frown f \longrightarrow \text{more} \frown f$ **by** (*rule LeftSChopImpSChop*)
have 4: $\vdash f \longrightarrow \text{more} \frown f$ **using** 1 3 **by** *fastforce*
from 4 **show** ?thesis **using** *MoreSChopLoop* **by** *blast*
qed

lemma *SChopContra*:

assumes $\vdash f \wedge \neg g \longrightarrow h \frown f \wedge \neg (h \frown g)$
 $\vdash h \longrightarrow \text{fmore}$

shows $\vdash f \wedge \text{finite} \longrightarrow g$

proof –

have 1: $\vdash f \wedge \neg g \longrightarrow h \frown f \wedge \neg (h \frown g)$ **using** *assms* **by** *auto*
have 2: $\vdash h \longrightarrow \text{more}$ **using** *assms* **by** (*simp add: Prop12 fmore-d-def*)
have 3: $\vdash h \frown f \wedge \neg (h \frown g) \longrightarrow h \frown (f \wedge \neg g)$ **by** (*rule SChopAndNotSChopImp*)
have 4: $\vdash h \frown (f \wedge \neg g) \longrightarrow \text{more} \frown (f \wedge \neg g)$ **using** 2 **by** (*rule LeftSChopImpSChop*)
have 5: $\vdash f \wedge \neg g \longrightarrow \text{more} \frown (f \wedge \neg g)$ **using** 1 3 4 **by** *fastforce*
from 5 **show** ?thesis **using** *MoreSChopContra* **by** *auto*
qed

lemma *SChopContraB*:

assumes $\vdash f \wedge \neg g \longrightarrow h \frown f \wedge \neg (h \frown g)$
 $\vdash h \longrightarrow \text{more}$

shows $\vdash f \wedge \text{finite} \longrightarrow g$

proof –

have 1: $\vdash f \wedge \neg g \longrightarrow h \frown f \wedge \neg (h \frown g)$ **using** *assms* **by** *auto*
have 2: $\vdash h \longrightarrow \text{more}$ **using** *assms* **by** *auto*
have 3: $\vdash h \frown f \wedge \neg (h \frown g) \longrightarrow h \frown (f \wedge \neg g)$ **by** (*rule SChopAndNotSChopImp*)
have 4: $\vdash h \frown (f \wedge \neg g) \longrightarrow \text{more} \frown (f \wedge \neg g)$ **using** 2 **by** (*rule LeftSChopImpSChop*)
have 5: $\vdash f \wedge \neg g \longrightarrow \text{more} \frown (f \wedge \neg g)$ **using** 1 3 4 **by** *fastforce*
from 5 **show** ?thesis **using** *MoreSChopContra* **by** *blast*
qed

10.7 Properties of SChopstar and SChopplus

lemma *AndEmptySChopAndEmptyEqvAndEmpty*:

$\vdash (f \wedge \text{empty}) \frown (f \wedge \text{empty}) = (f \wedge \text{empty})$

by (*simp add: Valid-def empty-defs schop-defs sum.case-eq-if*,
smt interval-prefix-intlen interval-prefix-length interval-suffix-zero
le-numeral-extra(3) sum.collapse(1))

lemma *SPowerImpFinite*:

$\vdash \text{spower } f \ n \longrightarrow \text{finite}$

proof

(*induct n*)

case 0

then show ?case

using *EmptyImpFinite* **by** *auto*

next

```

case (Suc n)
then show ?case
by (metis DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty Prop10 SChopSFinExportA
      SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection spow-Suc)
qed

```

lemma *SPowerCommute*:

$\vdash f \frown \text{spower } f \ n = \text{spower } f \ n \frown (f \wedge \text{finite})$

proof

(*induct n*)

case 0

then show ?case

by (metis ChopEmptySem EmptySChop intl inteq-reflection schop-d-def spow-0)

next

case (Suc n)

then show ?case

by (metis SChopAssoc inteq-reflection spow-Suc)

qed

lemma *SChopInductL*:

assumes $\vdash g \vee f \frown h \longrightarrow h$

shows $\vdash (\text{spower } f \ n) \frown g \longrightarrow h$

using *assms*

by (metis ChopInductFiniteL PowerSpowerdef Prop10 SPowerImpFinite inteq-reflection schop-d-def)

lemma *SChopInductMoreL*:

assumes $\vdash g \vee (f \wedge \text{more}) \frown h \longrightarrow h$

shows $\vdash (\text{spower } f \ n) \frown g \longrightarrow h$

proof

(*induct n*)

case 0

then show ?case **using** *assms* **by** (metis SChopInductL spow-0)

next

case (Suc n)

then show ?case

proof –

have 1: $\vdash \text{spower } f \ (\text{Suc } n) \frown g = (f \frown \text{spower } f \ n) \frown g$

by *simp*

have 2: $\vdash (f \frown \text{spower } f \ n) \frown g = f \frown ((\text{spower } f \ n) \frown g)$

by (meson SChopAssoc Prop11)

have 3: $\vdash f \frown ((\text{spower } f \ n) \frown g) \longrightarrow f \frown h$

by (*simp add: RightSChopImpSChop Suc.hyps*)

have 4: $\vdash f \frown h = ((f \wedge \text{more}) \frown h \vee ((f \wedge \text{empty})) \frown h)$

using *neq0-conv* **by** (*simp add: Valid-def finite-defs schop-defs more-defs empty-defs sum.case-eq-if, blast*)

have 5: $\vdash ((f \wedge \text{more})) \frown h \longrightarrow h$ **using** *assms* **by** *auto*

have 6: $\vdash ((f \wedge \text{empty})) \frown h \longrightarrow h$

by (*smt AndSChopA AndSChopB EmptySChop Prop10 Prop12 inteq-reflection*)

from 5 6 4 3 2 1 **show** ?thesis **by** *fastforce*

qed
qed

lemma *SChopImpFinite*:

assumes $\vdash f \longrightarrow \text{finite}$

shows $\vdash g \frown f \longrightarrow \text{finite}$

using *assms*

by (*metis* *DiamondImpDiamond* *FiniteChopEqvDiamond* *FiniteChopFiniteEqvFinite* *SChopImpDiamond* *inteq-reflection* *lift-imp-trans*)

lemma *SChopInductR*:

assumes $\vdash g \vee h \frown f \longrightarrow h$

shows $\vdash g \frown (\text{spower } f \ n) \longrightarrow h$

proof

(*induct* *n*)

case 0

then show ?case **using** *assms* *SChopEmpty* *SPowerImpFinite*

by (*smt* *AndSFinEqvSChopAndEmpty* *FiniteAndEmptyEqvEmpty* *MP* *Prop12* *int-iffD1* *int-simps*(33) *inteq-reflection* *lift-imp-trans* *spow-0*)

next

case (*Suc* *n*)

then show ?case

proof –

have 1: $\vdash g \frown (\text{spower } f \ (\text{Suc } n)) = g \frown (f \frown (\text{spower } f \ n))$

by *simp*

have 2: $\vdash g \frown (f \frown (\text{spower } f \ n)) = g \frown ((\text{spower } f \ n) \frown (f \wedge \text{finite}))$

using *SPowerCommute* **by** (*simp* *add*: *SPowerCommute* *RightSChopEqvSChop*)

have 3: $\vdash g \frown ((\text{spower } f \ n) \frown (f \wedge \text{finite})) =$

$(g \frown (\text{spower } f \ n)) \frown (f \wedge \text{finite})$

using *SChopAssoc* **by** *blast*

have 4: $\vdash (g \frown (\text{spower } f \ n)) \frown (f \wedge \text{finite}) \longrightarrow h \frown (f \wedge \text{finite})$

using *LeftSChopImpSChop* *Suc.hyps* **by** *blast*

have 5: $\vdash h \frown (f \wedge \text{finite}) \longrightarrow h$

using *assms*

by (*metis* *Prop03* *Prop10* *SChopAndA* *inteq-reflection* *lift-imp-trans*)

from 1 2 3 4 5 **show** ?thesis **by** *fastforce*

qed

qed

lemma *SChopExistSPower*:

$\vdash (g \frown (\exists n. \text{spower } f \ n)) = (\exists n. g \frown \text{spower } f \ n)$

using *SChopExist* **by** *fastforce*

lemma *ExistSChopSPower*:

$\vdash (\exists n. (\text{spower } f \ n) \frown g) = (\exists n. \text{spower } f \ n) \frown g$

using *ExistSChop* **by** *fastforce*

lemma *SPowerStarCommute*:

$\vdash f \frown (\exists n. \text{spower } f \ n) = (\exists n. \text{spower } f \ n) \frown (f \wedge \text{finite})$

proof –

```

have 1:  $\vdash f \frown (\exists n. \text{spower } f \ n) =$ 
   $(\exists n. f \frown \text{spower } f \ n)$ 
using SChopExistSPower by blast
have 2:  $\vdash (\exists n. f \frown \text{spower } f \ n) =$ 
   $(\exists n. (\text{spower } f \ n) \frown (f \wedge \text{finite}))$ 
using SPowerCommute by fastforce
have 3:  $\vdash (\exists n. (\text{spower } f \ n) \frown (f \wedge \text{finite})) =$ 
   $(\exists n. (\text{spower } f \ n)) \frown (f \wedge \text{finite})$ 
using ExistSChopSPower by blast
from 1 2 3 show ?thesis by fastforce
qed

```

```

lemma SPowerSucAndEmptyEqvAndEmpty:
 $\vdash (\text{spower } (f \wedge \text{empty}) (\text{Suc } n)) = (f \wedge \text{empty})$ 
proof
  (induct n)
  case 0
  then show ?case
    by (metis PowerSpowerdef PowerSucAndEmptyEqvAndEmpty inteq-reflection)
  next
  case (Suc n)
  then show ?case
    by (metis AndEmptySChopAndEmptyEqvAndEmpty inteq-reflection spow-Suc)
qed

```

```

lemma SPowerOr:
 $\vdash (\text{spower } (f \vee g) (\text{Suc } n)) = ((f \frown \text{spower } (f \vee g) \ n) \vee$ 
 $(g \frown \text{spower } (f \vee g) \ n))$ 
by (simp add: FiniteOr OrSChopEqvRule)

```

```

lemma PowerEmptyOrMore:
 $\vdash (\text{spower } ((f \wedge \text{empty}) \vee (f \wedge \text{more})) (\text{Suc } n)) =$ 
 $((f \wedge \text{empty}) \frown (\text{spower } ((f \wedge \text{empty}) \vee (f \wedge \text{more})) \ n) \vee$ 
 $(f \wedge \text{more}) \frown (\text{power } ((f \wedge \text{empty}) \vee (f \wedge \text{more})) \ n))$ 
using SPowerOr
by (metis PowerSpowerdef inteq-reflection)

```

```

lemma SPSEqvEmptyOrSChopSPS:
 $\vdash \text{spowerstar } f = (\text{empty} \vee f \frown \text{spowerstar } f)$ 
by (simp add: spowerstar-d-def spowersem)

```

```

lemma EmptyImpSCS:
 $\vdash \text{empty} \longrightarrow \text{schopstar } f$ 
proof —
  have 1:  $\vdash \text{schopstar } f = (\text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f)$ 
    by (rule SChopstarEqv)
  have 2:  $\vdash \text{empty} \longrightarrow \text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f$  by auto
  from 1 2 show ?thesis by fastforce

```

qed

lemma *SCSEqvOrSChopSCS*:

$\vdash \text{schopstar } f = (\text{empty} \vee (f \frown \text{schopstar } f))$

proof –

have 1: $\vdash \text{schopstar } f = (\text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f)$

by (*rule SChopstarEqv*)

have 2: $\vdash (f \wedge \text{more}) \frown \text{schopstar } f \longrightarrow f \frown \text{schopstar } f$

by (*rule AndSChopA*)

have 3: $\vdash \text{schopstar } f \longrightarrow \text{empty} \vee f \frown \text{schopstar } f$

using 1 2 **by** (*metis int-iffD1 Prop08*)

have 4: $\vdash \text{empty} \longrightarrow \text{schopstar } f$ **by** (*rule EmptyImpSCS*)

have 5: $\vdash f \longrightarrow \text{empty} \vee (f \wedge \text{more})$ **by** (*auto simp: empty-d-def*)

have 6: $\vdash f \frown \text{schopstar } f \longrightarrow \text{schopstar } f \vee (f \wedge \text{more}) \frown \text{schopstar } f$

using 5 **by** (*rule EmptyOrSChopImpRule*)

have 7: $\vdash \text{schopstar } f \longrightarrow \text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f$

using 1 **by** *fastforce*

have 8: $\vdash f \frown \text{schopstar } f \longrightarrow \text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f$

using 6 7 **by** *fastforce*

hence 9: $\vdash f \frown \text{schopstar } f \longrightarrow \text{schopstar } f$ **using** 1 **by** *fastforce*

have 10: $\vdash \text{empty} \vee f \frown \text{schopstar } f \longrightarrow \text{schopstar } f$ **using** 9 4 **by** *fastforce*

from 3 10 **show** ?thesis **by** *fastforce*

qed

lemma *SPowerSChopCommute*:

$\vdash ((f \wedge \text{more}) \frown \text{spower } (f \wedge \text{more}) \ n = \text{spower } (f \wedge \text{more}) \ n \frown ((f \wedge \text{more}) \wedge \text{finite}))$

using *SPowerCommute* **by** *auto*

lemma *SChopExist*:

$\vdash (g \frown (\exists n. \text{spower } (f \wedge \text{more}) \ n)) = (\exists n. g \frown \text{spower } (f \wedge \text{more}) \ n)$

using *SChopExistSPower* **by** *auto*

lemma *ExistSChop*:

$\vdash (\exists n. (\text{spower } (f \wedge \text{more}) \ n) \frown g) = (\exists n. \text{spower } (f \wedge \text{more}) \ n) \frown g$

using *ExistSChopSPower* **by** *auto*

lemma *SPowerstarInductL*:

assumes $\vdash g \vee f \frown h \longrightarrow h$

shows $\vdash (\text{spowerstar } f) \frown g \longrightarrow h$

proof –

have 1: $\vdash (\text{spowerstar } f) \frown g = ((\exists n. \text{spower } f \ n) \frown g)$

by (*simp add: spowerstar-d-def LeftChopEqvChop*)

have 2: $\vdash (\exists n. \text{spower } f \ n) \frown g =$

$(\exists n. (\text{spower } f \ n) \frown g)$

using *ExistSChopSPower* **by** *fastforce*

have 3: $\bigwedge n. \vdash (\text{spower } f \ n) \frown g \longrightarrow h$

using *SChopInductL assms* **by** *blast*

have 4: $\vdash (\exists n. (\text{spower } f \ n) \frown g) \longrightarrow h$

using 3 **by** (simp add: Valid-def,
 smt ChopAssoc ChopOrEqv EmptyOrChopEqv inteq-reflection unl-lift2)
from 1 2 4 **show** ?thesis **by** (metis inteq-reflection)
qed

lemma *SChopstarInductL*:

assumes $\vdash g \vee f \frown h \longrightarrow h$
shows $\vdash (\text{schopstar } f) \frown g \longrightarrow h$

proof –

have 1: $\vdash (\text{schopstar } f) \frown g = (\exists n. \text{spower } (f \wedge \text{more}) n) \frown g$
by (simp add: chopstar-d-def spowerstar-d-def LeftChopEqvChop)
have 2: $\vdash (\exists n. \text{spower } (f \wedge \text{more}) n) \frown g =$
 $(\exists n. (\text{spower } (f \wedge \text{more}) n) \frown g)$
using ExistSChopSPower **by** fastforce
have 3: $\bigwedge n. \vdash (\text{spower } (f \wedge \text{more}) n) \frown g \longrightarrow h$
using SChopInductL assms
by (smt AndSChopA MP Prop02 Prop12 int-iffD2 int-simps(33) inteq-reflection lift-imp-trans)
have 4: $\vdash (\exists n. (\text{spower } (f \wedge \text{more}) n) \frown g) \longrightarrow h$
using 3 **by** (simp add: Valid-def,
 smt SChopAssoc SChopOrEqv EmptyOrSChopEqv inteq-reflection unl-lift2)
from 1 2 4 **show** ?thesis **by** (metis inteq-reflection)
qed

lemma *SChopstarInductMoreL*:

assumes $\vdash g \vee (f \wedge \text{more}) \frown h \longrightarrow h$
shows $\vdash (\text{schopstar } f) \frown g \longrightarrow h$

proof –

have 1: $\vdash (\text{schopstar } f) \frown g = (\exists n. \text{spower } (f \wedge \text{more}) n) \frown g$
by (simp add: chopstar-d-def spowerstar-d-def LeftChopEqvChop)
have 2: $\vdash (\exists n. \text{spower } (f \wedge \text{more}) n) \frown g =$
 $(\exists n. (\text{spower } (f \wedge \text{more}) n) \frown g)$
using ExistSChopSPower **by** fastforce
have 3: $\bigwedge n. \vdash (\text{spower } (f \wedge \text{more}) n) \frown g \longrightarrow h$
using SChopInductL assms **by** (metis)
have 4: $\vdash (\exists n. (\text{spower } (f \wedge \text{more}) n) \frown g) \longrightarrow h$
using 3 **by** fastforce
from 1 2 4 **show** ?thesis
by (metis inteq-reflection)
qed

lemma *SPowerstarInductR*:

assumes $\vdash g \vee h \frown f \longrightarrow h$
shows $\vdash g \frown (\text{spowerstar } f) \longrightarrow h$

proof –

have 1: $\vdash g \frown (\text{spowerstar } f) = g \frown ((\exists n. \text{spower } f n))$
by (simp add: spowerstar-d-def)
have 2: $\vdash (g \frown (\exists n. \text{spower } f n)) = (\exists n. g \frown (\text{spower } f n))$
using SChopExistSPower **by** blast
have 3: $\bigwedge n. \vdash g \frown (\text{spower } f n) \longrightarrow h$

```

using SChopInductR assms by blast
have 4:  $\vdash (\exists n. g \wedge (\text{spower } f \ n)) \longrightarrow h$ 
using 3 by (simp add: Valid-def,
  smt SChopEmpty SChopOrEqv inteq-reflection unl-lift2)
from 1 2 4 show ?thesis by (metis inteq-reflection)
qed

```

```

lemma SChopstarInductR:
  assumes  $\vdash g \vee h \wedge f \longrightarrow h$ 
  shows  $\vdash g \wedge (\text{s chopstar } f) \longrightarrow h$ 
proof -
  have 1:  $\vdash g \wedge (\text{s chopstar } f) =$ 
     $g \wedge ((\exists n. \text{spower } (f \wedge \text{more}) \ n))$ 
  by (simp add: s chopstar-d-def spowerstar-d-def)
  have 2:  $\vdash (g \wedge (\exists n. \text{spower } (f \wedge \text{more}) \ n)) =$ 
     $((\exists n. g \wedge \text{spower } (f \wedge \text{more}) \ n))$ 
  using SChopExistSPower by fastforce
  have 3:  $\bigwedge n. \vdash g \wedge (\text{spower } (f \wedge \text{more}) \ n) \longrightarrow h$ 
  using SChopInductR assms
  by (smt SChopAndA MP Prop02 Prop12 int-iffD2 int-simps(33) inteq-reflection lift-imp-trans)
  have 4:  $\vdash (\exists n. g \wedge (\text{spower } (f \wedge \text{more}) \ n)) \longrightarrow h$ 
  using 3 by (simp add: Valid-def,
    smt SChopEmpty SChopOrEqv inteq-reflection unl-lift2)
  from 1 2 4 show ?thesis by (metis inteq-reflection)
qed

```

```

lemma SChopstarInductMoreR:
  assumes  $\vdash g \vee h \wedge (f \wedge \text{more}) \longrightarrow h$ 
  shows  $\vdash g \wedge (\text{s chopstar } f) \longrightarrow h$ 
proof -
  have 1:  $\vdash g \wedge (\text{s chopstar } f) = g \wedge ((\exists n. \text{spower } (f \wedge \text{more}) \ n))$ 
  by (simp add: s chopstar-d-def spowerstar-d-def)
  have 2:  $\vdash (g \wedge (\exists n. \text{spower } (f \wedge \text{more}) \ n)) =$ 
     $((\exists n. g \wedge \text{spower } (f \wedge \text{more}) \ n))$ 
  using SChopExistSPower by fastforce
  have 3:  $\bigwedge n. \vdash g \wedge (\text{spower } (f \wedge \text{more}) \ n) \longrightarrow h$ 
  using SChopInductR assms by (metis)
  have 4:  $\vdash (\exists n. g \wedge (\text{spower } (f \wedge \text{more}) \ n)) \longrightarrow h$ 
  using 3 by (simp add: Valid-def,
    smt SChopEmpty SChopOrEqv inteq-reflection unl-lift2)
  from 1 2 4 show ?thesis by (metis inteq-reflection)
qed

```

```

lemma SChopstarImpSPowerstar:
   $\vdash \text{s chopstar } f \longrightarrow \text{spowerstar } f$ 
by (metis (mono-tags, lifting) PowerPowerdef SChopstarFPowerstardef fpowerstar-d-def intl
  intensional-rews(3) intensional-rews(6) inteq-reflection spowerstar-d-def)

```

lemma *SPowerstarImpSChopstar*:

$\vdash \text{spowerstar } f \longrightarrow \text{schopstar } f$

by (*metis* (*mono-tags*, *lifting*) *PowerSpowerdef SChopstarFPowerstardef fpowerstar-d-def intl*
intensional-rews(3) *intensional-rews*(6) *inteq-reflection spowerstar-d-def*)

lemma *SChopstarEqvSPowerstar*:

$\vdash \text{schopstar } f = \text{spowerstar } f$

using *SChopstarImpSPowerstar SPowerstarImpSChopstar* **by** *fastforce*

lemma *SCSAndMoreEqvAndMoreSChop*:

$\vdash (\text{schopstar } f \wedge \text{more}) = (f \wedge \text{more}) \frown \text{schopstar } f$

proof –

have 1: $\vdash (\text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f) \wedge \text{more} \longrightarrow (f \wedge \text{more}) \frown \text{schopstar } f$
by (*auto simp: empty-d-def*)

have 2: $\vdash \text{schopstar } f = (\text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f)$
by (*rule SChopstarEqv*)

have 3: $\vdash \text{schopstar } f \wedge \text{more} \longrightarrow (f \wedge \text{more}) \frown \text{schopstar } f$
using 1 2 **by** *fastforce*

have 4: $\vdash (f \wedge \text{more}) \frown \text{schopstar } f \longrightarrow \text{schopstar } f$
using 2 **by** *fastforce*

have 5: $\vdash (f \wedge \text{more}) \longrightarrow \text{more}$
by *auto*

hence 6: $\vdash (f \wedge \text{more}) \frown \text{schopstar } f \longrightarrow \text{more}$
by (*rule LeftSChopImpMoreRule*)

have 7: $\vdash (f \wedge \text{more}) \frown \text{schopstar } f \longrightarrow \text{schopstar } f \wedge \text{more}$
using 4 6 **by** *fastforce*

from 3 7 **show** *?thesis* **by** *fastforce*

qed

lemma *SPowerAndMoreAndFinite*:

$\vdash ((\text{spower } (f \wedge \text{more}) \text{ } n) \wedge \text{finite}) = (\text{spower } (f \wedge \text{more}) \text{ } n)$

by (*meson Prop10 Prop11 SPowerImpFinite*)

lemma *SCSAndFinite*:

$\vdash (\text{schopstar } f \wedge \text{finite}) = \text{schopstar } f$

proof –

have 1: $\vdash (\text{schopstar } f \wedge \text{finite}) = ((\exists n. \text{spower } (f \wedge \text{more}) \text{ } n) \wedge \text{finite})$
by (*simp add: chopstar-d-def spowerstar-d-def intl*)

have 2: $\vdash ((\exists n. \text{spower } (f \wedge \text{more}) \text{ } n) \wedge \text{finite}) =$
 $(\exists n. \text{spower } (f \wedge \text{more}) \text{ } n \wedge \text{finite})$
by (*simp add: Valid-def*)

have 3: $\vdash (\exists n. \text{spower } (f \wedge \text{more}) \text{ } n \wedge \text{finite}) =$
 $(\exists n. (\text{spower } (f \wedge \text{more}) \text{ } n))$

using *SPowerAndMoreAndFinite* **by** *fastforce*

have 4: $\vdash (\exists n. (\text{spower } (f \wedge \text{more}) \text{ } n)) = \text{schopstar } f$
by (*simp add: chopstar-d-def spowerstar-d-def*)

from 1 2 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *SPowerchopAndFmore*:

$\vdash ((\text{spower } (f \wedge \text{more}) (\text{Suc } n)) \wedge \text{fmore}) = (\text{spower } (f \wedge \text{more}) (\text{Suc } n))$

proof

(*induct n*)

case 0

then show ?case

by (*metis LeftSChopImpMoreRule Prop11 Prop12 SPowerAndMoreAndFinite fmore-d-def lift-and-com spow-Suc*)

next

case (*Suc n*)

then show ?case

by (*smt LeftSChopImpFMoreRule Prop10 Prop12 SPowerCommute int-iffD1 inteq-reflection spow-Suc*)

qed

lemma *ExistSPowerAndMoreExpand*:

$\vdash (\exists n. \text{spower } (f \wedge \text{more}) n) = (\text{empty} \vee (\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n))))$

using *spowersem1*[of *LIFT*(*f* \wedge *more*)] **by** *auto*

lemma *SCSAndMoreEqvAndFMoreSChop*:

$\vdash (\text{schopstar } f \wedge \text{fmore}) = (f \wedge \text{more}) \frown \text{schopstar } f$

proof –

have 1: $\vdash (\text{schopstar } f \wedge \text{fmore}) = ((\exists n. \text{spower } (f \wedge \text{more}) n) \wedge \text{fmore})$

by (*simp add: chopstar-d-def spowstar-d-def intl*)

have 2: $\vdash ((\exists n. \text{spower } (f \wedge \text{more}) n) \wedge \text{fmore}) =$

$(\exists n. \text{spower } (f \wedge \text{more}) n \wedge \text{fmore})$

by (*simp add: Valid-def*)

have 3: $\vdash (\exists n. \text{spower } (f \wedge \text{more}) n \wedge \text{fmore}) =$

$((\text{spower } (f \wedge \text{more}) 0 \vee (\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n)))) \wedge \text{fmore})$

using *ExistSPowerAndMoreExpand* **by** *fastforce*

have 4: $\vdash ((\text{spower } (f \wedge \text{more}) 0 \vee (\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n)))) \wedge \text{fmore}) =$

$((\text{spower } (f \wedge \text{more}) 0 \wedge \text{fmore}) \vee ((\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n))) \wedge \text{fmore}))$

by *auto*

have 5: $\vdash (((\text{spower } (f \wedge \text{more}) 0 \wedge \text{fmore}) \vee ((\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n))) \wedge \text{fmore}))) =$

$((\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n))) \wedge \text{fmore})$

using *NotFmoreAndEmpty* **by** *fastforce*

have 6: $\vdash ((\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n))) \wedge \text{fmore}) =$

$(\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n)))$

using *SPowerchopAndFmore* **by** *fastforce*

have 7: $\vdash (\exists n. (\text{spower } (f \wedge \text{more}) (\text{Suc } n))) =$

$(\exists n. ((f \wedge \text{more}) \frown (\text{spower } (f \wedge \text{more}) n)))$

by (*simp*)

have 8: $\vdash (\exists n. ((f \wedge \text{more}) \frown (\text{spower } (f \wedge \text{more}) n))) =$

$(f \wedge \text{more}) \frown (\exists n. (\text{spower } (f \wedge \text{more}) n))$

using *SChopExist* **by** *fastforce*

have 9: $\vdash (\exists n. (\text{spower } (f \wedge \text{more}) n)) =$

schopstar f

by (simp add: schopstar-d-def spowerstar-d-def intl)
 hence 10: $\vdash (f \wedge \text{more}) \frown (\exists n. (\text{spower } (f \wedge \text{more}) n)) =$
 $(f \wedge \text{more}) \frown \text{schopstar } f$
 by (simp add: RightSCHopEqvSCHop)
 from 1 2 3 4 5 6 7 8 10 show ?thesis by (metis inteq-reflection)
 qed

lemma *SCSAndMoreImpSCHopSCS*:

$\vdash \text{schopstar } f \wedge \text{more} \longrightarrow f \frown \text{schopstar } f$

proof –

have 1: $\vdash (\text{schopstar } f \wedge \text{more}) = (f \wedge \text{more}) \frown \text{schopstar } f$ by (rule SCSAndMoreEqvAndMoreSCHop)
 have 2: $\vdash (f \wedge \text{more}) \frown \text{schopstar } f \longrightarrow f \frown \text{schopstar } f$ by (rule AndSCHopA)
 from 1 2 show ?thesis by fastforce
 qed

lemma *SCSMoreNotImpSCHopSCSAndMore*:

$\vdash \text{schopstar } f \wedge \text{more} \wedge \neg f \longrightarrow (f \wedge \text{more}) \frown (\text{schopstar } f \wedge \text{more})$

proof –

have 1: $\vdash (\text{schopstar } f \wedge \text{more}) = (f \wedge \text{more}) \frown \text{schopstar } f$
 by (rule SCSAndMoreEqvAndMoreSCHop)
 have 2: $\vdash \text{empty} \vee \text{more}$
 by (auto simp: empty-d-def)
 hence 3: $\vdash \text{schopstar } f \longrightarrow \text{empty} \vee (\text{schopstar } f \wedge \text{more})$
 by auto
 hence 4: $\vdash (f \wedge \text{more}) \frown \text{schopstar } f \longrightarrow (f \wedge \text{more}) \vee ((f \wedge \text{more}) \frown (\text{schopstar } f \wedge \text{more}))$
 using SCHopEmptyOrImpRule
 by (metis 1 AndMoreAndFiniteEqvAndFmore SCSAndMoreEqvAndFmoreSCHop inteq-reflection)
 hence 5: $\vdash (f \wedge \text{more}) \frown \text{schopstar } f \wedge \neg(f \wedge \text{more}) \longrightarrow ((f \wedge \text{more}) \frown (\text{schopstar } f \wedge \text{more}))$
 by fastforce
 have 6: $\vdash (f \wedge \text{more}) \frown \text{schopstar } f = ((f \wedge \text{more}) \frown \text{schopstar } f \wedge \text{more})$ using 1
 by auto
 have 7: $\vdash ((f \wedge \text{more}) \frown \text{schopstar } f \wedge \neg(f \wedge \text{more})) =$
 $((f \wedge \text{more}) \frown \text{schopstar } f \wedge \text{more} \wedge \neg(f \wedge \text{more}))$
 using 6 by auto
 have 8: $\vdash (f \wedge \text{more}) \frown \text{schopstar } f \wedge \text{more} \wedge \neg f \longrightarrow (f \wedge \text{more}) \frown (\text{schopstar } f \wedge \text{more})$
 using 5 7 by auto
 have 9: $\vdash (\text{schopstar } f \wedge \text{more} \wedge \neg f) = ((\text{schopstar } f \wedge \text{more}) \wedge (\text{more} \wedge \neg f))$
 by auto
 have 10: $\vdash ((\text{schopstar } f \wedge \text{more}) \wedge (\text{more} \wedge \neg f)) =$
 $((f \wedge \text{more}) \frown \text{schopstar } f \wedge (\text{more} \wedge \neg f))$
 using 1 by fastforce
 from 1 8 9 10 show ?thesis by fastforce
 qed

lemma *SChopplusCommutelmpA*:

$\vdash \text{schopstar } f \frown (f \wedge \text{finite}) \longrightarrow f \frown \text{schopstar } f$

by (metis SCHopstarEqvSPowerstar SPowerStarCommutate int-iffD1 inteq-reflection spowerstar-d-def)

lemma *SChopplusCommutelmpB*:

$\vdash f \frown \text{schopstar } f \longrightarrow \text{schopstar } f \frown (f \wedge \text{finite})$

by (*metis* *SChopstarEqvSPowerstar* *SPowerStarCommute* *int-iffD2* *inteq-reflection* *spowerstar-d-def*)

lemma *SChopplusCommute*:

$\vdash f \frown \text{schopstar } f = \text{schopstar } f \frown (f \wedge \text{finite})$

using *SChopplusCommuteImpA* *SChopplusCommuteImpB* **by** *fastforce*

lemma *SCSEqvOrChopSCSB*:

$\vdash \text{schopstar } f = (\text{empty} \vee (\text{schopstar } f \frown (f \wedge \text{finite})))$

by (*meson* *SCSEqvOrSChopSCS* *SChopplusCommute* *Prop06*)

lemma *SCSAndMoreImpSCSSChop*:

$\vdash \text{schopstar } f \wedge \text{more} \longrightarrow \text{schopstar } f \frown (f \wedge \text{finite})$

using *SCSAndMoreEqvAndMoreSChop* *SChopplusCommute* *SCSAndMoreImpSChopSCS* **by** *fastforce*

lemma *SPowerSChopSPower*:

$\vdash (\text{spower } (f \wedge \text{more}) \ n) \frown (\text{spower } (f \wedge \text{more}) \ k) = (\text{spower } (f \wedge \text{more}) \ (n+k))$

proof

(*induct* *n* *arbitrary*: *k*)

case *0*

then show ?*case* **by** (*metis* *EmptySChop* *add.left-neutral* *spow-0*)

next

case (*Suc* *n*)

then show ?*case*

by (*metis* *PowerChopPower* *PowerSpowerdef* *SPowerAndMoreAndFinite* *inteq-reflection* *schop-d-def*)

qed

lemma *SCSSChopSCS*:

$\vdash \text{schopstar } f \frown \text{schopstar } f = \text{schopstar } f$

by (*smt* *AndSFinEqvSChopAndEmpty* *DiamondEmptyEqvFinite* *EmptyImpSCS* *FiniteAndEmptyEqvEmpty* *Prop02* *Prop03* *Prop11* *RightSChopImpSChop* *SCSAndFinite* *SCSEqvOrSChopSCS* *SChopstarInductL* *SFinEqvTrueSChopAndEmpty* *TrueSChopEqvDiamond* *inteq-reflection*)

lemma *NotSCSImpMore*:

$\vdash \neg (\text{schopstar } f) \longrightarrow \text{more}$

proof –

have *1*: $\vdash \text{empty} \longrightarrow \text{schopstar } f$ **using** *EmptyImpSCS* **by** *blast*

hence *2*: $\vdash \neg \text{empty} \vee \text{schopstar } f$ **by** *fastforce*

from *2* **show** ?*thesis* **using** *1* *NotEmptyEqvMore* **by** *fastforce*

qed

lemma *NotSCSAndInf*:

$\vdash \neg (\text{schopstar } f \wedge \text{inf})$

using *InfEqvNotFinite* *SCSAndFinite* **by** *fastforce*

lemma *SCSSChopSCSImpSCS*:

$\vdash (\text{schopstar } f \frown \text{schopstar } f) \longrightarrow \text{schopstar } f$

by (*simp* *add*: *SCSSChopSCS* *int-iffD1*)

lemma *ImpSChopPlus*:

$\vdash f \wedge \text{finite} \longrightarrow f \frown \text{s chopstar } f$

proof –

have 1: $\vdash \text{s chopstar } f = (\text{empty} \vee f \frown \text{s chopstar } f)$ **by** (rule *SCSEqvOrSChopSCS*)

hence 2: $\vdash f \frown \text{s chopstar } f = (f \frown \text{empty} \vee f \frown (f \frown \text{s chopstar } f))$ **using** *SChopOrEqvRule* **by** *blast*

have 3: $\vdash \text{finite} \longrightarrow f \frown \text{empty} = f$ **using** *SChopEmpty* **by** *blast*

from 2 3 **show** ?thesis **by** *fastforce*

qed

lemma *ImpSCS*:

$\vdash f \wedge \text{finite} \longrightarrow \text{s chopstar } f$

proof –

have 1: $\vdash f \wedge \text{finite} \longrightarrow f \frown \text{s chopstar } f$ **by** (rule *ImpSChopPlus*)

hence 2: $\vdash f \wedge \text{finite} \longrightarrow \text{empty} \vee f \frown \text{s chopstar } f$ **by** *auto*

from 2 **show** ?thesis **using** *SCSEqvOrSChopSCS* **by** *fastforce*

qed

lemma *SCSSChopImpSCS*:

$\vdash \text{s chopstar } f \frown (f \wedge \text{finite}) \longrightarrow \text{s chopstar } f$

proof –

have 1: $\vdash f \wedge \text{finite} \longrightarrow \text{s chopstar } f$ **by** (rule *ImpSCS*)

hence 2: $\vdash \text{s chopstar } f \frown (f \wedge \text{finite}) \longrightarrow \text{s chopstar } f \frown \text{s chopstar } f$ **by** (rule *RightSChopImpSChop*)

hence 3: $\vdash \text{s chopstar } f \frown (f \wedge \text{finite}) \longrightarrow \text{s chopstar } f \frown \text{s chopstar } f$ **by** *auto*

have 4: $\vdash \text{s chopstar } f \frown \text{s chopstar } f \longrightarrow \text{s chopstar } f$ **by** (rule *SCSSChopSCSImpSCS*)

from 2 3 4 **show** ?thesis **using** *lift-imp-trans* **by** *blast*

qed

lemma *SChopPlusImpSCS*:

$\vdash f \frown \text{s chopstar } f \longrightarrow \text{s chopstar } f$

proof –

have 1: $\vdash f \frown \text{s chopstar } f \longrightarrow \text{empty} \vee f \frown \text{s chopstar } f$ **by** *auto*

from 1 **show** ?thesis **using** *SCSEqvOrSChopSCS* **by** *fastforce*

qed

lemma *SCSSChopEqvOrSChopPlusSChop*:

$\vdash \text{s chopstar } f \frown g = (g \vee (f \frown \text{s chopstar } f) \frown g)$

proof –

have 1: $\vdash \text{s chopstar } f = (\text{empty} \vee f \frown \text{s chopstar } f)$ **by** (rule *SCSEqvOrSChopSCS*)

from 1 **show** ?thesis **using** *EmptyOrSChopEqvRule* **by** *blast*

qed

lemma *SCSElim*:

assumes $\vdash \text{empty} \longrightarrow g$

$\vdash (f \wedge \text{more}) \frown g \longrightarrow g$

shows $\vdash \text{s chopstar } f \longrightarrow g$

proof –

have 1: $\vdash \text{empty} \vee (f \wedge \text{more}) \frown g \longrightarrow g$

using *assms* **using** *Prop02* **by** *blast*

have 2: $\vdash (\text{s chopstar } f) \frown \text{empty} \longrightarrow g$

using *SChopstarInductMoreL 1* **by** *blast*
from 2 **show** *?thesis*
by (*metis AndSFinEqvSChopAndEmpty DiamondEmptyEqvFinite FiniteAndEmptyEqvEmpty SCSAndFinite*
SFinEqvTrueSChopAndEmpty TrueSChopEqvDiamond inteq-reflection)
qed

lemma *SChopstarImp*:
assumes $\vdash f \frown (\text{schopstar } g) \vee \text{empty} \longrightarrow (\text{schopstar } g)$
shows $\vdash (\text{schopstar } f) \longrightarrow (\text{schopstar } g)$
using *assms SChopstarInductL SChopEmpty*
by (*smt AndSChopB MP Prop12 SCSElim int-iffD1 int-simps(33) inteq-reflection lift-and-com*
lift-imp-trans)

lemma *SCSSCSImpSCS*:
 $\vdash \text{schopstar } (\text{schopstar } f) \longrightarrow \text{schopstar } f$
proof –
have 1: $\vdash ((\text{schopstar } f) \frown (\text{schopstar } f)) \vee \text{empty} \longrightarrow (\text{schopstar } f)$
by (*meson SCSSChopSCSImpSCS EmptyImpSCS Prop02*)
from 1 **show** *?thesis* **using** *SChopstarImp* **by** *blast*
qed

lemma *SCSImpSCSSCS*:
 $\vdash \text{schopstar } f \longrightarrow \text{schopstar } (\text{schopstar } f)$
using *ImpSCS* **by** (*metis SCSAndFinite inteq-reflection*)

lemma *SCSSCSEqvSCS*:
 $\vdash \text{schopstar } (\text{schopstar } f) = \text{schopstar } f$
by (*simp add: SCSSCSImpSCS SCSImpSCSSCS int-iff1*)

lemma *RightEmptyOrSChopEqv*:
 $\vdash g \frown (\text{empty} \vee f) = ((g \wedge \text{finite}) \vee (g \frown f))$
proof –
have 1: $\vdash g \frown (\text{empty} \vee f) = (g \frown \text{empty} \vee g \frown f)$ **by** (*rule SChopOrEqv*)
have 2: $\vdash \text{finite} \longrightarrow g \frown \text{empty} = g$ **by** (*rule SChopEmpty*)
from 1 2 **show** *?thesis* **by** (*simp add: RightEmptyOrChopEqv chop-d-def*)
qed

lemma *RightEmptyOrSChopEqvRule*:
assumes $\vdash f = (\text{empty} \vee f1)$
shows $\vdash g \frown f = ((g \wedge \text{finite}) \vee (g \frown f1))$
proof –
have 1: $\vdash f = (\text{empty} \vee f1)$ **using** *assms* **by** *auto*
hence 2: $\vdash g \frown f = g \frown (\text{empty} \vee f1)$ **by** (*rule RightSChopEqvSChop*)
have 3: $\vdash g \frown (\text{empty} \vee f1) = ((g \wedge \text{finite}) \vee (g \frown f1))$ **by** (*rule RightEmptyOrSChopEqv*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *SChopPlusEqvOrSChopSChopPlus*:
 $\vdash (f \frown \text{schopstar } f) = ((f \wedge \text{finite}) \vee f \frown (f \frown \text{schopstar } f))$

proof –
have 1: $\vdash \text{schopstar } f = (\text{empty} \vee f \frown \text{schopstar } f)$ **by** (rule SCSEqvOrSCHopSCS)
from 1 **show** ?thesis **by** (rule RightEmptyOrSCHopEqvRule)
qed

lemma SCSAndEmptyEqvEmpty:
 $\vdash (\text{schopstar } f \wedge \text{empty}) = \text{empty}$
using EmptyImpSCS **by** fastforce

lemma NotAndMoreSCHopAndEmpty:
 $\vdash \neg(((f \wedge \text{more}) \frown g) \wedge \text{empty})$
by (metis LeftSCHopImpMoreRule Prop05 Prop12 Prop13 empty-d-def int-iffD1 int-simps(15)
 inteq-reflection lift-and-com)

lemma NotSCHopAndMoreAndEmpty:
 $\vdash \neg((f \frown (g \wedge \text{more})) \wedge \text{empty})$
by (simp add: NotChopAndMoreAndEmpty chop-d-def)

lemma SCHopSCSAndEmptyEqvAndEmpty:
 $\vdash ((f \frown \text{schopstar } f) \wedge \text{empty}) = (f \wedge \text{empty})$
proof –
have 1: $\vdash ((f \frown \text{schopstar } f) \wedge \text{empty}) = (f \wedge \text{empty}) \frown (\text{schopstar } f \wedge \text{empty})$
using SCHopAndEmptyEqvEmptySCHopEmpty **by** blast
have 2: $\vdash (f \wedge \text{empty}) \frown (\text{schopstar } f \wedge \text{empty}) = (f \wedge \text{empty}) \frown \text{empty}$
using SCSAndEmptyEqvEmpty **using** RightSCHopEqvSCHop **by** blast
have 3: $\vdash (f \wedge \text{empty}) \frown \text{empty} = (f \wedge \text{empty})$
by (smt AndEmptySCHopAndEmptyEqvAndEmpty AndSFinEqvSCHopAndEmpty FiniteAndEmptyEqvEmpty
 Prop11 Prop12 inteq-reflection)
from 1 2 3 **show** ?thesis **by** fastforce
qed

lemma AndMoreSCHopAndMoreEqvAndMoreSCHop:
 $\vdash ((f \wedge \text{more}) \frown g \wedge \text{more}) = (f \wedge \text{more}) \frown g$
by (meson AndSCHopB MoreSCHopImpMore Prop10 Prop11 lift-imp-trans)

lemma AndFmoreOrAndEmptyEqvAndFinite:
 $\vdash ((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{empty})) = (f \wedge \text{finite})$
by (simp add: Valid-def empty-defs more-defs finite-defs sum.case-eq-if, auto)

lemma SCHopPlusEqv:
 $\vdash (f \frown \text{schopstar } f) = ((f \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f))$
proof –
have 1: $\vdash \text{schopstar } f = (\text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f)$
by (rule SCHopstarEqv)
have 2: $\vdash \text{schopstar } f = (\text{empty} \vee f \frown \text{schopstar } f)$
by (rule SCSEqvOrSCHopSCS)
hence 3: $\vdash (\text{empty} \vee f \frown \text{schopstar } f) = (\text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f)$
using 1 2 **by** fastforce
have 4: $\vdash (f \wedge \text{more}) \frown \text{schopstar } f = (f \wedge \text{more}) \frown (\text{empty} \vee f \frown \text{schopstar } f)$

```

    using 2 using RightSChopEqvSChop by blast
  hence 5:  $\vdash \text{empty} \vee f \frown \text{schopstar } f = \text{empty} \vee (f \wedge \text{more}) \frown (\text{empty} \vee f \frown \text{schopstar } f)$ 
    using 3 4 by fastforce
  have 6:  $\vdash (f \wedge \text{more}) \frown (\text{empty} \vee f \frown \text{schopstar } f) =$ 
     $((f \wedge \text{more}) \frown \text{empty} \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f))$ 
    using SChopOrEqv by blast
  have 7:  $\vdash (f \wedge \text{more}) \frown \text{empty} = (f \wedge \text{more} \wedge \text{finite})$ 
    by (metis AndMoreAndFiniteEqvAndFmore ChopEmpty fmore-d-def inteq-reflection schop-d-def)
  have 8:  $\vdash (\text{empty} \vee f \frown \text{schopstar } f) =$ 
     $(\text{empty} \vee (f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f))$ 
    using 5 6 7 by (metis 2 3 inteq-reflection)
  have 9:  $\vdash ((\text{empty} \vee f \frown \text{schopstar } f) \wedge \text{more}) = (f \frown \text{schopstar } f \wedge \text{more})$ 
    by (auto simp: empty-d-def)
  have 10:  $\vdash ((\text{empty} \vee (f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f)) \wedge \text{more}) =$ 
     $((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f)) \wedge \text{more}$ 
    by (auto simp: empty-d-def)
  have 11:  $\vdash (((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f)) \wedge \text{more}) =$ 
     $((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f))$ 
    using 10 6 7 int-eq
    using AndMoreSChopAndMoreEqvAndMoreSChop by fastforce
  have 12:  $\vdash (f \frown \text{schopstar } f \wedge \text{more}) = ((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f))$ 
    using 8 9 10 11 by fastforce
  have 13:  $\vdash (f \frown \text{schopstar } f \wedge \text{empty}) = (f \wedge \text{empty})$ 
    by (rule SChopSCSAndEmptyEqvAndEmpty)
  have 14:  $\vdash ((f \wedge \text{more} \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f) \vee (f \wedge \text{empty})) =$ 
     $((f \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f))$ 
    using AndFmoreOrAndEmptyEqvAndFinite by (smt intl inteq-reflection unl-lift2)
  have 15:  $\vdash f \frown \text{schopstar } f = ((f \frown \text{schopstar } f \wedge \text{empty}) \vee (f \frown \text{schopstar } f \wedge \text{more}))$ 
    by (auto simp: empty-d-def)
  from 12 13 14 15 show ?thesis by fastforce
qed

```

lemma *SChopSChopPlusImpSChopPlus*:

$\vdash f \frown (f \frown \text{schopstar } f) \longrightarrow f \frown \text{schopstar } f$

proof –

```

  have 1:  $\vdash \text{empty} \vee \text{more}$  by (auto simp: empty-d-def)
  hence 2:  $\vdash f \longrightarrow \text{empty} \vee (f \wedge \text{more})$  by auto
  hence 3:  $\vdash f \frown (f \frown \text{schopstar } f) \longrightarrow (f \frown \text{schopstar } f) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f)$ 
    by (rule EmptyOrSChopImpRule)
  have 4:  $\vdash f \frown \text{schopstar } f = ((f \wedge \text{finite}) \vee (f \wedge \text{more}) \frown (f \frown \text{schopstar } f))$ 
    by (rule SChopPlusEqv)
  hence 5:  $\vdash (f \wedge \text{more}) \frown (f \frown \text{schopstar } f) \longrightarrow f \frown \text{schopstar } f$  by auto
  from 3 5 show ?thesis using SChopPlusImpSCS RightSChopImpSChop by blast
qed

```

lemma *SCSImpSCS*:

assumes $\vdash f \longrightarrow g$

shows $\vdash \text{schopstar } f \longrightarrow \text{schopstar } g$

using *assms*

by (*smt AndSChopB EmptyImpSCS Prop02 Prop10 SChopPlusImpSCS SChopstarImp inteq-reflection lift-imp-trans*)

lemma *SChopPlusImpSChopPlus*:

assumes $\vdash f \longrightarrow g$

shows $\vdash f \frown \text{s chopstar } f \longrightarrow g \frown \text{s chopstar } g$

using *assms* **by** (*simp add: SCSImpSCS SChopImpSChop*)

lemma *SChopPlusIntro*:

assumes $\vdash f \longrightarrow (g \wedge \text{finite}) \vee (g \wedge \text{more}) \frown f$

shows $\vdash f \wedge \text{finite} \longrightarrow g \frown \text{s chopstar } g$

proof –

have 1: $\vdash f \wedge \neg (g \wedge \text{finite}) \longrightarrow (g \wedge \text{more}) \frown f$ **using** *assms* **by** *auto*

have 2: $\vdash g \frown \text{s chopstar } g = ((g \wedge \text{finite}) \vee (g \wedge \text{more}) \frown (g \frown \text{s chopstar } g))$
by (*rule SChopPlusEqv*)

have 3: $\vdash f \wedge \neg (g \frown \text{s chopstar } g) \longrightarrow$
 $(g \wedge \text{more}) \frown f \wedge \neg ((g \wedge \text{more}) \frown (g \frown \text{s chopstar } g))$ **using** 1 2

by *fastforce*

have 4: $\vdash g \wedge \text{more} \longrightarrow \text{more}$ **by** *auto*

from 3 4 **show** *?thesis* **using** *SChopContraB* **by** *blast*

qed

lemma *SChopPlusElim*:

assumes $\vdash f \longrightarrow g$

$\vdash (f \wedge \text{more}) \frown g \longrightarrow g$

shows $\vdash f \frown \text{s chopstar } f \longrightarrow g$

proof –

have 1: $\vdash f \vee (f \wedge \text{more}) \frown g \longrightarrow g$

using *assms Prop02* **by** *blast*

have 2: $\vdash \text{s chopstar } f \frown f \longrightarrow g$

using *SChopstarInductMoreL 1* **by** *blast*

from 2 **show** *?thesis*

using *SChopplusCommutate* **by** (*metis Prop10 Prop12 SChopAndA inteq-reflection*)

qed

lemma *SChopPlusElimWithoutMore*:

assumes $\vdash f \longrightarrow g$

$\vdash f \frown g \longrightarrow g$

shows $\vdash f \frown \text{s chopstar } f \longrightarrow g$

proof –

have 1: $\vdash f \longrightarrow g$ **using** *assms* **by** *blast*

have 2: $\vdash (f \frown g) \longrightarrow g$ **using** *assms* **by** *blast*

have 3: $\vdash (f \wedge \text{more}) \frown g \longrightarrow f \frown g$ **by** (*rule AndSChopA*)

have 4: $\vdash (f \wedge \text{more}) \frown g \longrightarrow g$ **using** 2 3 *lift-imp-trans* **by** *blast*

from 1 4 **show** *?thesis* **using** *SChopPlusElim* **by** *blast*

qed

lemma *SChopPlusEqvSChopPlus*:

assumes $\vdash f = g$
shows $\vdash f \frown \text{schopstar } f = g \frown \text{schopstar } g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash f \longrightarrow g$ **by** *auto*
hence 3: $\vdash f \frown \text{schopstar } f \longrightarrow g \frown \text{schopstar } g$ **by** (*rule SCChopPlusImpSCHopPlus*)
have 4: $\vdash g \longrightarrow f$ **using** 1 **by** *auto*
hence 5: $\vdash g \frown \text{schopstar } g \longrightarrow f \frown \text{schopstar } f$ **by** (*rule SCChopPlusImpSCHopPlus*)
from 3 5 **show** *?thesis* **by** *fastforce*
qed

lemma *SCSEqvSCS*:

assumes $\vdash f = g$
shows $\vdash \text{schopstar } f = \text{schopstar } g$
proof –
have 1: $\vdash f = g$ **using** *assms* **by** *auto*
hence 2: $\vdash f \frown \text{schopstar } f = g \frown \text{schopstar } g$ **by** (*rule SCChopPlusEqvSCHopPlus*)
hence 3: $\vdash (\text{empty} \vee f \frown \text{schopstar } f) = (\text{empty} \vee g \frown \text{schopstar } g)$ **by** *auto*
from 3 **show** *?thesis* **using** *SCSEqvOrSCHopSCS* **by** (*metis int-eq*)
qed

lemma *AndSCSA*:

$\vdash \text{schopstar } (f \wedge g) \longrightarrow \text{schopstar } f$
proof –
have 1: $\vdash f \wedge g \longrightarrow f$ **by** *auto*
from 1 **show** *?thesis* **using** *SCSImpSCS* **by** *blast*
qed

lemma *AndSCSB*:

$\vdash \text{schopstar } (f \wedge g) \longrightarrow \text{schopstar } g$
proof –
have 1: $\vdash f \wedge g \longrightarrow g$ **by** *auto*
from 1 **show** *?thesis* **using** *SCSImpSCS* **by** *blast*
qed

lemma *SCSIntro*:

assumes $\vdash f \wedge \text{more} \longrightarrow (g \wedge \text{more}) \frown f$
shows $\vdash f \wedge \text{finite} \longrightarrow \text{schopstar } g$
proof –
have 1: $\vdash f \wedge \text{more} \longrightarrow (g \wedge \text{more}) \frown f$
using *assms* **by** *auto*
have 2: $\vdash \text{more} = (\neg \text{empty})$
by (*auto simp: empty-d-def*)
have 3: $\vdash f \wedge \neg \text{empty} \longrightarrow (g \wedge \text{more}) \frown f$
using 1 2 **by** *fastforce*
have 4: $\vdash \text{schopstar } g = (\text{empty} \vee (g \wedge \text{more}) \frown \text{schopstar } g)$
by (*rule SCChopstarEqv*)
hence 41: $\vdash (\neg(\text{empty} \vee (g \wedge \text{more}) \frown \text{schopstar } g)) = (\neg \text{empty} \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g))$
by *fastforce*
have 411: $\vdash (\neg \text{empty} \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g)) = (\text{more} \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g))$

using NotEmptyEqvMore by fastforce
 have 42: $\vdash \neg(\text{schopstar } g) = (\text{more} \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g))$
 using 4 41 411 by fastforce
 have 43: $\vdash f \wedge \neg(\text{schopstar } g) \longrightarrow f \wedge \text{more} \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g)$
 using 42 by fastforce
 have 44: $\vdash f \wedge \text{more} \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g) \longrightarrow$
 $(g \wedge \text{more}) \frown f \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g)$
 using 3 43 1 by auto
 have 5: $\vdash f \wedge \neg(\text{schopstar } g) \longrightarrow$
 $(g \wedge \text{more}) \frown f \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g)$
 using 43 44 lift-imp-trans by fastforce
 have 6: $\vdash g \wedge \text{more} \longrightarrow \text{more}$
 by auto
 from 5 6 show ?thesis using SChopContraB by blast
 qed

lemma SCSElimWithoutMore:

assumes $\vdash \text{empty} \longrightarrow g$
 $\vdash f \frown g \longrightarrow g$
 shows $\vdash \text{schopstar } f \longrightarrow g$
 proof –
 have 1: $\vdash \text{empty} \longrightarrow g$ using assms by blast
 have 2: $\vdash f \frown g \longrightarrow g$ using assms by blast
 have 3: $\vdash (f \wedge \text{more}) \frown g \longrightarrow f \frown g$ by (rule AndSChopA)
 have 4: $\vdash (f \wedge \text{more}) \frown g \longrightarrow g$ using 2 3 lift-imp-trans by blast
 from 1 4 show ?thesis using SCSElim by blast
 qed

lemma SChopAssocB:

$\vdash (f \frown g) \frown h = f \frown (g \frown h)$
 using SChopAssoc by fastforce

lemma SCSChopEqvSChopOrRule:

assumes $\vdash f = (\text{schopstar } g \frown h)$
 shows $\vdash f = ((g \frown f) \vee h)$
 proof –
 have 1: $\vdash f = (\text{schopstar } g \frown h)$ using assms by auto
 have 2: $\vdash \text{schopstar } g = (\text{empty} \vee (g \frown \text{schopstar } g))$ by (rule SCSEqvOrSChopSCS)
 hence 3: $\vdash \text{schopstar } g \frown h = (h \vee ((g \frown \text{schopstar } g) \frown h))$ by (rule EmptyOrSChopEqvRule)
 have 4: $\vdash (g \frown \text{schopstar } g) \frown h = g \frown (\text{schopstar } g \frown h)$ by (rule SChopAssocB)
 hence 41: $\vdash \text{schopstar } g \frown h = (h \vee g \frown (\text{schopstar } g \frown h))$ using 3 by fastforce
 have 5: $\vdash g \frown f = g \frown (\text{schopstar } g \frown h)$ using 1 by (rule RightSChopEqvSChop)
 hence 6: $\vdash (\text{schopstar } g \frown h) = (h \vee g \frown f)$ using 41 by fastforce
 hence 61: $\vdash (\text{schopstar } g \frown h) = ((g \frown f) \vee h)$ by auto
 from 1 61 show ?thesis by fastforce
 qed

lemma SCSChopIntroRule:

assumes $\vdash f \wedge \neg h \longrightarrow g \frown f$
 $\vdash g \longrightarrow \text{more}$

shows $\vdash f \wedge \text{finite} \longrightarrow \text{schopstar } g \frown h$
proof –
have 1: $\vdash f \wedge \neg h \longrightarrow g \frown f$
 using *assms* **by** *blast*
have 2: $\vdash g \longrightarrow \text{more}$
 using *assms* **by** *blast*
hence 3: $\vdash g \longrightarrow g \wedge \text{more}$
 by *auto*
hence 4: $\vdash g \frown f \longrightarrow (g \wedge \text{more}) \frown f$
 by (*rule LeftSChopImpSChop*)
have 5: $\vdash f \longrightarrow (g \wedge \text{more}) \frown f \vee h$
 using 1 4 **by** *fastforce*
have 6: $\vdash \text{schopstar } g = (\text{empty} \vee (g \wedge \text{more}) \frown \text{schopstar } g)$
 by (*rule SChopstarEqv*)
hence 7: $\vdash (\text{schopstar } g) \frown h = (h \vee ((g \wedge \text{more}) \frown \text{schopstar } g) \frown h)$
 by (*rule EmptyOrSChopEqvRule*)
have 8: $\vdash ((g \wedge \text{more}) \frown \text{schopstar } g) \frown h = (g \wedge \text{more}) \frown (\text{schopstar } g \frown h)$
 by (*rule SChopAssocB*)
have 9: $\vdash (\text{schopstar } g) \frown h = (h \vee (g \wedge \text{more}) \frown (\text{schopstar } g \frown h))$
 using 7 8 **by** *fastforce*
have 10: $\vdash f \wedge \neg (\text{schopstar } g \frown h) \longrightarrow (g \wedge \text{more}) \frown f \wedge \neg ((g \wedge \text{more}) \frown (\text{schopstar } g \frown h))$
 using 5 9 **by** *fastforce*
have 11: $\vdash g \wedge \text{more} \longrightarrow \text{more}$
 by *fastforce*
from 10 11 **show** ?thesis **using** *SChopContraB* **by** *blast*
qed

lemma *BoxImpTrueSChopAndEmpty*:
 $\vdash \Box f \wedge \text{finite} \longrightarrow \# \text{True} \frown (f \wedge \text{empty})$
by (*metis* *BoxAndSChopImport* *DiamondEmptyEqvFinite* *TrueSChopEqvDiamond* *inteq-reflection*)

lemma *BoxInitAndMoreImpBoxInitAndMoreAndSFinInit*:
 $\vdash \Box (\text{init } w) \wedge \text{more} \wedge \text{finite} \longrightarrow (\Box (\text{init } w) \wedge \text{more}) \wedge \text{sfin } (\text{init } w)$
proof –
have 1: $\vdash \text{sfin } (\text{init } w) = \# \text{True} \frown (\text{init } w \wedge \text{empty})$ **using** *SFinEqvTrueSChopAndEmpty* **by** *blast*
have 2: $\vdash \Box (\text{init } w) \wedge \text{finite} \longrightarrow \# \text{True} \frown (\text{init } w \wedge \text{empty})$ **by** (*rule* *BoxImpTrueSChopAndEmpty*)
from 1 2 **show** ?thesis **by** *fastforce*
qed

lemma *SCSImpBox*:
assumes $\vdash f \longrightarrow \text{empty} \vee ((\Box (\text{init } w) \wedge \text{more}) \frown f)$
shows $\vdash (\text{init } w \wedge f) \wedge \text{finite} \longrightarrow \Box (\text{init } w)$
proof –
have 1: $\vdash f \longrightarrow \text{empty} \vee ((\Box (\text{init } w) \wedge \text{more}) \frown f)$
 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: $\vdash \text{init } w \wedge f \wedge \neg (\Box (\text{init } w)) \longrightarrow ((\Box (\text{init } w) \wedge \text{more}) \frown f)$
 using 1 2 **by** *fastforce*
have 4: $\vdash \Box (\text{init } w) \wedge \text{more} \wedge \text{finite} \longrightarrow (\Box (\text{init } w) \wedge \text{more}) \wedge \text{sfin } (\text{init } w)$

by (rule *BoxInitAndMoreImpBoxInitAndMoreAndSFinInit*)
 have 41: $\vdash (\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite} \longrightarrow ((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfin}(\text{init } w)$
 using 4 by auto
 hence 5: $\vdash ((\Box(\text{init } w) \wedge \text{more}) \frown f \longrightarrow (((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfin}(\text{init } w)) \frown f$
 by (metis *FinitImp LeftChopImpChop inteq-reflection schop-d-def*)
 have 6: $\vdash (((\Box(\text{init } w) \wedge \text{more}) \wedge \text{finite}) \wedge \text{sfin}(\text{init } w)) \frown f =$
 $((\Box(\text{init } w) \wedge \text{more}) \frown (\text{init } w \wedge f))$
 using *AndSFinSChopEqvStateAndSChop* by (smt 5 *AndSChopA Prop11 lift-imp-trans*)
 have 7: $\vdash \neg(\Box(\text{init } w)) \longrightarrow (\Box(\text{init } w)) \text{syields}(\neg(\Box(\text{init } w)))$
 by (rule *NotBoxStateImpBoxSYieldsNotBox*)
 have 8: $\vdash (\Box(\text{init } w)) \text{syields}(\neg(\Box(\text{init } w))) \longrightarrow$
 $((\Box(\text{init } w) \wedge \text{more}) \text{syields}(\neg(\Box(\text{init } w))))$
 using *AndSYieldsA* by (metis)
 have 9: $\vdash ((\Box(\text{init } w) \wedge \text{more}) \frown (\text{init } w \wedge f) \wedge ((\Box(\text{init } w) \wedge \text{more}) \text{syields}(\neg(\Box(\text{init } w))))$
 \longrightarrow
 $((\Box(\text{init } w) \wedge \text{more}) \frown ((\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w))))$
 by (rule *SChopAndSYieldsImp*)
 have 10: $\vdash (\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)) \longrightarrow$
 $((\Box(\text{init } w) \wedge \text{more}) \frown ((\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w))))$
 using 3 5 6 7 8 9 by fastforce
 have 11: $\vdash ((\Box(\text{init } w) \wedge \text{more}) \frown ((\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)))) \longrightarrow$
 $\text{more} \frown ((\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)))$
 using *AndSChopB* by blast
 have 12: $\vdash (\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)) \longrightarrow$
 $\text{more} \frown ((\text{init } w \wedge f) \wedge \neg(\Box(\text{init } w)))$
 using 10 11 by fastforce
 from 12 show ?thesis using *MoreSChopContra* by blast
 qed

lemma *BoxSCSEqvBox*:

$\vdash (\text{init } w \wedge \text{s chopstar}(\Box(\text{init } w))) = (\Box(\text{init } w) \wedge \text{finite})$

proof –

have 1: $\vdash \Box(\text{init } w) \frown (\Box(\text{init } w) \wedge \text{finite}) \longrightarrow \Box(\text{init } w) \wedge \text{finite}$
 by (metis *BoxStateAndChopEqvChop FiniteChopFiniteEqvFinite int-iffD2 inteq-reflection schop-d-def*)
 have 2: $\vdash (\text{init } w \wedge \text{empty}) \longrightarrow \Box(\text{init } w) \wedge \text{finite}$
 using *EmptyImpFinite StateAndEmptyImpBoxState* by fastforce
 have 3: $\vdash (\text{init } w \wedge \text{empty}) \vee \Box(\text{init } w) \frown (\Box(\text{init } w) \wedge \text{finite}) \longrightarrow \Box(\text{init } w) \wedge \text{finite}$
 using 1 2 by fastforce
 have 4: $\vdash (\text{init } w \wedge \text{empty}) \frown \text{s chopstar}(\Box(\text{init } w)) \longrightarrow \Box(\text{init } w) \wedge \text{finite}$
 using *SChopstarInductR* 3
 by (metis (no-types, lifting) *BoxBoxImpBox BoxEqvBoxBox BoxStateSChopBoxEqvBox Prop02 Prop12*
SCSAndFinite SCSImpSCSSCS SChopImpFinite StateAndEmptyImpBoxState int-eq)
 have 5: $\vdash \text{init } w \wedge \text{s chopstar}(\Box(\text{init } w)) \longrightarrow \Box(\text{init } w) \wedge \text{finite}$
 using 4 *StateAndEmptySChop* by fastforce
 have 11: $\vdash \Box(\text{init } w) \longrightarrow (\text{init } w)$
 using *BoxElim* by blast
 have 12: $\vdash \Box(\text{init } w) \wedge \text{finite} \longrightarrow \text{s chopstar}(\Box(\text{init } w))$
 by (rule *ImpSCS*)
 have 13: $\vdash \Box(\text{init } w) \wedge \text{finite} \longrightarrow \text{init } w \wedge \text{s chopstar}(\Box(\text{init } w))$
 using 11 12 by fastforce

from 5 13 show ?thesis by fastforce
qed

lemma *BoxStateAndSCSEqvSCS*:

$\vdash (\Box(\text{init } w) \wedge \text{schopstar } f) = (\text{init } w \wedge \text{schopstar } (\Box(\text{init } w) \wedge f))$

proof –

have 1: $\vdash \Box(\text{init } w) \longrightarrow \text{init } w$

using *BoxElim* **by** *blast*

have 2: $\vdash (\text{schopstar } f \wedge \text{more}) = (f \wedge \text{more}) \frown \text{schopstar } f$

by (*rule SCSAndMoreEqvAndMoreSChop*)

have 3: $\vdash (\Box(\text{init } w) \wedge ((f \wedge \text{more}) \frown \text{schopstar } f)) =$

$((\Box(\text{init } w) \wedge f \wedge \text{more}) \frown (\Box(\text{init } w) \wedge \text{schopstar } f))$

using *BoxStateAndSChopEqvSChop*

by (*smt Prop10 Prop12 SCSAndFinite SChopImpFinite int-iffD1 inteq-reflection lift-and-com*)

have 4: $\vdash \Box(\text{init } w) \wedge f \wedge \text{more} \longrightarrow (\Box(\text{init } w) \wedge f) \wedge \text{more}$

by *auto*

hence 5: $\vdash (\Box(\text{init } w) \wedge f \wedge \text{more}) \frown (\Box(\text{init } w) \wedge \text{schopstar } f) \longrightarrow$

$((\Box(\text{init } w) \wedge f) \wedge \text{more}) \frown (\Box(\text{init } w) \wedge \text{schopstar } f)$

by (*rule LeftSChopImpSChop*)

have 6: $\vdash (\Box(\text{init } w) \wedge \text{schopstar } f) \wedge \text{more} \longrightarrow$

$((\Box(\text{init } w) \wedge f) \wedge \text{more}) \frown (\Box(\text{init } w) \wedge \text{schopstar } f)$

using 2 3 5 **by** *fastforce*

hence 7: $\vdash (\Box(\text{init } w) \wedge \text{schopstar } f) \wedge \text{finite} \longrightarrow \text{schopstar } (\Box(\text{init } w) \wedge f)$

using *SCSIntro* **by** *blast*

have 70: $\vdash \Box(\text{init } w) \wedge \text{schopstar } f \longrightarrow \text{schopstar } (\Box(\text{init } w) \wedge f)$

using *SCSAndFinite* 7 **using** *Valid-def* **by** *fastforce*

have 71: $\vdash \text{init } w \wedge \Box(\text{init } w) \wedge \text{schopstar } f \longrightarrow \text{init } w \wedge \text{schopstar } (\Box(\text{init } w) \wedge f)$

using 70 *SCSAndFinite* **by** *fastforce*

have 8: $\vdash \Box(\text{init } w) \wedge \text{schopstar } f \longrightarrow \text{init } w \wedge \text{schopstar } (\Box(\text{init } w) \wedge f)$

using 1 71 **by** *fastforce*

have 11: $\vdash \text{schopstar } (\Box(\text{init } w) \wedge f) \longrightarrow \text{schopstar } (\Box(\text{init } w))$

by (*rule AndSCSA*)

have 12: $\vdash (\text{init } w \wedge \text{schopstar } (\Box(\text{init } w))) = (\Box(\text{init } w) \wedge \text{finite})$

by (*rule BoxSCSEqvBox*)

have 13: $\vdash \text{schopstar } (\Box(\text{init } w) \wedge f) \longrightarrow \text{schopstar } f$

by (*rule AndSCSB*)

have 14: $\vdash \text{init } w \wedge \text{schopstar } (\Box(\text{init } w) \wedge f) \longrightarrow \text{init } w \wedge \text{schopstar } (\Box(\text{init } w)) \wedge \text{schopstar } f$

using 11 13 **by** *fastforce*

have 15: $\vdash \text{init } w \wedge \text{schopstar } (\Box(\text{init } w)) \wedge \text{schopstar } f \longrightarrow \Box(\text{init } w) \wedge \text{schopstar } f$

using 12 **by** *auto*

have 16: $\vdash \text{init } w \wedge \text{schopstar } (\Box(\text{init } w) \wedge f) \longrightarrow \Box(\text{init } w) \wedge \text{schopstar } f$

using 14 15 *lift-imp-trans* **by** *blast*

from 8 16 show ?thesis **by** *fastforce*

qed

lemma *SBaSCSImpSCS*:

$\vdash \text{sb}a(f \longrightarrow g) \longrightarrow \text{schopstar } f \longrightarrow \text{schopstar } g$

proof –

have 1: $\vdash \text{schopstar } f = (\text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f)$

```

    by (rule SChopstarEqv)
have 2:  $\vdash \text{schopstar } g = (\text{empty} \vee (g \wedge \text{more}) \frown \text{schopstar } g)$ 
    by (rule SChopstarEqv)
have 21:  $\vdash (\neg(\text{schopstar } g)) = (\neg \text{empty} \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g))$ 
    using 2 by fastforce
have 22:  $\vdash (\neg(\text{schopstar } g)) = (\text{more} \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g))$ 
    using NotEmptyEqvMore by fastforce
have 3:  $\vdash \text{schopstar } f \wedge \neg(\text{schopstar } g) \longrightarrow$ 
     $(\text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f) \wedge \text{more} \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g)$ 
    using 1 22 by fastforce
have 31:  $\vdash ((\text{empty} \vee (f \wedge \text{more}) \frown \text{schopstar } f) \wedge \text{more}) = ((f \wedge \text{more}) \frown \text{schopstar } f \wedge \text{more})$ 
    by (auto simp: empty-d-def)
have 32:  $\vdash \text{schopstar } f \wedge \neg(\text{schopstar } g) \longrightarrow$ 
     $(f \wedge \text{more}) \frown \text{schopstar } f \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g)$ 
    using 3 31 by fastforce
have 4:  $\vdash (f \longrightarrow g) \longrightarrow (f \wedge \text{more} \longrightarrow g \wedge \text{more})$ 
    by auto
have 5:  $\vdash \text{sba } (f \longrightarrow g) \longrightarrow \text{sba } (f \wedge \text{more} \longrightarrow g \wedge \text{more})$ 
    by (rule SBaImpSBa)
have 6:  $\vdash \text{sba } (f \wedge \text{more} \longrightarrow g \wedge \text{more}) \longrightarrow$ 
     $(f \wedge \text{more}) \frown \text{schopstar } f \longrightarrow (g \wedge \text{more}) \frown \text{schopstar } f$ 
    by (rule SBaLeftSChopImpSChop)
have 7:  $\vdash \text{sba } (f \longrightarrow g) \wedge (f \wedge \text{more}) \frown \text{schopstar } f \longrightarrow (g \wedge \text{more}) \frown \text{schopstar } f$ 
    using 5 6 by fastforce
have 8:  $\vdash (g \wedge \text{more}) \frown \text{schopstar } f \wedge \neg((g \wedge \text{more}) \frown \text{schopstar } g)$ 
     $\longrightarrow (g \wedge \text{more}) \frown (\text{schopstar } f \wedge \neg(\text{schopstar } g))$ 
    by (rule SChopAndNotSChopImp)
have 9:  $\vdash (g \wedge \text{more}) \frown (\text{schopstar } f \wedge \neg(\text{schopstar } g)) \longrightarrow$ 
     $\text{more} \frown (\text{schopstar } f \wedge \neg(\text{schopstar } g))$ 
    by (rule AndSChopB)
have 10:  $\vdash \text{sba } (f \longrightarrow g) \wedge \text{finite} \longrightarrow \text{more} \frown (\text{schopstar } f \wedge \neg(\text{schopstar } g)) \longrightarrow$ 
     $\text{more} \frown (\text{sba } (f \longrightarrow g) \wedge \text{schopstar } f \wedge \neg(\text{schopstar } g))$ 
    using SBaSChopImpSChopSBa by fastforce
have 11:  $\vdash \text{sba } (f \longrightarrow g) \wedge \text{finite} \wedge \text{schopstar } f \wedge \neg(\text{schopstar } g) \longrightarrow$ 
     $\text{more} \frown (\text{sba } (f \longrightarrow g) \wedge \text{schopstar } f \wedge \neg(\text{schopstar } g))$ 
    using 32 7 8 9 10 by fastforce
have 12:  $\vdash \text{sba } (f \longrightarrow g) \wedge \text{schopstar } f \wedge \neg(\text{schopstar } g) \longrightarrow$ 
     $\text{more} \frown (\text{sba } (f \longrightarrow g) \wedge \text{schopstar } f \wedge \neg(\text{schopstar } g))$ 
    using 11 SCSAndFinite by fastforce
have 12:  $\vdash \text{finite} \longrightarrow \neg(\text{sba } (f \longrightarrow g)) \wedge (\text{schopstar } f) \wedge (\neg(\text{schopstar } g))$ 
    using MoreSChopLoop by blast
have 13:  $\vdash (\text{sba } (f \longrightarrow g)) \wedge \text{finite} \wedge (\text{schopstar } f) \longrightarrow (\text{schopstar } g)$ 
    using 12 by fastforce
have 14:  $\vdash (\text{sba } (f \longrightarrow g)) \wedge (\text{schopstar } f) \longrightarrow (\text{schopstar } g)$ 
    using SCSAndFinite 13 by fastforce
from 14 show ?thesis by fastforce
qed

```

lemma SBaSCSEqvSCS:

$\vdash \text{ sba } (f = g) \longrightarrow (\text{schopstar } f = \text{schopstar } g)$
proof –
have 1: $\vdash \text{ sba } (f = g) = (\text{ sba } (f \longrightarrow g) \wedge \text{ sba } (g \longrightarrow f))$
by (*auto simp: sba-defs sum.case-eq-if*)
have 2: $\vdash \text{ sba } (f \longrightarrow g) \longrightarrow (\text{schopstar } f \longrightarrow \text{schopstar } g)$ **by** (*rule SBaSCSImpSCS*)
have 3: $\vdash \text{ sba } (g \longrightarrow f) \longrightarrow (\text{schopstar } g \longrightarrow \text{schopstar } f)$ **by** (*rule SBaSCSImpSCS*)
have 4: $\vdash \text{ sba } (f = g) \longrightarrow (\text{schopstar } f \longrightarrow \text{schopstar } g) \wedge (\text{schopstar } g \longrightarrow \text{schopstar } f)$
using 1 2 3 **by** *fastforce*
have 5: $\vdash ((\text{schopstar } f \longrightarrow \text{schopstar } g) \wedge (\text{schopstar } g \longrightarrow \text{schopstar } f)) =$
 $(\text{schopstar } f = \text{schopstar } g)$ **by** *auto*
from 4 5 **show** *?thesis* **by** *auto*
qed

lemma *SBaAndSCSImpImport:*

$\vdash \text{ sba } f \wedge \text{schopstar } g \longrightarrow \text{schopstar } (f \wedge g)$
proof –
have 1: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ **by** *auto*
hence 2: $\vdash \text{ sba } f \longrightarrow \text{ sba } (g \longrightarrow f \wedge g)$ **by** (*rule SBaImpSBa*)
have 3: $\vdash \text{ sba } (g \longrightarrow f \wedge g) \longrightarrow \text{schopstar } g \longrightarrow \text{schopstar } (f \wedge g)$ **by** (*rule SBaSCSImpSCS*)
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *SCSSkipImpFinite:*

$\vdash \text{schopstar skip} \longrightarrow \text{finite}$
by (*simp add: EmptyImpFinite SCSElim SChopImpFinite*)

lemma *FinitImpSCSSkip:*

$\vdash \text{finite} \longrightarrow \text{schopstar skip}$
by (*metis (no-types, hide-lams) AndMoreAndFiniteEqvAndFmore ChopAndB ChopEmpty EmptyImpFinite*
FiniteChopSkipEqvFiniteAndMore FiniteChopSkipEqvSkipChopFinite FmoreEqvSkipChopFinite
Prop10 SCSEIntro inteq-reflection lift-and-com chop-d-def)

lemma *SCSSkipEqvFinite:*

$\vdash \text{schopstar skip} = \text{finite}$
using *SCSSkipImpFinite FinitImpSCSSkip* **by** *fastforce*

10.8 Properties of Omega

lemma *SOmegaIntro:*

assumes $\vdash h \longrightarrow (f \wedge \text{more}) \frown h$
shows $\vdash h \wedge \text{inf} \longrightarrow f^\omega$
proof –
have 1: $\vdash h \longrightarrow (f \wedge \text{more}) \frown h$ **using** *assms* **by** *auto*
have 2: $\vdash \Box (h \longrightarrow (f \wedge \text{more}) \frown h)$ **by** (*simp add: BoxGen assms*)
from 1 2 **show** *?thesis* **using** *SOmegaInduct* **by** *fastforce*
qed

10.9 Properties of SWhile

lemma *SWhileEqvIf*:

$\vdash \text{swhile } (\text{init } w) \text{ do } f = \text{if}_i (\text{init } w) \text{ then } (f \frown (\text{swhile } (\text{init } w) \text{ do } f)) \text{ else empty}$

proof –

have 1: $\vdash \text{swhile } (\text{init } w) \text{ do } f = (\text{s chopstar } ((\text{init } w) \wedge f) \wedge \text{sfin } (\neg (\text{init } w)))$
by (*simp add: swhile-d-def*)

have 2: $\vdash \text{s chopstar } (\text{init } w \wedge f) = (\text{empty} \vee ((\text{init } w \wedge f) \frown \text{s chopstar } (\text{init } w \wedge f)))$
by (*rule SCSEqvOrSChopSCS*)

have 21: $\vdash (\text{s chopstar } ((\text{init } w) \wedge f) \wedge \text{sfin } (\neg (\text{init } w))) =$
 $((\text{empty} \vee ((\text{init } w \wedge f) \frown \text{s chopstar } (\text{init } w \wedge f))) \wedge \text{sfin } (\neg (\text{init } w)))$
using 2 **by** *fastforce*

have 22: $\vdash ((\text{empty} \vee ((\text{init } w \wedge f) \frown \text{s chopstar } (\text{init } w \wedge f))) \wedge \text{sfin } (\neg (\text{init } w))) =$
 $((\text{empty} \wedge \text{sfin } (\neg (\text{init } w))) \vee$
 $((\text{init } w \wedge f) \frown \text{s chopstar } (\text{init } w \wedge f)) \wedge \text{sfin } (\neg (\text{init } w))))$
by *auto*

have 3: $\vdash (\text{empty} \wedge \text{sfin } (\neg (\text{init } w))) = (\neg (\text{init } w) \wedge \text{empty})$
by (*metis Prop04 SFinAndEmpty lift-and-com*)

have 4: $\vdash (\text{init } w \wedge f) \frown \text{s chopstar } (\text{init } w \wedge f) = (\text{init } w \wedge (f \frown \text{s chopstar } (\text{init } w \wedge f)))$
by (*rule StateAndSChop*)

have 41: $\vdash (((\text{init } w \wedge f) \frown \text{s chopstar } (\text{init } w \wedge f)) \wedge \text{sfin } (\neg (\text{init } w))) =$
 $(\text{init } w \wedge (f \frown \text{s chopstar } (\text{init } w \wedge f)) \wedge \text{sfin } (\neg (\text{init } w)))$
using 4 **by** *auto*

have 42: $\vdash (\text{init } w \wedge (f \frown \text{s chopstar } (\text{init } w \wedge f)) \wedge \text{sfin } (\neg (\text{init } w))) =$
 $(\text{init } w \wedge (f \frown \text{s chopstar } (\text{init } w \wedge f)) \wedge \text{sfin } (\text{init } (\neg w)))$
using *Initprop(2)* **by** (*metis StateAndEmptySChop int-eq*)

have 5: $\vdash ((f \frown (\text{s chopstar } (\text{init } w \wedge f))) \wedge (\text{sfin } (\text{init } (\neg w))))$
 $= (f \frown (\text{s chopstar } (\text{init } w \wedge f) \wedge (\text{sfin } (\text{init } (\neg w))))$
by (*rule SChopAndSFin*)

have 51: $\vdash (f \frown (\text{s chopstar } (\text{init } w \wedge f) \wedge (\text{sfin } (\text{init } (\neg w)))) =$
 $(f \frown (\text{s chopstar } (\text{init } w \wedge f) \wedge (\text{sfin } (\neg (\text{init } w))))$
using *Initprop(2)* **by** (*smt RightSChopEqvSChop int-eq lift-and-com*)

have 52: $\vdash (\text{init } w \wedge (f \frown \text{s chopstar } (\text{init } w \wedge f)) \wedge \text{sfin } (\neg (\text{init } w))) =$
 $(\text{init } w \wedge (f \frown (\text{s chopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w))))$
using 42 5 51 **by** *fastforce*

have 6: $\vdash (f \frown (\text{s chopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w)))) = f \frown \text{swhile } (\text{init } w) \text{ do } f$
by (*simp add: swhile-d-def*)

have 61: $\vdash (\text{init } w \wedge (f \frown (\text{s chopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w)))) =$
 $(\text{init } w \wedge (f \frown \text{swhile } (\text{init } w) \text{ do } f))$ **using** 6
by *auto*

have 62: $\vdash (\text{empty} \wedge \text{sfin } (\neg (\text{init } w))) \vee$
 $((\text{init } w \wedge f) \frown \text{s chopstar } (\text{init } w \wedge f)) \wedge \text{sfin } (\neg (\text{init } w)))$
 $= (\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f \frown \text{swhile } (\text{init } w) \text{ do } f))$
using 21 22 3 4 52 61 **by** *fastforce*

have 7: $\vdash \text{swhile } (\text{init } w) \text{ do } f$
 $= ((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f \frown \text{swhile } (\text{init } w) \text{ do } f)))$
using 1 21 22 62
by (*metis 3 41 42 5 51 inteq-reflection*)

have 71: $\vdash \text{if}_i (\text{init } w) \text{ then } (f \frown (\text{swhile } (\text{init } w) \text{ do } f)) \text{ else empty} =$
 $((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f \frown \text{swhile } (\text{init } w) \text{ do } f)))$
by (*auto simp: ifthenelse-d-def*)

from 7 71 show ?thesis by fastforce
qed

lemma *SWhileSChopEqvIf*:

$\vdash (\text{swhile } (\text{init } w) \text{ do } f) \frown g = \text{if}_i (\text{init } w) \text{ then } (f \frown ((\text{swhile } (\text{init } w) \text{ do } f) \frown g)) \text{ else } g$
proof –
have 1: $\vdash \text{swhile } (\text{init } w) \text{ do } f =$
 $\text{if}_i (\text{init } w) \text{ then } (f \frown (\text{swhile } (\text{init } w) \text{ do } f)) \text{ else } \text{empty}$
by (rule *SWhileEqvIf*)
hence 2: $\vdash (\text{swhile } (\text{init } w) \text{ do } f) \frown g =$
 $\text{if}_i (\text{init } w) \text{ then } ((f \frown \text{swhile } (\text{init } w) \text{ do } f) \frown g) \text{ else } (\text{empty} \frown g)$
by (rule *IfSChopEqvRule*)
have 3: $\vdash \text{empty} \frown g = g$
by (rule *EmptySChop*)
have 4: $\vdash \text{if}_i (\text{init } w) \text{ then } ((f \frown \text{swhile } (\text{init } w) \text{ do } f) \frown g) \text{ else } (\text{empty} \frown g) =$
 $\text{if}_i (\text{init } w) \text{ then } ((f \frown \text{swhile } (\text{init } w) \text{ do } f) \frown g) \text{ else } g$
using 3 **using** *inteq-reflection* **by** *fastforce*
have 5: $\vdash ((f \frown \text{swhile } (\text{init } w) \text{ do } f) \frown g) = (f \frown (\text{swhile } (\text{init } w) \text{ do } f \frown g))$
by (rule *SChopAssocB*)
have 6: $\vdash \text{if}_i (\text{init } w) \text{ then } ((f \frown \text{swhile } (\text{init } w) \text{ do } f) \frown g) \text{ else } g =$
 $\text{if}_i (\text{init } w) \text{ then } (f \frown ((\text{swhile } (\text{init } w) \text{ do } f) \frown g)) \text{ else } g$
using 5 **using** *inteq-reflection* **by** *fastforce*
from 1 2 4 6 **show** ?thesis **by** *fastforce*
qed

lemma *SWhileSChopEqvIfRule*:

assumes $\vdash f = (\text{swhile } (\text{init } w) \text{ do } g) \frown h$
shows $\vdash f = \text{if}_i (\text{init } w) \text{ then } (g \frown f) \text{ else } h$
proof –
have 1: $\vdash f = (\text{swhile } (\text{init } w) \text{ do } g) \frown h$
using *assms* **by** *auto*
have 2: $\vdash (\text{swhile } (\text{init } w) \text{ do } g) \frown h =$
 $\text{if}_i (\text{init } w) \text{ then } (g \frown ((\text{swhile } (\text{init } w) \text{ do } g) \frown h)) \text{ else } h$
by (rule *SWhileSChopEqvIf*)
have 3: $\vdash (g \frown f) = (g \frown ((\text{swhile } (\text{init } w) \text{ do } g) \frown h))$
using 1 **by** (rule *RightSChopEqvSChop*)
have 4: $\vdash (g \frown ((\text{swhile } (\text{init } w) \text{ do } g) \frown h)) = (g \frown f)$
using 3 **by** *auto*
have 5: $\vdash \text{if}_i (\text{init } w) \text{ then } (g \frown ((\text{swhile } (\text{init } w) \text{ do } g) \frown h)) \text{ else } h =$
 $\text{if}_i (\text{init } w) \text{ then } (g \frown f) \text{ else } h$
using 4 **using** *inteq-reflection* **by** *fastforce*
from 1 2 5 **show** ?thesis **by** *fastforce*
qed

lemma *WhileImpFin*:

$\vdash \text{while } (\text{init } w) \text{ do } f \longrightarrow \text{fin } (\neg (\text{init } w))$
proof –
have 1: $\vdash (\text{init } w \wedge f)^* \wedge \text{fin } (\neg (\text{init } w)) \longrightarrow \text{fin } (\neg (\text{init } w))$ **by** *auto*
from 1 **show** ?thesis **by** (*simp add: while-d-def*)

qed

lemma *SWhileEqvEmptyOrSChopSWhile*:

$\vdash \text{swhile } (\text{init } w) \text{ do } f = ((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (f \wedge \text{more}) \curvearrowright \text{swhile } (\text{init } w) \text{ do } f))$

proof –

have 1: $\vdash \text{s chopstar } (\text{init } w \wedge f) = (\text{empty} \vee ((\text{init } w \wedge f) \wedge \text{more}) \curvearrowright \text{s chopstar } (\text{init } w \wedge f))$

by (rule *SChopstarEqv*)

have 2: $\vdash ((\text{init } w \wedge f) \wedge \text{more}) = (\text{init } w \wedge (f \wedge \text{more}))$

by *auto*

hence 3: $\vdash ((\text{init } w \wedge f) \wedge \text{more}) \curvearrowright \text{s chopstar } (\text{init } w \wedge f) =$

$(\text{init } w \wedge f \wedge \text{more}) \curvearrowright \text{s chopstar } (\text{init } w \wedge f)$

by (rule *LeftSChopEqvSChop*)

have 4: $\vdash \text{s chopstar } (\text{init } w \wedge f) = (\text{empty} \vee (\text{init } w \wedge f \wedge \text{more}) \curvearrowright \text{s chopstar } (\text{init } w \wedge f))$

using 1 3 **by** *fastforce*

have 5: $\vdash (\text{s chopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w))) =$

$((\text{empty} \wedge \text{sfin } (\neg (\text{init } w))) \vee$

$((\text{init } w \wedge f \wedge \text{more}) \curvearrowright \text{s chopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w))))$

using 1 4 **by** *fastforce*

have 6: $\vdash (\text{empty} \wedge \text{sfin } (\neg (\text{init } w))) = (\neg (\text{init } w) \wedge \text{empty})$

by (*meson Prop04 SFinAndEmpty lift-and-com*)

have 7: $\vdash (\text{init } w \wedge f \wedge \text{more}) \curvearrowright \text{s chopstar } (\text{init } w \wedge f) =$

$(\text{init } w \wedge (f \wedge \text{more}) \curvearrowright \text{s chopstar } (\text{init } w \wedge f))$

by (rule *StateAndSChop*)

have 8: $\vdash (((f \wedge \text{more}) \curvearrowright \text{s chopstar } (\text{init } w \wedge f)) \wedge \text{sfin } (\text{init } (\neg w))) =$

$((f \wedge \text{more}) \curvearrowright (\text{s chopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\text{init } (\neg w))))$

by (rule *SChopAndSFin*)

have 81: $\vdash \text{sfin } (\text{init } (\neg w)) = \text{sfin } (\neg (\text{init } w))$

by (*metis Initprop(2) SFinStateEqvStateAndEmptyOrNextSFinState inteq-reflection*)

have 82: $\vdash ((f \wedge \text{more}) \curvearrowright \text{s chopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w))) =$

$((f \wedge \text{more}) \curvearrowright (\text{s chopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w))))$

using 8 81

by (*metis inteq-reflection*)

have 9: $\vdash (\text{s chopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w))) =$

$((\neg (\text{init } w) \wedge \text{empty}) \vee$

$(\text{init } w \wedge (f \wedge \text{more}) \curvearrowright (\text{s chopstar } (\text{init } w \wedge f) \wedge \text{sfin } (\neg (\text{init } w))))$

using 5 6 7 82 **by** *fastforce*

from 9 **show** *?thesis* **by** (*simp add: swhile-d-def*)

qed

lemma *SWhileIntro*:

assumes $\vdash \neg (\text{init } w) \wedge f \longrightarrow \text{empty}$

$\vdash \text{init } w \wedge f \longrightarrow (g \wedge \text{more}) \curvearrowright f$

shows $\vdash f \wedge \text{finite} \longrightarrow \text{swhile } (\text{init } w) \text{ do } g$

proof –

have 1: $\vdash \neg (\text{init } w) \wedge f \longrightarrow \text{empty}$

using *assms* **by** *blast*

have 2: $\vdash \text{init } w \wedge f \longrightarrow (g \wedge \text{more}) \curvearrowright f$

using *assms* **by** *blast*

have 3: $\vdash \text{swhile } (\text{init } w) \text{ do } g =$

$((\neg (\text{init } w) \wedge \text{empty}) \vee (\text{init } w \wedge (g \wedge \text{more}) \curvearrowright \text{swhile } (\text{init } w) \text{ do } g))$

by (rule SWhileEqvEmptyOrSChopSWhile)
 hence 31: $\vdash \neg (\text{swhile } (init\ w) \text{ do } g) =$
 $(\neg (\neg (init\ w) \wedge \text{empty}) \vee (init\ w \wedge (g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g)))$
 by fastforce
 hence 32: $\vdash (f \wedge \neg (\text{swhile } (init\ w) \text{ do } g)) =$
 $(f \wedge \neg (\neg (init\ w) \wedge \text{empty}) \vee (init\ w \wedge (g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g)))$
 by fastforce
 have 33: $\vdash (f \wedge \neg (\neg (init\ w) \wedge \text{empty}) \vee (init\ w \wedge (g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g))) =$
 $(f \wedge \neg (\neg (init\ w) \wedge \text{empty}) \wedge \neg (init\ w \wedge (g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g)))$
 by auto
 have 34: $\vdash (f \wedge \neg (\neg (init\ w) \wedge \text{empty}) \wedge \neg ((init\ w) \wedge ((g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g))) =$
 $(f \wedge ((init\ w) \vee \text{more}) \wedge (\neg (init\ w) \vee \neg ((g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g))))$
 by (auto simp: empty-d-def)
 have 35: $\vdash (f \wedge ((init\ w) \vee \text{more}) \wedge (\neg (init\ w) \vee \neg ((g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g))) =$
 $((f \wedge (init\ w) \wedge \neg ((g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g)) \vee$
 $(f \wedge (init\ w) \wedge \neg (init\ w)) \vee$
 $(f \wedge \text{more} \wedge \neg ((g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g)) \vee$
 $(f \wedge \text{more} \wedge \neg (init\ w)))$
 by auto
 have 36: $\vdash (f \wedge \neg (\text{swhile } (init\ w) \text{ do } g)) =$
 $((f \wedge (init\ w) \wedge \neg ((g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g)) \vee$
 $(f \wedge (init\ w) \wedge \neg (init\ w)) \vee$
 $(f \wedge \text{more} \wedge \neg ((g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g)) \vee$
 $(f \wedge \text{more} \wedge \neg (init\ w)))$ using 32 33 34 35 by fastforce
 have 37: $\vdash \neg (f \wedge \text{more} \wedge \neg (init\ w))$
 using 1 by (auto simp: empty-d-def)
 have 38: $\vdash (f \wedge \text{more} \wedge \neg ((g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g)) \longrightarrow$
 $((g \wedge \text{more}) \frown f \wedge \neg ((g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g))$
 using 1 2 by (auto simp: empty-d-def Valid-def)
 have 39: $\vdash (f \wedge (init\ w) \wedge \neg ((g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g)) \longrightarrow$
 $((g \wedge \text{more}) \frown f \wedge \neg ((g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g))$
 using 2 by auto
 have 40: $\vdash ((f \wedge (init\ w) \wedge \neg ((g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g)) \vee$
 $(f \wedge (init\ w) \wedge \neg (init\ w)) \vee$
 $(f \wedge \text{more} \wedge \neg ((g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g)) \vee$
 $(f \wedge \text{more} \wedge \neg (init\ w))) \longrightarrow$
 $(g \wedge \text{more}) \frown f \wedge \neg ((g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g)$
 using 39 38 37 38 by fastforce
 have 4: $\vdash f \wedge \neg (\text{swhile } (init\ w) \text{ do } g) \longrightarrow$
 $(g \wedge \text{more}) \frown f \wedge \neg ((g \wedge \text{more}) \frown \text{swhile } (init\ w) \text{ do } g)$
 using 36 40 by fastforce
 have 5: $\vdash g \wedge \text{more} \longrightarrow \text{more}$
 by auto
 from 4 5 show ?thesis using SChopContraB by blast
 qed

lemma SWhileElim:

assumes $\vdash \neg (init\ w) \wedge \text{empty} \longrightarrow g$
 $\vdash init\ w \wedge (f \wedge \text{more}) \frown g \longrightarrow g$
 shows $\vdash \text{swhile } (init\ w) \text{ do } f \longrightarrow g$

proof –

have 1: $\vdash \text{swwhile } (init\ w) \text{ do } f =$
 $((\neg (init\ w) \wedge \text{empty}) \vee (init\ w \wedge (f \wedge \text{more}) \frown \text{swwhile } (init\ w) \text{ do } f))$
by (rule *SWhileEqvEmptyOrSChopSWhile*)
hence 11: $\vdash ((\text{swwhile } (init\ w) \text{ do } f) \wedge \neg g) =$
 $((\neg (init\ w) \wedge \text{empty}) \vee (init\ w \wedge (f \wedge \text{more}) \frown \text{swwhile } (init\ w) \text{ do } f)) \wedge \neg g$
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: $\vdash ((\neg (init\ w) \wedge \text{empty}) \vee (init\ w \wedge (f \wedge \text{more}) \frown \text{swwhile } (init\ w) \text{ do } f)) \wedge \neg g \longrightarrow$
 $(init\ w \wedge (f \wedge \text{more}) \frown \text{swwhile } (init\ w) \text{ do } f)$
using 21 **by** *auto*
have 23: $\vdash (\text{swwhile } (init\ w) \text{ do } f) \wedge \neg g \longrightarrow$
 $(init\ w \wedge (f \wedge \text{more}) \frown \text{swwhile } (init\ w) \text{ do } f) \wedge \neg g$
using 11 21 **by** *fastforce*
have 3: $\vdash (init\ w) \wedge ((f \wedge \text{more}) \frown g) \longrightarrow g$
using *assms* **by** *blast*
hence 31: $\vdash \neg g \longrightarrow \neg ((init\ w) \wedge ((f \wedge \text{more}) \frown g))$
by *fastforce*
have 32: $\vdash (init\ w \wedge (f \wedge \text{more}) \frown \text{swwhile } (init\ w) \text{ do } f) \wedge \neg g \longrightarrow$
 $((f \wedge \text{more}) \frown (\text{swwhile } (init\ w) \text{ do } f)) \wedge \neg ((f \wedge \text{more}) \frown g) \wedge \neg g$
using 31 **by** *auto*
have 4: $\vdash (\text{swwhile } (init\ w) \text{ do } f) \wedge \neg g \longrightarrow$
 $((f \wedge \text{more}) \frown (\text{swwhile } (init\ w) \text{ do } f)) \wedge \neg ((f \wedge \text{more}) \frown g)$
using 23 32 **by** *fastforce*
have 5: $\vdash f \wedge \text{more} \longrightarrow \text{more}$
by *auto*
have 6: $\vdash (\text{swwhile } (init\ w) \text{ do } f) \wedge \text{finite} \longrightarrow g$
using *ChopContraB* 4 5 **using** *SChopContraB* **by** *blast*
have 7: $\vdash ((\text{swwhile } (init\ w) \text{ do } f) \wedge \text{finite}) = (\text{swwhile } (init\ w) \text{ do } f)$
using *swwhile-d-def*
by (metis 1 *DiamondEmptyEqvFinite Prop10 Prop11 Prop12 SChopAndB SFinEqvTrueSChopAndEmpty*
TrueSChopEqvDiamond lift-imp-trans)
from 6 7 **show** ?thesis **by** *fastforce*
qed

lemma *SBaSWhileImpSWhile*:

$\vdash \text{sba } (f \longrightarrow g) \longrightarrow (\text{swwhile } (init\ w) \text{ do } f) \longrightarrow (\text{swwhile } (init\ w) \text{ do } g)$

proof –

have 1: $\vdash (f \longrightarrow g) \longrightarrow ((init\ w \wedge f) \longrightarrow (init\ w \wedge g))$
by *auto*
hence 2: $\vdash \text{sba } (f \longrightarrow g) \longrightarrow \text{sba } ((init\ w \wedge f) \longrightarrow (init\ w \wedge g))$
by (rule *SBaImpSBa*)
have 3: $\vdash \text{sba } ((init\ w \wedge f) \longrightarrow (init\ w \wedge g)) \longrightarrow$
 $(\text{s chopstar } (init\ w \wedge f) \longrightarrow \text{s chopstar } (init\ w \wedge g))$
by (rule *SBaSCSImpSCS*)
have 4: $\vdash \text{sba } (f \longrightarrow g) \longrightarrow (\text{s chopstar } (init\ w \wedge f) \wedge \text{sfin } (\neg (init\ w))) \longrightarrow$
 $\text{s chopstar } (init\ w \wedge g) \wedge \text{sfin } (\neg (init\ w)))$

using 2 3 by fastforce
 from 4 show ?thesis by (simp add: swhile-d-def)
 qed

lemma *SWhileImpSWhile*:

assumes $\vdash f \longrightarrow g$
 shows $\vdash (\text{swhile } (\text{init } w) \text{ do } f) \longrightarrow (\text{swhile } (\text{init } w) \text{ do } g)$
 proof –
 have 1: $\vdash f \longrightarrow g$
 using assms by auto
 hence 2: $\vdash \text{sba } (f \longrightarrow g)$
 by (rule SBaGen)
 have 3: $\vdash \text{sba } (f \longrightarrow g) \longrightarrow (\text{swhile } (\text{init } w) \text{ do } f) \longrightarrow (\text{swhile } (\text{init } w) \text{ do } g)$
 by (rule SBaSWhileImpSWhile)
 from 2 3 show ?thesis using MP by blast
 qed

10.10 Properties of Halt

lemma *HaltSCHopEqv*:

$\vdash ((\text{halt } (\text{init } w)) \frown f) = (\text{if}_i (\text{init } w) \text{ then } (f) \text{ else } (\bigcirc(\text{halt } (\text{init } w)) \frown f))$
 proof –
 have 1: $\vdash \text{halt}(\text{init } w) =$
 $(\text{if}_i (\text{init } w) \text{ then } \text{empty} \text{ else } (\bigcirc(\text{halt } (\text{init } w))))$
 by (rule HaltStateEqvIfStateThenEmptyElseNext)
 hence 2: $\vdash ((\text{halt}(\text{init } w)) \frown f) =$
 $(\text{if}_i (\text{init } w) \text{ then } (\text{empty} \frown f) \text{ else } (\bigcirc(\text{halt } (\text{init } w)) \frown f))$
 by (rule IfSCHopEqvRule)
 have 3: $\vdash \text{empty} \frown f = f$
 by (rule EmptySCHop)
 have 4: $\vdash (\bigcirc(\text{halt } (\text{init } w))) \frown f = \bigcirc(\text{halt } (\text{init } w) \frown f)$
 by (rule NextSCHop)
 from 2 3 4 show ?thesis by (metis inteq-reflection)
 qed

lemma *AndHaltSCHopImp*:

$\vdash \text{init } w \wedge (\text{halt } (\text{init } w) \frown f) \longrightarrow f$
 proof –
 have 1: $\vdash \text{halt } (\text{init } w) \frown f = \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt } (\text{init } w) \frown f))$
 by (rule HaltSCHopEqv)
 have 2: $\vdash \text{init } w \wedge \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt } (\text{init } w) \frown f)) \longrightarrow f$
 by (auto simp: ifthenelse-d-def)
 from 1 2 show ?thesis by fastforce
 qed

lemma *NotAndHaltSCHopImpNext*:

$\vdash \neg (\text{init } w) \wedge (\text{halt } (\text{init } w) \frown f) \longrightarrow \bigcirc(\text{halt } (\text{init } w) \frown f)$
 proof –
 have 1: $\vdash \text{halt } (\text{init } w) \frown f = \text{if}_i (\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt } (\text{init } w) \frown f))$
 by (rule HaltSCHopEqv)

have 2: $\vdash \neg (init\ w) \wedge if_i\ (init\ w)\ then\ f\ else\ (\bigcirc(halt\ (init\ w) \frown f)) \longrightarrow$
 $\bigcirc(halt\ (init\ w) \frown f)$
by (*auto simp: ifthenelse-d-def*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *NotAndHaltSChopImpSkipSYields*:
 $\vdash \neg (init\ w) \wedge (halt\ (init\ w) \frown f) \longrightarrow skip\ syields\ (halt\ (init\ w) \frown f)$
proof –
have 1: $\vdash \neg (init\ w) \wedge (halt\ (init\ w) \frown f) \longrightarrow \bigcirc(halt\ (init\ w) \frown f)$
by (*rule NotAndHaltSChopImpNext*)
have 2: $\vdash \bigcirc(halt\ (init\ w) \frown f) \longrightarrow skip\ syields\ (halt\ (init\ w) \frown f)$
by (*rule NextImpSkipSYields*)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma *SChopAndEmptyEqvSChopAndEmpty*:
 $\vdash ((\#True \frown (f \wedge empty)) \wedge g) = (g \frown (f \wedge empty))$
by (*smt AndSFinEqvSChopAndEmpty Prop11 Prop12 SFinEqvTrueSChopAndEmpty inteq-reflection lift-and-com*)

lemma *NotSChopSkipEqvFmoreAndNotSChopSkip*:
 $\vdash (\neg f) \frown skip = (fmore \wedge \neg(f \frown skip))$
proof –
have 1: $\vdash (\neg f) \frown skip = ((\neg f \wedge finite); skip)$
by (*simp add: schop-d-def*)
have 2: $\vdash (\neg f \wedge finite); skip = (\neg(f \vee inf)); skip$
by (*metis (no-types, lifting) LeftChopEqvChop finite-d-def int-simps(14) int-simps(33) inteq-reflection*)
have 3: $\vdash (\neg(f \vee inf)); skip = (more \wedge \neg((f \vee inf); skip))$
using *NotChopSkipEqvMoreAndNotChopSkip* **by** blast
have 4: $\vdash (f \vee inf); skip = (f; skip \vee inf)$
by (*metis AndInfChopEqvAndInf MoreAndInfEqvInf OrChopEqv inteq-reflection*)
have 5: $\vdash (more \wedge \neg((f \vee inf); skip)) = (more \wedge \neg(f; skip \vee inf))$
using 4 **by** auto
have 6: $\vdash (more \wedge \neg(f; skip \vee inf)) = (more \wedge \neg(f; skip) \wedge finite)$
using *finite-d-def*
by (*metis 3 4 int-simps(14) int-simps(33) inteq-reflection*)
have 7: $\vdash (more \wedge \neg(f; skip) \wedge finite) = (more \wedge \neg(f \frown skip \vee (f \wedge inf)) \wedge finite)$
using *ChopSChopdef* **by** fastforce
have 8: $\vdash (more \wedge \neg(f \frown skip \vee (f \wedge inf)) \wedge finite) =$
 $(more \wedge \neg(f \frown skip) \wedge \neg(f \wedge inf) \wedge finite)$
by auto
have 9: $\vdash (\neg(f \wedge inf) \wedge finite) = finite$
using *finite-d-def*
by (*metis (no-types, lifting) AndInfChopAndInfEqvAndInf AndInfEqvChopFalse ChopAndB ChopLoopB FiniteChopMoreEqvMore NotEmptyEqvMore Prop10 RightChopImpMoreRule int-simps(21) inteq-reflection lift-and-com*)
have 10: $\vdash (more \wedge \neg(f \frown skip) \wedge \neg(f \wedge inf) \wedge finite) =$
 $(more \wedge \neg(f \frown skip) \wedge finite)$

using 9 **by** fastforce
have 11: $\vdash (\text{more} \wedge \neg(f \frown \text{skip}) \wedge \text{finite}) = (\text{fmore} \wedge \neg(f \frown \text{skip}))$
using fmore-d-def
by (metis Prop11 Prop12 lift-and-com)
from 1 2 3 5 6 7 8 10 11 **show** ?thesis **by** (metis inteq-reflection)
qed

lemma HaltSCHopImpNotHaltSCHopNot:

$\vdash \text{halt}(\text{init } w) \frown f \wedge \text{finite} \longrightarrow \neg(\text{halt}(\text{init } w) \frown (\neg f))$
proof –
have 1: $\vdash \text{halt}(\text{init } w) \frown f = \text{if}_i(\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt}(\text{init } w) \frown f))$
by (rule HaltSCHopEqv)
have 2: $\vdash \text{if}_i(\text{init } w) \text{ then } f \text{ else } (\bigcirc(\text{halt}(\text{init } w) \frown f)) \longrightarrow$
 $(((\text{init } w) \longrightarrow f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt}(\text{init } w) \frown f))))$
by (rule IfThenElseImp)
have 3: $\vdash \text{halt}(\text{init } w) \frown (\neg f) =$
 $\text{if}_i(\text{init } w) \text{ then } (\neg f) \text{ else } (\bigcirc(\text{halt}(\text{init } w) \frown (\neg f)))$
by (rule HaltSCHopEqv)
have 4: $\vdash \text{if}_i(\text{init } w) \text{ then } (\neg f) \text{ else } (\bigcirc(\text{halt}(\text{init } w) \frown (\neg f))) \longrightarrow$
 $((\text{init } w) \longrightarrow \neg f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt}(\text{init } w) \frown (\neg f))))$
by (rule IfThenElseImp)
have 5: $\vdash \text{halt}(\text{init } w) \frown f \wedge \text{halt}(\text{init } w) \frown (\neg f) \longrightarrow$
 $((\text{init } w) \longrightarrow f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt}(\text{init } w) \frown f))) \wedge$
 $((\text{init } w) \longrightarrow \neg f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt}(\text{init } w) \frown (\neg f))))$
using 1 2 3 4 **by** fastforce
have 6: $\vdash ((\text{init } w) \longrightarrow f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt}(\text{init } w) \frown f))) \wedge$
 $((\text{init } w) \longrightarrow \neg f) \wedge (\neg(\text{init } w) \longrightarrow (\bigcirc(\text{halt}(\text{init } w) \frown (\neg f)))) \longrightarrow$
 $(\bigcirc(\text{halt}(\text{init } w) \frown f)) \wedge (\bigcirc(\text{halt}(\text{init } w) \frown (\neg f)))$
by auto
have 7: $\vdash \text{halt}(\text{init } w) \frown f \wedge \text{halt}(\text{init } w) \frown (\neg f) \longrightarrow$
 $(\bigcirc(\text{halt}(\text{init } w) \frown f)) \wedge (\bigcirc(\text{halt}(\text{init } w) \frown (\neg f)))$
using 5 6 lift-imp-trans **by** blast
have 8: $\vdash ((\bigcirc(\text{halt}(\text{init } w) \frown f)) \wedge (\bigcirc(\text{halt}(\text{init } w) \frown (\neg f)))) =$
 $\bigcirc(\text{halt}(\text{init } w) \frown f \wedge \text{halt}(\text{init } w) \frown (\neg f))$
using NextAndEqvNextAndNext **by** fastforce
have 9: $\vdash \text{halt}(\text{init } w) \frown f \wedge \text{halt}(\text{init } w) \frown (\neg f) \longrightarrow$
 $\bigcirc(\text{halt}(\text{init } w) \frown f \wedge \text{halt}(\text{init } w) \frown (\neg f))$
using 7 8 **by** fastforce
hence 10: $\vdash \text{finite} \longrightarrow \neg(\text{halt}(\text{init } w) \frown f \wedge \text{halt}(\text{init } w) \frown (\neg f))$
using NextLoop **by** blast
from 10 **show** ?thesis **by** auto
qed

lemma HaltSCHopImpHaltSYields:

$\vdash \text{halt}(\text{init } w) \frown f \wedge \text{finite} \longrightarrow (\text{halt}(\text{init } w)) \text{ syields } f$
proof –
have 1: $\vdash \text{halt}(\text{init } w) \frown f \wedge \text{finite} \longrightarrow \neg(\text{halt}(\text{init } w) \frown (\neg f))$
by (rule HaltSCHopImpNotHaltSCHopNot)
from 1 **show** ?thesis **by** (simp add: syields-d-def)
qed

lemma *HaltSCHopAnd*:

$\vdash (\text{halt } (\text{init } w)) \frown f \wedge (\text{halt } (\text{init } w)) \frown g \wedge \text{finite} \longrightarrow (\text{halt } (\text{init } w)) \frown (f \wedge g)$

proof –

have 1: $\vdash (\text{halt } (\text{init } w)) \frown g \wedge \text{finite} \longrightarrow (\text{halt } (\text{init } w)) \text{ syields } g$
by (rule *HaltSCHopImpHaltSYields*)

hence 2: $\vdash (\text{halt } (\text{init } w)) \frown f \wedge (\text{halt } (\text{init } w)) \frown g \wedge \text{finite} \longrightarrow$
 $(\text{halt } (\text{init } w)) \frown f \wedge (\text{halt } (\text{init } w)) \text{ syields } g$ **by** *auto*

have 3: $\vdash (\text{halt } (\text{init } w)) \frown f \wedge (\text{halt } (\text{init } w)) \text{ syields } g \longrightarrow$
 $(\text{halt } (\text{init } w)) \frown (f \wedge g)$ **by** (rule *SCHopAndSYieldsImp*)

from 2 3 **show** ?thesis **by** *fastforce*

qed

lemma *HaltAndSCHopAndHaltSCHopImpHaltAndSCHopAnd*:

$\vdash (\text{halt } (\text{init } w) \wedge f) \frown f1 \wedge (\text{halt } (\text{init } w) \frown g) \wedge \text{finite}$
 $\longrightarrow (\text{halt } (\text{init } w) \wedge f) \frown (f1 \wedge g)$

proof –

have 1: $\vdash f1 \longrightarrow \neg g \vee (f1 \wedge g)$
by *auto*

hence 2: $\vdash (\text{halt } (\text{init } w) \wedge f) \frown f1 \longrightarrow$
 $(\text{halt } (\text{init } w) \wedge f) \frown (\neg g) \vee ((\text{halt } (\text{init } w) \wedge f) \frown (f1 \wedge g))$
by (rule *SCHopOrImpRule*)

have 3: $\vdash (\text{halt } (\text{init } w) \wedge f) \frown (\neg g) \longrightarrow \text{halt } (\text{init } w) \frown (\neg g)$
by (rule *AndSCHopA*)

have 31: $\vdash (\text{halt } (\text{init } w) \wedge f) \frown f1 \longrightarrow$
 $\text{halt } (\text{init } w) \frown (\neg g) \vee ((\text{halt } (\text{init } w) \wedge f) \frown (f1 \wedge g))$
using 23 **by** *fastforce*

have 4: $\vdash \text{halt } (\text{init } w) \frown g \wedge \text{finite} \longrightarrow \neg (\text{halt } (\text{init } w) \frown (\neg g))$
by (rule *HaltSCHopImpNotHaltSCHopNot*)

hence 41: $\vdash (\text{halt } (\text{init } w) \frown (\neg g)) \wedge \text{finite} \longrightarrow \neg (\text{halt } (\text{init } w) \frown g)$
by *auto*

have 42: $\vdash (\text{halt } (\text{init } w) \wedge f) \frown f1 \wedge \text{finite} \longrightarrow$
 $\neg (\text{halt } (\text{init } w) \frown g) \vee ((\text{halt } (\text{init } w) \wedge f) \frown (f1 \wedge g))$
using 31 41 **by** *fastforce*

from 42 **show** ?thesis **by** *auto*

qed

lemma *HaltImpBoxSYields*:

$\vdash (\text{halt } (\text{init } w)) \frown f \wedge \text{finite} \longrightarrow (\Box (\neg (\text{init } w))) \text{ syields } ((\text{halt } (\text{init } w)) \frown f)$

proof –

have 1: $\vdash (\Box (\neg (\text{init } w))) \frown (\neg (\text{halt } (\text{init } w) \frown f)) \longrightarrow \text{df } (\Box (\neg (\text{init } w)))$
by (rule *SCHopImpDf*)

have 2: $\vdash \Box (\neg (\text{init } w)) \longrightarrow \neg (\text{init } w)$
by (rule *BoxElim*)

hence 3: $\vdash \text{df } (\Box (\neg (\text{init } w))) \longrightarrow \text{df } (\neg (\text{init } w))$
by (rule *DfImpDf*)

have 4: $\vdash \text{df } (\text{init } (\neg w)) = (\text{init } (\neg w))$
by (rule *DfState*)

have 41: $\vdash (\text{init } (\neg w)) = (\neg (\text{init } w))$
using *Initprop*(2) **by** *fastforce*

have 42: $\vdash df (\neg (init\ w)) = (\neg (init\ w))$
using 4 41 **by** (*metis inteq-reflection*)
have 5: $\vdash ((\Box(\neg (init\ w))) \frown (\neg (halt\ (init\ w) \frown f))) \longrightarrow \neg (init\ w)$
using 1 2 42 **using** 3 **by** *fastforce*
hence 51: $\vdash (halt\ (init\ w) \frown f) \wedge ((\Box(\neg (init\ w))) \frown (\neg (halt\ (init\ w) \frown f))) \longrightarrow$
 $(halt\ (init\ w) \frown f) \wedge \neg (init\ w)$
by *fastforce*
have 6: $\vdash halt\ (init\ w) \frown f = if_i\ (init\ w)\ then\ f\ else\ (\Box(halt\ (init\ w) \frown f))$
by (*rule HaltSChopEqv*)
hence 61: $\vdash (halt\ (init\ w) \frown f \wedge \neg (init\ w)) =$
 $((if_i\ (init\ w)\ then\ f\ else\ (\Box(halt\ (init\ w) \frown f))) \wedge \neg (init\ w))$
using 6 **by** *auto*
have 62: $\vdash (if_i\ (init\ w)\ then\ f\ else\ (\Box(halt\ (init\ w) \frown f))) \wedge$
 $\neg (init\ w) \longrightarrow (\Box(halt\ (init\ w) \frown f))$
by (*auto simp: ifthenelse-d-def*)
have 63: $\vdash halt\ (init\ w) \frown f \wedge \neg (init\ w) \longrightarrow (\Box(halt\ (init\ w) \frown f))$
using 61 62 **by** *fastforce*
have 7: $\vdash (halt\ (init\ w) \frown f) \wedge (\Box(\neg (init\ w))) \frown (\neg (halt\ (init\ w) \frown f)) \longrightarrow$
 $\Box((halt\ (init\ w)) \frown f)$
using 51 63 **using** *lift-imp-trans* **by** *blast*
have 8: $\vdash \Box(\neg (init\ w)) \longrightarrow empty \vee \Box(\neg (init\ w))$
using *BoxBoxImpBox BoxEqvAndEmptyOrNextBox* **by** *fastforce*
hence 9: $\vdash ((\Box(\neg (init\ w))) \frown (\neg (halt\ (init\ w) \frown f))) \longrightarrow$
 $\neg (halt\ (init\ w) \frown f) \vee \Box((\Box(\neg (init\ w))) \frown (\neg (halt\ (init\ w) \frown f)))$
by (*rule EmptyOrNextSChopImpRule*)
hence 10: $\vdash ((halt\ (init\ w)) \frown f) \wedge (\Box(\neg (init\ w))) \frown (\neg (halt\ (init\ w) \frown f)) \longrightarrow$
 $\Box((\Box(\neg (init\ w))) \frown (\neg (halt\ (init\ w) \frown f)))$
by *fastforce*
have 11: $\vdash (halt\ (init\ w)) \frown f \wedge (\Box(\neg (init\ w))) \frown (\neg (halt\ (init\ w) \frown f)) \longrightarrow$
 $\Box((halt\ (init\ w)) \frown f) \wedge \Box((\Box(\neg (init\ w))) \frown (\neg (halt\ (init\ w) \frown f)))$
using 7 10 **by** *fastforce*
have 12: $\vdash \Box((halt\ (init\ w)) \frown f) \wedge \Box((\Box(\neg (init\ w))) \frown (\neg (halt\ (init\ w) \frown f)))$
 $\longrightarrow \Box(((halt\ (init\ w)) \frown f) \wedge ((\Box(\neg (init\ w))) \frown (\neg (halt\ (init\ w) \frown f))))$
using *NextAndEqvNextAndNext* **by** *fastforce*
have 13: $\vdash (halt\ (init\ w)) \frown f \wedge (\Box(\neg (init\ w))) \frown (\neg (halt\ (init\ w) \frown f)) \longrightarrow$
 $\Box(((halt\ (init\ w)) \frown f) \wedge ((\Box(\neg (init\ w))) \frown (\neg (halt\ (init\ w) \frown f))))$
using 11 12 **by** *fastforce*
hence 14: $\vdash finite \longrightarrow \neg ((halt\ (init\ w)) \frown f \wedge (\Box(\neg (init\ w))) \frown (\neg (halt\ (init\ w) \frown f)))$
using *NextLoop* **by** *blast*
hence 15: $\vdash (halt\ (init\ w)) \frown f \wedge finite \longrightarrow \neg ((\Box(\neg (init\ w))) \frown (\neg (halt\ (init\ w) \frown f)))$
by *auto*
from 15 **show** *?thesis* **by** (*simp add: syields-d-def*)
qed

10.11 Properties of Groups of strong chops

lemma *NestedSChopImpSChop*:

assumes $\vdash init\ w \wedge f \longrightarrow g \frown (init\ w1 \wedge f1)$
 $\vdash init\ w1 \wedge f1 \longrightarrow g1 \frown (init\ w2 \wedge f2)$
shows $\vdash init\ w \wedge f \longrightarrow g \frown (g1 \frown (init\ w2 \wedge f2))$

```

proof –
  have 1:  $\vdash \text{init } w \wedge f \longrightarrow g \frown (\text{init } w1 \wedge f1)$  using assms(1) by auto
  have 2:  $\vdash \text{init } w1 \wedge f1 \longrightarrow g1 \frown (\text{init } w2 \wedge f2)$  using assms(2) by auto
  hence 3:  $\vdash g \frown (\text{init } w1 \wedge f1) \longrightarrow g \frown (g1 \frown (\text{init } w2 \wedge f2))$  by (rule RightSChopImpSChop)
  from 1 3 show ?thesis by fastforce
qed

```

end

11 First Order Finite ITL theorems

theory *FOTheorems*

imports

Theorems

begin

We give the proofs of a list of first order Finite ITL theorems.

lemma *EExI-unI*:

$w \models f\ x \implies w \models (\exists\exists\ x. f\ x)$

using *EExVal* **by** *auto*

lemma *EExNoDep*:

$\vdash (\exists\exists\ x. g) = g$

proof –

have 1: $\vdash g \longrightarrow (\exists\exists\ x. g)$ **by** (*meson EExI*)

have 2: $\bigwedge x. \vdash g \longrightarrow g$ **by** *simp*

have 3: $\vdash (\exists\exists\ x. g) \longrightarrow g$ **using** 2 **by** (*meson EExE*)

from 1 3 **show** *?thesis* **using** *int-iffI* **by** *blast*

qed

lemma *AAxNoDep*:

$\vdash (\forall\forall\ x. g) = g$

using *EExNoDep AAxDef EExE EExI*

by (*smt Valid-def exist-state-d-def intensional-rews(2) intensional-rews(3)*)

lemma *EExEqvRule*:

assumes $\bigwedge x. \vdash f\ x = g\ x$

shows $\vdash (\exists\exists\ x. f\ x) = (\exists\exists\ x. g\ x)$

by (*metis EExE EExI assms int-iffD1 int-iffD2 int-iffI lift-imp-trans*)

lemma *AAxImpEEx*:

$\vdash (\forall\forall\ x. f\ x) \longrightarrow (\exists\exists\ x. f\ x)$

by (*simp add: exist-state-d-def forall-state-d-def intI*)

lemma *EExImpRule*:
assumes $\vdash f\ x \longrightarrow g\ x$
shows $\vdash (\exists\exists\ x. f\ x \longrightarrow g\ x)$
using *assms* **by** (*meson MP EExI*)

lemma *EExImpRuleDist*:
assumes $\vdash f\ x \longrightarrow g\ x$
shows $\vdash (\forall\forall\ x. f\ x) \longrightarrow (\exists\exists\ x. g\ x)$
proof —
have 1: $\vdash (f\ x) \longrightarrow (\exists\exists\ x. g\ x)$ **using** *EExI assms lift-imp-trans* **by** *blast*
have 2: $\vdash \neg(f\ x) \vee (\exists\exists\ x. g\ x)$ **using** 1 **by** *auto*
have 3: $\vdash \neg(f\ x) \longrightarrow (\exists\exists\ x. \neg(f\ x))$ **by** (*meson EExI*)
have 4: $\vdash (\exists\exists\ x. \neg(f\ x)) = (\neg(\forall\forall\ x. f\ x))$ **using** *AAxDef* **by** *fastforce*
from 2 3 4 **show** *?thesis* **by** *fastforce*
qed

lemma *EExImpNoDepDist*:
assumes $\vdash f \longrightarrow g\ x$
shows $\vdash f \longrightarrow (\exists\exists\ x. g\ x)$
using *assms* **by** (*metis EExI lift-imp-trans*)

lemma *EExOrDist-1*:
 $\vdash (\exists\exists\ x. h\ x) \longrightarrow (\exists\exists\ x. (f\ x) \vee (h\ x))$
proof —
have 1: $\bigwedge\ x. \vdash h\ x \longrightarrow f\ x \vee h\ x$ **by** (*simp add: Valid-def*)
have 2: $\bigwedge\ x. \vdash f\ x \vee h\ x \longrightarrow (\exists\exists\ x. (f\ x) \vee (h\ x))$ **by** (*meson EExI*)
have 3: $\bigwedge\ x. \vdash h\ x \longrightarrow (\exists\exists\ x. (f\ x) \vee (h\ x))$ **using** 1 2 **by** (*meson lift-imp-trans*)
from 3 **show** *?thesis* **using** *EExE* **by** *blast*
qed

lemma *EExOrDist-2*:
 $\vdash (\exists\exists\ x. f\ x) \longrightarrow (\exists\exists\ x. (f\ x) \vee (h\ x))$
proof —
have 1: $\bigwedge\ x. \vdash f\ x \longrightarrow f\ x \vee h\ x$ **by** (*simp add: Valid-def*)
have 2: $\bigwedge\ x. \vdash f\ x \vee h\ x \longrightarrow (\exists\exists\ x. (f\ x) \vee (h\ x))$ **by** (*meson EExI*)
have 3: $\bigwedge\ x. \vdash f\ x \longrightarrow (\exists\exists\ x. (f\ x) \vee (h\ x))$ **using** 1 2 **by** (*meson lift-imp-trans*)
from 3 **show** *?thesis* **using** *EExE* **by** *blast*
qed

lemma *EExOrDist-3*:
 $\vdash (\exists\exists\ x. f\ x) \vee (\exists\exists\ x. h\ x) \longrightarrow (\exists\exists\ x. (f\ x) \vee (h\ x))$
using *EExOrDist-2 EExOrDist-1* **by** *fastforce*

lemma *EExOrDist-4*:
 $\vdash (\exists\exists\ x. (f\ x) \vee (h\ x)) \longrightarrow (\exists\exists\ x. f\ x) \vee (\exists\exists\ x. h\ x)$
proof —
have 1: $\bigwedge\ x. \vdash (f\ x) \vee (h\ x) \longrightarrow (\exists\exists\ x. f\ x) \vee (\exists\exists\ x. h\ x)$
by (*simp add: EExI-unl intl*)
from 1 **show** *?thesis* **by** (*simp add: EExE*)
qed

lemma EExOrDist:

$\vdash ((\exists \exists x. f x) \vee (\exists \exists x. h x)) = (\exists \exists x. (f x) \vee (h x))$

using EExOrDist-3 EExOrDist-4 **by** fastforce

lemma EExOrImport-1:

$\vdash g \longrightarrow (\exists \exists x. g \vee (f x))$

by (simp add: EExI-unl Valid-def)

lemma EExOrImport-2:

$\vdash (\exists \exists x. f x) \longrightarrow (\exists \exists x. g \vee (f x))$

by (simp add: EExOrDist-1)

lemma EExOrImport-3:

$\vdash (g \vee (\exists \exists x. f x)) \longrightarrow (\exists \exists x. g \vee (f x))$

using EExOrImport-1 EExOrImport-2 **by** fastforce

lemma EExOrImport-4:

$\vdash (\exists \exists x. g \vee f x) \longrightarrow (g \vee (\exists \exists x. f x))$

proof —

have 1: $\bigwedge x. \vdash g \vee f x \longrightarrow g \vee (\exists \exists x. f x)$ **by** (meson EExI int-iffD2 int-simps(27) Prop04 Prop08)

from 1 **show** ?thesis **by** (simp add: EExE)

qed

lemma EExOrImport:

$\vdash (g \vee (\exists \exists x. f x)) = (\exists \exists x. g \vee f x)$

by (metis EExOrImport-3 EExOrImport-4 int-iffI)

lemma EExAndImport-1:

$\vdash g \wedge (\exists \exists x. f x) \longrightarrow (\exists \exists x. g \wedge f x)$

proof —

have 1: $\vdash (g \wedge (\exists \exists x. f x) \longrightarrow (\exists \exists x. g \wedge f x)) =$
 $((\exists \exists x. f x) \longrightarrow (g \longrightarrow (\exists \exists x. g \wedge f x)))$ **by** fastforce

have 2: $\bigwedge x. \vdash f x \longrightarrow (g \longrightarrow (\exists \exists x. g \wedge f x))$ **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 $\vdash (\forall\forall\ x. f\ x)$
unfolding *AAxDef*
using *AAxDef EExE assms*
by (*smt Valid-def int-simps(15) unl-lift unl-lift2*)

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 $\vdash (\exists\exists x. h_i(f x) \longrightarrow h_i(g x))$

using *RightChopImpChop EExImpRule assms* **by** (*smt MP EExI*)

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 (*simp add: exist-state-d-def Valid-def chop-defs, auto*)

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 (*simp add: exist-state-d-def Valid-def chop-defs, auto*)

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 chop-defs, blast*)

lemma *EExEExChopEqvEExChopEExA*:

$\vdash (\exists \exists v. (\exists \exists y. (f\ v);(g\ y))) = (\exists \exists v. (f\ v);(\exists \exists y. (g\ y)))$

by (*simp add: exist-state-d-def Valid-def chop-defs, blast*)

lemma *EExEExChopEqvEExChopEExB*:

$\vdash (\exists \exists y. (\exists \exists v. (f\ v);(g\ y))) = (\exists \exists y. (\exists \exists v. (f\ v));(g\ y))$

by (*simp add: exist-state-d-def Valid-def chop-defs, blast*)

lemma *EExEExChopEqvEExChopEExC*:

$\vdash (\exists \exists v. (\exists \exists y. (f\ v);(g\ y))) = (\exists \exists v. (f\ v));(\exists \exists y. (g\ y))$

by (*metis EExChopRightNoDep EExEExChopEqvEExChopEExA EExNoDep Prop04*)

lemma *ExLen*:

$\vdash \exists n. \text{len}(n)$

by (*simp add: Valid-def len-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 (*simp add: Valid-def chop-defs, auto*)

```

lemma ExChopLeftNoDep:
   $\vdash (\exists x. g.(f\ x)) = g.(\exists x. f\ x)$ 
by (simp add: Valid-def chop-defs, auto)

lemma ExExEqvExEx:
   $\vdash (\exists x. (\exists y. (f\ x).(g\ y))) = (\exists y. (\exists x. (f\ x).(g\ y)))$ 
by (simp add: Valid-def chop-defs, auto)

```

end

12 Time Reversal

```

theory TimeReversal
imports
  Theorems FOTheorems
begin

```

Time reversal operator is defined in [5].

12.1 Definition

```

definition reverse-d :: ('a::world, 'b) formfun  $\Rightarrow$  ('a, 'b) formfun
where reverse-d F  $\equiv \lambda s. \text{intrev } s \models F$ 

```

```

syntax
  -reverse-d      :: lift  $\Rightarrow$  lift      ((-r) [85] 85)

```

```

syntax (ASCII)
  -reverse-d      :: lift  $\Rightarrow$  lift      ((reverse -) [85] 85)

```

```

translations
  -reverse-d       $\Rightarrow$  CONST reverse-d

```

12.2 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 :
   $\vdash (f<x>)^r = f<x^r>$ 
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: interval-intrev-nth current-val-d-def fin-val-d-def reverse-d-def*)

lemma *rev-next*:
 $\vdash (v\$)^r = (v!)$
by (*auto simp: interval-intrev-nth next-val-d-def penult-val-d-def reverse-d-def*)

lemma *rev-penult*:
 $\vdash (v!)^r = (v\$)$
by (*auto simp: interval-intrev-nth next-val-d-def penult-val-d-def reverse-d-def*)

lemma *rev-fin*:
 $\vdash (!v)^r = (\$v)$
by (*auto simp: interval-intrev-nth fin-val-d-def current-val-d-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 interval-rev-swap reverse-d-def*)

lemma *RevSkip*:
 $\vdash skip^r = skip$
by (*simp add: Valid-def reverse-d-def skip-defs*)

lemma *RevChop*:
 $\vdash (f;g)^r = (g^r;f^r)$
using *interval-intrev-prefix interval-intrev-suffix*

by (*smt chop-d-def diff-le-self intl intensional-rews(3) interval-intrev-intlen interval-rev-swap reverse-d-def zero-order(1)*)

lemma *RMoreEqvMore*:

$\vdash \text{more}^r = \text{more}$

by (*simp add: Valid-def more-d-def next-d-def chop-d-def skip-d-def reverse-d-def interval-prefix-length*)

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*:

$\vdash (f^*)^r = (f^r)^*$

proof —

have 1: $\vdash (f^*) = (\exists n. \text{power } (f \wedge \text{more}) \ n)$

by (*simp add: chopstar-d-def powerstar-d-def Valid-def*)

have 2: $\vdash (f^*)^r = (\exists n. \text{power } (f \wedge \text{more}) \ n)^r$

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]*

12.3 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 EqvReverseReverse 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** *EqvReverseReverse* **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** *EqvReverseReverse* **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 empty-defs reverse-d-def, metis interval-st-intlen intrev.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 (*smt RNextEqvPrev REmptyEqvEmpty WnextEqvEmptyOrNext WprevEqvEmptyOrPrev int-eq rev-fun2*)

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 RWPprevEqvWNNext:
 $\vdash (wprev\ f)^r = wnext(f^r)$
by (metis EqvReverseReverse RRWNNextEqvWPrev int-eq)

lemma RRWPprevEqvWNNext:
 $\vdash (wprev\ (f^r))^r = wnext(f)$
proof –
have 1: $\vdash (wprev\ (f^r))^r = wnext\ ((f^r)^r)$ **using** RWPprevEqvWNNext **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 RRBBoxEqvBi:
 $\vdash (\Box\ (f^r))^r = bi\ (f)$
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*:

$\vdash (bi\ (f^r))^r = \Box (f)$

proof —

have 1: $\vdash (bi\ (f^r))^r = \Box ((f^r)^r)$ **using** *RBiEqvBox* **by** *blast*

have 2: $\vdash \Box ((f^r)^r) = \Box f$ **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 *ChopCslmpCSChop*:

$\vdash f;f^* \longrightarrow f^*;f$

by (*meson CSChopEqvChopOrRule CSChopEqvOrChopPlusChop ChopAssocB*)

lemma *CSChopImpChopCS*:

$\vdash f^*;f \longrightarrow f;f^*$

proof —

have 1: $\vdash (f^r);(f^r)^* \longrightarrow (f^r)^*;(f^r)$

using *ChopCslmpCSChop* **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 (*smt 1 2 RImpRule Valid-def unl-lift2*)

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 ChopCslmpCSChop RImpRule int-eq int-iff1*)

from 2 3 8 12 **show** *?thesis* **by** *fastforce*

qed

lemma *CSChopEqvChopCS*:

$\vdash f;f^* = f^*;f$

using *ChopCslmpCSChop 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:  $\vdash ((f \wedge \text{empty}); \# \text{True})^r = (\# \text{True}; (f \wedge \text{empty}))^r$ 
  using RTrue by (metis RevChop int-eq)
have 3:  $\vdash \# \text{True}; (f \wedge \text{empty})^r = \# \text{True}; (f^r \wedge \text{empty})$ 
  by (metis RAnd REmptyEqvEmpty RightChopEqvChop int-eq)
have 4:  $\vdash \# \text{True}; (f^r \wedge \text{empty}) = \# \text{True}; (f \wedge \text{empty})$ 
  using RAndEmptyEqvAndEmpty
  by (metis REmptyEqvEmpty RightChopEqvChop all-rev-eq(3) int-eq)
have 5:  $\vdash \# \text{True}; (f \wedge \text{empty}) = \text{fin}(f)$ 
  using FinEqvTrueChopAndEmpty by fastforce
from 1 2 3 4 5 show ?thesis by fastforce
qed

```

```

lemma RFinEqvInit:
 $\vdash (\text{fin } f)^r = \text{init } (f)$ 
proof -
  have 1:  $\vdash \text{fin } f = \# \text{True}; (f \wedge \text{empty})$ 
    using FinEqvTrueChopAndEmpty by auto
  have 2:  $\vdash (\text{fin } f)^r = (\# \text{True}; (f \wedge \text{empty}))^r$ 
    using 1 REqvRule by blast
  have 3:  $\vdash (\# \text{True}; (f \wedge \text{empty}))^r = (f \wedge \text{empty})^r; \# \text{True}$ 
    using RTrue by (metis RevChop int-eq)
  have 4:  $\vdash (f \wedge \text{empty})^r; \# \text{True} = (f^r \wedge \text{empty}); \# \text{True}$ 
    using LeftChopEqvChop RAnd REmptyEqvEmpty by (metis int-eq)
  have 5:  $\vdash (f \wedge \text{empty})^r; \# \text{True} = (f \wedge \text{empty}); \# \text{True}$ 
    by (simp add: RAndEmptyEqvAndEmpty LeftChopEqvChop)
  have 6:  $\vdash (f \wedge \text{empty}); \# \text{True} = \text{init}(f)$ 
    by (simp add: 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:  $\vdash (\text{halt } f)^r = (\square ( \text{empty} = f ))^r$  by (simp add: halt-d-def)
  have 2:  $\vdash (\square ( \text{empty} = f ))^r = \text{bi } ( \text{empty} = f )^r$  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: $\vdash (bi\ (empty = f))^r = \square((empty = f)^r)$ **by** (simp add: RBEqvBox)
have 3: $\vdash (empty = f)^r = (empty = f^r)$ **by** (metis REmptyEqvEmpty inteq-reflection rev-fun2)
hence 4: $\vdash \square((empty = f)^r) = \square(empty = f^r)$ **by** (simp add: BEqvBox)
have 5: $\vdash \square(empty = f^r) = halt(f^r)$ **by** (simp add: halt-d-def)
from 1 2 4 5 **show** ?thesis **by** fastforce
qed

lemma RRHaltEqvInitonly:

$\vdash (halt\ (f^r))^r = initonly\ (f)$
proof –
have 1: $\vdash (halt\ (f^r))^r = initonly\ ((f^r)^r)$ **using** RHaltEqvInitonly **by** blast
have 2: $\vdash initonly\ ((f^r)^r) = initonly(f)$ **using** EqvReverseReverse **by** (metis inteq-reflection)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma RRInitonlyEqvHalt :

$\vdash (intonly\ (f^r))^r = halt(f)$
proof –
have 1: $\vdash (intonly\ (f^r))^r = halt((f^r)^r)$ **using** RInitonlyEqvHalt **by** blast
have 2: $\vdash halt((f^r)^r) = halt(f)$ **using** EqvReverseReverse **by** (metis inteq-reflection)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma RKeepEqvKeep :

$\vdash (keep\ f)^r = keep(f^r)$
proof –
have 1: $\vdash (keep\ f)^r = (ba(skip \longrightarrow f))^r$ **by** (simp add: keep-d-def)
have 2: $\vdash (ba(skip \longrightarrow f))^r = ba((skip \longrightarrow f)^r)$ **by** (simp add: RBaEqvBa)
have 3: $\vdash (skip \longrightarrow f)^r = (skip \longrightarrow f^r)$ **by** (metis all-rev-eq(12) rev-fun2)
hence 4: $\vdash ba((skip \longrightarrow f)^r) = ba(skip \longrightarrow f^r)$ **by** (simp add: BaEqvBa)
have 5: $\vdash ba(skip \longrightarrow f^r) = keep(f^r)$ **by** (simp add: keep-d-def)
from 1 2 4 5 **show** ?thesis **by** fastforce
qed

lemma RRKeepEqvKeep :

$\vdash (keep\ (f^r))^r = keep(f)$
proof –
have 1: $\vdash (keep\ (f^r))^r = keep((f^r)^r)$ **using** RKeepEqvKeep **by** blast
have 2: $\vdash keep((f^r)^r) = keep(f)$ **using** EqvReverseReverse **by** (metis inteq-reflection)
from 1 2 **show** ?thesis **by** fastforce
qed

lemma NextDiamondEqvDiamondNext:

$\vdash \circ(\diamond f) = \diamond(\circ f)$
proof –
have 1: $\vdash \#True;skip = skip;\#True$ **by** (rule TrueChopSkipEqvSkipChopTrue)
hence 2: $\vdash (\#True;skip);f = (skip;\#True);f$ **using** LeftChopEqvChop **by** blast
have 3: $\vdash (\#True;skip);f = \#True;(skip;f)$ **by** (simp add: ChopAssocB)
have 4: $\vdash (skip;\#True);f = skip;(\#True;f)$ **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: intl 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 intl 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 intl 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 intl 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 *RGetsEqvBaSkiplmp*:

$\vdash (v \text{ gets } e)^r = \text{ba}(\text{skip} \longrightarrow (\$v = e^r))$

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*:

$\vdash (\text{if}_i f_0 \text{ then } f_1 \text{ else } f_2)^r = \text{if}_i (f_0^r) \text{ then } (f_1^r) \text{ else } (f_2^r)$

by (*simp add: all-rev-eq(2) all-rev-eq(3) ifthenelse-d-def*)

lemma *RWhile*:

$\vdash (\text{init } f \wedge \text{while } f_0 \text{ do } f_1)^r = (\text{fin}(f) \wedge ((f_0^r) \wedge (f_1^r))^* \wedge \text{init} (\neg(f_0)))$

proof —

have 1: $\vdash (\text{init } f \wedge \text{while } f_0 \text{ do } f_1)^r = (\text{init } f \wedge (f_0 \wedge f_1)^* \wedge \text{fin} (\neg f_0))^r$

by (*simp add: while-d-def*)

have 2: $\vdash (\text{init } f \wedge (f_0 \wedge f_1)^* \wedge \text{fin} (\neg f_0))^r = ((\text{init } f)^r \wedge ((f_0 \wedge f_1)^*)^r \wedge (\text{fin} (\neg f_0))^r)$

by (*simp add: all-rev-eq(3)*)

have 3: $\vdash (\text{init } f)^r = \text{fin}(f)$

by (*simp add: RInitEqvFin*)

have 4: $\vdash ((f_0 \wedge f_1)^*)^r = ((f_0^r) \wedge (f_1^r))^*$

by (*metis RevChopstar all-rev-eq(3)*)

have 5: $\vdash (\text{fin} (\neg f_0))^r = \text{init} (\neg(f_0))$

by (*metis RFinEqvInit*)

have 6: $\vdash ((\text{init } f)^r \wedge ((f_0 \wedge f_1)^*)^r \wedge (\text{fin} (\neg f_0))^r) = (\text{fin}(f) \wedge ((f_0^r) \wedge (f_1^r))^* \wedge \text{init} (\neg(f_0)))$ **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 EExEqvRule 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

13 First Order Infinite ITL theorems

theory InfiniteFOTheorems

imports

InfiniteSChopTheorems

begin

We give the proofs of a list of first order infinite ITL theorems.

lemma EExI-unl:

$w \models f x \implies w \models (\exists \exists x. f x)$

using EExValInfinite EExValFinite

by (cases w, blast, meson exist-state-d-def)

lemma EExNoDep:

$\vdash (\exists \exists x. g) = g$

proof –

have 1: $\vdash g \longrightarrow (\exists \exists x. g)$ **by** (meson EExI)

have 2: $\bigwedge x. \vdash g \longrightarrow g$ **by** simp

have 3: $\vdash (\exists \exists x. g) \longrightarrow g$ **using** 2 **by** (meson EExE)

from 1 3 **show** ?thesis **using** int-iff1 **by** blast

qed

lemma AAXNoDep:

$\vdash (\forall \forall x. g) = g$

using EExNoDep AAXDef EExE EExI

by (smt Valid-def exist-state-d-def intensional-rews(2) intensional-rews(3))

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-iff1 lift-imp-trans)

lemma AAXImpEEx:

$\vdash (\forall \forall x. f x) \longrightarrow (\exists \exists x. f x)$

by (simp add: exist-state-d-def forall-state-d-def intI)

lemma *EExImpRule*:

assumes $\vdash f\ x \longrightarrow g\ x$

shows $\vdash (\exists\exists\ x. f\ x \longrightarrow g\ x)$

using *assms* **by** (*meson MP EExI*)

lemma *EExImpRuleDist*:

assumes $\vdash f\ x \longrightarrow g\ x$

shows $\vdash (\forall\forall\ x. f\ x) \longrightarrow (\exists\exists\ x. g\ x)$

proof —

have 1: $\vdash (f\ x) \longrightarrow (\exists\exists\ x. g\ x)$ **using** *EExI assms lift-imp-trans* **by** *blast*

have 2: $\vdash \neg(f\ x) \vee (\exists\exists\ x. g\ x)$ **using** 1 **by** *auto*

have 3: $\vdash \neg(f\ x) \longrightarrow (\exists\exists\ x. \neg(f\ x))$ **by** (*meson EExI*)

have 4: $\vdash (\exists\exists\ x. \neg(f\ x)) = (\neg(\forall\forall\ x. f\ x))$ **using** *AAxDef* **by** *fastforce*

from 2 3 4 **show** *?thesis* **by** *fastforce*

qed

lemma *EExImpNoDepDist*:

assumes $\vdash f \longrightarrow g\ x$

shows $\vdash f \longrightarrow (\exists\exists\ x. g\ x)$

using *assms* **by** (*metis EExI lift-imp-trans*)

lemma *EExOrDist-1*:

$\vdash (\exists\exists\ x. h\ x) \longrightarrow (\exists\exists\ x. (f\ x) \vee (h\ x))$

proof —

have 1: $\bigwedge x. \vdash h\ x \longrightarrow f\ x \vee h\ x$ **by** (*simp add: Valid-def*)

have 2: $\bigwedge x. \vdash f\ x \vee h\ x \longrightarrow (\exists\exists\ x. (f\ x) \vee (h\ x))$ **by** (*meson EExI*)

have 3: $\bigwedge x. \vdash h\ x \longrightarrow (\exists\exists\ x. (f\ x) \vee (h\ x))$ **using** 1 2 **by** (*meson lift-imp-trans*)

from 3 **show** *?thesis* **using** *EExE* **by** *blast*

qed

lemma *EExOrDist-2*:

$\vdash (\exists\exists\ x. f\ x) \longrightarrow (\exists\exists\ x. (f\ x) \vee (h\ x))$

proof —

have 1: $\bigwedge x. \vdash f\ x \longrightarrow f\ x \vee h\ x$ **by** (*simp add: Valid-def*)

have 2: $\bigwedge x. \vdash f\ x \vee h\ x \longrightarrow (\exists\exists\ x. (f\ x) \vee (h\ x))$ **by** (*meson EExI*)

have 3: $\bigwedge x. \vdash f\ x \longrightarrow (\exists\exists\ x. (f\ x) \vee (h\ x))$ **using** 1 2 **by** (*meson lift-imp-trans*)

from 3 **show** *?thesis* **using** *EExE* **by** *blast*

qed

lemma *EExOrDist-3*:

$\vdash (\exists\exists\ x. f\ x) \vee (\exists\exists\ x. h\ x) \longrightarrow (\exists\exists\ x. (f\ x) \vee (h\ x))$

using *EExOrDist-2 EExOrDist-1* **by** *fastforce*

lemma *EExOrDist-4*:

$\vdash (\exists\exists\ x. (f\ x) \vee (h\ x)) \longrightarrow (\exists\exists\ x. f\ x) \vee (\exists\exists\ x. h\ x)$

proof —

have 1: $\bigwedge x. \vdash (f\ x) \vee (h\ x) \longrightarrow (\exists\exists\ x. f\ x) \vee (\exists\exists\ x. h\ x)$

by (*simp add: EExI-unl intI*)

from 1 **show** *?thesis* **by** (*simp add: EExE*)

qed

lemma *EExOrDist*:

$\vdash ((\exists \exists x. f x) \vee (\exists \exists x. h x)) = (\exists \exists x. (f x) \vee (h x))$

using *EExOrDist-3 EExOrDist-4* **by** *fastforce*

lemma *EExOrImport-1*:

$\vdash g \longrightarrow (\exists \exists x. g \vee (f x))$

by (*simp add: EExl-unl Valid-def*)

lemma *EExOrImport-2*:

$\vdash (\exists \exists x. f x) \longrightarrow (\exists \exists x. g \vee (f x))$

by (*simp add: EExOrDist-1*)

lemma *EExOrImport-3*:

$\vdash (g \vee (\exists \exists x. f x)) \longrightarrow (\exists \exists x. g \vee (f x))$

using *EExOrImport-1 EExOrImport-2* **by** *fastforce*

lemma *EExOrImport-4*:

$\vdash (\exists \exists x. g \vee f x) \longrightarrow (g \vee (\exists \exists x. f x))$

proof —

have 1: $\bigwedge x. \vdash g \vee f x \longrightarrow g \vee (\exists \exists x. f x)$ **by** (*meson EExl int-iffD2 int-simps(27) Prop04 Prop08*)

from 1 **show** *?thesis* **by** (*simp add: EExE*)

qed

lemma *EExOrImport*:

$\vdash (g \vee (\exists \exists x. f x)) = (\exists \exists x. g \vee f x)$

by (*metis EExOrImport-3 EExOrImport-4 int-iffI*)

lemma *EExAndImport-1*:

$\vdash g \wedge (\exists \exists x. f x) \longrightarrow (\exists \exists x. g \wedge f x)$

proof —

have 1: $\vdash (g \wedge (\exists \exists x. f x) \longrightarrow (\exists \exists x. g \wedge f x)) =$
 $((\exists \exists x. f x) \longrightarrow (g \longrightarrow (\exists \exists x. g \wedge f x)))$ **by** *fastforce*

have 2: $\bigwedge x. \vdash f x \longrightarrow (g \longrightarrow (\exists \exists x. g \wedge f x))$ **by** (*metis EExl 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 EExl 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 $\vdash (\forall\forall\ x. f\ x)$
unfolding *AAxDef*
using *AAxDef* *EExE* *assms*
by (*smt* *Valid-def* *int-simps*(15) *unl-lift* *unl-lift2*)

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: $\vdash (\neg(\exists\exists x. \neg(f x))) = (\neg(\neg(\forall\forall x. f x)))$ **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: $\vdash (\exists\exists x. \neg(f x)) = (\neg(\neg(\forall\forall x. f x)))$ **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 $\vdash (\exists\exists x. h_i(f x) \longrightarrow h_i(g x))$

using *RightChopImpChop EExImpRule assms* **by** (*smt MP EExI*)

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 (simp add: exist-state-d-def Valid-def chop-defs sum.case-eq-if, auto)

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 (simp add: exist-state-d-def Valid-def chop-defs sum.case-eq-if, auto)

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 chop-defs, blast)

lemma EExEExChopEqvEExChopEExA:

$\vdash (\exists \exists v. (\exists \exists y. (f v); (g y))) = (\exists \exists v. (f v); (\exists \exists y. (g y)))$

by (simp add: exist-state-d-def Valid-def chop-defs sum.case-eq-if, blast)

lemma EExEExChopEqvEExChopEExB:

$\vdash (\exists \exists y. (\exists \exists v. (f v); (g y))) = (\exists \exists y. (\exists \exists v. (f v)); (g y))$

by (simp add: exist-state-d-def Valid-def chop-defs sum.case-eq-if, blast)

lemma EExEExChopEqvEExChopEExC:

$\vdash (\exists \exists v. (\exists \exists y. (f v); (g y))) = (\exists \exists v. (f v); (\exists \exists y. (g y)))$

by (simp add: exist-state-d-def Valid-def chop-defs sum.case-eq-if, blast)

lemma ExLenOrInf:

$\vdash (\exists n. \text{len}(n)) \vee \text{inf}$

by (simp add: Valid-def len-defs sum.case-eq-if infinite-defs)

lemma CSPowerChop:

$\vdash (f^*) = (\exists n. \text{power } (f \wedge \text{more}) n); (\text{empty} \vee (f \wedge \text{more}) \wedge \text{inf})$

by (simp add: chopstar-d-def powerstar-d-def Valid-def sum.case-eq-if)

lemma ExChopRightNoDep:

$\vdash (\exists x. (f x); g) = (\exists x. (f x)); g$

by (simp add: Valid-def chop-defs sum.case-eq-if, auto)

lemma ExChopLeftNoDep:

$\vdash (\exists x. g;(f x)) = g;(\exists x. f x)$
by (*simp add: Valid-def chop-defs sum.case-eq-if, auto*)

lemma *ExExEqvExEx*:

$\vdash (\exists x. (\exists y. (f x);(g y))) = (\exists y. (\exists x. (f x);(g y)))$
by (*simp add: Valid-def chop-defs, auto*)

end

14 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 construct 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.

14.1 Definitions

14.1.1 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*;(□ *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

$-bs-d \Rightarrow CONST\ bs-d$

$-bt-d \Rightarrow CONST\ bt-d$

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

$-ds-d \Rightarrow CONST\ ds-d$

$-dt-d \Rightarrow CONST\ dt-d$

14.1.2 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

$-first-d \Rightarrow CONST\ first-d$

$\neg last-d \Rightarrow CONST last-d$

14.2 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: $\vdash skip;\Box(f) = skip;\Box(g)$ **using** *RightChopEqvChop* **by** *blast*

hence 4: $\vdash (empty \vee skip;\Box(f)) = (empty \vee skip;\Box(g))$ **by** *auto*

hence 5: $\vdash bt(f) = bt(g)$ **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: $\vdash bs(\neg f) = bs(\neg g)$ **by** (*simp add: BsEqvRule*)

hence 4: $\vdash (f \wedge bs(\neg f)) = (g \wedge bs(\neg g))$ **using** 1 **by** *fastforce*

from 4 **show** *?thesis* **by** (*simp add: first-d-def*)

qed

lemma *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: $\vdash bt(\neg f) = bt(\neg g)$ **by** (*simp add: BtEqvRule*)

hence 4: $\vdash (f \wedge bt(\neg f)) = (g \wedge bt(\neg g))$ **using** 1 **by** *fastforce*

from 4 **show** *?thesis* **by** (*simp add: last-d-def*)

qed

lemma *RBsEqvBt*:

$\vdash (bs\ f)^r = (bt\ (f^r))$

proof –

have 1: $\vdash (bs\ f)^r = (empty \vee ((bi\ f) ; skip))^r$

by (*simp add: bs-d-def*)

have 2: $\vdash (empty \vee ((bi\ f) ; skip))^r = (empty^r \vee ((bi\ f) ; skip)^r)$

using *ROr* **by** *blast*

have 3: $\vdash (empty^r \vee ((bi\ f) ; skip)^r) = (empty \vee (skip^r ; (bi\ f)^r))$

using *REmptyEqvEmpty RevChop* **by** *fastforce*

have 4: $\vdash (empty \vee (skip^r ; (bi\ f)^r)) = (empty \vee (skip ; \Box (f^r)))$

by (*metis 3 RBoxEqvBi RevSkip int-eq*)

have 5: $\vdash (empty \vee (skip ; \Box (f^r))) = (bt\ (f^r))$

by (*simp add: bt-d-def*)

from 1 2 3 4 5 **show** *?thesis* **by** *fastforce*

qed

lemma *RRBsEqvBt*:

$\vdash (bs\ (f^r))^r = (bt\ (f))$

proof –

have 1: $\vdash (bs\ (f^r))^r = bt\ ((f^r)^r)$ **using** *RBsEqvBt* **by** *blast*

have 2: $\vdash bt\ ((f^r)^r) = bt\ f$ **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: $\vdash (bt\ f)^r = (empty \vee (skip ; \Box f))^r$

by (*simp add: bt-d-def*)

have 2: $\vdash (empty \vee (skip ; \Box f))^r = (empty^r \vee (skip ; \Box f)^r)$

using *ROr* **by** *blast*

have 3: $\vdash (empty^r \vee (skip ; \Box f)^r) = (empty \vee (\Box f)^r ; skip^r)$

using *REmptyEqvEmpty RevChop* **by** *fastforce*

have 4: $\vdash (empty \vee (\Box f)^r ; skip^r) = (empty \vee (bi\ (f^r)) ; skip)$

by (*metis 3 RBoxEqvBi RevSkip int-eq*)

have 5: $\vdash (empty \vee (bi\ (f^r)) ; skip) = (bs\ (f^r))$

by (*simp add: bs-d-def*)

from 1 2 3 4 5 **show** *?thesis* **by** *fastforce*

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 RRFistEqvLast:

$\vdash (\triangleright (f^r))^r = (\triangleleft f)$

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

14.3 Semantic Theorems

14.3.1 Semantics First and Last Operators

lemma FstAndBisem:

$(\text{intlen } \sigma > 0 \wedge (\sigma \models f) \wedge (\sigma \models \text{bi } (\neg f); \text{skip})) =$
 $(\text{intlen } \sigma > 0 \wedge (\sigma \models f) \wedge (\forall ia < \text{intlen } (\sigma). (\text{prefix } ia \ \sigma \models \neg f)))$

proof –

have $(\text{intlen } \sigma > 0 \wedge (\sigma \models f) \wedge (\sigma \models \text{bi } (\neg f); \text{skip})) =$
 $(0 < \text{intlen } \sigma \wedge (\sigma \models f) \wedge$
 $(\exists i. (i \leq \text{intlen } \sigma \longrightarrow (\forall ia \leq i. \neg (\text{prefix } ia (\text{prefix } i \ \sigma) \models f))) \wedge$
 $\text{intlen } \sigma - i = \text{Suc } 0) \wedge i \leq \text{intlen } \sigma)$
 $)$
by (simp add: chop-defs bi-defs skip-defs interval-prefix-length interval-suffix-length)
also have ... =

```

    (0 < intlen σ ∧ (σ ⊨ f) ∧
    (∃ i. (i ≤ intlen σ → (∀ ia ≤ i. ¬ (prefix ia (prefix i σ) ⊨ f)) ∧
    i = intlen σ - Suc 0) ∧ i ≤ intlen σ)
    )
  by auto
also have ... =
  (0 < intlen σ ∧ (σ ⊨ f) ∧
  (∀ ia ≤ (intlen σ - Suc 0). ¬ (prefix ia (prefix (intlen σ - Suc 0) σ) ⊨ f))
  )
  using diff-le-self by blast
also have ... =
  (intlen σ > 0 ∧ (σ ⊨ f) ∧
  (∀ ia < intlen (σ). ¬ (prefix ia (prefix (intlen σ - Suc 0) σ) ⊨ f))
  ) by (metis Suc-pred less-Suc-eq-le)
also have ... =
  (intlen σ > 0 ∧ (σ ⊨ f) ∧
  (∀ ia < intlen (σ). (prefix ia (prefix (intlen σ - Suc 0) σ) ⊨ ¬f))
  )
  by auto
also have ... =
  (intlen σ > 0 ∧ (σ ⊨ f) ∧ (∀ ia < intlen (σ). (prefix ia σ ⊨ ¬f)))
  by (simp add: interval-pref-pref-help)
finally show (intlen σ > 0 ∧ (σ ⊨ f) ∧ (σ ⊨ bi (¬f); skip)) =
  (intlen σ > 0 ∧ (σ ⊨ f) ∧ (∀ ia < intlen (σ). (prefix ia σ ⊨ ¬f))) .
qed

```

lemma Fstsem-0:

```

(σ ⊨ ▷f) =
(
  (σ ⊨ f ∧ empty) ∨ (intlen σ > 0 ∧ (σ ⊨ f) ∧ (σ ⊨ bi (¬f); skip))
)
using empty-defs by (simp add: first-d-def bs-d-def, auto)

```

lemma Emptysem:

```

(σ ⊨ f ∧ empty) = ((σ ⊨ f) ∧ intlen σ = 0)
using empty-defs by auto

```

lemma Fstsem:

```

(σ ⊨ ▷f) =
(
  ((σ ⊨ f) ∧ intlen σ = 0) ∨
  (intlen σ > 0 ∧ (σ ⊨ f) ∧ (∀ ia < intlen (σ). (prefix ia σ ⊨ ¬f)))
)
using Fstsem-0 Emptysem FstAndBisem by metis

```

lemma Lstsem:

```

(σ ⊨ ◁f) =
(
  ((σ ⊨ f) ∧ intlen σ = 0) ∨
  (intlen σ > 0 ∧ (σ ⊨ f) ∧ (∀ ia < intlen σ. (suffix ((intlen σ) - ia) σ ⊨ ¬f)))
)

```

```

proof –
  have  $(\sigma \models \triangleleft f) = (\sigma \models (\triangleright (f^r))^r)$ 
    using RRFirstEqvLast by fastforce
  also have  $\dots = (\text{intrev } \sigma \models \triangleright (f^r))$ 
    by (metis reverse-d-def)
  also have  $\dots =$ 
    (
      (  $(\text{intrev } \sigma \models f^r) \wedge \text{intlen } (\text{intrev } \sigma) = 0$ )  $\vee$ 
      (  $\text{intlen } (\text{intrev } \sigma) > 0 \wedge (\text{intrev } \sigma \models f^r) \wedge$ 
         $(\forall ia < \text{intlen } (\text{intrev } \sigma). (\text{prefix } ia \ (\text{intrev } \sigma) \models \neg(f^r)))$ 
      )
    )
    using Fstsem by blast
  also have  $\dots =$ 
    (
      (  $(\sigma \models f) \wedge \text{intlen } (\sigma) = 0$ )  $\vee$ 
      (  $\text{intlen } (\sigma) > 0 \wedge (\sigma \models f) \wedge$ 
         $(\forall ia < \text{intlen } (\text{intrev } \sigma). (\text{prefix } ia \ (\text{intrev } \sigma) \models (\neg(f))^r))$ 
      )
    )
    by (simp add: reverse-d-def)
  also have  $\dots =$ 
    (
      (  $(\sigma \models f) \wedge \text{intlen } (\sigma) = 0$ )  $\vee$ 
      (  $\text{intlen } (\sigma) > 0 \wedge (\sigma \models f) \wedge$ 
         $(\forall ia < \text{intlen } (\text{intrev } \sigma). (\text{intrev } (\text{prefix } ia \ (\text{intrev } \sigma)) \models (\neg(f))))$ 
      )
    )
    by (simp add: reverse-d-def)
  also have  $\dots =$ 
    (
      (  $(\sigma \models f) \wedge \text{intlen } (\sigma) = 0$ )  $\vee$ 
      (  $\text{intlen } (\sigma) > 0 \wedge (\sigma \models f) \wedge$ 
         $(\forall ia < \text{intlen } (\sigma). ((\text{suffix } ((\text{intlen } \sigma) - ia) \ (\sigma)) \models (\neg(f))))$ 
      )
    )
    by (simp add: interval-intrev-prefix)
  finally show
     $(\sigma \models \triangleleft f) =$ 
    (  $(\sigma \models f) \wedge \text{intlen } \sigma = 0$ )  $\vee$ 
    (  $\text{intlen } \sigma > 0 \wedge (\sigma \models f) \wedge$ 
       $(\forall ia < \text{intlen } \sigma. (\text{suffix } ((\text{intlen } \sigma) - ia) \ \sigma \models \neg(f)))$ 
    )
    .
qed

```

14.3.2 Various Semantic Lemmas

lemma *DiLensem*:

$(\sigma \models di \ (f \wedge \text{len}(i))) =$
 ($(\text{prefix } i \ \sigma \models f) \wedge i \leq \text{intlen } \sigma$)

using *interval-prefix-length-good* **by** (*simp add: di-defs len-defs, auto*)

lemma *PrefixFstsem*:


```

( (prefix i σ ⊨ ▷f) ∧ i ≤ intlen σ) =
  ( i ≤ intlen σ ∧
    (
      ( (prefix i σ ⊨ f) ∧ i = 0) ∨
      ( i > 0 ∧ (prefix i σ ⊨ f) ∧ (∀ ia < i. (prefix ia σ ⊨ ¬f)))
    )
  )

```

proof –

```

have 1: ( ((prefix i σ) ⊨ ▷f)) =
  (
    ( ((prefix i σ) ⊨ f) ∧ intlen (prefix i σ) = 0) ∨
    ( intlen (prefix i σ) > 0 ∧ ((prefix i σ) ⊨ f) ∧
      (∀ ia < intlen (prefix i σ). (prefix ia (prefix i σ) ⊨ ¬f)) )
  )

```

using *Fstsem* **by** *blast*

```

hence 2: ( ((prefix i σ) ⊨ ▷f) ∧ i ≤ intlen σ) =
  ( i ≤ intlen σ ∧ (
    ( ((prefix i σ) ⊨ f) ∧ intlen (prefix i σ) = 0) ∨
    ( intlen (prefix i σ) > 0 ∧ ((prefix i σ) ⊨ f) ∧
      (∀ ia < intlen (prefix i σ). (prefix ia (prefix i σ) ⊨ ¬f)) )
  )
)

```

by *auto*

```

hence 3: ( ((prefix i σ) ⊨ ▷f) ∧ i ≤ intlen σ) =
  ( i ≤ intlen σ ∧ (
    ( ((prefix i σ) ⊨ f) ∧ i = 0) ∨
    ( i > 0 ∧ ((prefix i σ) ⊨ f) ∧ (∀ ia < i. (prefix ia (prefix i σ) ⊨ ¬f)))
  )
)

```

by *auto*

```

hence 4: ( ((prefix i σ) ⊨ ▷f) ∧ i ≤ intlen σ) =
  ( i ≤ intlen σ ∧ (
    ( ((prefix i σ) ⊨ f) ∧ i = 0) ∨
    ( i > 0 ∧ ((prefix i σ) ⊨ f) ∧ (∀ ia < i. (prefix ia σ ⊨ ¬f)))
  )
)

```

using *interval-pref-pref-3* **using** *less-imp-add-positive* **by** *fastforce*

from 4 **show** *?thesis* **by** *auto*

qed

lemma *PrefixFstAndsem*:

```

( (prefix i σ ⊨ ▷f ∧ g) ∧ i ≤ intlen σ) =
  ( i ≤ intlen σ ∧
    (
      ( (prefix i σ ⊨ f ∧ g) ∧ i = 0) ∨
      ( i > 0 ∧ (prefix i σ ⊨ f ∧ g) ∧ (∀ ia < i. (prefix ia σ ⊨ ¬f)))
    )
  )

```

using *PrefixFstsem* **by** (*metis* *unl-lift2*)

lemma *DiLenFstsem*:

$$(\sigma \models di (\triangleright f \wedge len(i))) =$$

$$(\ i \leq intlen \ \sigma \wedge$$

$$(\$$

$$(\ prefix \ i \ \sigma \models f) \wedge i = 0) \vee$$

$$(i > 0 \wedge (prefix \ i \ \sigma \models f) \wedge (\forall ia < i. (prefix \ ia \ \sigma \models \neg f)))$$

$$)$$

$$)$$

by (*simp add: DiLensem PrefixFstsem*)

lemma *DiLenFstAndsem*:

$$(\sigma \models di ((\triangleright f \wedge g) \wedge len(i))) =$$

$$(\ i \leq intlen \ \sigma \wedge$$

$$(\$$

$$(\ prefix \ i \ \sigma \models f \wedge g) \wedge i = 0) \vee$$

$$(i > 0 \wedge (prefix \ i \ \sigma \models f \wedge g) \wedge (\forall ia < i. (prefix \ ia \ \sigma \models \neg f)))$$

$$)$$

$$)$$

using *DiLensem PrefixFstAndsem* **by** *metis*

lemma *FstLenSamesem*:

$$(\ (\ i \leq intlen \ \sigma \wedge$$

$$(\$$

$$(\ prefix \ i \ \sigma \models f) \wedge i = 0) \vee$$

$$(i > 0 \wedge (prefix \ i \ \sigma \models f) \wedge (\forall ia < i. (prefix \ ia \ \sigma \models \neg f)))$$

$$)$$

$$) \wedge$$

$$(j \leq intlen \ \sigma \wedge$$

$$(\$$

$$(\ prefix \ j \ \sigma \models f) \wedge j = 0) \vee$$

$$(j > 0 \wedge (prefix \ j \ \sigma \models f) \wedge (\forall ia < j. (prefix \ ia \ \sigma \models \neg f)))$$

$$)$$

$$) \longrightarrow (i=j)$$

by (*metis not-less-iff-gr-or-eq unl-lift*)

14.4 Theorems

14.4.1 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*:

$\vdash \text{len}(n+1) = \text{skip}; \text{len}(n)$
by (*simp add: len-d-def*)

lemma *LenEqvLenChopLen*:

$\vdash \text{len}(i+j) = \text{len}(i); \text{len}(j)$

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 LenNPlusOneA add.commute add-Suc inteq-reflection plus-1-eq-Suc*)

qed

lemma *LenNPlusOneB*:

$\vdash \text{len}(n+1) = \text{len}(n); \text{skip}$

proof –

have 1: $\vdash \text{len}(n+1) = \text{len}(n); \text{len}(1)$ **by** (*rule LenEqvLenChopLen*)

have 2: $\vdash \text{len}(1) = \text{skip}$ **by** (*rule LenOneEqvSkip*)

hence 3: $\vdash \text{len}(n); \text{len}(1) = \text{len}(n); \text{skip}$ **using** *RightChopEqvChop* **by** *blast*

from 1 3 **show** ?thesis **by** *fastforce*

qed

lemma *LenCommute*:

$\vdash (\text{skip}; (\text{len } n)) = (\text{len } n); \text{skip}$

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*:

$\vdash \text{skip}; \# \text{True} = \# \text{True}; \text{skip}$

using *TrueChopSkipEqvSkipChopTrue* **by** *fastforce*

lemma *PowerCommute*:

$\vdash (f; (\text{power } f \ n)) = ((\text{power } f \ n); f)$

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:

$\vdash (\text{power skip } n)^r = (\text{power skip } n)$

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:

$\vdash (\text{len } k)^r = (\text{len } k)$

proof

(induct k)

case 0

then show ?case

using LenZeroEqvEmpty REmptyEqvEmpty inteq-reflection **by** force

next

case (Suc k)

then show ?case

by (metis PowerRev len-d-def)

qed

lemma PowerSkipEqvLen:

$\vdash (\text{power skip } n) = (\text{len } n)$

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 (simp add: Valid-def len-defs chop-defs interval-prefix-length interval-suffix-length, blast)

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 len-defs chop-defs interval-prefix-length interval-suffix-length, 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: $\vdash (f0;(\# \text{True} \wedge \text{len}(1)) \wedge f1;(\# \text{True} \wedge \text{len}(1))) = (f0;\text{skip} \wedge f1;\text{skip})$

using 3 4 **by** *fastforce*

have 6: $\vdash (f0 \wedge f1);(\# \text{True} \wedge \text{len}(1)) = (f0 \wedge f1);\text{skip}$

using 2 *RightChopEqvChop* **by** *blast*

from 1 5 6 **show** ?thesis **by** *fastforce*

qed

lemma *BiAndChopSkipEqv*:

$\vdash (bi (f \wedge g));\text{skip} = ((bi f);\text{skip} \wedge (bi g);\text{skip})$

proof —

have 1: $\vdash bi (f \wedge g) = ((bi f) \wedge (bi g))$

by (*simp add: bi-defs Valid-def, auto*)

hence 2: $\vdash (bi (f \wedge g));\text{skip} = (bi f \wedge bi g);\text{skip}$

by (*rule LeftChopEqvChop*)

have 3: $\vdash (bi f \wedge bi g);\text{skip} = ((bi f);\text{skip} \wedge (bi g);\text{skip})$

using *ChopSkipAndChopSkip* **by** *fastforce*

from 2 3 **show** ?thesis **by** *fastforce*

qed

lemma *DiAndChopSkipEqv*:

$\vdash (di (f \wedge g));\text{skip} \longrightarrow (di f);\text{skip} \wedge (di g);\text{skip}$

proof —

have 1: $\vdash di (f \wedge g) \longrightarrow (di f) \wedge (di g)$

by (*simp add: DiAndImpAnd*)

hence 2: $\vdash (di (f \wedge g));\text{skip} \longrightarrow (di f \wedge di g);\text{skip}$

by (*rule LeftChopImpChop*)

have 3: $\vdash (di f \wedge di g);\text{skip} = ((di f);\text{skip} \wedge (di g);\text{skip})$

using *ChopSkipAndChopSkip* **by** *fastforce*
from 2 3 **show** *?thesis* **by** *fastforce*
qed

lemma *ChopEmptyAndEmpty*:
 $\vdash (f;g \wedge \text{empty}) = (f \wedge g \wedge \text{empty})$
by (*simp add: Valid-def chop-defs empty-defs,*
metis interval-prefix-intlen interval-suffix-zero le-zero-eq)

lemma *ChopSkipImpMore*:
 $\vdash f;\text{skip} \longrightarrow \text{more}$
using *ChopImpDiamond MoreEqvSkipChopTrue SkipTrueEqvTrueSkip TrueChopEqvDiamond* **by** *fastforce*

lemma *MoreEqvMoreChopTrue*:
 $\vdash \text{more} = \text{more};\# \text{True}$
proof –
have 1: $\vdash \text{more} = \text{skip};\# \text{True}$
using *MoreEqvSkipChopTrue* **by** *blast*
have 2: $\vdash \# \text{True} = \# \text{True};\# \text{True}$
by (*simp add: Valid-def chop-defs, auto*)
hence 3: $\vdash \text{skip};\# \text{True} = \text{skip};(\# \text{True};\# \text{True})$
using *RightChopEqvChop* **by** *blast*
have 4: $\vdash \text{skip};(\# \text{True};\# \text{True}) = (\text{skip};\# \text{True});\# \text{True}$
using *ChopAssoc* **by** *blast*
have 5: $\vdash (\text{skip};\# \text{True});\# \text{True} = \text{more};\# \text{True}$
using *MoreEqvSkipChopTrue* **by** (*simp add: more-d-def next-d-def*)
from 1 3 4 5 **show** *?thesis* **by** *fastforce*
qed

lemma *NotNotChopSkip*:
 $\vdash (\neg(\neg f);\text{skip}) = (\text{empty} \vee (f;\text{skip}))$
by (*metis WprevEqvEmptyOrPrev prev-d-def wprev-d-def*)

lemma *NotChopFixed*:
 $\vdash (\neg(f;(g \wedge \text{len}(k)))) = (\neg(\Diamond(g \wedge \text{len}(k))) \vee ((\neg f);(g \wedge \text{len}(k))))$
by (*simp add: len-defs Valid-def sometimes-defs chop-defs interval-suffix-length,*
smt diff-diff-cancel)

lemma *NotFixedChop*:
 $\vdash (\neg((g \wedge \text{len}(k));f)) = (\neg(\text{di}(g \wedge \text{len}(k))) \vee ((g \wedge \text{len}(k));(\neg f)))$
by (*simp add: len-defs Valid-def di-defs chop-defs interval-prefix-length, auto*)

lemma *NotChopNotSkip*:
 $\vdash (\neg(f;\text{skip})) = (\text{empty} \vee ((\neg f);\text{skip}))$
proof –
have 1: $\vdash (\neg(\neg(\neg f));\text{skip})) = (\text{empty} \vee ((\neg f);\text{skip}))$ **using** *NotNotChopSkip* **by** *blast*
have 2: $\vdash (\neg(\neg(\neg f));\text{skip})) = (\neg(f;\text{skip}))$ **by** *auto*
from 1 2 **show** *?thesis* **by** *auto*
qed

14.4.2 Additional ITL theorems

lemma *BiOrBilmpBiOr*:

$\vdash bi\ f \vee bi\ g \longrightarrow bi(f \vee g)$

proof —

have 1: $\vdash f \longrightarrow f \vee g$ **by** *auto*

hence 2: $\vdash bi\ f \longrightarrow bi(f \vee g)$ **by** (*rule BilmpBiRule*)

have 3: $\vdash g \longrightarrow f \vee g$ **by** *auto*

hence 4: $\vdash bi\ g \longrightarrow bi(f \vee g)$ **by** (*rule BilmpBiRule*)

from 2 4 **show** *?thesis* **by** *fastforce*

qed

lemma *MoreAndBilmpBiChopSkip*:

$\vdash more \wedge bi\ f \longrightarrow (bi\ f);skip$

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 *NotAndMoreEqvEmptyOr* **by** *fastforce*

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 *BiBiOrlmpBi*:

$\vdash bi\ (bi\ f \vee bi\ g) \longrightarrow bi\ f \vee bi\ g$

using *BiElim* **by** *auto*

lemma *BilmpBiBiOr*:

$\vdash bi\ f \longrightarrow bi\ (bi\ f \vee bi\ g)$

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** *BilmpBiRule* **by** *blast*

have 3: $\vdash bi\ (bi\ f) = bi\ f$ **using** *BiEqvBiBi* **by** *fastforce*

from 2 3 **show** *?thesis* **by** *fastforce*

qed

lemma *BilmpBiBiOrB*:

$\vdash bi\ g \longrightarrow bi\ (bi\ f \vee bi\ g)$
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 BilmpBiBiOr BilmpBiBiOrB Prop02 int-iff1)
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 BilInduct:

$\vdash bi(f \longrightarrow wprev f) \wedge f \longrightarrow bi f$

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: $\vdash ((f^r) \longrightarrow wnext(f^r))^r = (f^r)^r \longrightarrow (wnext(f^r))^r$ **by** (simp add: rev-fun2)
have 6: $\vdash (wnext(f^r))^r = wprev(f)$ **using** RRWNNextEqvWPrev **by** blast
have 7: $\vdash ((f^r)^r \longrightarrow (wnext(f^r))^r) = (f \longrightarrow wprev(f))$ **using** 6 3 **by** fastforce
have 8: $\vdash bi((f^r)^r \longrightarrow (wnext(f^r))^r) = bi(f \longrightarrow wprev(f))$ **using** 7 3 BiEqvBi **by** blast

have 9: $\vdash (\Box((f') \rightarrow \text{wnext}(f')))^r = \text{bi} \ (((f') \rightarrow \text{wnext}(f'))^r)$ **using** *RBoxEqvBi* **by** *blast*
have 10: $\vdash (\Box((f') \rightarrow \text{wnext}(f')))^r = \text{bi} \ (f \rightarrow \text{wprev}(f))$ **using** 8 9 5 *int-eq* **by** *fastforce*
have 11: $\vdash (\Box((f') \rightarrow \text{wnext}(f')) \wedge f' \rightarrow \Box(f'))^r =$
 $((\Box((f') \rightarrow \text{wnext}(f'))^r \wedge (f')^r \rightarrow (\Box(f'))^r))$ **by** (*metis int-eq rev-fun2*)
have 12: $\vdash ((\Box((f') \rightarrow \text{wnext}(f')))^r \wedge (f')^r \rightarrow (\Box(f'))^r) =$
 $(\text{bi} \ (f \rightarrow \text{wprev}(f)) \wedge f \rightarrow \text{bi} \ f)$ **using** 8 3 4 10 **by** *fastforce*
from 2 11 12 **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *PrevLoop*:

assumes $\vdash f \rightarrow \text{prev} \ f$

shows $\vdash \neg f$

proof —

have 1: $\vdash f \rightarrow \text{prev} \ f$ **using** *assms* **by** *auto*

hence 2: $\vdash f \rightarrow (\text{more} \wedge \text{wprev} \ f)$

by (*smt intl int-eq more-defs prev-defs Prop10 unl-lift2 wprev-defs*)

hence 3: $\vdash f \rightarrow \text{wprev} \ f$ **by** *auto*

hence 4: $\vdash \text{bi} \ (f \rightarrow \text{wprev} \ f)$ **by** (*rule BiGen*)

have 5: $\vdash \text{bi} \ (f \rightarrow \text{wprev} \ f) \wedge f \rightarrow \text{bi} \ f$ **by** (*rule BiInduct*)

hence 6: $\vdash \text{bi} \ (f \rightarrow \text{wprev} \ f) \rightarrow (f \rightarrow \text{bi} \ f)$ **by** *fastforce*

have 7: $\vdash (f \rightarrow \text{bi} \ f)$ **using** 4 6 *MP* **by** *blast*

have 8: $\vdash \text{bi} \ f \rightarrow f$ **by** (*rule BiElim*)

have 9: $\vdash f = \text{bi} \ f$ **using** 7 8 **by** *fastforce*

have 10: $\vdash f \rightarrow \text{more}$ **using** 2 **by** *auto*

hence 11: $\vdash \text{bi} \ f \rightarrow \text{bi} \ \text{more}$ **using** *BilmpBiRule* **by** *blast*

have 12: $\vdash \neg(\text{bi} \ \text{more})$ **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*:

$\vdash \text{prev} \ f \rightarrow \neg(\text{prev} \ (\neg f))$

by (*metis (no-types, lifting) NextImpNotNextNot RPrevEqvNext ReverseEqv inteq-reflection rev-fun1 rev-fun2*)

lemma *BiEqvAndWprevBi*:

$\vdash \text{bi} \ f = (f \wedge \text{wprev} \ (\text{bi} \ f))$

using *BoxEqvAndWnextBox*

by (*metis (no-types, lifting) RBiEqvBox RRand RBoxEqvBi RWPrevEqvWNext int-eq*)

lemma *DiIntroLoop*:

assumes $\vdash (f \wedge \neg g) \rightarrow \text{prev} \ f$

shows $\vdash f \rightarrow \text{di} \ g$

using *assms DiamondIntro*

by (*metis (no-types, lifting) RDiEqvDiamond RPrevEqvNext ReverseEqv inteq-reflection rev-fun2 rev-fun1*)

lemma *DiEqvOrChopMore*:

$\vdash \text{di} \ f = (f \vee f ; \text{more})$

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: $\vdash more = \#True; skip$
using MoreEqvSkipChopTrue SkipTrueEqvTrueSkip **by** fastforce
hence 6: $\vdash f; more = f; (\#True; skip)$
using RightChopEqvChop **by** blast
have 7: $\vdash f; (\#True; skip) = (f; \#True); skip$
by (rule ChopAssoc)
have 8: $\vdash f; more = prev\ (di\ f)$
using 6 7 **by** (metis di-d-def int-eq prev-d-def)
have 9: $\vdash g; more = g; (\#True; skip)$
using 5 RightChopEqvChop **by** blast
have 10: $\vdash g; (\#True; skip) = (g; \#True); skip$
by (rule ChopAssoc)
have 11: $\vdash g; more = prev\ (di\ g)$
using 9 10 **by** (metis di-d-def int-eq prev-d-def)
have 12: $\vdash (f; more \wedge g; more) = (prev\ (di\ f) \wedge prev\ (di\ g))$
using 8 11 **by** fastforce
hence 13: $\vdash (f; more \wedge g; more) = prev\ (di\ f \wedge di\ g)$
by (metis ChopSkipAndChopSkip int-eq prev-d-def)
have 14: $\vdash (di\ f \wedge di\ g) =$
 $((f \wedge g) \vee (f \wedge g; more) \vee (g \wedge f; more)) \vee (f; more \wedge g; more)$
using 3 4 **by** auto
have 15: $\vdash (di\ f \wedge di\ g) =$
 $((f \wedge g) \vee (f \wedge g; more) \vee (g \wedge f; more)) \vee prev\ (di\ f \wedge di\ g)$
using 13 14 **by** fastforce
hence 16: $\vdash (di\ f \wedge di\ g) \longrightarrow$
 $((f \wedge g) \vee (f \wedge g; more) \vee (g \wedge f; more)) \vee prev\ (di\ f \wedge di\ g)$
by fastforce
hence 17: $\vdash (di\ f \wedge di\ g) \wedge \neg((f \wedge g) \vee (f \wedge g; more) \vee (g \wedge f; more)) \longrightarrow$
 $prev\ (di\ f \wedge di\ g)$

by *fastforce*
 hence 18: $\vdash (di\ f \wedge di\ g) \longrightarrow di((f \wedge g) \vee (f \wedge g;more) \vee (g \wedge f;more))$
 using *DilIntroLoop* by *blast*
 have 19: $\vdash di((f \wedge g) \vee (f \wedge g;more) \vee (g \wedge f;more)) =$
 $(di(f \wedge g) \vee di(f \wedge g;more) \vee di(g \wedge f;more))$
 by (*meson DiOrEqv Prop06*)
 have 20: $\vdash f \longrightarrow di\ f$
 using *DilIntro* 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 *DilmpDi* 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: $\vdash g;more \longrightarrow di\ g$
 by (*simp add: ChopImpDi*)
 hence 25: $\vdash f \wedge g;more \longrightarrow f \wedge di\ g$
 by *auto*
 hence 26: $\vdash di(f \wedge g;more) \longrightarrow di(f \wedge di\ g)$
 using *DilmpDi* by *blast*
 hence 27: $\vdash di(f \wedge g;more) \longrightarrow di(f \wedge di\ g) \vee di(g \wedge di\ f)$
 by *auto*
 have 28: $\vdash f;more \longrightarrow di\ f$
 by (*simp add: ChopImpDi*)
 hence 29: $\vdash g \wedge f;more \longrightarrow g \wedge di\ f$
 by *auto*
 hence 30: $\vdash di(g \wedge f;more) \longrightarrow di(g \wedge di\ f)$
 using *DilmpDi* by *blast*
 hence 31: $\vdash di(g \wedge f;more) \longrightarrow di(f \wedge di\ g) \vee di(g \wedge di\ f)$
 by *auto*
 have 32: $\vdash di(f \wedge g) \vee di(f \wedge g;more) \vee di(g \wedge f;more) \longrightarrow$
 $di(f \wedge di\ g) \vee di(g \wedge di\ f)$
 using 23 27 31 by *fastforce*
 have 33: $\vdash di((f \wedge g) \vee (f \wedge g;more) \vee (g \wedge f;more)) \longrightarrow$
 $di(f \wedge di\ g) \vee di(g \wedge di\ f)$
 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 *DilIntro* 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 *DilmpDi* 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 *DilIntro* 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 *DilmpDi* **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*:

$\vdash \Box (init\ w) = bi\ (fin\ (init\ w))$

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: $\vdash (\neg (\Box (\text{init } (\neg w)))) = (\neg (\text{bi } (\text{fin } (\text{init } (\neg w))))$
using 1 by auto
have 3: $\vdash \Diamond (\neg (\text{init } (\neg w))) = \text{di } (\neg (\text{fin } (\text{init } (\neg w))))$
using 2 by (simp add: always-d-def bi-d-def)
have 4: $\vdash \Diamond (\text{init } w) = \text{di } (\neg (\text{fin } (\text{init } (\neg w))))$
by (metis 3 DiEqvNotBiNot DiState Initprop(2) StateEqvBi int-eq)
have 5: $\vdash \Diamond (\text{init } w) = \text{di } (\text{fin } (\text{init } w))$ **using 4 FinNotStateEqvNotFinState**
by (metis DiEqvNotBiNot DiNotEqvNotBi inteq-reflection)
from 1 2 3 4 5 show ?thesis by simp
qed

lemma OrDiEqvDi:
 $\vdash (f \vee \text{di } f) = \text{di } f$
proof –
have 1: $\vdash f \longrightarrow \text{di } f$ **using Dilntro by blast**
from 1 show ?thesis by auto
qed

lemma AndDiEqv:
 $\vdash (f \wedge \text{di } f) = f$
proof –
have 1: $\vdash f \longrightarrow \text{di } f$ **using Dilntro by blast**
from 1 show ?thesis by auto
qed

lemma BiEmptyEqvEmpty:
 $\vdash \text{bi empty} = \text{empty}$
proof –
have 1: $\vdash \text{bi empty} = (\neg (\text{di } (\neg \text{empty})))$ **by (simp add: bi-d-def)**
have 2: $\vdash (\neg (\text{di } (\neg \text{empty}))) = (\neg ((\neg \text{empty}); \# \text{True}))$ **by (simp add: di-d-def)**
have 3: $\vdash (\neg ((\neg \text{empty}); \# \text{True})) = (\neg (\text{more}; \# \text{True}))$ **by (simp add: empty-d-def)**
have 4: $\vdash \text{more}; \# \text{True} = \text{more}$ **using MoreEqvMoreChopTrue by auto**
hence 5: $\vdash (\neg (\text{more}; \# \text{True})) = (\neg \text{more})$ **by fastforce**
from 1 2 3 5 show ?thesis using NotEmptyEqvMore by fastforce
qed

lemma EmptyChopSkipInduct:
assumes $\vdash \text{empty} \longrightarrow f$
 $\vdash \text{prev } f \longrightarrow f$
shows $\vdash f$
proof –
have 1: $\vdash \text{empty} \longrightarrow f$ **using assms(1) by auto**
have 2: $\vdash \text{prev } f \longrightarrow f$ **using assms(2) by blast**
have 3: $\vdash (\text{empty} \vee \text{prev } f) \longrightarrow f$ **using 1 2 by fastforce**
have 4: $\vdash \text{wprev } f = (\text{empty} \vee \text{prev } f)$ **by (simp add: WprevEqvEmptyOrPrev)**
hence 5: $\vdash \text{wprev } f \longrightarrow f$ **using 3 by fastforce**
hence 6: $\vdash \neg f \longrightarrow \neg (\text{wprev } f)$ **by fastforce**
hence 7: $\vdash \neg f \longrightarrow \text{prev } (\neg f)$ **by (simp add: wprev-d-def)**
hence 8: $\vdash \neg \neg f$ **by (rule PrevLoop)**
from 8 show ?thesis by auto

qed

lemma *MoreImplmpChopSkipEqv*:

$\vdash \text{more} \longrightarrow ((f \longrightarrow g); \text{skip} = ((f; \text{skip}) \longrightarrow (g; \text{skip})))$

proof –

have 01: $\vdash (f \longrightarrow g) = (\neg f \vee g)$ **by** *auto*

hence 02: $\vdash (f \longrightarrow g); \text{skip} = (\neg f \vee g); \text{skip}$ **by** (*simp add: LeftChopEqvChop*)

hence 1: $\vdash (\text{more} \wedge (f \longrightarrow g); \text{skip}) = (\text{more} \wedge (\neg f \vee g); \text{skip})$ **by** *fastforce*

have 2: $\vdash (\neg f \vee g); \text{skip} = ((\neg f); \text{skip} \vee g; \text{skip})$

using *OrChopEqv* **by** *auto*

hence 3: $\vdash (\text{more} \wedge (\neg f \vee g); \text{skip}) = (\text{more} \wedge ((\neg f); \text{skip} \vee g; \text{skip}))$

by *auto*

have 4: $\vdash (\neg((\neg f); \text{skip})) = (\text{empty} \vee (f; \text{skip}))$

using *NotNotChopSkip* **by** *blast*

hence 5: $\vdash ((\neg f); \text{skip}) = (\neg(\text{empty} \vee (f; \text{skip})))$

by *fastforce*

have 6: $\vdash \neg(\text{empty} \vee (f; \text{skip})) = (\text{more} \wedge \neg(f; \text{skip}))$

using 5 *NotChopSkipEqvMoreAndNotChopSkip* **by** *fastforce*

have 7: $\vdash ((\neg f); \text{skip} \vee g; \text{skip}) = ((\text{more} \wedge \neg(f; \text{skip})) \vee g; \text{skip})$

using 5 6 **by** *fastforce*

hence 8: $\vdash (\text{more} \wedge (\neg f; \text{skip} \vee g; \text{skip})) = (\text{more} \wedge ((\text{more} \wedge \neg(f; \text{skip})) \vee g; \text{skip}))$

by *auto*

have 9: $\vdash (\text{more} \wedge ((\text{more} \wedge \neg(f; \text{skip})) \vee g; \text{skip})) = (\text{more} \wedge (\neg(f; \text{skip}) \vee g; \text{skip}))$

by *auto*

have 10: $\vdash (\text{more} \wedge (\neg(f; \text{skip}) \vee g; \text{skip})) = (\text{more} \wedge ((f; \text{skip}) \longrightarrow (g; \text{skip})))$

by *auto*

have 11: $\vdash (\text{more} \wedge (f \longrightarrow g); \text{skip}) = (\text{more} \wedge ((f; \text{skip}) \longrightarrow (g; \text{skip})))$

using 1 2 3 8 9 10 7 **by** *fastforce*

from 11 **show** *?thesis* **using** *MP* **by** *fastforce*

qed

lemma *MoreImplmpPrevEqv*:

$\vdash \text{more} \longrightarrow (\text{prev}(f \longrightarrow g) = (\text{prev } f \longrightarrow \text{prev } g))$

by (*simp add: MoreImplmpChopSkipEqv prev-d-def*)

lemma *BiBoxNotEqvNotTrueChopChopTrue*:

$\vdash \text{bi}(\Box (\neg f)) = (\neg((\# \text{True}; f); \# \text{True}))$

by (*simp add: bi-d-def always-d-def di-d-def sometimes-d-def*)

lemma *DiAndEmptyEqvAndEmpty*:

$\vdash (di f \wedge \text{empty}) = (f \wedge \text{empty})$

proof –

have 1 : $\vdash di f = (f \vee di f; \text{skip})$

using *DiEqvOrDiChopSkipB* **by** *blast*

hence 2: $\vdash (di f \wedge \text{empty}) = ((f \vee di f; \text{skip}) \wedge \text{empty})$

by *fastforce*

have 3 : $\vdash ((f \vee di f; \text{skip}) \wedge \text{empty}) = ((f \wedge \text{empty}) \vee (di f; \text{skip} \wedge \text{empty}))$

by *auto*

have 4: $\vdash \neg(di f; \text{skip} \wedge \text{empty})$

by (metis AndChopB AndDiEqv ChopAndEmptyEqvEmptyChopEmpty DiEmpty MoreEqvSkipChopTrue
 TrueChopSkipEqvSkipChopTrue empty-d-def int-eq int-eq-true int-simps(14) int-simps(21)
 lift-and-com)
 hence 5 : $\vdash ((f \wedge \text{empty}) \vee (di\ f; \text{skip} \wedge \text{empty})) = (f \wedge \text{empty})$
 by auto
 from 2 3 5 show ?thesis by fastforce
 qed

14.4.3 Strict initial intervals

lemma *DsMoreDi*:

$\vdash ds\ f = (\text{more} \wedge (di\ f); \text{skip})$

proof —

have 1: $\vdash ds\ f = (\neg(bs\ (\neg f)))$
 by (simp add: ds-d-def)
 have 2: $\vdash (\neg(bs\ (\neg f))) = (\neg(\text{empty} \vee (bi\ (\neg f)); \text{skip}))$
 by (simp add: bs-d-def)
 have 3: $\vdash (\neg(\text{empty} \vee (bi\ (\neg f)); \text{skip})) = (\neg \text{empty} \wedge \neg((bi\ (\neg f)); \text{skip}))$
 by auto
 have 4: $\vdash (\neg \text{empty} \wedge \neg((bi\ (\neg f)); \text{skip})) = (\text{more} \wedge \neg((bi\ (\neg f)); \text{skip}))$
 using NotEmptyEqvMore by auto
 have 5: $\vdash (\text{more} \wedge \neg((bi\ (\neg f)); \text{skip})) = (\text{more} \wedge \neg(\neg(di\ f); \text{skip}))$
 by (metis DiEqvNotBiNot DiIntro DiSkipEqvMore NotChopSkipEqvMoreAndNotChopSkip
 Prop10 RightChopImpMoreRule int-simps(4) inteq-reflection lift-and-com)
 have 6: $\vdash (\text{more} \wedge \neg(\neg(di\ f); \text{skip})) = (\text{more} \wedge (\text{empty} \vee (di\ f); \text{skip}))$
 using NotNotChopSkip by fastforce
 have 7: $\vdash (\text{more} \wedge (\text{empty} \vee (di\ f); \text{skip})) = (\text{more} \wedge (di\ f); \text{skip})$
 using NotEmptyEqvMore by auto
 from 1 2 3 4 5 6 7 show ?thesis by fastforce
 qed

lemma *DsDi*:

$\vdash ds\ f = (di\ f); \text{skip}$

proof —

have 1: $\vdash ds\ f = (\text{more} \wedge (di\ f); \text{skip})$ by (rule DsMoreDi)
 have 2: $\vdash (di\ f); \text{skip} \longrightarrow \text{more}$ by (metis DiIntro DiSkipEqvMore RightChopImpMoreRule int-eq)
 hence 3: $\vdash (\text{more} \wedge (di\ f); \text{skip}) = (di\ f); \text{skip}$ by auto
 from 1 2 show ?thesis by fastforce
 qed

lemma *BsEqvNotDsNot*:

$\vdash bs\ f = (\neg(ds\ (\neg f)))$

proof —

have 1: $\vdash ds\ (\neg f) = (\text{more} \wedge (di\ (\neg f)); \text{skip})$
 by (rule DsMoreDi)
 hence 2: $\vdash (\neg(ds\ (\neg f))) = (\neg(\text{more} \wedge (di\ (\neg f)); \text{skip}))$
 by auto
 have 3: $\vdash (\neg(\text{more} \wedge (di\ (\neg f)); \text{skip})) = (\text{empty} \vee \neg((di\ (\neg f)); \text{skip}))$
 using NotEmptyEqvMore by auto
 have 4: $\vdash (\text{empty} \vee \neg((di\ (\neg f)); \text{skip})) = (\text{empty} \vee \neg(\neg(bi\ f); \text{skip}))$

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 BiOrBImpBiOr*)
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** (*simp add: empty-d-def, auto*)
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 BilmpBiRule*)

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 BilmpBiBiOr BilmpBiBiOrB Prop02 int-iff1*)

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* **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* **using** *BiEqvAndEmptyOrBiChopSkip* **by** *fastforce*

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 *StatImpBs*:

$\vdash init\ w \longrightarrow bs\ (init\ w)$

proof —

have 1: $\vdash \text{init } w = \text{bi } (\text{init } w)$ **by** (rule StateEqvBi)
have 2: $\vdash \text{bi } (\text{init } w) \longrightarrow \text{bs } (\text{init } w)$ **by** (rule BilmpBs)
from 1 2 **show** ?thesis **using** StatImpBi **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: $\vdash (di \ f \wedge di \ g); \text{skip} = (di \ (f \wedge di \ g) \vee di \ (g \wedge di \ f)); \text{skip}$
by (rule LeftChopEqvChop)
have 3: $\vdash (di \ f \wedge di \ g); \text{skip} = ((di \ f); \text{skip} \wedge (di \ g); \text{skip})$
using ChopSkipAndChopSkip **by** fastforce
have 4: $\vdash ((di \ f); \text{skip} \wedge (di \ g); \text{skip}) = (di \ (f \wedge di \ g) \vee di \ (g \wedge di \ f)); \text{skip}$
using 2 3 **by** fastforce
have 5: $\vdash (di \ (f \wedge di \ g) \vee di \ (g \wedge di \ f)); \text{skip} = (di \ (f \wedge di \ g); \text{skip} \vee di \ (g \wedge di \ f); \text{skip})$
using OrChopEqv **by** blast
have 6: $\vdash ds \ f = (di \ f); \text{skip}$
using DsDi **by** blast
have 7: $\vdash ds \ g = (di \ g); \text{skip}$
using DsDi **by** blast
have 8: $\vdash ((di \ f); \text{skip} \wedge (di \ g); \text{skip}) = (ds \ f \wedge ds \ g)$
using 6 7 **by** fastforce
have 9: $\vdash ds \ (f \wedge di \ g) = di \ (f \wedge di \ g); \text{skip}$
using DsDi **by** blast
have 10: $\vdash ds \ (g \wedge di \ f) = di \ (g \wedge di \ f); \text{skip}$
using DsDi **by** blast
have 11: $\vdash (di \ (f \wedge di \ g); \text{skip} \vee di \ (g \wedge di \ f); \text{skip}) = (ds \ (f \wedge di \ g) \vee ds \ (g \wedge di \ f))$
using 9 10 **by** fastforce
from 4 5 8 11 **show** ?thesis **by** fastforce

qed

lemma BsEqvBiMoreImpChop:

$\vdash \text{bs } f = \text{bi}(\text{more} \longrightarrow f; \text{skip})$

proof –

have 1: $\vdash \text{bs } f = (\text{empty} \vee (\text{bi } f; \text{skip}))$
by (simp add: bs-d-def)
have 2: $\vdash (\text{empty} \vee (\text{bi } f; \text{skip})) = ((\neg(\neg(\text{bi } f)); \text{skip}))$
using NotNotChopSkip **by** fastforce
have 3: $\vdash \neg(\neg(\text{bi } f)); \text{skip} = (\neg(di \ (\neg f)); \text{skip}))$
by (simp add: bi-d-def)
have 4: $\vdash (\neg(di \ (\neg f)); \text{skip})) = (\neg(((\neg f) ; \# \text{True}); \text{skip}))$
by (simp add: di-d-def)
have 5: $\vdash (\neg(((\neg f) ; \# \text{True}); \text{skip})) = (\neg((\neg f) ; (\# \text{True}; \text{skip})))$
using ChopAssocB **by** fastforce
have 6: $\vdash (\neg((\neg f) ; (\# \text{True}; \text{skip}))) = (\neg((\neg f) ; (\text{skip}; \# \text{True})))$
using SkipTrueEqvTrueSkip TrueChopSkipEqvSkipChopTrue RightChopEqvChop **by** fastforce
have 7: $\vdash (\neg((\neg f) ; (\text{skip}; \# \text{True}))) = (\neg(((\neg f) ; \text{skip}); \# \text{True}))$
using ChopAssoc **by** fastforce

have 8: $\vdash (\neg(((\neg f); \text{skip}); \# \text{True})) = (\neg(\text{di}((\neg f); \text{skip})))$
by (*simp add: di-d-def*)
have 9: $\vdash (\neg(\text{di}((\neg f); \text{skip}))) = \text{bi}(\neg((\neg f); \text{skip}))$
using *NotDiEqvBiNot* **by** *blast*
have 10: $\vdash \text{bi}(\neg((\neg f); \text{skip})) = \text{bi}(\text{empty} \vee (f; \text{skip}))$
using *NotNotChopSkip* **using** *BiEqvBi* **by** *blast*
have 11: $\vdash \text{bi}(\text{empty} \vee (f; \text{skip})) = \text{bi}(\neg \text{more} \vee (f; \text{skip}))$
by (*simp add: empty-d-def*)
have 12: $\vdash (\neg \text{more} \vee (f; \text{skip})) = (\text{more} \longrightarrow f; \text{skip})$ **by** *auto*
have 13: $\vdash \text{bi}(\neg \text{more} \vee (f; \text{skip})) = \text{bi}(\text{more} \longrightarrow f; \text{skip})$ **using** 12 **using** *BiEqvBi* **by** *blast*
have 14: $\vdash \text{bs } f = (\neg(((\neg f); \text{skip}); \# \text{True}))$ **using** 1 2 3 4 5 6 7 **by** *fastforce*
have 15: $\vdash (\neg(((\neg f); \text{skip}); \# \text{True})) = \text{bi}(\text{more} \longrightarrow f; \text{skip})$ **using** 8 9 10 11 13 **by** *fastforce*
from 14 15 **show** ?thesis **by** *fastforce*
qed

lemma *BoxMoreStateEqvBsFinState:*

$\vdash \Box(\text{more} \longrightarrow \neg(\text{init } w)) = \text{bs}(\neg(\text{fin}(\text{init } w)))$

proof –

have 1: $\vdash \Box(\text{more} \longrightarrow \neg(\text{init } w)) = (\neg(\Diamond(\neg(\text{more} \longrightarrow \neg(\text{init } w)))))$
by (*simp add: always-d-def*)
have 01: $\vdash (\neg(\text{more} \longrightarrow \neg(\text{init } w))) = (\text{init } w \wedge \text{more})$ **by** *auto*
hence 2: $\vdash \neg(\Diamond(\neg(\text{more} \longrightarrow \neg(\text{init } w)))) = (\neg(\# \text{True}; (\text{init } w \wedge \text{more})))$
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: $\vdash \Box(\neg(\text{init } w) \vee \neg \text{empty}) = \Box(\neg \text{empty} \vee \neg(\text{init } w))$
by (*simp add: BoxEqvBox*)
have 191: $\vdash (\neg \text{empty} \vee \neg(\text{init } w)) = (\text{empty} \longrightarrow \neg(\text{init } w))$
by *auto*
hence 19: $\vdash \Box(\neg \text{empty} \vee \neg(\text{init } w)) = \Box(\text{empty} \longrightarrow \neg(\text{init } w))$
by (*simp add: BoxEqvBox*)
have 20: $\vdash \Box(\text{empty} \longrightarrow \neg(\text{init } w)) = \text{fin}(\neg(\text{init } w))$
by (*simp add: fin-d-def*)
have 21: $\vdash \text{fin}(\neg(\text{init } w)) = (\neg(\text{fin}(\text{init } w)))$
using *FinEqvFin FinNotStateEqvNotFinState Initprop(2)* **by** *fastforce*
have 22: $\vdash \text{bi}(\Box(\neg(\text{init } w) \vee \neg \text{empty})) = \text{bi}(\neg(\text{fin}(\text{init } w)))$
using 18 19 20 21 *BiEqvBi* **by** (*metis int-eq*)
hence 23: $\vdash (\text{bi}(\Box(\neg(\text{init } w) \vee \neg \text{empty}))); \text{skip} = (\text{bi}(\neg(\text{fin}(\text{init } w)))); \text{skip}$
using *RightChopEqvChop* **by** (*simp add: LeftChopEqvChop*)
hence 24: $\vdash (\text{empty} \vee (\text{bi}(\Box(\neg(\text{init } w) \vee \neg \text{empty}))); \text{skip} =$
 $(\text{empty} \vee (\text{bi}(\neg(\text{fin}(\text{init } w)))); \text{skip}))$
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*)
from 1 2 11 16 17 24 25 **show** *?thesis* **by** *fastforce*
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

```

14.4.4 First occurrence

lemma *FstWithAndImp*:

$\vdash \triangleright f \wedge g \longrightarrow \triangleright (f \wedge g)$

proof –

have 1: $\vdash (\triangleright f \wedge g) = ((f \wedge (bs (\neg f))) \wedge g)$

by (*simp add: first-d-def*)

have 2: $\vdash ((f \wedge (bs (\neg f))) \wedge g) = (f \wedge \neg(ds f) \wedge g)$

using *NotDsEqvBsNot* **by** *fastforce*

have 3: $\vdash \neg(ds f) \longrightarrow \neg(ds(f \wedge g))$

using *DsAndImpElimL* **by** *fastforce*

hence 4: $\vdash f \wedge \neg(ds f) \wedge g \longrightarrow f \wedge g \wedge \neg(ds(f \wedge g))$

by *auto*

have 5: $\vdash (f \wedge g \wedge \neg(ds(f \wedge g))) = ((f \wedge g) \wedge (bs (\neg(f \wedge g))))$

using *NotDsEqvBsNot* **by** *fastforce*

have 6: $\vdash ((f \wedge g) \wedge (bs (\neg(f \wedge g)))) = \triangleright(f \wedge g)$

by (*simp add: first-d-def*)

from 1 2 4 5 6 **show** ?thesis **by** *fastforce*

qed

lemma *FstWithOrEqv*:

$\vdash \triangleright(f \vee g) = ((\triangleright f \wedge bs (\neg g)) \vee (\triangleright g \wedge bs (\neg f)))$

proof –

have 1: $\vdash \triangleright(f \vee g) = ((f \vee g) \wedge bs (\neg(f \vee g)))$

by (*simp add: first-d-def*)

have 2: $\vdash (\neg(f \vee g)) = (\neg f \wedge \neg g)$

by *auto*

hence 3: $\vdash bs (\neg(f \vee g)) = bs (\neg f \wedge \neg g)$

using *BsEqvRule* **by** *blast*

have 4: $\vdash bs (\neg f \wedge \neg g) = (bs (\neg f) \wedge bs (\neg g))$

using *BsAndEqv* **by** *fastforce*

have 5: $\vdash ((f \vee g) \wedge bs (\neg(f \vee g))) = ((f \vee g) \wedge bs (\neg f) \wedge bs (\neg g))$

using 3 4 **by** *fastforce*

have 6: $\vdash ((f \vee g) \wedge bs (\neg f) \wedge bs (\neg g)) =$
 $((f \wedge bs (\neg f)) \wedge bs (\neg g)) \vee (g \wedge bs (\neg f) \wedge bs (\neg g))$

by *auto*

have 7: $\vdash ((f \wedge bs (\neg f)) \wedge bs (\neg g)) = (\triangleright f \wedge bs (\neg g))$

by (*simp add: first-d-def*)

have 8: $\vdash (g \wedge bs (\neg f) \wedge bs (\neg g)) = ((g \wedge bs (\neg g)) \wedge bs (\neg f))$

by *auto*

have 9: $\vdash ((g \wedge bs (\neg g)) \wedge bs (\neg f)) = (\triangleright g \wedge bs (\neg f))$

by (*simp add: first-d-def*)

have 10: $\vdash ((f \wedge bs (\neg f)) \wedge bs (\neg g)) \vee (g \wedge bs (\neg f) \wedge bs (\neg g)) =$
 $(\triangleright f \wedge bs (\neg g)) \vee (\triangleright g \wedge bs (\neg f))$

using 7 8 9 by fastforce
 from 1 5 6 10 show ?thesis by (metis 7 8 9 int-eq)
 qed

lemma *FstFstAndEqvFstAnd*:

$\vdash \triangleright(\triangleright f \wedge g) = (\triangleright f \wedge g)$

proof –

have 1: $\vdash (\triangleright f \wedge g) = ((f \wedge (bs (\neg f))) \wedge g)$ by (simp add: first-d-def)
 hence 2: $\vdash \triangleright f \wedge g \longrightarrow (bs (\neg f))$ by auto
 hence 3: $\vdash \triangleright f \wedge g \longrightarrow \triangleright f \wedge g \wedge (bs (\neg f))$ by auto
 have 4: $\vdash \neg f \longrightarrow \neg f \vee \neg(bs (\neg f)) \vee \neg g$ by auto
 hence 5: $\vdash bs (\neg f) \longrightarrow bs(\neg f \vee \neg(bs (\neg f)) \vee \neg g)$ using BslmpBsRule by blast
 have 6: $\vdash (\neg f \vee \neg(bs (\neg f)) \vee \neg g) = (\neg(f \wedge bs (\neg f) \wedge g))$ by auto
 hence 7: $\vdash bs(\neg f \vee \neg(bs (\neg f)) \vee \neg g) = bs(\neg(f \wedge bs (\neg f) \wedge g))$ using BsEqvRule by blast
 have 8: $\vdash ((f \wedge bs (\neg f)) \wedge g) = (\triangleright f \wedge g)$ by (simp add: first-d-def)
 hence 9: $\vdash (\neg(f \wedge bs (\neg f) \wedge g)) = (\neg(\triangleright f \wedge g))$ by auto
 hence 10: $\vdash bs (\neg(f \wedge bs (\neg f) \wedge g)) = bs (\neg(\triangleright f \wedge g))$ using BsEqvRule by blast
 have 11: $\vdash \triangleright f \wedge g \longrightarrow (\triangleright f \wedge g) \wedge bs (\neg(\triangleright f \wedge g))$ using 3 5 7 10 by fastforce
 hence 12: $\vdash \triangleright f \wedge g \longrightarrow \triangleright(\triangleright f \wedge g)$ by (simp add: first-d-def)
 have 13: $\vdash \triangleright(\triangleright f \wedge g) = ((\triangleright f \wedge g) \wedge bs (\neg(\triangleright f \wedge g)))$ by (simp add: first-d-def)
 hence 14: $\vdash \triangleright(\triangleright f \wedge g) \longrightarrow \triangleright f \wedge g$ by auto
 from 12 14 show ?thesis by fastforce
 qed

lemma *FstTrue*:

$\vdash \triangleright \#True = \text{empty}$

proof –

have 1: $\vdash \triangleright \#True = (\#True \wedge bs (\neg \#True))$
 by (simp add: first-d-def)
 have 2: $\vdash bs (\neg \#True) = (\text{empty} \vee (bi (\neg \#True));\text{skip})$
 by (simp add: bs-d-def)
 have 3: $\vdash \neg(bi (\neg \#True))$
 using BiElim by fastforce
 have 4: $\vdash \neg((bi (\neg \#True));\text{skip})$
 by (metis AndChopA BiEqvAndEmptyOrBiChopSkip MoreEqvSkipChopTrue
 NotChopSkipEqvMoreAndNotChopSkip SkipTrueEqvTrueSkip int-eq int-simps(14) int-simps(21))
 have 5: $\vdash bs (\neg \#True) = \text{empty}$
 using 2 4 by fastforce
 from 1 5 show ?thesis by fastforce
 qed

lemma *FstFalse*:

$\vdash \neg(\triangleright \#False)$

proof –

have 1: $\vdash \triangleright \#False = (\#False \wedge bs \#True)$ by (simp add: first-d-def)
 from 1 show ?thesis by auto
 qed

lemma *FstChopFalseEqvFalse*:

$\vdash \neg(\triangleright f ; \#False)$

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: $\vdash (\neg(\text{ds } f) \wedge (\text{ds } f \vee f)) = (\neg(\text{ds } f) \wedge f)$ **by** auto
have 6: $\vdash (\neg(\text{ds } f) \wedge f) = (f \wedge \text{bs } (\neg f))$ **using** 1 **by** auto
from 2 4 5 6 **show** ?thesis **by** (metis first-d-def int-eq)
qed

lemma FstOrDiEqvDi:

$\vdash (\triangleright f \vee \text{di } f) = \text{di } f$

proof —

have 1: $\vdash (\triangleright f \vee \text{di } f) = ((f \wedge \text{bs } (\neg f)) \vee \text{di } f)$ **by** (simp add: first-d-def)
have 2: $\vdash ((f \wedge \text{bs } (\neg f)) \vee \text{di } f) = ((f \vee \text{di } f) \wedge (\text{bs } (\neg f) \vee \text{di } f))$ **by** auto
have 3: $\vdash (f \vee \text{di } f) = \text{di } f$
by (metis 2 DiIntro RRDiamondEqvDi int-eq Prop02 Prop03 Prop11 Prop12)
hence 4: $\vdash ((f \vee \text{di } f) \wedge (\text{bs } (\neg f) \vee \text{di } f)) = (\text{di } f \wedge (\text{bs } (\neg f) \vee \text{di } f))$ **by** auto
have 5: $\vdash (\text{di } f \wedge (\text{bs } (\neg f) \vee \text{di } f)) = \text{di } f$ **by** auto
from 1 2 4 5 **show** ?thesis **by** fastforce
qed

lemma FstAndDiEqvFst:

$\vdash (\triangleright f \wedge \text{di } f) = \triangleright f$

proof —

have 1: $\vdash (\triangleright f \wedge \text{di } f) = ((f \wedge \text{bs } (\neg f)) \wedge \text{di } f)$ **by** (simp add: first-d-def)
have 2: $\vdash (f \wedge \text{di } f) = f$ **by** (meson DiIntro Prop10 Prop11)
hence 3: $\vdash (f \wedge \text{bs } (\neg f) \wedge \text{di } f) = (f \wedge \text{bs } (\neg f))$ **by** auto
from 1 3 **show** ?thesis **by** (metis first-d-def int-iffD2 int-iffI Prop12)
qed

lemma DiEqvDiFst:

$\vdash \text{di } f = \text{di } (\triangleright f)$

proof —

have 1: $\vdash \text{di } (\triangleright f) = \text{di } (f \wedge \text{bs } (\neg f))$
by (simp add: first-d-def)

have 2: $\vdash di (f \wedge bs (\neg f)) \longrightarrow di f \wedge di (bs (\neg f))$
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 *DilIntro* **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: ChopSkiplmpMore prev-d-def*)
have 12: $\vdash more \longrightarrow (prev (di f \longrightarrow di (\triangleright f)) = (prev(di f) \longrightarrow prev(di (\triangleright f))))$
using *MoreImplmpPrevEqv* **by** *auto*
have 13: $\vdash (more \wedge prev (di f \longrightarrow di (\triangleright f))) = (more \wedge (prev(di f) \longrightarrow prev(di (\triangleright f))))$
using 12 **by** *fastforce*
have 14: $\vdash prev (di f \longrightarrow di (\triangleright f)) = (more \wedge (prev(di f) \longrightarrow prev(di (\triangleright f))))$
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 *DilIntro* **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: $\vdash (di\ f \wedge bs\ (\neg f) \wedge (prev\ (di\ f \longrightarrow di\ (\triangleright f)))) \longrightarrow di\ (\triangleright f)$
 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: $\vdash (di\ f \wedge \neg(bs\ (\neg f)) \wedge (prev\ (di\ f \longrightarrow di\ (\triangleright f)))) =$
 $(di\ f \wedge ds\ f \wedge (prev\ (di\ f \longrightarrow di\ (\triangleright f))))$
 by *auto*
 have 30: $\vdash ds\ f = prev(di\ f)$
 using *DsDi* by (*metis prev-d-def*)
 hence 31: $\vdash (di\ f \wedge ds\ f \wedge (prev\ (di\ f \longrightarrow di\ (\triangleright f)))) =$
 $(di\ f \wedge prev(di\ f) \wedge (prev\ (di\ f \longrightarrow di\ (\triangleright f))))$
 by *auto*
 have 32: $\vdash prev\ (di\ f \longrightarrow di\ (\triangleright f)) \longrightarrow (prev(di\ f) \longrightarrow prev(di\ (\triangleright f)))$
 using 14 by *auto*
 hence 33: $\vdash di\ f \wedge prev(di\ f) \wedge prev\ (di\ f \longrightarrow di\ (\triangleright f)) \longrightarrow$
 $di\ f \wedge prev(di\ f) \wedge (prev(di\ f) \longrightarrow prev(di\ (\triangleright f)))$
 by *auto*
 have 34: $\vdash di\ f \wedge prev(di\ f) \wedge (prev(di\ f) \longrightarrow prev(di\ (\triangleright f))) \longrightarrow prev(di\ (\triangleright f))$
 by *auto*
 have 35: $\vdash prev(di\ (\triangleright f)) = (di\ (\triangleright f)); skip$
 by (*simp add: prev-d-def*)
 have 36: $\vdash (di\ (\triangleright f)); skip \longrightarrow di(di\ (\triangleright f))$
 using *ChopImpDi* by *auto*
 have 37: $\vdash di(di\ (\triangleright f)) = di\ (\triangleright f)$
 using *DiEqvDiDi* by *fastforce*
 have 38: $\vdash di\ f \wedge prev(di\ f) \wedge (prev(di\ f) \longrightarrow prev(di\ (\triangleright f))) \longrightarrow di\ (\triangleright f)$
 using 37 36 35 34 by *fastforce*
 have 39: $\vdash di\ f \wedge \neg(bs\ (\neg f)) \wedge (prev\ (di\ f \longrightarrow di\ (\triangleright f))) \longrightarrow di\ (\triangleright f)$
 using 29 31 33 38 by *fastforce*
 hence 40: $\vdash \neg(bs\ (\neg f)) \wedge (prev\ (di\ f \longrightarrow di\ (\triangleright f))) \longrightarrow (di\ f \longrightarrow di\ (\triangleright f))$
 by *fastforce*
 have 41: $\vdash bs\ (\neg f) \wedge (prev\ (di\ f \longrightarrow di\ (\triangleright f))) \longrightarrow (di\ f \longrightarrow di\ (\triangleright f))$
 using 27 by *fastforce*
 have 42: $\vdash (\neg(bs\ (\neg f)) \vee bs\ (\neg f)) \wedge (prev\ (di\ f \longrightarrow di\ (\triangleright f))) \longrightarrow (di\ f \longrightarrow di\ (\triangleright f))$
 using 40 41 by *fastforce*
 have 43: $\vdash (\neg(bs\ (\neg f)) \vee bs\ (\neg f))$
 by *auto*
 have 44: $\vdash (prev\ (di\ f \longrightarrow di\ (\triangleright f))) \longrightarrow (di\ f \longrightarrow di\ (\triangleright f))$
 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** (*simp add: bi-d-def, auto*)

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*

have 12: $\vdash (di f \wedge di g \wedge (bs (\neg f) \vee bs (\neg g))) =$
 $((\triangleright f \wedge di g) \vee (di f \wedge \triangleright g))$ **using** 7 9 11 **by** (*metis int-eq*)

from 1 4 6 12 show ?thesis using *inteq-reflection lift-and-com* **by** *fastforce*

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: $\vdash (empty \wedge init\ w) = \triangleright (init\ w)$
 using *FstState* by *fastforce*
 hence 6: $\vdash (empty \wedge init\ w \wedge bs\ (\neg f)) = (\triangleright (init\ w) \wedge bs\ (\neg f))$
 by *auto*
 have 7: $\vdash \triangleright (init\ w) \wedge bs\ (\neg f) \longrightarrow (\triangleright (init\ w) \wedge bs\ (\neg f)) \vee (\triangleright f \wedge bs\ (\neg (init\ w)))$
 by *auto*
 have 8: $\vdash \triangleright (init\ w \vee f) = ((\triangleright (init\ w) \wedge bs\ (\neg f)) \vee (\triangleright f \wedge bs\ (\neg (init\ w))))$
 using *FstWithOrEqv* by *blast*
 from 1 2 3 4 5 6 7 8 show ?thesis by *fastforce*
 qed

lemma *FstLenSame*:

$(\forall\ \sigma. (\sigma \models di\ (\triangleright f \wedge len(i)) \wedge di\ (\triangleright f \wedge len(j))) \longrightarrow (i=j))$

by (*simp add: DiLenFstsem FstLenSamesem*)

lemma *FstLenSame-1*:

$\vdash di\ (\triangleright f \wedge len(i)) \wedge di\ (\triangleright f \wedge len(j)) \longrightarrow (\#i=\#j)$

using *FstLenSame Valid-def* by *fastforce*

lemma *FstAndLenSame*:

$(\forall\ \sigma. (\sigma \models di\ ((\triangleright f \wedge g1) \wedge len(i)) \wedge di\ ((\triangleright f \wedge g2) \wedge len(j))) \longrightarrow (i=j))$

using *linorder-neqE-nat* by (*simp add: DiLenFstAndsem, blast*)

lemma *FstAndLenSame-1*:

$\vdash di\ ((\triangleright f \wedge g1) \wedge len(i)) \wedge di\ ((\triangleright f \wedge g2) \wedge len(j)) \longrightarrow (\#i=\#j)$

using *FstAndLenSame Valid-def* by *fastforce*

lemma *FstLenSameChop*:

$(\forall \sigma. (\sigma \models ((\triangleright f \wedge g1) \wedge \text{len}(i)); h1 \wedge ((\triangleright f \wedge g2) \wedge \text{len}(j)); h2) \longrightarrow (i=j))$
proof
fix σ
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 *DilmpExistsOneDiLenAndFst*:
 $(\forall \sigma. (\sigma \models \text{di } f) \longrightarrow (\exists! k. (\sigma \models \text{di}(\triangleright f \wedge \text{len}(k)))))$
proof
fix σ
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 (∃! k. (σ ⊨ di( ▷ f ∧ len(k) )))
    using 11 5 by blast
qed
qed

```

```

lemma DilmpExistsOneDiLenAndFst-1:
  ⊢ di f ⟶ (∃! k. ( di( ▷ f ∧ len(k) )))
using Valid-def DilmpExistsOneDiLenAndFst by fastforce

```

```

lemma LFstAndDist-help:
  (σ ⊨ ((▷ f ∧ g1) ∧ len(k)); h1 ∧ ((▷ f ∧ g2) ∧ len(k)); h2) =
  (σ ⊨ (((▷ f ∧ g1) ∧ (▷ f ∧ g2)) ∧ len(k)); (h1 ∧ h2) )
using LFixedAndDistr by fastforce

```

```

lemma LFstAndDist-help-1:
  (∃ k. (σ ⊨ ((▷ f ∧ g1) ∧ len(k)); h1 ∧ ((▷ f ∧ g2) ∧ len(k)); h2)) =
  (∃ k. (σ ⊨ (((▷ f ∧ g1) ∧ (▷ f ∧ g2)) ∧ len(k)); (h1 ∧ h2) ))

```

```

proof
  assume 0: ∃ k. σ ⊨ ((▷ f ∧ g1) ∧ len k) ; h1 ∧ ((▷ f ∧ g2) ∧ len k) ; h2
  obtain k where 1: σ ⊨ ((▷ f ∧ g1) ∧ len k) ; h1 ∧ ((▷ f ∧ g2) ∧ len k) ; h2
  using 0 by auto
  hence 2: (σ ⊨ (((▷ f ∧ g1) ∧ (▷ f ∧ g2)) ∧ len(k)); (h1 ∧ h2))
  using LFstAndDist-help by blast
  show (∃ k. (σ ⊨ (((▷ f ∧ g1) ∧ (▷ f ∧ g2)) ∧ len(k)); (h1 ∧ h2) ))
  using 2 by auto
  next
  assume 3: (∃ k. (σ ⊨ (((▷ f ∧ g1) ∧ (▷ f ∧ g2)) ∧ len(k)); (h1 ∧ h2) ))
  obtain k where 4: (σ ⊨ (((▷ f ∧ g1) ∧ (▷ f ∧ g2)) ∧ len(k)); (h1 ∧ h2) )
  using 3 by auto
  hence 5: (σ ⊨ ((▷ f ∧ g1) ∧ len(k)); h1 ∧ ((▷ f ∧ g2) ∧ len(k)); h2)
  using LFstAndDist-help by blast
  show (∃ k. (σ ⊨ ((▷ f ∧ g1) ∧ len(k)); h1 ∧ ((▷ f ∧ g2) ∧ len(k)); h2))
  using 5 by auto
qed

```

```

lemma LFstAndDistrsem:
  (∀ σ. (σ ⊨ ((▷ f ∧ g1); h1 ∧ (▷ f ∧ g2); h2) = (▷ f ∧ g1 ∧ g2); (h1 ∧ h2)))

```

```

proof
  fix σ
  show (σ ⊨ ((▷ f ∧ g1); h1 ∧ (▷ f ∧ g2); h2) = (▷ f ∧ g1 ∧ g2); (h1 ∧ h2))
  proof -
    have 1: (σ ⊨ (▷ f ∧ g1); h1) = (∃ i. (σ ⊨ ((▷ f ∧ g1) ∧ len(i)); h1) )
    using AndExistsLenChop[of TEMP (▷ f ∧ g1)] by fastforce
    have 2: (σ ⊨ (▷ f ∧ g2); h2) = (∃ j. (σ ⊨ ((▷ f ∧ g2) ∧ len(j)); h2) )
    using AndExistsLenChop[of TEMP (▷ f ∧ g2)] by fastforce
    have 3: (σ ⊨ (▷ f ∧ g1); h1 ∧ (▷ f ∧ g2); h2) =
      ( (∃ i j. (σ ⊨ ((▷ f ∧ g1) ∧ len(i)); h1 ∧
        ((▷ f ∧ g2) ∧ len(j)); h2) )
      )
  qed

```

```

using 1 2 by auto
have 4: ( (∃ i j. (σ ⊨ ((▷ f ∧ g1) ∧ len(i)); h1 ∧
                ((▷ f ∧ g2) ∧ len(j)); h2) )
  ) =
  ( (∃ k. (σ ⊨ ((▷ f ∧ g1) ∧ len(k)); h1 ∧
                ((▷ f ∧ g2) ∧ len(k)); h2) )
  )
using FstLenSameChop by blast
have 5: (∃ k. (σ ⊨ ((▷ f ∧ g1) ∧ len(k)); h1 ∧ ((▷ f ∧ g2) ∧ len(k)); h2)) =
  (∃ k. (σ ⊨ (((▷ f ∧ g1) ∧ (▷ f ∧ g2)) ∧ len(k)); (h1 ∧ h2) ))
using LFstAndDist-help-1 by blast
have 6 : (∃ k. (σ ⊨ (((▷ f ∧ g1) ∧ (▷ f ∧ g2)) ∧ len(k)); (h1 ∧ h2) )) =
  (σ ⊨ ((▷ f ∧ g1) ∧ (▷ f ∧ g2)); (h1 ∧ h2))
using AndExistsLenChop[of TEMP ((▷ f ∧ g1) ∧ ▷ f ∧ g2)] by fastforce
have 7 : (σ ⊨ ((▷ f ∧ g1) ∧ (▷ f ∧ g2)); (h1 ∧ h2)) =
  (σ ⊨ (▷ f ∧ g1 ∧ g2); (h1 ∧ h2))
by (simp add: chop-defs, auto)
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 –

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*
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*
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*
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*
have 5: $\vdash (h1^r)^r = h1$
using *EqvReverseReverse* **by** *blast*
have 6: $\vdash (h2^r)^r = h2$
using *EqvReverseReverse* **by** *blast*
have 7: $\vdash (g1^r)^r = g1$
using *EqvReverseReverse* **by** *blast*
have 8: $\vdash (g2^r)^r = g2$
using *EqvReverseReverse* **by** *blast*
have 9: $\vdash (f^r)^r = f$
using *EqvReverseReverse* **by** *blast*
have 10: $\vdash (\triangleright(f^r) \wedge g1^r)^r = ((\triangleright(f^r))^r \wedge (g1^r)^r)$
using *RAnd* **by** *blast*
have 11: $\vdash (\triangleright(f^r) \wedge g2^r)^r = ((\triangleright(f^r))^r \wedge (g2^r)^r)$
using *RAnd* **by** *blast*

have 12: $\vdash (\triangleright(f'))^r = \triangleleft(f)$
using *RRFirstEqvLast* **by** *blast*
have 13: $\vdash ((\triangleright(f'))^r \wedge (g1')^r) = (\triangleleft f \wedge g1)$
using 12 7 **by** *fastforce*
have 14: $\vdash ((\triangleright(f'))^r \wedge (g2')^r) = (\triangleleft f \wedge g2)$
using 12 8 **by** *fastforce*
have 15: $\vdash (h1;(\triangleleft f \wedge g1) \wedge h2;(\triangleleft f \wedge g2)) =$
 $((h1')^r;(\triangleright(f') \wedge g1')^r \wedge (h2')^r;(\triangleright(f') \wedge (g2')^r))^r$
using 14 13 10 11 5 6 **by** (*metis 4 int-eq*)
have 16: $\vdash (((\triangleright(f') \wedge (g1') \wedge (g2'))^r);((h1') \wedge (h2'))^r) =$
 $((h1')^r \wedge (h2')^r);((\triangleright(f') \wedge (g1') \wedge (g2'))^r$
by (*simp add: RevChop*)
have 17: $\vdash ((\triangleright(f')) \wedge (g1')^r \wedge (g2')^r)^r = ((\triangleright(f'))^r \wedge (g1')^r \wedge (g2')^r)$
by (*metis inteq-reflection rev-fun2*)
have 18: $\vdash ((\triangleright(f'))^r \wedge (g1')^r \wedge (g2')^r) = (\triangleleft f \wedge g1 \wedge g2)$
using 12 7 8 **by** *fastforce*
have 19: $\vdash ((h1') \wedge (h2'))^r = (h1 \wedge h2)$
using *RRAnd* **by** *auto*
have 20: $\vdash ((h1') \wedge (h2'))^r;((\triangleright(f') \wedge (g1') \wedge (g2'))^r =$
 $(h1 \wedge h2);(\triangleleft f \wedge g1 \wedge g2)$
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: $\vdash (\text{empty} \vee (\neg(\triangleright f;g))); \text{skip} = (\text{empty} \vee (\neg((\triangleright f;g);\# \text{True}))); \text{skip}$
 by (simp add: di-d-def)
 have 4: $\vdash (\neg((\triangleright f;g);\# \text{True})); \text{skip} = (\neg(\triangleright f;(g;\# \text{True}))); \text{skip}$
 by (metis ChopAssocB LeftChopEqvChop int-simps(15) inteq-reflection)
 hence 5: $\vdash (\text{empty} \vee (\neg((\triangleright f;g);\# \text{True}))); \text{skip} = (\text{empty} \vee (\neg(\triangleright f;(g;\# \text{True}))); \text{skip}$
 by auto
 have 6: $\vdash (\text{empty} \vee (\neg(\triangleright f;(g;\# \text{True}))); \text{skip} = (\text{empty} \vee (\neg(\triangleright f;di(g)))); \text{skip}$
 by (simp add: di-d-def)
 have 7: $\vdash (\text{empty} \vee (\neg(\triangleright f;di(g)))); \text{skip} = (\text{empty} \vee \neg(\neg((\neg(\triangleright f;di(g)))); \text{skip})))$
 by auto
 have 8: $\vdash \neg(\neg(\neg(\triangleright f;di(g)))); \text{skip} = (\neg(\text{empty} \vee (\triangleright f;di(g)); \text{skip}))$
 using NotNotChopSkip by fastforce
 hence 9: $\vdash (\text{empty} \vee \neg(\neg(\neg(\triangleright f;di(g)))); \text{skip}))) = (\text{empty} \vee \neg(\text{empty} \vee (\triangleright f;di(g)); \text{skip}))$
 by auto
 have 10: $\vdash (\text{empty} \vee \neg(\text{empty} \vee (\triangleright f;di(g)); \text{skip})) = (\text{empty} \vee (\text{more} \wedge \neg((\triangleright f;di(g)); \text{skip})))$
 by (meson 6 7 9 NotChopSkipEqvMoreAndNotChopSkip Prop04 Prop06)
 have 11: $\vdash (\text{empty} \vee (\text{more} \wedge \neg((\triangleright f;di(g)); \text{skip}))) = (\text{empty} \vee \neg((\triangleright f;di(g)); \text{skip}))$
 by (simp add: empty-d-def, auto)
 have 12: $\vdash (\text{empty} \vee \neg((\triangleright f;di(g)); \text{skip})) = (\text{empty} \vee \neg(\triangleright f;(di(g); \text{skip})))$
 using ChopAssocB 11 by fastforce
 have 13: $\vdash (\neg(\triangleright f;(di(g); \text{skip}))) = (\neg(\triangleright f;(ds(g))))$
 using DsDi using RightChopEqvChop by fastforce
 hence 14: $\vdash (\text{empty} \vee \neg(\triangleright f;(di(g); \text{skip}))) = (\text{empty} \vee \neg(\triangleright f;(ds(g))))$
 by auto
 have 15: $\vdash (\text{empty} \vee \neg(\triangleright f;(ds(g)))) = (\text{empty} \vee \neg(di(\triangleright f)) \vee (\triangleright f;(\neg(ds\ g))))$
 using NotFstChop by fastforce
 have 16: $\vdash (\triangleright f;(\neg(ds\ g))) = (\triangleright f;(bs(\neg g)))$
 using NotDsEqvBsNot RightChopEqvChop by blast
 hence 17: $\vdash ((\text{empty} \vee \neg(di(\triangleright f))) \vee (\triangleright f;(\neg(ds\ g)))) = ((\text{empty} \vee \neg(di(\triangleright f))) \vee (\triangleright f;(bs(\neg g))))$
 by auto
 from 1 2 3 5 6 7 9 10 11 12 14 15 17 show ?thesis by fastforce
 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: $\vdash bs(\neg(\triangleright f;g)) = (\text{empty} \vee \neg(di(\triangleright f)) \vee (\triangleright f;bs(\neg g)))$

using BsNotFstChop by auto

hence 3: $\vdash ((\triangleright f;g) \wedge bs(\neg(\triangleright f;g))) = ((\triangleright f;g) \wedge (\text{empty} \vee \neg(di(\triangleright f)) \vee (\triangleright f;bs(\neg g))))$

by auto

have 4: $\vdash ((\triangleright f;g) \wedge (\text{empty} \vee \neg(di(\triangleright f)) \vee (\triangleright f;bs(\neg g)))) =$
 $((\triangleright f;g) \wedge \text{empty}) \vee ((\triangleright f;g) \wedge \neg(di(\triangleright f))) \vee ((\triangleright f;g) \wedge (\triangleright f;bs(\neg g)))$

by auto

have 5: $\vdash \neg((\triangleright f;g) \wedge \neg(di(\triangleright f)))$

using ChopImpDi by fastforce

hence 6: $\vdash (((\triangleright f;g) \wedge \text{empty}) \vee ((\triangleright f;g) \wedge \neg(di(\triangleright f))) \vee ((\triangleright f;g) \wedge (\triangleright f;bs(\neg g)))) =$
 $((\triangleright f;g) \wedge \text{empty}) \vee ((\triangleright f;g) \wedge (\triangleright f;bs(\neg g)))$

```

  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:  $\vdash \triangleright f = (\triangleright f);empty$  using ChopEmpty by (metis int-eq)
  hence 2:  $\vdash \triangleright(\triangleright f) = \triangleright((\triangleright f);empty)$  using FstEqvRule by blast
  have 3:  $\vdash \triangleright((\triangleright f);empty) = \triangleright f;\triangleright empty$  using FstFstChopEqvFstChopFst by auto
  have 4:  $\vdash \triangleright f;\triangleright empty = \triangleright f;empty$  using FstEmpty using RightChopEqvChop by blast
  have 5:  $\vdash \triangleright f;empty = \triangleright f$  using ChopEmpty by blast
  from 2 3 4 5 show ?thesis by fastforce
qed

```

lemma FstCSEqvEmpty:

```

 $\vdash \triangleright(f^*) = empty$ 
proof -
  have 1:  $\vdash \triangleright(f^*) = \triangleright(empty \vee ((f \wedge more);f^*))$  using ChopstarEqv FstEqvRule by blast
  from 1 show ?thesis using FstEmptyOrEqvEmpty by fastforce
qed

```

lemma FstIterFixFst:

```

 $\vdash power(\triangleright f) n = \triangleright(power(\triangleright f) n)$ 
proof
  (induct n)
  case 0
  then show ?case
  proof -
    have 1:  $\vdash power(\triangleright f) 0 = empty$  by auto
    have 2:  $\vdash empty = \triangleright empty$  using FstEmpty by auto
    have 3:  $\vdash \triangleright empty = \triangleright(power(\triangleright f) 0)$  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: $\vdash \text{len}(k) \wedge f \wedge \text{ds}(\text{len}(k) \wedge f) \longrightarrow (\text{len}(k); (\text{skip}; \# \text{True}))$
using *SkipTrueEqvTrueSkip TrueChopSkipEqvSkipChopTrue RightChopEqvChop* **by** *fastforce*
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 (*simp add: first-d-def, auto*)
from 13 14 **show** ?thesis **by** fastforce
qed

lemma *FstAndElimL*:
 $\vdash \triangleright f \longrightarrow f$
by (*simp add: first-d-def, auto*)

lemma *FstImpNotDiChopSkip*:
 $\vdash \triangleright f \longrightarrow \neg(\text{di } f ; \text{skip})$
proof –
have 1: $\vdash \triangleright f \longrightarrow \text{bs } (\neg f)$ **by** (*simp add: first-d-def, auto*)
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*:
 $\vdash \triangleright f \longrightarrow \neg(\text{di } (f ; \text{skip}))$
proof –
have 1: $\vdash \triangleright f \longrightarrow \text{bs } (\neg f)$
by (*simp add: first-d-def, auto*)
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
have 4: $\vdash \text{di } f ; \text{skip} = (f ; \# \text{True}) ; \text{skip}$
by (*simp add: di-d-def*)
have 5: $\vdash (f ; \# \text{True}) ; \text{skip} = f ; (\# \text{True} ; \text{skip})$
using *ChopAssocB* **by** blast
have 6: $\vdash f ; (\# \text{True} ; \text{skip}) = f ; (\text{skip} ; \# \text{True})$
using *SkipTrueEqvTrueSkip* **using** *TrueChopSkipEqvSkipChopTrue* *RightChopEqvChop* **by** blast
have 7: $\vdash f ; (\text{skip} ; \# \text{True}) = (f ; \text{skip}) ; \# \text{True}$
using *ChopAssoc* **by** blast
have 8: $\vdash (f ; \text{skip}) ; \# \text{True} = \text{di}(f ; \text{skip})$
by (*simp add: di-d-def*)
have 9: $\vdash (\neg(\text{ds } f)) = (\neg(\text{di}(f ; \text{skip})))$
using 3 4 5 6 7 8 **by** fastforce
from 2 9 **show** ?thesis **by** fastforce
qed

lemma *FstImpDiEqv*:
 $\vdash \triangleright f \longrightarrow (\text{di } f = f)$
proof –
have 1: $\vdash \triangleright f \longrightarrow \neg(\text{di } f ; \text{skip})$ **using** *FstImpNotDiChopSkip* **by** blast
have 2: $\vdash \text{di } f \longrightarrow f \vee (\text{di } f ; \text{skip})$ **using** *DiEqvOrDiChopSkipB* **by** fastforce
have 3: $\vdash \triangleright f \wedge \text{di } f \longrightarrow (f \vee (\text{di } f ; \text{skip})) \wedge \neg(\text{di } f ; \text{skip})$ **using** 1 2 **by** fastforce
have 4: $\vdash ((f \vee (\text{di } f ; \text{skip})) \wedge \neg(\text{di } f ; \text{skip})) = (f \wedge \neg(\text{di } f ; \text{skip}))$ **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** *DilIntro* **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 *DilIntro* **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 *FstAndDilImpBsNotAndDi*:

$\vdash (\triangleright f \wedge di\ g) \longrightarrow (bs\ (\neg(di\ f \wedge g)))$

proof –

have 1: $\vdash (\triangleright f \wedge di\ g) \wedge \neg(bs\ (\neg(di\ f \wedge g))) \longrightarrow ds(di\ f \wedge g)$

by (simp add: ds-d-def, auto)
 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: $\vdash \Box(\text{empty} = (\text{init } w)) = (\Box((\text{empty} \longrightarrow (\text{init } w)) \wedge ((\text{init } w) \longrightarrow \text{empty})))$
by (*simp add: BoxEqvBox*)
have 3: $\vdash (\Box((\text{empty} \longrightarrow (\text{init } w)) \wedge ((\text{init } w) \longrightarrow \text{empty}))) =$
 $(\Box((\text{empty} \longrightarrow (\text{init } w)))) \wedge \Box((\text{init } w) \longrightarrow \text{empty}))$
by (*metis 21 BoxAndBoxEqvBoxRule int-eq*)
have 4: $\vdash ((\text{init } w) \longrightarrow \text{empty}) = (\text{more} \longrightarrow \neg(\text{init } w))$
by (*simp add: empty-d-def, auto*)
hence 5: $\vdash \Box((\text{init } w) \longrightarrow \text{empty}) = \Box(\text{more} \longrightarrow \neg(\text{init } w))$ **using** *BoxEqvBox* **by** *blast*
have 6: $\vdash \Box(\text{more} \longrightarrow \neg(\text{init } w)) = \text{bs}(\neg(\text{fin}(\text{init } w)))$ **using** *BoxMoreStateEqvBsFinState* **by** *blast*
have 7: $\vdash \Box((\text{empty} \longrightarrow (\text{init } w))) = \text{fin}(\text{init } w)$ **by** (*simp add: fin-d-def*)
have 8: $\vdash (\Box((\text{empty} \longrightarrow (\text{init } w)))) \wedge \Box((\text{init } w) \longrightarrow \text{empty}) =$
 $(\text{fin}(\text{init } w) \wedge \text{bs}(\neg(\text{fin}(\text{init } w))))$ **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*:

$\vdash \triangleright (\Box (init\ w)) = ((init\ w) \wedge empty)$

proof –

have 1: $\vdash ((init\ w) \wedge (\Box (init\ w))^*) = \Box (init\ w)$

using *BoxCSEqvBox* **by** *blast*

have 2: $\vdash \Box (init\ w) = ((init\ w) \wedge (\Box (init\ w))^*)$

using 1 **by** *auto*

hence 3: $\vdash \Box (init\ w) = ((init\ w) \wedge (\Box (init\ w))^*)$

by *blast*

have 4: $\vdash ((init\ w) \wedge empty) ; (\Box (init\ w))^* = ((init\ w) \wedge (\Box (init\ w))^*)$

using *StateAndEmptyChop* **by** *blast*

have 5: $\vdash ((init\ w) \wedge (\Box (init\ w))^*) = ((init\ w) \wedge empty) ; (\Box (init\ w))^*$

using 4 **by** *fastforce*

have 6: $\vdash \Box (init\ w) = ((init\ w) \wedge empty) ; (\Box (init\ w))^*$

using 3 5 **by** *fastforce*

have 7: $\vdash ((init\ w) \wedge empty) ; (\Box (init\ w))^* = \triangleright (init\ w) ; (\Box (init\ w))^*$

using *FstState* **by** (*metis AndChopCommute int-eq*)

have 8: $\vdash \Box (init\ w) = \triangleright (init\ w) ; (\Box (init\ w))^*$

using 6 7 **by** *fastforce*

have 9: $\vdash \triangleright (\Box (init\ w)) = \triangleright (\triangleright (init\ w) ; (\Box (init\ w))^*)$

using 8 *FstEqvRule* **by** *blast*

have 10: $\vdash \triangleright (\triangleright (init\ w) ; (\Box (init\ w))^*) = \triangleright (init\ w) ; \triangleright ((\Box (init\ w))^*)$

using *FstFstChopEqvFstChopFst* **by** *blast*

have 11: $\vdash \triangleright (init\ w) ; \triangleright ((\Box (init\ w))^*) = \triangleright (init\ w) ; empty$

using *RightChopEqvChop FstCSEqvEmpty* **by** *blast*

have 12: $\vdash \triangleright (init\ w) ; empty = \triangleright (init\ w)$

using *RightChopEqvChop ChopEmpty* **by** *blast*

have 13: $\vdash \triangleright (init\ w) = ((init\ w) \wedge empty)$

using *FstState* **by** *fastforce*

from 9 10 11 12 13 **show** *?thesis* **by** *fastforce*

qed

lemma *FstAndFstStarEqvFst*:

$\vdash (\triangleright f \wedge (\triangleright f)^*) = \triangleright f$

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:  $\vdash (\triangleright f);((\triangleright f)^* \wedge \text{empty}) = \triangleright f$ 
    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:  $\vdash ((\triangleright f)^* \wedge \triangleright f) = ((\text{empty} \wedge \triangleright f) \vee \triangleright f)$ 
    using 4 9 by fastforce
  have 11:  $\vdash ((\text{empty} \wedge \triangleright f) \vee \triangleright f) = \triangleright f$ 
    by auto
  have 12:  $\vdash ((\triangleright f)^* \wedge \triangleright f) = \triangleright f$ 
    using 10 11 by fastforce
  from 12 show ?thesis by auto
qed

```

lemma *HaltStateEqvFstHaltState*:

$\vdash \text{halt}(\text{init}(w)) = \triangleright(\text{halt}(\text{init}(w)))$

proof –

```

  have 1:  $\vdash \text{halt}(\text{init } w) = \triangleright(\text{fin}(\text{init } w))$ 
    by (simp add: HaltStateEqvFstFinState)
  have 2:  $\vdash \triangleright(\text{fin}(\text{init } w)) = \triangleright(\triangleright(\text{fin}(\text{init } w)))$ 
    using FstEqvRule FstFixFst by fastforce
  have 3:  $\vdash \triangleright(\triangleright(\text{fin}(\text{init } w))) = \triangleright(\text{halt}(\text{init}(w)))$ 
    using FstEqvRule HaltStateEqvFstFinState by fastforce
  from 1 2 3 show ?thesis by fastforce

```

qed

lemma *DiHaltAndDiHaltAndEqvDiHaltAndAnd*:

$\vdash (\text{di}(\text{halt}(\text{init } w) \wedge f) \wedge \text{di}(\text{halt}(\text{init } w) \wedge g)) = \text{di}(\text{halt}(\text{init } w) \wedge f \wedge g)$

proof –

```

  have 1:  $\vdash (\text{di}(\text{halt}(\text{init } w) \wedge f) \wedge \text{di}(\text{halt}(\text{init } w) \wedge g)) =$ 
     $(\text{di}(\triangleright(\text{fin}(\text{init } w)) \wedge f) \wedge \text{di}(\triangleright(\text{fin}(\text{init } w)) \wedge g))$ 
    using HaltStateEqvFstFinState by (metis LFstAndDistrD inteq-reflection)
  have 2:  $\vdash (\text{di}(\triangleright(\text{fin}(\text{init } w)) \wedge f) \wedge \text{di}(\triangleright(\text{fin}(\text{init } w)) \wedge g)) =$ 
     $\text{di}(\triangleright(\text{fin}(\text{init } w)) \wedge f \wedge g)$ 
    using LFstAndDistrD by fastforce
  have 3:  $\vdash \text{di}(\triangleright(\text{fin}(\text{init } w)) \wedge f \wedge g) = \text{di}(\text{halt}(\text{init } w) \wedge f \wedge g)$ 
    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*:

$\vdash ((\triangleright(\text{len}(5)) \wedge \triangleright(\text{len}(2))) ; (\text{len}(5) \vee \text{len}(2))) = \#False$

proof –

```

  have 1:  $\vdash ((\triangleright(\text{len}(5)) \wedge \triangleright(\text{len}(2))) ; (\text{len}(5) \vee \text{len}(2))) =$ 
     $(\text{len}(5) \wedge \text{len}(2)); (\text{len}(5) \vee \text{len}(2)))$ 
    by (metis FstLenAndEqvLenAnd FstLenEqvLen LeftChopEqvChop inteq-reflection)
  have 2:  $\vdash (\text{len}(5) \wedge \text{len}(2)) = \#False$ 

```

```

    by (simp add: Valid-def len-defs)
have 3:  $\vdash ((\text{len}(5) \wedge \text{len}(2)); (\text{len}(5) \vee \text{len}(2))) = (\#False; (\text{len}(5) \vee \text{len}(2)))$ 
    by (simp add: 2 LeftChopEqvChop)
have 4:  $\vdash (\#False; (\text{len}(5) \vee \text{len}(2))) = \#False$ 
    by (simp add: Valid-def chop-defs)
from 1 3 4 show ?thesis by fastforce
qed

```

lemma counter-ex-rhs:

```

 $\vdash ((\triangleright (\text{len}(5)); (\text{len}(5) \vee \text{len}(2))) \wedge (\triangleright (\text{len}(2)); (\text{len}(5) \vee \text{len}(2)))) = \text{len}(7)$ 
proof –
have 1:  $\vdash (\triangleright (\text{len}(5)); (\text{len}(5) \vee \text{len}(2))) =$ 
     $\text{len}(5); (\text{len}(5) \vee \text{len}(2))$ 
    using FstLenEqvLen LeftChopEqvChop by blast
have 2:  $\vdash (\triangleright (\text{len}(2)); (\text{len}(5) \vee \text{len}(2))) =$ 
     $\text{len}(2); (\text{len}(5) \vee \text{len}(2))$ 
    using FstLenEqvLen LeftChopEqvChop by blast
have 3:  $\vdash \text{len}(5); (\text{len}(5) \vee \text{len}(2)) =$ 
     $((\text{len}(5); \text{len}(5)) \vee (\text{len}(5); \text{len}(2)))$ 
    by (simp add: ChopOrEqv)
have 4:  $\vdash ((\text{len}(5); \text{len}(5)) \vee (\text{len}(5); \text{len}(2))) =$ 
     $(\text{len}(10) \vee \text{len}(7))$ 
    using LenEqvLenChopLen inteq-reflection by fastforce
have 5:  $\vdash \text{len}(2); (\text{len}(5) \vee \text{len}(2)) =$ 
     $((\text{len}(2); \text{len}(5)) \vee (\text{len}(2); \text{len}(2)))$ 
    by (simp add: ChopOrEqv)
have 6:  $\vdash ((\text{len}(2); \text{len}(5)) \vee (\text{len}(2); \text{len}(2))) =$ 
     $(\text{len}(7) \vee \text{len}(4))$ 
    using LenEqvLenChopLen inteq-reflection by fastforce
have 7:  $\vdash ((\text{len}(10) \vee \text{len}(7)) \wedge (\text{len}(7) \vee \text{len}(4))) =$ 
     $((\text{len}(7) \vee \text{len}(10)) \wedge (\text{len}(7) \vee \text{len}(4)))$ 
    by fastforce
have 8:  $\vdash ((\text{len}(7) \vee \text{len}(10)) \wedge (\text{len}(7) \vee \text{len}(4))) =$ 
     $(\text{len}(7) \vee (\text{len}(10) \wedge \text{len}(4)))$ 
    by fastforce
have 9:  $\vdash (\text{len}(10) \wedge \text{len}(4)) = \#False$ 
    by (simp add: Valid-def len-defs)
have 10:  $\vdash (\text{len}(7) \vee (\text{len}(10) \wedge \text{len}(4))) = \text{len}(7)$ 
    using 9 by auto
have 11:  $\vdash ((\triangleright (\text{len}(5)); (\text{len}(5) \vee \text{len}(2))) \wedge (\triangleright (\text{len}(2)); (\text{len}(5) \vee \text{len}(2)))) =$ 
     $(\text{len}(5); (\text{len}(5) \vee \text{len}(2)) \wedge \text{len}(2); (\text{len}(5) \vee \text{len}(2)))$ 
    using 1 2 by fastforce
have 12:  $\vdash (\text{len}(5); (\text{len}(5) \vee \text{len}(2)) \wedge \text{len}(2); (\text{len}(5) \vee \text{len}(2))) = \text{len}(7)$ 
    using 10 3 4 5 6
    by fastforce
from 11 12 show ?thesis by fastforce
qed

```

end

15 Monitors

theory *Monitor*

imports *First*

begin

The RV monitors language is introduced plus the algebraic properties of the monitor operators.

15.1 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 $(a \text{ WITH } f) \simeq (b \text{ WITH } f)$
using *assms* **by** (*metis MON.simps(5) eq-d-def inteq-reflection lift-and-com*)

lemma *MonEqSubstThen*:
assumes $a1 \simeq b1$
 $a2 \simeq b2$
shows $(a1 \text{ THEN } a2) \simeq (b1 \text{ THEN } b2)$
using *assms(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) assms(2)* **by** (*metis (mono-tags, lifting) MON.simps(2) eq-d-def int-eq MonEqRefl*)

lemma *MonEqSubstThru*:
assumes $a1 \simeq b1$
 $a2 \simeq b2$
shows $(a1 \text{ THRU } a2) \simeq (b1 \text{ THRU } b2)$
using *assms(1) assms(2)* **by** (*metis (mono-tags, lifting) MON.simps(3) eq-d-def int-eq MonEqRefl*)

15.2 Derived Monitors

definition *HALT-d* :: $('a :: \text{world}) \text{ formula} \Rightarrow 'a \text{ monitor}$
where $\text{HALT-d } w \equiv \text{FIRST}(\text{LIFT}(\text{fin}(\text{init } w)))$

definition *LEN-d* :: $\text{nat} \Rightarrow ('a :: \text{world}) \text{ monitor}$
where
 $\text{LEN-d } k \equiv \text{FIRST}(\text{LIFT}(\text{len } k))$

definition *EMPTY-d* :: $('a :: \text{world}) \text{ monitor}$
where
 $\text{EMPTY-d} \equiv \text{FIRST}(\text{LIFT}(\text{empty}))$

definition *SKIP-d* :: $('a :: \text{world}) \text{ monitor}$
where
 $\text{SKIP-d} \equiv \text{FIRST}(\text{LIFT}(\text{skip}))$

syntax
 $\text{-HALT-d} :: \text{lift} \Rightarrow 'a \text{ monitor} \quad ((\text{HALT -}) [84] 83)$
 $\text{-LEN-d} :: \text{nat} \Rightarrow 'a \text{ monitor} \quad ((\text{LEN -}) [84] 83)$

-EMPTY-d :: 'a monitor ((EMPTY))
 -SKIP-d :: 'a monitor ((SKIP))

syntax (ASCII)

-HALT-d :: lift \Rightarrow 'a monitor ((HALT -) [84] 83)
 -LEN-d :: nat \Rightarrow 'a monitor ((LEN -) [84] 83)
 -EMPTY-d :: 'a monitor ((EMPTY))
 -SKIP-d :: 'a monitor ((SKIP))

translations

-HALT-d \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) formula \Rightarrow 'a monitor

where

GUARD-d w \equiv (EMPTY WITH LIFT(init w))

primrec TIMES-d :: ('a :: world) monitor \Rightarrow nat \Rightarrow 'a monitor

where

TIMES-0 : TIMES-d a 0 = EMPTY

| TIMES-Suc: TIMES-d a (Suc k) = (a THEN (TIMES-d a k))

syntax

-GUARD-d :: lift \Rightarrow 'a monitor ((GUARD -) [84] 83)
 -TIMES-d :: ['a monitor, nat] \Rightarrow 'a monitor ((- TIMES -) [84,84] 83)

syntax (ASCII)

-GUARD-d :: lift \Rightarrow 'a monitor ((GUARD -) [84] 83)
 -TIMES-d :: ['a monitor, nat] \Rightarrow 'a monitor ((- TIMES -) [84,84] 83)

translations

-GUARD-d \Rightarrow CONST GUARD-d
 -TIMES-d \Rightarrow 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 (fin (init w)))))

definition SOMETIME-d :: ('a :: world) monitor \Rightarrow 'a formula \Rightarrow 'a monitor

where

SOMETIME-d a w \equiv (a WITH LIFT((di (fin (init w)))))

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

where

LIMIT-d f \equiv *LIFT*(bs (\neg f))

definition *UNTIL-d* :: ('a :: world) formula \Rightarrow 'a formula \Rightarrow 'a monitor

where

UNTIL-d w1 w2 \equiv (*HALT* w2) *WITH* (*LIFT*(bm w1))

syntax

-*FAIL-d* :: 'a monitor (FAIL)
-*ALWAYS-d* :: ['a monitor, lift] \Rightarrow 'a monitor ((- *ALWAYS* -) [84,84] 83)
-*SOMETIME-d* :: ['a monitor, lift] \Rightarrow 'a monitor ((- *SOMETIME* -) [84,84] 83)
-*LIMIT-d* :: lift \Rightarrow lift ((Limit -) [84] 83)
-*UNTIL-d* :: [lift, lift] \Rightarrow 'a monitor ((- *UNTIL* -) [84,84] 83)

syntax (ASCII)

-*FAIL-d* :: 'a monitor (FAIL)
-*ALWAYS-d* :: ['a monitor, lift] \Rightarrow 'a monitor ((- *ALWAYS* -) [84,84] 83)
-*SOMETIME-d* :: ['a monitor, lift] \Rightarrow 'a monitor ((- *SOMETIME* -) [84,84] 83)
-*LIMIT-d* :: lift \Rightarrow lift ((Limit -) [84] 83)
-*UNTIL-d* :: [lift, lift] \Rightarrow 'a monitor ((- *UNTIL* -) [84,84] 83)

translations

-*FAIL-d* \Rightarrow *CONST FAIL-d*
-*ALWAYS-d* \Rightarrow *CONST ALWAYS-d*
-*SOMETIME-d* \Rightarrow *CONST SOMETIME-d*
-*LIMIT-d* \Rightarrow *CONST LIMIT-d*
-*UNTIL-d* \Rightarrow *CONST UNTIL-d*

definition *WITHIN-d* :: ('a :: world) monitor \Rightarrow 'a formula \Rightarrow 'a monitor

where

WITHIN-d a f \equiv (a *WITH* *LIFT*(Limit f))

syntax

-*WITHIN-d* :: ['a monitor, lift] \Rightarrow 'a monitor ((- *WITHIN* -) [84,84] 83)

syntax (ASCII)

-*WITHIN-d* :: ['a monitor, lift] \Rightarrow 'a monitor ((- *WITHIN* -) [84,84] 83)

translations

-*WITHIN-d* \Rightarrow *CONST WITHIN-d*

definition *AND-d* :: ('a :: world) monitor \Rightarrow 'a monitor \Rightarrow 'a monitor

where

AND-d a b \equiv (a *WITH* *LIFT*(\mathcal{M} b))

definition *ITERATE-d* :: ('a :: world) monitor \Rightarrow 'a monitor \Rightarrow 'a monitor

where

$ITERATE\text{-}d\ a\ b \equiv (a\ \text{WITH}\ (LIFT\ (\mathcal{M}\ b)^*))$

syntax

$\text{-}AND\text{-}d \quad :: ['a\ monitor, 'a\ monitor] \Rightarrow 'a\ monitor\ ((- AND -) [84,84]\ 83)$

$\text{-}ITERATE\text{-}d \quad :: ['a\ monitor, 'a\ monitor] \Rightarrow 'a\ monitor\ ((- ITERATE -) [84,84]\ 83)$

syntax (ASCII)

$\text{-}AND\text{-}d \quad :: ['a\ monitor, 'a\ monitor] \Rightarrow 'a\ monitor\ ((- AND -) [84,84]\ 83)$

$\text{-}ITERATE\text{-}d \quad :: ['a\ monitor, 'a\ monitor] \Rightarrow 'a\ monitor\ ((- ITERATE -) [84,84]\ 83)$

translations

$\text{-}AND\text{-}d \quad \Rightarrow CONST\ AND\text{-}d$

$\text{-}ITERATE\text{-}d \Rightarrow CONST\ ITERATE\text{-}d$

definition $STAR\text{-}d \quad :: ('a \quad :: world)\ monitor \Rightarrow 'a\ formula \Rightarrow 'a\ monitor$

where

$STAR\text{-}d\ a\ f \equiv ((FIRST\ LIFT(\diamond f))\ ITERATE\ (a))$

definition $REPEAT\text{-}d \quad :: ('a \quad :: world)\ monitor \Rightarrow 'a\ formula \Rightarrow 'a\ monitor$

where

$REPEAT\text{-}d\ a\ w \equiv ((HALT\ w)\ ITERATE\ (a\ \text{WITH}\ LIFT(keep(\neg (init\ w)))))$

syntax

$\text{-}STAR\text{-}d \quad :: ['a\ monitor, lift] \Rightarrow 'a\ monitor\ ((- STAR -) [84,84]\ 83)$

$\text{-}REPEAT\text{-}d \quad :: ['a\ monitor, lift] \Rightarrow 'a\ monitor\ ((- REPEATUNTIL -) [84,84]\ 83)$

syntax (ASCII)

$\text{-}STAR\text{-}d \quad :: ['a\ monitor, lift] \Rightarrow 'a\ monitor\ ((- STAR -) [84,84]\ 83)$

$\text{-}REPEAT\text{-}d \quad :: ['a\ monitor, lift] \Rightarrow 'a\ monitor\ ((- REPEATUNTIL -) [84,84]\ 83)$

translations

$\text{-}STAR\text{-}d \quad \Rightarrow CONST\ STAR\text{-}d$

$\text{-}REPEAT\text{-}d \Rightarrow CONST\ REPEAT\text{-}d$

15.3 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-d a1 a2)
then show ?case
proof -
have 1:  $\vdash (\mathcal{M} (a1 \text{ UPTO } a2)) = \triangleright (\mathcal{M} a1) \vee (\mathcal{M} a2)$ 
  by (simp)
have 2:  $\vdash \triangleright (\mathcal{M} a1) \vee (\mathcal{M} a2) = \triangleright (\triangleright (\mathcal{M} a1) \vee (\mathcal{M} a2))$ 
  using FstFixFst by fastforce
have 3:  $\vdash \triangleright (\triangleright (\mathcal{M} a1) \vee (\mathcal{M} a2)) = \triangleright (\mathcal{M} (a1 \text{ UPTO } a2))$ 
  using 2 by simp
from 1 2 3 show ?thesis by fastforce
qed
next
case (mTHRU-d a1 a2)
then show ?case
proof -
have 1:  $\vdash (\mathcal{M} (a1 \text{ THRU } a2)) = \triangleright (di (\mathcal{M} a1) \wedge di (\mathcal{M} a2))$ 
  by (simp)
have 2:  $\vdash \triangleright (di (\mathcal{M} a1) \wedge di (\mathcal{M} a2)) = \triangleright (\triangleright (di (\mathcal{M} a1) \wedge di (\mathcal{M} a2)))$ 
  using FstFixFst by fastforce
have 3:  $\vdash \triangleright (\triangleright (di (\mathcal{M} a1) \wedge di (\mathcal{M} a2))) = \triangleright (\mathcal{M} (a1 \text{ THRU } a2))$ 
  using 2 by simp
from 1 2 3 show ?thesis by fastforce
qed
next
case (mTHEN-d a1 a2)
then show ?case
proof -
have 1:  $\vdash (\mathcal{M} (a1 \text{ THEN } a2)) = (\mathcal{M} a1) ; (\mathcal{M} a2)$ 
  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:

$\vdash \mathcal{M}(\text{GUARD } \#False) = \#False$

proof –

```

  have 1:  $\vdash \mathcal{M}(\text{GUARD } \#False) = \mathcal{M}(\text{EMPTY WITH LIFT}(\text{init } \#False))$  by (simp add: GUARD-d-def)
  have 2:  $\vdash \mathcal{M}(\text{EMPTY WITH LIFT}(\text{init } \#False)) = (\mathcal{M}(\text{EMPTY}) \wedge (\text{init } \#False))$  by (simp )
  have 3:  $\vdash \#False = (\text{init } \#False)$  by (simp add: init-defs Valid-def)
  have 4:  $\vdash (\mathcal{M}(\text{EMPTY}) \wedge (\text{init } \#False)) = (\mathcal{M}(\text{EMPTY}) \wedge \#False)$  using 3 by auto
  have 5:  $\vdash (\mathcal{M}(\text{EMPTY}) \wedge \#False) = \#False$  by simp
  have 6:  $\vdash (\mathcal{M}(\text{EMPTY}) \wedge (\text{init } \#False)) = \#False$  using 4 5 by simp
  have 7:  $\vdash \mathcal{M}(\text{EMPTY WITH LIFT}(\text{init } \#False)) = \#False$  using 2 6 by fastforce
  have 8:  $\vdash \mathcal{M}(\text{GUARD } \#False) = \#False$  using 1 7 by fastforce
  from 8 show ?thesis by auto
qed

```

lemma MFirstFalseEqvFalse:

$\vdash \mathcal{M}(\text{FIRST LIFT } \#False) = \#False$

proof –

```

  have 1:  $\vdash \mathcal{M}(\text{FIRST LIFT } \#False) = \triangleright \#False$  by (simp )
  have 2:  $\vdash \mathcal{M}(\text{FIRST LIFT } \#False) = \#False$  using FstFalse by fastforce
  from 2 show ?thesis by auto
qed

```

lemma MFailAlt:

$\vdash \mathcal{M} \text{ FAIL} = \#False$

proof –

```

  have 1:  $\vdash \mathcal{M} \text{ FAIL} = \mathcal{M}(\text{GUARD } (\#False))$  by (simp add: FAIL-d-def)
  have 2:  $\vdash \mathcal{M}(\text{GUARD } (\#False)) = \#False$  using MGuardFalseEqvFalse by auto
  from 1 2 show ?thesis by fastforce
qed

```

lemma MFailEqvFirstFalseWithinEmpty:

$\text{FAIL} \simeq ((\text{FIRST LIFT } \#False) \text{ WITHIN } \text{empty})$

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: $\vdash \mathcal{M}(\text{EMPTY}) = \mathcal{M}(\text{FIRST LIFT empty})$ **by** (simp add: EMPTY-d-def)
have 2: $\vdash \mathcal{M}(\text{FIRST LIFT empty}) = \triangleright \text{empty}$ **by** (simp)
have 3: $\vdash \triangleright \text{empty} = \text{empty}$ **using** FstEmpty **by** auto
from 1 2 3 show ?thesis by fastforce

qed

lemma MSkipAlt:

$\vdash \mathcal{M} \text{ SKIP} = \text{skip}$

proof –

have 1: $\vdash \mathcal{M} \text{ SKIP} = \mathcal{M}(\text{FIRST LIFT skip})$ **by** (simp add: SKIP-d-def)
have 2: $\vdash \mathcal{M}(\text{FIRST LIFT skip}) = \triangleright \text{skip}$ **by** (simp)
have 3: $\vdash \triangleright \text{skip} = \text{skip}$ **using** FstSkip **by** simp
from 1 2 3 show ?thesis by fastforce

qed

lemma MGuardAlt:

$\vdash \mathcal{M}(\text{GUARD}(w)) = (\text{empty} \wedge \text{init } w)$

proof –

have 1: $\vdash \mathcal{M}(\text{GUARD}(w)) = \mathcal{M}(\text{EMPTY WITH } (\text{LIFT } (\text{init } w)))$ **by** (simp add: GUARD-d-def)
have 2: $\vdash \mathcal{M}(\text{EMPTY WITH } (\text{LIFT } (\text{init } w))) = (\mathcal{M}(\text{EMPTY}) \wedge (\text{init } w))$ **by** (simp)
have 3: $\vdash (\mathcal{M}(\text{EMPTY}) \wedge (\text{init } w)) = (\text{empty} \wedge (\text{init } w))$ **using** 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:

$\vdash \mathcal{M}(a \text{ ALWAYS } w) = (\mathcal{M}(a) \wedge \Box (\text{init } w))$

proof –

have 1: $\vdash \mathcal{M}(a \text{ ALWAYS } w) = \mathcal{M}(a \text{ WITH LIFT } (bi (\text{fin } (\text{init } w))))$
by (simp add: ALWAYS-d-def)
have 2: $\vdash \mathcal{M}(a \text{ WITH LIFT } (bi (\text{fin } (\text{init } w)))) = (\mathcal{M}(a) \wedge (bi (\text{fin } (\text{init } w))))$

by (simp)
 have 3: $\vdash (\mathcal{M}(a) \wedge (bi (fin (init w)))) = (\mathcal{M}(a) \wedge \Box (init w))$
 using BoxStateEqvBiFinState by fastforce
 from 1 2 3 show ?thesis by fastforce
 qed

lemma MSometimeAlt:

$\vdash \mathcal{M}(a \text{ SOMETIME } w) = (\mathcal{M}(a) \wedge \Diamond (init w))$

proof –

have 1: $\vdash \mathcal{M}(a \text{ SOMETIME } w) = \mathcal{M}(a \text{ WITH LIFT } (di (fin (init w))))$
 by (simp add: SOMETIME-d-def)
 have 2: $\vdash \mathcal{M}(a \text{ WITH LIFT } (di (fin (init w)))) = (\mathcal{M}(a) \wedge (di (fin (init w))))$
 by (simp)
 have 3: $\vdash \mathcal{M}(a \text{ WITH LIFT } (di (fin (init w)))) = (\mathcal{M}(a) \wedge \Diamond (init w))$
 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 MEmptyAlt 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: $\vdash \mathcal{M}(a \text{ TIMES } Suc \ k) = \mathcal{M}(a \text{ THEN } (a \text{ TIMES } k))$
 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*:

$\vdash \mathcal{M}(a \text{ UPTO } b) = ((\mathcal{M} a) \wedge \text{bi } (\neg(\mathcal{M} b))) \vee ((\mathcal{M} b) \wedge \text{bi } (\neg(\mathcal{M} a))) \vee ((\mathcal{M} a) \wedge (\mathcal{M} b))$

proof –

have 1: $\vdash \mathcal{M}(a \text{ UPTO } b) = \triangleright((\mathcal{M} a) \vee (\mathcal{M} b))$

by *(simp)*

have 2: $\vdash \triangleright((\mathcal{M} a) \vee (\mathcal{M} b)) = ((\triangleright(\mathcal{M} a) \wedge (\text{bs } (\neg(\mathcal{M} b)))) \vee (\triangleright(\mathcal{M} b) \wedge (\text{bs } (\neg(\mathcal{M} a))))$

using *FstWithOrEqv* by *blast*

have 3: $\vdash ((\triangleright(\mathcal{M} a) \wedge (\text{bs } (\neg(\mathcal{M} b)))) \vee (\triangleright(\mathcal{M} b) \wedge (\text{bs } (\neg(\mathcal{M} a)))) =$
 $((\mathcal{M} a) \wedge ((\mathcal{M} b) \vee \neg(\mathcal{M} b)) \wedge (\text{bs } (\neg(\mathcal{M} b)))) \vee$
 $((\mathcal{M} b) \wedge ((\mathcal{M} a) \vee \neg(\mathcal{M} a)) \wedge (\text{bs } (\neg(\mathcal{M} a))))$

using *MFixFst* by *fastforce*

have 4: $\vdash (((\mathcal{M} a) \wedge ((\mathcal{M} b) \vee \neg(\mathcal{M} b)) \wedge (\text{bs } (\neg(\mathcal{M} b)))) \vee$
 $((\mathcal{M} b) \wedge ((\mathcal{M} a) \vee \neg(\mathcal{M} a)) \wedge (\text{bs } (\neg(\mathcal{M} a)))) =$
 $((\mathcal{M} a) \wedge ((\mathcal{M} b) \wedge \text{bs } (\neg(\mathcal{M} b))) \vee (\neg(\mathcal{M} b) \wedge \text{bs } (\neg(\mathcal{M} b)))) \vee$
 $((\mathcal{M} b) \wedge ((\mathcal{M} a) \wedge \text{bs } (\neg(\mathcal{M} a))) \vee (\neg(\mathcal{M} a) \wedge \text{bs } (\neg(\mathcal{M} a))))$

by *auto*

have 5: $\vdash (((\mathcal{M} a) \wedge ((\mathcal{M} b) \wedge \text{bs } (\neg(\mathcal{M} b))) \vee (\neg(\mathcal{M} b) \wedge \text{bs } (\neg(\mathcal{M} b)))) \vee$
 $((\mathcal{M} b) \wedge ((\mathcal{M} a) \wedge \text{bs } (\neg(\mathcal{M} a))) \vee (\neg(\mathcal{M} a) \wedge \text{bs } (\neg(\mathcal{M} a)))) =$
 $((\mathcal{M} a) \wedge ((\triangleright(\mathcal{M} b)) \vee (\neg(\mathcal{M} b) \wedge \text{bs } (\neg(\mathcal{M} b)))) \vee$
 $((\mathcal{M} b) \wedge ((\triangleright(\mathcal{M} a)) \vee (\neg(\mathcal{M} a) \wedge \text{bs } (\neg(\mathcal{M} a))))$

by *(simp add: first-d-def)*

have 6: $\vdash (((\mathcal{M} a) \wedge ((\triangleright(\mathcal{M} b)) \vee (\neg(\mathcal{M} b) \wedge \text{bs } (\neg(\mathcal{M} b)))) \vee$
 $((\mathcal{M} b) \wedge ((\triangleright(\mathcal{M} a)) \vee (\neg(\mathcal{M} a) \wedge \text{bs } (\neg(\mathcal{M} a)))) =$
 $((\mathcal{M} a) \wedge ((\mathcal{M} b) \vee (\neg(\mathcal{M} b) \wedge \text{bs } (\neg(\mathcal{M} b)))) \vee$
 $((\mathcal{M} b) \wedge ((\mathcal{M} a) \vee (\neg(\mathcal{M} a) \wedge \text{bs } (\neg(\mathcal{M} a))))$

using *MFixFst* by *fastforce*

have 7: $\vdash (\neg(\mathcal{M} b) \wedge \text{bs } (\neg(\mathcal{M} b))) = \text{bi}(\neg(\mathcal{M} b))$

using *AndBsEqvBi* by *blast*

have 8: $\vdash (\neg(\mathcal{M} a) \wedge \text{bs } (\neg(\mathcal{M} a))) = \text{bi}(\neg(\mathcal{M} a))$

using *AndBsEqvBi* by *blast*

have 9: $\vdash (((\mathcal{M} a) \wedge ((\mathcal{M} b) \vee ((\neg(\mathcal{M} b)) \wedge \text{bs}(\neg(\mathcal{M} b)))) \vee$
 $((\mathcal{M} b) \wedge ((\mathcal{M} a) \vee ((\neg(\mathcal{M} a)) \wedge \text{bs}(\neg(\mathcal{M} a)))) =$
 $((\mathcal{M} a) \wedge ((\mathcal{M} b) \vee (\text{bi}(\neg(\mathcal{M} b)))) \vee$
 $((\mathcal{M} b) \wedge ((\mathcal{M} a) \vee (\text{bi}(\neg(\mathcal{M} a))))$

using 7 8 by *fastforce*

have 10: $\vdash (((\mathcal{M} a) \wedge ((\mathcal{M} b) \vee (\text{bi}(\neg(\mathcal{M} b)))) \vee$
 $((\mathcal{M} b) \wedge ((\mathcal{M} a) \vee (\text{bi}(\neg(\mathcal{M} a)))) =$
 $((\mathcal{M} a) \wedge ((\mathcal{M} b) \vee ((\mathcal{M} a) \wedge \text{bi}(\neg(\mathcal{M} b)))) \vee$
 $((\mathcal{M} b) \wedge ((\mathcal{M} a) \vee ((\mathcal{M} b) \wedge \text{bi}(\neg(\mathcal{M} a))))$

by *auto*

have 11: $\vdash (((\mathcal{M} a) \wedge (\mathcal{M} b)) \vee ((\mathcal{M} a) \wedge \text{bi}(\neg(\mathcal{M} b))) \vee$
 $((\mathcal{M} b) \wedge (\mathcal{M} a)) \vee ((\mathcal{M} b) \wedge \text{bi}(\neg(\mathcal{M} a)))) =$
 $((\mathcal{M} a) \wedge \text{bi}(\neg(\mathcal{M} b))) \vee ((\mathcal{M} b) \wedge \text{bi}(\neg(\mathcal{M} a))) \vee ((\mathcal{M} a) \wedge (\mathcal{M} b))$

by auto
 from 1 2 3 4 5 6 9 10 11 show ?thesis by (metis 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*:
 $(a \text{ THEN FAIL}) \simeq \text{FAIL}$
proof –
 have 1: $\vdash \mathcal{M} (a \text{ THEN FAIL}) = (\mathcal{M} a);(\mathcal{M} \text{ FAIL})$ 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: $\vdash \mathcal{M} (\text{EMPTY UPTO } a) = \triangleright(\mathcal{M} \text{ EMPTY} \vee (\mathcal{M} a))$ **by** (simp)

have 2: $\vdash \mathcal{M} \text{ EMPTY} = \text{empty}$ **using** MEmptyAlt **by** auto

hence 3: $\vdash (\mathcal{M} \text{ EMPTY} \vee (\mathcal{M} a)) = (\text{empty} \vee (\mathcal{M} a))$ **by** auto

hence 4: $\vdash \triangleright(\mathcal{M} \text{ EMPTY} \vee \mathcal{M} a) = \triangleright(\text{empty} \vee \mathcal{M} a)$ **using** FstEqvRule **by** blast

have 5: $\vdash \triangleright(\text{empty} \vee \mathcal{M} a) = \text{empty}$ **using** FstEmptyOrEqvEmpty **by** blast

have 6: $\vdash \text{empty} = \mathcal{M} \text{ EMPTY}$ **using** MEmptyAlt **by** auto

from 1 4 5 6 **show** ?thesis **using** MonEq **by** (metis int-eq)

qed

lemma MEmptyThru:

$(\text{EMPTY THRU } a) \simeq (a)$

proof –

have 1: $\vdash \mathcal{M}(\text{EMPTY THRU } a) = \triangleright(\text{di}(\mathcal{M} \text{ EMPTY}) \wedge \text{di}(\mathcal{M} a))$ **by** (simp)

have 2: $\vdash \text{di}(\mathcal{M} \text{ EMPTY}) = \text{di empty}$ **using** MEmptyAlt DiEqvDi **by** blast

hence 3: $\vdash (\text{di}(\mathcal{M} \text{ EMPTY}) \wedge \text{di}(\mathcal{M} a)) = (\text{di empty} \wedge \text{di}(\mathcal{M} a))$ **by** auto

hence 4: $\vdash (\text{di empty} \wedge \text{di}(\mathcal{M} a)) = \text{di}(\mathcal{M} a)$ **using** DiEmpty **by** auto

have 5: $\vdash (\text{di}(\mathcal{M} \text{ EMPTY}) \wedge \text{di}(\mathcal{M} a)) = \text{di}(\mathcal{M} a)$ **using** 3 4 **by** fastforce

hence 6: $\vdash \triangleright(\text{di}(\mathcal{M} \text{ EMPTY}) \wedge \text{di}(\mathcal{M} a)) = \triangleright(\text{di}(\mathcal{M} a))$ **using** FstEqvRule **by** blast

have 7: $\vdash \triangleright(\text{di}(\mathcal{M} a)) = \triangleright(\mathcal{M} a)$ **using** FstDiEqvFst **by** blast

have 8: $\vdash \triangleright(\mathcal{M} a) = (\mathcal{M} a)$ **using** MFixFst **by** fastforce

from 1 6 7 8 **show** ?thesis **using** MonEq **by** (metis int-eq)

qed

lemma MThenEmpty:

$(a \text{ THEN } \text{EMPTY}) \simeq (a)$

proof –

have 1: $\vdash \mathcal{M}(a \text{ THEN } \text{EMPTY}) = (\mathcal{M} a); (\mathcal{M} \text{ EMPTY})$ **by** (simp)

have 2: $\vdash (\mathcal{M} a); (\mathcal{M} \text{ EMPTY}) = (\mathcal{M} a); \text{empty}$ **by** (simp add: MEmptyAlt RightChopEqvChop)

have 3: $\vdash (\mathcal{M} a); \text{empty} = (\mathcal{M} a)$ **using** ChopEmpty **by** auto

from 1 2 3 **show** ?thesis **using** MonEq **by** (metis int-eq)

qed

lemma *MEmpyThen:*

$(\text{EMPTY THEN } a) \simeq a$

proof –

have 1: $\vdash \mathcal{M}(\text{EMPTY THEN } a) = (\mathcal{M} \text{ EMPTY});(\mathcal{M} a)$ **by** (*simp*)

have 2: $\vdash (\mathcal{M} \text{ EMPTY});(\mathcal{M} a) = \text{empty};(\mathcal{M} a)$ **by** (*simp add: MEmpyAlt LeftChopEqvChop*)

have 3: $\vdash \text{empty};(\mathcal{M} a) = (\mathcal{M} a)$ **by** (*simp add: EmptyChop*)

from 1 2 3 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)

qed

lemma *MEmpyIterate:*

$(\text{EMPTY ITERATE } b) \simeq \text{EMPTY}$

proof –

have 1: $\vdash \mathcal{M}(\text{EMPTY ITERATE } b) = \mathcal{M}(\text{EMPTY WITH LIFT } (\mathcal{M} b)^*)$
by (*simp add: ITERATE-d-def*)

have 2: $\vdash \mathcal{M}(\text{EMPTY WITH LIFT } (\mathcal{M} b)^*) = (\mathcal{M} \text{ EMPTY} \wedge (\mathcal{M} b)^*)$
by (*simp*)

have 3: $\vdash (\mathcal{M} \text{ EMPTY} \wedge (\mathcal{M} b)^*) = (\text{empty} \wedge (\mathcal{M} b)^*)$
using *MEmpyAlt* **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 *MIterateDemp:*

$(a \text{ ITERATE } a) \simeq (a)$

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 *MUptoldemp:*

$(a \text{ UPTO } a) \simeq (a)$

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 *MThruDemp:*

$(a \text{ THRU } a) \simeq (a)$

proof –
have 1: $\vdash \mathcal{M}(a \text{ THRU } a) = \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} a))$ **by** *auto*
have 2: $\vdash \triangleright(di(\mathcal{M} a) \wedge di(\mathcal{M} a)) = \triangleright(di(\mathcal{M} a))$ **using** *FstEqvRule* **by** *fastforce*
have 3: $\vdash \triangleright(di(\mathcal{M} a)) = \triangleright(\mathcal{M} a)$ **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 (*simp add: first-d-def, auto*)
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 (simp add: first-d-def, auto)

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: $\vdash ((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)) \wedge$

$$bs(\neg((di(\mathcal{M} a)) \wedge ((\mathcal{M} a) \vee di(\mathcal{M} b)))) =$$

$((\mathcal{M} a) \wedge \text{bs}(\neg(\text{di}(\mathcal{M} a) \wedge \text{di}(\mathcal{M} b))))$
using 15 16 17 18 19 **by** (metis int-eq)
have 34: $\vdash ((\mathcal{M} a) \wedge \text{bs}(\neg(\text{di}(\mathcal{M} a) \wedge \text{di}(\mathcal{M} b)))) = (\mathcal{M} a)$
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: $\vdash \mathcal{M}(a \text{ THRU } (a \text{ UPTO } b)) = \triangleright(\text{di}(\mathcal{M} a) \wedge \text{di}(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))))$
by simp
have 2: $\vdash \triangleright(\text{di}(\mathcal{M} a) \wedge \text{di}(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b)))) =$
 $\triangleright(\text{di}(\mathcal{M} a) \wedge \text{di}(((\mathcal{M} a) \vee (\mathcal{M} b))))$
by (metis DiEqvDiFst FstEqvRule inteq-reflection lift-and-com)
have 3: $\vdash \triangleright(\text{di}(\mathcal{M} a) \wedge \text{di}(((\mathcal{M} a) \vee (\mathcal{M} b)))) =$
 $\triangleright(\text{di}(\mathcal{M} a) \wedge (\text{di}(\mathcal{M} a) \vee \text{di}(\mathcal{M} b)))$
by (metis DiOrEqv FstEqvRule inteq-reflection lift-and-com)
have 4: $\vdash (\text{di}(\mathcal{M} a) \wedge (\text{di}(\mathcal{M} a) \vee \text{di}(\mathcal{M} b))) = (\text{di}(\mathcal{M} a))$
by auto
hence 5: $\vdash \triangleright(\text{di}(\mathcal{M} a) \wedge (\text{di}(\mathcal{M} a) \vee \text{di}(\mathcal{M} b))) = \triangleright(\text{di}(\mathcal{M} a))$
using FstEqvRule **by** blast
have 6: $\vdash \triangleright(\text{di}(\mathcal{M} a)) = \triangleright(\mathcal{M} a)$
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: $\vdash \mathcal{M}((a \text{ UPTO } b) \text{ THRU } (a \text{ UPTO } c)) =$
 $\triangleright(\text{di}(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge \text{di}(\triangleright((\mathcal{M} a) \vee (\mathcal{M} c))))$
by simp
have 2: $\vdash (\text{di}(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge \text{di}(\triangleright((\mathcal{M} a) \vee (\mathcal{M} c)))) =$
 $(\text{di}(((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge \text{di}(((\mathcal{M} a) \vee (\mathcal{M} c))))$
using DiEqvDiFst **by** fastforce
have 3: $\vdash (\text{di}(((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge \text{di}(((\mathcal{M} a) \vee (\mathcal{M} c)))) =$
 $((\text{di}(\mathcal{M} a) \vee \text{di}(\mathcal{M} b)) \wedge (\text{di}(\mathcal{M} a) \vee \text{di}(\mathcal{M} c)))$
using DiOrEqv **by** fastforce
have 4: $\vdash ((\text{di}(\mathcal{M} a) \vee \text{di}(\mathcal{M} b)) \wedge (\text{di}(\mathcal{M} a) \vee \text{di}(\mathcal{M} c))) =$
 $(\text{di}(\mathcal{M} a) \vee (\text{di}(\mathcal{M} b) \wedge \text{di}(\mathcal{M} c)))$
by auto
have 5: $\vdash (\text{di}(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge \text{di}(\triangleright((\mathcal{M} a) \vee (\mathcal{M} c)))) =$
 $(\text{di}(\mathcal{M} a) \vee (\text{di}(\mathcal{M} b) \wedge \text{di}(\mathcal{M} c)))$
using 2 3 4 **by** fastforce
hence 6: $\vdash \triangleright(\text{di}(\triangleright((\mathcal{M} a) \vee (\mathcal{M} b))) \wedge \text{di}(\triangleright((\mathcal{M} a) \vee (\mathcal{M} c)))) =$
 $\triangleright(\text{di}(\mathcal{M} a) \vee (\text{di}(\mathcal{M} b) \wedge \text{di}(\mathcal{M} c)))$
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 UPTO a)  $\simeq$  (a UPTO c)
    using MUptoCommut by auto
  have 4: (c UPTO b)  $\simeq$  (b UPTO c)
    using MUptoCommut by auto
  have 5: ((c UPTO a) THRU (c UPTO b))  $\simeq$  ((a UPTO c) THRU (c UPTO b))
    using 3 by (simp add: MonEqRefl MonEqSubstThru)
  have 6: ((a UPTO c) THRU (c UPTO b))  $\simeq$  ((c UPTO b) THRU (a UPTO c))
    using MThruCommut by auto
  have 7: ((c UPTO b) THRU (a UPTO c))  $\simeq$  ((b UPTO c) THRU (a 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 UPTO b) THRU c)  $\simeq$  (c THRU (a 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:

$((a \text{ AND } b) \text{ WITH } f) \simeq ((a \text{ WITH } f) \text{ AND } (b \text{ WITH } f))$

proof —

```

  have 1:  $\vdash \mathcal{M}((a \text{ AND } b) \text{ WITH } f) = (\mathcal{M}(a \text{ AND } b) \wedge f)$ 
    by (simp)
  have 2:  $\vdash \mathcal{M}(a \text{ AND } b) = \mathcal{M}(a \text{ WITH LIFT } (\mathcal{M} b))$ 
    by (simp add: AND-d-def)
  have 3:  $\vdash (\mathcal{M}(a \text{ AND } b) \wedge f) = (\mathcal{M}(a \text{ WITH LIFT } (\mathcal{M} b)) \wedge f)$ 
    using 2 by auto
  have 4:  $\vdash \mathcal{M}(a \text{ WITH } (\text{LIFT } (\mathcal{M} b) \wedge f)) = (\mathcal{M}(a) \wedge \mathcal{M}(b) \wedge f)$ 
    by simp
  have 5:  $\vdash (\mathcal{M}(a) \wedge \mathcal{M}(b) \wedge f) = ((\mathcal{M}(a) \wedge f) \wedge (\mathcal{M}(b) \wedge f))$ 
    by auto
  have 6:  $\vdash ((\mathcal{M}(a) \wedge f) \wedge (\mathcal{M}(b) \wedge f)) = (\mathcal{M}(a \text{ WITH } f) \wedge \mathcal{M}(b \text{ WITH } f))$ 
    by simp
  have 7:  $\vdash (\mathcal{M}(a \text{ WITH } f) \wedge \mathcal{M}(b \text{ WITH } f)) = \mathcal{M}((a \text{ WITH } f) \text{ WITH LIFT } (\mathcal{M}(b \text{ WITH } f)))$ 

```

by simp
 have 8: $\vdash \mathcal{M}((a \text{ WITH } f) \text{ WITH LIFT}(\mathcal{M}(b \text{ WITH } f))) = \mathcal{M}((a \text{ WITH } f) \text{ AND } (b \text{ WITH } f))$
 by (simp add: AND-d-def)
 from 1 2 3 4 5 6 7 8 show ?thesis using MonEq by (metis AND-d-def MWithAbsorp int-eq)
 qed

lemma *MHalWithAndDistrib:*

$((\text{HALT } w) \text{ WITH } f) \text{ AND } ((\text{HALT } w) \text{ WITH } g) \simeq ((\text{HALT } w) \text{ WITH LIFT}(f \wedge g))$

proof –

have 1: $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ AND } ((\text{HALT } w) \text{ WITH } g)) =$
 $\mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ WITH LIFT}(\mathcal{M}((\text{HALT } w) \text{ WITH } g)))$
 by (simp add: AND-d-def)
 have 2: $\vdash \mathcal{M}(((\text{HALT } w) \text{ WITH } f) \text{ WITH LIFT}(\mathcal{M}((\text{HALT } w) \text{ WITH } g))) =$
 $(\mathcal{M}(\text{HALT } w) \wedge f \wedge \mathcal{M}(\text{HALT } w) \wedge g)$
 by auto
 have 3: $\vdash (\mathcal{M}(\text{HALT } w) \wedge f \wedge \mathcal{M}(\text{HALT } w) \wedge g) = (\mathcal{M}(\text{HALT } w) \wedge f \wedge g)$
 by auto
 have 4: $\vdash (\mathcal{M}(\text{HALT } w) \wedge f \wedge g) = \mathcal{M}((\text{HALT } w) \text{ WITH LIFT}(f \wedge g))$
 by auto
 from 1 2 3 4 show ?thesis using MonEq by (metis int-eq)

qed

lemma *MHalWithUptoHalWithEqvHalWithOr:*

$((\text{HALT } w) \text{ WITH } f) \text{ UPTO } ((\text{HALT } w) \text{ WITH } g) \simeq ((\text{HALT } w) \text{ WITH LIFT}(f \vee g))$

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 fastforce
 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 *MHalWithThruHalWithEqvHalWithAndHalWith:*

$((\text{HALT } w) \text{ WITH } f) \text{ THRU } ((\text{HALT } w) \text{ WITH } g) \simeq ((\text{HALT } w) \text{ WITH } f) \text{ AND } ((\text{HALT } w) \text{ WITH } g)$

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) inteq-reflection lift-and-com*)
have 3: $\vdash (di(halt(init\ w) \wedge f) \wedge di(halt(init\ w) \wedge g)) =$
 $di(halt(init\ w) \wedge f \wedge g)$
using *DiHaltAndDiHaltAndEqvDiHaltAndAnd* **by** *fastforce*
have 4: $\vdash di(halt(init\ w) \wedge f \wedge g) = di(\mathcal{M}(HALT\ w) \wedge f \wedge g)$
by (*metis DiEqvDi MHaltAlt inteq-reflection lift-and-com*)
have 5: $\vdash (di(\mathcal{M}(HALT\ w) \wedge f) \wedge di(\mathcal{M}(HALT\ w) \wedge g)) = di(\mathcal{M}(HALT\ w) \wedge f \wedge g)$
using 2 3 4 **by** *fastforce*
have 6: $\vdash \triangleright(di(\mathcal{M}(HALT\ w) \wedge f) \wedge di(\mathcal{M}(HALT\ w) \wedge g)) = \triangleright(di(\mathcal{M}(HALT\ w) \wedge f \wedge g))$
using 5 *FstEqvRule* **by** *blast*
have 7: $\vdash \triangleright(di(\mathcal{M}(HALT\ w) \wedge f \wedge g)) = \triangleright(\mathcal{M}(HALT\ w) \wedge f \wedge g)$
using *FstDiEqvFst* **by** *fastforce*
have 8: $\vdash \triangleright(\mathcal{M}(HALT\ w) \wedge f \wedge g) = \triangleright(\mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)))$
by *simp*
have 9: $\vdash \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)) = \triangleright(\mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)))$
using *MFixFst* **by** *blast*
have 10: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ THRU\ ((HALT\ w)\ WITH\ g)) = \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g))$
using 1 2 3 4 5 6 7 8 9 *int-eq* **by** *metis*
have 11: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g)) = \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g))$
using *MHaltWithAndDistrib* **using** *eq-d-def* **by** *blast*
have 12: $\vdash \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)) = \mathcal{M}(((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g))$
using 11 **by** *fastforce*
from 10 12 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThenAndDistrib*:

$(a\ THEN\ (b\ AND\ c)) \simeq ((a\ THEN\ b)\ AND\ (a\ THEN\ c))$

proof –

have 1: $\vdash \mathcal{M}(a\ THEN\ (b\ AND\ c)) = (\mathcal{M}(a)) ; (\mathcal{M}(b\ AND\ c))$
by *simp*
have 2: $\vdash (\mathcal{M}(a)) ; (\mathcal{M}(b\ AND\ c)) = (\mathcal{M}(a)) ; (\mathcal{M}(b) \wedge \mathcal{M}(c))$
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: $\vdash ((\mathcal{M}(a)) ; (\mathcal{M}(b))) \wedge ((\mathcal{M}(a)) ; (\mathcal{M}(c))) =$
 $(\mathcal{M}(a\ THEN\ b) \wedge \mathcal{M}(a\ THEN\ c))$
by *simp*
have 7: $\vdash (\mathcal{M}(a\ THEN\ b) \wedge \mathcal{M}(a\ THEN\ c)) = \mathcal{M}((a\ THEN\ b)\ AND\ (a\ THEN\ c))$
by (*simp add: AND-d-def*)
from 1 2 3 4 5 6 7 **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThenUptoDistrib*:

$(a\ THEN\ (b\ UPTO\ c)) \simeq ((a\ THEN\ b)\ UPTO\ (a\ THEN\ c))$

proof –

have 1: $\vdash (\mathcal{M} (a \text{ THEN } (b \text{ UPTO } c))) = ((\mathcal{M} a); \triangleright((\mathcal{M} b) \vee (\mathcal{M} c)))$
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 *MThenThruDistrib*:

$(a \text{ THEN } (b \text{ THRU } c)) \simeq ((a \text{ THEN } b) \text{ THRU } (a \text{ THEN } c))$

proof –

have 1: $\vdash \mathcal{M}(a \text{ THEN } (b \text{ THRU } c)) = (\mathcal{M} a); \triangleright(di(\mathcal{M} b) \wedge di(\mathcal{M} c))$
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

16 Finite ITL Examples

```

theory Example
imports
  FOTheorems TimeReversal
begin

```

16.1 Example 1

```

definition F1 :: nat statefun  $\Rightarrow$  temporal
where F1 w  $\equiv$  TEMP  $\square$  ( #0  $\leq$  $w )

```

```

definition Init1 :: nat statefun  $\Rightarrow$  temporal
where Init1 w  $\equiv$  TEMP $w = #0

```

```

lemma init1:
  ( $\langle s0, s1, s2 \rangle \models len(2) \wedge Init1 w$ ) = ( $(w s0) = 0$ )
by (simp add: Init1-def current-val-d-def len-defs)

```

```

lemma exist-test-F1 :
   $\vdash \exists \exists w. F1 w$ 

```

```

proof -
  have 1:  $\bigwedge w. \vdash F1 w$  by (simp add: always-defs current-val-d-def F1-def Valid-def)
  from 1 show ?thesis by (meson EExI MP)
qed

```

16.2 Example 2

```

locale Test =
  fixes v :: state  $\Rightarrow$  nat
  fixes v1 :: state  $\Rightarrow$  nat

```

```

fixes y :: state ⇒ bool
fixes z :: state ⇒ int
fixes F2 :: nat statefun ⇒ temporal
fixes F3 :: bool statefun ⇒ temporal
fixes F4 :: int statefun ⇒ temporal
fixes F5 :: nat statefun ⇒ temporal
fixes Init2 :: nat statefun ⇒ temporal
fixes Init3 :: bool statefun ⇒ temporal
defines F2 ≡ (λ v. TEMP □ ( #0 ≤ $v ))
defines F3 ≡ (λ p. TEMP □ ( $p ∨ ¬ $p))
defines F4 ≡ (λ z. TEMP □ ( #0 ≤ $z ∨ $z < #0))
defines F5 ≡ (λ v. TEMP $v = #0 ∧ v gets $v + #1)
defines Init2 ≡ (λ v. TEMP $v = #0)
defines Init3 ≡ (λ p. TEMP $p)

```

```

lemma (in Test) currentval-test :
  (s ⊨ ($v = #0)) = ( (v (nth s 0)) = 0)
by (simp add: current-val-d-def)

```

```

lemma (in Test) nextempty-test :
  (⟨s0⟩ ⊨ v$) = (ε x. x=x)
by (simp add: next-val-d-def)

```

```

lemma (in Test) nextempty-test-1 :
  (⟨s0⟩ ⊨ v$ = v$)
by simp

```

```

lemma (in Test) nextempty-test-2 :
  (⟨s0⟩ ⊨ v$ = v1$)
by (simp add: Test.nextempty-test)

```

```

lemma (in Test) nextcurrent-test:
  (⟨s0,s1⟩ ⊨ skip ∧ ($v = #0) ∧ (v$ = $v + #1)) = (((v s0) = 0) ∧ ((v s1) = 1 ))
unfolding current-val-d-def next-val-d-def skip-defs by auto

```

```

lemma (in Test) nextcurrentfinpenult-test:
  (⟨s0,s1,s2,s3⟩ ⊨ len(3) ∧ v =: !v - #1 ∧ v ← #3 ∧ $v = #0 ∧ v := $v + #1 ) =
  (((v s0) = 0) ∧ ((v s1) = 1 ∧ (v (s2)) = 2 ∧ ((v s3) = 3 )))
unfolding current-val-d-def next-val-d-def fin-val-d-def penult-val-d-def
  next-assign-d-def prev-assign-d-def temporal-assign-d-def len-defs by auto

```

```

lemma (in Test) stable-test:
  (⟨s0,s1,s2,s3⟩ ⊨ len(3) ∧ stable v ∧ $v = #0) =
  ((v s0) = 0 ∧ (v s1) = 0 ∧ (v s2) = 0 ∧ (v s3) = 0)
by (auto simp: stable-defs len-defs
  current-val-d-def next-val-d-def Nitpick.case-nat-unfold)

```

```

lemma (in Test) revnextcurrentfinpenult-test:

```

$(\langle s0, s1, s2, s3 \rangle \models (\text{len } 3 \wedge v! = !v - \#1 \wedge !v = \#3 \wedge \$v = \#0 \wedge v\$ = \$v + \#1)^r) =$
 $((v \text{ s3}) = 0) \wedge ((v \text{ s2}) = 1 \wedge (v \text{ (s1)}) = 2 \wedge ((v \text{ s0}) = 3 \text{ })))$
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: always-defs current-val-d-def 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: always-defs current-val-d-def F3-def Valid-def)

from 1 **show** ?thesis **by** (meson EExI MP)

qed

16.3 Example 3

locale Test1 =

fixes $v :: \text{state} \Rightarrow \text{nat}$

fixes $F5 :: \text{nat} \text{ statefun} \Rightarrow \text{nat} \Rightarrow \text{temporal}$

defines $F5 \equiv (\lambda v \ n. \text{TEMP } \$v = \#0 \wedge v \text{ gets } \$v + \#1 \wedge \text{fin}(\$v = \#n))$

lemma (in Test1) test-E-F5-1:

(

$x \ (\text{Interval.nth } w \ (0::\text{nat})) = (0::\text{nat}) \wedge$

$(\forall i < \text{intlen } w. x \ (\text{Interval.nth } w \ (\text{Suc } i)) = \text{Suc } (x \ (\text{Interval.nth } w \ i))) \wedge$

$x \ (\text{Interval.nth } w \ (\text{intlen } w)) = n \longrightarrow$

(

$x \ (\text{Interval.nth } w \ (0::\text{nat})) = (0::\text{nat}) \wedge$

$(\forall i \leq \text{intlen } w. x \ (\text{Interval.nth } w \ (i)) = i) \wedge$

$x \ (\text{Interval.nth } w \ (\text{intlen } w)) = n$)

apply simp

proof

assume 0: $x \ (\text{Interval.nth } w \ (0::\text{nat})) = (0::\text{nat}) \wedge$

$(\forall i < \text{intlen } w. x \ (\text{Interval.nth } w \ (\text{Suc } i)) = \text{Suc } (x \ (\text{Interval.nth } w \ i))) \wedge$

$x \ (\text{Interval.nth } w \ (\text{intlen } w)) = n$

have 1: $x \ (\text{Interval.nth } w \ (0::\text{nat})) = (0::\text{nat})$ **using** 0 **by** auto

have 2: $x \ (\text{Interval.nth } w \ (\text{intlen } w)) = n$ **using** 0 **by** auto

have 3: $(\forall i < \text{intlen } w. x \ (\text{Interval.nth } w \ (\text{Suc } i)) = \text{Suc } (x \ (\text{Interval.nth } w \ i)))$ **using** 0 **by** auto

show $\forall i \leq \text{intlen } w. x \ (\text{Interval.nth } w \ i) = i$

proof

fix i

show $i \leq \text{intlen } w \longrightarrow x \ (\text{Interval.nth } w \ i) = i$

proof

(*induct* i)

```

case 0
then show ?case using 1 by simp
next
case (Suc i)
then show ?case by (simp add: 3)
qed
qed
qed

```

lemma (in Test1) test-E-F5-2:

```

(
  x (Interval.nth w (0::nat)) = (0::nat) ∧
  (∀ i ≤ intlen w. x (Interval.nth w (i)) = i) ∧
  x (Interval.nth w (intlen w)) = n) → (
  x (Interval.nth w (0::nat)) = (0::nat) ∧
  (∀ i < intlen w. x (Interval.nth w (Suc i)) = Suc (x (Interval.nth w i))) ∧
  x (Interval.nth w (intlen w)) = n)

```

by simp

lemma (in Test1) test-E-F5-3:

```

(
  x (Interval.nth w (0::nat)) = (0::nat) ∧
  (∀ i < intlen w. x (Interval.nth w (Suc i)) = Suc (x (Interval.nth w i))) ∧
  x (Interval.nth w (intlen w)) = n) =
  (
    x (Interval.nth w (0::nat)) = (0::nat) ∧
    (∀ i ≤ intlen w. x (Interval.nth w (i)) = i) ∧
    x (Interval.nth w (intlen w)) = n)

```

using test-E-F5-1 test-E-F5-2 **by** auto

lemma (in Test1) test-E-F5-4:

```

(∃ x::state ⇒ nat.
  x (Interval.nth w (0::nat)) = (0::nat) ∧
  (∀ i < intlen w. x (Interval.nth w (Suc i)) = Suc (x (Interval.nth w i))) ∧
  x (Interval.nth w (intlen w)) = n) =
  (∃ x::state ⇒ nat.
    x (Interval.nth w (0::nat)) = (0::nat) ∧
    (∀ i ≤ intlen w. x (Interval.nth w (i)) = i) ∧
    x (Interval.nth w (intlen w)) = n)

```

by (simp add: Test1.test-E-F5-3)

lemma (in Test1) test-E-F5:

$\vdash (\exists \exists v. (F5 \vee n)) \longrightarrow (\text{len } n)$

apply (simp add: Valid-def F5-def exist-state-d-def gets-defs current-val-d-def
fin-defs sub-def len-defs)

proof

fix w

show (∃ x::state ⇒ nat.


```

x (Interval.nth w (0::nat)) = (0::nat) ∧
(∀ i < intlen w. x (Interval.nth w (Suc i)) = Suc (x (Interval.nth w i))) ∧
x (Interval.nth w (intlen w)) = n) →
(intlen w = n)
proof -
have 1: (∃ x::state ⇒ nat.
x (Interval.nth w (0::nat)) = (0::nat) ∧
(∀ i < intlen w. x (Interval.nth w (Suc i)) = Suc (x (Interval.nth w i))) ∧
x (Interval.nth w (intlen w)) = n) =
(∃ x::state ⇒ nat.
x (Interval.nth w (0::nat)) = (0::nat) ∧
(∀ i ≤ intlen w. x (Interval.nth w (i)) = i) ∧
x (Interval.nth w (intlen w)) = n) using test-E-F5-4 by auto
have 2: (∃ x::state ⇒ nat.
x (Interval.nth w (0::nat)) = (0::nat) ∧
(∀ i ≤ intlen w. x (Interval.nth w (i)) = i) ∧
x (Interval.nth w (intlen w)) = n) → (intlen w = n)
by auto
from 1 2 show ?thesis by auto
qed
qed

```

16.4 Example 4

```

locale Testrev =
fixes x :: state ⇒ nat
fixes F1 :: nat statefun ⇒ temporal
defines F1 ≡ (λ v. TEMP $v=#0 ∧ skip ∧ v:= $v+#1 )

lemma (in Testrev) testrev1:
(σ ⊨ F1 (x)) = (intlen σ = 1 ∧ (x (nth σ 0)) = 0 ∧ (x (nth σ 1)) = 1)
by (simp add: F1-def skip-defs next-assign-d-def next-val-d-def current-val-d-def, auto)

lemma (in Testrev) testrev2:
(σ ⊨ (F1 (x))r) = (intlen σ = 1 ∧ (x (nth σ 0)) = 1 ∧ (x (nth σ 1)) = 0)
proof -
have (σ ⊨ (F1 (x))r) = (σ ⊨ ($x=#0 ∧ skip ∧ x := $x+#1)r)
by (simp add: F1-def)
also have ... =
(σ ⊨ (($x=#0)r ∧ skipr ∧ (x := $x+#1)r))
by (simp add: all-rev-eq)
also have ... =
(σ ⊨ ((!x=#0) ∧ skip ∧ (x! = !x+#1)))
by (smt RRAnd all-rev-eq(1) all-rev-eq(10) all-rev-eq(11) all-rev-eq(12)
all-rev-eq(3) int-eq next-assign-d-def)
also have ... =
(σ ⊨ ((x$=#0) ∧ skip ∧ ($x = x$+#1)))
by (simp add: skip-defs next-val-d-def finval-defs penultval-defs current-val-d-def, auto)
also have ... =
(intlen σ = 1 ∧ (x (nth σ 0)) = 1 ∧ (x (nth σ 1)) = 0)

```

```

    by (simp add: skip-defs next-val-d-def current-val-d-def, auto)
  finally show  $(\sigma \models (F1(x))^r) = (\text{intlen } \sigma = 1 \wedge (x(\text{nth } \sigma 0)) = 1 \wedge (x(\text{nth } \sigma 1)) = 0)$  .
qed

```

16.5 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

17 Monitor Example

theory *MonitorExample*

imports

FOTheorems Monitor

begin

locale *Test* =

fixes *v* :: *state* \Rightarrow *nat*

fixes *y* :: *state* \Rightarrow *bool*

fixes *z* :: *state* \Rightarrow *nat*

fixes *F2* :: *nat* *statefun* \Rightarrow *temporal*

fixes *F3* :: *bool* *statefun* \Rightarrow *temporal*

fixes *F4* :: *nat* *statefun* \Rightarrow *temporal*

fixes *F5* :: *nat* *statefun* \Rightarrow *temporal*

fixes *Init2* :: *nat* *statefun* \Rightarrow *temporal*

fixes *Init3* :: *bool* *statefun* \Rightarrow *temporal*

fixes *Mon1* :: *state* *monitor*

fixes *Mon2* :: *state* *monitor*

fixes *Mon3* :: *state* *monitor*

fixes *Mon4* :: *state* *monitor*

fixes *Mon5* :: *state* *monitor*

fixes *Mon6* :: *state* *monitor*

defines *F2* $\equiv (\lambda v. \text{TEMP } \Box (\#0 \leq \$v))$

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 Init2  $\equiv (\lambda v. \text{TEMP } \$v = \#0)$ 
defines Init3  $\equiv (\lambda p. \text{TEMP } \$p)$ 
defines Mon1  $\equiv \text{FIRST}(F2\ v)$ 
defines Mon2  $\equiv \text{EMPTY UPTO Mon1}$ 
defines Mon3  $\equiv \text{Mon1 WITH } (F2\ v)$ 
defines Mon4  $\equiv \text{Mon2 THEN Mon1}$ 
defines Mon5  $\equiv \text{Mon3 THRU Mon4}$ 
defines 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 always-defs current-val-d-def)
  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 always-defs current-val-d-def)
  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:  $\vdash \mathcal{M}(\text{Mon3 THRU Mon4}) = \triangleright (di(\mathcal{M}(\text{Mon3})) \wedge di(\mathcal{M}(\text{Mon4})))$ 
    by fastforce
have 3:  $\vdash (di(\mathcal{M}(\text{Mon3})) \wedge di(\mathcal{M}(\text{Mon4}))) = (di(\text{empty}) \wedge di(\text{empty}))$ 
    using test3 test2 by (metis inteq-reflection lift-and-com)
hence 4:  $\vdash \triangleright (di(\mathcal{M}(\text{Mon3})) \wedge di(\mathcal{M}(\text{Mon4}))) = \triangleright (di(\text{empty}) \wedge di(\text{empty}))$ 
    by (simp add: FstEqvRule)
have 5:  $\vdash \triangleright (di(\text{empty}) \wedge di(\text{empty})) = \triangleright (di(\text{empty}))$ 
    by simp
have 6:  $\vdash \triangleright (di(\text{empty})) = \text{empty}$ 
    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:  $\vdash \mathcal{M}(\text{Mon6}) = (\mathcal{M}(\text{FIRST } F4 \text{ } z) \wedge (F5 \text{ } z))$ 
    using Mon6-def by fastforce
have 2:  $\vdash (\mathcal{M}(\text{FIRST } F4 \text{ } z) \wedge (F5 \text{ } z)) = (\triangleright (F4 \text{ } z) \wedge \text{fin}(\$z = \#4))$ 
    using F5-def by fastforce
have 3:  $\vdash (\triangleright (F4 \text{ } z) \wedge \text{fin}(\$z = \#4)) = (\triangleright (\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))$ 
    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:
 $(s \models (\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) =$ 
 $(z(\text{nth } s \text{ } 0) = 0 \wedge (\forall i < \text{intlen } s. z(\text{nth } s (\text{Suc } i)) = \text{Suc}(z(\text{nth } s i))) \wedge$ 
 $z(\text{nth } s (\text{intlen } s)) = 4)$ 

```

by (simp add: gets-defs fin-defs current-val-d-def sub-def)

lemma (in Test) test5-3:

$(z (nth\ s\ 0) = 0 \wedge (\forall\ i < intlen\ s. z (nth\ s\ (Suc\ i)) = Suc(z (nth\ s\ i))) \wedge$
 $z (nth\ s\ (intlen\ s)) = 4)$

\implies

$(z (nth\ s\ 0) = 0 \wedge (\forall\ i \leq intlen\ s. z (nth\ s\ i) = i)$
 $\wedge z (nth\ s\ (intlen\ s)) = 4)$

proof —

assume 0: $(z (nth\ s\ 0) = 0 \wedge (\forall\ i < intlen\ s. z (nth\ s\ (Suc\ i)) = Suc(z (nth\ s\ i))) \wedge$
 $z (nth\ s\ (intlen\ s)) = 4)$

show $(z (nth\ s\ 0) = 0 \wedge (\forall\ i \leq intlen\ s. z (nth\ s\ i) = i)$
 $\wedge z (nth\ s\ (intlen\ s)) = 4)$

proof —

have 1: $z (nth\ s\ 0) = 0$ **using** 0 **by** auto

have 2: $z (nth\ s\ (intlen\ s)) = 4$ **using** 0 **by** auto

have 3: $(\forall\ i \leq intlen\ s. z (nth\ s\ i) = i)$

proof

fix i

show $i \leq intlen\ s \longrightarrow z (Interval.nth\ s\ i) = i$

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:

$(z (nth\ s\ 0) = 0 \wedge (\forall\ i \leq intlen\ s. z (nth\ s\ i) = i)$

$\wedge z (nth\ s\ (intlen\ s)) = 4) \implies$

$(z (nth\ s\ 0) = 0 \wedge (\forall\ i < intlen\ s. z (nth\ s\ (Suc\ i)) = Suc(z (nth\ s\ i))) \wedge$
 $z (nth\ s\ (intlen\ s)) = 4)$

proof —

assume 0: $(z (nth\ s\ 0) = 0 \wedge (\forall\ i \leq intlen\ s. z (nth\ s\ i) = i)$

$\wedge z (nth\ s\ (intlen\ s)) = 4)$

show $(z (nth\ s\ 0) = 0 \wedge (\forall\ i < intlen\ s. z (nth\ s\ (Suc\ i)) = Suc(z (nth\ s\ i))) \wedge$
 $z (nth\ s\ (intlen\ s)) = 4)$

proof —

have 1: $z (nth\ s\ 0) = 0$ **using** 0 **by** auto

have 2: $z (nth\ s\ (intlen\ s)) = 4$ **using** 0 **by** auto

have 3: $(\forall\ i < intlen\ s. z (nth\ s\ (Suc\ i)) = Suc(z (nth\ s\ i)))$ **by** (simp add: 0)

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

qed
qed

lemma (in Test) test5-5:

$(z \text{ (nth } s \ 0) = 0 \wedge (\forall i < \text{intlen } s. z \text{ (nth } s \ (\text{Suc } i)) = \text{Suc}(z \text{ (nth } s \ i))) \wedge$
 $z \text{ (nth } s \ (\text{intlen } s)) = 4)$
=
 $(z \text{ (nth } s \ 0) = 0 \wedge (\forall i \leq \text{intlen } s. z \text{ (nth } s \ i) = i)$
 $\wedge z \text{ (nth } s \ (\text{intlen } s)) = 4)$

using test5-3 test5-4 **by** blast

lemma (in Test) test5-6 :

$(z \text{ (nth } s \ 0) = 0 \wedge (\forall i \leq \text{intlen } s. z \text{ (nth } s \ i) = i)$
 $\wedge z \text{ (nth } s \ (\text{intlen } s)) = 4) =$
 $(\text{intlen } s = 4 \wedge (\forall i \leq \text{intlen } s. z \text{ (nth } s \ i) = i))$

by auto

lemma (in Test) test5-7 :

$(s \models (\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) =$
 $(\text{intlen } s = 4 \wedge (\forall i \leq \text{intlen } s. z \text{ (nth } s \ i) = i))$

using test5-6 test5-5 test5-2 **by** fastforce

lemma (in Test) test5-8 :

$(s \models \Diamond((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))) =$
(
 ($(s \models (\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{intlen } s = 0) \vee$
 ($0 < \text{intlen } s \wedge (s \models \$z = \#0 \wedge z \text{ gets } \$z + \#1 \wedge \text{fin}(\$z = \#4)) \wedge$
 $(\forall ia < \text{intlen } s. (\text{prefix } ia \ s \models \neg((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))))$
)

using Fstsem[of TEMP $(\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)$]

by simp

lemma (in Test) test5-9 :

$\neg(s \models (\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)) \wedge \text{intlen } s = 0$

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
 $0 < \text{intlen } s \wedge$
 $(\forall ia < \text{intlen } s. (\text{prefix } ia \ s \models \neg((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))))$

proof —

assume 0: $s \models (\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4)$

show $0 < \text{intlen } s \wedge$

$(\forall ia < \text{intlen } s. (\text{prefix } ia \ s \models \neg((\$z = \#0 \wedge z \text{ gets } \$z + \#1) \wedge \text{fin}(\$z = \#4))))$

proof —

```

have 1: 0 < intlen s using test5-7 0 by simp
have 2: (∀ ia < intlen s. (prefix ia s ⊨ ¬(($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4))))
proof
  fix ia
  show ia < intlen s ⟶
    (prefix ia s ⊨ ¬(($z = #0 ∧ z gets $z + #1) ∧ fin ($z = #4)))
  proof -
    have 1: (prefix ia s ⊨ ¬(($z = #0 ∧ z gets $z + #1) ∧ fin ($z = #4))) =
      (¬(prefix ia s ⊨ (($z = #0 ∧ z gets $z + #1) ∧ fin ($z = #4))))
      by auto
    have 2: (prefix ia s ⊨ (($z = #0 ∧ z gets $z + #1) ∧ fin ($z = #4))) =
      (intlen (prefix ia s) = 4 ∧ (∀ i ≤ intlen(prefix ia s) . z (nth (prefix ia s) i) = i))
      using test5-7 by simp
    have 3: ia < intlen s ⟶ ¬(intlen (prefix ia s) = 4 ∧
      (∀ i ≤ intlen(prefix ia s) . z (nth (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 ⊨ ▷(($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4))) =
(s ⊨ ($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4))
using test5-8 test5-9 test5-10 by fastforce

```

lemma (in Test) test5-12 :

```

⊢ ▷(($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4)) = (($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4))
using test5-11 by (simp add: Valid-def)

```

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. A Hierarchical Completeness Proof for Propositional Interval Temporal Logic with Finite Time. *Journal of Applied Non-Classical Logics*, 14(1–2):55–104, 2004.

- [5] B. Moszkowski. Compositional reasoning using intervals and time reversal. *Annals of Mathematics and Artificial Intelligence*, 71(1-3):175–250, 2014.
- [6] B. C. Moszkowski. Imperative reasoning in interval temporal logic. Technical report, University of Newcastle upon Tyne, 1996.
- [7] B. C. Moszkowski. A Complete Axiom System for Propositional Interval Temporal Logic with Infinite Time. *Logical Methods in Computer Science Journal*, 8(3), 2012.
- [8] 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.