An encoding of Interval Temporal Logic in Isabelle/HOL

Antonio Cau   Ben Moszkowski   David Smallwood

December 8, 2018

Abstract

These Isabelle theories introduce the semantics and syntax of Interval Temporal Logic (ITL). The ITL proof system, as introduced in [5], has been encoded and its soundness has been checked. The encoding is shallow using the Intensional Logic technique of [4]. An extensive library of ITL theorems, taken from [6], has been checked. Furthermore we provide examples of using quantification over both static (rigid) and state (flexible) variables.

Contents

theory *Interval*
 imports
  Main
begin

1 Intervals

An interval is a 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*.

In order to define the semantics of the ITL chopstar we introduce *index-sequence* which is a sequence of chop (fuse) points. This sequence is again of type interval but the elements are natural numbers. Two functions *shift* and *shiftm* 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 *'a interval* =
  St *'a* (⌈-⌉)
 | *Cons* *'a* *'a interval* (**infixr** ⊙ *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*⊙⟨*xs*⟩
 ⟨*x*⟩ == ⌈*x*⌉

primrec (*nonexhaustive*) *intlen* :: *'a interval* ⇒ *nat* **where**

intlen (*St x*) = *0*
| *intlen* (*x*⊙*xs*) = *1*+ (*intlen xs*)

primrec (*nonexhaustive*) *nth* :: *'a interval* => *nat* => *'a* **where**
nth (*St x*) *n* = *x*
| *nth* (*Cons x xs*) *n* = (*case n of 0* ⇒ *x* | *Suc k* ⇒ *nth xs k*)

primrec *prefix*:: *nat* ⇒ *'a interval* ⇒ *'a interval* **where**
prefix n (*St x*) = (*St x*)
| *prefix n* (*Cons x xs*) = (*case n of 0* ⇒ (*St x*) | *Suc m* ⇒ (*Cons x* (*prefix m xs*)))

primrec *suffix*:: *nat* ⇒ *'a interval* ⇒ *'a interval* **where**
suffix n (*St x*) = (*St x*)
| *suffix n* (*Cons x xs*) = (*case n of 0* ⇒ (*Cons x xs*) | *Suc m* ⇒ *suffix m xs*)

definition *sub*:: *nat* ⇒ *nat* ⇒ *'a interval* ⇒ *'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* ⇒ *'a* **where**
intfirst (*St x*) = *x*
| *intfirst* (*Cons x* -) = *x*

primrec *intlast* :: *'a interval* ⇒ *'a* **where**
intlast (*St x*) = *x*
| *intlast* (*Cons* - *xs*) = *intlast xs*

primrec *intapp* :: *'a interval* ⇒ *'a interval* ⇒ *'a interval* (**infixr** ⊖ *65*) **where**
intapp-St: (*St x*) ⊖ *ys* = *x* ⊙ *ys* |
intapp-Cons: (*x*⊙*xs*) ⊖ *ys* = *x* ⊙ (*xs* ⊖ *ys*)

primrec *intrev* :: *'a interval* ⇒ *'a interval* **where**
intrev (*St x*) = (*St x*)
| *intrev* (*Cons x xs*) = (*intrev xs*) ⊖ (*St x*)

definition *index-sequence* :: *nat* ⇒ *index* ⇒ *bool* **where**
index-sequence x idx ≡ (*nth idx 0* = *x*) ∧ (∀ *n*. *n*<*intlen idx* ⟶ *nth idx n* < *nth idx* (*Suc n*))

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

definition *shiftm* :: *nat* ⇒ *nat* ⇒ *nat* **where**
shiftm 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-st* :
  $intlen\ (St\ x) = 0$
by *simp*

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

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

lemma *interval-intlen-map*:
  $intlen\ (map\ f\ xs) = intlen\ xs$
by (*induct xs*) *simp-all*

1.2.2 nth

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

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

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

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

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

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

lemma *interval-st-intlen* :

$(xs = (St\ x)) \longleftrightarrow intlen\ xs = 0 \wedge nth\ xs\ 0 = x$
by (*induct xs*) *simp-all*

lemma *interval-eq-nth-eq* :
$(xs = ys) = (intlen\ xs = intlen\ ys \wedge (\forall\ i \leq intlen\ xs.\ nth\ xs\ i = nth\ ys\ i))$
apply (*induct xs arbitrary*: *ys*)
apply (*metis interval-st-intlen le-numeral-extra*(*3*))
apply (*case-tac ys*, *simp*)
by *fastforce*

lemma *interval-nth-map* :
$nth\ (map\ f\ xs)\ i = f\ (nth\ xs\ i)$
apply (*induct xs arbitrary*: *i*, *simp*)
apply (*case-tac i*, *simp*, *simp*)
done

1.2.3 index sequence

lemma *interval-idx-less*:
assumes *iseq*: *index-sequence x idx*
shows $(n < intlen\ idx \wedge n+k < intlen\ idx) \longrightarrow nth\ idx\ n < nth\ idx\ (Suc(n+k))$
apply (*induct k*)
using *index-sequence-def iseq* **apply** *auto*[*1*]
using *index-sequence-def iseq* **by** *auto*

lemma *interval-idx-less-last* :
assumes *index-sequence x idx*
shows $(i < intlen\ idx \wedge i+(intlen\ idx - (i+1)) < intlen\ idx)$
$\longrightarrow nth\ idx\ i < nth\ idx\ (Suc(i+(intlen\ idx - (i+1))))$
using *assms interval-idx-less* **by** *blast*

lemma *interval-idx-less-last-1*:
assumes *index-sequence x idx*
shows $i < intlen\ idx \longrightarrow nth\ idx\ i < nth\ idx\ (intlen\ idx)$
using *assms interval-idx-less-last* **by** *auto*

lemma *interval-idx-greater-first*:
assumes *index-sequence x idx*
shows $(i > 0 \wedge i \leq intlen\ idx) \longrightarrow x < nth\ idx\ i$
apply (*induct i*, *simp*)
using *assms*
by (*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*:
$index\text{-}sequence\ 0\ (x \odot ls) =$
$(x=0 \wedge x < nth\ ls\ 0 \wedge index\text{-}sequence\ (nth\ ls\ 0)\ ls)$
apply (*simp add*: *index-sequence-def*)
using *less-Suc-eq-0-disj* **by** *auto*

lemma *interval-idx-shift-mono*:
  mono (*shift k*)
by (*simp add*: *Interval.shift-def mono-def*)

lemma *interval-idx-expand*:
 index-sequence 0 l $\wedge$ (*nth l* (*intlen l*)) = (*intlen xs*) $\wedge$ *0*$\leq$*i* $\wedge$ *i*< (*intlen l*)
 $\Longrightarrow$ *0*$\leq$ (*nth l i*) $\wedge$ (*nth l i*) $\leq$ (*nth l* (*i*+*1*)) $\wedge$ (*nth l* (*i*+*1*)) $\leq$ (*intlen xs*)
apply (*simp add*: *index-sequence-def*)
apply (*induct l*, *simp*)
by (*metis Suc-lessI eq-imp-le index-sequence-def interval-idx-less-last-1 less-imp-le-nat*)

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

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

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

lemma *interval-idx-link*:
  (*lsk* = *map* (*shift k*) *ls* $\wedge$ *index-sequence 0* (*ls*)) =
  (*lsk* = *map* (*shift k*) *ls* $\wedge$ *index-sequence k* (*lsk*) $\wedge$ *index-sequence 0* (*ls*)$\wedge$
  (*intlen ls*) =(*intlen lsk*))
by (*metis Interval.shift-def add-diff-cancel-left′ diff-diff-cancel diff-is-0-eq′*
  interval-idx-shift-idx interval-idx-shift-mono interval-intlen-map le-numeral-extra(*3*) *mono-def*)

lemma *interval-idx-bound-0* :
 assumes *index-sequence 0 ls* $\wedge$ *Interval.nth ls* (*intlen ls*) = *intlen* (*suffix k xs*)
 shows ((*i*$\leq$*intlen ls*) $\longrightarrow$ ((*nth ls* (*i*)) $\leq$ (*intlen* (*suffix k xs*))))
using *assms*
by (*metis add.commute add-eq-if eq-iff interval-idx-less le-add-diff-inverse2*
  le-neq-implies-less lessI less-imp-le-nat)

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

1.2.4 prefix, suffix and sub

lemma *interval-prefix-state* [*simp*]:
$prefix\ m\ (St\ x) = (St\ x)$
by *simp*

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

lemma *interval-prefix-zero* [*simp*]:
$prefix\ 0\ (x \odot xs) = St\ x$
by *auto*

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

lemma *interval-intfirst-prefix* [*simp*]:
$i \le intlen\ xs \Longrightarrow\ \ intfirst\ (prefix\ i\ xs) = intfirst\ xs$
by (*induct xs arbitrary*: *i*, *auto*) (*case-tac i*, *auto*)

lemma *interval-prefix-intlen* [*simp*]:
$(prefix\ (intlen\ xs)\ xs) = xs$
by (*induct xs*) *simp-all*

lemma *interval-prefix-intlen-gr-1* [*simp*]:
$(prefix\ ((intlen\ xs)+i)\ xs) = xs$
by (*induct xs*) *simp-all*

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

lemma *interval-prefix-length* :
$intlen\ (prefix\ i\ xs) = (if\ i \le\ intlen\ xs\ then\ i\ else\ intlen\ xs)$
by (*induct xs arbitrary*: *i*, *simp*) (*case-tac i*, *auto*)

lemma *interval-prefix-length-good* [*simp*]:
assumes $i \le\ intlen\ xs$
shows $(intlen\ (\ prefix\ i\ xs)) = i$
using *assms* **by** (*simp add*: *interval-prefix-length*)

```
lemma interval-prefix-length-bad [simp] :
  assumes    i > intlen xs
  shows      intlen (prefix i xs) = intlen xs
using assms by (simp add: interval-prefix-length)

lemma interval-pref-intlen-bound :
 assumes    i ≤ (intlen xs)
 shows      intlen (prefix i xs) ≤ intlen xs
using assms by (induct xs, simp) (metis interval-prefix-length)

lemma interval-suffix-length:
     intlen (suffix i xs ) = (if i≤ intlen xs then (intlen xs)−i else 0)
by (induct xs arbitrary: i, simp) (case-tac i, auto)

lemma interval-suffix-length-good [simp]:
   assumes  i≤ intlen xs
   shows    intlen (suffix i xs ) = (intlen xs)−i
using assms by (simp add: interval-suffix-length)

lemma interval-suffix-length-bad [simp]:
  assumes   i> intlen xs
  shows     intlen (suffix i xs ) = 0
using assms by (simp add: interval-suffix-length)

lemma interval-nth-prefix [simp]:
  i≤intlen xs ∧ k ≤ i ⟹ nth (prefix i xs) k = nth xs k
apply (induct xs arbitrary: i k, auto)
apply (case-tac i, auto)
apply (case-tac k, auto)
done

lemma interval-nth-suffix [simp]:
  i≤intlen xs ∧ k ≤ intlen xs −i ⟹ nth (suffix i xs) k = nth xs (i+k)
by (induct xs arbitrary: i k, auto) (case-tac i, auto)

lemma interval-suffix-prefix-help-1:
 assumes  ia+i ≤ intlen xs ∧ k ≤ ia
 shows  nth (prefix ia (suffix i xs)) k = nth (suffix i (prefix (ia +i) xs)) k
proof −
 have 1:  nth (prefix ia (suffix i xs)) k = nth (suffix i xs) k
 using interval-nth-prefix assms by (metis interval-prefix-intlen-gr-1 le-cases le-iff-add)
 have 2:  nth (suffix i xs) k = nth xs (i+k)
 using interval-nth-suffix assms by (simp add: add-le-imp-le-diff)
 have 3:  nth xs (i+k) = nth (prefix (ia+i) xs) (i+k)
 using interval-nth-prefix assms by simp
 have 4:  nth (prefix (ia+i) xs) (i+k) = nth (suffix i (prefix (ia +i) xs)) 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* ≤ *intlen xs*
 shows (∀ *k* ≤ *ia* . *nth* (*prefix ia* (*suffix i xs*)) *k* = *nth* (*suffix i* (*prefix* (*ia* +*i*) *xs*)) *k*)
using *interval-suffix-prefix-help-1* **using** *assms* **by** *fastforce*

lemma *interval-suffix-prefix-help-3*:
 assumes *ia*+*i* ≤ *intlen xs*
 shows *intlen* (*prefix ia* (*suffix i xs*)) = *intlen* (*suffix i* (*prefix* (*ia* +*i*) *xs*))
using *assms interval-prefix-length-good interval-suffix-length-good* **by** *auto*

lemma *interval-suffix-prefix-swap*:
 assumes *ia*+*i* ≤ *intlen xs*
 shows *prefix ia* (*suffix i xs*) = *suffix i* (*prefix* (*ia* +*i*) *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*]:
 prefix 0 (*prefix 0 xs*) = *prefix 0 xs*
by (*induct xs*) *simp-all*

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

lemma *interval-pref-help*:
 assumes *i*≤*intlen* (*prefix* (*intlen xs* − *Suc 0*) *xs*)
 shows (*prefix i* (*prefix* (*intlen xs* − *Suc 0*) *xs*)) = (*prefix i 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 *intlen xs* >*0* ∧ *ia*<*intlen* (*xs*)
 shows (*prefix ia* (*prefix* (*intlen xs* − *Suc 0*) *xs*)) = (*prefix ia xs*)
using *assms*
by (*metis Suc-leI Suc-le-mono Suc-pred diff-le-self interval-pref-help interval-prefix-length-good*)

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

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

suffix (*Suc m*) (*x* ⊙ *xs*) = *suffix m xs*
by *auto*

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

lemma *interval-suffix-intlen* [*simp*]:
suffix (*intlen xs*) *xs* = (*St* (*nth xs* (*intlen xs*)))
by (*induct xs*) *simp-all*

lemma *interval-suffix-intlast* [*simp*]:
suffix (*intlen xs*) *xs* = *St* (*intlast xs*)
by (*induct xs*) *simp-all*

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

lemma *interval-prefix-suffix-intlen*:
intlen (*prefix ia* (*suffix i xs*)) =
(*if i* ≤ *intlen xs then*
(*if ia*≤ *intlen xs* −*i* *then ia else* (*intlen xs*) −*i*)
else 0)
by (*metis interval-prefix-length interval-suffix-length le-zero-eq*)

lemma *interval-prefix-suffix-intlen-good* [*simp*]:
assumes *ia*≤ *intlen xs* −*i* ∧ *i* ≤ *intlen xs*
shows *intlen* (*prefix ia* (*suffix i xs*)) = *ia*
using *assms* **by** (*simp add*: *interval-prefix-suffix-intlen*)

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

lemma *interval-prefix-suffix-intlen-bad-1* [*simp*] :
assumes *i* ≤ *intlen xs* ∧ *ia* > *intlen xs* −*i*
shows *intlen* (*prefix ia* (*suffix i xs*)) = (*intlen xs*) −*i*
using *assms* **by** (*simp add*: *interval-prefix-suffix-intlen*)

lemma *interval-suffix-suffix-3*:
assumes *i*>*0* ∧ *ia*<*i* ∧ *i*≤ *intlen xs*
shows (*suffix* (*i*−*ia*) (*suffix* ((*intlen xs*)−*i*) *xs*)) = (*suffix* (((*intlen xs*)−*ia*)) *xs*)
using *assms* **by** *simp*

lemma *interval-sub-zero-prefix* :
sub 0 k xs = *prefix k xs*

by (*simp add*: *Interval.sub-def*)

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

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

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

1.2.5 Reverse

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

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

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

lemma *interval-suffix-intapp2* [*simp*]:
 $suffix\ (intlen\ xs - k)\ (\ xs) \ominus ys = suffix\ (intlen\ xs - k)\ (\ xs \ominus ys)$
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*:
  $nth\ (xs \ominus ys)\ k = (if\ k \leq intlen\ xs$
    $then\ (nth\ xs\ k)$
    $else\ (nth\ ys\ (k - (intlen\ xs) - 1))\ )$
apply (*induct xs arbitrary*: *k*)
apply (*case-tac k*, *simp*, *simp*)
apply (*case-tac k*, *simp*, *simp*)
done

lemma *interval-rev-intapp* [*simp*]:
intrev (*xs* ⊖ *ys*) = (*intrev ys*) ⊖ (*intrev xs*)
by (*induct xs*) *simp-all*

lemma *interval-rev-rev-ident* [*simp*]:
intrev (*intrev xs*) = *xs*
by (*induct xs*) *auto*

lemma *interval-rev-swap* :
((*intrev xs*) = *ys*) = (*xs* = *intrev ys*)
by *auto*

lemma *interval-intlast-intrev*:
intlast (*intrev xs*) = *intfirst xs*
by (*induct xs*, *auto*)
(*metis Suc-eq-plus1 add.right-neutral interval.inject*(*1*) *interval-intlen-intapp*
interval-intlen-st interval-suffix-intapp interval-suffix-intlast)

lemma *interval-intfirst-intrev*:
intfirst (*intrev xs*) = *intlast xs*
by (*induct xs*, *auto*)
(*metis intapp-St interval-intlast-intrev interval-rev-intapp intlast.simps*(*2*) *intrev.simps*(*1*))

lemma *interval-intrev-nth*:
k≤ *intlen* (*intrev xs*) ⟹ (*nth* (*intrev xs*) *k*) = (*nth* *xs* ((*intlen xs*) −*k*))
apply (*induct xs*, *simp*)
apply *simp*
apply (*case-tac k*)
apply (*simp add*: *interval-intapp-nth*)
by (*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*:
k≤ *intlen xs* ⟹ *intrev*(*prefix k xs*) = *suffix* ((*intlen xs*) − *k*) (*intrev xs*)
apply (*induct xs arbitrary*: *k*, *simp*)
apply *simp*
apply (*case-tac k*)
apply (*metis diff-zero interval-intrev-intlen interval-suffix-intapp intrev.simps*(*1*) *old.nat.simps*(*4*))
by (*metis Suc-le-mono diff-Suc-Suc interval-intrev-intlen interval-suffix-intapp2*
intrev.simps(*2*) *old.nat.simps*(*5*))

lemma *interval-intrev-suffix*:
k≤ *intlen xs* ⟹ *intrev*(*suffix k xs*) = *prefix* ((*intlen xs*) − *k*) (*intrev xs*)
by (*induct xs arbitrary*: *k*, *simp*, *simp add*: *interval-intrev-prefix interval-rev-swap*)

lemma *interval-intrev-sub1*:
assumes *0* ≤ *i* ∧ *i*≤*j* ∧ *j* ≤ *intlen xs*
shows *intrev* (*sub i j xs*) = *intrev* (*prefix* (*j*−*i*) (*suffix i xs*))
using *assms interval-sub-prefix-suffix* **by** (*simp add*: *interval-sub-prefix-suffix*)

lemma *interval-intrev-sub2*:
assumes $0 \le i \wedge i \le j \wedge j \le intlen\ xs$
shows $intrev\ (prefix\ (j-i)\ (suffix\ i\ xs)) = suffix\ ((intlen\ xs) - j)\ (intrev\ (suffix\ i\ xs))$
using *assms interval-intrev-prefix*[*of j−i suffix i xs*] **by** *auto*

lemma *interval-intrev-sub3*:
assumes $0 \le i \wedge i \le j \wedge j \le intlen\ xs$
shows $suffix\ ((intlen\ xs) - j)\ (intrev\ (suffix\ i\ xs)) =$
$suffix\ ((intlen\ xs) - j)\ (prefix\ ((intlen\ xs) - i)\ (intrev\ xs))$
using *assms interval-intrev-suffix*[*of i xs*] **by** *auto*

lemma *interval-intrev-sub4*:
assumes $0 \le i \wedge i \le j \wedge j \le intlen\ xs$
shows $suffix\ ((intlen\ xs) - j)\ (prefix\ ((intlen\ xs) - i)\ (intrev\ xs)) =$
$sub\ ((intlen\ xs) - j)\ ((intlen\ xs) - i)\ (intrev\ xs)$
using *assms* **by** (*simp add*: *diff-le-mono2 interval-sub-prefix-suffix interval-suffix-prefix-swap*)

lemma *interval-intrev-sub*:
assumes $0 \le i \wedge i \le j \wedge j \le intlen\ xs$
shows $intrev\ (sub\ i\ j\ xs) = sub\ ((intlen\ xs) - j)\ ((intlen\ xs) - i)\ (intrev\ xs)$
using *assms*
by (*simp add*: *interval-intrev-sub1 interval-intrev-sub2 interval-intrev-sub3 interval-intrev-sub4*)

lemma *interval-intrev-idx-2*:
assumes $index\text{-}sequence\ 0\ l\ \wedge (nth\ l\ (intlen\ l)) = (intlen\ xs) \wedge$
$0 \le i \wedge i < (intlen\ l)$
shows $(intrev\ (sub\ (nth\ l\ i)\ (nth\ l\ (i+1))\ xs)) =$
$(\ (sub\ ((intlen\ xs) - (nth\ l\ (i+1)))\ ((intlen\ xs) - (nth\ l\ i))\ (intrev\ xs)))$
using *assms interval-idx-expand interval-intrev-sub*[*of* (*nth l i*) (*nth l* (*i*+*1*)) *xs*]
by *blast*

lemma *interval-intrev-idx-3*:
assumes $index\text{-}sequence\ 0\ l\ \wedge (nth\ l\ (intlen\ l)) = (intlen\ xs) \wedge$
$ls = map\ (\lambda\ x.\ (intlen\ xs) - x)\ (intrev\ l)$
shows $(nth\ ls\ 0) = 0 \wedge (nth\ ls\ (intlen\ ls)) = (intlen\ xs) \wedge intlen\ ls = intlen\ l$
using *assms*
by (*metis diff-self-eq-0 diff-zero index-sequence-def interval-intfirst-intrev*
interval-intlast-intrev interval-intlen-map interval-intrev-intlen
interval-nth-intlen-intlast interval-nth-map interval-nth-zero-intfirst)

lemma *interval-intrev-idx-4*:
$index\text{-}sequence\ 0\ l\ \wedge (nth\ l\ (intlen\ l)) = (intlen\ xs) \wedge$
$ls = map\ (\lambda\ x.\ (intlen\ xs) - x)\ (intrev\ l)$
$\Longrightarrow i \le intlen\ ls \longrightarrow (nth\ ls\ i) = (intlen\ xs) - (nth\ l\ ((intlen\ l) - i))$
apply (*induct ls*)
apply (*metis diff-zero interval-intlen-st interval-intrev-idx-3 le-0-eq*)
by (*simp add*: *interval-intlen-map interval-intrev-nth interval-nth-map*)

lemma *interval-intrev-idx-5*:
assumes $(index\text{-}sequence\ 0\ l\ \wedge (nth\ l\ (intlen\ l)) = (intlen\ xs))$

```
 shows   (i< intlen l ⟶
         (intlen xs) − (nth l ((intlen l)−i)) < (intlen xs) − (nth l ((intlen l)−(i+1))))
using assms
by (smt Suc-diff-Suc Suc-eq-plus1 add-gr-0 add-less-cancel-left diff-less
      index-sequence-def le-add-diff-inverse2 le-numeral-extra(3) less-diff-conv
      less-imp-le-nat not-gr-zero interval-idx-expand)

lemma interval-intrev-idx-6:
 assumes (index-sequence 0 l ∧ (nth l (intlen l)) = (intlen xs) ∧
         ls = map (λ x. (intlen xs) −x) (intrev l))
 shows   (i< intlen ls ⟶
         ((nth ls i) = (intlen xs) − (nth l ((intlen l)−i)) ∧
         (nth ls (i+1)) = (intlen xs) − (nth l ((intlen l)−(i+1))) ∧
         (nth ls i) < (nth ls (i+1))))
proof −
 have 1: (i< intlen ls ⟶ (nth ls i) = (intlen xs) − (nth l ((intlen l)−i)))
        using assms interval-intrev-idx-4 less-imp-le-nat by blast
 have 2: (i< intlen ls ⟶ (nth ls (i+1)) = (intlen xs) − (nth l ((intlen l)−(i+1))))
        using assms by (simp add: interval-intrev-idx-4)
 have 3: ( i< intlen ls ⟶
         ((nth ls i) = (intlen xs) − (nth l ((intlen l)−i)) ∧
         (nth ls (i+1)) = (intlen xs) − (nth l ((intlen l)−(i+1)))))
        using 1 2 by auto
 have 4: (i< intlen ls ⟶
           ((nth ls i) = (intlen xs) − (nth l ((intlen l)−i)) ∧
           (nth ls (i+1)) = (intlen xs) − (nth l ((intlen l)−(i+1))) ∧
           (nth ls i) < (nth ls (i+1))))
         using assms 3 index-sequence-def interval-intrev-idx-5
        by (metis interval-intlen-map interval-intrev-intlen)
 from 4 show ?thesis by blast
qed

lemma interval-intrev-idx-7:
 assumes (index-sequence 0 l ∧ (nth l (intlen l)) = (intlen xs) ∧
         ls = map (λ x. (intlen xs) −x) (intrev l))
   shows index-sequence 0 ls
using assms interval-intrev-idx-6 interval-intrev-idx-3
by (metis Suc-eq-plus1 index-sequence-def)

lemma interval-intrev-idx-8:
 assumes index-sequence 0 l ∧ (nth l (intlen l)) = (intlen xs) ∧
        ls = map (λ x. (intlen xs) −x) (intrev l) ∧ index-sequence 0 ls
 shows   i<intlen ls ⟶
        (intlen xs)− (nth l (i+1)) = nth ls ((intlen ls)−(i+1)) ∧
        (intlen xs) − (nth l i)    = (nth ls ((intlen ls) −i))
using assms interval-intrev-idx-4
by (smt Suc-eq-plus1 Suc-leI add-diff-cancel-right' assms diff-diff-cancel diff-diff-left
      diff-le-self interval-intrev-idx-3)

lemma interval-intrev-idx-9:
```

assumes *index-sequence 0 l* $\wedge$ (*nth l* (*intlen l*)) = (*intlen xs*) $\wedge$
ls = *map* (λ *x*. (*intlen xs*) $-x$) (*intrev l*) $\wedge$ *index-sequence 0 ls*
shows *i*<*intlen ls* $\longrightarrow$
sub ((*intlen xs*)$-$ (*nth l* (*i*+*1*))) ((*intlen xs*) $-$ (*nth l i*)) (*intrev xs*) =
sub (*nth ls* ((*intlen ls*)$-$(*i*+*1*))) ((*nth ls* ((*intlen ls*) $-i$))) (*intrev xs*)

using *interval-intrev-idx-8* **using** *assms* **by** *fastforce*

lemma *interval-intrev-idx-11*:
assumes (*index-sequence 0 l* $\wedge$ (*nth l* (*intlen l*)) = (*intlen xs*))
shows *i*$\leq$*intlen l* $\longrightarrow$
(*nth l i*) = (*nth* (*map* (λ *x*.(*intlen xs*)$-x$) (*intrev* (*map* (λ *x*.(*intlen xs*)$-x$) (*intrev l*)))) *i*)
using *assms index-sequence-def*
by (*smt diff-diff-cancel diff-is-0-eq diff-less diff-zero leD le-cases not-gr-zero*
interval-intrev-idx-3 interval-intrev-idx-6 interval-intrev-idx-7)

lemma *interval-intrev-idx-12*:
assumes (*index-sequence 0 l* $\wedge$ (*nth l* (*intlen l*)) = (*intlen xs*))
shows *l* = *map* (λ *x*. (*intlen xs*) $-x$) (*intrev* (*map* (λ *x*. (*intlen xs*) $-x$) (*intrev l*)))
using *assms interval-intrev-idx-11*
by (*simp add*: *interval-intrev-idx-11 interval-eq-nth-eq interval-intlen-map*)

end

2 Representing Intensional Logic

theory *Intensional*
imports *Main*
begin

In higher-order logic, every proof rule has a corresponding tautology, i.e. the *deduction theorem* holds. Isabelle/HOL implements this since object-level implication ($\longrightarrow$) and meta-level entailment ($\Longrightarrow$) commute, viz. the proof rule *impI*: ($?P \Longrightarrow ?Q$) $\Longrightarrow$ $?P \longrightarrow ?Q$. However, the deduction theorem does not hold for most modal and temporal logics [3, page 95][4]. For example $A \vdash \Box A$ holds, meaning that if A holds in any world, then it always holds. However, $\vdash A \longrightarrow \Box A$, stating that A always holds if it initially holds, is not valid.

Merz [4] overcame this problem by creating an *Intensional* logic. It exploits Isabelle's axiomatic type class feature [7] by creating a type class *world*, which provides Skolem constants to associate formulas with the world they hold in. The class is trivial, not requiring any axioms.

class *world*

world is a type class of possible worlds. It is a subclass of all HOL types *type*. No axioms are provided, since its only purpose is to avoid silly use of the *Intensional* syntax.

2.1 Abstract Syntax and Definitions

type-synonym ($'w$,$'a$) *expr* = $'w \Rightarrow 'a$
type-synonym $'w$ *form* = ($'w$, *bool*) *expr*

The intention is that $'a$ will be used for unlifted types (class *type*), while $'w$ is lifted (class *world*).

definition *Valid* :: $('w{::}world)$ *form* $\Rightarrow$ *bool*
  where *Valid* $A \equiv \forall\, w.\ A\ w$

definition *const* :: $'a \Rightarrow ('w{::}world,\ 'a)$ *expr*
  where *unl-con*: *const c w* $\equiv$ *c*

definition *lift* :: $['a \Rightarrow 'b,\ ('w{::}world,\ 'a)\ expr] \Rightarrow ('w,'b)\ expr$
  where *unl-lift*: *lift f x w* $\equiv f\ (x\ w)$

definition *lift2* :: $['a \Rightarrow 'b \Rightarrow 'c,\ ('w{::}world,'a)\ expr,\ ('w,'b)\ expr] \Rightarrow ('w,'c)\ expr$
  where *unl-lift2*: *lift2 f x y w* $\equiv f\ (x\ w)\ (y\ w)$

definition *lift3* :: $['a \Rightarrow 'b => 'c \Rightarrow 'd,\ ('w{::}world,'a)\ expr,\ ('w,'b)\ expr,\ ('w,'c)\ expr] \Rightarrow ('w,'d)\ expr$
  where *unl-lift3*: *lift3 f x y z w* $\equiv f\ (x\ w)\ (y\ w)\ (z\ w)$

definition *lift4* :: $['a \Rightarrow 'b => 'c \Rightarrow 'd \Rightarrow 'e,\ ('w{::}world,'a)\ expr,\ ('w,'b)\ expr,\ ('w,'c)\ expr,('w,'d)\ expr] \Rightarrow ('w,'e)\ expr$
  where *unl-lift4*: *lift4 f x y z zz w* $\equiv f\ (x\ w)\ (y\ w)\ (z\ w)\ (zz\ w)$

Valid F asserts that the lifted formula *F* holds everywhere. *const* allows lifting of a constant, while *lift* through *lift4* allow functions with arity 1–4 to be lifted. (Note that there is no way to define a generic lifting operator for functions of arbitrary arity.)

definition *RAll* :: $('a \Rightarrow ('w{::}world)\ form) \Rightarrow 'w\ form$ (**binder** *Rall* *10*)
  where *unl-Rall*: $(Rall\ x.\ A\ x)\ w \equiv \forall\, x.\ A\ x\ w$

definition *REx* :: $('a \Rightarrow ('w{::}world)\ form) \Rightarrow 'w\ form$ (**binder** *Rex* *10*)
  where *unl-Rex*: $(Rex\ x.\ A\ x)\ w \equiv \exists\, x.\ A\ x\ w$

definition *REx1* :: $('a \Rightarrow ('w{::}world)\ form) \Rightarrow 'w\ form$ (**binder** *Rex!* *10*)
  where *unl-Rex1*: $(Rex!\ x.\ A\ x)\ w \equiv \exists!x.\ A\ x\ w$

RAll, *REx* and *REx1* introduces "rigid" quantification over values (of non-world types) within "intensional" formulas. *RAll* is universal quantification, *REx* is existential quantifcation. *REx1* requires unique existence.

We declare the "unlifting rules" as rewrite rules that will be applied automatically.

lemmas *intensional-rews*[*simp*] =
  unl-con unl-lift unl-lift2 unl-lift3 unl-lift4
  unl-Rall unl-Rex unl-Rex1

2.2 Concrete Syntax

nonterminal
  lift **and** *liftargs*

The non-terminal *lift* represents lifted expressions. The idea is to use Isabelle's macro mechanism to convert between the concrete and abstract syntax.

syntax
  :: *id* $\Rightarrow$ *lift* (-)
  :: *longid* $\Rightarrow$ *lift* (-)

```
           :: var ⇒ lift                         (-)
-applC      :: [lift, cargs] ⇒ lift              ((1-/ -) [1000, 1000] 999)
           :: lift ⇒ lift                        ('(-'))
-lambda     :: [idts, 'a] ⇒ lift                 ((3%-./ -) [0, 3] 3)
-constrain  :: [lift, type] ⇒ lift               ((-::-) [4, 0] 3)
           :: lift ⇒ liftargs                    (-)
-liftargs   :: [lift, liftargs] ⇒ liftargs       (-,/ -)
-Valid      :: lift ⇒ bool                       ((⊢ -) 5)
-holdsAt    :: ['a, lift] ⇒ bool                 ((- ⊨ -) [100,10] 10)


LIFT        :: lift ⇒ 'a                         (LIFT -)


-const      :: 'a ⇒ lift                         ((#-) [1000] 999)
-lift       :: ['a, lift] ⇒ lift                 ((-<->) [1000] 999)
-lift2      :: ['a, lift, lift] ⇒ lift           ((-<-,/ ->) [1000] 999)
-lift3      :: ['a, lift, lift, lift] ⇒ lift     ((-<-,/ -,/ ->) [1000] 999)
-lift4      :: ['a, lift, lift, lift,lift] ⇒ lift      ((-<-,/ -,/ -,/ ->) [1000] 999)


-liftEqu    :: [lift, lift] ⇒ lift               ((- =/ -) [50,51] 50)
-liftNeq    :: [lift, lift] ⇒ lift               (infixl ≠ 50)
-liftNot    :: lift ⇒ lift                       (¬ - [90] 90)
-liftAnd    :: [lift, lift] ⇒ lift               (infixr ∧ 35)
-liftOr     :: [lift, lift] ⇒ lift               (infixr ∨ 30)
-liftImp    :: [lift, lift] ⇒ lift               (infixr ⟶ 25)
-liftIf     :: [lift, lift, lift] ⇒ lift         ((if (-)/ then (-)/ else (-)) 10)
-liftPlus   :: [lift, lift] ⇒ lift               ((- +/ -) [66,65] 65)
-liftMinus  :: [lift, lift] ⇒ lift               ((- −/ -) [66,65] 65)
-liftTimes  :: [lift, lift] ⇒ lift               ((- ∗/ -) [71,70] 70)
-liftDiv    :: [lift, lift] ⇒ lift               ((- div -) [71,70] 70)
-liftMod    :: [lift, lift] ⇒ lift               ((- mod -) [71,70] 70)
-liftLess   :: [lift, lift] ⇒ lift               ((-/ < -)  [50, 51] 50)
-liftLeq    :: [lift, lift] ⇒ lift               ((-/ ≤ -) [50, 51] 50)
-liftMem    :: [lift, lift] ⇒ lift               ((-/ ∈ -) [50, 51] 50)
-liftNotMem :: [lift, lift] ⇒ lift               ((-/ ∉ -) [50, 51] 50)
-liftFinset :: liftargs => lift                  ({(-)})

-liftPair   :: [lift,liftargs] ⇒ lift            ((1'(-,/ -')))

-liftCons :: [lift, lift] ⇒ lift                 ((- #/ -) [65,66] 65)
-liftApp  :: [lift, lift] ⇒ lift                 ((- @/ -) [65,66] 65)
-liftList :: liftargs ⇒ lift                     ([(-)])


-ARAll  :: [idts, lift] ⇒ lift                   ((3! -./ -) [0, 10] 10)
-AREx   :: [idts, lift] ⇒ lift                   ((3? -./ -) [0, 10] 10)
-AREx1  :: [idts, lift] ⇒ lift                   ((3?! -./ -) [0, 10] 10)
-RAll   :: [idts, lift] ⇒ lift                   ((3∀ -./ -) [0, 10] 10)
```

-REx :: [*idts*, *lift*] ⇒ *lift* ((3∃ -./ -) [0, 10] 10)
-REx1 :: [*idts*, *lift*] ⇒ *lift* ((3∃!-./ -) [0, 10] 10)

translations
-const ⇌ *CONST const*

translations
-lift ⇌ *CONST lift*
-lift2 ⇌ *CONST lift2*
-lift3 ⇌ *CONST lift3*
-lift4 ⇌ *CONST lift4*
-Valid ⇌ *CONST Valid*

translations
-RAll x A ⇌ *Rall x. A*
-REx x A ⇌ *Rex x. A*
-REx1 x A ⇌ *Rex! x. A*

translations
-ARAll ⇀ *-RAll*
-AREx ⇀ *-REx*
-AREx1 ⇀ *-REx1*

w ⊨ *A* ⇀ *A w*
LIFT A ⇀ *A*::-⇒-

translations
-liftEqu ⇌ *-lift2* (=)
-liftNeq u v ⇌ *-liftNot* (*-liftEqu u v*)
-liftNot ⇌ *-lift* (*CONST Not*)
-liftAnd ⇌ *-lift2* (&)
-liftOr ⇌ *-lift2* ((|))
-liftImp ⇌ *-lift2* (−−>)
-liftIf ⇌ *-lift3* (*CONST If*)
-liftPlus ⇌ *-lift2* (+)
-liftMinus ⇌ *-lift2* (−)
-liftTimes ⇌ *-lift2* ((∗))
-liftDiv ⇌ *-lift2* (*div*)
-liftMod ⇌ *-lift2* (*mod*)
-liftLess ⇌ *-lift2* (<)
-liftLeq ⇌ *-lift2* (<=)
-liftMem ⇌ *-lift2* (:)
-liftNotMem x xs ⇌ *-liftNot* (*-liftMem x xs*)

translations
-liftFinset (*-liftargs x xs*) ⇌ *-lift2* (*CONST insert*) *x* (*-liftFinset xs*)
-liftFinset x ⇌ *-lift2* (*CONST insert*) *x* (*-const* (*CONST Set.empty*))
-liftPair x (*-liftargs y z*) ⇌ *-liftPair x* (*-liftPair y z*)
-liftPair ⇌ *-lift2* (*CONST Pair*)
-liftCons ⇌ *-lift2* (*CONST Cons*)

-*liftApp* $\rightleftharpoons$ -*lift2* (@)
-*liftList* (-*liftargs x xs*) $\rightleftharpoons$ -*liftCons x* (-*liftList xs*)
-*liftList x* $\rightleftharpoons$ -*liftCons x* (-*const* [])

$w \models \neg A \leftharpoondown$ -*liftNot A w*
$w \models A \wedge B \leftharpoondown$ -*liftAnd A B w*
$w \models A \vee B \leftharpoondown$ -*liftOr A B w*
$w \models A \longrightarrow B \leftharpoondown$ -*liftImp A B w*
$w \models u = v \leftharpoondown$ -*liftEqu u v w*
$w \models \forall x.\ A \leftharpoondown$ -*RAll x A w*
$w \models \exists x.\ A \leftharpoondown$ -*REx x A w*
$w \models \exists !x.\ A \leftharpoondown$ -*REx1 x A w*

syntax (*ASCII*)
-*Valid* :: *lift* ⇒ *bool* ((|− -) *5*)
-*holdsAt* :: ['*a*, *lift*] ⇒ *bool* ((- |= -) [*100*,*10*] *10*)
-*liftNeq* :: [*lift*, *lift*] ⇒ *lift* ((- ~=/ -) [*50*,*51*] *50*)
-*liftNot* :: *lift* ⇒ *lift* ((~ -) [*90*] *90*)
-*liftAnd* :: [*lift*, *lift*] ⇒ *lift* ((- &/ -) [*36*,*35*] *35*)
-*liftOr* :: [*lift*, *lift*] ⇒ *lift* ((- |/ -) [*31*,*30*] *30*)
-*liftImp* :: [*lift*, *lift*] ⇒ *lift* ((- −−>/ -) [*26*,*25*] *25*)
-*liftLeq* :: [*lift*, *lift*] ⇒ *lift* ((-/ <= -) [*50*, *51*] *50*)
-*liftMem* :: [*lift*, *lift*] ⇒ *lift* ((-/ : -) [*50*, *51*] *50*)
-*liftNotMem* :: [*lift*, *lift*] ⇒ *lift* ((-/ ~: -) [*50*, *51*] *50*)
-*RAll* :: [*idts*, *lift*] ⇒ *lift* ((*3ALL* -./ -) [*0*, *10*] *10*)
-*REx* :: [*idts*, *lift*] ⇒ *lift* ((*3EX* -./ -) [*0*, *10*] *10*)
-*REx1* :: [*idts*, *lift*] ⇒ *lift* ((*3EX*! -./ -) [*0*, *10*] *10*)

2.3 Lemmas and Tactics

lemma *intD*[*dest*]: $\vdash A \Longrightarrow w \models A$
proof −
assume *a*:$\vdash A$
from *a* **have** $\forall w.\ w \models A$ **by** (*auto simp add*: *Valid-def*)
thus *?thesis* **..**
qed

lemma *intI* [*intro*!]: **assumes** *P1*:($\bigwedge w.\ w \models A$) **shows** $\vdash A$
using *assms* **by** (*auto simp*: *Valid-def*)

Basic unlifting introduces a parameter w and applies basic rewrites, e.g $\vdash F = G$ becomes $F\ w = G\ w$ and $\vdash F \longrightarrow G$ becomes $F\ w \longrightarrow G\ w$.

method-setup *int-unlift* = ⟪
Scan.succeed (*fn ctxt* => *SIMPLE-METHOD'*
(*resolve-tac ctxt* @{*thms intI*} *THEN'* *rewrite-goal-tac ctxt* @{*thms intensional-rews*}))
⟫ *method to unlift and followed by intensional rewrites*

lemma *inteq-reflection*: **assumes** *P1*: $\vdash x{=}y$ **shows** ($x \equiv y$)
proof −
from *P1* **have** *P2*: $\forall w.\ x\ w = y\ w$ **by** (*unfold Valid-def unl-lift2*)

hence $P3$:x=y **by** *blast*
thus $x \equiv y$ **by** (*rule eq-reflection*)
qed

lemma *int-simps*:
$\vdash (x{=}x) = \#True$
$\vdash (\neg\ \#True) = \#False$
$\vdash (\neg\ \#False) = \#True$
$\vdash (\neg\neg\ P) = P$
$\vdash ((\neg\ P) = P) = \#False$
$\vdash (P = (\neg P)) = \#False$
$\vdash (P \neq Q) = (P = (\neg\ Q))$
$\vdash (\#True{=}P) = P$
$\vdash (P{=}\#True) = P$
$\vdash (\#True \longrightarrow P) = P$
$\vdash (\#False \longrightarrow P) = \#True$
$\vdash (P \longrightarrow \#True) = \#True$
$\vdash (P \longrightarrow P) = \#True$
$\vdash (P \longrightarrow \#False) = (\neg P)$
$\vdash (P \longrightarrow {\sim}P) = (\neg P)$
$\vdash (P \wedge \#True) = P$
$\vdash (\#True \wedge P) = P$
$\vdash (P \wedge \#False) = \#False$
$\vdash (\#False \wedge P) = \#False$
$\vdash (P \wedge P) = P$
$\vdash (P \wedge {\sim}P) = \#False$
$\vdash (\neg P \wedge P) = \#False$
$\vdash (P \vee \#True) = \#True$
$\vdash (\#True \vee P) = \#True$
$\vdash (P \vee \#False) = P$
$\vdash (\#False \vee P) = P$
$\vdash (P \vee P) = P$
$\vdash (P \vee \neg P) = \#True$
$\vdash (\neg P \vee P) = \#True$
$\vdash (\forall\ x.\ P) = P$
$\vdash (\exists\ x.\ P) = P$
by *auto*

lemmas *intensional-simps*[*simp*] = *int-simps*[*THEN inteq-reflection*]

method-setup *int-rewrite* = ⟪
Scan.succeed (*fn ctxt* => *SIMPLE-METHOD′* (*rewrite-goal-tac ctxt* @{*thms intensional-simps*}))
⟫ *rewrite method at intensional level*

lemma *Not-Rall*: $\vdash (\neg(\forall\ x.\ F\ x)) = (\exists\ x.\ \neg F\ x)$
by *auto*

lemma *Not-Rex*: $\vdash (\neg(\exists\ x.\ F\ x)) = (\forall\ x.\ \neg F\ x)$
by *auto*

lemma *TrueW* [*simp*]: $\vdash \#True$
  by *auto*

lemma *int-eq*: $\vdash X = Y \Longrightarrow X = Y$
  by (*auto simp*: *inteq-reflection*)

lemma *int-iffI*:
  assumes $\vdash F \longrightarrow G$ **and** $\vdash G \longrightarrow F$
  shows $\vdash F = G$
  using *assms* **by** *force*

lemma *int-iffD1*: **assumes** *h*: $\vdash F = G$ **shows** $\vdash F \longrightarrow G$
  using *h* **by** *auto*

lemma *int-iffD2*: **assumes** *h*: $\vdash F = G$ **shows** $\vdash G \longrightarrow F$
  using *h* **by** *auto*

lemma *lift-imp-trans*:
  assumes $\vdash A \longrightarrow B$ **and** $\vdash B \longrightarrow C$
  shows $\vdash A \longrightarrow C$
  using *assms* **by** *force*

lemma *lift-imp-neg*: **assumes** $\vdash A \longrightarrow B$ **shows** $\vdash \neg B \longrightarrow \neg A$
  using *assms* **by** *auto*

lemma *lift-and-com*: $\vdash (A \wedge B) = (B \wedge A)$
  by *auto*

end

3 Semantics

theory *Semantics*
imports *Interval Intensional*
begin

This theory mechanises a *shallow* embedding of 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 [4, 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* ⇒ *'b*
type-synonym *'a formula* = *('a,bool) formfun*
type-synonym *('a,'b) stfun* = *'a* ⇒ *'b*
type-synonym *'a stpred* = *('a,bool) stfun*

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

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

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

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

3.2 Semantics of ITL

The semantics of ITL is defined.

definition *skip-d* :: *('a ::world) formula*
where *skip-d* ≡ λ*s. intlen s =1*

definition *chop-d* :: *('a ::world) formula* ⇒ *('a ::world) formula* ⇒ *('a ::world) formula*
where *chop-d F1 F2* ≡ λ *s*. ∃ *n. 0*≤ *n* ∧ *n*≤*intlen s* ∧ ((*prefix n s*) ⊨ *F1*) ∧ ((*suffix n s*) ⊨ *F2*)

definition *chopstar-d* :: *('a::world) formula* ⇒ *'a formula*
where *chopstar-d F* ≡ λ*s*. (∃ (*l::index*). *index-sequence 0 l* ∧ (*nth l* (*intlen l*)) = (*intlen s*) ∧
(∀ *i*. (*0*≤*i* ∧ *i*< (*intlen l*)) ⟶
((*sub* (*nth l i*) (*nth l* (*i*+*1*)) *s*) ⊨ *F*)
)
)

definition *reverse-d* :: *('a::world, 'b) formfun* ⇒ *('a, 'b) formfun*
where *reverse-d F* ≡ λ *s. intrev s* ⊨ *F*

definition *current-val-d* :: *('a::world,'b) stfun* ⇒ *('a,'b) formfun*
where *current-val-d f* ≡ λ *s*. (*nth s 0*) ⊨ *f*

definition *next-val-d* :: *('a::world,'b) stfun* ⇒ *('a,'b) formfun*
where *next-val-d f* ≡ λ *s. if intlen s* >*0 then* ((*nth s 1*) ⊨ *f*) *else* (ϵ (*x::'b*). *x*=*x*)

definition *fin-val-d* :: *('a::world,'b) stfun* ⇒ *('a,'b) formfun*
where *fin-val-d f* ≡ λ *s*. (*nth s* (*intlen s*)) ⊨ *f*

definition *penult-val-d* :: *('a::world,'b) stfun* ⇒ *('a,'b) formfun*
where *penult-val-d f* ≡ λ *s. if intlen s* >*0 then* (*nth s* ((*intlen s*)−*1*) ⊨ *f*) *else* (ϵ (*x::'b*). *x*=*x*)

3.2.1 Concrete Syntax

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

```
syntax
 -skip-d        :: lift                ((skip))
 -chop-d        :: [lift,lift] ⇒ lift ((-;-) [84,84] 83)
 -chopstar-d    :: lift ⇒ lift        ((-⋆) [85] 85)
 -reverse-d     :: lift ⇒ lift        ((-ʳ) [85] 85)
 -current-val-d :: lift ⇒ lift        (($-) [100] 99)
 -next-val-d    :: lift ⇒ lift        ((-$) [100] 99)
 -fin-val-d     :: lift ⇒ lift        ((!-) [100] 99)
 -penult-val-d  :: lift ⇒ lift        ((-!) [100] 99)
 TEMP           :: lift ⇒ 'b          ((TEMP -))

syntax (ASCII)
 -skip-d        :: lift                ((skip))
 -chop-d        :: [lift,lift] ⇒ lift ((-;-) [84,84] 83)
 -chopstar-d    :: lift ⇒ lift        ((chopstar -) [85] 85)
 -reverse-d     :: lift ⇒ lift        ((reverse -) [85] 85)
 -current-val-d :: lift ⇒ lift        (($-) [100] 99)
 -next-val-d    :: lift ⇒ lift        ((-$) [100] 99)
 -fin-val-d     :: lift ⇒ lift        ((!-) [100] 99)
 -penult-val-d  :: lift ⇒ lift        ((-!) [100] 99)
translations
 -skip-d        ⇌ CONST skip-d
 -chop-d        ⇌ CONST chop-d
 -chopstar-d    ⇌ CONST chopstar-d
 -reverse-d     ⇌ CONST reverse-d
 -current-val-d ⇌ CONST current-val-d
 -next-val-d    ⇌ CONST next-val-d
 -fin-val-d     ⇌ CONST fin-val-d
 -penult-val-d  ⇌ CONST penult-val-d
 TEMP F         ⇀ (F:: (- interval) ⇒ -)
```

3.3 Abbreviations

Some standard temporal abbreviations, with their concrete syntax.

definition *sometimes-d* :: ('a::world) *formula* ⇒ 'a *formula*
where *sometimes-d F* ≡ *LIFT*(#*True*;*F*)

definition *di-d* :: ('a::world) *formula* ⇒ 'a *formula*
where *di-d F* ≡ *LIFT*(*F*;#*True*)

definition *da-d* :: ('a::world) *formula* ⇒ 'a *formula*
where *da-d F* ≡ *LIFT*(#*True*;(*F*;#*True*))

definition *next-d* :: ('a::world) *formula* ⇒ 'a *formula*
where *next-d F* ≡ *LIFT*(*skip*;*F*)

definition *prev-d* :: ('a::world) formula ⇒ 'a formula
where *prev-d F* ≡ *LIFT*(F;skip)

3.3.1 Concrete Syntax

syntax
-sometimes-d :: lift ⇒ lift ((◇-) [88] 87)
-di-d :: lift ⇒ lift ((di -) [88] 87)
-da-d :: lift ⇒ lift ((da -) [88] 87)
-next-d :: lift ⇒ lift ((○ -) [88] 87)
-prev-d :: lift ⇒ lift ((prev -) [88] 87)

syntax (ASCII)
-sometimes-d :: lift ⇒ lift ((<>-) [88] 87)
-di-d :: lift ⇒ lift ((di -) [88] 87)
-da-d :: lift ⇒ lift ((da -) [88] 87)
-next-d :: lift ⇒ lift ((next -) [88] 87)
-prev-d :: lift ⇒ lift ((prev -) [88] 87)

translations
-sometimes-d ⇌ CONST sometimes-d
-di-d ⇌ CONST di-d
-da-d ⇌ CONST da-d
-next-d ⇌ CONST next-d
-prev-d ⇌ CONST prev-d

definition *always-d* :: ('a::world) formula ⇒ 'a formula
where *always-d F* ≡ *LIFT*(¬(◇(¬F)))

definition *bi-d* :: ('a::world) formula ⇒ 'a formula
where *bi-d F* ≡ *LIFT*(¬(di(¬F)))

definition *ba-d* :: ('a::world) formula ⇒ 'a formula
where *ba-d F* ≡ *LIFT*(¬(da(¬F)))

definition *wnext-d* :: ('a::world) formula ⇒ 'a formula
where *wnext-d F* ≡ *LIFT*(¬(○(¬F)))

definition *wprev-d* :: ('a::world) formula ⇒ 'a formula
where *wprev-d F* ≡ *LIFT*(¬(prev(¬F)))

definition *more-d* :: ('a::world) formula
where *more-d* ≡ *LIFT*(○(#True))

syntax
-always-d :: lift ⇒ lift ((□-) [88] 87)

-bi-d :: lift ⇒ lift ((bi -) [88] 87)
-ba-d :: lift ⇒ lift ((ba -) [88] 87)
-wnext-d :: lift ⇒ lift ((wnext -) [88] 87)
-wprev-d :: lift ⇒ lift ((wprev -) [88] 87)
-more-d :: lift ((more))

syntax (ASCII)
-always-d :: lift ⇒ lift (([]-) [88] 87)
-bi-d :: lift ⇒ lift ((bi -) [88] 87)
-ba-d :: lift ⇒ lift ((ba -) [88] 87)
-wnext-d :: lift ⇒ lift ((wnext -) [88] 87)
-wprev-d :: lift ⇒ lift ((wprev -) [88] 87)
-more-d :: lift ((more))

translations
-always-d ⇌ CONST always-d
-bi-d ⇌ CONST bi-d
-ba-d ⇌ CONST ba-d
-wnext-d ⇌ CONST wnext-d
-wprev-d ⇌ CONST wprev-d
-more-d ⇌ CONST more-d

definition empty-d :: ('a::world) formula
where empty-d ≡ LIFT(¬(more))

definition dm-d :: ('a::world) formula ⇒ 'a formula
where dm-d F ≡ LIFT(#True;(more ∧ F))

syntax
-empty-d :: lift ((empty))
-dm-d :: lift ⇒ lift ((dm -) [88] 87)

syntax (ASCII)
-empty-d :: lift ((empty))
-dm-d :: lift ⇒ lift ((dm -) [88] 87)

translations
-empty-d ⇌ CONST empty-d
-dm-d ⇌ CONST dm-d

definition bm-d :: ('a::world) formula ⇒ 'a formula
where bm-d F ≡ LIFT(¬(dm(¬F)))

definition init-d :: ('a::world) formula ⇒ 'a formula
where init-d F ≡ LIFT((empty ∧ F);#True)

definition fin-d :: ('a::world) formula ⇒ 'a formula

where *fin-d F* ≡ *LIFT*(□(*empty* ⟶ *F*))

definition *halt-d* :: (′*a*::*world*) *formula* ⇒ ′*a formula*
where *halt-d F* ≡ *LIFT*(□(*empty* = *F*))

definition *keep-d* :: (′*a*::*world*) *formula* ⇒ ′*a formula*
where *keep-d F* ≡ *LIFT*(*ba*(*skip* ⟶ *F*))

definition *yields-d* :: (′*a*::*world*) *formula* ⇒ ′*a formula* ⇒ ′*a formula*
where *yields-d F1 F2* ≡ *LIFT*(¬(*F1*;¬*F2*))

definition *ifthenelse-d* :: (′*a*::*world*) *formula* ⇒ ′*a formula* ⇒ ′*a formula* ⇒ ′*a formula*
where *ifthenelse-d F G H* ≡ *LIFT*((*F* ∧ *G*) ∨ (¬*F* ∧ *H*))

primrec *power-chop-d* :: (′*a*::*world*) *formula* ⇒ *nat* ⇒ ′*a formula*
where *power-0* : (*power-chop-d F 0*) = *LIFT*(*empty*)
| *power-Suc*: (*power-chop-d F* (*Suc n*)) = *LIFT*((*F* ∧ *more*);(*power-chop-d F n*))

primrec *len-d* :: *nat* ⇒ (′*a*::*world*) *formula*
where *len-0* : (*len-d 0*) = *LIFT*(*empty*)
| *len-Suc*: (*len-d* (*Suc n*)) = *LIFT*(*skip*;(*len-d n*))

primrec *power-d* :: (′*a*::*world*) *formula* ⇒ *nat* ⇒ ′*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* ⇒ *lift* ((*bm* -) [*88*] *87*)
-init-d :: *lift* ⇒ *lift* ((*init* -) [*88*] *87*)
-fin-d :: *lift* ⇒ *lift* ((*fin* -) [*88*] *87*)
-halt-d :: *lift* ⇒ *lift* ((*halt* -) [*88*] *87*)
-keep-d :: *lift* ⇒ *lift* ((*keep* -) [*88*] *87*)
-yields-d :: [*lift*,*lift*] ⇒ *lift* ((- *yields* -) [*88*,*88*] *87*)
-ifthenelse-d :: [*lift*,*lift*,*lift*] ⇒ *lift* ((*if* $_i$ - *then* - *else* -) [*88*,*88*,*88*] *87*)
-len-d :: *nat* ⇒ *lift* ((*len* -) [*88*] *87*)
-power-chop-d :: [*lift*,*nat*] ⇒ *lift* ((*powerchop* - -) [*88*,*88*] *87*)
-power-d :: [*lift*,*nat*] ⇒ *lift* ((*power* - -) [*88*,*88*] *87*)

syntax (*ASCII*)
-bm-d :: *lift* ⇒ *lift* ((*bm* -) [*88*] *87*)
-init-d :: *lift* ⇒ *lift* ((*init* -) [*88*] *87*)
-fin-d :: *lift* ⇒ *lift* ((*fin* -) [*88*] *87*)
-halt-d :: *lift* ⇒ *lift* ((*halt* -) [*88*] *87*)
-keep-d :: *lift* ⇒ *lift* ((*keep* -) [*88*] *87*)
-yields-d :: [*lift*,*lift*] ⇒ *lift* ((- *yields* -) [*88*,*88*] *87*)
-ifthenelse-d :: [*lift*,*lift*,*lift*] ⇒ *lift* ((*if* $_i$ - *then* - *else* -) [*88*,*88*,*88*] *87*)
-len-d :: *nat* ⇒ *lift* ((*len* -) [*88*] *87*)
-power-chop-d :: [*lift*,*nat*] ⇒ *lift* ((*powerchop* - -) [*88*,*88*] *87*)

-power-d :: [*lift*,*nat*] ⇒ *lift* ((*power* - -) [*88*,*88*] *87*)

translations
-bm-d ⇌ *CONST bm-d*
-init-d ⇌ *CONST init-d*
-fin-d ⇌ *CONST fin-d*
-halt-d ⇌ *CONST halt-d*
-keep-d ⇌ *CONST keep-d*
-yields-d ⇌ *CONST yields-d*
-ifthenelse-d ⇌ *CONST ifthenelse-d*
-len-d ⇌ *CONST len-d*
-power-chop-d ⇌ *CONST power-chop-d*
-power-d ⇌ *CONST power-d*

definition *ifthen-d* :: (*'a*::*world*) *formula* ⇒ *'a formula* ⇒ *'a formula*
where *ifthen-d F G* ≡ *LIFT*(*if* $_i$ *F then G else* #*True*)

definition *while-d* :: (*'a*::*world*) *formula* ⇒ *'a formula* ⇒ *'a formula*
where *while-d F G* ≡ *LIFT*((*F* ∧ *G*)* ∧ (*fin* ((¬*F*))))

syntax
-ifthen-d :: [*lift*,*lift*] ⇒ *lift* ((*if* $_i$ - *then* -) [*88*,*88*] *87*)
-while-d :: [*lift*,*lift*] ⇒ *lift* ((*while* - *do* -) [*88*,*88*] *87*)

syntax (*ASCII*)
-ifthen-d :: [*lift*,*lift*] ⇒ *lift* ((*if* $_i$ - *then* -) [*88*,*88*] *87*)
-while-d :: [*lift*,*lift*] ⇒ *lift* ((*while* - *do* -) [*88*,*88*] *87*)

translations
-ifthen-d ⇌ *CONST ifthen-d*
-while-d ⇌ *CONST while-d*

definition *repeat-d* :: (*'a*::*world*) *formula* ⇒ *'a formula* ⇒ *'a formula*
where *repeat-d F G* ≡ *LIFT*(*F*;*while* (¬ *G*) *do F*)

syntax
-repeat-d :: [*lift*,*lift*] ⇒ *lift* ((*repeat* - *until* -) [*88*,*88*] *87*)

syntax (*ASCII*)
-repeat-d :: [*lift*,*lift*] ⇒ *lift* ((*repeat* - *until* -) [*88*,*88*] *87*)

translations
-repeat-d ⇌ *CONST repeat-d*

definition *assign-d* :: (*'a*::*world*,*'b*) *stfun* ⇒ (*'a*,*'b*) *formfun* ⇒ *'a formula*
where *assign-d v e* ≡ *LIFT*(*v*$ = *e*)

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

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

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

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

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

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

translations
-assign-d $\rightleftharpoons$ *CONST assign-d*
-prev-assign-d $\rightleftharpoons$ *CONST prev-assign-d*
-always-eq-d $\rightleftharpoons$ *CONST always-eq-d*
-temporal-assign-d $\rightleftharpoons$ *CONST temporal-assign-d*
-gets-d $\rightleftharpoons$ *CONST gets-d*
-stable-d $\rightleftharpoons$ *CONST stable-d*
-padded-d $\rightleftharpoons$ *CONST padded-d*
-padded-temp-assign-d $\rightleftharpoons$ *CONST padded-temp-assign-d*

3.4 Properties of Operators

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

lemma *skip-defs* :
$(w \models skip) = (intlen\ w = 1)$
by (*simp add*: *skip-d-def*)

lemma *chop-defs* :
$(w \models F1 ; F2) = (\exists\ n .\ 0 \le n \wedge n \le intlen\ w \wedge ((prefix\ n\ w) \models F1) \wedge ((suffix\ n\ w) \models F2)\ )$
by (*simp add*: *chop-d-def*)

lemma *sometimes-defs* :
$(w \models \diamond F) = (\exists\ n.\ 0 \le n \wedge n \le intlen\ w \wedge ((suffix\ n\ w) \models F))$
by (*simp add*: *Semantics.sometimes-d-def chop-defs*)

lemma *always-defs* :
$(w \models \Box F) = (\forall\ n.\ 0 \le n \wedge n \le intlen\ w \longrightarrow ((suffix\ n\ w) \models F))$
by (*simp add*: *always-d-def sometimes-defs*)

lemma *di-defs* :
$(w \models di\ F) = (\exists\ n.\ 0 \le n \wedge n \le intlen\ w \wedge ((prefix\ n\ w) \models F))$
by (*simp add*: *Semantics.di-d-def chop-defs*)

lemma *bi-defs* :
$(w \models bi\ F) = (\forall\ n.\ 0 \le n \wedge n \le intlen\ w \longrightarrow ((prefix\ n\ w) \models F))$
by (*simp add*: *Semantics.bi-d-def di-defs*)

lemma *da-defs* :
$(w \models da\ F) = (\exists\ n\ na.\ 0 \le n \wedge\ na+n \le intlen\ w \wedge ((sub\ n\ (na+n)\ w) \models F))$
apply (*simp add*: *Semantics.da-d-def chop-defs*)
using *interval-prefix-length-good interval-suffix-length-good*
by (*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* :
$(w \models ba\ F) = (\forall\ n\ na.\ 0 \le n \wedge\ na+n \le intlen\ w \longrightarrow ((sub\ n\ (na+n)\ w) \models F))$
by (*simp add*: *ba-d-def da-defs*)

lemma *next-defs* :
$(w \models \bigcirc F) = (intlen\ w > 0 \wedge ((suffix\ 1\ w) \models F)\ )$
apply (*simp add*: *next-d-def chop-defs skip-defs*)
using *Suc-le-eq* **by** *force*

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

lemma *prev-defs* :
$(w \models prev\ F) = (intlen\ w > 0 \wedge ((prefix\ ((intlen\ w)-1)\ w) \models F)\ )$
by (*simp add*: *prev-d-def chop-defs skip-defs*)

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

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

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

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

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

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

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

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

lemma *len-defs* :
$(w \models len\ n) = (intlen\ w = n)$
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* :
$(s \models \$v) = (v\ (nth\ s\ 0))$
by (*simp add*: *current-val-d-def*)

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

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

lemma *penultval-defs* :

(*s* ⊨ *v*!) = (*if intlen s* >*0 then* (*v* (*nth s* ((*intlen s*)−*1*))) *else* (ϵ *x*. *x*=*x*))
by (*simp add*: *penult-val-d-def*)

lemma *assign-defs* :
intlen s >*0* ⟹ (*s* ⊨ *v* := *e*) = *v* (*Interval.nth s 1*) = *e s*
by (*auto simp*: *assign-d-def next-val-d-def*)

lemma *prev-assign-defs* :
intlen s >*0* ⟹ (*s* ⊨ *v* =: *e*) = *v* (*Interval.nth s* ((*intlen s*)−*1*)) = *e s*
by (*auto simp*: *prev-assign-d-def penult-val-d-def*)

lemma *always-eqv-defs* :
(*s* ⊨ *v* ≈ *e*) = (∀ *i*≤ *intlen s*. *v* (*Interval.nth s i*) = *e* (*suffix i s*))
by (*simp add*: *always-eq-d-def always-defs current-val-d-def*)

lemma *temporal-assign-defs* :
(*s* ⊨ *v* ← *e*) = (*v* (*Interval.nth s* (*intlen s*)) = *e s*)
by (*simp add*: *temporal-assign-d-def fin-val-d-def*)

lemma *gets-defs* :
(*s* ⊨ *v gets e*) = (∀ *i* < *intlen s*. *v* (*Interval.nth s* (*Suc i*)) = *e* (*sub i* (*i*+*1*) *s*))
apply (*simp add*: *gets-d-def keep-d-def ba-defs skip-defs sub-def temporal-assign-defs*)
using *Suc-le-eq* **by** *blast*

lemma *stable-defs-help*:
(∀ *i*<*intlen s*. *v* (*Interval.nth s* (*Suc i*)) = *v* (*Interval.nth s i*)) =
(∀ *i*≤*intlen s*. *v* (*Interval.nth s i*) = *v* (*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-lessI 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* ⊨ *stable v*) = (∀ *i*≤*intlen s*. (*v* (*nth s i*)) = (*v* (*nth s 0*)))
by (*simp add*: *stable-d-def gets-defs current-val-d-def sub-def stable-defs-help*)

lemma *padded-defs* :
(*s* ⊨ *padded v*) = ((∀ *i*< *intlen s*. (*v* (*nth s i*)) = (*v* (*nth s 0*))) ∨ *intlen s* =*0*)
apply (*simp add*: *padded-d-def stable-defs chop-d-def skip-defs empty-defs interval-suffix-length*)
by (*smt Suc-leI 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* ⊨ *v* <∼ *e*) =

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

lemma *linalw*:
$a \leq b \wedge b \leq \mathit{intlen}\ w \wedge ((\mathit{suffix}\ a\ w\ ) \models \Box\ A) \longrightarrow ((\mathit{suffix}\ b\ w\ ) \models \Box\ A)$
apply (*simp add*: *always-defs*)
by (*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 Axioms

3.5.1 ChopAssoc

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

lemma *ChopAssocSem*:
$(\sigma \models\ f\ ; (g\ ;\ h) =\ (f;g);h)$
using *ChopAssocSemHelp2* **using** *unl-lift2* **by** *blast*

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*:
  $(\sigma \models \mathit{empty}\ ;\ f = f)$
by (*simp add*: *empty-defs chop-defs*) *auto*

3.5.5 ChopEmpty

lemma *ChopEmptySem*:
  $(\sigma \models f;\mathit{empty} = f)$
by (*simp add*: *empty-defs chop-defs*) *auto*

3.5.6 StateImpBi

lemma *StateImpBiSem*:
  $(\sigma \models \mathit{init}\ f \longrightarrow \mathit{bi}\ (\mathit{init}\ f))$
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*:
  $(\sigma \models \mathit{bi}\ (f \longrightarrow f1) \wedge \Box(g \longrightarrow g1) \longrightarrow f;g \longrightarrow f1;g1)$
by (*simp add*: *bi-defs always-defs chop-defs*) *fastforce*

3.5.9 BoxInduct

lemma *box-induct-help-1* :
 $(\sigma \models f) \wedge (\forall i.\ \mathit{Suc}\ 0 \leq \mathit{intlen}\ \sigma - i \longrightarrow$
   $i \leq \mathit{intlen}\ \sigma \longrightarrow (\mathit{suffix}\ i\ \sigma \models f) \longrightarrow (\mathit{suffix}\ (\mathit{Suc}\ i)\ \sigma \models f))$
  $\Longrightarrow (\forall\ j.\ j \leq \mathit{intlen}\ \sigma \longrightarrow (\mathit{suffix}\ j\ \sigma \models f))$
proof
  fix *j*
  show $(\sigma \models f) \wedge (\forall i.\ \mathit{Suc}\ 0 \leq \mathit{intlen}\ \sigma - i \longrightarrow$
   $i \leq \mathit{intlen}\ \sigma \longrightarrow (\mathit{suffix}\ i\ \sigma \models f) \longrightarrow (\mathit{suffix}\ (\mathit{Suc}\ i)\ \sigma \models f))$

$\Longrightarrow\ j \leq intlen\ \sigma \longrightarrow (suffix\ j\ \sigma \models f)$
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*:
$(\sigma \models \Box\ (f \longrightarrow wnext\ f) \wedge f \longrightarrow \Box\ f)$
apply (*simp add*: *always-defs wnext-defs*)
using *box-induct-help-1* **by** (*metis One-nat-def diff-self-eq-0 not-one-le-zero*)

3.5.10 ChopStarEqv

lemma *chopstar-help-1*:
$(\ \exists\, l.\ l = \langle 0 \rangle \wedge index\text{-}sequence\ 0\ l\ \wedge$
$Interval.nth\ l\ (intlen\ l) = (intlen\ \sigma)\ \wedge$
$(\forall\, i.\ (0 \leq i \wedge i < (intlen\ l)) \longrightarrow$
$((sub\ (nth\ l\ i)\ (nth\ l\ (i+1))\ \sigma) \models f)$
$)) \longleftrightarrow (intlen\ \sigma = 0)$
by (*simp add*: *index-sequence-def*)

lemma *chopstar-help-2*:
$(\forall\, i.\ (0 < i \wedge i < 1 + (intlen\ ls)) \longrightarrow$
$((sub\ (nth\ ls\ (i-1))\ (nth\ ls\ ((i-1)+1))\ \sigma) \models f)$
$) =$
$(\forall\, i.\ (0 \leq i \wedge i < (intlen\ ls)) \longrightarrow$
$((sub\ (nth\ ls\ (i))\ (nth\ ls\ ((i)+1))\ \sigma) \models f)$
$)$
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*:
$(\exists\ (l{::}index).\ (intlen\ l) = (Suc\ n) \wedge index\text{-}sequence\ 0\ l \wedge (nth\ l\ (intlen\ l)) = (intlen\ \sigma)\ \wedge$
$(\forall\, i.\ (0 \leq i \wedge i < (intlen\ l)) \longrightarrow$
$((sub\ (nth\ l\ i)\ (nth\ l\ (i+1))\ \sigma) \models f)$
$)$
$) =$
$(\exists\ k.\ 0 \leq k \wedge k \leq intlen\ \sigma \wedge k > 0\ \wedge$
$(sub\ 0\ k\ \sigma \models f)\ \wedge$
$(\exists\ ls.\ (intlen\ ls) = n \wedge index\text{-}sequence\ 0\ (ls)\ \wedge$
$(nth\ (ls)\ (intlen\ (ls))) = (intlen\ (suffix\ k\ \sigma))$
$\wedge\ (\forall\, i.\ (0 \leq i \wedge i < (intlen\ ls)) \longrightarrow$
$((sub\ (nth\ ls\ (i))\ (nth\ ls\ ((i)+1))\ (suffix\ k\ \sigma)) \models f)$
$))$

```
        )
proof -
 have (∃ (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)
                       )
              )
      =
      (∃ x ls l.  (intlen l) = (Suc n) ∧ l=x⊙ls ∧ 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)
                       )
             )
 using interval-intlen-cons-1 by (metis zero-less-Suc)
 also have ... =
          (∃ x ls l.  (intlen l) = (Suc n) ∧ l=x⊙ls ∧ index-sequence 0 (x⊙ls) ∧
                      (nth (x⊙ls) (intlen (x⊙ls))) = (intlen σ) ∧
                       (∀ i. (0≤i ∧ i< (intlen (x⊙ls))) ⟶
                          ((sub (nth (x⊙ls) i) (nth (x⊙ls) (i+1)) σ) ⊨ f)
                       )
                 )
 by auto
 also have ... =
          (∃ x ls . (intlen ls) = n ∧ index-sequence 0 (x⊙ls) ∧
                    (nth (x⊙ls) (intlen (x⊙ls))) = (intlen σ) ∧
                       (∀ i. (0≤i ∧ i< (intlen (x⊙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 0 (x ⊙ ls) ∧
                    (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 ... =
$(\exists\ x\ ls\ .\ (intlen\ ls) = n \wedge x = 0 \wedge index\text{-}sequence\ (nth\ ls\ 0)\ (ls) \wedge$
$(nth\ (ls)\ (intlen\ (ls))) = (intlen\ \sigma) \wedge$
$(x < (nth\ ls\ 0) \wedge$
$((sub\ (nth\ (x \odot ls)\ 0)\ (nth\ (x \odot ls)\ (1))\ \sigma) \models f)$
$\wedge$
$((\forall\ i.\ (0 < i \wedge i < 1 + (intlen\ (ls))) \longrightarrow$
$((sub\ (nth\ (x \odot ls)\ i)\ (nth\ (x \odot ls)\ (i+1))\ \sigma) \models f))$
$)$
$)$
$)$
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 ... =
$(\exists\ x\ ls\ .\ (intlen\ ls) = n \wedge x = 0 \wedge index\text{-}sequence\ (nth\ ls\ 0)\ (ls) \wedge$
$(nth\ (ls)\ (intlen\ (ls))) = (intlen\ \sigma) \wedge$
$(x < (nth\ ls\ 0) \wedge (nth\ (x \odot ls)\ 0) = x \wedge (nth\ (x \odot ls)\ (1)) = (nth\ ls\ 0) \wedge$
$((sub\ (nth\ (x \odot ls)\ 0)\ (nth\ (x \odot ls)\ (1))\ \sigma) \models f)$
$\wedge$
$((\forall\ i.\ (0 < i \wedge i < 1 + (intlen\ (ls))) \longrightarrow$
$((sub\ (nth\ (x \odot ls)\ i)\ (nth\ (x \odot ls)\ (i+1))\ \sigma) \models f))$
$)$
$)$
$)$
by *auto*
also have ... =
$(\exists\ x\ ls\ .\ (intlen\ ls) = n \wedge x = 0 \wedge index\text{-}sequence\ (nth\ ls\ 0)\ (ls) \wedge$
$(nth\ (ls)\ (intlen\ (ls))) = (intlen\ \sigma) \wedge$
$(x < (nth\ ls\ 0) \wedge (nth\ (x \odot ls)\ 0) = x \wedge (nth\ (x \odot ls)\ (1)) = (nth\ ls\ 0) \wedge$
$((sub\ x\ (nth\ ls\ 0)\ \sigma) \models f)$
$\wedge$
$((\forall\ i.\ (0 < i \wedge i < 1 + (intlen\ (ls))) \longrightarrow$
$((sub\ (nth\ (x \odot ls)\ i)\ (nth\ (x \odot ls)\ (i+1))\ \sigma) \models f))$
$)$
$)$
$)$
by *auto*
also have ... =
$(\exists\ x\ ls\ .\ (intlen\ ls) = n \wedge x = 0 \wedge index\text{-}sequence\ (nth\ ls\ 0)\ (ls) \wedge$
$(nth\ (ls)\ (intlen\ (ls))) = (intlen\ \sigma) \wedge$
$(x < (nth\ ls\ 0) \wedge$
$((sub\ x\ (nth\ ls\ 0)\ \sigma) \models f)$
$\wedge$
$((\forall\ i.\ (0 < i \wedge i < 1 + (intlen\ (ls))) \longrightarrow$
$((sub\ (nth\ (x \odot ls)\ i)\ (nth\ (x \odot ls)\ (i+1))\ \sigma) \models f))$
$)$
$)$
$)$
by *auto*
also have ... =

$(\exists\ x\ ls\ .\ (intlen\ ls) = n \wedge x = 0 \wedge index\text{-}sequence\ (nth\ ls\ 0)\ (ls) \wedge$
$(nth\ (ls)\ (intlen\ (ls))) = (intlen\ \sigma) \wedge$
$(\ x < (nth\ ls\ 0) \wedge$
$((sub\ x\ (nth\ ls\ 0)\ \sigma) \models f)$
$\wedge$
$(\forall i.\ (0<i \wedge i< 1+(intlen\ ls)) \longrightarrow$
$((sub\ (nth\ ls\ (i-1))\ (nth\ ls\ ((i-1)+1))\ \sigma) \models f)$
$)))$
using *interval-nth-cons* **by** *metis*
also have ... =
$(\exists\ x\ ls\ .\ (intlen\ ls) = n \wedge x = 0 \wedge index\text{-}sequence\ (nth\ ls\ 0)\ (ls) \wedge$
$(nth\ (ls)\ (intlen\ (ls))) = (intlen\ \sigma) \wedge$
$(x < (nth\ ls\ 0) \wedge$
$((sub\ x\ (nth\ ls\ 0)\ \sigma) \models f))$
$\wedge (\forall i.\ (0\leq i \wedge i< (intlen\ ls)) \longrightarrow$
$((sub\ (nth\ ls\ (i))\ (nth\ ls\ ((i)+1))\ \sigma) \models f)$
$)$
$)$
using *chopstar-help-2* **by** (*metis* (*mono-tags*))
also have ... =
$(\exists\ \ ls\ .\ (intlen\ ls) = n \wedge index\text{-}sequence\ (nth\ ls\ 0)\ (ls) \wedge$
$(nth\ (ls)\ (intlen\ (ls))) = (intlen\ \sigma) \wedge$
$(0 < (nth\ ls\ 0) \wedge$
$((sub\ 0\ (nth\ ls\ 0)\ \sigma) \models f))$
$\wedge (\forall i.\ (0\leq i \wedge i< (intlen\ ls)) \longrightarrow$
$((sub\ (nth\ ls\ (i))\ (nth\ ls\ ((i)+1))\ \sigma) \models f)$
$)$
$)$
by *simp*
also have ... =
$(\exists\ \ lsk\ .\ \ (intlen\ lsk) = n \wedge (nth\ lsk\ 0) \leq intlen\ \sigma \wedge (nth\ lsk\ 0) > 0 \wedge$
$((sub\ 0\ (nth\ lsk\ 0)\ \sigma) \models f) \wedge$
$index\text{-}sequence\ (nth\ lsk\ 0)\ (lsk) \wedge$
$(nth\ (lsk)\ (intlen\ (lsk))) = (intlen\ \sigma) \wedge$
$(\forall i.\ (0\leq i \wedge i< (intlen\ lsk)) \longrightarrow$
$((sub\ (nth\ lsk\ (i))\ (nth\ lsk\ ((i)+1))\ \sigma) \models f)$
$)$
$)$
by (*metis Suc-eq-plus1 Suc-pred add.left-neutral eq-iff interval-idx-less-last interval-intlen-gr-zero le-neq-implies-less lessI less-imp-le-nat*)
also have ... =
$(\exists\ k\ lsk.\ \ (intlen\ lsk) = n \wedge (nth\ lsk\ 0) \leq intlen\ \sigma \wedge$
$(nth\ lsk\ 0) > 0 \wedge k=(nth\ lsk\ 0) \wedge$
$(sub\ 0\ (nth\ lsk\ 0)\ \sigma \models f) \wedge$
$index\text{-}sequence\ (nth\ lsk\ 0)\ (lsk) \wedge$
$(nth\ (lsk)\ (intlen\ (lsk))) = (intlen\ (\sigma)) \wedge$
$(\forall i.\ (0\leq i \wedge i< (intlen\ lsk)) \longrightarrow$
$((sub\ ((nth\ lsk\ (i)))\ ((nth\ lsk\ ((i)+1)))\ (\sigma)) \models f)$
$)$
$)$

by *auto*
also have ... =
$(\exists\ k\ lsk.\ (intlen\ lsk) = n \wedge 0 \leq k \wedge k \leq intlen\ \sigma \wedge k > 0 \wedge k{=}(nth\ lsk\ 0) \wedge$
$(sub\ 0\ k\ \sigma \models f) \wedge$
$(index\text{-}sequence\ k\ (lsk) \wedge$
$(nth\ (lsk)\ (intlen\ (lsk))) = ((intlen\ (suffix\ k\ \sigma)) + k) \wedge$
$(\forall i.\ (0 \leq i \wedge i < (intlen\ lsk)) \longrightarrow$
$((sub\ ((nth\ lsk\ (i)))\ ((nth\ lsk\ ((i)+1)))\ (\sigma)) \models f)$
$))$
$)$
apply (*simp add*: *interval-prefix-suffix-intlen interval-suffix-length interval-prefix-length*)
by *auto*
also have ... =
$(\exists\ k\ lsk.\ (intlen\ lsk) = n \wedge 0 \leq k \wedge k \leq intlen\ \sigma \wedge k > 0 \wedge$
$(sub\ 0\ k\ \sigma \models f) \wedge$
$(index\text{-}sequence\ k\ (lsk) \wedge$
$(nth\ (lsk)\ (intlen\ (lsk))) = ((intlen\ (suffix\ k\ \sigma)) + k) \wedge$
$(\forall i.\ (0 \leq i \wedge i < (intlen\ lsk)) \longrightarrow$
$((sub\ ((nth\ lsk\ (i)))\ ((nth\ lsk\ ((i)+1)))\ (\sigma)) \models f)$
$))$
$)$
using *index-sequence-def* **by** *auto*
also have ... =
$(\exists\ k.\ 0 \leq k \wedge k \leq intlen\ \sigma \wedge k > 0 \wedge$
$(sub\ 0\ k\ \sigma \models f) \wedge$
$(\exists\ ls\ lsk.\ (intlen\ lsk) = n \wedge index\text{-}sequence\ k\ (lsk) \wedge$
$ls = map\ (shiftm\ k)\ lsk \wedge$
$(nth\ (lsk)\ (intlen\ (lsk))) = ((intlen\ (suffix\ k\ \sigma)) + k) \wedge$
$(\forall i.\ (0 \leq i \wedge i < (intlen\ lsk)) \longrightarrow$
$((sub\ ((nth\ lsk\ (i)))\ ((nth\ lsk\ ((i)+1)))\ (\sigma)) \models f)$
$))$
$)$
by *blast*
also have ... =
$(\exists\ k.\ 0 \leq k \wedge k \leq intlen\ \sigma \wedge k > 0 \wedge$
$(sub\ 0\ k\ \sigma \models f) \wedge$
$(\exists\ ls\ lsk.\ (intlen\ lsk) = n \wedge index\text{-}sequence\ k\ (lsk) \wedge$
$ls = map\ (shiftm\ k)\ lsk \wedge$
$index\text{-}sequence\ 0\ (ls) \wedge (intlen\ ls) = n \wedge$
$(nth\ (lsk)\ (intlen\ (lsk))) = ((intlen\ (suffix\ k\ \sigma)) + k) \wedge$
$(\forall i.\ (0 \leq i \wedge i < (intlen\ lsk)) \longrightarrow$
$((sub\ ((nth\ lsk\ (i)))\ ((nth\ lsk\ ((i)+1)))\ (\sigma)) \models f)$
$))$
$)$
using *interval-idx-link-shiftm* **by** *blast*
also have ... =
$(\exists\ k.\ 0 \leq k \wedge k \leq intlen\ \sigma \wedge k > 0 \wedge$
$(sub\ 0\ k\ \sigma \models f) \wedge$
$(\exists\ ls\ lsk.\ (intlen\ lsk) = n \wedge index\text{-}sequence\ k\ (lsk) \wedge$
$lsk = map\ (shift\ k)\ ls \wedge$

index-sequence 0 (ls) $\wedge$ (intlen ls) $= n$ $\wedge$
(nth (lsk) (intlen (lsk))) $=$ ((intlen (suffix k σ))$+k$) $\wedge$
($\forall\, i$. ($0 \leq i \wedge i <$ (intlen lsk)) $\longrightarrow$
((sub ((nth lsk (i))) ((nth lsk ((i)$+1$))) (σ)) $\models f$)
))
)
using *interval-lsk-ls* **by** *blast*
also have ... $=$
($\exists$ k ls lsk . $0 \leq k \wedge k \leq$ intlen $\sigma \wedge k > 0 \wedge$
(sub 0 k $\sigma \models f$) $\wedge$
((intlen lsk) $= n$ $\wedge$ lsk $=$ map (shift k) ls $\wedge$
index-sequence 0 (ls) $\wedge$
index-sequence k (lsk) $\wedge$
(nth (ls) (intlen (ls))) $=$ (intlen (suffix k σ)) $\wedge$
($\forall\, i$. ($0 \leq i \wedge i <$ (intlen ls)) $\longrightarrow$
((sub ((nth ls (i))$+k$) ((nth ls ((i)$+1$))$+k$) (σ)) $\models f$)
))
)
apply (*simp add*: *Interval.shift-def interval-intlen-map interval-nth-map*) **by** *blast*
also have ... $=$
($\exists$ k ls lsk. $0 \leq k \wedge k \leq$ intlen $\sigma \wedge k > 0 \wedge$
(sub 0 k $\sigma \models f$) $\wedge$
((intlen lsk) $= n$ $\wedge$ lsk $=$ map (shift k) ls $\wedge$
(intlen ls) $= n$ $\wedge$ index-sequence 0 (ls) $\wedge$
(nth (ls) (intlen (ls))) $=$ (intlen (suffix k σ)) $\wedge$
($\forall\, i$. ($0 \leq i \wedge i <$ (intlen ls)) $\longrightarrow$
((sub ((nth ls (i))$+k$) ((nth ls ((i)$+1$))$+k$) (σ)) $\models f$)
))
)
using *interval-idx-link* **by** *blast*
also have ... $=$
($\exists$ 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 σ)) $\wedge$
($\forall\, i$. ($0 \leq i \wedge i <$ (intlen ls)) $\longrightarrow$
((sub ((nth ls (i))$+k$) ((nth ls ((i)$+1$))$+k$) (σ)) $\models f$)
))
)
by (*simp add*: *interval-intlen-map*)
also have ... $=$
($\exists$ 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 σ)) $\wedge$
($\forall\, i \leq$intlen ls. Interval.nth ls i $\leq$ intlen (suffix k σ)) $\wedge$
($\forall\, i$. ($0 \leq i \wedge i <$ (intlen ls)) $\longrightarrow$
((sub ((nth ls (i))$+k$) ((nth ls ((i)$+1$))$+k$) (σ)) $\models f$)
)
)

```
        )
 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-eqv-sem:
  (∃ n. (σ ⊨ (power-chop-d f n))) =
   ((σ ⊨ empty) ∨ (∃ n. (σ ⊨ (f ∧ more); (power-chop-d f n))))
by (metis not0-implies-Suc power-chop-d.power-0 power-chop-d.power-Suc)

lemma chopstar-eqv-power-chop-help:
   ( σ ⊨ power-chop-d f 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 interval-intlen-st power-chop-d.power-0)
 next
 case (Suc n)
 then show ?case
  proof −
   have 1: (σ ⊨ power-chop-d f (Suc n)) = (σ ⊨ ((f ∧ more);(power-chop-d f n)))
   by simp
   have 2: (σ ⊨ ((f ∧ more);(power-chop-d f n))) =
           (∃ k. 0 ≤k ∧ k ≤ intlen (σ) ∧ k > 0 ∧
             (prefix k (σ) ⊨ f) ∧
             (suffix k (σ) ⊨ power-chop-d f 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-chop-d f n)
         ) =
         (∃ k. 0 ≤k ∧ k ≤ intlen (σ) ∧ k > 0 ∧
             (sub 0 k (σ) ⊨ f) ∧
             (suffix k (σ) ⊨ power-chop-d f n)
         )
  by (simp add: interval-sub-zero-prefix)
  have 4: (∃ k. 0 ≤k ∧ k ≤ intlen (σ) ∧ k > 0 ∧
             (sub 0 k (σ) ⊨ f) ∧
             (suffix k (σ) ⊨ power-chop-d f 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 Suc.hyps by auto
   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 auto
  qed
qed

lemma chopstar-eqv-power-chop:
  (σ ⊨ f⋆) = (∃ k. (σ ⊨ power-chop-d f k))
by (simp add: chopstar-d-def chopstar-eqv-power-chop-help)

lemma ChopstarEqvSem:
   (σ ⊨ (f⋆ =  (empty ∨  (f ∧ more); (f⋆))) )
using chopstar-eqv-power-chop
by (smt chop-d-def chop-power-eqv-sem unl-lift2)
```

3.6 Quantification over State (Flexible) Variables

The hidden state approach, as used in the embedding of TLA in Isabelle/HOL TLA embedding [4, 2], is used. Here [4, 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.

```
typedecl state

instance state :: world ..

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

Similar to [4, 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 [4, 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*] ⇒ *lift* ((*3*∃∃ -./ -) [*0*,*10*] *10*)

translations
-Eex v A == *Eex v*. *A*

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

syntax
-Aall :: [*idts*, *lift*] ⇒ *lift* ((*3*∀∀ -./ -) [*0*,*10*] *10*)

translations
-Aall v A == *Aall v*. *A*

3.8 Unlifting attributes and methods

The following is again from [4, 2] but adapted for our need.

lemma *int-eq-true*: $\vdash P \Longrightarrow \vdash P = \#True$
by *auto*

Attribute which unlifts an intensional formula

ML ⟪
```
fun unl-rewr ctxt thm =
  let
    val unl = (thm RS @{thm intD})
                        handle THM - => thm
    val rewr = rewrite-rule ctxt @{thms intensional-rews}
   in
    unl |> rewr
   end;
```
⟫
attribute-setup *unlifted* = ⟪
Scan.succeed (*Thm.rule-attribute* [] (*unl-rewr o Context.proof-of*))
⟫ *unlift intensional formulas*

attribute-setup *unlift-rule* = ⟪
Scan.succeed
(*Thm.rule-attribute* []
(*Context.proof-of* #> (*fn ctxt* => *Object-Logic.rulify ctxt o unl-rewr ctxt*)))
⟫ *unlift and rulify intensional formulas*

Attribute which turns an intensional formula into a rewrite rule. Formulas F that are not equalities are turned into $F \equiv \#True$.

ML ⟪

```
fun int-rewr thm =
  (thm RS @{thm inteq-reflection})
    handle THM - => ((thm RS @{thm int-eq-true}) RS @{thm inteq-reflection});
⟫
```

attribute-setup *simp-unl* = ⟪
    Attrib.add-del
      (*Thm.declaration-attribute*
        (*fn th* => *Simplifier.map-ss* (*Simplifier.add-simp* (*int-rewr th*))))
      (*K* (*NONE*, *NONE*)) — note only adding – removing is ignored
⟫ *add thm unlifted from rewrites from intensional formulas*

attribute-setup *int-rewrite* = ⟪ *Scan.succeed* (*Thm.rule-attribute* [] (*fn* - => *int-rewr*)) ⟫
  produce rewrites from intensional formulas

end

theory *ITL*
imports
  Semantics
begin

4 Axioms and Rules

The ITL axiom and proof rules are introduced (taken from [5]) together with the validity operation. The soundness of the rules and axioms are checked using the lemmas of Semantics.thy.

4.1 Rules

lemma *MP* :
 assumes $\vdash f \longrightarrow g$
        $\vdash f$
 shows $\vdash g$
using *assms*(*1*) *assms*(*2*) **by** *fastforce*

lemma *BoxGen* :
 assumes $\vdash f$
 shows $\vdash \Box f$
using *assms* **by** (*auto simp*: *always-defs*)

lemma *BiGen*:
 assumes $\vdash f$
 shows $\vdash$ *bi f*
using *assms* **by** (*auto simp*: *bi-defs*)

4.2 Axioms

lemma *ChopAssoc* :

$\vdash\ f\ ;\ (g\ ;\ h) =\ (f;g);h$
using *ChopAssocSem Valid-def* **by** *blast*

lemma *OrChopImp* :
$\vdash (\ f \lor g);h \ \longrightarrow\ f;h \lor g;h$
using *OrChopImpSem Valid-def* **by** *blast*

lemma *ChopOrImp* :
$\vdash\ f;(g \lor h) \longrightarrow\ f;g \lor\ f;h$
using *ChopOrImpSem Valid-def* **by** *blast*

lemma *EmptyChop* :
$\vdash\ empty\ ;\ f = f$
using *EmptyChopSem Valid-def* **by** *blast*

lemma *ChopEmpty* :
$\vdash f;empty = f$
using *ChopEmptySem Valid-def* **by** *blast*

lemma *StateImpBi* :
$\vdash init\ f \longrightarrow\ bi\ (init\ f)$
using *StateImpBiSem Valid-def* **by** *blast*

lemma *NextImpNotNextNot* :
$\vdash\ \bigcirc f \longrightarrow\ \neg\ (\bigcirc \neg f)$
using *NextImpNotNextNotSem Valid-def* **by** *blast*

lemma *BiBoxChopImpChop* :
$\vdash\ bi\ (\ f \longrightarrow f1) \land \Box(g \longrightarrow g1) \longrightarrow f;g \longrightarrow f1;g1$
using *BiBoxChopImpChopSem Valid-def* **by** *blast*

lemma *BoxInduct* :
$\vdash \Box\ (f \longrightarrow wnext\ f) \land f \longrightarrow \Box\ f$
using *BoxInductSem Valid-def* **by** *blast*

lemma *ChopstarEqv* :
$\vdash\ f^{\star} =\ (empty \lor\ (f \land more);\ f^{\star})$
using *ChopstarEqvSem Valid-def* **by** *blast*

4.3 Quantification

lemma *EExI* :
$\vdash F\ y \longrightarrow (\exists\exists\ x\ .\ F\ x)$
by (*simp add*: *exist-state-d-def Valid-def*, *auto*)

lemma *EExE*:
$[\![\bigwedge x.\ \vdash F\ x \longrightarrow G]\!] \Longrightarrow \vdash (\exists\exists\ x.\ F\ x) \longrightarrow G$
by (*metis* (*mono-tags*, *lifting*) *Valid-def exist-state-d-def unl-lift2*)

lemma *EExVal*:
$(w \models (\exists\exists\ x.\ F\ x)) =$
$(\exists\ x\ (val :: 'a\ interval).\ (\ (\ val = (map\ x\ w) \wedge\ (w \models F\ x))))$
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 *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 *ExEqvRule*:
assumes $\bigwedge x.\ \vdash (f\ x) = (g\ x)$
shows $\vdash (\exists\ x.\ f\ x) = (\exists\ x.\ g\ x)$
using *assms* **by** *fastforce*

4.4 Lemmas about *current-val*

lemma *current-const*: $\vdash \$(\#c) = \#c$
by (*auto simp*: *current-val-d-def*)

lemma *current-fun1*: $\vdash \$(f{<}x{>}) = f\ {<}\$x{>}$
by (*auto simp*: *current-val-d-def*)

lemma *current-fun2*: $\vdash \$(f{<}x,y{>}) = f\ {<}\$x,\$y{>}$
by (*auto simp*: *current-val-d-def*)

lemma *current-fun3*: $\vdash \$(f{<}x,y,z{>}) = f\ {<}\$x,\$y,\$z{>}$
by (*auto simp*: *current-val-d-def*)

lemma *current-fun4*: $\vdash \$(f{<}x,y,z,zz{>}) = f\ {<}\$x,\$y,\$z,\$zz{>}$
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-fun4 current-forall current-exists current-exists1*

lemmas *all-current-unl* = *all-current*[*THEN intD*]
lemmas *all-current-eq* = *all-current*[*THEN inteq-reflection*]

4.5 Lemmas about *next-val*

lemma *next-const*: $\vdash$ *more* $\longrightarrow$ (#*c*)\$ = #*c*
by (*auto simp*: *next-val-d-def more-defs*)

lemma *next-fun1*: $\vdash$ *more* $\longrightarrow$ *f*<*x*>\$ = *f*<*x*\$>
by (*auto simp*: *next-val-d-def more-defs*)

lemma *next-fun2*: $\vdash$ *more* $\longrightarrow$ *f*<*x*,*y*>\$ = *f* <*x*\$,*y*\$>
by (*auto simp*: *next-val-d-def more-defs*)

lemma *next-fun3*: $\vdash$ *more* $\longrightarrow$ *f*<*x*,*y*,*z*>\$ = *f* <*x*\$,*y*\$,*z*\$>
by (*auto simp*: *next-val-d-def more-defs*)

lemma *next-fun4*: $\vdash$ *more* $\longrightarrow$ *f*<*x*,*y*,*z*,*zz*>\$ = *f* <*x*\$,*y*\$,*z*\$,*zz*\$>
by (*auto simp*: *next-val-d-def more-defs*)

lemma *next-forall*: $\vdash$ *more* $\longrightarrow$ ($\forall$ *x*. *P x*)\$ = ($\forall$ *x*. (*P x*)\$)
by (*auto simp*: *next-val-d-def*)

lemma *next-exists*: $\vdash$ *more* $\longrightarrow$ ($\exists$ *x*. *P x*)\$ = ($\exists$ *x*. (*P x*)\$)
by (*auto simp*: *next-val-d-def*)

lemma *next-exists1*: $\vdash$ *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-fun4*
next-forall next-exists next-exists1

lemmas *all-next-unl* = *all-next*[*THEN intD*]

4.6 Lemmas about *fin-val*

lemma *fin-const*: $\vdash$!(#*c*) = #*c*
by (*auto simp*: *fin-val-d-def*)

lemma *fin-fun1*: $\vdash$!(*f*<*x*>) = *f* <!*x*>
by (*auto simp*: *fin-val-d-def*)

lemma *fin-fun2*: $\vdash$!(*f*<*x*,*y*>) = *f* <!*x*, !*y*>
by (*auto simp*: *fin-val-d-def*)

lemma *fin-fun3*: $\vdash$!(*f*<*x*,*y*,*z*>) = *f* <!*x*,!*y*,!*z*>
by (*auto simp*: *fin-val-d-def*)

lemma *fin-fun4*: $\vdash$!(*f*<*x*,*y*,*z*,*zz*>) = *f* <!*x*,!*y*,!*z*,!*zz*>
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-fun4*
fin-forall fin-exists fin-exists1

lemmas *all-fin-unl* = *all-fin*[*THEN intD*]
lemmas *all-fin-eq* = *all-fin*[*THEN inteq-reflection*]

4.7 Lemmas about *penult-val*

lemma *penult-const*: $\vdash more \longrightarrow (\#c)! = \#c$
by (*auto simp*: *penult-val-d-def more-defs*)

lemma *penult-fun1*: $\vdash more \longrightarrow f<x>! = f<x!>$
by (*auto simp*: *penult-val-d-def more-defs*)

lemma *penult-fun2*: $\vdash more \longrightarrow f<x,y>! = f\ <x!,y!>$
by (*auto simp*: *penult-val-d-def more-defs*)

lemma *penult-fun3*: $\vdash more \longrightarrow f<x,y,z>! = f\ <x!,y!,z!>$
by (*auto simp*: *penult-val-d-def more-defs*)

lemma *penult-fun4*: $\vdash more \longrightarrow f<x,y,z,zz>! = f\ <x!,y!,z!,zz!>$
by (*auto simp*: *penult-val-d-def more-defs*)

lemma *penult-forall*: $\vdash more \longrightarrow (\forall\ x.\ P\ x)! = (\forall\ x.\ (P\ x)!)$
by (*auto simp*: *penult-val-d-def*)

lemma *penult-exists*: $\vdash more \longrightarrow (\exists\ x.\ P\ x)! = (\exists\ x.\ (P\ x)!)$
by (*auto simp*: *penult-val-d-def*)

lemma *penult-exists1*: $\vdash more \longrightarrow (\exists!\ x.\ P\ x)! = (\exists!\ x.\ (P\ x)!)$
by (*auto simp*: *penult-val-d-def more-defs*)

lemmas *all-penult* = *penult-const penult-fun1 penult-fun2 penult-fun3 penult-fun4*
penult-forall penult-exists penult-exists1

lemmas *all-penult-unl* = *all-penult*[*THEN intD*]

4.8 Time reversal properties

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*:
$\vdash (f{<}x,y{>})^r = f{<}x^r,y^r{>}$
by (*auto simp*: *reverse-d-def*)

lemma *rev-fun3*:
$\vdash (f{<}x,y,z{>})^r = f{<}x^r,y^r,z^r{>}$
by (*auto simp*: *reverse-d-def*)

lemma *rev-fun4*:
$\vdash (f{<}x,y,z,zz{>})^r = f{<}x^r,y^r,z^r,zz^r{>}$
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*:
$\vdash more \longrightarrow (v\$)^r = (v!)$
by (*auto simp*: *more-defs interval-intrev-nth next-val-d-def penult-val-d-def reverse-d-def*)

lemma *rev-penult*:
$\vdash more \longrightarrow (v!)^r = (v\$)$
by (*auto simp*: *more-defs 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*:
$\vdash$ $(f;g)^r = (g^r;f^r)$
apply (*simp add*: *Valid-def reverse-d-def chop-d-def*)
using *interval-intrev-prefix interval-intrev-suffix*
by (*metis diff-diff-cancel diff-le-self*)

lemma *RMoreEqvMore*:
$\vdash$ *more*^{r} = *more*
apply (*simp add*: *Valid-def more-d-def next-d-def chop-d-def skip-d-def reverse-d-def*)
by (*simp add*: *interval-prefix-length*)

lemma *REmptyEqvEmpty*:
$\vdash$ *empty*^{r} = *empty*
by (*metis RMoreEqvMore empty-d-def int-eq rev-fun1*)

lemma *PowerChopCommute*:
$\vdash$ ((*f* $\wedge$ *more*);(*powerchop f n*)) = (*powerchop f n*);(*f* $\wedge$ *more*)
proof
(*induct n*)
case *0*
then show *?case* **using** *EmptyChopSem ChopEmptySem power-0 Valid-def* **by** (*metis inteq-reflection*)
next
case (*Suc n*)
then show *?case*
by (*metis ChopAssocSem intI inteq-reflection power-chop-d.power-Suc*)
qed

lemma *REqvRule*:
assumes $\vdash f = g$
shows $\vdash (f^r) = (g^r)$
using *assms*
using *inteq-reflection* **by** *force*

lemma *RevPowerChop*:
$\vdash$ (*powerchop f n*)^r = (*powerchop* (f^r) *n*)
proof
(*induct n*)
case *0*
then show *?case* **using** *REmptyEqvEmpty* **by** *auto*
next
case (*Suc n*)
then show *?case*
by (*metis PowerChopCommute RevChop RMoreEqvMore int-eq power-chop-d.power-Suc rev-fun2*)
qed

lemma *RevChopstar*:
$\vdash (f^\star)^r = (f^r)^\star$
proof −
 have *1*: $\vdash (f^\star) = (\exists\, n.\ powerchop\ f\ n)$
  by (*simp add*: *chopstar-eqv-power-chop Valid-def*)
 have *2*: $\vdash (f^\star)^r = (\exists\ n.\ powerchop\ f\ n)^r$
  using *REqvRule 1* **by** *blast*
 have *3*: $\vdash (\exists\ n.\ powerchop\ f\ n)^r = (\exists\ n.\ (powerchop\ f\ n)^r)$
  by (*simp add*: *rev-exists*)
 have *4*: $\vdash (\exists\ n.\ (powerchop\ f\ n)^r) = (\exists\ n.\ (powerchop\ (f^r)\ n))$
  by (*simp add*: *RevPowerChop ExEqvRule*)
 have *5*: $\vdash (\exists\ n.\ (powerchop\ (f^r)\ n)) = (f^r)^\star$
  by (*simp add*: *chopstar-eqv-power-chop Valid-def*)
 from *2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemmas *all-rev* = *rev-const rev-fun1 rev-fun2 rev-fun3 rev-fun4 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*]

end

theory *Theorems*
 imports
  ITL
begin

5 ITL theorems

We give the proofs of a list of ITL theorems. These proofs and theorems were from [6].

5.1 Propositional reasoning

This is a list of propositional logic theorems used in the proofs of the ITL theorems.

lemma *IfThenElseImp*:
$\vdash (if_i\ \ g\ \ then\ \ f\ \ else\ f1) \longrightarrow ((\ g \longrightarrow f) \wedge (\neg g \longrightarrow f1))$
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 $\vdash f \lor f1 \longrightarrow g$
using *assms*(1) *assms*(2) **by** *fastforce*

lemma *Prop03*:
assumes $\vdash f = (g \lor h)$
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 $\vdash f \longrightarrow h \lor g$
using *assms* **by** *auto*

lemma *Prop06*:
assumes $\vdash f = (g \lor h)$
$\vdash h = h1$
shows $\vdash f = (g \lor h1)$
using *assms*(1) *assms*(2) **by** *fastforce*

lemma *Prop07*:
assumes $\vdash f \longrightarrow g \lor h$
shows $\vdash f \land \neg g \longrightarrow h$
using *assms* **by** *auto*

lemma *Prop08*:
assumes $\vdash f \longrightarrow g \lor h$
$\vdash h \longrightarrow h1$
shows $\vdash f \longrightarrow g \lor h1$
using *assms*(1) *assms*(2) **by** *fastforce*

lemma *Prop09*:
assumes $\vdash f \land g \longrightarrow h$
shows $\vdash f \longrightarrow (g \longrightarrow h)$
using *assms* **by** *auto*

lemma *Prop10*:
assumes $\vdash f \longrightarrow g$
shows $\vdash f = (f \land g)$
using *assms* **by** *auto*

lemma *Prop11*:
$(\vdash f = f1) = ((\vdash f \longrightarrow f1) \land (\vdash f1 \longrightarrow f))$

by (*auto simp*: *Valid-def*)

lemma *Prop12*:
$(\vdash f \longrightarrow (f1 \wedge f2)) = ((\vdash f \longrightarrow f1) \wedge (\vdash f \longrightarrow f2))$
by (*auto simp*: *Valid-def*)

5.2 State formulas

The *init* operator denotes state formulas, i.e., ITL formula that only constrain the first state of an interval.

lemma *Initprop* :
$\vdash ((init\ f) \wedge (init\ g)) = init(f \wedge g)$
$\vdash (\neg\ (init\ f)) = init\ (\ \neg\ f\ )$
$\vdash ((init\ f) \vee (init\ g)) = init\ (f \vee g)$
$\vdash init\ \#True$
by (*auto simp*: *init-defs*)

lemma *Finprop* :
$\vdash ((\#True;(f \wedge empty)) \wedge (\#True;(g \wedge empty))) = (\#True;((f \wedge g) \wedge empty))$
$\vdash ((\#True;(f \wedge empty)) \vee (\#True;(g \wedge empty))) = (\#True;((f \vee g) \wedge empty))$
$\vdash (\#True;((\#True) \wedge empty))$
$\vdash \neg\ (\#True;(f \wedge empty)) = (\#True;(\neg f \wedge empty))$
by (*auto simp*: *finalt-defs*) (*simp add*: *chop-defs empty-defs interval-suffix-length*, *fastforce*)

5.3 Basic Theorems

lemma *BiChopImpChop* :
$\vdash bi\ (f \longrightarrow f1) \longrightarrow f;g \longrightarrow f1;g$
proof −
have *1*: $\vdash g \longrightarrow g$ **by** *auto*
hence *2*: $\vdash \Box\ (\ g \longrightarrow g)$ **by** (*rule BoxGen*)
have *3*: $\vdash bi\ (\ f \longrightarrow f1) \wedge \Box(g \longrightarrow g) \longrightarrow f;g \longrightarrow f1;g$ **by** (*rule BiBoxChopImpChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *AndChopA*:
$\vdash (f \wedge f1);g \longrightarrow f;g$
proof −
have *1*: $\vdash f \wedge f1 \longrightarrow f$ **by** *auto*
hence *2*: $\vdash bi\ (f \wedge f1 \longrightarrow f)$ **by** (*rule BiGen*)
have *3*: $\vdash bi\ (f \wedge f1 \longrightarrow f) \longrightarrow (f \wedge f1);g \longrightarrow f;g$ **by** (*rule BiChopImpChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *AndChopB*:
$\vdash (f \wedge f1);g \longrightarrow f1;g$
proof −
have *1*: $\vdash f \wedge f1 \longrightarrow f1$ **by** *auto*
hence *2*: $\vdash\ bi\ (f \wedge f1 \longrightarrow f1)$ **by** (*rule BiGen*)
have *3*: $\vdash\ bi\ (f \wedge f1 \longrightarrow f1) \longrightarrow (f \wedge f1);g \longrightarrow f1;g$ **by** (*rule BiChopImpChop*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*

qed

lemma *NextChop*:
 $\vdash (\bigcirc\ f);g\ \ = \bigcirc(f;g)$
proof −
 have *1*: $\vdash$ *skip*;$(f;g)\ \ = (skip;f);g$ **by** (*rule ChopAssoc*)
 show *?thesis* **by** (*metis 1 int-eq next-d-def*)
qed

lemma *BoxChopImpChop* :
 $\vdash\ \ \Box\ (g \longrightarrow g1) \longrightarrow f;g \longrightarrow f;g1$
 proof −
 have *1*: $\vdash g \longrightarrow g$ **by** *auto*
 hence *2*: $\vdash$ *bi* $(\ g \longrightarrow g)$ **by** (*rule BiGen*)
 have *3*: $\vdash$ *bi* $(\ f \longrightarrow f) \wedge \Box(g \longrightarrow g1) \longrightarrow f;g \longrightarrow f;g1$ **by** (*rule BiBoxChopImpChop*)
 from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *LeftChopImpChop*:
 assumes $\vdash f \longrightarrow f1$
 shows $\vdash f;g \longrightarrow f1;g$
proof −
 have *1*: $\vdash f \longrightarrow f1$ **using** *assms* **by** *auto*
 hence *2*: $\vdash$ *bi* $(f \longrightarrow f1)$ **by** (*rule BiGen*)
 have *3*: $\vdash$ *bi* $(f \longrightarrow f1) \longrightarrow f;g \longrightarrow f1;g$ **by** (*rule BiChopImpChop*)
 from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *RightChopImpChop*:
 assumes $\vdash g \longrightarrow g1$
 shows $\vdash f;g \longrightarrow f;g1$
proof −
 have *1*: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*
 hence *2*: $\vdash \Box\ (g \longrightarrow g1)$ **by** (*rule BoxGen*)
 have *3*: $\vdash \Box\ (g \longrightarrow g1) \longrightarrow f;g \longrightarrow f;g1$ **by** (*rule BoxChopImpChop*)
 from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *RightChopEqvChop*:
 assumes $\vdash g = g1$
 shows $\vdash (f;g) = (f;g1)$
proof −
 have *1*: $\vdash g = g1$ **using** *assms* **by** *auto*
 have *2*: $(\vdash g \longrightarrow g1) \Longrightarrow (\vdash f;g \longrightarrow f;g1)$ **by** (*rule RightChopImpChop*)
 have *3*: $(\vdash g1 \longrightarrow g) \Longrightarrow (\vdash f;g1 \longrightarrow f;g)$ **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*: ⊢ *g* ⟶ *g* ∨ *g1* **by** *auto*
 hence *2*: ⊢ *f* ;*g* ⟶ *f* ;(*g* ∨ *g1*) **by** (*rule RightChopImpChop*)
 have *3*: ⊢ *g1* ⟶ *g* ∨ *g1* **by** *auto*
 hence *4*: ⊢ *f* ;*g1* ⟶ *f* ;(*g* ∨ *g1*) **by** (*rule RightChopImpChop*)
 from *2 4* **show** *?thesis* **by** (*simp add*: *ChopOrImp int-iffI Prop02*)
qed

lemma *OrChopEqv*:
 ⊢ (*f* ∨ *f1*);*g* = (*f* ;*g* ∨ *f1* ;*g*)
proof −
 have *1*: ⊢ *f* ⟶ *f* ∨ *f1* **by** *auto*
 hence *2*: ⊢ *f* ;*g* ⟶ (*f*∨ *f1*);*g* **by** (*rule LeftChopImpChop*)
 have *3*: ⊢ *f1* ⟶ *f* ∨ *f1* **by** *auto*
 hence *4*: ⊢ *f1* ;*g* ⟶ (*f*∨ *f1*);*g* **by** (*rule LeftChopImpChop*)
 from *2 4* **show** *?thesis*
 by (*meson OrChopImp int-iffI Prop02*)
qed

lemma *OrChopImpRule*:
 assumes ⊢ *f* ⟶ *f1* ∨ *f2*
 shows ⊢ *f* ;*g* ⟶ (*f1* ;*g*) ∨ (*f2* ;*g*)
proof −
 have *1*: ⊢ *f* ⟶ *f1* ∨ *f2* **using** *assms* **by** *auto*
 hence *2*: ⊢ *f* ;*g* ⟶ (*f1* ∨ *f2*);*g* **by** (*rule LeftChopImpChop*)
 have *3*: ⊢ (*f1* ∨ *f2*); *g* = (*f1* ;*g* ∨ *f2* ;*g*) **by** (*rule OrChopEqv*)
 from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *LeftChopEqvChop*:
 assumes ⊢ *f* = *f1*
 shows ⊢ *f* ;*g* = (*f1* ;*g*)
proof −
 have *1*: ⊢ *f* = *f1* **using** *assms* **by** *auto*
 hence *2*: ⊢ *f* ⟶ *f1* **by** *auto*
 hence *3*: ⊢ *f* ;*g* ⟶ *f1* ;*g* **by** (*rule LeftChopImpChop*)
 from *1* **have** ⊢ *f1* ⟶ *f* **by** *auto*
 hence *4*: ⊢ *f1* ;*g* ⟶ *f* ;*g* **by** (*rule LeftChopImpChop*)
 from *3 4* **show** *?thesis* **by** (*simp add*: *int-iffI*)
qed

lemma *OrChopEqvRule*:
 assumes ⊢ *f* = (*f1* ∨ *f2*)
 shows ⊢ *f* ;*g* = ((*f1* ;*g*) ∨ (*f2* ;*g*))
proof −
 have *1*: ⊢ *f* = (*f1* ∨ *f2*) **using** *assms* **by** *auto*
 hence *2*: ⊢ *f* ;*g* = ((*f1* ∨ *f2*);*g*) **by** (*rule LeftChopEqvChop*)
 have *3*: ⊢ (*f1* ∨ *f2*);*g* = (*f1* ;*g* ∨ *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 $\vdash g \longrightarrow g1 \lor g2$
shows $\vdash f;g \longrightarrow (f;g1) \lor (f;g2)$
proof −
have *1*: $\vdash g \longrightarrow g1 \lor g2$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;g \longrightarrow f;(g1 \lor g2)$ **by** (*rule RightChopImpChop*)
have *3*: $\vdash f;(g1 \lor g2) = (f;g1 \lor f;g2)$ **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*: $\vdash \neg\, (f \longrightarrow g) = (f \land \neg\, g)$ **by** *auto*
hence *2*: $\vdash skip;\neg\, (f \longrightarrow g) = skip;(f \land \neg\, g)$ **by** (*rule RightChopEqvChop*)
have *3*: $\vdash f \longrightarrow g \lor (f \land \neg\, g)$ **by** *auto*
hence *4*: $\vdash skip;f \longrightarrow (skip;g) \lor (skip;(f \land \neg\, g))$ **by** (*rule ChopOrImpRule*)
hence *5*: $\vdash \neg\, (skip;(f \land \neg\, g)) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **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 *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*: $\vdash skip;(\#True;f) = ((skip;\#True);f)$ **by** (*rule ChopAssoc*)
hence *2*: $\vdash (skip;\#True);f = skip;(\#True;f)$ **by** *auto*
hence *3*: $\vdash (skip;\#True);f = \bigcirc(\Diamond f)$ **by** (*simp add*: *next-d-def sometimes-d-def*)
have *4*: $\vdash (skip;\#True);f \longrightarrow \Diamond\ f$ **by** (*rule ChopImpDiamond*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxImpNowAndWeakNext*:
$\vdash \Box\ f \longrightarrow (f \wedge wnext\ (\ \Box\ f)\ )$
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*: $\vdash \Box f \longrightarrow wnext\ (\ \Box\ f\ )$ **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*: $\vdash \Box(\neg\ g \longrightarrow \neg\ f) \longrightarrow (\#True;\neg g) \longrightarrow (\#True;\neg f)$ **by** (*rule BoxChopImpChop*)
have *5*: $\vdash (\#True;\neg g) \longrightarrow (\#True;\neg f)$ **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*:
$\vdash\ \Diamond\ empty$
proof −
have *1*: $\vdash \#True$ **by** *auto*
have *2*: $\vdash \#True;\ empty = \#True$ **by** (*rule ChopEmpty*)
have *3*: $\vdash \#True;\ empty$ **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*: $\vdash skip;f = skip;g$ **by** (*rule RightChopEqvChop*)
from *1* **show** *?thesis* **by** (*metis 2 next-d-def*)
qed

lemma *NextAndNextImpNextRule*:
assumes $\vdash (f \wedge g) \longrightarrow h$
shows $\vdash (\bigcirc f \wedge \bigcirc g) \longrightarrow \bigcirc h$
using *assms* **by** (*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 $\vdash wnext\ f = wnext\ g$
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 $\vdash f \longrightarrow g$
$\vdash$ *more* $\wedge f \longrightarrow \bigcirc f$
shows $\vdash f \longrightarrow \Box g$
proof −
have *1*: $\vdash$ *more* $\wedge f \longrightarrow \bigcirc f$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \longrightarrow$ (*empty* $\vee \bigcirc f$) **by** (*auto simp*: *next-defs empty-defs more-defs*)
hence *3*: $\vdash f \longrightarrow$ *wnext* f **by** (*auto simp*: *wnext-defs empty-defs next-defs*)
hence *4*: $\vdash \Box(f \longrightarrow$ *wnext* $f)$ **by** (*rule BoxGen*)
have *5*: $\vdash (\Box (f \longrightarrow$ *wnext* $f)) \wedge f \longrightarrow \Box f$ **by** (*rule BoxInduct*)
hence *6*: $\vdash (\Box (f \longrightarrow$ *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: ⊢ f ⟶ ○ f  using assms by auto
 hence  2: ⊢ f  ⟶ (more ∧ wnext f) by (auto simp: more-defs wnext-defs next-defs)
 hence  3: ⊢ f  ⟶ wnext f  by auto
 hence  4: ⊢ □(f  ⟶ wnext f) by (rule BoxGen)
 have   5: ⊢ □ (f  ⟶ wnext f) ∧ f  ⟶ □ f   by (rule BoxInduct)
 hence  6: ⊢ □ (f  ⟶ wnext f) ⟶ (f  ⟶ □f) by fastforce
 have   7: ⊢ f  ⟶ □f  using 4 6 MP by blast
 have   8: ⊢ □f  ⟶ f  by (rule BoxElim)
 have   9: ⊢ f = □ f  using 7 8 by fastforce
 have  10: ⊢ f  ⟶ more using 2 by auto
 hence 11: ⊢ □ f  ⟶ □ more by (rule ImpBoxRule)
 have  12: ⊢ ¬(□ more)  by (auto simp: always-defs more-defs)
 from 7 9 11 12 show ?thesis by fastforce
qed

lemma WnextEqvEmptyOrNext:
⊢ wnext f = (empty ∨ ○ f)
by (auto simp: empty-defs wnext-defs next-defs)

lemma NotEmptyAndNext:
⊢ ¬(empty ∧ ○ f)
by (auto simp: empty-defs next-defs)

lemma BoxEqvAndWnextBox:
⊢ □ f = (f ∧ wnext ( □ f))
proof -
 have 1: ⊢ □ f  ⟶ f ∧ wnext ( □ f)
     using BoxImpNowAndWeakNext by blast
 have 2: ⊢ f ∧ wnext ( □ f) ⟶ f
     by auto
 have 3: ⊢ more ∧ (f ∧ wnext ( □ f) ) ⟶ ○ (f ∧ wnext ( □ f) )
     using 1 NextImpNext WnextEqvEmptyOrNext empty-d-def int-iffD1
     by (metis Prop01 Prop05 Prop08)
 have 4: ⊢ f ∧ wnext ( □ f) ⟶ □ f
     using 2 3 BoxIntro by blast
 from 1 4 show ?thesis by fastforce
qed

lemma BoxEqvAndEmptyOrNextBox:
⊢ □f = (f ∧ (empty  ∨  ○(□ f)))
using BoxEqvAndWnextBox WnextEqvEmptyOrNext by (metis int-eq)

lemma BoxEqvBoxBox:
⊢ □f = □ (□ f)
using BoxGen BoxInduct
by (metis BoxImpNowAndWeakNext MP int-iffI Prop09 Prop12)

lemma BoxBoxImpBox:
⊢ □(□h) ⟶ □ 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*: $\vdash \Box \neg g = ((\neg g) \wedge wnext(\Box \neg g))$
  using *BoxEqvAndWnextBox* **by** *metis*
have *7*: $\vdash \bigcirc f \wedge \Box \neg g \longrightarrow \bigcirc f \wedge wnext(\Box \neg g)$
  using *6* **by** *auto*
have *8*: $\vdash f \wedge (\Box \neg g) \longrightarrow \bigcirc f \wedge wnext(\Box \neg g)$
  using *5 7* **using** *lift-imp-trans* **by** *blast*
hence *9*: $\vdash f \wedge (\Box \neg g) \longrightarrow more \wedge wnext\ f \wedge wnext(\Box \neg g)$
  by (*auto simp*: *always-defs more-defs next-defs wnext-defs*)
hence *10*: $\vdash f \wedge (\Box \neg g) \longrightarrow wnext\ f \wedge wnext(\Box \neg g)$
  by *auto*
hence *11*: $\vdash f \wedge (\Box \neg g) \longrightarrow wnext\ (f \wedge \Box \neg g)$
  by (*auto simp*: *wnext-defs always-defs next-defs*)
hence *12*: $\vdash \Box(f \wedge (\Box \neg g) \longrightarrow wnext\ (f \wedge \Box \neg g))$
  by (*rule BoxGen*)
have *13*: $\vdash \Box(f \wedge (\Box \neg g) \longrightarrow wnext\ (f \wedge \Box \neg g)) \wedge f \wedge (\Box \neg g) \longrightarrow \Box(f \wedge (\Box \neg g))$
  by (*rule BoxInduct*)
hence *14*: $\vdash \Box(f \wedge (\Box \neg g) \longrightarrow wnext\ (f \wedge \Box \neg g)) \longrightarrow ((f \wedge (\Box \neg g)) \longrightarrow \Box(f \wedge (\Box \neg g)))$
  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*: $\vdash (f \wedge (\Box \neg g)) \longrightarrow more$
  using *9* **by** *auto*
hence *19*: $\vdash \Box(f \wedge (\Box \neg g)) \longrightarrow \Box\ more$
  by (*rule ImpBoxRule*)
have *20*: $\vdash \neg(\Box\ more)$
  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* $\longrightarrow f$
$\vdash \bigcirc f \longrightarrow f$
shows $\vdash f$
proof −
have *1*: $\vdash$ *empty* $\longrightarrow f$ **using** *assms* **by** *auto*
have *2*: $\vdash \bigcirc f \longrightarrow f$ **using** *assms* **by** *blast*
have *3*: $\vdash$ (*empty* $\vee \bigcirc f$) $\longrightarrow f$ **using** *1 2* **by** *fastforce*
have *4*: $\vdash$ *wnext* $f =$ (*empty* $\vee \bigcirc f$) **by** (*rule WnextEqvEmptyOrNext*)
hence *5*: $\vdash$ *wnext* $f \longrightarrow f$ **using** *3* **by** *fastforce*
hence *6*: $\vdash \neg f \longrightarrow \neg$ (*wnext* f) **by** *auto*
hence *7*: $\vdash \neg f \longrightarrow \bigcirc(\neg\ f)$ **by** (*auto simp*: *wnext-d-def*)
hence *8*: $\vdash \neg\ \neg\ f$ **by** (*rule NextLoop*)
from *8* **show** *?thesis* **by** *auto*
qed

lemma *EmptyNextInductb*:
assumes $\vdash$ *empty* $\wedge f \longrightarrow g$
$\vdash \bigcirc(f \longrightarrow g) \wedge f \longrightarrow g$
shows $\vdash f \longrightarrow g$
proof −
have *1*: $\vdash$ *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*: $\vdash$ (*empty* $\vee \bigcirc(f \longrightarrow g)$) $\wedge f \longrightarrow g$ **using** *1 2* **by** *fastforce*
hence *4*: $\vdash$ *wnext* $(f \longrightarrow g) \wedge f \longrightarrow g$ **using** *WnextEqvEmptyOrNext* **by** *fastforce*
hence *5*: $\vdash$ *wnext* $(f \longrightarrow g) \longrightarrow (\ f \longrightarrow g)$ **by** *fastforce*
hence *6*: $\vdash \neg\ (\ f \longrightarrow g) \longrightarrow \neg$ (*wnext* $(f \longrightarrow g)$) **by** *fastforce*
hence *7*: $\vdash \neg\ (\ f \longrightarrow g) \longrightarrow \bigcirc\ (\ \neg(f \longrightarrow g))$ **by** (*simp add*: *wnext-d-def*)
hence *8*: $\vdash \neg\ \neg\ (\ f \longrightarrow g)$ **by** (*rule NextLoop*)
from *8* **show** *?thesis* **by** *auto*
qed

lemma *FinImpFin*:
assumes $\vdash f \longrightarrow g$
shows $\vdash$ *fin* $f \longrightarrow$ *fin* g
using *ImpBoxRule*[*of TEMP* (*empty* $\longrightarrow f$) *TEMP* (*empty* $\longrightarrow g$)] *assms fin-d-def*
by (*smt intI intensional-rews*(*3*) *inteq-reflection Prop10*)

lemma *FinEqvFin*:
assumes $\vdash f = g$
shows $\vdash$ *fin* $f =$ *fin* g
using *assms* **by** (*metis intensional-simps*(*1*) *inteq-reflection*)

lemma *FinAndFinImpFinRule*:
assumes $\vdash f \wedge g \longrightarrow h$

shows $\vdash$ *fin* $f \wedge$ *fin* $g \longrightarrow$ *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 $\vdash$ (*fin* $f \wedge$ *fin* $g) =$ *fin* h
using *assms*
by (*simp add*: *FinAndFinImpFinRule FinImpFin Prop11 Prop12*)

lemma *HaltEqvHalt*:
assumes $\vdash f = g$
shows $\vdash$ *halt* $f =$ *halt* g
proof $-$
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash$ (*empty* $= f$) $=$ (*empty* $= g$) **by** *auto*
hence *3*: $\vdash \Box$(*empty* $= f$) $= \Box$ (*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*: ⊢ (*init w*)⟶ *bi* (*init w*) **by** (*rule StateImpBi*)
 hence *2*: ⊢ (*init w*) ∧ (*f*; *g*) ⟶ *bi* (*init w*) ∧ (*f*; *g*) **by** *auto*
 have *3*: ⊢ *bi* (*init w*) ∧ (*f*; *g*) ⟶ ((*init w*) ∧ *f*); *g* **by** (*rule BiAndChopImport*)
 from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

5.4 Further Properties Di and Bi

lemma *ImpDi*:
⊢ *f* ⟶ *di f*
proof −
 have *1*: ⊢ *f*; *empty* = *f* **by** (*rule ChopEmpty*)
 have *2*: ⊢ *empty* ⟶ #*True* **by** *auto*
 hence *3*: ⊢ *f*; *empty* ⟶ *f*; #*True* **by** (*rule RightChopImpChop*)
 have *4* : ⊢ *f* ⟶ *f*; #*True* **using** *1 3* **by** *fastforce*
 from *4* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiState*:
⊢ *di* (*init w*) = (*init w*)
proof −
 have *0*: ⊢ (*init* ¬*w*) ⟶ *bi* (*init* ¬*w*) **using** *StateImpBi* **by** *fastforce*
 hence *1*: ⊢ ¬(*init w*) ⟶ *bi* ¬ (*init w*) **using** *Initprop*(*2*) **by** (*metis inteq-reflection*)
 hence *2*: ⊢ ¬ (*init w*)⟶ ¬ (*di* ¬ ¬ (*init w*)) **by** (*simp add*: *bi-d-def*)
 have *3*: ⊢ (¬ (*init w*) ⟶¬ (*di* ¬ ¬ (*init w*))) ⟶ (*di* ¬ ¬ (*init w*) ⟶ (*init w*)) **by** *auto*
 have *4*: ⊢ *di* ¬ ¬ (*init w*) ⟶ (*init w*) **using** *2 3 MP* **by** *blast*
 have *5*: ⊢ (*init w*) ⟶ ¬ ¬ (*init w*) **by** *auto*
 hence *6*: ⊢ *di* (*init w*) ⟶ *di* ¬ ¬ (*init w*) **by** (*rule DiImpDi*)
 have *7*: ⊢ *di* (*init w*) ⟶ (*init w*) **using** *6 4* **using** *lift-imp-trans* **by** *metis*
 have *8*: ⊢ (*init w*) ⟶ *di* (*init w*) **by** (*rule ImpDi*)
 from *7 8* **show** *?thesis* **by** *fastforce*
qed

lemma *StateChop*:
⊢ (*init w*); *f* ⟶ (*init w*)
using *DiState* **by** (*auto simp*: *di-defs init-defs chop-defs*)

lemma *StateChopExportA*:
⊢ ((*init w*) ∧ *f*); *g* ⟶ (*init w*)
using *DiState* **by** (*auto simp*: *init-defs chop-defs*)

lemma *StateAndChop*:
⊢ ((*init w*) ∧ *f*); *g* = ((*init w*) ∧ (*f*; *g*))
by (*simp add*: *AndChopB StateAndChopImport StateChopExportA Prop11 Prop12*)

lemma *StateAndChopImpChopRule*:
 assumes ⊢ (*init w*) ∧ *f* ⟶ *f1*
 shows ⊢ (*init w*) ∧ (*f*; *g*) ⟶ (*f1*; *g*)
proof −

have *1*: $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *f1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ ((*init w*) $\wedge$ *f*); *g* $\longrightarrow$ *f1*; *g* **by** (*rule LeftChopImpChop*)
have *3*: $\vdash$ ((*init w*) $\wedge$ *f*); *g* = ((*init w*) $\wedge$ (*f*; *g*)) **by** (*rule StateAndChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *StateImpChopEqvChop* :
assumes $\vdash$(*init w*) $\longrightarrow$ (*f* = *f1*)
shows $\vdash$ (*init w*) $\longrightarrow$ ((*f*; *g*) = (*f1*; *g*))
proof −
have *1*: $\vdash$ (*init w*) $\longrightarrow$ (*f*= *f1*) **using** *assms* **by** *auto*
hence *2*: $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *f1* **by** *auto*
hence *3*: $\vdash$ (*init w*) $\wedge$ (*f*; *g*) $\longrightarrow$ (*f1*; *g*) **by** (*rule StateAndChopImpChopRule*)
have *4*: $\vdash$ (*init w*) $\wedge$ *f1* $\longrightarrow$ *f* **using** *1* **by** *auto*
hence *5*: $\vdash$ (*init w*) $\wedge$ (*f1*; *g*) $\longrightarrow$ (*f*; *g*) **by** (*rule StateAndChopImpChopRule*)
from *3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopEqvStateAndChop*:
assumes $\vdash$ *f* = (*init w*) $\wedge$ *f1*
shows $\vdash$ (*f*; *g*) = ((*init w*) $\wedge$ (*f1*; *g*))
proof −
have *1*: $\vdash$ *f* = ((*init w*) $\wedge$ *f1*) **using** *assms* **by** *auto*
hence *2*: $\vdash$ *f*; *g* = (((*init w*) $\wedge$ *f1*); *g*) **by** (*rule LeftChopEqvChop*)
have *3*: $\vdash$ ((*init w*) $\wedge$ *f1*); *g* = ((*init w*) $\wedge$ (*f1*; *g*)) **by** (*rule StateAndChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DiIntro*:
$\vdash$ *f* $\longrightarrow$ *di f*
proof −
have *1*: $\vdash$ *f*; *empty* = *f* **by** (*rule ChopEmpty*)
have *2*: $\vdash$ *empty* $\longrightarrow$ #*True* **by** *auto*
hence *3*: $\vdash$ □(*empty* $\longrightarrow$ #*True*) **by** (*rule BoxGen*)
have *4*: $\vdash$ □(*empty* $\longrightarrow$ #*True*) $\longrightarrow$ (*f*; *empty* $\longrightarrow$ *f*; #*True*) **by** (*rule BoxChopImpChop*)
have *5*: $\vdash$ *f*; *empty* $\longrightarrow$ *f*; #*True* **using** *3 4 MP* **by** *fastforce*
hence *6*: $\vdash$ *f*; *empty* $\longrightarrow$ *di f* **by** (*simp add*: *di-d-def*)
from *1 6* **show** *?thesis* **by** *fastforce*
qed

lemma *BiElim*:
$\vdash$ *bi f* $\longrightarrow$ *f*
proof −
have *1*: $\vdash$ ¬ *f* $\longrightarrow$ *di* ¬ *f* **by** (*rule DiIntro*)
have *2*: $\vdash$ (¬ *f* $\longrightarrow$ *di* ¬ *f*) $\longrightarrow$ (¬ (*di* ¬ *f*) $\longrightarrow$ *f*) **by** *auto*
have *3*: $\vdash$ ¬ (*di* ¬ *f*) $\longrightarrow$ *f* **using** *1 2 MP* **by** *blast*
from *3* **show** *?thesis* **by** (*metis bi-d-def*)
qed

lemma *BiContraPosImpDist*:

⊢ *bi* (¬ *g* ⟶ ¬ *f*) ⟶ (*bi f*) ⟶ (*bi g*)
proof −
have *1*: ⊢ *bi* (¬ *g* ⟶ ¬ *f*) ⟶ (*di* ¬ *g*) ⟶ (*di* ¬ *f*) **by** (*rule BiImpDiImpDi*)
hence *2*: ⊢ *bi* (¬ *g* ⟶ ¬ *f*) ⟶ (¬ (*di* ¬ *f*)) ⟶ (¬ (*di* ¬ *g*)) **by** *auto*
from *2* **show** *?thesis* **by** (*metis bi-d-def*)
qed

lemma *BiImpDist*:
⊢ *bi* (*f* ⟶ *g*) ⟶ (*bi f*) ⟶ (*bi g*)
proof −
have *1*: ⊢ (*f* ⟶ *g*) ⟶ (¬ *g* ⟶ ¬ *f*) **by** *auto*
hence *2*: ⊢ ¬ (¬ *g* ⟶ ¬ *f*) ⟶ ¬ (*f* ⟶ *g*) **by** *auto*
hence *3*: ⊢ *bi* (¬ (¬ *g* ⟶ ¬ *f*) ⟶ ¬ (*f* ⟶ *g*)) **by** (*rule BiGen*)
have *4*: ⊢ *bi* (¬ (¬ *g* ⟶ ¬ *f*) ⟶ ¬ (*f* ⟶ *g*))
⟶
bi (*f* ⟶ *g*) ⟶ *bi* (¬ *g* ⟶ ¬ *f*) **by** (*rule BiContraPosImpDist*)
have *5*: ⊢ *bi* (*f* ⟶ *g*) ⟶ *bi* (¬ *g* ⟶ ¬ *f*) **using** *3 4 MP* **by** *blast*
have *6*: ⊢ *bi* (¬ *g* ⟶ ¬ *f*) ⟶ (*bi f*) ⟶ (*bi g*) **by** (*rule BiContraPosImpDist*)
from *5 6* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *IfChopEqvRule*:
assumes ⊢ *f* = *if*$_i$ (*init w*) *then f1 else f2*
shows ⊢ *f*; *g* = *if*$_i$ (*init w*) *then* (*f1*; *g*) *else* (*f2*; *g*)
proof −
have *1*: ⊢ *f* = *if*$_i$ (*init w*) *then f1 else f2*
using *assms* **by** *auto*
hence *2*: ⊢ *f* = (((*init w*) ∧ *f1*) ∨ ((*init* ¬ *w*) ∧ *f2*))
by (*simp add*: *ifthenelse-d-def init-defs Valid-def*)
hence *3*: ⊢ *f*; *g* = (((*init w*) ∧ *f1*); *g* ∨ ((*init* ¬ *w*) ∧ *f2*); *g*)
by (*rule OrChopEqvRule*)
have *4*: ⊢ ((*init w*) ∧ *f1*); *g* = ((*init w*) ∧ (*f1*; *g*))
by (*rule StateAndChop*)
have *5*: ⊢ ((*init* ¬ *w*) ∧ *f2*); *g* = ((*init* ¬ *w*) ∧ (*f2*; *g*))
by (*rule StateAndChop*)
have *6*: ⊢ *f*; *g* = (((*init w*) ∧ *f1*; *g*) ∨ ((*init* ¬ *w*) ∧ *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 ⊢ *g* = (*g1* ∨ *g2*)
shows ⊢ *f*; *g* = ((*f*; *g1*) ∨ (*f*; *g2*))
proof −
have *1*: ⊢ *g* = (*g1* ∨ *g2*) **using** *assms* **by** *auto*
hence *2*: ⊢ *f*; *g* = (*f*; (*g1* ∨ *g2*)) **by** (*rule RightChopEqvChop*)
have *3*: ⊢ *f*; (*g1* ∨ *g2*) = (*f*; *g1* ∨ *f*; *g2*) **by** (*rule ChopOrEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrChopEqv*:
$\vdash$ (*empty* $\vee$ f); g = (g $\vee$ (f; g))
proof –
have *1*: $\vdash$ (*empty* $\vee$ f); g = ((*empty* ; g) $\vee$ (f; g)) **by** (*rule OrChopEqv*)
have *2*: $\vdash$ *empty* ; g = g **by** (*rule EmptyChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextChopEqv*:
$\vdash$ (*empty* $\vee$ $\bigcirc$ f); g = (g $\vee$ $\bigcirc$(f; g))
proof –
have *1*: $\vdash$ (*empty* $\vee$ $\bigcirc$ f); g = (g $\vee$ (($\bigcirc$ f); g)) **by** (*rule EmptyOrChopEqv*)
have *2*: $\vdash$ ($\bigcirc$ f); g = $\bigcirc$(f; g) **by** (*rule NextChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrChopImpRule*:
assumes $\vdash$ f $\longrightarrow$ *empty* $\vee$ *f1*
shows $\vdash$ f; g $\longrightarrow$ g $\vee$ (*f1*; g)
proof –
have *1*: $\vdash$ f $\longrightarrow$ *empty* $\vee$ *f1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ f; g $\longrightarrow$ (*empty* $\vee$ *f1*); g **by** (*rule LeftChopImpChop*)
have *3*: $\vdash$ (*empty* $\vee$ *f1*); g = (g $\vee$ (*f1*; g)) **by** (*rule EmptyOrChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrChopEqvRule*:
assumes $\vdash$ f = (*empty* $\vee$ *f1*)
shows $\vdash$ f; g = (g $\vee$ (*f1*; g))
proof –
have *1*: $\vdash$ f = (*empty* $\vee$ *f1*) **using** *assms* **by** *auto*
hence *2*: $\vdash$ f; g = ((*empty* $\vee$ *f1*); g) **by** (*rule LeftChopEqvChop*)
have *3*: $\vdash$ (*empty* $\vee$ *f1*); g = (g $\vee$ (*f1*; g)) **by** (*rule EmptyOrChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextChopImpRule*:
assumes $\vdash$ f $\longrightarrow$ *empty* $\vee$ $\bigcirc$ *f1*
shows $\vdash$ f; g $\longrightarrow$ g $\vee$ $\bigcirc$(*f1*; g)
proof –
have *1*: $\vdash$ f $\longrightarrow$ *empty* $\vee$ $\bigcirc$ *f1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ f; g $\longrightarrow$ (*empty* $\vee$ $\bigcirc$ *f1*); g **by** (*rule LeftChopImpChop*)
have *3*: $\vdash$ (*empty* $\vee$ $\bigcirc$ *f1*); g = (g $\vee$ $\bigcirc$(*f1*; g)) **by** (*rule EmptyOrNextChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *EmptyOrNextChopEqvRule*:
assumes $\vdash$ f = (*empty* $\vee$ $\bigcirc$ *f1*)
shows $\vdash$ f; g = (g $\vee$ $\bigcirc$(*f1*; g))
proof –

have *1*: $\vdash f = (empty \ \vee \ \bigcirc f1)$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;\ g = ((\ empty \ \vee \ \bigcirc f1);\ g)$ **by** (*rule LeftChopEqvChop*)
have *3*: $\vdash (\ empty \ \vee \ \bigcirc f1);\ g = (g \vee \ \bigcirc(f1;\ g))$ **by** (*rule EmptyOrNextChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopEmptyOrImpRule*:
assumes $\vdash \ g \longrightarrow \ empty \ \vee \ g1$
shows $\vdash f;\ g \longrightarrow f \vee \ (f;\ g1)$
proof −
have *1*: $\vdash g \longrightarrow \ empty \ \vee \ g1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;\ g \longrightarrow (f;\ \ empty\ ) \vee \ (f;\ g1)$ **by** (*rule ChopOrImpRule*)
have *3*: $\vdash f;\ \ empty \ = f$ **by** (*rule ChopEmpty*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *StateAndEmptyImpBoxState*:
$\vdash (init\ w) \wedge empty \longrightarrow \Box\ (init\ w)$
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*:
$\vdash \ \ \Box\ (init\ w);\ \Box\ (init\ w) = \Box\ (init\ w)$
proof −
have *1*: $\vdash (\Box\ (init\ w)) = ((init\ w) \wedge (\ empty \ \vee \ \bigcirc(\Box\ (init\ w))))$
by (*rule BoxEqvAndEmptyOrNextBox*)
hence *2*: $\vdash (\Box\ (init\ w);\ \Box\ (init\ w)) =$
$((init\ w) \wedge ((\ empty \ \vee \ \bigcirc(\Box\ (init\ w)));\ \Box\ (init\ w)))$
by (*metis StateAndChop inteq-reflection*)
have *3*: $\vdash ((\ empty \ \vee \ \bigcirc(\Box\ (init\ w)));\ \Box\ (init\ w)) =$
$(\Box\ (init\ w) \vee \ \bigcirc(\Box\ (init\ w);\ \Box\ (init\ w)))$
by (*rule EmptyOrNextChopEqv*)
have *4*: $\vdash (\Box\ (init\ w);\ \Box\ (init\ w)) =$
$((init\ w) \wedge (\Box\ (init\ w) \vee \ \bigcirc(\Box\ (init\ w);\ \Box\ (init\ w))))$
using *2 3* **by** *fastforce*
have *5*: $\vdash \neg\ (\Box\ (init\ w)) \longrightarrow \neg\ \ (init\ w) \vee \ \neg(\ \bigcirc(\Box\ (init\ w)))$
by (*rule NotBoxImpNotOrNotNextBox*)

have *6*: $\vdash$ ($\Box$ (*init w*); $\Box$ (*init w*)) $\wedge$ $\neg$($\Box$ (*init w*)) $\longrightarrow$
$\bigcirc$($\Box$ (*init w*); $\Box$ (*init w*)) $\wedge$ $\neg$($\bigcirc$($\Box$ (*init w*)))
using *4 5* **by** *fastforce*
hence *7*: $\vdash$ $\Box$ (*init w*); $\Box$ (*init w*) $\longrightarrow$ $\Box$ (*init w*)
by (*rule NextContra*)
have *11*: $\vdash$ $\Box$ (*init w*) = ((*init w*) $\wedge$ $\Box$ (*init w*))
by (*rule BoxEqvAndBox*)
have *12*: $\vdash$ *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 *7 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 *TrueChopEqvDiamond*:
$\vdash$ #*True*; *f* = $\Diamond$ *f*
by (*simp add*: *sometimes-d-def*)

5.5 Properties of Da and Ba

lemma *DaEqvDtDi*:
$\vdash$ *da f* = $\Diamond$ (*di f*)
proof –

have *1*: $\vdash$ #*True*; (*f*; #*True*) = #*True*; (*f*; #*True*) **by** *auto*
hence *2*: $\vdash$ #*True*; (*f*; #*True*) = #*True*; *di f* **by** (*simp add*: *di-d-def*)
have *3*: $\vdash$ #*True*; *di f* = $\diamond$(*di f*) **by** (*rule TrueChopEqvDiamond*)
have *4*: $\vdash$ #*True*; (*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*: $\vdash$ #*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*:
$\vdash$ $\diamond$ ¬ *f* = ¬ ($\square$ *f*)
by (*simp add*: *always-d-def*)

lemma *BaEqvBiBt*:
$\vdash$ *ba f* = *bi*($\square$ *f*)
proof −
have *1*: $\vdash$ *da* ¬ *f* = *di*($\diamond$ ¬ *f*) **by** (*rule DaEqvDiDt*)
have *2*: $\vdash$ $\diamond$ ¬ *f* = ¬($\square$ *f*) **by** (*rule DiamondNotEqvNotBox*)
hence *3*: $\vdash$ *di* ($\diamond$¬ *f*) = *di* ¬ ($\square$ *f*) **by** (*rule DiEqvDi*)
have *4*: $\vdash$ *da* ¬ *f* = *di* ¬($\square$ *f*) **using** *1 3* **by** *fastforce*
hence *5*: $\vdash$ ¬ (*da* ¬ *f*) = ¬ (*di* ¬($\square$ *f*)) **by** *auto*
hence *6*: $\vdash$ ¬ (*da* ¬ *f*) = *bi*($\square$ *f*) **by** (*simp add*: *bi-d-def*)
from *6* **show** *?thesis* **by** (*simp add*: *ba-d-def*)
qed

lemma *DiNotEqvNotBi*:
$\vdash$ *di* ¬ *f* = ¬(*bi f*)
proof −
have *1*: $\vdash$ *bi f* = ¬ (*di* ¬ *f*) **by** (*simp add*: *bi-d-def*)
from *1* **show** *?thesis* **by** *auto*
qed

lemma *NotDiamondNotEqvBox*:
$\vdash$ ¬ ($\diamond$¬ *f*) = $\square$ *f*
by (*simp add*: *always-d-def*)

lemma *BaEqvBtBi*:

$\vdash$ *ba* *f* = $\Box$ (*bi* *f*)
proof −
 have *1*: $\vdash$ *da* ¬ *f* = ◇ (*di* ¬ *f*) **by** (*rule DaEqvDtDi*)
 have *2*: $\vdash$ *di* ¬ *f* = ¬ (*bi* *f*) **by** (*rule DiNotEqvNotBi*)
 hence *3*: $\vdash$ ◇ (*di* ¬ *f*) = ◇¬ (*bi* *f*) **by** (*rule DiamondEqvDiamond*)
 have *4*: $\vdash$ ¬ (◇¬ (*bi* *f*)) = $\Box$(*bi* *f*) **by** (*rule NotDiamondNotEqvBox*)
 have *5*: $\vdash$ ¬ (*da* ¬ *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* ⟶ *bi* *f*
proof −
 have *1*: $\vdash$ *ba* *f* = $\Box$(*bi* *f*) **by** (*rule BaEqvBtBi*)
 have *2*: $\vdash$ $\Box$(*bi* *f*) ⟶ *bi* *f* **by** (*rule BoxElim*)
 from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *BaImpBt*:
$\vdash$ *ba* *f* ⟶ $\Box$ *f*
proof −
 have *1*: $\vdash$ *ba* *f* = *bi*($\Box$ *f*) **by** (*rule BaEqvBiBt*)
 have *2*: $\vdash$ *bi*($\Box$ *f*) ⟶ $\Box$ *f* **by** (*rule BiElim*)
 from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

lemma *DiamondImpDa*:
$\vdash$ ◇ *f* ⟶ *da* *f*
by (*metis DiIntro DiamondImpDiamond da-d-def di-d-def sometimes-d-def*)

lemma *DiImpDa*:
$\vdash$ *di* *f* $\longrightarrow$ *da* *f*
by (*metis NowImpDiamond da-d-def di-d-def sometimes-d-def*)

lemma *BoxAndChopImport*:
$\vdash$ $\Box\, h \wedge f;\, g \longrightarrow f;\, (h \wedge g)$
proof –
have *1*: $\vdash h \longrightarrow g \longrightarrow (h \wedge g)$ **by** *auto*
hence *2*: $\vdash \Box\, h \longrightarrow \Box(g \longrightarrow (h \wedge g))$ **by** (*rule ImpBoxRule*)
have *3*: $\vdash \Box(g \longrightarrow (h \wedge g)) \longrightarrow f;\, g \longrightarrow f;\, (h \wedge g)$ **by** (*rule BoxChopImpChop*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndChopImport*:
$\vdash$ *ba* $f \wedge (g;\, g1) \longrightarrow (f \wedge g);\, (f \wedge g1)$
proof –
have *1*: $\vdash$ *ba* $f \longrightarrow$ *bi* f **by** (*rule BaImpBi*)
have *2*: $\vdash$ *bi* $f \wedge (g;\, g1) \longrightarrow (f \wedge g);\, g1$ **by** (*rule BiAndChopImport*)
have *3*: $\vdash$ *ba* $f \longrightarrow \Box\, f$ **by** (*rule BaImpBt*)
have *4*: $\vdash \Box\, f \wedge (f \wedge g);\, g1 \longrightarrow (f \wedge g);\, (f \wedge g1)$ **by** (*rule BoxAndChopImport*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopAndCommute*:
$\vdash$ $f;\, (g \wedge g1) = f;\, (g1 \wedge g)$
proof –
have *1*: $\vdash (g \wedge g1) = (g1 \wedge g)$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule RightChopEqvChop*)
qed

lemma *ChopAndA*:
$\vdash$ $f;\, (g \wedge g1) \longrightarrow f;\, g$
proof –
have *1*: $\vdash (g \wedge g1) \longrightarrow g$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule RightChopImpChop*)
qed

lemma *ChopAndB*:
$\vdash$ $f;\, (g \wedge g1) \longrightarrow f;\, g1$
proof –
have *1*: $\vdash (g \wedge g1) \longrightarrow g1$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule RightChopImpChop*)
qed

lemma *BoxStateAndChopEqvChop*:
$\vdash$ $(\Box\, (init\ w) \wedge (f;\, g)) = ((\Box\, (init\ w) \wedge f);\, (\Box\, (init\ w) \wedge g))$
proof –
have *1*: $\vdash \Box\, (init\ w) =$ *ba*$(\ \Box\, (init\ w))$
by (*rule BoxStateEqvBaBoxState*)
have *2*: $\vdash$ *ba*$(\ \Box\, (init\ w)) \wedge (f;\, g) \longrightarrow (\Box\, (init\ w) \wedge f);\, (\Box\, (init\ w) \wedge g)$

by (*rule BaAndChopImport*)
have *3*: $\vdash \Box$ (*init w*) $\wedge$ (*f*; *g*) $\longrightarrow$ ($\Box$ (*init w*) $\wedge$ *f*); ($\Box$ (*init w*) $\wedge$ *g*)
using *1 2* **by** *fastforce*
have *11*: $\vdash$ ($\Box$ (*init w*) $\wedge$ *f*); ($\Box$ (*init w*) $\wedge$ *g*) $\longrightarrow$ ($\Box$ (*init w*)); ($\Box$ (*init w*) $\wedge$ *g*)
by (*rule AndChopA*)
have *12*: $\vdash$ ($\Box$ (*init w*)); ($\Box$ (*init w*) $\wedge$ *g*) $\longrightarrow$ ($\Box$ (*init w*)); ($\Box$ (*init w*))
by (*rule ChopAndA*)
have *13*: $\vdash$ ($\Box$ (*init w*)); ($\Box$ (*init w*)) = $\Box$ (*init w*)
by (*rule BoxStateChopBoxEqvBox*)
have *14*: $\vdash$ ($\Box$ (*init w*) $\wedge$ *f*); ($\Box$ (*init w*) $\wedge$ *g*) $\longrightarrow$ *f*; ($\Box$ (*init w*) $\wedge$ *g*)
by (*rule AndChopB*)
have *15*: $\vdash$ *f*; ($\Box$ (*init w*) $\wedge$ *g*) $\longrightarrow$ *f*; *g*
by (*rule ChopAndB*)
have *16*: $\vdash$ ($\Box$ (*init w*) $\wedge$ *f*); ($\Box$ (*init w*) $\wedge$ *g*) $\longrightarrow$ $\Box$ (*init w*) $\wedge$ (*f*; *g*)
using *11 12 13 14 15* **by** *fastforce*
from *3 16* **show** *?thesis* **by** *fastforce*
qed

lemma *DiEqvNotBiNot*:
$\vdash$ *di f* = $\neg$(*bi* $\neg$ *f*)
proof –
have *1*: $\vdash$ *bi* $\neg$ *f* = $\neg$ (*di* $\neg$ $\neg$ *f*) **by** (*simp add*: *bi-d-def*)
hence *2*: $\vdash$ *di* $\neg$ $\neg$ *f* = $\neg$(*bi* $\neg$ *f*) **by** *auto*
have *3*: $\vdash$ *f* = $\neg$ $\neg$ *f* **by** *auto*
hence *4*: $\vdash$ *di f* = *di* $\neg$ $\neg$ *f* **by** (*rule DiEqvDi*)
from *2 4* **show** *?thesis* **by** *auto*
qed

lemma *ChopAndBoxImport*:
$\vdash$ *f*; *g* $\wedge$ $\Box$ *h* $\longrightarrow$ *f*; (*g* $\wedge$ *h*)
proof –
have *1*: $\vdash$ $\Box$ *h* $\wedge$ *f*; *g* $\longrightarrow$ *f*; (*h* $\wedge$ *g*) **by** (*rule BoxAndChopImport*)
have *2*: $\vdash$ *f*; (*h* $\wedge$ *g*) = *f*; (*g* $\wedge$ *h*) **by** (*rule ChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *AndChopAndCommute*:
$\vdash$ (*f* $\wedge$ *g*); (*f1* $\wedge$ *g1*) = (*g* $\wedge$ *f*); (*g1* $\wedge$ *f1*)
proof –
have *1*: $\vdash$ (*f* $\wedge$ *g*); (*f1* $\wedge$ *g1*) = (*g* $\wedge$ *f*); (*f1* $\wedge$ *g1*) **by** (*rule AndChopCommute*)
have *2*: $\vdash$ (*g* $\wedge$ *f*); (*f1* $\wedge$ *g1*) = (*g* $\wedge$ *f*); (*g1* $\wedge$ *f1*) **by** (*rule ChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopImpChop*:
assumes $\vdash$ *f* $\longrightarrow$ *f1* $\vdash$ *g* $\longrightarrow$ *g1*
shows $\vdash$ *f*; *g* $\longrightarrow$ *f1*; *g1*
proof –
have *1*: $\vdash$ *f* $\longrightarrow$ *f1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ *f*; *g* $\longrightarrow$ *f1*; *g* **by** (*rule LeftChopImpChop*)

have *3*: $\vdash g \longrightarrow g1$ **using** *assms* **by** *auto*
hence *4*: $\vdash f1;\ g \longrightarrow f1;\ g1$ **by** (*rule RightChopImpChop*)
from *2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopEqvChop*:
assumes $\vdash\ f = f1\ \vdash g = g1$
shows $\vdash f;\ g = f1;\ g1$
proof −
have *1*: $\vdash f = f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;\ g = f1;\ g$ **by** (*rule LeftChopEqvChop*)
have *3*: $\vdash g = g1$ **using** *assms* **by** *auto*
hence *4*: $\vdash f1;\ g = f1;\ g1$ **by** (*rule RightChopEqvChop*)
from *2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxImpBoxImpBox*:
$\vdash \Box\ h \longrightarrow \Box(g \longrightarrow \Box\ h \wedge g\ )$
proof −
have *1*: $\vdash \Box\ h \longrightarrow (g \longrightarrow \Box\ h \wedge g\ )$ **by** *auto*
hence *2*: $\vdash \Box(\Box\ h) \longrightarrow \Box(g \longrightarrow \Box\ h \wedge g\ )$ **by** (*rule ImpBoxRule*)
have *3*: $\vdash \Box\ h = \Box(\Box h)$ **by** (*rule BoxEqvBoxBox*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxChopImpChopBox*:
$\vdash\ \Box\ h \longrightarrow f;\ g \longrightarrow f;\ (\Box\ h \wedge g)$
proof −
have *1*: $\vdash \Box\ h \longrightarrow \Box(g \longrightarrow \Box\ h \wedge g\ )$ **by** (*rule BoxImpBoxImpBox*)
have *2*: $\vdash \Box(g \longrightarrow \Box\ h \wedge g\ ) \longrightarrow f;\ g \longrightarrow f;\ (\Box\ h \wedge g)$ **by** (*rule BoxChopImpChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotChopEqvYieldsNot*:
$\vdash\ \neg\ (f;\ g) = f$ *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*:

$\vdash \quad di(\bigcirc f) = \bigcirc (di \ \ f)$
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*:
$\vdash \quad di(\bigcirc (init\ w)) = \bigcirc (init\ w)$
proof –
 have *1*: $\vdash \ di(\bigcirc (init\ w)) = \bigcirc(\ di \ \ (init\ w))$ **by** (*rule DiNext*)
 have *2*: $\vdash \ di \ \ (init\ w) = (init\ w)$ **by** (*rule DiState*)
 hence *3*: $\vdash \bigcirc(\ di \ \ (init\ w)) = \bigcirc (init\ w)$ **by** (*rule NextEqvNext*)
 from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *StateImpBiGen*:
 assumes $\vdash \quad (init\ w) \longrightarrow f$
 shows $\vdash (init\ w) \longrightarrow bi \ \ f$
proof –
 have *1*: $\vdash (init\ w) \longrightarrow f$ **using** *assms* **by** *auto*
 hence *2*: $\vdash \neg \ \ f \longrightarrow \neg \ \ (init\ w)$ **by** *auto*
 hence *3*: $\vdash \ di \neg \ \ f \longrightarrow \ \ di \neg \ \ (init\ w)$ **by** (*rule DiImpDi*)
 hence *4*: $\vdash \ di \neg \ \ f \longrightarrow \ \ di \ \ (init\ \neg w)$ **by** (*metis Initprop*(*2*) *inteq-reflection*)
 have *5*: $\vdash \ di\ (init\ \neg\ w) = \ \ (init\ \neg\ w)$ **by** (*rule DiState*)
 have *6*: $\vdash \ di \neg \ \ f \longrightarrow \neg \ \ (init\ w)$ **using** *4 5* **using** *Initprop*(*2*) **by** *fastforce*
 hence *7*: $\vdash (init\ w) \longrightarrow \neg\ (\ di \neg \ \ f)$ **by** *auto*
 from *7* **show** *?thesis* **by** (*simp add*: *bi-d-def*)
qed

lemma *ChopAndNotChopImp*:
$\vdash \quad f; g \wedge \neg\ (f; g1) \longrightarrow f; (g \wedge \neg \ \ g1)$
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*:
$\vdash \quad f; g \wedge f\ yields \ \ g1 \longrightarrow f; (g \wedge g1)$
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*:
$\vdash$ $(f \vee f1)$ *yields* $g = ((f$ *yields* $g) \wedge (f1$ *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 $\vdash (f1$ *yields* $g) \longrightarrow (f$ *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 $\vdash (f$ *yields* $g) = (f1$ *yields* $g)$
proof −
have *1*: $\vdash f = f1$ **using** *assms* **by** *auto*
hence *2*: $\vdash f; \neg g = f1; \neg g$ **by** (*rule LeftChopEqvChop*)
hence *3*: $\vdash \neg (f; \neg g) = \neg (f1; \neg g)$ **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

5.6 Properties of Fin

lemma *FinEqvTrueChopAndEmpty*:
$\vdash$ *fin* $f = \#True;(f \wedge$ *empty*$)$
proof −
have *1*: $\vdash$*fin* $f = \Box($*empty* $\longrightarrow f)$
by (*simp add*: *fin-d-def*)
have *2*: $\vdash \Box($*empty* $\longrightarrow f) = \neg(\Diamond(\neg($*empty* $\longrightarrow f)))$
by (*simp add*: *always-d-def*)
have *3*: $\vdash (\neg($*empty* $\longrightarrow f)) = (\neg f \wedge$ *empty*$)$
by *auto*
hence *4*: $\vdash \Diamond(\neg($*empty* $\longrightarrow f)) = \Diamond(\neg f \wedge$ *empty*$)$
using *DiamondEqvDiamond* **by** *blast*

hence *5*: $\vdash \neg(\Diamond(\neg(\mathit{empty} \longrightarrow f))) = \neg(\Diamond(\neg f \wedge \mathit{empty}))$
  by *auto*
have *6*: $\vdash \neg(\Diamond(\neg f \wedge \mathit{empty})) = \#\mathit{True};(f \wedge \mathit{empty})$
  using *Finprop*(*4*) *sometimes-d-def*
  by (*metis* (*no-types*, *lifting*) *TrueW int-eq intensional-simps*(*3*)
    intensional-simps(*6*) *intensional-simps*(*7*))
from *1 2 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *DiamondFin*:
$\vdash \Diamond(\mathit{fin}\ w) = \mathit{fin}\ w$
by (*metis DiamondDiamondEqvDiamond FinEqvTrueChopAndEmpty TrueChopEqvDiamond inteq-reflection*)

lemma *ChopFinExportA*:
$\vdash f;(g \wedge \mathit{fin}\ w) \longrightarrow \mathit{fin}\ w$
using *DiamondFin*
by (*metis ChopAndB ChopImpDiamond inteq-reflection lift-imp-trans*)

lemma *FinImpBox*:
$\vdash \mathit{fin}\ w \longrightarrow \Box(\mathit{fin}\ w)$
by (*metis BoxImpBoxBox fin-d-def*)

lemma *FinAndChopImport*:
$\vdash (\mathit{fin}\ w) \wedge (f;g) \longrightarrow f;((\mathit{fin}\ w) \wedge g)$
proof –
have *1*: $\vdash \mathit{fin}\ w \longrightarrow \Box(\mathit{fin}\ w)$ **by** (*rule FinImpBox*)
hence *2*: $\vdash \mathit{fin}\ w \wedge f;g \longrightarrow \Box(\mathit{fin}\ w) \wedge (f;g)$ **by** *auto*
have *3*: $\vdash \Box(\mathit{fin}\ w) \wedge (f;g) \longrightarrow f;((\mathit{fin}\ w) \wedge g)$ **using** *BoxAndChopImport* **by** *blast*
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *FinAndChop*:
$\vdash (f;(g \wedge \mathit{fin}\ w)) = (\mathit{fin}\ w \wedge f;g)$
using *FinAndChopImport ChopFinExportA ChopAndA ChopAndCommute* **by** *fastforce*

lemma *ChopAndEmptyEqvEmptyChopEmpty*:
$\vdash ((f;g) \wedge \mathit{empty}) = (f \wedge \mathit{empty});(g \wedge \mathit{empty})$
by (*auto simp*: *empty-defs chop-defs*)

lemma *FinAndEmpty*:
$\vdash ((\mathit{fin}\ w) \wedge \mathit{empty}) = (w \wedge \mathit{empty})$
proof –
have *1*: $\vdash ((\mathit{fin}\ w) \wedge \mathit{empty}) = (\#\mathit{True};(w \wedge \mathit{empty}) \wedge \mathit{empty})$
  using *FinEqvTrueChopAndEmpty* **by** *fastforce*
have *2*: $\vdash (\#\mathit{True};(w \wedge \mathit{empty}) \wedge \mathit{empty}) = ((\#\mathit{True} \wedge \mathit{empty});(w \wedge \mathit{empty}))$
  using *ChopAndEmptyEqvEmptyChopEmpty*
  by (*smt int-eq int-iffD2 lift-and-com Prop10 Prop12*)
have *3*: $\vdash (\#\mathit{True} \wedge \mathit{empty});(w \wedge \mathit{empty}) = (\mathit{empty};(w \wedge \mathit{empty}))$

```
    using LeftChopEqvChop by fastforce
 have 4: ⊢ (empty;(w ∧ empty)) = (w ∧ empty)
    using EmptyChop by blast
 from 1 2 3 4 show ?thesis by fastforce
qed

lemma AndFinEqvChopAndEmpty:
⊢  (f ∧  fin g) = f; (g ∧  empty )
proof −
 have 1: ⊢ (f ∧  fin g) = (f;empty ∧ fin g)
    using ChopEmpty by (metis int-eq)
 have 2: ⊢ (fin g ∧ f;empty) = (f;(empty ∧ fin g))
    using FinAndChop by fastforce
 have 3: ⊢ (empty ∧ fin g) =  (fin g ∧ empty)
    by auto
 have 4: ⊢ (fin g ∧ empty) = (g ∧ empty)
    using FinAndEmpty by metis
 have 5: ⊢ (empty ∧ fin g) = (g ∧ empty)
    using 3 4 by auto
 hence 6: ⊢ f;(empty ∧ fin g) = f;(g ∧ empty)
    using RightChopEqvChop by blast
 from 1 2 5 show ?thesis by (metis inteq-reflection lift-and-com)
qed

lemma AndFinEqvChopStateAndEmpty:
⊢  (f ∧  fin  (init w)) = f; ((init w) ∧  empty )
using AndFinEqvChopAndEmpty by blast

lemma FinStateEqvStateAndEmptyOrNextFinState:
⊢  fin  (init w) = (((init w) ∧  empty ) ∨  ○( fin  (init w)))
proof −
 have 1: ⊢ fin (init w) = □( empty ⟶ init w)
    by (simp add: fin-d-def)
 have 2 : ⊢ □(empty ⟶ init w) =
           ((empty ⟶ init w) ∧ wnext (□ (empty ⟶ init w)))
    by (rule BoxEqvAndWnextBox)
 have 3: ⊢ fin (init w) =  ((empty ⟶ init w) ∧ wnext (fin (init w)))
    using 1 2 by (simp add: fin-d-def)
 have 4: ⊢ wnext (fin (init w)) = (empty ∨ ○ (fin (init w)))
    by (rule WnextEqvEmptyOrNext)
 have 5: ⊢ fin (init w) =  ((empty ⟶ init w) ∧ (empty ∨ ○ (fin (init w))))
    using 3 4 by fastforce
 have 6: ⊢ ((empty ⟶ init w) ∧ (empty ∨ ○ (fin (init w)))) =
          (((empty ⟶ init w) ∧ empty) ∨ ((empty ⟶ init w) ∧ ○ (fin (init w))))
    by auto
 have 7: ⊢ ((empty ⟶ init w) ∧ empty) = ((init w) ∧ empty)
    by auto
 have 8: ⊢ ((empty ⟶ init w) ∧ ○ (fin (init w))) =  ○ (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*) $\wedge$ $\bigcirc$ (*fin* (*init w*)))) =
((*init w*) $\wedge$ *empty*) $\vee$ $\bigcirc$(*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*) $\vee$ $\bigcirc$((*fin* (*init w*)); *f*))
proof −
have *1*: $\vdash$ *fin* (*init w*) = (((*init w*) $\wedge$ *empty*) $\vee$ $\bigcirc$(*fin* (*init w*)))
by (*rule FinStateEqvStateAndEmptyOrNextFinState*)
hence *2*: $\vdash$ (*fin* (*init w*)); *f* = (((*init w*) $\wedge$ *empty*)$\vee$ $\bigcirc$(*fin* (*init w*))); *f*
by (*rule LeftChopEqvChop*)
have *3*: $\vdash$ (((*init w*) $\wedge$ *empty*)$\vee$ $\bigcirc$ (*fin* (*init w*))); *f*
= (((*init w*) $\wedge$ *empty*); *f* $\vee$ ($\bigcirc$ (*fin* (*init w*))); *f*)
by (*rule OrChopEqv*)
have *4*: $\vdash$ ((*init w*) $\wedge$ *empty*); *f* = ((*init w*) $\wedge$ *f*)
by (*rule StateAndEmptyChop*)
have *5*: $\vdash$ ($\bigcirc$ (*fin* (*init w*))); *f* = $\bigcirc$((*fin* (*init w*)); *f*)
by (*rule NextChop*)
from *2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *FinChopEqvDiamond*:
$\vdash$ (*fin* (*init w*)); *f* = $\Diamond$ ((*init w*) $\wedge$ *f*)
proof −
have *1*: $\vdash$ (*fin* (*init w*)) = (#*True*;((*init w*) $\wedge$ *empty*))
by (*rule FinEqvTrueChopAndEmpty*)
hence *2*: $\vdash$ (*fin* (*init w*));*f* = (#*True*;((*init w*) $\wedge$ *empty*));*f*
by (*rule LeftChopEqvChop*)
have *3*: $\vdash$ #*True*;(((*init w*) $\wedge$ *empty*);*f*) = (#*True*;((*init w*) $\wedge$ *empty*));*f*
by (*rule ChopAssoc*)
have *4*: $\vdash$ #*True*;(((*init w*) $\wedge$ *empty*);*f*)= $\Diamond$ (((*init w*) $\wedge$ *empty*);*f*)
by (*simp add*: *sometimes-d-def*)
have *5*: $\vdash$ ((*init w*) $\wedge$ *empty*);*f* = ((*init w*) $\wedge$ *f*)
using *StateAndEmptyChop* **by** *blast*
hence *6*: $\vdash$ $\Diamond$ (((*init w*) $\wedge$ *empty*);*f*) = $\Diamond$ ((*init w*) $\wedge$ *f*)
by (*rule DiamondEqvDiamond*)
from *2 3 4 6* **show** *?thesis* **by** *fastforce*
qed

lemma *NotDiamondAndNot*:
$\vdash$ ¬($\Diamond$ (*f* $\wedge$ ¬ *f*))
by (*metis BoxGen DiamondEmpty always-d-def int-eq-true intensional-simps*(*2*)
intensional-simps(*21*) *inteq-reflection*)

lemma *FinYields*:
$\vdash$ (*fin* (*init w*)) *yields* (*init w*)
proof −

have *1*: $\vdash$ (*fin* (*init w*)); $\neg$(*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*)); $\neg$ (*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*)) $\vee$ *fin* $\neg$ (*init w*)
by (*simp add*: *fin-defs Valid-def*)

lemma *AndFinChopEqvStateAndChop*:
$\vdash$ (*f* $\wedge$ *fin* (*init w*)); *g* = *f*; ((*init w*) $\wedge$ *g*)
proof −
have *1*: $\vdash$ (*fin* (*init w*)) *yields* (*init w*)
by (*rule FinYields*)
have *2*: $\vdash$ *f* $\wedge$ *fin* (*init w*) $\longrightarrow$ *fin* (*init w*)
by *auto*
hence *3*: $\vdash$ (*fin* (*init w*)) *yields* (*init w*) $\longrightarrow$ (*f* $\wedge$ *fin* (*init w*)) *yields* (*init w*)
by (*rule LeftYieldsImpYields*)
have *4*: $\vdash$ (*f* $\wedge$ *fin* (*init w*)) *yields* (*init w*)
using *1 3 MP* **by** *fastforce*
have *5*: $\vdash$ (*f* $\wedge$ *fin* (*init w*)); *g* $\wedge$ (*f* $\wedge$ *fin* (*init w*)) *yields* (*init w*)
$\longrightarrow$ (*f* $\wedge$ *fin* (*init w*)); (*g* $\wedge$ (*init w*))
by (*rule ChopAndYieldsImp*)
have *6*: $\vdash$ (*f* $\wedge$ *fin* (*init w*)); *g* $\longrightarrow$ (*f* $\wedge$ *fin* (*init w*)); (*g* $\wedge$ (*init w*))
using *4 5* **by** *fastforce*
have *7*: $\vdash$ (*f* $\wedge$ *fin* (*init w*)); (*g* $\wedge$ (*init w*)) $\longrightarrow$ *f*; (*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*; (*g* $\wedge$ (*init w*)) $\longrightarrow$ *f*; ((*init w*) $\wedge$ *g*)
by (*rule RightChopImpChop*)
have *10*: $\vdash$ (*f* $\wedge$ *fin* (*init w*)); *g* $\longrightarrow$ *f*; ((*init w*) $\wedge$ *g*)
using *6 7 9* **by** *fastforce*
have *11*: $\vdash$ *f* $\longrightarrow$ (*f* $\wedge$ *fin* (*init w*)) $\vee$ *fin* $\neg$ (*init w*)
by (*rule ImpAndFinStateOrFinNotState*)
hence *12*: $\vdash$ *f*; ((*init w*) $\wedge$ *g*) $\longrightarrow$
((*f* $\wedge$ *fin* (*init w*))$\vee$ *fin* $\neg$ (*init w*)); ((*init w*) $\wedge$ *g*)
by (*rule LeftChopImpChop*)
have *13*: $\vdash$ ((*f* $\wedge$ *fin* (*init w*))$\vee$ *fin* $\neg$ (*init w*)); ((*init w*) $\wedge$ *g*)
=
((*f* $\wedge$ *fin* (*init w*)); ((*init w*) $\wedge$ *g*) $\vee$ (*fin* $\neg$ (*init w*)); ((*init w*) $\wedge$ *g*))
by (*rule OrChopEqv*)
have *14*: $\vdash$ (*fin* (*init* ($\neg$ *w*))); ((*init w*) $\wedge$ *g*) $\longrightarrow$ $\Diamond$((*init* ($\neg$ *w*)) $\wedge$ ((*init w*) $\wedge$ *g*))
using *FinChopEqvDiamond* **by** *fastforce*
have *141*: $\vdash$ $\neg$($\Diamond$((*init* ($\neg$ *w*)) $\wedge$ ((*init w*) $\wedge$ *g*))) $\longrightarrow$
$\neg$ ((*fin* (*init* ($\neg$ *w*))); ((*init w*) $\wedge$ *g*))
using *14* **by** *fastforce*
have *15*: $\vdash$ $\neg$($\Diamond$((*init* ($\neg$ *w*)) $\wedge$ ((*init w*) $\wedge$ *g*)))
using *NotDiamondAndNot Initprop*(*2*) **by** (*auto simp*: *sometimes-defs init-defs*)

have *151*: $\vdash \neg\ (\ (\ fin\ \ (init\ (\neg\ w)));\ ((init\ w) \wedge g))$
    using *15 141* **by** *fastforce*
have *1511*: $\vdash (\ fin\ \neg\ \ (init\ w));\ ((init\ w) \wedge g) \longrightarrow \#False$
    using *151* **by** (*metis Initprop*(*2*) *int-eq intensional-simps*(*14*))
have *152*: $\vdash (f \wedge\ \ fin\ \ (init\ w));\ ((init\ w) \wedge g) \vee\ \ (\ fin\ \neg\ \ (init\ w));\ ((init\ w) \wedge g) \longrightarrow$
        $(f \wedge\ \ fin\ \ (init\ w));\ ((init\ w) \wedge g)$
    using *1511* **by** *fastforce*
have *16*: $\vdash f;\ ((init\ w) \wedge g) \longrightarrow (f \wedge\ \ fin\ \ (init\ w));\ ((init\ w) \wedge g)$
    using *12 13 152* **by** *fastforce*
have *17*: $\vdash (f \wedge\ \ fin\ \ (init\ w));\ ((init\ w) \wedge g) \longrightarrow (f\wedge\ \ fin\ \ (init\ w));\ g$
    by (*rule ChopAndB*)
have *18*: $\vdash f;\ ((init\ w) \wedge g) \longrightarrow (f \wedge\ \ fin\ \ (init\ w));\ g$
    using *16 17* **by** *fastforce*
from *10 18* **show** *?thesis* **by** *fastforce*
qed

lemma *DiAndFinEqvChopState*:
$\vdash\ \ di\ (f \wedge\ \ fin\ \ (init\ w)) = f;\ (init\ w)$
proof $-$
have *1*: $\vdash (f \wedge fin(init\ w));\ \#True = f;((init\ w) \wedge \#True)$ **by** (*rule AndFinChopEqvStateAndChop*)
have *2*: $\vdash ((init\ w) \wedge \#True) = (init\ w)$ **by** *auto*
hence *3*: $\vdash (f;\ ((init\ w) \wedge \#True)) = (f;\ (init\ w))$ **by** (*rule RightChopEqvChop*)
have *4*: $\vdash (f \wedge\ \ fin\ \ (init\ w));\ \#True = f;\ (init\ w)$ **using** *1 3* **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *FinNotStateEqvNotFinState*:
$\vdash fin\ \ \ (init\ \neg\ w) = \neg(\ fin\ \ \ (init\ w))$
by (*metis FinEqvTrueChopAndEmpty Finprop*(*4*) *Initprop*(*2*) *inteq-reflection*)

lemma *BiImpFinEqvYieldsState*:
$\vdash\ \ bi\ (f \longrightarrow\ \ fin\ \ (init\ w)) = f\ yields\ \ (init\ w)$
proof $-$
have *1*: $\vdash\ \ di\ (f \wedge fin\ (init\ \neg\ w)) = f;\ \ \ (init\ \neg\ w)$
    by (*rule DiAndFinEqvChopState*)
have *2*: $\vdash (f \wedge fin(init\ \neg\ w))\ = (f \wedge \neg(fin(init\ w)))$
    using *FinNotStateEqvNotFinState* **by** *fastforce*
have *3*: $\vdash (f \wedge \neg\ (fin(init\ w))) = \neg\ (f \longrightarrow\ \ fin\ \ (init\ w))$
    by *auto*
have *4*: $\vdash (f \wedge fin(init\ \ \neg\ w)) = \neg\ (f \longrightarrow\ \ fin(init\ w))$
    using *2 3* **by** *fastforce*
hence *5*: $\vdash\ \ di\ (f \wedge\ \ fin\ \ \ (init\ \neg\ w)) =\ \ di\ \neg\ (f \longrightarrow fin(init\ w))$
    by (*rule DiEqvDi*)
have *6*: $\vdash\ \ di\ \neg\ (f \longrightarrow\ \ fin\ \ (init\ w)) = \neg(\ bi\ (f \longrightarrow fin(init\ w)))$
    by (*rule DiNotEqvNotBi*)
have *7*: $\vdash \neg\ (bi\ (f \longrightarrow\ \ fin\ \ (init\ w))) = f;(init\ \neg\ w)$
    using *1 5 6 Initprop* **by** *fastforce*
hence *8*: $\vdash bi\ (f \longrightarrow\ \ fin\ \ (init\ w)) = \neg\ (f;\ \neg\ \ (init\ w))$
    by (*metis 6 Initprop*(*2*) *bi-d-def int-eq int-simps*(*15*))
from *8* **show** *?thesis* **by** (*simp add*: *yields-d-def*)

qed

lemma *StateImpYields*:
assumes $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *fin* (*init w1*)
shows $\vdash$ (*init w*) $\longrightarrow$ (*f yields* (*init w1*))
proof −
have *1*: $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *fin* (*init w1*) **using** *assms* **by** *auto*
hence *2*: $\vdash$ (*init w*) $\longrightarrow$ (*f* $\longrightarrow$ *fin* (*init w1*)) **by** *auto*
hence *3*: $\vdash$ (*init w*) $\longrightarrow$ *bi* (*f* $\longrightarrow$ *fin* (*init w1*)) **by** (*rule StateImpBiGen*)
have *4*: $\vdash$ *bi* (*f* $\longrightarrow$ *fin* (*init w1*)) = *f yields* (*init w1*) **by** (*rule BiImpFinEqvYieldsState*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *StateAndYieldsImpYields*:
assumes $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *f1*
shows $\vdash$ (*init w*) $\wedge$ (*f1 yields g*) $\longrightarrow$ (*f yields g*)
proof −
have *1*: $\vdash$ (*init w*) $\wedge$ *f* $\longrightarrow$ *f1* **using** *assms* **by** *auto*
hence *2*: $\vdash$ (*init w*) $\wedge$ (*f*; $\neg$ *g*) $\longrightarrow$ *f1*; $\neg$ *g* **by** (*rule StateAndChopImpChopRule*)
hence *3*: $\vdash$ (*init w*) $\wedge$ $\neg$ (*f1*; $\neg$ *g*) $\longrightarrow$ $\neg$ (*f*; $\neg$ *g*) **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *AndYieldsA*:
$\vdash$ *f yields g* $\longrightarrow$ (*f* $\wedge$ *f1*) *yields g*
proof −
have *1*: $\vdash$ *f* $\wedge$ *f1* $\longrightarrow$ *f* **by** *auto*
from *1* **show** *?thesis* **by** (*rule LeftYieldsImpYields*)
qed

lemma *AndYieldsB*:
$\vdash$ *f1 yields g* $\longrightarrow$ (*f* $\wedge$ *f1*) *yields g*
proof −
have *1*: $\vdash$ *f* $\wedge$ *f1* $\longrightarrow$ *f1* **by** *auto*
from *1* **show** *?thesis* **by** (*rule LeftYieldsImpYields*)
qed

lemma *RightYieldsImpYields*:
assumes $\vdash$ *g* $\longrightarrow$ *g1*
shows $\vdash$ (*f yields g*) $\longrightarrow$ (*f 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 $\vdash (f\ yields\ \ g) = (f\ yields\ \ g1)$
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*:
$\vdash\ \ \Box\, g \longrightarrow f\ yields\ \ g$
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*:
$\vdash\ \ \Box\, f = \#True\ yields\ \ f$
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 $\vdash f\ yields\ \ g$
proof −
have *1*: $\vdash g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \Box\, g$ **by** (*rule BoxGen*)
have *3*: $\vdash \Box\, g \longrightarrow f\ yields\ \ g$ **by** (*rule BoxImpYields*)
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *YieldsAndYieldsEqvYieldsAnd*:
$\vdash\ \ ((f\ yields\ \ g) \wedge (f\ \ yields\ \ g1)) = f\ yields\ (g \wedge g1)$
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*:

$\vdash \ (f\ \mathit{yields}\ \ g) \wedge (f1\ \mathit{yields}\ \ g1) \longrightarrow (f \wedge f1)\ \mathit{yields}\ (g \wedge g1)$
proof −
have *1*: $\vdash f\ \mathit{yields}\ \ g \longrightarrow (f \wedge f1)\ \mathit{yields}\ \ g$
by (*rule AndYieldsA*)
have *2*: $\vdash f1\ \mathit{yields}\ \ g1 \longrightarrow (f \wedge f1)\ \mathit{yields}\ \ g1$
by (*rule AndYieldsB*)
have *3*: $\vdash ((f \wedge f1)\ \mathit{yields}\ \ g \wedge (f \wedge f1)\ \mathit{yields}\ \ g1) = (f \wedge f1)\ \mathit{yields}\ \ (g \wedge g1)$
by (*rule YieldsAndYieldsEqvYieldsAnd*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *YieldsYieldsEqvChopYields*:
$\vdash \ f\ \mathit{yields}\ (g\ \mathit{yields}\ \ h) = (f;\ g)\ \mathit{yields}\ h$
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*: $\vdash f;\ \neg\ (g\ \mathit{yields}\ \ h) = (f;\ g);\ \neg\ \ h$ **by** (*simp add*: *yields-d-def*)
hence *7*: $\vdash \neg\ (f;\ \neg\ (g\ \mathit{yields}\ \ h)) = \neg\ ((f;\ g);\ \neg\ \ h)$ **by** *auto*
from *7* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *EmptyYields*:
$\vdash \ \ \mathit{empty}\ \ \ \mathit{yields}\ \ f = f$
proof −
have *1*: $\vdash\ \ \mathit{empty}\ ;\ \neg\ \ f = \neg\ \ f$ **by** (*rule EmptyChop*)
hence *2*: $\vdash \neg\ (\ \mathit{empty}\ ;\ \neg\ \ f) = f$ **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *NextYields*:
$\vdash \ (\bigcirc\ f)\ \mathit{yields}\ \ g = \mathit{wnext}\ (f\ \mathit{yields}\ \ g)$
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*: $\vdash (\bigcirc\ f)\ \mathit{yields}\ \ g = \neg\ (\bigcirc(f;\ \neg\ \ g))$ **by** (*simp add*: *yields-d-def*)
have *4*: $\vdash \neg(\ \bigcirc(f;\ \neg\ \ g)) = \mathit{wnext}\ \neg\ (f;\ \neg\ \ g)$ **by** (*auto simp*: *wnext-d-def*)
have *5*: $\vdash (\bigcirc\ f)\ \mathit{yields}\ \ g = \mathit{wnext}\ \neg\ (f;\ \neg\ \ g)$ **using** *3 4* **by** *fastforce*
from *5* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *SkipChopEqvNext*:
$\vdash \ \ \mathit{skip}\ ;\ f = \bigcirc\ f$
by (*simp add*: *next-d-def*)

lemma *SkipYieldsEqvWeakNext*:
$\vdash \ \ \mathit{skip}\ \ \mathit{yields}\ \ f = \mathit{wnext}\ \ f$
proof −

have *1*: $\vdash$ *skip* ; $\neg$ *f* = $\bigcirc\neg$ *f* **by** (*rule SkipChopEqvNext*)
hence *2*: $\vdash$ $\neg$ (*skip* ; $\neg$ *f*) = $\neg$($\bigcirc\neg$ *f*) **by** *auto*
have *3*: $\vdash$ $\neg$ ($\bigcirc\neg$ *f*) = *wnext* *f* **by** (*auto simp*: *wnext-d-def*)
have *4*: $\vdash$ $\neg$ (*skip* ; $\neg$ *f*) = *wnext* *f* **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *NextImpSkipYields*:
$\vdash$ $\bigcirc$ *f* $\longrightarrow$ *skip* *yields* *f*
proof –
have *1*: $\vdash$ $\bigcirc$ *f* $\longrightarrow$ *wnext* *f* **using** *WnextEqvEmptyOrNext* **by** *fastforce*
have *2*: $\vdash$ *skip* *yields* *f* = *wnext* *f* **by** (*rule SkipYieldsEqvWeakNext*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *MoreEqvSkipChopTrue*:
$\vdash$ *more* = *skip* ; #*True*
proof –
have *1*: $\vdash$ *skip* ; #*True* = $\bigcirc$#*True* **by** (*rule SkipChopEqvNext*)
hence *2*: $\vdash$ $\bigcirc$#*True* = *skip* ; #*True* **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *more-d-def*)
qed

lemma *MoreChopImpMore*:
$\vdash$ *more* ; *f* $\longrightarrow$ *more*
proof –
have *1*: $\vdash$ ($\bigcirc$#*True*); *f* = $\bigcirc$(#*True*; *f*) **by** (*rule NextChop*)
have *2*: $\vdash$ $\bigcirc$(#*True*; *f*) $\longrightarrow$ *more* **by** (*auto simp*: *more-defs next-defs*)
have *3*: $\vdash$ ($\bigcirc$#*True*; *f*) $\longrightarrow$ *more* **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*metis more-d-def*)
qed

lemma *ChopMoreImpMore*:
$\vdash$ *f*; *more* $\longrightarrow$ *more*
proof –
have *1*: $\vdash$ *f*; *more* $\longrightarrow$ $\Diamond$ *more* **by** (*rule ChopImpDiamond*)
have *2*: $\vdash$ $\Diamond$ *more* $\longrightarrow$ *more* **by** (*auto simp*: *more-defs sometimes-defs*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *MoreChopEqvNextDiamond*:
$\vdash$ *more* ; *f* = $\bigcirc$($\Diamond$ *f*)
proof –
have *1*: $\vdash$ *more* ; *f* = ($\bigcirc$ #*True*); *f* **by** (*simp add*: *more-d-def*)
have *2*: $\vdash$ ($\bigcirc$#*True*); *f* = $\bigcirc$(#*True*; *f*) **by** (*rule NextChop*)
have *3*: $\vdash$ *more* ; *f* = $\bigcirc$(#*True*; *f*) **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *sometimes-d-def*)
qed

lemma *WeakNextBoxImpMoreYields*:

$\vdash$ *more yields f* $=$ *wnext*($\square$ *f*)
proof $-$
have *1*: $\vdash$ *more* ; $\neg$ *f* $= \bigcirc(\Diamond \neg f)$ **by** (*rule MoreChopEqvNextDiamond*)
have *2*: $\vdash \bigcirc(\Diamond \neg f) = \bigcirc(\neg(\square f))$ **by** (*auto simp*: *always-d-def*)
have *3*: $\vdash \bigcirc(\neg(\square f)) = \neg$ (*wnext*($\square$ *f*)) **by** (*auto simp*: *wnext-d-def*)
have *4*: $\vdash$ *more* ; $\neg$ *f* $= \neg$(*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 yields more*
proof $-$
have *1*: $\vdash$ *f*; *empty* $=$ *f* **by** (*rule ChopEmpty*)
hence *2*: $\vdash \neg$ (*f*; *empty*) $= \neg$ *f* **by** *auto*
have *3*: $\vdash$ *empty* $= \neg$ *more* **by** (*auto simp*: *empty-d-def*)
hence *4*: $\vdash$ *f*; *empty* $=$ *f*; $\neg$ *more* **by** (*rule RightChopEqvChop*)
hence *5*: $\vdash \neg$ (*f*; *empty*) $= \neg$ (*f*; $\neg$ *more*) **by** *auto*
have *6*: $\vdash \neg$ *f* $= \neg$ (*f*; $\neg$ *more*) **using** *2 5* **by** *fastforce*
from *6* **show** *?thesis* **by** (*metis yields-d-def*)
qed

lemma *LeftChopImpMoreRule*:
assumes $\vdash$ *f* $\longrightarrow$ *more*
shows $\vdash$ *f*; *g* $\longrightarrow$ *more*
proof $-$
have *1*: $\vdash$ *f* $\longrightarrow$ *more* **using** *assms* **by** *auto*
hence *2*: $\vdash$ *f*; *g* $\longrightarrow$ *more* ; *g* **by** (*rule LeftChopImpChop*)
have *3*: $\vdash$ *more* ; *g* $\longrightarrow$ *more* **by** (*rule MoreChopImpMore*)
from *2 3* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *RightChopImpMoreRule*:
assumes $\vdash$ *g* $\longrightarrow$ *more*
shows $\vdash$ *f*; *g* $\longrightarrow$ *more*
proof $-$
have *1*: $\vdash$ *g* $\longrightarrow$ *more* **using** *assms* **by** *auto*
hence *2*: $\vdash$ *f*; *g* $\longrightarrow$ *f*; *more* **by** (*rule RightChopImpChop*)
have *3*: $\vdash$ *f*; *more* $\longrightarrow$ *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*:
$\vdash \ di \ (f \lor \ g) = \ (di \ f \lor \ di \ g)$
proof –
 have *1*: $\vdash (f \lor \ g); \#True = (f; \#True \lor \ g; \#True)$ **by** (*rule OrChopEqv*)
 from *1* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiAndA*:
$\vdash \ di \ (f \land g) \longrightarrow \ di \ f$
proof –
 have *1*: $\vdash (f \land g); \#True \longrightarrow f; \#True$ **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* ; $\#True = \bigcirc\#True$ **by** (*rule SkipChopEqvNext*)
have *2*: $\vdash \bigcirc\#True =$ *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* $(\bigcirc \#True) = \bigcirc($ *di* $\#True)$ **by** (*rule DiNext*)
have *2*: $\vdash \bigcirc($ *di* $\#True) \longrightarrow$ *more* **by** (*auto simp*: *next-defs di-defs more-defs*)
have *3*: $\vdash$ *di*$(\bigcirc \#True) \longrightarrow$ *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: ⊢  empty ; #True = #True  by (rule EmptyChop)
 have 3: ⊢  empty ; #True  using 1 2 by auto
 from 3 show ?thesis by (simp add: di-d-def)
qed

lemma DaNotEqvNotBa:
⊢   da ¬  f = ¬ ( ba  f)
proof −
 have 1: ⊢  ba  f = ¬ ( da ¬  f)  by (simp add: ba-d-def)
 from 1 show ?thesis by fastforce
qed


lemma DaEqvDa:
 assumes ⊢   f = g
 shows ⊢ da  f =  da  g
using assms using int-eq by force

lemma DaEqvNotBaNot:
⊢   da  f = ¬ ( ba ¬  f)
proof −
 have  1: ⊢  ba ¬  f = ¬ ( da ¬ ¬  f) by (simp add: ba-d-def)
 hence 2: ⊢  da ¬ ¬  f = ¬(  ba ¬  f)  by fastforce
 have  3: ⊢ f = ¬ ¬  f  by simp
 hence 4: ⊢  da  f =  da  ¬ ¬  f  by (rule DaEqvDa)
 from 2 4 show ?thesis by simp
qed

lemma BaElim:
⊢   ba  f ⟶ f
proof −
 have  1: ⊢  ba  f = □(bi  f)  by (rule BaEqvBtBi)
 have  2: ⊢ bi  f ⟶ f  by (rule BiElim)
 hence 3: ⊢ □(bi  f ⟶ f)  by (rule BoxGen)
 have  4: ⊢ □(bi  f ⟶ f) ⟶ □(bi  f) ⟶ □ f  by (rule BoxImpDist)
 have  5: ⊢ □(bi  f) ⟶ □ f  using 3 4 MP by fastforce
 have  6: ⊢ □ f ⟶ f  by (rule BoxElim)
 from 1 5 6 show ?thesis  using BaImpBt lift-imp-trans by metis
qed

lemma DaIntro:
⊢  f ⟶  da  f
proof −
 have  1: ⊢  ba  ¬  f ⟶ ¬  f   by (rule BaElim)
 hence 2: ⊢ ¬ ¬  f ⟶ ¬ (  ba  ¬  f)  by fastforce
 have  3: ⊢ f = ¬ ¬  f  by simp
 have  4: ⊢  da  f = ¬  ( ba  ¬  f)  by (rule DaEqvNotBaNot)
 from 2 3 4 show ?thesis by fastforce
qed
```

lemma *BaGen*:
 assumes $\vdash$ *f*
 shows $\vdash$ *ba f*
proof –
 have *1*: $\vdash$ *f* **using** *assms* **by** *auto*
 hence *2*: $\vdash \Box$ *f* **by** (*rule BoxGen*)
 hence *3*: $\vdash$ *bi*($\Box$ *f*) **by** (*rule BiGen*)
 have *4*: $\vdash$ *ba f* = *bi* ($\Box$ *f*) **by** (*rule BaEqvBiBt*)
 from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BaImpDist*:
$\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\longrightarrow$ *ba f* $\longrightarrow$ *ba g*
proof –
 have *1*: $\vdash$ *bi* (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*bi f* $\longrightarrow$ *bi g*) **by** (*rule BiImpDist*)
 hence *2*: $\vdash \Box$(*bi* (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*bi f* $\longrightarrow$ *bi g*)) **by** (*rule BoxGen*)
 have *3*: $\vdash \Box$(*bi* (*f* $\longrightarrow$ *g*) $\longrightarrow$ (*bi f* $\longrightarrow$ *bi g*))
 $\longrightarrow$
 ($\Box$ (*bi* (*f* $\longrightarrow$ *g*)) $\longrightarrow$ ($\Box$(*bi f*) $\longrightarrow$ $\Box$(*bi g*)))
 by (*meson 2 BoxImpDist MP lift-imp-trans Prop01 Prop05 Prop09*)
 have *4*: $\vdash \Box$(*bi* (*f* $\longrightarrow$ *g*)) $\longrightarrow$ ($\Box$(*bi f*) $\longrightarrow$ $\Box$(*bi g*)) **using** *2 3 MP* **by** *fastforce*
 have *5*: $\vdash$ *ba* (*f* $\longrightarrow$ *g*) = $\Box$(*bi* (*f* $\longrightarrow$ *g*)) **by** (*rule BaEqvBtBi*)
 have *6*: $\vdash$ *ba f* = $\Box$(*bi f*) **by** (*rule BaEqvBtBi*)
 have *7*: $\vdash$ *ba g* = $\Box$(*bi g*) **by** (*rule BaEqvBtBi*)
 from *4 5 6 7* **show** *?thesis* **by** *fastforce*
qed

lemma *BaAndEqv*:
$\vdash$ *ba* (*f* $\wedge$ *g*) = (*ba f* $\wedge$ *ba g*)
proof –
 have *1*: $\vdash$ *ba* (*f* $\wedge$ *g*) = $\Box$(*bi* (*f* $\wedge$ *g*))
 by (*rule BaEqvBtBi*)
 have *2*: $\vdash$ *bi* (*f* $\wedge$ *g*) = (*bi f* $\wedge$ *bi g*)
 by (*auto simp*: *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 *BaImpBaEqvBa*:
$\vdash$ *ba* (*f* = *g*) $\longrightarrow$ (*ba f* = *ba g*)
proof –
 have *1*: $\vdash$ *ba* (*f* $\longrightarrow$ *g*) $\longrightarrow$ *ba f* $\longrightarrow$ *ba g* **by** (*rule BaImpDist*)
 have *2*: $\vdash$ *ba* (*g* $\longrightarrow$ *f*) $\longrightarrow$ *ba g* $\longrightarrow$ *ba f* **by** (*rule BaImpDist*)

have *3*: $\vdash$ *ba* $(f = g) =$ *ba* $((f \longrightarrow g) \wedge (g \longrightarrow f))$ **by** (*auto simp*: *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 *BaImpBa*:
assumes $\vdash\ \ f \longrightarrow g$
shows $\vdash ba\ \ f \longrightarrow\ \ ba\ \ g$
using *BaGen BaImpDist MP assms* **by** *metis*

lemma *BaEqvBa*:
assumes $\vdash\ \ f = g$
shows $\vdash ba\ \ f =\ \ ba\ \ g$
using *BaGen BaImpBaEqvBa MP assms* **by** *metis*

lemma *DaImpDa*:
assumes $\vdash\ \ f \longrightarrow g$
shows $\vdash da\ \ f \longrightarrow\ \ da\ \ g$
using *assms* **by** (*metis DaEqvDtDi DiAndB DiamondImpDiamond inteq-reflection Prop10*)

lemma *DiamondEqvDiamondDiamond*:
$\vdash \Diamond f = \Diamond (\Diamond f)$
proof $-$
have *1*: $\vdash \Diamond (\Diamond f) = \#True;(\#True;f)$
by (*simp add*: *sometimes-d-def*)
have *2*: $\vdash \#True;(\#True;f)\ \ = (\#True;\#True);f$
by (*rule ChopAssoc*)
have *3*: $\vdash (\#True;\#True);f\ = \#True;f$
using *LeftChopEqvChop TrueEqvTrueChopTrue* **by** (*metis int-eq*)
have *4*: $\vdash \#True;f\ = \Diamond f$
by (*simp add*: *sometimes-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *DaEqvDaDa*:
$\vdash\ \ da\ \ f =\ \ da\ (\ \ da\ \ f)$
proof $-$
have *1*: $\vdash\ \ da\ \ f = \Diamond(\ di\ \ f)$
by (*rule DaEqvDtDi*)
have *2*: $\vdash\ \ di\ \ f =\ \ (di\ (\ di\ \ f))$
by (*rule DiEqvDiDi*)
hence *3*: $\vdash \Diamond\ (\ di\ \ f) = \Diamond\ (di\ \ (di\ \ f))$
by (*rule DiamondEqvDiamond*)
have *4*: $\vdash \Diamond\ \ (di\ \ f) = \Diamond(\Diamond\ (di\ \ (di\ \ f)))$

using *DiamondEqvDiamondDiamond DiEqvDiDi* **using** *3* **by** *fastforce*
have *5*: $\vdash \diamond\ (di\ (di\ f)) = di\ (\diamond\ (di\ f))$
by (*rule DtDiEqvDiDt*)
hence *6*: $\vdash \diamond(\diamond\ (di\ (di\ f))) = \diamond\ (di\ (\diamond\ (di\ f)))$
by (*rule DiamondEqvDiamond*)
have *7*: $\vdash\ da\ f = \diamond\ (di(\ \diamond\ (\ di\ f)))$
using *1 3 4 6* **by** *fastforce*
have *8*: $\vdash\ da\ (\diamond\ (\ di\ f)) = \diamond(\ di\ (\diamond\ (di\ f)))$
by (*rule DaEqvDtDi*)
have *9*: $\vdash\ da\ (\ da\ f) = da\ (\diamond\ (di\ f))$
using *1* **by** (*rule DaEqvDa*)
from *7 8 9* **show** *?thesis* **by** *fastforce*
qed

lemma *BaEqvBaBa*:
$\vdash\ ba\ f = ba\ (ba\ f)$
proof −
have *1*: $\vdash\ da\ (\neg\ f) = da\ (da\ (\neg\ f))$ **by** (*rule DaEqvDaDa*)
have *2*: $\vdash\ da\ (da\ (\neg\ f)) = \neg\ (ba\ (\neg\ (da\ (\neg\ f))))$ **by** (*rule DaEqvNotBaNot*)
have *3*: $\vdash \neg\ (da\ (da\ (\neg\ f))) = ba\ (\neg\ (da\ (\neg\ f)))$ **by** (*auto simp*: *ba-d-def*)
have *4*: $\vdash \neg\ (da\ (\neg\ f)) = ba\ (\neg\ (da\ (\neg\ f)))$ **using** *1 2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*metis ba-d-def*)
qed

lemma *BaLeftChopImpChop*:
$\vdash\ ba\ (f \longrightarrow f1) \longrightarrow f;\ g \longrightarrow f1;\ g$
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*: $\vdash\ ba\ (g \longrightarrow g1) \longrightarrow \square(g \longrightarrow g1)$ **by** (*rule BaImpBt*)
have *2*: $\vdash \square(g \longrightarrow g1) \longrightarrow f;\ g \longrightarrow f;\ g1$ **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*; *fin* (*init* $\neg$ *w*) $\longrightarrow$ *fin* (*init* $\neg$ *w*)
by (*rule ChopFinImpFin*)
have *2*: $\vdash$ *fin* (*init* $\neg$ *w*) = $\neg$ (*fin* (*init w*))
using *FinNotStateEqvNotFinState* **by** *blast*
hence *3*: $\vdash$ *f*; *fin* (*init* $\neg$ *w*) = *f*; $\neg$ (*fin* (*init w*))
by (*rule RightChopEqvChop*)
have *4*: $\vdash$ *f*; $\neg$ (*fin* (*init w*)) $\longrightarrow$ $\neg$ (*fin* (*init w*))
using *1 2 3* **by** *fastforce*
hence *5*: $\vdash$ *fin* (*init w*) $\longrightarrow$ $\neg$ (*f*; $\neg$ (*fin* (*init w*)))
by *fastforce*
from *5* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *ChopAndFin*:
$\vdash$ ((*f*; *g*) $\wedge$ *fin* (*init w*)) = *f*; (*g* $\wedge$ *fin* (*init w*))
proof −
have *1*: $\vdash$ *fin* (*init w*) $\longrightarrow$ *f yields* (*fin* (*init w*))
by (*rule FinImpYieldsFin*)
hence *2*: $\vdash$ (*f*; *g*) $\wedge$ *fin* (*init w*) $\longrightarrow$ (*f*; *g*) $\wedge$ *f yields* (*fin* (*init w*))
by *auto*
have *3*: $\vdash$ (*f*; *g*) $\wedge$ *f yields* (*fin* (*init w*)) $\longrightarrow$ *f*; (*g* $\wedge$ *fin* (*init w*))
by (*rule ChopAndYieldsImp*)
have *4*: $\vdash$ (*f*; *g*) $\wedge$ *fin* (*init w*) $\longrightarrow$ *f*; (*g* $\wedge$ *fin* (*init w*))
using *2 3* **by** *fastforce*
have *11*: $\vdash$ *f*; (*g* $\wedge$ *fin* (*init w*)) $\longrightarrow$ *f*; *g*
by (*rule ChopAndA*)
have *12*: $\vdash$ *f*; (*g* $\wedge$ *fin* (*init w*)) $\longrightarrow$ *f*; *fin* (*init w*)
by (*rule ChopAndB*)
have *13*: $\vdash$ *f*; *fin* (*init w*) $\longrightarrow$ $\diamond$ (*fin* (*init w*))
by (*rule ChopImpDiamond*)
have *14*: $\vdash$ $\diamond$(*fin* (*init w*)) $\longrightarrow$ *fin* (*init w*)
by (*rule DiamondFinImpFin*)
have *15*: $\vdash$ *f*; (*g* $\wedge$ *fin* (*init w*)) $\longrightarrow$ (*f*; *g*) $\wedge$ *fin* (*init w*)
using *11 12 13 14* **by** *fastforce*
from *4 15* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopAndNotFin*:
$\vdash$ $(f; g \wedge \neg (\; fin \; (init \; w))) = f; (g \wedge \neg (\; fin \; (init \; w)))$
proof –
have *1*: $\vdash (f; g \wedge \; fin \; (init \neg w)) = f; (g \wedge \; fin \; (init \neg w))$
by (*rule ChopAndFin*)
have *2*: $\vdash \; fin \; (init \neg w) = \neg \; (\; fin \; (init \; w) \;)$
using *FinNotStateEqvNotFinState* **by** *blast*
hence *3*: $\vdash (g \wedge \; fin \; (init \neg w)) = (g \wedge \neg(\; fin \; (init \; w)))$
by *auto*
hence *4*: $\vdash f; (g \wedge \; fin \; (init \neg w)) = f; (g \wedge \neg (\; fin \; (init \; w)))$
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 StateAndChopImport*)
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 StateAndChopImport*)

have *2*: ⊢ (*init w*) ∧ *f* ⟶ *fin* (*init w1*) **using** *assms* **by** *auto*
hence *3*: ⊢ ((*init w*) ∧ *f*); *f1* ⟶ (*fin* (*init w1*)); *f1* **by** (*rule LeftChopImpChop*)
have *4*: ⊢ (*fin* (*init w1*)); *f1* = ◇((*init w1*) ∧ *f1*) **by** (*rule FinChopEqvDiamond*)
have *5*: ⊢ (*init w1*) ∧ *f1* ⟶ *fin* (*init w2*) **using** *assms* **by** *auto*
hence *6*: ⊢ ◇((*init w1*) ∧ *f1*) ⟶ ◇ (*fin* (*init w2*)) **by** (*rule DiamondImpDiamond*)
have *7*: ⊢ ◇(*fin* (*init w2*)) ⟶ *fin* (*init w2*) **using** *DiamondFinImpFin* **by** *blast*
from *1 3 4 6 7* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopRep*:
assumes ⊢ (*init w*) ∧ *f* ⟶ *f1* ∧ *fin* (*init w1*)
⊢ (*init w1*) ∧ *g* ⟶ *g1*
shows ⊢ (*init w*) ∧ (*f*; *g*) ⟶ (*f1*; *g1*)
proof −
have *1*: ⊢ (*init w*) ∧ *f* ⟶ *f1* ∧ *fin* (*init w1*) **using** *assms* **by** *auto*
hence *2*: ⊢ (*init w*) ∧ (*f*; *g*) ⟶ (*f1* ∧ *fin* (*init w1*)); *g* **by** (*rule StateAndChopImpChopRule*)
have *3*: ⊢ (*f1* ∧ *fin* (*init w1*)); *g* = *f1*; ((*init w1*) ∧ *g*) **by** (*rule AndFinChopEqvStateAndChop*)
have *4*: ⊢ (*init w1*)∧ *g* ⟶ *g1* **using** *assms* **by** *auto*
hence *5*: ⊢ *f1*; ((*init w1*) ∧ *g*) ⟶ *f1*; *g1* **by** (*rule RightChopImpChop*)
from *2 3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopRepAndFin*:
assumes ⊢ (*init w*) ∧ *f* ⟶ *f1* ∧ *fin* (*init w1*)
⊢ (*init w1*) ∧ *g* ⟶ *g1* ∧ *fin* (*init w2*)
shows ⊢ (*init w*) ∧ (*f*; *g*) ⟶ (*f1*; *g1*) ∧ *fin* (*init w2*)
proof −
have *1*: ⊢ (*init w*) ∧ *f* ⟶ *f1* ∧ *fin* (*init w1*) **using** *assms* **by** *auto*
have *2*: ⊢ (*init w1*) ∧ *g* ⟶ *g1* ∧ *fin* (*init w2*) **using** *assms* **by** *auto*
have *3*: ⊢ (*init w*) ∧ (*f*; *g*) ⟶ *f1*; (*g1* ∧ *fin* (*init w2*)) **using** *1 2* **by** (*rule ChopRep*)
have *4*: ⊢ *f1*; (*g1* ∧ *fin* (*init w2*)) ⟶ *f1*; *g1* **by** (*rule ChopAndA*)
have *5*: ⊢ *f1*; (*g1* ∧ *fin* (*init w2*)) ⟶ *f1*; *fin* (*init w2*) **by** (*rule ChopAndB*)
have *6*: ⊢ *f1*; *fin* (*init w2*) ⟶ *fin* (*init w2*) **by** (*rule ChopFinImpFin*)
from *1 2 3 4 5 6* **show** *?thesis* **using** *ChopRep ChopRule* **by** *fastforce*
qed

lemma *TrueChopMoreEqvMore*:
⊢ #*True* ; *more* = *more*
by (*metis ChopMoreImpMore NowImpDiamond TrueChopEqvDiamond int-eq int-iffI*)

lemma *MoreChopLoop*:
assumes ⊢ *f* ⟶ *more* ; *f*
shows ⊢ ¬ *f*
proof −
have *1*: ⊢ *f* ⟶ *more* ; *f*
using *assms* **by** *auto*
hence *11*: ⊢ ◇ *f* ⟶ ◇ (*more*;*f*)
by (*rule DiamondImpDiamond*)

have *12*: ⊢ ◇ (*more*;*f*) = #*True*;(*more*;*f*)
  by (*simp add*: *sometimes-d-def*)
have *13*: ⊢ #*True*;(*more*;*f*) = (#*True*;*more*);*f*
  by (*rule ChopAssoc*)
have *14*: ⊢ ◇ (*more*;*f*) = *more*;*f*
  using *TrueChopMoreEqvMore 12 13* **by** (*metis int-eq*)
have *2*: ⊢ *more* ; *f* = ○(◇ *f*)
  by (*rule MoreChopEqvNextDiamond*)
have *3*: ⊢ ◇ *f* ⟶ ○(◇ *f*)
  using *11 14 2* **by** *fastforce*
hence *4*: ⊢ ¬ (◇ *f*)
  by (*rule NextLoop*)
have *5*: ⊢ ¬ (◇ *f*) ⟶ ¬ *f*
  using *NowImpDiamond* **by** *fastforce*
from *4 5* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *MoreChopContra*:
assumes ⊢ *f* ∧ ¬ *g* ⟶ (*more* ; (*f* ∧ ¬ *g*))
shows ⊢ *f* ⟶ *g*
proof −
have *1*: ⊢ *f* ∧ ¬ *g* ⟶ (*more* ; (*f* ∧ ¬ *g*)) **using** *assms* **by** *auto*
hence *2*: ⊢ ¬ (*f* ∧ ¬ *g*) **by** (*rule MoreChopLoop*)
from *2* **show** *?thesis* **by** *auto*
qed

lemma *ChopLoop*:
assumes ⊢ *f* ⟶ *g*;*f*
  ⊢ *g* ⟶ *more*
shows ⊢ ¬ *f*
proof −
have *1*: ⊢ *f* ⟶ *g*; *f* **using** *assms* **by** *auto*
have *2*: ⊢ *g* ⟶ *more* **using** *assms* **by** *auto*
hence *3*: ⊢ *g*; *f* ⟶ *more* ; *f* **by** (*rule LeftChopImpChop*)
have *4*: ⊢ *f* ⟶ *more* ; *f* **using** *1 3* **by** *fastforce*
from *4* **show** *?thesis* **using** *MoreChopLoop* **by** *auto*
qed

lemma *ChopContra*:
assumes ⊢ *f* ∧ ¬ *g* ⟶ *h*; *f* ∧ ¬ (*h*; *g*)
  ⊢ *h* ⟶ *more*
shows ⊢ *f* ⟶ *g*
proof −
have *1*: ⊢ *f* ∧ ¬ *g* ⟶ *h*; *f* ∧ ¬ (*h*; *g*) **using** *assms* **by** *auto*
have *2*: ⊢ *h* ⟶ *more* **using** *assms* **by** *auto*
have *3*: ⊢ *h*; *f* ∧ ¬ (*h*; *g*) ⟶ *h*; (*f* ∧ ¬ *g*) **by** (*rule ChopAndNotChopImp*)
have *4*: ⊢ *h*; (*f* ∧ ¬ *g*) ⟶ *more* ; (*f* ∧ ¬ *g*) **using** *2* **by** (*rule LeftChopImpChop*)
have *5*: ⊢ *f* ∧ ¬ *g* ⟶ *more* ; (*f* ∧ ¬ *g*) **using** *1 3 4* **by** *fastforce*
from *5* **show** *?thesis* **using** *MoreChopContra* **by** *auto*
qed

5.7 Properties of Chopstar and Chopplus

lemma *EmptyImpCS*:
$\vdash$ *empty* $\longrightarrow f^\star$
proof –
 have *1*: $\vdash f^\star = (\textit{empty} \lor (f \land \textit{more});f^\star)$ **by** (*rule ChopstarEqv*)
 have *2*: $\vdash \textit{empty} \longrightarrow \textit{empty} \lor (f \land \textit{more});f^\star$ **by** *auto*
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *CSEqvOrChopCS*:
$\vdash\ f^\star = (\textit{empty} \lor (f;\ f^\star))$
proof –
 have *1*: $\vdash f^\star = (\textit{empty} \lor (f \land \textit{more});f^\star)$ **by** (*rule ChopstarEqv*)
 have *2*: $\vdash (f \land \textit{more});f^\star \longrightarrow f;f^\star$ **by** (*rule AndChopA*)
 have *3*: $\vdash f^\star \longrightarrow \textit{empty} \lor f;\ f^\star$ **using** *1 2* **by** (*metis int-iffD1 Prop08*)
 have *4*: $\vdash \textit{empty} \longrightarrow f^\star$ **by** (*rule EmptyImpCS*)
 have *5*: $\vdash f \longrightarrow \textit{empty} \lor (f \land \textit{more})$ **by** (*auto simp*: *empty-d-def*)
 have *6*: $\vdash f;\ f^\star \longrightarrow f^\star \lor (f \land \textit{more});\ f^\star$ **using** *5* **by** (*rule EmptyOrChopImpRule*)
 have *7*: $\vdash f^\star \longrightarrow \textit{empty} \lor (f \land \textit{more});f^\star$ **using** *1* **by** *fastforce*
 have *8*: $\vdash f;\ f^\star \longrightarrow \textit{empty} \lor (f \land \textit{more});\ f^\star$ **using** *6 7* **by** *fastforce*
 hence *9*: $\vdash f;\ f^\star \longrightarrow f^\star$ **using** *1* **by** *fastforce*
 have *10*: $\vdash \textit{empty} \lor f;\ f^\star \longrightarrow f^\star$ **using** *9 4* **by** *fastforce*
 from *3 10* **show** *?thesis* **by** *fastforce*
qed

lemma *CSAndMoreEqvAndMoreChop*:
$\vdash (f^\star \land \textit{more}) = (f \land \textit{more});\ f^\star$
proof –
 have *1*: $\vdash (\textit{empty} \lor (f \land \textit{more});\ f^\star) \land \textit{more} \longrightarrow (f \land \textit{more});\ f^\star$
  by (*auto simp*: *empty-d-def*)
 have *2*: $\vdash f^\star = (\textit{empty} \lor (f \land \textit{more});f^\star)$
  by (*rule ChopstarEqv*)
 have *3*: $\vdash f^\star \land \textit{more} \longrightarrow (f \land \textit{more});\ f^\star$
  using *1 2* **by** *fastforce*
 have *4*: $\vdash (f \land \textit{more});\ f^\star \longrightarrow f^\star$
  using *2* **by** *fastforce*
 have *5*: $\vdash (f \land \textit{more}) \longrightarrow \textit{more}$
  by *auto*
 hence *6*: $\vdash (f \land \textit{more});\ f^\star \longrightarrow \textit{more}$
  by (*rule LeftChopImpMoreRule*)
 have *7*: $\vdash (f \land \textit{more});\ f^\star \longrightarrow f^\star \land \textit{more}$
  using *4 6* **by** *fastforce*
 from *3 7* **show** *?thesis* **by** *fastforce*
qed

lemma *CSAndMoreImpChopCS*:
$\vdash\ f^\star \land \textit{more} \longrightarrow f;\ f^\star$
proof –
 have *1*: $\vdash (f^\star \land \textit{more}) = (f \land \textit{more});\ f^\star$ **by** (*rule CSAndMoreEqvAndMoreChop*)
 have *2*: $\vdash (f \land \textit{more});\ f^\star \longrightarrow f;\ f^\star$ **by** (*rule AndChopA*)

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

lemma *NotAndMoreEqvEmptyOr*:
$\vdash \neg\, (f \wedge \mathit{more}) = (\mathit{empty} \vee \neg f)$
by (*auto simp*: *empty-d-def*)

lemma *MoreAndEmptyOrEqvMoreAnd*:
$\vdash (\mathit{more} \wedge (\mathit{empty} \vee \neg f)) = (\mathit{more} \wedge \neg\, f)$
by (*auto simp*: *empty-d-def*)

lemma *CSMoreNotImpChopCSAndMore*:
$\vdash\ \ f^{\star} \wedge\ \mathit{more}\ \wedge \neg\ \ f \longrightarrow (f \wedge\ \mathit{more}\ );\ (f^{\star} \wedge\ \mathit{more}\ )$
proof −
 have *1*: $\vdash (f^{\star} \wedge\ \mathit{more})\ = (f \wedge\ \mathit{more}\ );\ f^{\star}$
  by (*rule CSAndMoreEqvAndMoreChop*)
 have *2*: $\vdash\ \mathit{empty}\ \vee\ \ \mathit{more}$
  by (*auto simp*: *empty-d-def*)
 hence *3*: $\vdash f^{\star} \longrightarrow\ \mathit{empty}\ \vee\ (f^{\star} \wedge\ \mathit{more}\ )$
  by *auto*
 hence *4*: $\vdash (f \wedge\ \mathit{more}\ );\ f^{\star} \longrightarrow (f \wedge\ \mathit{more}\ ) \vee\ ((f \wedge\ \mathit{more}\ );\ (f^{\star} \wedge\ \mathit{more}\ ))$
  by (*rule ChopEmptyOrImpRule*)
 hence *5*: $\vdash (f \wedge\ \mathit{more}\ );\ f^{\star} \wedge \neg(f \wedge \mathit{more}) \longrightarrow\ \ ((f \wedge\ \mathit{more}\ );\ (f^{\star} \wedge\ \mathit{more}\ ))$
  by *fastforce*
 have *6*: $\vdash (f \wedge\ \mathit{more}\ );\ f^{\star} =\ ((f \wedge\ \mathit{more}\ );\ f^{\star} \wedge \mathit{more})$ **using** *1*
  by *auto*
 have *7*: $\vdash ((f \wedge\ \mathit{more}\ );\ f^{\star} \wedge \neg(f \wedge \mathit{more})) = ((f \wedge\ \mathit{more}\ );\ f^{\star} \wedge \mathit{more} \wedge \neg(f \wedge \mathit{more}))$
  using *6* **by** *auto*
 have *8*: $\vdash (f \wedge\ \mathit{more}\ );\ f^{\star} \wedge\ \mathit{more}\ \wedge \neg\ \ f \longrightarrow (f \wedge\ \mathit{more}\ );\ (f^{\star} \wedge\ \mathit{more}\ )$
  using *5 7* **by** *auto*
 have *9*: $\vdash (f^{\star} \wedge\ \mathit{more}\ \wedge \neg\ \ f) = ((f^{\star} \wedge\ \mathit{more}) \wedge (\mathit{more}\ \wedge \neg\ \ f))$
  by *auto*
 have *10*: $\vdash ((f^{\star} \wedge\ \mathit{more}) \wedge (\mathit{more}\ \wedge \neg\ \ f)) = ((f \wedge\ \mathit{more}\ );\ f^{\star} \wedge\ (\mathit{more}\ \wedge \neg\ \ f))$
  using *1* **by** *fastforce*
 from *1 8 9 10* **show** *?thesis* **by** *fastforce*
qed

lemma *CSAndMoreImpCSChop*:
$\vdash\ \ f^{\star} \wedge\ \mathit{more}\ \longrightarrow f^{\star};\ f$
proof −
 have *1*: $\vdash (f^{\star} \wedge\ \mathit{more})\ = (f \wedge\ \mathit{more}\ );\ f^{\star}$
  by (*rule CSAndMoreEqvAndMoreChop*)
 have *2*: $\vdash\ \mathit{empty}\ \vee\ \ \mathit{more}$
  by (*auto simp*: *empty-d-def*)
 hence *3*: $\vdash f^{\star} \longrightarrow\ \mathit{empty}\ \vee\ (f^{\star} \wedge\ \mathit{more}\ )$
  by *auto*
 hence *4*: $\vdash (f \wedge\ \mathit{more}\ );\ f^{\star} \longrightarrow$
   $(f \wedge\ \mathit{more}\ ) \vee\ ((f \wedge\ \mathit{more}\ );\ (f^{\star} \wedge \mathit{more}\ ))$
  by (*rule ChopEmptyOrImpRule*)
 have *5*: $\vdash f^{\star} \wedge\ \mathit{more}\ \wedge \neg\ \ f \longrightarrow (f \wedge\ \mathit{more}\ );\ (f^{\star} \wedge\ \mathit{more}\ )$

by (*rule CSMoreNotImpChopCSAndMore*)
have *6*: $\vdash f^{\star} = (empty \vee (f \wedge more); f^{\star})$
by (*rule ChopstarEqv*)
hence *7*: $\vdash f^{\star}; f = (f \vee ((f \wedge more); f^{\star}); f)$
by (*rule EmptyOrChopEqvRule*)
have *8*: $\vdash (f \wedge more); (f^{\star}; f) = ((f \wedge more); f^{\star}); f$
by (*rule ChopAssoc*)
have *9*: $\vdash (f^{\star} \wedge more) \wedge \neg (f^{\star}; f) \longrightarrow$
$(f \wedge more); (f^{\star} \wedge more) \wedge \neg ((f \wedge more); (f^{\star}; f))$
using *5 7 8* **by** *fastforce*
have *10*: $\vdash f \wedge more \longrightarrow more$
by *auto*
from *9 10* **show** *?thesis* **by** (*rule ChopContra*)
qed

lemma *NotEmptyEqvMore*:
$\vdash \neg empty = more$
by (*simp add*: *empty-d-def*)

lemma *NotCSImpMore*:
$\vdash \neg (f^{\star}) \longrightarrow more$
proof −
have *1*: $\vdash empty \longrightarrow (f^{\star})$ **using** *EmptyImpCS* **by** *blast*
hence *2*: $\vdash \neg empty \vee (f^{\star})$ **by** *fastforce*
from *2* **show** *?thesis* **using** *1 NotEmptyEqvMore* **by** *fastforce*
qed

lemma *CSChopCSImpCS*:
$\vdash f^{\star}; f^{\star} \longrightarrow f^{\star}$
proof −
have *1*: $\vdash f^{\star} = (empty \vee (f \wedge more); f^{\star})$
by (*rule ChopstarEqv*)
hence *2*: $\vdash f^{\star}; f^{\star} = (f^{\star} \vee ((f \wedge more); f^{\star}); f^{\star})$
by (*rule EmptyOrChopEqvRule*)
have *21*: $\vdash f^{\star}; f^{\star} \wedge \neg (f^{\star}) \longrightarrow ((f \wedge more); f^{\star}); f^{\star}$
using *2* **by** *auto*
have *22*: $\vdash \neg (f^{\star}) = (\neg empty \wedge \neg ((f \wedge more); f^{\star}))$
using *1* **by** *fastforce*
have *23*: $\vdash \neg (f^{\star}) \longrightarrow \neg ((f \wedge more); f^{\star})$
using *2 22* **by** *fastforce*
have *24*: $\vdash f^{\star}; f^{\star} \wedge \neg (f^{\star}) \longrightarrow \neg (f^{\star})$
by *auto*
have *25*: $\vdash f^{\star}; f^{\star} \wedge \neg (f^{\star}) \longrightarrow \neg ((f \wedge more); f^{\star})$
using *23 24 MP* **by** *auto*
have *3*: $\vdash f^{\star}; f^{\star} \wedge \neg (f^{\star}) \longrightarrow ((f \wedge more); f^{\star}); f^{\star} \wedge \neg ((f \wedge more); f^{\star})$
using *21 25* **by** *fastforce*
have *4*: $\vdash (f \wedge more);(f^{\star}; f^{\star}) = ((f \wedge more); f^{\star}); f^{\star}$
by (*rule ChopAssoc*)
have *5*: $\vdash f^{\star}; f^{\star} \wedge \neg (f^{\star}) \longrightarrow (f \wedge more); (f^{\star}; f^{\star}) \wedge \neg ((f \wedge more); f^{\star})$

using *3 4* **by** *fastforce*
have *6*: $\vdash f \wedge$ *more* $\longrightarrow$ *more*
by *auto*
from *5 6* **show** *?thesis* **using** *ChopContra* **by** *blast*
qed

lemma *ImpChopPlus*:
$\vdash\ f \longrightarrow f;f^\star$
proof −
have *1*: $\vdash f^\star = (empty \vee\ f;\ f^\star)$ **by** (*rule CSEqvOrChopCS*)
hence *2*: $\vdash f;f^\star = (f;empty \vee\ f;\ (f;f^\star))$ **using** *ChopOrEqvRule* **by** *blast*
have *3*: $\vdash f;empty = f$ **using** *ChopEmpty* **by** *blast*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *ImpCS*:
$\vdash\ f \longrightarrow f^\star$
proof −
have *1*: $\vdash f \longrightarrow f;f^\star$ **by** (*rule ImpChopPlus*)
hence *2*: $\vdash f \longrightarrow\ empty\ \vee\ f;f^\star$ **by** *auto*
from *2* **show** *?thesis* **using** *CSEqvOrChopCS* **by** *fastforce*
qed

lemma *CSChopImpCS*:
$\vdash\ f^\star;\ f \longrightarrow f^\star$
proof −
have *1*: $\vdash f \longrightarrow f^\star$ **by** (*rule ImpCS*)
hence *2*: $\vdash f^\star;\ f \longrightarrow f^\star;\ f^\star$ **by** (*rule RightChopImpChop*)
have *3*: $\vdash f^\star;\ f^\star \longrightarrow f^\star$ **by** (*rule CSChopCSImpCS*)
from *2 3* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *ChopPlusImpCS*:
$\vdash\ f;f^\star \longrightarrow f^\star$
proof −
have *1*: $\vdash\ f;f^\star \longrightarrow empty\ \vee f;f^\star$ **by** *auto*
from *1* **show** *?thesis* **using** *CSEqvOrChopCS* **by** *fastforce*
qed

lemma *CSChopEqvOrChopPlusChop*:
$\vdash\ f^\star;\ g = (g \vee\ (f;f^\star)\ ;\ g)$
proof −
have *1*: $\vdash f^\star =\ (empty\ \vee\ f;f^\star)$ **by** (*rule CSEqvOrChopCS*)
from *1* **show** *?thesis* **using** *EmptyOrChopEqvRule* **by** *blast*
qed

lemma *CSElim*:
assumes $\vdash\ empty \longrightarrow g$
$\vdash (f \wedge\ more\ );\ g \longrightarrow g$

shows $\vdash f^\star \longrightarrow g$
proof −
have *1*: $\vdash f^\star = (empty \lor (f \land more); f^\star)$
by (*rule ChopstarEqv*)
have *2*: $\vdash empty \longrightarrow g$
using *assms* **by** *blast*
have *3*: $\vdash (f \land more); g \longrightarrow g$
using *assms* **by** *blast*
have *31*: $\vdash \neg g \longrightarrow more$
using *2* **by** (*auto simp*: *empty-d-def*)
have *32*: $\vdash \neg g \longrightarrow \neg ((f \land more); g)$
using *3* **by** *fastforce*
have *33*: $\vdash f^\star \land more \longrightarrow (f \land more); f^\star$
using *1* **using** *CSAndMoreEqvAndMoreChop* **by** *fastforce*
have *34*: $\vdash f^\star \land \neg g \longrightarrow f^\star \land more$
using *31* **by** *auto*
have *35*: $\vdash f^\star \land \neg g \longrightarrow (f \land more); f^\star$
using *33 34* **by** *fastforce*
have *36*: $\vdash f^\star \land \neg g \longrightarrow \neg ((f \land more); g)$
using *32* **by** *auto*
have *4*: $\vdash f^\star \land \neg g \longrightarrow (f \land more); f^\star \land \neg ((f \land more); g)$
using *35 36* **by** *fastforce*
have *5*: $\vdash f \land more \longrightarrow more$
by *auto*
from *4 5* **show** *?thesis* **using** *ChopContra* **by** *blast*
qed

lemma *CSCSImpCS*:
$\vdash (f^\star)^\star \longrightarrow f^\star$
proof −
have *1*: $\vdash empty \longrightarrow f^\star$ **by** (*rule EmptyImpCS*)
have *2*: $\vdash (f^\star \land more); f^\star \longrightarrow f^\star; f^\star$ **by** (*rule AndChopA*)
have *3*: $\vdash f^\star; f^\star \longrightarrow f^\star$ **by** (*rule CSChopCSImpCS*)
have *4*: $\vdash (f^\star \land more); f^\star \longrightarrow f^\star$ **using** *2 3 lift-imp-trans* **by** *blast*
from *1 4* **show** *?thesis* **using** *CSElim* **by** *blast*
qed

lemma *RightEmptyOrChopEqv*:
$\vdash g;(empty \lor f) = (g \lor (g; f))$
proof −
have *1*: $\vdash g;(empty \lor f) = (g;empty \lor g;f)$ **by** (*rule ChopOrEqv*)
have *2*: $\vdash g;empty = g$ **by** (*rule ChopEmpty*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RightEmptyOrChopEqvRule*:
assumes $\vdash f = (empty \lor f1)$
shows $\vdash g;f = (g \lor (g;f1))$
proof −

have *1*: ⊢ f = (*empty* ∨ *f1*) **using** *assms* **by** *auto*
hence *2*: ⊢ g;f = g;(*empty* ∨ *f1*) **by** (*rule RightChopEqvChop*)
have *3*: ⊢ g;(*empty* ∨ *f1*) = (g ∨ (g;*f1*)) **by** (*rule RightEmptyOrChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopPlusEqvOrChopChopPlus*:
⊢ (f;$f^\star$) = (f ∨ f; (f;$f^\star$))
proof −
have *1*: ⊢ $f^\star$ = (*empty* ∨ f;$f^\star$) **by** (*rule CSEqvOrChopCS*)
from *1* **show** *?thesis* **by** (*rule RightEmptyOrChopEqvRule*)
qed

lemma *CSAndEmptyEqvEmpty*:
⊢ (($f^\star$) ∧ *empty*) = *empty*
using *EmptyImpCS* **by** *fastforce*

lemma *NotAndMoreChopAndEmpty*:
⊢ ¬(((f ∧ *more*);g) ∧ *empty*)
by (*metis AndChopB MoreChopImpMore empty-d-def intensional-simps*(*14*) *intensional-simps*(*25*) *lift-imp-trans Prop07*)

lemma *NotChopAndMoreAndEmpty*:
⊢ ¬((f;(g∧ *more*)) ∧ *empty*)
by (*metis ChopAndB ChopMoreImpMore empty-d-def intensional-simps*(*14*) *intensional-simps*(*25*) *lift-imp-trans Prop07*)

lemma *ChopCsAndEmptyEqvAndEmpty*:
⊢ ((f;$f^\star$) ∧ *empty*) = (f ∧ *empty*)
proof −
have *1*: ⊢ ((f;$f^\star$) ∧ *empty*) = (f∧ *empty*);($f^\star$ ∧ *empty*)
using *ChopAndEmptyEqvEmptyChopEmpty* **by** *blast*
have *2*: ⊢ (f∧ *empty*);($f^\star$ ∧ *empty*) = (f∧ *empty*);*empty*
using *CSAndEmptyEqvEmpty* **using** *RightChopEqvChop* **by** *blast*
have *3*: ⊢ (f∧ *empty*);*empty* = (f∧ *empty*)
by (*rule ChopEmpty*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *AndMoreChopAndMoreEqvAndMoreChop*:
⊢ ((f ∧ *more*);g ∧ *more*) = (f ∧ *more*);g
using *ChopImpDi DiAndB DiMoreEqvMore* **by** *fastforce*

lemma *ChopPlusEqv*:
⊢ (f;$f^\star$) = (f ∨ (f∧ *more*); (f;$f^\star$))
proof −
have *1*: ⊢ $f^\star$ = (*empty* ∨ (f ∧ *more*); $f^\star$)
by (*rule ChopstarEqv*)
have *2*: ⊢ $f^\star$ = (*empty* ∨ f;$f^\star$)
by (*rule CSEqvOrChopCS*)

hence *3*: $\vdash$ (*empty* $\vee$ *f*;*f**) = (*empty* $\vee$ (*f* $\wedge$ *more*);*f**)
  using *1 2* **by** *fastforce*
have *4*: $\vdash$ (*f* $\wedge$ *more*);(*f**) = (*f* $\wedge$ *more*);(*empty* $\vee$ *f*;*f**)
  using *2* **using** *RightChopEqvChop* **by** *blast*
hence *5*: $\vdash$ *empty* $\vee$ *f*;*f** = *empty* $\vee$ (*f* $\wedge$ *more*);(*empty* $\vee$ *f*;*f**)
  using *3 4* **by** *fastforce*
have *6*: $\vdash$ (*f* $\wedge$ *more*); (*empty* $\vee$ *f*;*f**) =
    ((*f* $\wedge$ *more*); *empty* $\vee$ (*f* $\wedge$ *more*); (*f*;*f**))
  using *ChopOrEqv* **by** *blast*
have *7*: $\vdash$ (*f* $\wedge$ *more*); *empty* = (*f* $\wedge$ *more*)
  using *ChopEmpty* **by** *blast*
have *8*: $\vdash$ (*empty* $\vee$ *f*;*f**) =
    (*empty* $\vee$ (*f* $\wedge$ *more*) $\vee$ (*f* $\wedge$ *more*); (*f*;*f**))
  using *5 6 7* **by** (*metis 2 3 inteq-reflection*)
have *9*: $\vdash$ ((*empty* $\vee$ *f*;*f**) $\wedge$ *more*) = (*f*;*f** $\wedge$ *more*)
  by (*auto simp*: *empty-d-def*)
have *10*: $\vdash$ ((*empty* $\vee$ (*f* $\wedge$ *more*) $\vee$ (*f* $\wedge$ *more*); (*f*;*f**)) $\wedge$ *more*) =
    (((*f* $\wedge$ *more*) $\vee$ (*f* $\wedge$ *more*); (*f*;*f**)) $\wedge$ *more*)
  by (*auto simp*: *empty-d-def*)
have *11*: $\vdash$ (((*f* $\wedge$ *more*) $\vee$ (*f* $\wedge$ *more*); (*f*;*f**)) $\wedge$ *more*) =
    ((*f* $\wedge$ *more*) $\vee$ (*f* $\wedge$ *more*); (*f*;*f**))
  using *10 6 7 int-eq*
  using *AndMoreChopAndMoreEqvAndMoreChop* **by** *fastforce*
have *12*: $\vdash$ (*f*;*f** $\wedge$ *more*) = ((*f* $\wedge$ *more*) $\vee$ (*f* $\wedge$ *more*); (*f*;*f**))
  using *8 9 10 11* **by** *fastforce*
have *13*: $\vdash$ (*f*;*f** $\wedge$ *empty*) = (*f* $\wedge$ *empty*)
  by (*rule ChopCsAndEmptyEqvAndEmpty*)
have *14*: $\vdash$ ((*f* $\wedge$ *more*) $\vee$ (*f* $\wedge$ *more*); (*f*;*f**) $\vee$ (*f* $\wedge$ *empty*)) =
    (*f* $\vee$ (*f* $\wedge$ *more*);(*f*;*f**))
  by (*auto simp*: *empty-d-def*)
have *15*: $\vdash$ *f*;*f** = ((*f*;*f** $\wedge$ *empty*) $\vee$ (*f*;*f** $\wedge$ *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*$\wedge$ *more*); (*f*;*f**))
  by (*rule ChopPlusEqv*)
have *3*: $\vdash$ *g*;*g** = (*g* $\vee$ (*g* $\wedge$ *more*);(*g*;*g**))
  by (*rule ChopPlusEqv*)
have *4*: $\vdash$ *f*;*f** $\wedge$ $\neg$ (*g*;*g**) $\longrightarrow$ ((*f*$\wedge$ *more*); (*f*;*f**)) $\wedge$ $\neg$ ((*g* $\wedge$ *more*); (*g*;*g**))
  using *1 2 3* **by** *fastforce*
have *5*: $\vdash$ *f* $\wedge$ *more* $\longrightarrow$ *g* $\wedge$ *more* **using** *1*
  by *auto*

have *6*: $\vdash (f \wedge \ more\ ); (f;f^{\star}) \longrightarrow (g \wedge \ more\ ); (f;f^{\star})$
using *5* **by** (*rule LeftChopImpChop*)
have *7*: $\vdash f;f^{\star} \wedge \neg\, (g;g^{\star}\ ) \longrightarrow$
$((g \wedge \ more\ ); (f;f^{\star})\ ) \wedge \neg\, ((g \wedge \ more\ ); (g;g^{\star})\ )$
using *4 6* **by** *fastforce*
have *8*: $\vdash g \wedge \ more \ \longrightarrow \ more$
by *auto*
from *7 8* **show** *?thesis* **using** *ChopContra* **by** *blast*
qed

lemma *ChopChopPlusImpChopPlus*:
$\vdash \ f; (f;f^{\star}) \longrightarrow f;f^{\star}$
proof –
have *1*: $\vdash \ empty \ \vee \ \ more$ **by** (*auto simp*: *empty-d-def*)
hence *2*: $\vdash f \longrightarrow \ empty \ \vee \ (f \wedge \ more\ )$ **by** *auto*
hence *3*: $\vdash f; (f;f^{\star}) \ \longrightarrow (f;f^{\star}) \ \vee \ (f \wedge \ more\ );(f;f^{\star})$ **by** (*rule EmptyOrChopImpRule*)
have *4*: $\vdash \ f;f^{\star} = (f \vee \ (f \wedge \ more\ ); (f;f^{\star}))$ **by** (*rule ChopPlusEqv*)
hence *5*: $\vdash (f \wedge \ more\ ); (f;f^{\star}) \longrightarrow f;f^{\star}$ **by** *auto*
from *3 5* **show** *?thesis* **using** *ChopPlusImpCS RightChopImpChop* **by** *blast*
qed

lemma *CSImpCS*:
assumes $\vdash \ f \longrightarrow g$
shows $\vdash f^{\star} \longrightarrow g^{\star}$
proof –
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;f^{\star} \longrightarrow g;g^{\star}$ **by** (*rule ChopPlusImpChopPlus*)
hence *3*: $\vdash \ empty \ \vee \ f;f^{\star} \ \longrightarrow \ empty \ \vee \ \ g;g^{\star}$ **by** *auto*
from *2 3* **show** *?thesis* **using** *CSEqvOrChopCS* **by** (*metis inteq-reflection*)
qed

lemma *ChopPlusIntro*:
assumes $\vdash \ f \wedge \neg \ g \longrightarrow (g \wedge \ more\ ); f$
shows $\vdash f \longrightarrow g;g^{\star}$
proof –
have *1*: $\vdash f \wedge \neg \ g \longrightarrow (g \wedge \ more\ ); f$ **using** *assms* **by** *auto*
have *2*: $\vdash g;g^{\star} \ = (g \vee \ (g \wedge \ more\ ); (g;g^{\star}))$ **by** (*rule ChopPlusEqv*)
have *3*: $\vdash f \wedge \neg\, (g;g^{\star}\ ) \longrightarrow$
$(g \wedge \ more\ ); f \wedge \neg\, ((g \wedge \ more\ ); (g;g^{\star})\ )$ **using** *1 2* **by** *fastforce*
have *4*: $\vdash g \wedge \ more \ \longrightarrow \ more$ **by** *auto*
from *3 4* **show** *?thesis* **using** *ChopContra* **by** *blast*
qed

lemma *ChopPlusElim*:
assumes $\vdash \ f \longrightarrow g$
$\vdash (f \wedge \ more\ ); g \longrightarrow g$
shows $\vdash f;f^{\star} \ \longrightarrow g$
proof –
have *1*: $\vdash f;f^{\star} \ = (f \vee \ (f \wedge \ more\ ); (f;f^{\star}))$ **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*: $\vdash (f \wedge\ more\ );\ g \longrightarrow g$ **using** *assms* **by** *blast*
hence *31*: $\vdash \neg\ g \longrightarrow \neg\ ((f \wedge\ more\ );\ g)$ **by** *fastforce*
hence *32*: $\vdash f;f^\star \wedge \neg\ g \longrightarrow \neg\ ((f \wedge\ more\ );\ g)$ **by** *auto*
have *33*: $\vdash f;f^\star \wedge \neg\ g \longrightarrow (f \wedge\ more\ );\ (f;f^\star)$ **using** *1 21* **by** *fastforce*
have *4*: $\vdash f;f^\star \wedge \neg\ g \longrightarrow$
$(f \wedge\ more\ );\ (f;f^\star) \wedge \neg\ ((f \wedge\ more\ );\ g)$ **using** *31 33* **by** *fastforce*
have *5*: $\vdash f \wedge\ more \longrightarrow more$ **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 $\vdash f;f^\star \longrightarrow g$
proof −
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *blast*
have *2*: $\vdash (f;\ g) \longrightarrow g$ **using** *assms* **by** *blast*
have *3*: $\vdash (f \wedge\ more\ );\ g \longrightarrow f;\ g$ **by** (*rule AndChopA*)
have *4*: $\vdash (f \wedge\ more\ );\ g \longrightarrow g$ **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 $\vdash f;f^\star = g;g^\star$
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \longrightarrow g$ **by** *auto*
hence *3*: $\vdash f;f^\star \longrightarrow g;g^\star$ **by** (*rule ChopPlusImpChopPlus*)
have *4*: $\vdash g \longrightarrow f$ **using** *1* **by** *auto*
hence *5*: $\vdash g;g^\star \longrightarrow f;f^\star$ **by** (*rule ChopPlusImpChopPlus*)
from *3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *CSEqvCS*:
assumes $\vdash f = g$
shows $\vdash f^\star = g^\star$
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;f^\star = g;g^\star$ **by** (*rule ChopPlusEqvChopPlus*)
hence *3*: $\vdash (empty\ \vee f;f^\star) = (empty\ \vee\ g;g^\star)$ **by** *auto*
from *3* **show** *?thesis* **using** *CSEqvOrChopCS* **by** (*metis int-eq*)
qed

lemma *AndCSA*:
$\vdash (f \wedge g)^\star \longrightarrow f^\star$
proof −
have *1*: $\vdash f \wedge g \longrightarrow f$ **by** *auto*
from *1* **show** *?thesis* **using** *CSImpCS* **by** *blast*

qed

lemma *AndCSB*:
$\vdash \ (f \wedge g)^\star \longrightarrow g^\star$
proof –
 have *1*: $\vdash f \wedge g \longrightarrow g$ **by** *auto*
 from *1* **show** *?thesis* **using** *CSImpCS* **by** *blast*
qed

lemma *CSIntro*:
 assumes $\vdash \ f \wedge \ more \ \longrightarrow (g \wedge \ more\ );\ f$
 shows $\vdash f \longrightarrow g^\star$
proof –
 have *1*: $\vdash f \wedge \ more \ \longrightarrow (g \wedge \ more\ );\ f$
  using *assms* **by** *auto*
 have *2*: $\vdash \ more \ = \neg \ empty$
  by (*auto simp*: *empty-d-def*)
 have *3*: $\vdash f \wedge \neg \ empty \ \longrightarrow (g \wedge \ more\ );\ f$
  using *1 2* **by** *fastforce*
 have *4*: $\vdash g^\star = \ (empty \ \vee \ (g \wedge \ more\ );\ g^\star)$
  by (*rule ChopstarEqv*)
 hence *41*: $\vdash (\neg(empty \ \vee \ (g \wedge \ more\ );\ g^\star)) = (\neg empty \wedge \neg((g \wedge \ more\ );\ g^\star))$
  by *fastforce*
 have *411*: $\vdash (\neg empty \wedge \neg((g \wedge \ more\ );\ g^\star)) = (more \wedge \neg((g \wedge \ more\ );\ g^\star))$
  using *NotEmptyEqvMore* **by** *fastforce*
 have *42*: $\vdash \neg(\ g^\star) \ = (more \wedge \neg((g \wedge \ more\ );\ g^\star))$
  using *4 41 411* **by** *fastforce*
 have *43*: $\vdash f \wedge \neg(\ g^\star) \longrightarrow f \wedge more \wedge \neg\ ((g \wedge \ more\ );\ g^\star)$
  using *42* **by** *fastforce*
 have *44*: $\vdash f \wedge more \wedge \neg\ ((g \wedge \ more\ );\ g^\star) \longrightarrow (g \wedge \ more\ );\ f \wedge \neg\ ((g \wedge \ more\ );\ g^\star)$
  using *3 43 1* **by** *auto*
 have *5*: $\vdash f \wedge \neg\ (\ g^\star) \longrightarrow$
   $(g \wedge \ more\ );\ f \wedge \neg\ ((g \wedge \ more\ );\ g^\star)$
  using *43 44 lift-imp-trans* **by** *fastforce*
 have *6*: $\vdash g \wedge \ more \ \longrightarrow \ more$
  by *auto*
 from *5 6* **show** *?thesis* **using** *ChopContra* **by** *blast*
qed

lemma *CSElimWithoutMore*:
 assumes $\vdash \ \ empty \ \longrightarrow g$
  $\vdash f;\ g \longrightarrow g$
 shows $\vdash f^\star \longrightarrow g$
proof –
 have *1*: $\vdash \ empty \ \longrightarrow g$ **using** *assms* **by** *blast*
 have *2*: $\vdash f;\ g \longrightarrow g$ **using** *assms* **by** *blast*
 have *3*: $\vdash (f \wedge \ more\ );\ g \longrightarrow f;\ g$ **by** (*rule AndChopA*)
 have *4*: $\vdash (f \wedge \ more\ );\ g \longrightarrow g$ **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 $\vdash \ f = (g^\star;\ h)$
shows $\vdash f = ((g;\ f) \vee \ h)$
proof −
have *1*: $\vdash f = (g^\star;\ h)$ **using** *assms* **by** *auto*
have *2*: $\vdash g^\star = \ (empty \ \vee \ (g;\ g^\star))$ **by** (*rule CSEqvOrChopCS*)
hence *3*: $\vdash g^\star;\ h = (h \vee \ ((g;\ g^\star);\ h))$ **by** (*rule EmptyOrChopEqvRule*)
have *4*: $\vdash (g;\ g^\star);\ h = \ g;\ (g^\star;\ h)$ **by** (*rule ChopAssocB*)
hence *41*: $\vdash g^\star;\ h = (h \vee g;\ (g^\star;\ h))$ **using** *3* **by** *fastforce*
have *5*: $\vdash g;\ f = g;\ (g^\star;\ h)$ **using** *1* **by** (*rule RightChopEqvChop*)
hence *6*: $\vdash (g^\star;\ h) = (h \vee \ g;\ f)$ **using** *41* **by** *fastforce*
hence *61*: $\vdash (g^\star;\ h) = ((g;\ f) \vee h)$ **by** *auto*
from *1 61* **show** *?thesis* **by** *fastforce*
qed

lemma *CSChopIntroRule*:
assumes $\vdash \ f \wedge \neg \ h \longrightarrow g;\ f$
$\vdash g \longrightarrow \ more$
shows $\vdash f \longrightarrow g^\star;\ h$
proof −
have *1*: $\vdash f \wedge \neg \ h \longrightarrow g;\ f$
using *assms* **by** *blast*
have *2*: $\vdash g \longrightarrow \ more$
using *assms* **by** *blast*
hence *3*: $\vdash g \longrightarrow g \wedge \ more$
by *auto*
hence *4*: $\vdash g;\ f \longrightarrow (g \wedge \ more\ );\ f$
by (*rule LeftChopImpChop*)
have *5*: $\vdash f \longrightarrow (g \wedge \ more\ );\ f \vee \ h$
using *1 4* **by** *fastforce*
have *6*: $\vdash g^\star = \ (empty \ \vee \ (g \wedge \ more\ );\ g^\star)$
by (*rule ChopstarEqv*)
hence *7*: $\vdash (g^\star);\ h = (h \vee \ ((g \wedge \ more\ );\ g^\star);\ h)$
by (*rule EmptyOrChopEqvRule*)
have *8*: $\vdash ((g \wedge \ more\ );\ g^\star);\ h = (g \wedge \ more\ );\ (g^\star;\ h)$
by (*rule ChopAssocB*)
have *9*: $\vdash (g^\star);\ h = (h \vee \ (g \wedge \ more\ );\ (g^\star;\ h))$
using *7 8* **by** *fastforce*
have *10*: $\vdash f \wedge \neg\ (g^\star;\ h) \longrightarrow (g \wedge more);\ f \wedge \neg\ ((g \wedge more);\ (g^\star;\ h))$
using *5 9* **by** *fastforce*
have *11*: $\vdash g \wedge \ more \ \longrightarrow \ more$
by *fastforce*
from *10 11* **show** *?thesis* **using** *ChopContra* **by** *blast*
qed

lemma *DiamondAndEmptyEqvAndEmpty*:
$\vdash (\Diamond\ f \wedge empty) = (f \wedge empty)$
by (*auto simp*: *sometimes-defs empty-defs*)

lemma *InitAndEmptyEqvAndEmpty*:
$\vdash ((init\ w) \wedge empty) = (w \wedge empty)$
proof –
have *1*: $\vdash ((init\ w) \wedge empty) = ((w \wedge empty);\#True \wedge empty)$
by (*metis init-d-def int-eq lift-and-com*)
have *2*: $\vdash ((w \wedge empty);\#True \wedge empty) = (w \wedge empty);(\#True \wedge empty)$
by (*meson AndChopA ChopAndA ChopAndEmptyEqvEmptyChopEmpty lift-imp-trans Prop11 Prop12*)
have *3*: $\vdash (w \wedge empty);(\#True \wedge empty) = (w \wedge empty);empty$
using *RightChopEqvChop* **by** *fastforce*
have *4*: $\vdash (w \wedge empty);empty = (w \wedge empty)$
using *ChopEmpty* **by** *blast*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *InitAndNotBoxInitImpNotEmpty*:
$\vdash init\ w \wedge \neg(\ \Box\ (init\ w)) \longrightarrow \neg\ \ empty$
proof –
have *1*: $\vdash ((init\ w) \wedge empty) = (w \wedge empty)$
by (*rule InitAndEmptyEqvAndEmpty*)
have *2*: $\vdash (\neg(\ \Box\ (init\ w)) \wedge empty) = (\Diamond\ \neg\ (init\ w) \wedge empty)$
by (*auto simp*: *always-d-def*)
have *3*: $\vdash (\Diamond\ \neg\ (init\ w) \wedge empty) = (\neg\ (init\ w) \wedge empty)$
using *DiamondAndEmptyEqvAndEmpty* **by** *blast*
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*: $\vdash$ *init w* $\wedge$ *f* $\wedge \neg$($\Box$ (*init w*)) $\longrightarrow$ ($\Box$ (*init w*) $\wedge$ *more*); *f*
  using *1 2* **by** *fastforce*
have *4*: $\vdash \Box$ (*init w*) $\wedge$ *more* $\longrightarrow$ ($\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *fin* (*init w*)
  by (*rule BoxInitAndMoreImpBoxInitAndMoreAndFinInit*)
hence *5*: $\vdash$ ($\Box$(*init w*) $\wedge$ *more*); *f* $\longrightarrow$ (($\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *fin* (*init w*)); *f*
  by (*rule LeftChopImpChop*)
have *6*: $\vdash$ (($\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *fin* (*init w*)); *f* =
   ($\Box$(*init w*) $\wedge$ *more*); (*init w* $\wedge$ *f*)
  by (*rule AndFinChopEqvStateAndChop*)
have *7*: $\vdash \neg$($\Box$(*init w*)) $\longrightarrow$ ($\Box$ (*init w*)) *yields* $\neg$($\Box$ (*init w*))
  by (*rule NotBoxStateImpBoxYieldsNotBox*)
have *8*: $\vdash$ ($\Box$(*init w*)) *yields* $\neg$ ($\Box$ (*init w*)) $\longrightarrow$
   ($\Box$ (*init w*) $\wedge$ *more*) *yields* $\neg$($\Box$(*init w*))
  by (*rule AndYieldsA*)
have *9*: $\vdash$ ($\Box$(*init w*) $\wedge$ *more*); (*init w* $\wedge$ *f*) $\wedge$ ($\Box$(*init w*) $\wedge$ *more*) *yields* $\neg$($\Box$ (*init w*))
   $\longrightarrow$
   ($\Box$ (*init w*) $\wedge$ *more*); ((*init w* $\wedge$ *f*) $\wedge \neg$ ($\Box$ (*init w*)))
  by (*rule ChopAndYieldsImp*)
have *10*: $\vdash$ (*init w* $\wedge$ *f*) $\wedge \neg$($\Box$ (*init w*)) $\longrightarrow$
   ($\Box$(*init w*) $\wedge$ *more*); ((*init w* $\wedge$ *f*) $\wedge \neg$($\Box$ (*init w*)))
  using *3 5 6 7 8 9* **by** *fastforce*
have *11*: $\vdash$ ($\Box$(*init w*) $\wedge$ *more*); ((*init w* $\wedge$ *f*) $\wedge \neg$($\Box$ (*init w*))) $\longrightarrow$
   more ; ((*init w* $\wedge$ *f*) $\wedge \neg$($\Box$ (*init w*)))
  by (*rule AndChopB*)
have *12*: $\vdash$ (*init w* $\wedge$ *f*) $\wedge \neg$($\Box$ (*init w*)) $\longrightarrow$
   more ; ((*init w* $\wedge$ *f*) $\wedge \neg$($\Box$ (*init w*)))
  using *10 11* **by** *fastforce*
from *12* **show** *?thesis* **using** *MoreChopContra* **by** *blast*
qed

lemma *BoxCSEqvBox*:
$\vdash$ (*init w* $\wedge$ ($\Box$(*init w*))*) = $\Box$ (*init w*)
proof −
have *1*: $\vdash$ ($\Box$ (*init w*))* = (*empty* $\vee$ ($\Box$ (*init w*) $\wedge$ *more*); ($\Box$ (*init w*))*)
  by (*rule ChopstarEqv*)
hence *2*: $\vdash$ ($\Box$ (*init w*))* $\longrightarrow$ *empty* $\vee$ ($\Box$ (*init w*) $\wedge$ *more*); ($\Box$ (*init w*))*
  by *fastforce*
hence *3*: $\vdash$ *init w* $\wedge$ ($\Box$ (*init w*))* $\longrightarrow \Box$ (*init w*)
  by (*rule CSImpBox*)
have *11*: $\vdash \Box$ (*init w*) $\longrightarrow$ (*init w*)
  using *BoxElim* **by** *blast*

have *12*: $\vdash \Box(\ init\ w) \longrightarrow (\Box\ (init\ w))^\star$
  by (*rule ImpCS*)
have *13*: $\vdash \Box\ (init\ w) \longrightarrow init\ w \wedge (\Box\ (init\ w))^\star$
  using *11 12* **by** *fastforce*
from *3 13* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateAndCSEqvCS*:
$\vdash\ \ (\Box(\ init\ w) \wedge f^\star) = (init\ w \wedge (\Box(\ init\ w) \wedge f)^\star)$
proof −
have *1*: $\vdash \Box\ (init\ w) \longrightarrow init\ w$
  using *BoxElim* **by** *blast*
have *2*: $\vdash (f^\star \wedge\ more) = (f \wedge\ more\ );\ f^\star$
  by (*rule CSAndMoreEqvAndMoreChop*)
have *3*: $\vdash (\Box(\ init\ w) \wedge ((f \wedge\ more\ );\ f^\star)) =$
    $((\Box\ (init\ w) \wedge f \wedge\ more\ );\ (\Box\ (init\ w) \wedge f^\star))$
  by (*rule BoxStateAndChopEqvChop*)
have *4*: $\vdash \Box\ (init\ w) \wedge f \wedge\ more\ \longrightarrow (\Box\ (init\ w) \wedge f) \wedge\ more$
  by *auto*
hence *5*: $\vdash (\Box\ (init\ w) \wedge f \wedge\ more\ );\ (\Box\ (init\ w) \wedge f^\star) \longrightarrow$
    $((\Box\ (init\ w) \wedge f) \wedge\ more\ );\ (\Box\ (init\ w) \wedge f^\star)$
  by (*rule LeftChopImpChop*)
have *6*: $\vdash (\Box(\ init\ w) \wedge f^\star) \wedge\ more\ \longrightarrow$
    $((\Box\ (init\ w) \wedge f) \wedge\ more\ );\ (\Box\ (init\ w) \wedge f^\star)$
  using *2 3 5* **by** *fastforce*
hence *7*: $\vdash \Box\ (init\ w) \wedge f^\star \longrightarrow (\Box\ (init\ w) \wedge f)^\star$
  by (*rule CSIntro*)
have *71*: $\vdash init\ w \wedge \Box\ (init\ w) \wedge f^\star \longrightarrow init\ w \wedge (\Box\ (init\ w) \wedge f)^\star$
  using *7* **by** *fastforce*
have *8*: $\vdash \Box(\ init\ w) \wedge f^\star \longrightarrow init\ w \wedge (\Box\ (init\ w) \wedge f)^\star$
  using *1 71* **by** *fastforce*
have *11*: $\vdash (\Box\ (init\ w) \wedge f)^\star \longrightarrow (\Box\ (init\ w))^\star$
  by (*rule AndCSA*)
have *12*: $\vdash (init\ w \wedge (\Box\ (init\ w))^\star) = \Box\ (init\ w)$
  by (*rule BoxCSEqvBox*)
have *13*: $\vdash (\Box\ (init\ w) \wedge f)^\star \longrightarrow f^\star$
  by (*rule AndCSB*)
have *14*: $\vdash init\ w \wedge (\Box\ (init\ w) \wedge f)^\star \longrightarrow init\ w \wedge (\Box\ (init\ w))^\star \wedge f^\star$
  using *11 13* **by** *fastforce*
have *15*: $\vdash init\ w \wedge (\Box\ (init\ w))^\star \wedge f^\star \longrightarrow \Box\ (init\ w) \wedge f^\star$
  using *12* **by** *auto*
have *16*: $\vdash init\ w \wedge (\Box\ (init\ w) \wedge f)^\star \longrightarrow \Box\ (init\ w) \wedge f^\star$
  using *14 15 lift-imp-trans* **by** *blast*
from *8 16* **show** *?thesis* **by** *fastforce*
qed

lemma *BaCSImpCS*:
$\vdash\ \ ba\ (f \longrightarrow g) \longrightarrow f^\star \longrightarrow g^\star$
proof −
have *1*: $\vdash f^\star =\ (empty\ \vee\ (f \wedge\ more\ );\ f^\star)$

by (*rule ChopstarEqv*)
have *2*: $\vdash g^\star = (empty \lor (g \land more); g^\star)$
by (*rule ChopstarEqv*)
have *21*: $\vdash \neg(g^\star) = (\neg empty \land \neg((g \land more); g^\star))$
using *2* **by** *fastforce*
hence *22*: $\vdash \neg(g^\star) = (more \land \neg((g \land more); g^\star))$
using *NotEmptyEqvMore* **by** *fastforce*
have *3*: $\vdash f^\star \land \neg (g^\star) \longrightarrow$
$(empty \lor (f \land more); f^\star) \land more \land \neg ((g \land more); g^\star)$
using *1 22* **by** *fastforce*
have *31*: $\vdash ((empty \lor (f \land more); f^\star) \land more) = ((f \land more); f^\star \land more)$
by (*auto simp*: *empty-d-def*)
have *32*: $\vdash f^\star \land \neg (g^\star) \longrightarrow (f \land more); f^\star \land \neg ((g \land more); g^\star)$
using *3 31* **by** *fastforce*
have *4*: $\vdash (f \longrightarrow g) \longrightarrow (f \land more \longrightarrow g \land more)$
by *auto*
hence *5*: $\vdash ba\ (f \longrightarrow g) \longrightarrow ba\ (f \land more \longrightarrow g \land more)$
by (*rule BaImpBa*)
have *6*: $\vdash ba\ (f \land more \longrightarrow g \land more) \longrightarrow$
$(f \land more); f^\star \longrightarrow (g \land more); f^\star$
by (*rule BaLeftChopImpChop*)
have *7*: $\vdash ba\ (f \longrightarrow g) \land (f \land more); f^\star \longrightarrow (g \land more); f^\star$
using *5 6* **by** *fastforce*
have *8*: $\vdash (g \land more); f^\star \land \neg ((g \land more); g^\star)$
$\longrightarrow (g \land more); (f^\star \land \neg (g^\star))$
by (*rule ChopAndNotChopImp*)
have *9*: $\vdash (g \land more); (f^\star \land \neg (g^\star)) \longrightarrow more ; (f^\star \land \neg (g^\star))$
by (*rule AndChopB*)
have *10*: $\vdash ba\ (f \longrightarrow g) \longrightarrow more ; (f^\star \land \neg (g^\star)) \longrightarrow$
$more ; (ba\ (f \longrightarrow g) \land f^\star \land \neg (g^\star))$
by (*rule BaChopImpChopBa*)
have *11*: $\vdash ba\ (f \longrightarrow g) \land f^\star \land \neg (g^\star) \longrightarrow$
$more ; (ba\ (f \longrightarrow g) \land f^\star \land \neg (g^\star))$
using *32 7 8 9 10* **by** *fastforce*
hence *12*: $\vdash \neg ((ba\ (f \longrightarrow g)) \land (f^\star) \land (\neg (g^\star)))$
using *MoreChopLoop* **by** *blast*
from *12* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *BaCSEqvCS*:
$\vdash ba\ (f = g) \longrightarrow (f^\star = g^\star)$
proof −
have *1*: $\vdash ba\ (f = g) = (ba\ (f \longrightarrow g) \land ba\ (g \longrightarrow f))$ **by** (*auto simp*: *ba-defs*)
have *2*: $\vdash ba\ (f \longrightarrow g) \longrightarrow (f^\star \longrightarrow g^\star)$ **by** (*rule BaCSImpCS*)
have *3*: $\vdash ba\ (g \longrightarrow f) \longrightarrow (g^\star \longrightarrow f^\star)$ **by** (*rule BaCSImpCS*)
have *4*: $\vdash ba\ (f = g) \longrightarrow (f^\star \longrightarrow g^\star) \land (g^\star \longrightarrow f^\star)$ **using** *1 2 3* **by** *fastforce*
have *5*: $\vdash ((f^\star \longrightarrow g^\star) \land (g^\star \longrightarrow f^\star)) = (f^\star = g^\star)$ **by** *auto*
from *4 5* **show** *?thesis* **by** *auto*
qed

lemma *BaAndCSImport*:
$\vdash$ *ba* $f \wedge g^\star \longrightarrow (f \wedge g)^\star$
proof $-$
have *1*: $\vdash f \longrightarrow (g \longrightarrow f \wedge g)$ **by** *auto*
hence *2*: $\vdash$ *ba* $f \longrightarrow$ *ba* $(g \longrightarrow f \wedge g)$ **by** (*rule BaImpBa*)
have *3*: $\vdash$ *ba* $(g \longrightarrow f \wedge g) \longrightarrow g^\star \longrightarrow (f \wedge g)^\star$ **by** (*rule BaCSImpCS*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *CSSkip*:
$\vdash$ *skip**
by (*metis ChopPlusImpCS EmptyImpCS EmptyNextInducta next-d-def*)

5.8 Properties of While

lemma *WhileEqvIf*:
$\vdash$ *while* (*init w*) *do* $f =$ *if*$_i$ (*init w*) *then* (f; (*while* (*init w*) *do* f)) *else* *empty*
proof $-$
have *1*: $\vdash$ *while* (*init w*) *do* $f = (((\textit{init } w) \wedge f)^\star \wedge$ *fin* $\neg$ (*init w*))
by (*simp add*: *while-d-def*)
have *2*: $\vdash (\textit{init } w \wedge f)^\star =$ (*empty* $\vee$ ((*init w* $\wedge f$); (*init w* $\wedge f)^\star$))
by (*rule CSEqvOrChopCS*)
have *21*: $\vdash (((\textit{init } w) \wedge f)^\star \wedge$ *fin* $\neg$ (*init w*)) $=$
((*empty* $\vee$ ((*init w* $\wedge f$); (*init w* $\wedge f)^\star$)) $\wedge$ *fin* $\neg$ (*init w*))
using *2* **by** *fastforce*
have *22*: $\vdash$ ((*empty* $\vee$ ((*init w* $\wedge f$); (*init w* $\wedge f)^\star$)) $\wedge$ *fin* $\neg$ (*init w*)) $=$
((*empty* $\wedge$ *fin* $\neg$(*init w*)) $\vee$ (((*init w* $\wedge f$); (*init w* $\wedge f)^\star$) $\wedge$ *fin* $\neg$ (*init w*)))
by *auto*
have *3*: $\vdash$ (*empty* $\wedge$ *fin* $\neg$ (*init w*)) $=$ ($\neg$ (*init w*) $\wedge$ *empty*)
using *AndFinEqvChopAndEmpty EmptyChop* **by** (*metis int-eq*)
have *4*: $\vdash$ (*init w* $\wedge f$); (*init w* $\wedge f)^\star =$ (*init w* $\wedge$ (f; (*init w* $\wedge f)^\star$))
by (*rule StateAndChop*)
have *41*: $\vdash$ (((*init w* $\wedge f$); (*init w* $\wedge f)^\star$) $\wedge$ *fin* $\neg$ (*init w*)) $=$
(*init w* $\wedge$ (f; (*init w* $\wedge f)^\star$) $\wedge$ *fin* $\neg$ (*init w*))
using *4* **by** *auto*
have *42*: $\vdash$ (*init w* $\wedge$ (f; (*init w* $\wedge f)^\star$) $\wedge$ *fin* $\neg$ (*init w*)) $=$
(*init w* $\wedge$ (f; (*init w* $\wedge f)^\star$) $\wedge$ *fin* (*init* $\neg$ *w*))
using *Initprop*(*2*) **by** (*metis StateAndEmptyChop int-eq*)
have *5*: $\vdash$ ((f; ((*init w* $\wedge f)^\star$)) $\wedge$ (*fin* (*init* $\neg$ *w*)))
$=$ (f; ((*init w* $\wedge f)^\star \wedge$ (*fin* (*init* $\neg$ *w*))))
by (*rule ChopAndFin*)
have *51*: $\vdash$ (f; ((*init w* $\wedge f)^\star \wedge$ (*fin* (*init* $\neg$ *w*)))) $=$
(f; ((*init w* $\wedge f)^\star \wedge$ (*fin* $\neg$ (*init w*))))
using *Initprop*(*2*) **by** (*smt RightChopEqvChop int-eq lift-and-com*)
have *52*: $\vdash$ (*init w* $\wedge$ (f; (*init w* $\wedge f)^\star$) $\wedge$ *fin* $\neg$ (*init w*)) $=$
(*init w* $\wedge$ (f; ((*init w* $\wedge f)^\star \wedge$ *fin* $\neg$ (*init w*))))
using *42 5 51* **by** *fastforce*
have *6*: $\vdash$ (f; ((*init w* $\wedge f)^\star \wedge$ *fin* $\neg$ (*init w*))) $= f$; *while* (*init w*) *do* f
by (*simp add*: *while-d-def*)
have *61*: $\vdash$ (*init w* $\wedge$ (f; ((*init w* $\wedge f)^\star \wedge$ *fin* $\neg$ (*init w*)))) $=$

```
      (init w ∧ (f; while (init w) do f)) using 6
    by auto
 have 62: ⊢ ( empty ∧ fin ¬ (init w)) ∨ ( ((init w ∧ f); (init w ∧ f)⋆) ∧ fin ¬ (init w))
        = (¬ ( init w) ∧ empty ) ∨ (init w ∧ (f; while (init w) do f))
    using 21 22 3 4 52 61 by fastforce
 have 7: ⊢ while (init w) do f
       = ((¬ ( init w) ∧ empty ) ∨ (init w ∧ (f; while (init w) do f)))
    using 1 21 22 62
    by (metis 3 41 42 5 51 inteq-reflection)
 have 71: ⊢ ifi (init w) then (f; ( while (init w) do f)) else empty =
         ((¬ ( init w) ∧ empty ) ∨ (init w ∧ (f; while (init w) do f)))
    by (auto simp: ifthenelse-d-def)
 from 7 71 show ?thesis by fastforce
qed

lemma WhileChopEqvIf:
⊢ ( while ( init w) do f); g = ifi(init w) then (f; ((while ( init w) do f); g)) else g
proof −
 have 1: ⊢ while (init w) do f =
         ifi (init w) then (f; ( while (init w) do f)) else empty
    by (rule WhileEqvIf)
 hence 2: ⊢ ( while (init w) do f); g =
         ifi (init w) then ((f; while (init w) do f); g) else (empty ; g)
    by (rule IfChopEqvRule)
 have 3: ⊢ empty ; g = g
    by (rule EmptyChop)
 have 4: ⊢ ifi (init w) then ((f; while (init w) do f); g) else (empty ; g) =
        ifi (init w) then ((f; while (init w) do f); g) else g
    using 3 using inteq-reflection by fastforce
 have 5: ⊢ ((f; while (init w) do f); g) = (f; (while (init w) do f; g))
    by (rule ChopAssocB)
 have 6: ⊢ ifi (init w) then ((f; while (init w) do f); g) else g =
        ifi (init w) then (f; ((while (init w) do f); g)) else g
    using 5 using inteq-reflection by fastforce
 from 1 2 4 6 show ?thesis by fastforce
qed

lemma WhileChopEqvIfRule:
 assumes ⊢ f = ( while (init w) do g); h
 shows ⊢ f = ifi (init w) then (g; f) else h
proof −
 have 1: ⊢ f = ( while (init w) do g); h
   using assms by auto
 have 2: ⊢ ( while (init w) do g); h =
        ifi (init w) then (g; (( while (init w) do g); h)) else h
   by (rule WhileChopEqvIf)
 have 3: ⊢ (g; f) = (g; (( while (init w) do g); h))
   using 1 by (rule RightChopEqvChop)
 have 4: ⊢ (g; (( while (init w) do g); h)) = (g; f)
   using 3 by auto
```

have *5*: $\vdash$ *if* $_i$ (*init w*) *then* (*g*; ((*while* (*init w*) *do* *g*); *h*)) *else* *h* =
if $_i$ (*init w*) *then* (*g*; *f*) *else* *h*
using *4* **using** *inteq-reflection* **by** *fastforce*
from *1 2 5* **show** *?thesis* **by** *fastforce*
qed

lemma *WhileImpFin*:
$\vdash$ *while* (*init w*) *do* *f* $\longrightarrow$ *fin* $\neg$ (*init w*)
proof –
have *1*: $\vdash$ (*init w* $\wedge$ *f*)* $\wedge$ *fin* $\neg$ (*init w*) $\longrightarrow$ *fin* $\neg$ (*init w*) **by** *auto*
from *1* **show** *?thesis* **by** (*simp add*: *while-d-def*)
qed

lemma *WhileEqvEmptyOrChopWhile*:
$\vdash$ *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*

⊢ *init w* ∧ *f* ⟶ (*g* ∧ *more*); *f*
shows ⊢ *f* ⟶ *while* (*init w*) *do* *g*
proof −
have *1*: ⊢ ¬ (*init w*) ∧ *f* ⟶ *empty*
using *assms* **by** *blast*
have *2*: ⊢ *init w* ∧ *f* ⟶ (*g* ∧ *more*); *f*
using *assms* **by** *blast*
have *3*: ⊢ *while* (*init w*) *do* *g* =
((¬ (*init w*) ∧ *empty*) ∨ (*init w* ∧ (*g* ∧ *more*); *while* (*init w*) *do* *g*))
by (*rule WhileEqvEmptyOrChopWhile*)
hence *31*: ⊢ ¬ (*while* (*init w*) *do* *g*) =
(¬((¬ (*init w*) ∧ *empty*) ∨ (*init w* ∧ (*g* ∧ *more*); *while* (*init w*) *do* *g*)))
by *fastforce*
hence *32*: ⊢ (*f* ∧ ¬ (*while* (*init w*) *do* *g*)) =
(*f* ∧ ¬((¬(*init w*) ∧ *empty*) ∨ (*init w* ∧ (*g* ∧ *more*); *while* (*init w*) *do* *g*)))
by *fastforce*
have *33*: ⊢ (*f* ∧ ¬((¬(*init w*) ∧ *empty*) ∨ (*init w* ∧ (*g* ∧ *more*); *while* (*init w*) *do* *g*))) =
(*f* ∧ ¬(¬(*init w*) ∧ *empty*) ∧ ¬(*init w* ∧ (*g* ∧ *more*); *while* (*init w*) *do* *g*))
by *auto*
have *34*: ⊢ (*f* ∧ ¬(¬(*init w*) ∧ *empty*) ∧ ¬((*init w*) ∧ ((*g* ∧ *more*); *while* (*init w*) *do* *g*))) =
(*f* ∧ ((*init w*) ∨ *more*) ∧ (¬(*init w*) ∨ ¬((*g* ∧ *more*); *while* (*init w*) *do* *g*)))
by (*auto simp*: *empty-d-def*)
have *35*: ⊢ (*f* ∧ ((*init w*) ∨ *more*) ∧ (¬(*init w*) ∨ ¬((*g* ∧ *more*);*while* (*init w*) *do g*))) =
((*f* ∧ (*init w*) ∧ ¬ ((*g* ∧ *more*); *while* (*init w*) *do* *g*)) ∨
(*f* ∧ (*init w*) ∧ ¬(*init w*)) ∨
(*f* ∧ *more* ∧ ¬ ((*g* ∧ *more*); *while* (*init w*) *do* *g*)) ∨
(*f* ∧ *more* ∧ ¬(*init w*)))
by *auto*
have *36*: ⊢ (*f* ∧ ¬ (*while* (*init w*) *do* *g*)) =
((*f* ∧ (*init w*) ∧ ¬ ((*g* ∧ *more*); *while* (*init w*) *do* *g*)) ∨
(*f* ∧ (*init w*) ∧ ¬(*init w*)) ∨
(*f* ∧ *more* ∧ ¬ ((*g* ∧ *more*); *while* (*init w*) *do* *g*)) ∨
(*f* ∧ *more* ∧ ¬(*init w*))) **using** *32 33 34 35* **by** *fastforce*
have *37*: ⊢ ¬(*f* ∧ *more* ∧ ¬(*init w*))
using *1* **by** (*auto simp*: *empty-d-def*)
have *38*: ⊢ (*f* ∧ *more* ∧ ¬ ((*g* ∧ *more*); *while* (*init w*) *do* *g*)) ⟶
((*g* ∧ *more*); *f* ∧ ¬ ((*g* ∧ *more*); *while* (*init w*) *do* *g*))
using *1 2* **by** (*auto simp*: *empty-d-def Valid-def*)
have *39*: ⊢ (*f* ∧ (*init w*) ∧ ¬ ((*g* ∧ *more*); *while* (*init w*) *do* *g*)) ⟶
((*g* ∧ *more*); *f* ∧ ¬ ((*g* ∧ *more*); *while* (*init w*) *do* *g*))
using *2* **by** *auto*
have *40*: ⊢ ((*f* ∧ (*init w*) ∧ ¬ ((*g* ∧ *more*); *while* (*init w*) *do* *g*)) ∨
(*f* ∧ (*init w*) ∧ ¬(*init w*)) ∨
(*f* ∧ *more* ∧ ¬ ((*g* ∧ *more*); *while* (*init w*) *do* *g*)) ∨
(*f* ∧ *more* ∧ ¬(*init w*))) ⟶
(*g* ∧ *more*); *f* ∧ ¬ ((*g* ∧ *more*); *while* (*init w*) *do* *g*)
using *39 38 37 38* **by** *fastforce*
have *4*: ⊢ *f* ∧ ¬ (*while* (*init w*) *do* *g*) ⟶
(*g* ∧ *more*); *f* ∧ ¬ ((*g* ∧ *more*); *while* (*init w*) *do* *g*)
using *36 40* **by** *fastforce*

have *5*: $\vdash g \wedge$ *more* $\longrightarrow$ *more*
by *auto*
from *4 5* **show** *?thesis* **using** *ChopContra* **by** *blast*
qed

lemma *WhileElim*:
assumes $\vdash \neg$ (*init w*) $\wedge$ *empty* $\longrightarrow g$
$\vdash$ *init w* $\wedge$ (*f* $\wedge$ *more*); *g* $\longrightarrow g$
shows $\vdash$ *while* (*init w*) *do f* $\longrightarrow g$
proof −
have *1*: $\vdash$ *while* (*init w*) *do f* =
((¬ (*init w*) ∧ *empty*) ∨ (*init w* ∧ (*f* ∧ *more*); *while* (*init w*) *do f*))
by (*rule WhileEqvEmptyOrChopWhile*)
hence *11*: ⊢ ((*while* (*init w*) *do f*) ∧ ¬ *g*) =
(((¬(*init w*) ∧ *empty*) ∨ (*init w* ∧ (*f* ∧ *more*);*while* (*init w*) *do f*)) ∧ ¬ *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*);*while* (*init w*) *do f*)) ∧ ¬ *g* ⟶
(*init w* ∧ (*f* ∧ *more*); *while* (*init w*) *do f*)
using *21* **by** *auto*
have *23*: ⊢ (*while* (*init w*) *do f*) ∧ ¬ *g* ⟶
(*init w* ∧ (*f* ∧ *more*); *while* (*init w*) *do f*) ∧ ¬ *g*
using *11 21* **by** *fastforce*
have *3*: ⊢ (*init w*) ∧ ((*f* ∧ *more*); *g*) ⟶ *g*
using *assms* **by** *blast*
hence *31*: ⊢ ¬ *g* ⟶ ¬((*init w*) ∧ ((*f* ∧ *more*); *g*))
by *fastforce*
have *32*: ⊢ (*init w* ∧ (*f* ∧ *more*); *while* (*init w*) *do f*) ∧ ¬ *g* ⟶
(((*f* ∧ *more*); (*while* (*init w*) *do f*)) ∧ ¬ ((*f* ∧ *more*); *g*)) ∧ ¬*g*
using *31* **by** *auto*
have *4*: ⊢ (*while* (*init w*) *do f*) ∧ ¬ *g* ⟶
((*f* ∧ *more*); (*while* (*init w*) *do f*)) ∧ ¬ ((*f* ∧ *more*); *g*)
using *23 32* **by** *fastforce*
have *5*: ⊢ *f* ∧ *more* ⟶ *more*
by *auto*
from *4 5* **show** *?thesis* **using** *ChopContra* **by** *blast*
qed

lemma *BaWhileImpWhile*:
⊢ *ba* (*f* ⟶ *g*) ⟶ (*while* (*init w*) *do f*) ⟶ (*while* (*init w*) *do g*)
proof −
have *1*: ⊢ (*f* ⟶ *g*) ⟶ ((*init w* ∧ *f*) ⟶ (*init w* ∧ *g*))
by *auto*
hence *2*: ⊢ *ba* (*f* ⟶ *g*) ⟶ *ba* ((*init w* ∧ *f*) ⟶ (*init w*∧ *g*))
by (*rule BaImpBa*)
have *3*: ⊢ *ba* ((*init w* ∧ *f*) ⟶ (*init w*∧ *g*)) ⟶ ((*init w*∧ *f*)* ⟶ (*init w*∧ *g*)*)
by (*rule BaCSImpCS*)

have *4*: ⊢ *ba* ($f \longrightarrow g$) $\longrightarrow$ ((*init w*∧ f)*∧ *fin* ¬ (*init w*) $\longrightarrow$ (*init w*∧ g)*∧ *fin* ¬ (*init w*))
    using *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *while-d-def*)
qed

lemma *WhileImpWhile*:
assumes ⊢ $f \longrightarrow g$
shows ⊢ (*while* (*init w*) *do* f) $\longrightarrow$ (*while* (*init w*) *do* g)
proof −
have *1*: ⊢ $f \longrightarrow g$
    using *assms* **by** *auto*
hence *2*: ⊢ *ba* ($f \longrightarrow g$)
    by (*rule BaGen*)
have *3*: ⊢ *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

5.9 Properties of Halt

lemma *WnextAndMoreEqvNext*:
⊢ (*wnext f* ∧ *more*) = ○ *f*
by (*auto simp*: *wnext-defs more-defs next-defs*)

lemma *BoxStateAndEmptyEqvStateAndEmpty*:
⊢ (□(*empty* = (*init w*)) ∧ *empty*) = ((*init w*) ∧ *empty*)
by (*auto simp*: *always-defs init-defs empty-defs*)

lemma *BoxEmptyEqvIStateqvEmptyAndStateOrNotStateNext*:
⊢ □(*empty* = (*init w*)) = ((*empty* ∧ *init w*) ∨ (¬(*init w*) ∧ ○(□(*empty* = (*init w*)))))
proof −
have *1*: ⊢ □(*empty* = (*init w*)) =
        ((□(*empty* = (*init w*)) ∧ *empty*) ∨ (□(*empty* = (*init w*)) ∧ *more*))
    by (*auto simp*: *empty-d-def*)
have *2*: ⊢ (□(*empty* = (*init w*)) ∧ *empty*) = ((*init w*) ∧ *empty*)
    using *BoxStateAndEmptyEqvStateAndEmpty* **by** *blast*
have *3*: ⊢ □(*empty* = (*init w*)) = ((*empty* = (*init w*)) ∧ *wnext*(□(*empty* = (*init w*))))
    using *BoxEqvAndWnextBox* **by** *blast*
hence *4*: ⊢ (□(*empty* = (*init w*)) ∧ *more*) =
        (((*empty* = (*init w*)) ∧ *more*) ∧ (*wnext*(□(*empty* = (*init w*))) ∧ *more*))
    by *auto*
have *5*: ⊢ ((*empty* = (*init w*)) ∧ *more*) = (¬(*init w*) ∧ *more*)
    by (*auto simp*: *empty-d-def*)
have *6*: ⊢ (*wnext*(□(*empty* = (*init w*))) ∧ *more*) = ○(□(*empty* = (*init w*)))
    using *WnextAndMoreEqvNext* **by** *metis*
have *7*: ⊢ (□(*empty* = (*init w*)) ∧ *more*) =
        ((¬ (*init w*) ∧ *more*) ∧ (*wnext*(□(*empty* = (*init w*))) ∧ *more*))
    using *4 5* **by** *fastforce*
have *8*: ⊢ ((¬ (*init w*) ∧ *more*) ∧ (*wnext*(□(*empty* = (*init w*))) ∧ *more*)) =
        ((¬ (*init w*)) ∧ (*wnext*(□(*empty* = (*init w*))) ∧ *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*:
$\vdash\ halt(\ init\ w) =\ if_i\ (init\ w)\ \ then\ \ empty\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w)))$
proof −
have *1*: $\vdash halt(\ init\ w) = \Box(empty = (init\ w))$
by (*simp add*: *halt-d-def*)
have *2*: $\vdash \Box(empty = (init\ w)) =$
$((empty \wedge init\ w) \vee (\neg(init\ w) \wedge \bigcirc(\Box(empty = (init\ w)))))$
by (*rule BoxEmptyEqvIStateqvEmptyAndStateOrNotStateNext*)
have *21*: $\vdash ((empty \wedge init\ w) \vee (\neg(init\ w) \wedge \bigcirc(\Box(empty = (init\ w))))) =$
$((init\ w \wedge empty) \vee (\neg(init\ w) \wedge \bigcirc(\Box(empty = (init\ w)))))$
by *auto*
have *22*: $\vdash \bigcirc(halt\ (\ init\ w)) = \bigcirc(\Box(empty = (init\ w)))$
using *NextEqvNext* **using** *1* **by** *blast*
have *3*: $\vdash if_i\ (init\ w)\ \ then\ \ empty\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w))) =$
$((init\ w \wedge empty\ ) \vee (\neg(init\ w) \wedge \bigcirc(\ halt\ (\ init\ w))))$
by (*simp add*: *ifthenelse-d-def*)
from *1 2 21 22 3* **show** *?thesis* **by** *fastforce*
qed

lemma *HaltChopEqv*:
$\vdash\ ((halt\ (\ init\ w))\ ;\ f) =\ (if_i\ (init\ w)\ \ then\ (\ f\ )\ else\ (\bigcirc(\ (halt\ (\ init\ w));\ f)))$
proof −
have *1*: $\vdash\ halt(init\ w) =$
$(if_i\ (init\ w)\ \ then\ \ empty\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w))))$
by (*rule HaltStateEqvIfStateThenEmptyElseNext*)
hence *2*: $\vdash\ ((halt(init\ w));f) =$
$(if_i\ (init\ w)\ \ then\ \ (empty;f)\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w));f))$
by (*rule IfChopEqvRule*)
have *3*: $\vdash\ empty\ ;\ f = f$
by (*rule EmptyChop*)
have *4*: $\vdash (\bigcirc\ (halt\ (\ init\ w)));\ f = \bigcirc(\ halt\ (\ init\ w);\ f)$
by (*rule NextChop*)
from *2 3 4* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *AndHaltChopImp*:
$\vdash\ init\ w \wedge (\ halt\ (\ init\ w);\ f) \longrightarrow f$
proof −
have *1*: $\vdash halt\ (\ init\ w);\ f =\ if_i\ (init\ w)\ \ then\ \ f\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w);\ f))$
by (*rule HaltChopEqv*)
have *2*: $\vdash init\ w \wedge if_i\ (init\ w)\ \ then\ \ f\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w);\ f)) \longrightarrow f$
by (*auto simp*: *ifthenelse-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*

qed

lemma *NotAndHaltChopImpNext*:
$\vdash\ \neg\ (\ init\ w)\ \wedge\ (\ halt\ \ (init\ w);\ f)\ \longrightarrow \bigcirc(\ halt\ (\ init\ w);\ f)$
proof −
have *1*: $\vdash\ \ halt\ (\ init\ w);\ f\ =\ \ if_i\ \ (init\ w)\ \ then\ \ f\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w);\ f))$
by (*rule HaltChopEqv*)
have *2*: $\vdash \neg\ (\ init\ w)\ \wedge\ if_i\ \ (init\ w)\ \ then\ \ f\ \ else\ (\ \bigcirc(\ halt\ (\ init\ w);\ f))\ \longrightarrow$
$\bigcirc(\ halt\ (\ init\ w);\ f)$
by (*auto simp*: *ifthenelse-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NotAndHaltChopImpSkipYields*:
$\vdash\ \neg\ (\ init\ w)\ \wedge\ (\ halt\ (\ init\ w);\ f)\ \longrightarrow\ \ skip\ \ yields\ (\ halt\ \ (init\ w);\ f)$
proof −
have *1*: $\vdash \neg\ (\ init\ w)\ \wedge\ (\ halt\ (\ init\ w);\ f)\ \longrightarrow \bigcirc(\ halt\ (\ init\ w);\ f)$
by (*rule NotAndHaltChopImpNext*)
have *2*: $\vdash \bigcirc(\ halt\ (\ init\ w);\ f)\ \longrightarrow\ \ skip\ \ yields\ (\ halt\ (\ init\ w);\ f)$
by (*rule NextImpSkipYields*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *TrueChopAndEmptyEqvChopAndEmpty*:
$\vdash ((\#True;(f\ \wedge\ empty))\ \wedge\ g) = (g;(f\wedge\ empty))$
by (*metis AndFinEqvChopAndEmpty int-eq intensional-simps*(*17*) *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*: $\vdash (\neg more\ \vee\ \neg\neg(f;skip)) = \neg(more\ \wedge\ \neg(f;skip))$ **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*:
$\vdash\ \ \ halt\ (\ init\ w);\ f\ \longrightarrow \neg\ (\ halt\ (\ init\ w);\ \neg\ \ f)$
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: $\vdash$ *halt* (*init w*); f $\wedge$ *halt* (*init w*); $\neg f$ $\longrightarrow$
$\bigcirc$ (*halt* (*init w*); f $\wedge$ *halt* (*init w*); $\neg f$)
using *7 8* **by** *fastforce*
hence 10: $\vdash$ $\neg$(*halt* (*init w*); f $\wedge$ *halt* (*init w*); $\neg f$)
using *NextLoop* **by** *blast*
from *10* **show** *?thesis* **by** *auto*
qed

lemma *HaltChopImpHaltYields*:
$\vdash$ *halt* (*init w*); f $\longrightarrow$ (*halt* (*init w*)) *yields* f
proof −
have 1: $\vdash$ *halt* (*init w*); f $\longrightarrow$ $\neg$ (*halt* (*init w*); $\neg$ f) **by** (*rule HaltChopImpNotHaltChopNot*)
from *1* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *HaltChopAnd*:
$\vdash$ (*halt* (*init w*)); f $\wedge$ (*halt* (*init w*)); g $\longrightarrow$ (*halt* (*init w*)); (f $\wedge$ g)
proof −
have 1: $\vdash$ (*halt* (*init w*)); g $\longrightarrow$ (*halt* (*init w*)) *yields* g **by** (*rule HaltChopImpHaltYields*)
hence 2: $\vdash$ (*halt* (*init w*)); f $\wedge$ (*halt* (*init w*)); g $\longrightarrow$
(*halt* (*init w*)); f $\wedge$ (*halt* (*init w*)) *yields* g **by** *auto*
have 3: $\vdash$ (*halt* (*init w*)); f $\wedge$ (*halt* (*init w*)) *yields* g $\longrightarrow$
(*halt* (*init w*)); (f $\wedge$ g) **by** (*rule ChopAndYieldsImp*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *HaltAndChopAndHaltChopImpHaltAndChopAnd*:
$\vdash$ (*halt* (*init w*) $\wedge$ *f*); *f1* $\wedge$ (*halt* (*init w*); *g*) $\longrightarrow$ (*halt* (*init w*) $\wedge$ *f*); (*f1* $\wedge$ *g*)
proof –
 have *1*: $\vdash$ *f1* $\longrightarrow$ $\neg$ *g* $\vee$ (*f1* $\wedge$ *g*)
  by *auto*
 hence *2*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*); *f1* $\longrightarrow$
   (*halt* (*init w*) $\wedge$ *f*); $\neg$ *g* $\vee$ ((*halt* (*init w*) $\wedge$ *f*); (*f1* $\wedge$ *g*))
  by (*rule ChopOrImpRule*)
 have *3*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*); $\neg$ *g* $\longrightarrow$ *halt* (*init w*); $\neg$ *g*
  by (*rule AndChopA*)
 have *31*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*); *f1* $\longrightarrow$
   halt (*init w*); $\neg$ *g* $\vee$ ((*halt* (*init w*) $\wedge$ *f*); (*f1* $\wedge$ *g*))
  using *23* **by** *fastforce*
 have *4*: $\vdash$ *halt* (*init w*); *g* $\longrightarrow$ $\neg$ (*halt* (*init w*); $\neg$ *g*)
  by (*rule HaltChopImpNotHaltChopNot*)
 hence *41*: $\vdash$ (*halt* (*init w*); $\neg$ *g*) $\longrightarrow$ $\neg$(*halt* (*init w*); *g*)
  by *auto*
 have *42*: $\vdash$ (*halt* (*init w*) $\wedge$ *f*); *f1* $\longrightarrow$
   $\neg$(*halt* (*init w*); *g*) $\vee$ ((*halt* (*init w*) $\wedge$ *f*); (*f1* $\wedge$ *g*))
  using *31 41* **by** *fastforce*
 from *42* **show** *?thesis* **by** *auto*
qed

lemma *HaltImpBoxYields*:
$\vdash$ (*halt* (*init w*)); *f* $\longrightarrow$ ($\Box\neg$ (*init w*)) *yields* ((*halt* (*init w*)); *f*)
proof –
 have *1*: $\vdash$ ($\Box$ $\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*) $\longrightarrow$ *di* ($\Box$ $\neg$ (*init w*))
  by (*rule ChopImpDi*)
 have *2*: $\vdash$ $\Box$ $\neg$ (*init w*) $\longrightarrow$ $\neg$ (*init w*)
  by (*rule BoxElim*)
 hence *3*: $\vdash$ *di* ($\Box$ $\neg$ (*init w*)) $\longrightarrow$ *di* $\neg$ (*init w*)
  by (*rule DiImpDi*)
 have *4*: $\vdash$ *di* (*init* $\neg$ *w*) = (*init* $\neg$*w*)
  by (*rule DiState*)
 have *41*: $\vdash$ (*init* $\neg$ *w*) = $\neg$ (*init w*)
  using *Initprop*(*2*) **by** *fastforce*
 have *42*: $\vdash$ *di* $\neg$ (*init w*) = $\neg$(*init w*)
  using *4 41* **by** (*metis inteq-reflection*)
 have *5*: $\vdash$ (($\Box\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*)) $\longrightarrow$ $\neg$ (*init w*)
  using *1 2 42* **using** *3* **by** *fastforce*
 hence *51*: $\vdash$ (*halt* (*init w*); *f*) $\wedge$ (($\Box\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*)) $\longrightarrow$
   (*halt* (*init w*); *f*) $\wedge$ $\neg$ (*init w*)
  by *fastforce*
 have *6*: $\vdash$ *halt* (*init w*); *f* = *if*$_i$ (*init w*) *then* *f* *else* ($\bigcirc$(*halt* (*init w*); *f*))
  by (*rule HaltChopEqv*)
 hence *61*: $\vdash$ (*halt* (*init w*); *f* $\wedge$ $\neg$ (*init w*)) =
   ((*if*$_i$ (*init w*) *then* *f* *else* ($\bigcirc$(*halt* (*init w*); *f*))) $\wedge$ $\neg$ (*init w*))
  using *6* **by** *auto*
 have *62*: $\vdash$ (*if*$_i$ (*init w*) *then* *f* *else* ($\bigcirc$(*halt* (*init w*); *f*))) $\wedge$

$\neg$ (*init w*) $\longrightarrow$ ($\bigcirc$(*halt* (*init w*); *f*))
by (*auto simp*: *ifthenelse-d-def*)
have *63*: $\vdash$ *halt* (*init w*); *f* $\wedge$ $\neg$ (*init w*) $\longrightarrow$ ($\bigcirc$(*halt* (*init w*); *f*))
using *61 62* **by** *fastforce*
have *7*: $\vdash$ (*halt* (*init w*); *f*) $\wedge$ ($\Box\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*) $\longrightarrow$
$\bigcirc$((*halt* (*init w*)); *f*)
using *51 63* **using** *lift-imp-trans* **by** *blast*
have *8*: $\vdash$ $\Box$ $\neg$ (*init w*) $\longrightarrow$ *empty* $\vee$ $\bigcirc$($\Box\neg$(*init w*))
using *BoxBoxImpBox BoxEqvAndEmptyOrNextBox* **by** *fastforce*
hence *9*: $\vdash$ (($\Box$ $\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*)) $\longrightarrow$
$\neg$ (*halt* (*init w*); *f*) $\vee$ $\bigcirc$(($\Box\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*))
by (*rule EmptyOrNextChopImpRule*)
hence *10*: $\vdash$ ((*halt* (*init w*)); *f*) $\wedge$ ($\Box$ $\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*) $\longrightarrow$
$\bigcirc$(($\Box\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*))
by *fastforce*
have *11*: $\vdash$ (*halt* (*init w*)); *f* $\wedge$ ($\Box$ $\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*) $\longrightarrow$
$\bigcirc$((*halt* (*init w*)); *f*) $\wedge$ $\bigcirc$(($\Box\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*))
using *7 10* **by** *fastforce*
have *12*: $\vdash$ $\bigcirc$((*halt* (*init w*)); *f*) $\wedge$ $\bigcirc$(($\Box\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*))
$\longrightarrow$ $\bigcirc$(((*halt* (*init w*)); *f*) $\wedge$ (($\Box\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*)))
using *NextAndEqvNextAndNext* **by** *fastforce*
have *13*: $\vdash$ (*halt* (*init w*)); *f* $\wedge$ ($\Box$ $\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*) $\longrightarrow$
$\bigcirc$(((*halt* (*init w*)); *f*) $\wedge$ (($\Box\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*)))
using *11 12* **by** *fastforce*
hence *14*: $\vdash$ $\neg$ ((*halt* (*init w*)); *f* $\wedge$ ($\Box$ $\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*))
using *NextLoop* **by** *blast*
hence *15*: $\vdash$ (*halt* (*init w*)); *f* $\longrightarrow$ $\neg$ (($\Box$ $\neg$ (*init w*)); $\neg$ (*halt* (*init w*); *f*))
by *auto*
from *15* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

5.10 Properties of Groups of chops

lemma *NestedChopImpChop*:
assumes $\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ *g*; (*init w1* $\wedge$ *f1*)
$\vdash$ *init w1* $\wedge$ *f1* $\longrightarrow$ *g1*; (*init w2* $\wedge$ *f2*)
shows $\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ *g*; (*g1*; (*init w2* $\wedge$ *f2*))
proof −
have *1*: $\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ *g*; (*init w1* $\wedge$ *f1*) **using** *assms*(*1*) **by** *auto*
have *2*: $\vdash$ *init w1* $\wedge$ *f1* $\longrightarrow$ *g1*; (*init w2* $\wedge$ *f2*) **using** *assms*(*2*) **by** *auto*
hence *3*: $\vdash$ *g*; (*init w1* $\wedge$ *f1*) $\longrightarrow$ *g*; (*g1*; (*init w2* $\wedge$ *f2*)) **by** (*rule RightChopImpChop*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

5.11 Properties of Time Reversal

lemma *RNot*:
$\vdash (\neg f)^r = \neg\, f^r$
by *simp*

lemma *RRNot*:
$\vdash (\neg(f^r))^r = \neg f$
by (*metis EqvReverseReverse int-eq rev-fun1*)

lemma *RTrue*:
$\vdash (\#True)^r = \#True$
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 *RNextEqvPrev*:
$\vdash (\bigcirc f)^r = prev\ (f^r)$
by (*metis RevChop RevSkip inteq-reflection next-d-def prev-d-def*)

lemma *RRNextEqvPrev*:
$\vdash (\bigcirc (f^r))^r = prev\ (f)$
proof −
have *1*: $\vdash (\bigcirc (f^r))^r = prev\ ((f^r)^r)$ **using** *RNextEqvPrev* **by** *blast*
have *2*: $\vdash prev\ ((f^r)^r) = prev\ f$ **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RWNextEqvWPrev*:

$\vdash (wnext\ f)^r = wprev(f^r)$
by (*smt RNextEqvPrev REmptyEqvEmpty WnextEqvEmptyOrNext WprevEqvEmptyOrPrev int-eq rev-fun2*)

lemma *RRWNextEqvWPrev*:
$\vdash (wnext\ (f^r))^r = wprev(f)$
proof −
have *1*: $\vdash (wnext\ (f^r))^r = wprev\ ((f^r)^r)$ **using** *RWNextEqvWPrev* **by** *blast*
have *2*: $\vdash wprev\ ((f^r)^r) = wprev\ f$ **using** *EqvReverseReverse* **by** (*metis integ-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RPrevEqvNext*:
$\vdash (prev\ f)^r = \bigcirc\ (f^r)$
by (*metis RevChop RevSkip inteq-reflection next-d-def prev-d-def*)

lemma *RRPrevEqvNext*:
$\vdash (prev\ (f^r))^r = \bigcirc\ (f)$
proof −
have *1*: $\vdash (prev\ (f^r))^r = \bigcirc\ ((f^r)^r)$ **using** *RPrevEqvNext* **by** *blast*
have *2*: $\vdash \bigcirc\ ((f^r)^r) = \bigcirc\ f$ **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RWPrevEqvWNext*:
$\vdash (wprev\ f)^r = wnext(f^r)$
by (*metis EqvReverseReverse RRWNextEqvWPrev int-eq*)

lemma *RRWPrevEqvWNext*:
$\vdash (wprev\ (f^r))^r = wnext(f)$
proof −
have *1*: $\vdash (wprev\ (f^r))^r = wnext\ ((f^r)^r)$ **using** *RWPrevEqvWNext* **by** *blast*
have *2*: $\vdash wnext\ ((f^r)^r) = wnext\ f$ **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RDiamondEqvDi*:
$\vdash (\Diamond f)^r = di\ (f^r)$
by (*simp add*: *di-d-def sometimes-d-def*, *metis RevChop RTrue inteq-reflection*)

lemma *RRDiamondEqvDi*:
$\vdash (\Diamond(f^r))^r = di\ (f)$
proof −
have *1*: $\vdash (\Diamond\ (f^r))^r = di\ ((f^r)^r)$ **using** *RDiamondEqvDi* **by** *blast*
have *2*: $\vdash di\ ((f^r)^r) = di\ f$ **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RBoxEqvBi*:
$\vdash (\Box\ f)^r = bi\ (f^r)$

by (*simp add*: *always-d-def bi-d-def*, *metis RDiamondEqvDi int-eq rev-fun1*)

lemma *RRBoxEqvBi*:
$\vdash (\Box\ (f^r))^r = bi\ (f)$
proof –
have *1*: $\vdash (\Box\ (f^r))^r = bi\ ((f^r)^r)$ **using** *RBoxEqvBi* **by** *blast*
have *2*: $\vdash bi\ ((f^r)^r) = bi\ f$ **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RDiEqvDiamond*:
$\vdash (di\ f)^r = \Diamond\ (f^r)$
by (*simp add*: *di-d-def sometimes-d-def*, *metis RevChop RTrue inteq-reflection*)

lemma *RRDiEqvDiamond*:
$\vdash (di\ (f^r))^r = \Diamond\ (f)$
proof –
have *1*: $\vdash (di\ (f^r))^r = \Diamond\ ((f^r)^r)$ **using** *RDiEqvDiamond* **by** *blast*
have *2*: $\vdash \Diamond\ ((f^r)^r) = \Diamond\ f$ **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RBiEqvBox*:
$\vdash (bi\ f)^r = \Box\ (f^r)$
by (*simp add*: *always-d-def bi-d-def*, *metis RDiEqvDiamond rev-fun1 int-eq*)

lemma *RRBiEqvBox*:
$\vdash (bi\ (f^r))^r = \Box\ (f)$
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 *ChopCsImpCSChop*:
$\vdash f;f^\star \longrightarrow f^\star;f$
by (*meson CSChopEqvChopOrRule CSChopEqvOrChopPlusChop ChopAssocB ChopPlusElimWithoutMore EmptyYields Prop03 Prop04 Prop06*)

lemma *CSChopImpChopCS*:
$\vdash f^\star;f \longrightarrow f;f^\star$
proof −
have *1*: $\vdash (f^r);(f^r)^\star \longrightarrow (f^r)^\star;(f^r)$
using *ChopCsImpCSChop* **by** *blast*
hence *2*: $\vdash ((f^r);(f^r)^\star \longrightarrow (f^r)^\star;(f^r)\ )^r$
using *ReverseEqv* **by** *blast*
have *3*: $\vdash (((f^r);(f^r)^\star \longrightarrow (f^r)^\star;(f^r)\ )^r) = (\ ((f^r);(f^r)^\star)^r \longrightarrow ((f^r)^\star;(f^r))^r)$
by (*smt 1 2 RImpRule Valid-def unl-lift2*)
have *4*: $\vdash ((f^r);(f^r)^\star)^r = ((f^r)^\star\ )^r;\ (f^r)^r$
by (*simp add*: *RevChop*)
have *5*: $\vdash ((f^r)^\star\ )^r;\ (f^r)^r = ((f^r)^r)^\star;(f^r)^r$
by (*simp add*: *LeftChopEqvChop RevChopstar*)
have *6*: $\vdash (f^r)^r = f$
using *EqvReverseReverse* **by** *blast*
have *7*: $\vdash ((f^r)^r)^\star;(f^r)^r = f^\star;f$
using *6 CSEqvCS ChopEqvChop* **by** *blast*
have *8*: $\vdash ((f^r);(f^r)^\star)^r = f^\star;f$
using *7 5* **using** *4* **by** *fastforce*
have *9*: $\vdash (((f^r)^\star;(f^r))^r = (f^r)^r;((f^r)^\star)^r$
by (*simp add*: *RevChop*)
have *10*: $\vdash (f^r)^r;((f^r)^\star)^r = (f^r)^r;\ ((f^r)^r)^\star$
by (*simp add*: *RevChopstar RightChopEqvChop*)
have *11*: $\vdash (f^r)^r;\ ((f^r)^r)^\star = f;f^\star$
using *6 ChopPlusEqvChopPlus* **by** *blast*
have *12*: $\vdash ((f^r);(f^r)^\star)^r = f;f^\star$
using *9 10 11* **by** (*metis 4 5 ChopCsImpCSChop RImpRule int-eq int-iffI*)

from *2 3 8 12* **show** *?thesis* **by** *fastforce*
qed

lemma *CSChopEqvChopCS*:
$\vdash f;f^\star = f^\star;f$
using *ChopCsImpCSChop CSChopImpChopCS* **by** *fastforce*

lemma *TrueChopSkipEqvSkipChopTrue*:
$\vdash \ \#True;skip = skip;\#True$
proof $-$
have *1*: $\vdash skip;skip^\star = skip^\star;skip$ **using** *CSChopEqvChopCS* **by** *blast*
have *2*: $\vdash skip^\star = \#True$ **using** *CSSkip* **by** *simp*
have *3*: $\vdash skip;skip^\star = skip;\#True$ **using** *2* **using** *RightChopEqvChop* **by** *blast*
have *4*: $\vdash skip^\star;skip = \#True;skip$ **using** *2* **using** *LeftChopEqvChop* **by** *blast*
from *1 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *RInitEqvFin*:
$\vdash (init\ f)^r = fin(f^r)$
proof $-$
have *1*: $\vdash (init\ f)^r = ((f \wedge empty);\#True)^r$
by (*metis AndChopCommute REqvRule init-d-def*)
have *2*: $\vdash ((f \wedge empty);\#True)^r = (\#True;(f \wedge empty)^r)$
using *RTrue* **by** (*metis RevChop int-eq*)
have *3*: $\vdash \#True;(f \wedge empty)^r = \#True;(f^r \wedge empty)$
by (*metis RAnd REmptyEqvEmpty RightChopEqvChop int-eq*)
have *4*: $\vdash \#True;(f^r \wedge empty) = fin(f^r)$
using *FinEqvTrueChopAndEmpty* **by** *fastforce*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *RRInitEqvFin*:
$\vdash (init\ (f^r))^r = fin(f)$
proof $-$
have *1*: $\vdash (init\ (f^r))^r = fin\ ((f^r)^r)$ **using** *RInitEqvFin* **by** *blast*
have *2*: $\vdash fin\ ((f^r)^r) = fin\ f$ **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RFinEqvInit*:
$\vdash (fin\ f)^r = init\ (f^r)$
proof $-$
have *1*: $\vdash fin\ f = \#True;(f \wedge empty)$
using *FinEqvTrueChopAndEmpty* **by** *auto*
have *2*: $\vdash (fin\ f)^r = (\#True;(f \wedge empty))^r$
using *1 REqvRule* **by** *blast*
have *3*: $\vdash (\#True;(f \wedge empty))^r = (f \wedge empty)^r;\#True$
using *RTrue* **by** (*metis RevChop int-eq*)
have *4*: $\vdash (f\wedge empty)^r;\#True = (f^r \wedge empty);\#True$
using *LeftChopEqvChop RAnd REmptyEqvEmpty* **by** (*metis int-eq*)

have *5*: $\vdash (f^r \wedge empty);\#True = init(f^r)$
  by (*simp add*: *AndChopCommute init-d-def*)
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *RRFinEqvInit*:
$\vdash (fin\ (f^r))^r = init\ (f)$
proof −
have *1*: $\vdash (fin\ (f^r))^r = init\ ((f^r)^r)$ **using** *RFinEqvInit* **by** *blast*
have *2*: $\vdash init\ ((f^r)^r) = init\ f$ **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *NextDiamondEqvDiamondNext*:
$\vdash \bigcirc(\ \Diamond\ f) = \Diamond(\bigcirc f)$
proof −
have *1*: $\vdash \#True;skip = skip;\#True$ **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 $\vdash wnext\ \ (\Box\ f) \longrightarrow f$
shows $\vdash f$
proof −
have *1*: $\vdash wnext\ \ (\Box\ f) \longrightarrow f$ **using** *assms* **by** *blast*
hence *2*: $\vdash \neg\ f \longrightarrow \neg\ (\ wnext\ \ (\Box\ f))$ **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

end

theory *FOTheorems*
imports
Theorems
begin
sledgehammer-params [*minimize=true,preplay-timeout=10,timeout=60,verbose=true,*
provers=vampire cvc4 z3 e spass]

6 First Order ITL theorems

We give the proofs of a list of first order ITL theorems.

lemmas *EExI-unl* = *EExI*[*unlift-rule*] — $w \models F\ x \Longrightarrow w \models (\exists\exists\ x.\ F\ x)$

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*
by (*metis* (*mono-tags*, *lifting*) *AAxDef EExE EExI int-iffI intensional-simps*(*4*) *inteq-reflection*)

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 (*smt AAxDef EExI-unl Valid-def intensional-simps*(*4*) *inteq-reflection unl-lift unl-lift2*)

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 $\vdash R \longrightarrow G\,x$
shows $\vdash R \longrightarrow (\exists\exists\ x.\ G\,x)$
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 (*metis EExI TrueW intensional-simps*(*11*) *intensional-simps*(*25*) *Prop02 Prop05 Prop08*)
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*:

$\vdash (\exists\exists\ x.\ F\ x) \longrightarrow (\exists\exists\ x.\ G \lor (F\ x))$
by (*simp add*: *EExOrDist-1*)

lemma *EExOrImport-3*:
$\vdash (G \lor (\exists\exists\ x.\ F\ x)) \longrightarrow (\exists\exists\ x.\ G \lor (F\ x))$
using *EExOrImport-1 EExOrImport-2* **by** *fastforce*

lemma *EExOrImport-4*:
$\vdash (\exists\exists\ x.\ G \lor F\ x) \longrightarrow (G \lor (\exists\exists\ x.\ F\ x))$
proof −
 have *1*: $\bigwedge x. \vdash G \lor F\ x \longrightarrow G \lor (\exists\exists\ x.\ F\ x)$ **by** (*meson EExI int-iffD2 int-simps*(*27*) *Prop04 Prop08*)
 from *1* **show** *?thesis* **by** (*simp add*: *EExE*)
qed

lemma *EExOrImport*:
$\vdash (G \lor (\exists\exists\ x.\ F\ x)) = (\exists\exists\ x.\ G \lor F\ x)$
by (*simp add*: *EExOrImport-3 EExOrImport-4 int-iffI*)

lemma *EExAndImport-1*:
$\vdash G \land (\exists\exists\ x.\ F\ x) \longrightarrow (\exists\exists\ x.\ G \land F\ x)$
proof −
 have *1*: $\vdash (G \land (\exists\exists\ x.\ F\ x) \longrightarrow (\exists\exists\ x.\ G \land F\ x))=$
 $((\exists\exists\ x.\ F\ x) \longrightarrow (G \longrightarrow (\exists\exists\ x.\ G \land F\ x)))$ **by** *fastforce*
 have *2*: $\bigwedge x. \vdash F\ x \longrightarrow (G \longrightarrow (\exists\exists\ x.\ G \land F\ x))$ **by** (*metis EExI int-eq lift-and-com Prop09*)
 hence *3*: $\vdash (\exists\exists\ x.\ F\ x) \longrightarrow (G \longrightarrow (\exists\exists\ x.\ G \land F\ x))$ **by** (*simp add*: *EExE*)
 from *1 3* **show** *?thesis* **by** *auto*
qed

lemma *EExAndImport-2*:
$\vdash (\exists\exists\ x.\ G \land F\ x) \longrightarrow G \land (\exists\exists\ x.\ F\ x)$
proof −
 have *1*: $\bigwedge x. \vdash G \land F\ x \longrightarrow G \land (\exists\exists\ x.\ F\ x)$
 by (*metis EExI int-iffD2 lift-and-com lift-imp-trans Prop12*)
 from *1* **show** *?thesis* **by** (*simp add*: *EExE*)
qed

lemma *EExAndImport*:
$\vdash (G \land (\exists\exists\ x.\ F\ x)) = (\exists\exists\ x.\ G \land F\ x)$
by (*simp add*: *EExAndImport-1 EExAndImport-2 int-iffI*)

lemma *EExAndDist*:
 assumes $\vdash F\ x \land G\ x$
 shows $\vdash (\exists\exists\ x.\ F\ x) \land (\exists\exists\ x.\ G\ x)$
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 ⊢ R ∧ G x
 shows  ⊢ R ∧ (∃∃ x. G x)
proof –
 have 1: ⊢ R  using assms by fastforce
 have 2: ⊢ G x using assms by fastforce
 have 3: ⊢ (∃∃ x. G x) using 2 by (meson EExI MP)
 from 1 3 show ?thesis by fastforce
qed

lemma Spec:
⊢ (∀∀ x. F x) ⟶ F x
proof –
 have 1: ⊢ ¬(F x) ⟶ (∃∃ x. ¬(F x)) by (meson EExI)
 have 2: ⊢ ¬(∃∃ x. ¬(F x)) ⟶ F x using 1 by auto
 from 2 show ?thesis using AAxDef by fastforce
qed

lemma AAxE:
 assumes ⊢ (∀∀ x. F x)
        ⊢ F x ⟶ R
 shows  ⊢ R
using MP Spec assms(1) assms(2) by blast

lemma AAxI:
 assumes ⋀ x. ⊢ F x
 shows  ⊢ (∀∀ x. F x)
unfolding AAxDef
by (smt AAxDef EExE assms intensional-simps(14) intensional-simps(4) inteq-reflection)

lemma AAxEqvRule:
 assumes ⋀ x. ⊢ F x = G x
 shows  ⊢ (∀∀ x. F x) = (∀∀ x. G x)
by (metis (mono-tags, lifting) AAxDef EExEqvRule assms int-iffD1 int-iffI
    inteq-reflection lift-imp-neg)


lemma AAxAndDist:
⊢ (∀∀ x. (F x) ∧ (G x)) = ((∀∀ x. F x) ∧ (∀∀ x. G x))
proof –
 have 1: ⊢ ((∃∃ x. ¬(F x)) ∨ (∃∃ x. ¬(G x))) = (∃∃ x. ¬(F x) ∨ ¬(G x)) by (simp add:EExOrDist)
 have 2: ⊢ ((∃∃ x. ¬(F x))) = (¬(∀∀ x. F x) )  using AAxDef by fastforce
 have 3: ⊢ ((∃∃ x. ¬(G x))) = (¬(∀∀ x. G x) )  using AAxDef by fastforce
 have 4: ⊢ ((∃∃ x. ¬(F x)) ∨ (∃∃ x. ¬(G x))) = (¬(∀∀ x. F x) ∨ ¬(∀∀ x. G x) )
     using 2 3 by fastforce
 have 5: ⋀ x . ⊢ (¬(F x) ∨ ¬(G x)) = (¬((F x) ∧ (G x))) by auto
```

have *6*: $\vdash (\exists\exists\ x.\ \neg(F\ x) \vee \neg(G\ x)) = (\exists\exists\ x.\ \neg((F\ x) \wedge (G\ x)))$ **using** *5* **by** (*simp add*: *EExEqvRule*)
have *7*: $\vdash (\exists\exists\ x.\ \neg((F\ x) \wedge (G\ x))) = (\neg(\forall\forall\ x.\ (F\ x) \wedge (G\ x)))$ **using** *AAxDef* **by** *fastforce*
have *8*: $\vdash (\neg(\forall\forall\ x.\ F\ x) \vee \neg(\forall\forall\ x.\ G\ x)) = \neg((\forall\forall\ x.\ F\ x) \wedge (\forall\forall\ x.\ G\ x))$ **by** *fastforce*
have *9*: $\vdash \neg((\forall\forall\ x.\ F\ x) \wedge (\forall\forall\ x.\ G\ x)) = (\neg(\forall\forall\ x.\ (F\ x) \wedge (G\ x)))$
using *1 4 6 7 8* **by** *fastforce*
from *9* **show** *?thesis* **by** *fastforce*
qed

lemma *AAxAndImport*:
$\vdash (G \wedge (\forall\forall\ x.\ F\ x)) = (\forall\forall\ x.\ G \wedge F\ x)$
proof −
have *1*: $\vdash (\neg\ G \vee (\exists\exists\ x.\ \neg(F\ x))) = (\exists\exists\ x.\ \neg\ G \vee \neg(F\ x))$ **by** (*simp add*: *EExOrImport*)
have *2*: $\vdash ((\exists\exists\ x.\ \neg(F\ x))) = \neg((\forall\forall\ x.\ F\ x))$ **using** *AAxDef* **by** *fastforce*
have *3*: $\vdash (\neg\ G \vee (\exists\exists\ x.\ \neg(F\ x))) = \neg(G \wedge (\forall\forall\ x.\ F\ x))$ **using** *2* **by** *fastforce*
have *4*: $\bigwedge x.\ \vdash (\neg\ G \vee \neg(F\ x)) = \neg(G \wedge F\ x)$ **by** *auto*
have *5*: $\vdash (\exists\exists\ x.\ \neg\ G \vee \neg(F\ x)) = (\exists\exists\ x.\ \neg(G \wedge F\ x))$ **using** *4* **by** (*simp add*: *EExEqvRule*)
have *6*: $\vdash (\exists\exists\ x.\ \neg(G \wedge F\ x)) = \neg(\forall\forall\ x.\ G \wedge F\ x)$ **using** *AAxDef* **by** *fastforce*
have *7*: $\vdash \neg(G \wedge (\forall\forall\ x.\ F\ x)) = \neg(\forall\forall\ x.\ G \wedge F\ x)$ **using** *1 3 5 6* **by** *fastforce*
from *7* **show** *?thesis* **by** *fastforce*
qed

lemma *AAxOrImport*:
$\vdash (G \vee (\forall\forall\ x.\ F\ x)) = (\forall\forall\ x.\ G \vee F\ x)$
proof −
have *1*: $\vdash (\neg\ G \wedge (\exists\exists\ x.\ \neg(F\ x))) = (\exists\exists\ x.\ \neg\ G \wedge \neg(F\ x))$ **by** (*simp add*: *EExAndImport*)
have *2*: $\vdash (\exists\exists\ x.\ \neg(F\ x)) = \neg((\forall\forall\ x.\ F\ x))$ **using** *AAxDef* **by** *fastforce*
have *3*: $\vdash (\neg\ G \wedge (\exists\exists\ x.\ \neg(F\ x))) = \neg(G \vee (\forall\forall\ x.\ F\ x))$ **using** *2* **by** *fastforce*
have *4*: $\bigwedge x.\ \vdash (\neg\ G \wedge \neg(F\ x)) = \neg(G \vee F\ x)$ **by** *auto*
have *5*: $\vdash (\exists\exists\ x.\ \neg\ G \wedge \neg(F\ x)) = (\exists\exists\ x.\ \neg(G \vee F\ x))$ **using** *4* **by** (*simp add*: *EExEqvRule*)
have *6*: $\vdash (\exists\exists\ x.\ \neg(G \vee F\ x)) = \neg(\forall\forall\ x.\ G \vee F\ x)$ **using** *AAxDef* **by** *fastforce*
have *7*: $\vdash \neg(G \vee (\forall\forall\ x.\ F\ x)) = \neg(\forall\forall\ x.\ G \vee F\ x)$ **using** *1 3 5 6* **by** *fastforce*
from *7* **show** *?thesis* **by** *auto*
qed

lemma *EExImpChopRule*:
assumes $\vdash F\ x \longrightarrow G\ x$
shows $\vdash (\exists\exists\ x.\ H;(F\ x) \longrightarrow H;(G\ x))$
using *RightChopImpChop EExImpRule assms* **by** (*smt MP EExI*)

lemma *EExChopRight*:
$\vdash (\exists\exists\ x.\ (F\ x);F1) \longrightarrow (\exists\exists\ x.\ F\ x);F1$
proof −
have *1*: $\bigwedge x.\ \vdash (F\ x);F1 \longrightarrow (\exists\exists\ x.\ F\ x);F1$ **by** (*simp add*: *EExI LeftChopImpChop*)
from *1* **show** *?thesis* **by** (*simp add*: *EExE*)
qed

lemma *EExChopRightNoDep*:
$\vdash (\exists\exists\ x.\ (F\ x);F1) = (\exists\exists\ x.\ (F\ x));F1$
by (*simp add*: *exist-state-d-def Valid-def chop-defs*, *auto*)

lemma *EExChopLeft* :
$\vdash (\exists\exists\ x.\ F1;(F\ x)\ ) \longrightarrow F1;(\exists\exists\ x.\ F\ x)$
proof –
have *1*: $\bigwedge x. \vdash F1;(F\ x) \longrightarrow F1;(\exists\exists\ x.\ F\ x)$ **by** (*simp add*: *EExI RightChopImpChop*)
from *1* **show** *?thesis* **by** (*simp add*: *EExE*)
qed

lemma *EExChopLeftNoDep*:
$\vdash (\exists\exists\ x.\ F1;(F\ x)\ ) = F1;(\exists\exists\ x.\ F\ x)$
by (*simp add*: *exist-state-d-def Valid-def chop-defs*, *auto*)

lemma *EExEExChopEqvEExEExChop*:
$\vdash (\exists\exists\ v.\ (\exists\exists\ y.\ (F2\ v);(F3\ y)\ )) = (\exists\exists\ y.\ (\exists\exists\ v.\ (F2\ v);(F3\ y)\ ))$
by (*simp add*: *exist-state-d-def Valid-def chop-defs*, *blast*)

lemma *EExEExChopEqvEExChopEExA*:
$\vdash (\exists\exists\ v.\ (\exists\exists\ y.\ (F2\ v);(F3\ y)\ )) = (\exists\exists\ v.\ (F2\ v);(\exists\exists\ y.\ (F3\ y)\ )\ )$
by (*simp add*: *exist-state-d-def Valid-def chop-defs*, *blast*)

lemma *EExEExChopEqvEExChopEExB*:
$\vdash (\exists\exists\ y.\ (\exists\exists\ v.\ (F2\ v);(F3\ y)\ )) = (\exists\exists\ y.\ (\exists\exists\ v.\ (F2\ v));\ (F3\ y))$
by (*simp add*: *exist-state-d-def Valid-def chop-defs*, *blast*)

lemma *EExEExChopEqvEExChopEExC*:
$\vdash (\exists\exists\ v.\ (\exists\exists\ y.\ (F2\ v);(F3\ y)\ )) = (\exists\exists\ v.\ (F2\ v));(\exists\exists\ y.\ (F3\ y))$
by (*metis EExChopRightNoDep EExEExChopEqvEExChopEExA EExNoDep Prop04*)

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

lemma *ExLen*:
$\vdash \exists\ n.\ len(n)$
by (*simp add*: *Valid-def len-defs*)

lemma *CSPowerChop*:
$\vdash (f^\star) = (\exists\, n.\ powerchop\ f\ n)$
by (*simp add*: *chopstar-eqv-power-chop Valid-def*)

lemma *ExChopRightNoDep*:
$\vdash (\exists\ x.\ (F\ x);F1) = (\exists\ x.\ (F\ x));F1$
by (*simp add*: *Valid-def chop-defs*, *auto*)

lemma *ExChopLeftNoDep*:
$\vdash (\exists\ x.\ F1;(F\ x)\ ) = F1;(\exists\ x.\ F\ x)$
by (*simp add*: *Valid-def chop-defs*, *auto*)

lemma *ExExEqvExEx*:
$\vdash (\exists\ x.\ (\exists\ y.\ (F1\ x);(F2\ y))) = (\exists\ y.\ (\exists\ x.\ (F1\ x);(F2\ y)))$
by (*simp add*: *Valid-def chop-defs*, *auto*)

end

theory *First*
imports
Theorems
begin

7 The First Occurrence Operator in ITL

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 thesis 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.

7.1 Definitions

7.1.1 Definitions Strict Initial and Final

definition *bs-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where
bs-d f ≡ *LIFT*(*empty* ∨ ((*bi f*) ; *skip*))

definition *bt-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where
bt-d f ≡ *LIFT*(*empty* ∨ (*skip*;(□ *f*)))

syntax
-bs-d :: *lift* ⇒ *lift* ((*bs* -) [*88*] *87*)
-bt-d :: *lift* ⇒ *lift* ((*bt* -) [*88*] *87*)

syntax (*ASCII*)
-bs-d :: *lift* ⇒ *lift* ((*bs* -) [*88*] *87*)
-bt-d :: *lift* ⇒ *lift* ((*bt* -) [*88*] *87*)

translations
-bs-d ⇌ *CONST bs-d*
-bt-d ⇌ *CONST bt-d*

definition *ds-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where
ds-d f ≡ *LIFT* (¬ (*bs* (¬ *f*)))

definition *dt-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where
dt-d f ≡ *LIFT* (¬ (*bt* (¬ *f*)))

syntax
-ds-d :: *lift* ⇒ *lift* ((*ds* -) [*88*] *87*)
-dt-d :: *lift* ⇒ *lift* ((*dt* -) [*88*] *87*)

syntax (*ASCII*)
-ds-d :: *lift* ⇒ *lift* ((*ds* -) [*88*] *87*)
-dt-d :: *lift* ⇒ *lift* ((*dt* -) [*88*] *87*)

translations
-ds-d ⇌ *CONST ds-d*
-dt-d ⇌ *CONST dt-d*

7.1.2 Definition First and Last Operators

definition *first-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where
first-d f ≡ *LIFT* (*f* ∧ (*bs* (¬ *f*)))

definition *last-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*
where
last-d f ≡ *LIFT* (*f* ∧ (*bt* (¬ *f*)))

syntax
-first-d :: *lift* ⇒ *lift* ((▷ -) [*88*] *87*)
-last-d :: *lift* ⇒ *lift* ((◁ -) [*88*] *87*)

syntax (*ASCII*)
-first-d :: *lift* ⇒ *lift* ((*first* -) [*88*] *87*)
-last-d :: *lift* ⇒ *lift* ((*last* -) [*88*] *87*)

translations
-first-d ⇌ *CONST first-d*
-last-d ⇌ *CONST last-d*

7.2 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 RBiEqvBox RevSkip int-eq intensional-simps*(*1*))
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 RBoxEqvBi RevSkip intensional-simps*(*1*) *inteq-reflection*)
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*:
$\vdash (\rhd\ f)^r = (\lhd\ (f^r))$
proof $-$
have *1*: $\vdash (\rhd\ f)^r = (f \wedge bs(\neg\ f))^r$ **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*: $\vdash (f^r \wedge bt\ ((\neg\ f)^r)) = (f^r \wedge bt\ (\neg(f^r)))$ **by** (*metis int-eq intensional-simps*(*1*) *rev-fun1*)
have *5*: $\vdash (f^r \wedge bt\ (\neg(f^r))) = (\lhd\ (f^r))$ **by** (*simp add*: *last-d-def*)
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *RRFirstEqvLast*:
$\vdash (\rhd\ (f^r))^r = (\lhd\ (f))$
proof $-$
have *1*: $\vdash (\rhd\ (f^r))^r = \lhd\ ((f^r)^r)$ **using** *RFirstEqvLast* **by** *blast*
have *2*: $\vdash \lhd\ ((f^r)^r) = \lhd\ f$ **using** *EqvReverseReverse* **using** *LstEqvRule* **by** *blast*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RLastEqvFirst*:
$\vdash (\lhd\ f)^r = (\rhd\ (f^r))$
proof $-$
have *1*: $\vdash (\lhd\ f)^r = (f \wedge bt(\neg\ f))^r$ **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*: $\vdash (f^r \wedge bs\ (\ (\neg f)^r)) = (f^r \wedge bs(\neg(f^r)))$ **by** (*metis int-eq intensional-simps*(*1*) *rev-fun1*)
have *5*: $\vdash (f^r \wedge bs(\neg(f^r))) = (\rhd\ (f^r))$ **by** (*simp add*: *first-d-def*)
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *RRLastEqvFirst*:
$\vdash (\lhd\ (f^r))^r = (\rhd\ (f))$
proof $-$
have *1*: $\vdash (\lhd\ (f^r))^r = \rhd\ ((f^r)^r)$ **using** *RLastEqvFirst* **by** *blast*
have *2*: $\vdash \rhd\ ((f^r)^r) = \rhd\ f$ **using** *EqvReverseReverse* **using** *FstEqvRule* **by** *blast*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

7.3 Semantic Theorems

7.3.1 Semantics First and Last Operators

lemma *FstAndBisem*:
($intlen\ \sigma > 0 \wedge (\sigma \models f) \wedge (\ \sigma \models\ \ bi\ \neg f;skip)) =$
($intlen\ \sigma > 0 \wedge (\sigma \models f) \wedge (\forall\, ia{<}intlen\ (\sigma).\ (prefix\ ia\ \ \sigma \models \neg f))\ )$
apply (*simp add*: *chop-defs bi-defs skip-defs*)

apply (*simp add*: *interval-prefix-length interval-suffix-length*)
proof $-$
have *1*: $(0 < \mathit{intlen}\ \sigma \wedge (\sigma \models f) \wedge$
$(\exists i.\ (i \leq \mathit{intlen}\ \sigma \longrightarrow (\forall ia{\leq}i.\ \neg\ (\mathit{prefix}\ ia\ (\mathit{prefix}\ i\ \sigma) \models f)) \wedge$
$\mathit{intlen}\ \sigma - i = \mathit{Suc}\ 0) \wedge i \leq \mathit{intlen}\ \sigma)$
$) =$
$(0 < \mathit{intlen}\ \sigma \wedge (\sigma \models f) \wedge$
$(\exists i.\ (i \leq \mathit{intlen}\ \sigma \longrightarrow (\forall ia{\leq}i.\ \neg\ (\mathit{prefix}\ ia\ (\mathit{prefix}\ i\ \sigma) \models f)) \wedge$
$i = \mathit{intlen}\ \sigma - \mathit{Suc}\ 0) \wedge i \leq \mathit{intlen}\ \sigma)$
$)$
by *auto*
also have ... $=$
$(0 < \mathit{intlen}\ \sigma \wedge (\sigma \models f) \wedge$
$(\forall ia{\leq}(\mathit{intlen}\ \sigma - \mathit{Suc}\ 0).\ \neg\ (\mathit{prefix}\ ia\ (\mathit{prefix}\ (\mathit{intlen}\ \sigma - \mathit{Suc}\ 0)\ \sigma) \models f)\ )$
$)$
using *diff-le-self* **by** *blast*
also have ... $=$
$(\mathit{intlen}\ \sigma > 0 \wedge (\sigma \models f) \wedge$
$(\forall ia{<}\mathit{intlen}\ (\ \sigma).\ \neg\ (\mathit{prefix}\ ia\ (\mathit{prefix}\ (\mathit{intlen}\ \sigma - \mathit{Suc}\ 0)\ \sigma) \models f))$
$)$ **by** (*metis Suc-pred less-Suc-eq-le*)
also have ... $=$
$(\mathit{intlen}\ \sigma > 0 \wedge (\sigma \models f) \wedge$
$(\forall ia{<}\mathit{intlen}\ (\ \sigma).\ (\mathit{prefix}\ ia\ (\mathit{prefix}\ (\mathit{intlen}\ \sigma - \mathit{Suc}\ 0)\ \sigma) \models \neg f))$
$)$
by *auto*
also have ... $=$
$(\mathit{intlen}\ \sigma > 0 \wedge (\sigma \models f) \wedge (\forall ia{<}\mathit{intlen}\ (\ \sigma).\ \neg\ (\mathit{prefix}\ ia\ \ \sigma \models f)))$
by (*simp add*: *interval-pref-pref-help*)
finally show $(0 < \mathit{intlen}\ \sigma \wedge (\sigma \models f) \wedge$
$(\exists i.\ (i \leq \mathit{intlen}\ \sigma \longrightarrow (\forall ia{\leq}i.\ \neg\ (\mathit{prefix}\ ia\ (\mathit{prefix}\ i\ \sigma) \models f)) \wedge$
$\mathit{intlen}\ \sigma - i = \mathit{Suc}\ 0) \wedge i \leq \mathit{intlen}\ \sigma)$
$) =$
$(0 < \mathit{intlen}\ \sigma \wedge (\sigma \models f) \wedge (\forall ia{<}\mathit{intlen}\ \sigma.\ \neg\ (\mathit{prefix}\ ia\ \sigma \models f)))$ **.**
qed

lemma *Fstsem-0*:
$(\sigma \models \triangleright f) =$
$($
$(\ \sigma \models f \wedge \mathit{empty}) \vee (\mathit{intlen}\ \sigma > 0\ \wedge (\sigma \models f) \wedge (\ \sigma \models\ \ \mathit{bi}\ \neg f;\mathit{skip}))$
$)$
apply (*simp add*: *first-d-def bs-d-def*) **using** *empty-defs* **by** *auto*

lemma *Emptysem*:
$(\sigma \models f \wedge \mathit{empty}) = ((\sigma \models f) \wedge \mathit{intlen}\ \sigma = 0)$
using *empty-defs* **by** *auto*

lemma *Fstsem*:
$(\sigma \models \triangleright f) =$
$($
$(\ (\sigma \models f) \wedge \mathit{intlen}\ \sigma = 0) \vee$

(*intlen* σ>*0* $\wedge$ ($\sigma \models f$) $\wedge$ ($\forall$ *ia*<*intlen* (σ). (*prefix ia* $\sigma \models \neg f$)))
)
using *Fstsem-0 Emptysem FstAndBisem* **by** *metis*

lemma *Lstsem*:
($\sigma \models \lhd f$) =
((($\sigma \models f$) $\wedge$ *intlen* σ = *0*) $\vee$
(*intlen* σ >*0* $\wedge$ ($\sigma \models f$) $\wedge$ ($\forall$ *ia* < *intlen* σ. (*suffix* ((*intlen* σ) −*ia*) $\sigma \models \neg f$)))
)

proof −
have ($\sigma \models \lhd f$) = ($\sigma \models (\rhd (f^r))^r$)
using *RRFirstEqvLast* **by** *fastforce*
also have ... = (*intrev* $\sigma \models \rhd (f^r)$)
by (*metis reverse-d-def*)
also have ... =
(
((*intrev* $\sigma \models f^r$) $\wedge$ *intlen* (*intrev* σ) = *0*) $\vee$
(*intlen* (*intrev* σ)>*0* $\wedge$ (*intrev* $\sigma \models f^r$) $\wedge$
($\forall$ *ia*<*intlen* (*intrev* σ). (*prefix ia* (*intrev* σ) $\models \neg(f^r)$)))
)
using *Fstsem* **by** *blast*
also have ... =
(
(($\sigma \models f$) $\wedge$ *intlen* (σ) = *0*) $\vee$
(*intlen* (σ)>*0* $\wedge$ ($\sigma \models f$) $\wedge$
($\forall$ *ia*<*intlen* (*intrev* σ). (*prefix ia* (*intrev* σ) $\models (\neg(f))^r$)))
)
by (*simp add*: *reverse-d-def*)
also have ... =
(
(($\sigma \models f$) $\wedge$ *intlen* (σ) = *0*) $\vee$
(*intlen* (σ)>*0* $\wedge$ ($\sigma \models f$) $\wedge$
($\forall$ *ia*<*intlen* (*intrev* σ). (*intrev* (*prefix ia* (*intrev* σ)) $\models$ ($\neg$(*f*)))))
)
by (*simp add*: *reverse-d-def*)
also have ... =
(
(($\sigma \models f$) $\wedge$ *intlen* (σ) = *0*) $\vee$
(*intlen* (σ)>*0* $\wedge$ ($\sigma \models f$) $\wedge$
($\forall$ *ia*<*intlen* (σ). ((*suffix* ((*intlen* σ) − *ia*) (σ)) $\models$ ($\neg$(*f*)))))
)
by (*simp add*: *interval-intrev-prefix*)
finally show
($\sigma \models \lhd f$) =
((($\sigma \models f$) $\wedge$ *intlen* σ = *0*) $\vee$
(*intlen* σ >*0* $\wedge$ ($\sigma \models f$) $\wedge$
($\forall$ *ia* < *intlen* σ. (*suffix* ((*intlen* σ) −*ia*) $\sigma \models \neg f$)))
) .
qed

7.3.2 Various Semantic Lemmas

lemma *DiLensem*:
$(\sigma \models di\ (f \wedge len(i))) =$
$(\ (prefix\ i\ \sigma \models f) \wedge i \leq intlen\ \sigma)$
apply (*simp add*: *di-defs len-defs*)
using *interval-prefix-length-good* **by** *auto*

lemma *PrefixFstsem*:
$(\ (prefix\ i\ \sigma \models \triangleright f) \wedge i \leq intlen\ \sigma) =$
$(\ i \leq intlen\ \sigma \wedge$
$($
$(\ (prefix\ i\ \sigma \models f) \wedge i = 0) \vee$
$(\ i > 0 \wedge (prefix\ i\ \sigma \models f) \wedge (\forall ia < i.\ (prefix\ ia\ \sigma \models \neg f)))$
$)$
$)$
proof −
have *1*: $(\ ((prefix\ i\ \sigma) \models \triangleright f)) =$
$($
$(\ ((prefix\ i\ \sigma) \models f) \wedge intlen\ (prefix\ i\ \sigma) = 0) \vee$
$(\ intlen\ (prefix\ i\ \sigma) > 0 \wedge ((prefix\ i\ \sigma) \models f) \wedge$
$(\forall ia < intlen\ (prefix\ i\ \sigma).\ (prefix\ ia\ \ (prefix\ i\ \sigma) \models \neg f))\ )$
$)$
using *Fstsem* **by** *blast*
hence *2*: $(\ ((prefix\ i\ \sigma) \models \triangleright f) \wedge i \leq intlen\ \sigma) =$
$(\ \ i \leq intlen\ \sigma \wedge ($
$(\ ((prefix\ i\ \sigma) \models f) \wedge intlen\ (prefix\ i\ \sigma) = 0) \vee$
$(\ intlen\ (prefix\ i\ \sigma) > 0 \wedge ((prefix\ i\ \sigma) \models f) \wedge$
$(\forall ia < intlen\ (prefix\ i\ \sigma).\ (prefix\ ia\ \ (prefix\ i\ \sigma) \models \neg f))\ )$
$)$
$)$
by *auto*
hence *3*: $(\ ((prefix\ i\ \sigma) \models \triangleright f) \wedge i \leq intlen\ \sigma) =$
$(\ \ i \leq intlen\ \sigma \wedge ($
$(\ ((prefix\ i\ \sigma) \models f) \wedge i = 0) \vee$
$(\ i > 0 \wedge ((prefix\ i\ \sigma) \models f) \wedge (\forall ia < i.\ (prefix\ ia\ \ (prefix\ i\ \sigma) \models \neg f)))$
$)$
$)$
by *auto*
hence *4*: $(\ ((prefix\ i\ \sigma) \models \triangleright f) \wedge i \leq intlen\ \sigma) =$
$(\ \ i \leq intlen\ \sigma \wedge ($
$(\ ((prefix\ i\ \sigma) \models f) \wedge i = 0) \vee$
$(\ i > 0 \wedge ((prefix\ i\ \sigma) \models f) \wedge (\forall ia < i.\ (prefix\ ia\ \sigma \models \neg f)))$
$)$
$)$
using *interval-pref-pref-3* **using** *less-imp-add-positive* **by** *fastforce*
from *4* **show** *?thesis* **by** *auto*
qed

lemma *PrefixFstAndsem*:
$(\ (prefix\ i\ \sigma \models \triangleright f \wedge g) \wedge i \leq intlen\ \sigma) =$

$(\ i \le intlen\ \sigma \ \wedge$
$($
$(\ (prefix\ i\ \sigma \models f \wedge g \ ) \wedge i = 0) \vee$
$(\ i > 0 \wedge (prefix\ i\ \sigma \models f \wedge g) \wedge (\forall\, ia < i.\ (prefix\ ia\ \sigma \models \neg f)))$
$)$
$)$
using *PrefixFstsem* **by** (*metis unl-lift2*)

lemma *DiLenFstsem*:
$(\sigma \models di\ (\rhd f \wedge len(i))) =$
$(\ i \le intlen\ \sigma \ \wedge$
$($
$(\ (prefix\ i\ \sigma \models f) \wedge i = 0) \vee$
$(\ i > 0 \wedge (prefix\ i\ \sigma \models f) \wedge (\forall\, ia < i.\ (prefix\ ia\ \sigma \models \neg f)))$
$)$
$)$
by (*simp add*: *DiLensem PrefixFstsem*)

lemma *DiLenFstAndsem*:
$(\sigma \models di\ ((\rhd f \wedge g) \wedge len(i))) =$
$(\ i \le intlen\ \sigma \ \wedge$
$($
$(\ (prefix\ i\ \sigma \models f \wedge g) \wedge i = 0) \vee$
$(\ i > 0 \wedge (prefix\ i\ \sigma \models f \wedge g) \wedge (\forall\, ia < i.\ (prefix\ ia\ \sigma \models \neg f)))$
$)$
$)$
using *DiLensem PrefixFstAndsem* **by** *metis*

lemma *FstLenSamesem*:
$(\ (\ i \le intlen\ \sigma \ \wedge$
$($
$(\ (prefix\ i\ \sigma \models f) \wedge i = 0) \vee$
$(\ i > 0 \wedge (prefix\ i\ \sigma \models f) \wedge (\forall\, ia < i.\ (prefix\ ia\ \sigma \models \neg f)))$
$)$
$) \wedge$
$(\ j \le intlen\ \sigma \ \wedge$
$($
$(\ (prefix\ j\ \sigma \models f) \wedge j = 0) \vee$
$(\ j > 0 \wedge (prefix\ j\ \sigma \models f) \wedge (\forall\, ia < j.\ (prefix\ ia\ \sigma \models \neg f)))$
$)$
$)$
$) \longrightarrow (i = j)$

by (*metis not-less-iff-gr-or-eq unl-lift*)

7.4 Theorems

7.4.1 Fixed length intervals

lemma *LenZeroEqvEmpty*:
$\vdash len(0) = empty$

by *simp*

lemma *LenOneEqvSkip*:
⊢ *len*(*1*) = *skip*
by (*simp add*: *len-d-def ChopEmpty*)

lemma *LenNPlusOneA*:
⊢ *len*(*n*+*1*) = *skip*;*len*(*n*)
by *simp*

lemma *LenEqvLenChopLen*:
⊢ *len*(*i*+*j*) = *len*(*i*);*len*(*j*)
proof
(*induct i*)
case *0*
then show *?case*
by (*metis EmptyChop comm-monoid-add-class.add-0 int-eq len-d.simps*(*1*))
next
case (*Suc i*)
then show *?case*
by (*metis ChopAssoc add-Suc inteq-reflection len-d.simps*(*2*))
qed

lemma *LenNPlusOneB*:
⊢ *len*(*n*+*1*) = *len*(*n*);*skip*
proof −
have *1*: ⊢ *len*(*n*+*1*) = *len*(*n*);*len*(*1*) **by** (*rule LenEqvLenChopLen*)
have *2*: ⊢ *len*(*1*) = *skip* **by** (*rule LenOneEqvSkip*)
hence *3*: ⊢ *len*(*n*);*len*(*1*) = *len*(*n*);*skip* **using** *RightChopEqvChop* **by** *blast*
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *LenCommute*:
⊢ (*skip*;(*len n*)) = (*len n*);*skip*
proof
(*induct n*)
case *0*
then show *?case* **using** *EmptyChop ChopEmpty len-0* **by** (*metis int-eq*)
next
case (*Suc n*)
then show *?case* **using** *ChopAssoc len-Suc* **by** (*metis inteq-reflection*)
qed

lemma *PowerCommute*:
⊢ (*f*;(*power f n*)) = ((*power f n*);*f*)
proof
(*induct n*)
case *0*
then show *?case* **using** *EmptyChop ChopEmpty pow-0* **by** (*metis int-eq*)
next

case (*Suc n*)
then show *?case* **using** *ChopAssoc pow-Suc* **by** (*metis inteq-reflection*)
qed

lemma *PowerRev*:
$\vdash$ (*power skip n*)^r = (*power skip n*)
proof
(*induct n*)
case *0*
then show *?case* **using** *REmptyEqvEmpty* **by** *auto*
next
case (*Suc n*)
then show *?case* **using** *PowerCommute RevChop pow-Suc* **by** (*metis RevSkip int-eq*)
qed

lemma *RLenEqvLen*:
$\vdash$ (*len k*)^r = (*len k*)
proof
(*induct k*)
case *0*
then show *?case* **using** *REmptyEqvEmpty* **by** *auto*
next
case (*Suc k*)
then show *?case* **using** *LenCommute RevChop len-Suc* **by** (*metis RevSkip int-eq*)
qed

lemma *PowerSkipEqvLen*:
$\vdash$ (*power skip n*) = (*len n*)
proof
(*induct n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then show *?case* **by** (*metis int-eq intensional-simps*(*1*) *len-Suc pow-Suc*)
qed

lemma *ExistsLen*:
$\vdash \exists\ k.\ len(k)$
by (*simp add*: *len-defs Valid-def*)

lemma *AndExistsLen*:
$\vdash f = (f \wedge (\exists k.\ len(k)))$
using *ExistsLen* **by** *fastforce*

lemma *AndExistsLenChop*:
$\vdash (f;g) = \ (\exists k.\ (f \wedge len(k));g)$
by (*simp add*: *Valid-def len-defs chop-defs*)

lemma *AndExistsLenChopR*:

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

lemma *LFixedAndDistr*:
$\vdash ((f0 \wedge len(k));g0 \wedge (f1 \wedge len(k));g1) = ((f0 \wedge f1) \wedge len(k));(g0 \wedge g1)$
apply (*simp add*: *Valid-def len-defs chop-defs interval-prefix-length interval-suffix-length*)
by *blast*

lemma *RFixedAndDistr*:
$\vdash (f0;(g0 \wedge len(k)) \wedge f1;(g1 \wedge len(k))) = (f0 \wedge f1);((g0 \wedge g1) \wedge len(k))$
apply (*simp add*: *Valid-def len-defs chop-defs interval-prefix-length interval-suffix-length*)
by (*metis diff-diff-cancel*)

lemma *LFixedAndDistrA*:
$\vdash ((f0 \wedge len(k));g0 \wedge (f1 \wedge len(k));g0) = ((f0 \wedge f1) \wedge len(k));g0$
proof −
 have *1*: $\vdash ((f0 \wedge len(k));g0 \wedge (f1 \wedge len(k));g0) = ((f0 \wedge f1) \wedge len(k));(g0 \wedge g0)$
  by (*rule LFixedAndDistr*)
 have *2*: $\vdash ((f0 \wedge f1) \wedge len(k));(g0 \wedge g0) = ((f0 \wedge f1) \wedge len(k));g0$
  by *auto*
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *LFixedAndDistrB*:
$\vdash ((f0 \wedge len(k));g0 \wedge (f0 \wedge len(k));g1) = (f0 \wedge len(k));(g0 \wedge g1)$
proof −
 have *1*: $\vdash ((f0 \wedge len(k));g0 \wedge (f0 \wedge len(k));g1) = ((f0 \wedge f0) \wedge len(k));(g0 \wedge g1)$
  by (*rule LFixedAndDistr*)
 have *2*: $\vdash ((f0 \wedge f0) \wedge len(k));(g0 \wedge g1) = (f0 \wedge len(k));(g0 \wedge g1)$
  by *auto*
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RFixedAndDistrA*:
$\vdash (f0;(g0 \wedge len(k)) \wedge f0;(g1 \wedge len(k))) = f0;((g0 \wedge g1) \wedge len(k))$
proof −
 have *1*: $\vdash (f0;(g0 \wedge len(k)) \wedge f0;(g1 \wedge len(k))) = (f0 \wedge f0);((g0 \wedge g1) \wedge len(k))$
  by (*rule RFixedAndDistr*)
 have *2*: $\vdash (f0 \wedge f0);((g0 \wedge g1) \wedge len(k)) = f0;((g0 \wedge g1) \wedge len(k))$
  by *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RFixedAndDistrB*:
$\vdash (f0;(g0 \wedge len(k)) \wedge f1;(g0 \wedge len(k))) = (f0 \wedge f1);(g0 \wedge len(k))$
proof −
 have *1*: $\vdash (f0;(g0 \wedge len(k)) \wedge f1;(g0 \wedge len(k))) = (f0 \wedge f1);((g0 \wedge g0) \wedge len(k))$
  by (*rule RFixedAndDistr*)
 have *2*: $\vdash (f0 \wedge f1);((g0 \wedge g0) \wedge len(k)) = (f0 \wedge f1);(g0 \wedge len(k))$
  by *auto*

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

lemma *ChopSkipAndChopSkip*:
$\vdash$ (*f0*;*skip* $\wedge$ *f1*;*skip*) = (*f0* $\wedge$ *f1*);*skip*
proof –
 have *1*: $\vdash$ (*f0*;(#*True* $\wedge$ *len*(*1*)) $\wedge$ *f1*;(#*True* $\wedge$ *len*(*1*))) = (*f0* $\wedge$ *f1*);(#*True* $\wedge$ *len*(*1*))
  by (*rule RFixedAndDistrB*)
 have *2*: $\vdash$ (#*True* $\wedge$ *len*(*1*)) = *skip*
  using *LenOneEqvSkip* **by** *fastforce*
 hence *3*: $\vdash$ *f0*;(#*True* $\wedge$ *len*(*1*)) = *f0*;*skip*
  using *RightChopEqvChop* **by** *blast*
 have *4*: $\vdash$ *f1*;(#*True* $\wedge$ *len*(*1*)) = *f1*;*skip*
  using *2 RightChopEqvChop* **by** *blast*
 have *5*: $\vdash$ (*f0*;(#*True* $\wedge$ *len*(*1*)) $\wedge$ *f1*;(#*True* $\wedge$ *len*(*1*))) = (*f0*;*skip* $\wedge$ *f1*;*skip*)
  using *3 4* **by** *fastforce*
 have *6*: $\vdash$ (*f0* $\wedge$ *f1*);(#*True* $\wedge$ *len*(*1*)) = (*f0* $\wedge$ *f1*);*skip*
  using *2 RightChopEqvChop* **by** *blast*
 from *1 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *BiAndChopSkipEqv*:
$\vdash$ (*bi* (*f* $\wedge$ *g*));*skip* = ((*bi f*);*skip* $\wedge$ (*bi g*);*skip*)
proof –
 have *1*: $\vdash$ *bi* (*f* $\wedge$ *g*) = ((*bi f*) $\wedge$ (*bi g*))
  by (*simp add*: *bi-defs Valid-def*, *auto*)
 hence *2*: $\vdash$ (*bi* (*f* $\wedge$ *g*));*skip* = (*bi f* $\wedge$ *bi g*);*skip*
  by (*rule LeftChopEqvChop*)
 have *3*: $\vdash$ (*bi f* $\wedge$ *bi g*);*skip* = ((*bi f*);*skip* $\wedge$ (*bi g*);*skip*)
  using *ChopSkipAndChopSkip* **by** *fastforce*
 from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DiAndChopSkipEqv*:
$\vdash$ (*di* (*f* $\wedge$ *g*));*skip* $\longrightarrow$ (*di f*);*skip* $\wedge$ (*di g*);*skip*
proof –
 have *1*: $\vdash$ *di* (*f* $\wedge$ *g*) $\longrightarrow$ (*di f*) $\wedge$ (*di g*)
  by (*simp add*: *DiAndImpAnd*)
 hence *2*: $\vdash$ (*di* (*f* $\wedge$ *g*));*skip* $\longrightarrow$ (*di f* $\wedge$ *di g*);*skip*
  by (*rule LeftChopImpChop*)
 have *3*: $\vdash$ (*di f* $\wedge$ *di g*);*skip* = ((*di f*);*skip* $\wedge$ (*di g*);*skip*)
  using *ChopSkipAndChopSkip* **by** *fastforce*
 from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *NotNotChopSkip*:
$\vdash$ $\neg$($\neg$ *f* ;*skip*) = (*empty* $\vee$ (*f* ;*skip*))
by (*metis WprevEqvEmptyOrPrev prev-d-def wprev-d-def*)

lemma *NotChopFixed*:

$\vdash \neg(f;(g \wedge len(k))) = (\neg(\Diamond(g \wedge len(k))) \vee (\neg f;(g \wedge len(k))))$
apply (*simp add*: *len-defs Valid-def sometimes-defs chop-defs interval-suffix-length*)
by (*smt diff-diff-cancel*)

lemma *NotFixedChop*:
$\vdash \neg((g \wedge len(k));f) = (\neg(di(g \wedge len(k))) \vee ((g \wedge len(k));\neg f))$
by (*simp add*: *len-defs Valid-def di-defs chop-defs interval-prefix-length*, *auto*)

7.4.2 Strict initial intervals

lemma *DsMoreDi*:
$\vdash ds\ f = (more \wedge (di\ f);skip)$
proof −
have *1*: $\vdash ds\ f = \neg(bs\ \neg\ f)$
  by (*simp add*: *ds-d-def*)
have *2*: $\vdash \neg(bs\ \neg\ f) = \neg(empty \vee (bi\ \neg\ f);skip)$
  by (*simp add*: *bs-d-def*)
have *3*: $\vdash\neg(empty \vee (bi\ \neg\ f);skip) = (\neg empty \wedge \neg((bi\ \neg f);skip))$
  by *auto*
have *4*: $\vdash (\neg empty \wedge \neg((bi\ \neg f);skip)) = (more \wedge \neg((bi\ \neg f);skip))$
  using *NotEmptyEqvMore* **by** *auto*
have *5*: $\vdash (more \wedge \neg((bi\ \neg f);skip)) = (more \wedge \neg(\neg(di\ f);skip))$
  by (*metis DiEqvNotBiNot NotChopSkipEqvMoreAndNotChopSkip intensional-simps*(*4*)
   inteq-reflection)
have *6*: $\vdash (more \wedge \neg(\neg(di\ f);skip)) = (more \wedge (empty \vee (di\ f);skip))$
  using *NotNotChopSkip* **by** *fastforce*
have *7*: $\vdash (more \wedge (empty \vee (di\ f);skip)) = (more \wedge\ \ (di\ f);skip)$
  using *NotEmptyEqvMore* **by** *auto*
from *1 2 3 4 5 6 7* **show** *?thesis* **by** *fastforce*
qed

lemma *DsDi*:
$\vdash ds\ f = (di\ f);skip$
proof −
have *1*: $\vdash ds\ f = (more \wedge (di\ f);skip)$ **by** (*rule DsMoreDi*)
have *2*: $\vdash (di\ f);skip \longrightarrow more$ **by** (*metis DiIntro DiSkipEqvMore RightChopImpMoreRule int-eq*)
hence *3*: $\vdash (more \wedge (di\ f);skip) = (di\ f);skip$ **by** *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BsEqvNotDsNot*:
$\vdash bs\ f = \neg(ds\ \neg\ f)$
proof −
have *1*: $\vdash ds\ \neg\ f\ = (more \wedge (di\ \neg\ f);skip)$
  by (*rule DsMoreDi*)
hence *2*: $\vdash \neg(ds\ \neg\ f)\ = \neg(more \wedge (di\ \neg\ f);skip)$
  by *auto*
have *3*: $\vdash \neg(more \wedge (di\ \neg\ f);skip) = (empty \vee \neg((di\ \neg\ f);skip))$
  using *NotEmptyEqvMore* **by** *auto*
have *4*: $\vdash (empty \vee \neg((di\ \neg\ f);skip)) = (empty \vee \neg(\neg(bi\ f);skip))$

using *DiNotEqvNotBi* **by** (*metis 3 inteq-reflection*)
have *5*: $\vdash\ \neg(\neg(bi\ f);skip) = (empty \vee (bi\ f);skip)$
by (*rule NotNotChopSkip*)
hence *6*: $\vdash (empty \vee \neg(\neg(bi\ f);skip)) = (empty \vee (bi\ f);skip)$
by *auto*
from *2 3 4 6* **show** *?thesis* **by** (*metis bs-d-def inteq-reflection*)
qed

lemma *NotBsEqvDsNot*:
$\vdash \neg(bs\ f) = ds \neg f$
proof −
have *1*: $\vdash bs\ f = \neg(ds \neg f)$ **by** (*rule BsEqvNotDsNot*)
hence *2*: $\vdash \neg(\ bs\ f) = \neg\neg(ds \neg f)$ **by** *auto*
from *2* **show** *?thesis* **by** *auto*
qed

lemma *NotDsEqvBsNot*:
$\vdash \neg\ (ds\ f) = bs \neg f$
proof −
have *1*: $\vdash \neg(ds\ \ f) = \neg\neg(bs\ \neg f)$ **by** (*simp add*: *ds-d-def*)
from *1* **show** *?thesis* **by** *auto*
qed

lemma *NotDsAndEmpty*:
$\vdash \neg(ds\ f \wedge empty)$
proof −
have *1*: $\vdash ds\ f = (more \wedge (di\ f);skip)$ **by** (*rule DsMoreDi*)
have *2*: $\vdash more \wedge (di\ f);skip \wedge empty \longrightarrow \#False$ **using** *NotEmptyEqvMore* **by** *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BsMoreEqvEmpty*:
$\vdash bs\ more = empty$
proof −
have *1*: $\vdash bs\ more = (empty \vee (bi\ more);skip)$ **by** (*simp add*: *bs-d-def*)
have *2*: $\vdash bi\ more \longrightarrow \#False$ **using** *DiEmpty NotEmptyEqvMore* **by** (*simp add*: *bi-d-def empty-d-def*)
hence *3*: $\vdash (bi\ more);skip \longrightarrow \#False;skip$ **using** *LeftChopImpChop* **by** *blast*
have *31*: $\vdash \#False;skip \longrightarrow \#False$ **by** (*simp add*: *Valid-def skip-defs chop-defs*)
have *32*: $\vdash (bi\ more);skip \longrightarrow \#False$ **using** *3 31* **by** *fastforce*
hence *4*: $\vdash (empty \vee ((bi\ more);skip)) = empty$ **by** *fastforce*
from *1 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BsAndEqv*:
$\vdash (bs\ f \wedge bs\ g) = bs(f \wedge g)$
proof −
have *1*: $\vdash bs\ f = (empty \vee (bi\ f)\ ;skip)$
by (*simp add*: *bs-d-def*)
have *2*: $\vdash bs\ g = (empty \vee (bi\ g)\ ;skip)$
by (*simp add*: *bs-d-def*)

have *3*: $\vdash (bs\ f \wedge bs\ g) = ((empty \vee (bi\ f)\ ;skip) \wedge (empty \vee (bi\ g)\ ;skip))$
  using *1 2* **by** *fastforce*
have *4*: $\vdash ((empty \vee (bi\ f)\ ;skip) \wedge (empty \vee (bi\ g)\ ;skip)) =$
    $(empty \vee ((bi\ f)\ ;skip \wedge (bi\ g)\ ;skip))$
  by *auto*
have *5*: $\vdash (((bi\ f)\ ;skip \wedge (bi\ g)\ ;skip)) = bi(f \wedge g);skip$
  using *BiAndChopSkipEqv* **by** *fastforce*
hence *6*: $\vdash (empty \vee ((bi\ f)\ ;skip \wedge (bi\ g)\ ;skip)) = (empty \vee bi(f \wedge g);skip)$
  by *auto*
from *3 4 6* **show** *?thesis* **by** (*metis bs-d-def inteq-reflection*)
qed

lemma *DsEqvRule*:
assumes $\vdash f = g$
shows $\vdash ds\ f = ds\ g$
using *assms* **using** *int-eq* **by** *force*

lemma *DsOrEqv*:
$\vdash (ds\ f \vee ds\ g) = ds\ (f \vee g)$
proof −
have *1*: $\vdash ds\ f = \neg(bs\ \neg\ f)$ **by** (*simp add*: *ds-d-def*)
have *2*: $\vdash ds\ g = \neg(bs\ \neg\ g)$ **by** (*simp add*: *ds-d-def*)
have *3*: $\vdash (ds\ f \vee ds\ g) = (\neg(bs\ \neg\ f) \vee \neg(bs\ \neg\ g))$ **using** *1 2* **by** *fastforce*
have *4*: $\vdash (\neg(bs\ \neg\ f) \vee \neg(bs\ \neg\ g)) = \neg(bs\ \neg\ f \wedge bs\ \neg\ g)$ **by** *auto*
have *5*: $\vdash (bs\ \neg\ f \wedge bs\ \neg\ g) = bs(\ \neg\ f \wedge \neg g)$ **by** (*rule BsAndEqv*)
hence *6*: $\vdash \neg(bs\ \neg\ f \wedge bs\ \neg\ g) = \neg(bs(\ \neg\ f \wedge \neg g))$ **by** *auto*
have *7*: $\vdash \neg(bs(\ \neg\ f \wedge \neg g)) = ds\ (\neg(\neg\ f \wedge \neg g))$ **by** (*rule NotBsEqvDsNot*)
have *8*: $\vdash \neg(\neg\ f \wedge \neg g) = (f \vee g)$ **by** *auto*
hence *9*: $\vdash ds(\neg(\neg\ f \wedge \neg g)) = ds\ (f \vee g)$ **by** (*rule DsEqvRule*)
from *3 4 6 7 9* **show** *?thesis* **by** *fastforce*
qed

lemma *BiOrBiImpBiOr*:
$\vdash bi\ f \vee bi\ g \longrightarrow bi(f \vee g)$
proof −
have *1*: $\vdash f \longrightarrow f \vee g$ **by** *auto*
hence *2*: $\vdash bi\ f \longrightarrow bi(f \vee g)$ **by** (*rule BiImpBiRule*)
have *3*: $\vdash g \longrightarrow f \vee g$ **by** *auto*
hence *4*: $\vdash bi\ g \longrightarrow bi(f \vee g)$ **by** (*rule BiImpBiRule*)
from *2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BsOrImp*:
$\vdash bs\ f \vee bs\ g \longrightarrow bs(f \vee g)$
proof −
have *1*: $\vdash bi\ f \vee bi\ g \longrightarrow bi(f \vee g)$
  by (*rule BiOrBiImpBiOr*)
hence *2*: $\vdash (bi\ f \vee bi\ g);skip \longrightarrow (bi(f \vee g));skip$
  by (*rule LeftChopImpChop*)
have *3*: $\vdash (bi\ f);skip \vee (bi\ g);skip \longrightarrow (bi(f \vee g));skip$

using *1 OrChopEqv 2* **by** *fastforce*
hence *4*: $\vdash$ *empty* $\vee$ (*bi f*);*skip* $\vee$ (*bi g*);*skip* $\longrightarrow$ *empty* $\vee$ (*bi*(*f* $\vee$ *g*));*skip*
by *auto*
hence *5*: $\vdash$ (*empty* $\vee$ (*bi f*);*skip*) $\vee$ (*empty* $\vee$ (*bi g*);*skip*) $\longrightarrow$ *empty* $\vee$ (*bi*(*f* $\vee$ *g*));*skip*
by *auto*
from *5* **show** *?thesis* **by** (*simp add*: *bs-d-def*)
qed

lemma *DsAndImp*:
$\vdash$ *ds* (*f* $\wedge$ *g*) $\longrightarrow$ *ds f* $\wedge$ *ds g*
proof –
have *1*: $\vdash$ *bs* $\neg$*f* $\vee$ *bs* $\neg$*g* $\longrightarrow$ *bs*($\neg$*f* $\vee$ $\neg$*g*) **by** (*rule BsOrImp*)
have *2*: $\vdash$ ($\neg$*f* $\vee$ $\neg$*g*) = $\neg$(*f* $\wedge$ *g*) **by** *auto*
hence *3*: $\vdash$ *bs*($\neg$*f* $\vee$ $\neg$*g*) = *bs* $\neg$(*f* $\wedge$ *g*) **by** (*rule BsEqvRule*)
have *4*: $\vdash$ *bs* $\neg$*f* $\vee$ *bs* $\neg$*g* $\longrightarrow$ *bs* $\neg$(*f* $\wedge$ *g*) **using** *1 3* **by** *fastforce*
have *5*: $\vdash$ *bs* $\neg$*f* = $\neg$(*ds f*) **using** *NotDsEqvBsNot* **by** *fastforce*
have *6*: $\vdash$ *bs* $\neg$*g* = $\neg$(*ds g*) **using** *NotDsEqvBsNot* **by** *fastforce*
have *7*: $\vdash$ *bs* $\neg$(*f* $\wedge$ *g*) = $\neg$(*ds* (*f* $\wedge$ *g*)) **using** *NotDsEqvBsNot* **by** *fastforce*
have *8*: $\vdash$ $\neg$(*ds f*) $\vee$ $\neg$(*ds g*) $\longrightarrow$ $\neg$(*ds* (*f* $\wedge$ *g*)) **using** *4 5 6 7* **by** *fastforce*
hence *9*: $\vdash$ $\neg$(*ds f* $\wedge$ *ds g*) $\longrightarrow$ $\neg$(*ds* (*f* $\wedge$ *g*)) **by** *auto*
from *9* **show** *?thesis* **by** *auto*
qed

lemma *DsAndImpElimL*:
$\vdash$ *ds* (*f* $\wedge$ *g*) $\longrightarrow$ *ds f*
using *DsAndImp* **by** *fastforce*

lemma *DsAndImpElimR*:
$\vdash$ *ds* (*f* $\wedge$ *g*) $\longrightarrow$ *ds g*
using *DsAndImp* **by** *fastforce*

lemma *MoreAndBiImpBiChopSkip*:
$\vdash$ *more* $\wedge$ *bi f* $\longrightarrow$ (*bi f*);*skip*
proof –
have *1*: $\vdash$ (*bi f*);*skip* = $\neg$(*di* $\neg$ *f*);*skip* **by** (*simp add*: *bi-d-def*)
have *2*: $\vdash$ $\neg$($\neg$(*di* $\neg$ *f*);*skip*) = (*empty* $\vee$ (*di* $\neg$ *f*);*skip*) **by** (*rule NotNotChopSkip*)
have *3*: $\vdash$ *empty* $\longrightarrow$ *empty* $\vee$ *di* $\neg$ *f* **by** *auto*
have *4*: $\vdash$ (*di* $\neg$ *f*);*skip* $\longrightarrow$ *di* $\neg$ *f* **using** *ChopImpDi DiEqvDiDi* **by** *fastforce*
hence *5*: $\vdash$ (*di* $\neg$ *f*);*skip* $\longrightarrow$ *empty* $\vee$ *di* $\neg$ *f* **by** (*rule Prop05*)
have *6*: $\vdash$ $\neg$($\neg$(*di* $\neg$ *f*);*skip*) $\longrightarrow$ *empty* $\vee$ *di* $\neg$ *f* **using** *2 3 5* **by** *fastforce*
hence *7*: $\vdash$ $\neg$(*empty* $\vee$ *di* $\neg$ *f*) $\longrightarrow$ $\neg$($\neg$($\neg$(*di* $\neg$ *f*);*skip*)) **by** *fastforce*
have *8*: $\vdash$ $\neg$($\neg$($\neg$(*di* $\neg$ *f*);*skip*)) = $\neg$(*di* $\neg$ *f*);*skip* **by** *auto*
have *9*: $\vdash$ $\neg$(*empty* $\vee$ *di* $\neg$ *f*) = (*more* $\wedge$ $\neg$(*di* $\neg$ *f*))
using *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 *BiImpBs*:
$\vdash$ *bi f* $\longrightarrow$ *bs f*

proof –
 have *1*: ⊢ *empty* ⟶ *empty* ∨ (*bi f*);*skip* **by** *auto*
 hence *2*: ⊢ *empty* ∧ *bi f* ⟶ *empty* ∨ (*bi f*);*skip* **by** *auto*
 have *2*: ⊢ *more* ∧ *bi f* ⟶ (*bi f*);*skip* **by** (*rule MoreAndBiImpBiChopSkip*)
 hence *3*: ⊢ *more* ∧ *bi f* ⟶ *empty* ∨ (*bi f*);*skip* **by** *auto*
 have *4*: ⊢ *bi f* = ((*bi f* ∧ *empty*) ∨ (*bi f* ∧ *more*)) **by** (*simp add*: *empty-d-def*, *auto*)
 have *5*: ⊢ (*empty* ∨ (*bi f*);*skip*) = *bs f* **by** (*simp add*: *bs-d-def*)
 from *2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *BsImpBsBs*:
⊢ *bs f* ⟶ *bs* (*bs f*)
proof –
 have *1*: ⊢ *bi f* ⟶ *bs f* **by** (*rule BiImpBs*)
 hence *2*: ⊢ *bi* (*bi f*) ⟶ *bi*(*bs f*) **by** (*rule BiImpBiRule*)
 hence *3*: ⊢ (*bi f*) ⟶ *bi*(*bs f*) **using** *BiEqvBiBi* **by** *fastforce*
 hence *4*: ⊢ (*bi f*);*skip* ⟶ (*bi*(*bs f*));*skip* **by** (*rule LeftChopImpChop*)
 hence *5*: ⊢ *empty* ∨ (*bi f*);*skip* ⟶ *empty* ∨ (*bi*(*bs f*));*skip* **by** *auto*
 from *5* **show** *?thesis* **by** (*simp add*: *bs-d-def*)
qed

lemma *DsImpDi*:
⊢ *ds f* ⟶ *di f*
proof –
 have *1*: ⊢ *bi* ¬ *f* ⟶ *bs* ¬ *f* **by** (*rule BiImpBs*)
 hence *2*: ⊢ ¬(*bs* ¬ *f*) ⟶ ¬(*bi* ¬ *f*) **by** *fastforce*
 from *2* **show** *?thesis* **using** *NotBsEqvDsNot DiEqvNotBiNot* **by** *fastforce*
qed

lemma *BsImpBsRule*:
 assumes ⊢ *f* ⟶ *g*
 shows ⊢ *bs f* ⟶ *bs g*
proof –
 have *1*: ⊢ *f* ⟶ *g* **using** *assms* **by** *auto*
 hence *2*: ⊢ *bi f* ⟶ *bi g* **by** (*rule BiImpBiRule*)
 hence *3*: ⊢ (*bi f*);*skip* ⟶ (*bi g*);*skip* **by** (*rule LeftChopImpChop*)
 hence *4*: ⊢ *empty* ∨ (*bi f*);*skip* ⟶ *empty* ∨ (*bi g*);*skip* **by** *auto*
 from *4* **show** *?thesis* **by** (*simp add*: *bs-d-def*)
qed

lemma *DiChopImpDiB*:
⊢ *di*(*f*;*g*) ⟶ *di f*
proof –
 have *1*: ⊢ *f* ; (*g*;#*True*) ⟶ *di f* **by** (*rule ChopImpDi*)
 have *2*: ⊢ *f* ; (*g*;#*True*) = (*f*;*g*);#*True* **by** (*rule ChopAssoc*)
 from *1 2* **show** *?thesis* **by** (*metis di-d-def int-eq*)
qed

lemma *DsChopImpDsB*:
⊢ *ds* (*f*;*g*) ⟶ *ds f*

proof −
 have *1*: ⊢ *di*(*f*;*g*) ⟶ *di f* **by** (*rule DiChopImpDiB*)
 hence *2*: ⊢ (*di*(*f*;*g*));*skip* ⟶ (*di f*);*skip* **by** (*rule LeftChopImpChop*)
 from *2* **show** *?thesis* **using** *DsDi* **by** *fastforce*
qed

lemma *NotBsImpBsNotChop*:
⊢ *bs* ¬ *f* ⟶ *bs* (¬(*f*;*g*))
proof −
 have *1*: ⊢ *ds* (*f*;*g*) ⟶ *ds f* **by** (*rule DsChopImpDsB*)
 hence *2*: ⊢ ¬(*ds f*) ⟶ ¬(*ds* (*f*;*g*)) **by** *fastforce*
 from *2* **show** *?thesis* **using** *NotDsEqvBsNot* **by** *fastforce*
qed

lemma *BiBiOrImpBi*:
⊢ *bi* (*bi f* ∨ *bi g*) ⟶ *bi f* ∨ *bi g*
using *BiElim* **by** *auto*

lemma *BiImpBiBiOr*:
⊢ *bi f* ⟶ *bi* (*bi f* ∨ *bi g*)
proof −
 have *1*: ⊢ *bi f* ⟶ *bi f* ∨ *bi g* **by** *auto*
 hence *2*: ⊢ *bi* (*bi f*) ⟶ *bi*(*bi f* ∨ *bi g*) **using** *BiImpBiRule* **by** *blast*
 have *3*: ⊢ *bi* (*bi f*) = *bi f* **using** *BiEqvBiBi* **by** *fastforce*
 from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BiImpBiBiOrB*:
⊢ *bi g* ⟶ *bi* (*bi f* ∨ *bi g*)
proof −
 have *1*: ⊢ *bi g* ⟶ *bi f* ∨ *bi g* **by** *auto*
 hence *2*: ⊢ *bi* (*bi g*) ⟶ *bi*(*bi f* ∨ *bi g*) **using** *BiImpBiRule* **by** *blast*
 have *3*: ⊢ *bi* (*bi g*) = *bi g* **using** *BiEqvBiBi* **by** *fastforce*
 from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BiBiOrEqvBi*:
⊢ *bi* (*bi f* ∨ *bi g*) = *bi f* ∨ *bi g*
proof −
 have *1*: ⊢ *bi* (*bi f* ∨ *bi g*) ⟶ *bi f* ∨ *bi g* **by** (*rule BiBiOrImpBi*)
 have *2*: ⊢ *bi f* ⟶ *bi* (*bi f* ∨ *bi g*) **by** (*rule BiImpBiBiOr*)
 have *3*: ⊢ *bi g* ⟶ *bi* (*bi f* ∨ *bi g*) **by** (*rule BiImpBiBiOrB*)
 have *4*: ⊢ *bi f* ∨ *bi g* ⟶ *bi* (*bi f* ∨ *bi g*) **using** *2 3* **by** *fastforce*
 from *1 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BsOrBsEqvBsBiOrBi*:
⊢ (*bs f* ∨ *bs g*) = *bs*(*bi f* ∨ *bi g*)
proof −

have *1*: ⊢ (*bs f* ∨ *bs g*) = ((*empty* ∨ (*bi f*);*skip*) ∨ (*empty* ∨ (*bi g*);*skip*))
by (*simp add*: *bs-d-def*)
have *2*: ⊢ ((*empty* ∨ (*bi f*);*skip*) ∨ (*empty* ∨ (*bi g*);*skip*)) = (*empty* ∨ (*bi f*);*skip* ∨ (*bi g*);*skip*)
by *auto*
have *3*: ⊢ ((*bi f*);*skip* ∨ (*bi g*);*skip*) = (*bi f* ∨ *bi g*);*skip*
using *OrChopEqv* **by** *fastforce*
hence *4*: ⊢ (*empty* ∨ (*bi f*);*skip* ∨ (*bi g*);*skip*) = (*empty* ∨ (*bi f* ∨ *bi g*);*skip*)
by *auto*
have *5*: ⊢ (*bi f* ∨ *bi g*) = *bi* (*bi f* ∨ *bi g*)
by (*simp add*: *BiBiOrImpBi BiImpBiBiOr BiImpBiBiOrB int-iffI Prop02*)
hence *6*: ⊢ (*bi f* ∨ *bi g*);*skip* = *bi* (*bi f* ∨ *bi g*);*skip*
by (*simp add*: *LeftChopEqvChop*)
hence *7*: ⊢ (*empty* ∨ *bi* (*bi f* ∨ *bi g*);*skip*) = (*empty* ∨ (*bi f* ∨ *bi g*);*skip*)
by *auto*
have *8*: ⊢ (*empty* ∨ (*bi f* ∨ *bi g*);*skip*) = *bs*(*bi f* ∨ *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*:
⊢ *di f* ∨ *ds f* = *di f*
proof −
have *1*: ⊢ *di f* ⟶ *di f* ∨ *ds f* **by** *auto*
have *2*: ⊢ *di f* ⟶ *di f* **by** *auto*
have *3*: ⊢ *ds f* ⟶ *di f* **by** (*rule DsImpDi*)
have *4*: ⊢ *di f* ∨ *ds f* ⟶ *di f* **using** *2 3* **by** *auto*
from *1 4* **show** *?thesis* **by** *auto*
qed

lemma *DiAndDsEqvDs*:
⊢ (*di f* ∧ *ds f*) = *ds f*
proof −
have *1*: ⊢ *di f* ∧ *ds f* ⟶ *ds f* **by** *auto*
have *2*: ⊢ *ds f* ⟶ *ds f* **by** *auto*
have *3*: ⊢ *ds f* ⟶ *di f* **by** (*rule DsImpDi*)
have *4*: ⊢ *ds f* ⟶ *di f* ∧ *ds f* **using** *2 3* **by** *auto*
from *1 4* **show** *?thesis* **by** *auto*
qed

lemma *DiEqvOrDiChopSkipA*:
⊢ *di f* = (*f* ∨ *di*(*f*;*skip*))
proof −
have *1*: ⊢ *di f* = *f* ;#*True* **by** (*simp add*: *di-d-def*)
hence *2*: ⊢ *di f* = *f*; (*empty* ∨ *more*) **by** (*simp add*: *empty-d-def*)
hence *3*: ⊢*f*; (*empty* ∨ *more*) = (*f*;*empty* ∨ *f*;*more*) **using** *ChopOrEqv* **by** *blast*
have *4*: ⊢ *f*;*empty* = *f* **by** (*rule ChopEmpty*)
have *5*: ⊢ *more* = *skip*;#*True* **using** *MoreEqvSkipChopTrue* **by** *blast*
hence *6*: ⊢ *f*;*more* = *f*;(*skip*;#*True*) **using** *RightChopEqvChop* **by** *blast*
have *7*: ⊢ *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 *SkipTrueEqvTrueSkip*:
$\vdash$ *skip*;#*True* = #*True*;*skip*
using *TrueChopSkipEqvSkipChopTrue* **by** *fastforce*

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 *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 *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 *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 *StateImpBs*:
$\vdash init\ w \longrightarrow bs\ (init\ w)$
proof –
have *1*: $\vdash init\ w = bi\ (init\ w)$ **by** (*rule StateEqvBi*)
have *2*: $\vdash bi\ (init\ w) \longrightarrow bs\ (init\ w)$ **by** (*rule BiImpBs*)
from *1 2* **show** *?thesis* **using** *StateImpBi* **by** *fastforce*
qed

lemma *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 (*simp add*: *BiBiOrImpBi BiImpBiBiOr BiImpBiBiOrB int-iffI Prop02*)
have *2*: $\vdash bi\ \neg f = \neg\ (di\ f)$
    by (*simp add*: *bi-d-def*)
have *3*: $\vdash bi\ \neg g = \neg\ (di\ g)$
    by (*simp add*: *bi-d-def*)
have *4*: $\vdash (bi\ \neg f \vee bi\ \neg\ g) = (\neg\ (di\ f) \vee \neg\ (di\ g))$
    using *2 3* **by** *fastforce*
have *5*: $\vdash (\neg\ (di\ f) \vee \neg\ (di\ g)) = \neg(di\ f \wedge di\ g)$
    by *auto*
have *6*: $\vdash bi\ (\ bi\ \neg\ f \vee bi\ \neg\ g) = \neg(di\ f \wedge di\ g)$
    using *1 5 4* **by** *fastforce*
hence *7*: $\vdash \neg(bi\ (\ bi\ \neg\ f \vee bi\ \neg\ g))= (di\ f \wedge di\ g)$
    by *auto*
have *8* : $\vdash \neg(bi\ (\ bi\ \neg\ f \vee bi\ \neg\ g)) = di\ (\ \neg(bi\ \neg\ f \vee bi\ \neg\ g))$
    using *DiNotEqvNotBi* **by** *fastforce*
have *9* : $\vdash \neg(bi\ \neg\ f \vee bi\ \neg\ g) = (di\ f \wedge di\ g)$
    using *1 7* **by** *fastforce*
hence *10*: $\vdash di\ (\ \neg(bi\ \neg\ f \vee bi\ \neg\ g)) = di\ (\ di\ f \wedge di\ g)$
    using *DiEqvDi* **by** *blast*
from *7 8 10* **show** *?thesis* **by** *fastforce*
qed

lemma *BiInduct*:
$\vdash bi(f \longrightarrow wprev\ f) \wedge f \longrightarrow bi\ f$
proof –

have *1*: $\vdash \Box((f^r) \longrightarrow wnext(f^r)) \wedge f^r \longrightarrow \Box(f^r)$ **using** *BoxInduct* **by** *blast*
hence *2*: $\vdash (\Box((f^r) \longrightarrow wnext(f^r)) \wedge f^r \longrightarrow \Box(f^r))^r$ **using** *ReverseEqv* **by** *blast*
have *3*: $\vdash ((f^r)^r\ ) = f$ **by** (*simp add*: *EqvReverseReverse*)
have *4*: $\vdash (\Box\ (f^r))^r = bi\ (f)$ **using** *RRBoxEqvBi* **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** *RRWNextEqvWPrev* **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*: $\vdash (\Box((f^r) \longrightarrow wnext(f^r)))^r = bi\ (\ ((f^r) \longrightarrow wnext(f^r))^r\ )$ **using** *RBoxEqvBi* **by** *blast*
have *10*: $\vdash (\Box((f^r) \longrightarrow wnext(f^r)))^r = bi(\ f \longrightarrow wprev(f)\ )$ **using** *8 9 5 int-eq* **by** *fastforce*
have *11*: $\vdash (\Box((f^r) \longrightarrow wnext(f^r)) \wedge f^r \longrightarrow \Box(f^r))^r =$
$(((\Box((f^r) \longrightarrow wnext(f^r)))^r \wedge (f^r)^r \longrightarrow (\Box(f^r))^r))$ **by** (*metis int-eq rev-fun2*)
have *12*: $\vdash ((\Box((f^r) \longrightarrow wnext(f^r)))^r \wedge (f^r)^r \longrightarrow (\Box(f^r))^r) =$
$(bi(\ f \longrightarrow wprev(f)\ ) \wedge f \longrightarrow bi\ f\ )$ **using** *8 3 4 10* **by** *fastforce*
from *2 11 12* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *PrevLoop*:
assumes $\vdash f \longrightarrow prev\ f$
shows $\vdash \neg\ f$
proof −
have *1*: $\vdash f \longrightarrow prev\ f$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \longrightarrow (\ more \wedge wprev\ f)$
by (*smt intI int-eq more-defs prev-defs Prop10 unl-lift2 wprev-defs*)
hence *3*: $\vdash f \longrightarrow wprev\ f$ **by** *auto*
hence *4*: $\vdash bi(f \longrightarrow wprev\ f)$ **by** (*rule BiGen*)
have *5*: $\vdash bi(f \longrightarrow wprev\ f) \wedge f \longrightarrow bi\ f$ **by** (*rule BiInduct*)
hence *6*: $\vdash bi(f \longrightarrow wprev\ f) \longrightarrow (f \longrightarrow bi\ f)$ **by** *fastforce*
have *7*: $\vdash (f \longrightarrow bi\ f)$ **using** *4 6 MP* **by** *blast*
have *8*: $\vdash bi\ f \longrightarrow f$ **by** (*rule BiElim*)
have *9*: $\vdash f = bi\ f$ **using** *7 8* **by** *fastforce*
have *10*: $\vdash f \longrightarrow more$ **using** *2* **by** *auto*
hence *11*: $\vdash bi\ f \longrightarrow\ bi\ more$ **using** *BiImpBiRule* **by** *blast*
have *12*: $\vdash \neg(bi\ more)$ **using** *DiEmpty bi-d-def empty-d-def* **by** (*simp add*: *bi-d-def empty-d-def*)
from *7 9 11 12* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *PrevImpNotPrevNot*:
$\vdash prev\ f \longrightarrow \neg(prev\ \neg\ f)$
by (*metis* (*no-types*, *lifting*) *NextImpNotNextNot RPrevEqvNext ReverseEqv inteq-reflection rev-fun1 rev-fun2*)

lemma *BiEqvAndWprevBi*:
$\vdash bi\ f = (f \wedge wprev(bi\ f))$
using *BoxEqvAndWnextBox*
by (*metis* (*no-types*, *lifting*) *RBiEqvBox RRAnd RRBoxEqvBi RWPrevEqvWNext int-eq*)

lemma *DiIntroLoop*:
assumes $\vdash (f \wedge \neg\ g) \longrightarrow prev\ f$

shows $\vdash f \longrightarrow di\ g$
using *assms DiamondIntro*
by (*metis* (*no-types, lifting*) *RDiEqvDiamond RPrevEqvNext ReverseEqv inteq-reflection rev-fun2 rev-fun1*)

lemma *DiEqvOrChopMore*:
$\vdash di\ f = (f \lor f;more)$
proof −
have *1*: $\vdash di\ f = f;\#True$ **by** (*simp add*: *di-d-def*)
hence *2*: $\vdash di\ f = f;\ (empty \lor more)$ **by** (*simp add*: *empty-d-def*)
have *3*: $\vdash f;\ (empty \lor more) = (f;empty \lor f;more)$ **by** (*simp add*: *ChopOrEqv*)
have *4*: $\vdash f;empty = f$ **by** (*rule ChopEmpty*)
from *2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *DiAndDiEqvDiAndDiOrDiAndDi*:
$\vdash (di\ f \land di\ g) = (di(f \land di\ g) \lor di(g \land di\ f))$
proof −
have *1*: $\vdash di\ f = (f \lor f;more)$
using *DiEqvOrChopMore* **by** *blast*
have *2*: $\vdash di\ g = (g \lor g;more)$
using *DiEqvOrChopMore* **by** *blast*
have *3*: $\vdash (di\ f \land di\ g) = ((f \lor f;more) \land (g \lor g;more))$
using *1 2* **by** *fastforce*
have *4*: $\vdash ((f \lor f;more) \land (g \lor g;more)) =$
$((f \land g) \lor (\ f \land g;more) \lor (g \land f;more) \lor (f;more \land g;more\ ))$
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*: $\vdash (f;more \land g;more) = (prev\ (di\ f) \land prev\ (di\ g))$
using *8 11* **by** *fastforce*
hence *13*: $\vdash (f;more \land g;more) = prev\ (di\ f \land di\ g)$
by (*metis ChopSkipAndChopSkip int-eq prev-d-def*)
have *14*: $\vdash (di\ f \land di\ g) =$
$((f \land g) \lor (\ f \land g;more) \lor (g \land f;more)) \lor (f;more \land g;more)$
using *3 4* **by** *auto*
have *15*: $\vdash (di\ f \land di\ g) =$

$((f \wedge g) \vee (f \wedge g;more) \vee (g \wedge f;more)) \vee prev\ (di\ f \wedge di\ g)$
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 *DiIntroLoop* **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 *DiIntro* **by** *blast*
hence *21*: $\vdash f \wedge g \longrightarrow g \wedge di\ f$
by *auto*
hence *22*: $\vdash di(f \wedge g) \longrightarrow di(g \wedge di\ f)$
using *DiImpDi* **by** *blast*
hence *23*: $\vdash di(f \wedge g) \longrightarrow di(g \wedge di\ f) \vee di(f \wedge di\ g)$
by *auto*
have *24*: $\vdash g;more \longrightarrow di\ g$
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 *DiImpDi* **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 *DiImpDi* **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 *DiIntro* **by** *blast*
hence *36*: $\vdash f \wedge di\ g \longrightarrow di\ f \wedge di\ g$
by *auto*
hence *37*: $\vdash di\ (f \wedge di\ g) \longrightarrow di\ (di\ f \wedge di\ g)$

using *DiImpDi* **by** *blast*
have *38*: $\vdash di\ (di\ f \wedge di\ g) = (di\ f \wedge di\ g)$
using *DiDiAndEqvDi* **by** *blast*
have *39*: $\vdash di\ (f \wedge di\ g) \longrightarrow di\ f \wedge di\ g$
using *37 38* **by** *fastforce*
have *40*: $\vdash\ g \longrightarrow di\ g$
using *DiIntro* **by** *blast*
hence *41*: $\vdash\ g \wedge di\ f \longrightarrow di\ f \wedge di\ g$
by *auto*
hence *42*: $\vdash di\ (g \wedge di\ f) \longrightarrow di\ (di\ f \wedge di\ g)$
using *DiImpDi* **by** *blast*
have *43*: $\vdash di\ (di\ f \wedge di\ g) = (di\ f \wedge di\ g)$
using *DiDiAndEqvDi* **by** *fastforce*
have *44*: $\vdash di\ (g \wedge di\ f) \longrightarrow di\ f \wedge di\ g$
using *42 43* **by** *fastforce*
have *45*: $\vdash di\ (f \wedge di\ g) \vee di\ (g \wedge di\ f) \longrightarrow di\ f \wedge di\ g$
using *39 44* **by** *fastforce*
from *34 45* **show** *?thesis* **by** *fastforce*
qed

lemma *DsAndDsEqvDsAndDiOrDsAndDi*:
$\vdash (ds\ f \wedge ds\ g) =(\ ds\ (f \wedge di\ g) \vee ds(g \wedge di\ f))$
proof −
have *1*: $\vdash (di\ f \wedge di\ g) = (di(f \wedge di\ g) \vee di(g \wedge di\ f))$
by (*rule DiAndDiEqvDiAndDiOrDiAndDi*)
hence *2*: $\vdash (di\ f \wedge di\ g);skip = (di(f \wedge di\ g) \vee di(g \wedge di\ f));skip$
by (*rule LeftChopEqvChop*)
have *3*: $\vdash (di\ f \wedge di\ g);skip = ((di\ f);skip \wedge (di\ g);skip)$
using *ChopSkipAndChopSkip* **by** *fastforce*
have *4*: $\vdash ((di\ f);skip \wedge (di\ g);skip) = (di(f \wedge di\ g) \vee di(g \wedge di\ f));skip$
using *2 3* **by** *fastforce*
have *5*: $\vdash (di(f \wedge di\ g) \vee di(g \wedge di\ f));skip = (di(f \wedge di\ g);skip \vee di(g \wedge di\ f);skip)$
using *OrChopEqv* **by** *blast*
have *6*: $\vdash ds\ f = (di\ f);skip$
using *DsDi* **by** *blast*
have *7*: $\vdash ds\ g = (di\ g);skip$
using *DsDi* **by** *blast*
have *8*: $\vdash\ ((di\ f);skip \wedge (di\ g);skip) = (ds\ f \wedge\ ds\ g)$
using *6 7* **by** *fastforce*
have *9*: $\vdash ds(f \wedge di\ g) =\ di(f \wedge di\ g);skip$
using *DsDi* **by** *blast*
have *10*: $\vdash ds(g \wedge di\ f) = di(g \wedge di\ f);skip$
using *DsDi* **by** *blast*
have *11*: $\vdash (di(f \wedge di\ g);skip \vee di(g \wedge di\ f);skip) = (ds(f \wedge di\ g) \vee ds(g \wedge di\ f))$
using *9 10* **by** *fastforce*
from *4 5 8 11* **show** *?thesis* **by** *fastforce*
qed

lemma *BsEqvBiMoreImpChop*:
$\vdash bs\ f = bi(more \longrightarrow f;skip)$

proof −
have *1*: $\vdash$ *bs f* = (*empty* ∨ (*bi f*;*skip*))
  by (*simp add*: *bs-d-def*)
have *2*: $\vdash$ (*empty* ∨ (*bi f*;*skip*)) = ¬(¬(*bi f*);*skip*)
  using *NotNotChopSkip* **by** *fastforce*
have *3*: $\vdash$ ¬(¬(*bi f*);*skip*) = ¬(*di* ¬ *f*;*skip*)
  by (*simp add*: *bi-d-def*)
have *4*: $\vdash$ ¬(*di* ¬ *f*;*skip*) = ¬((¬ *f* ;#*True*);*skip*)
  by (*simp add*:*di-d-def*)
have *5*: $\vdash$ ¬((¬ *f* ;#*True*);*skip*) = ¬(¬ *f* ;(#*True*;*skip*))
  using *ChopAssocB* **by** *fastforce*
have *6*: $\vdash$ ¬(¬ *f* ;(#*True*;*skip*)) = ¬(¬ *f* ;(*skip*;#*True*))
  using *SkipTrueEqvTrueSkip TrueChopSkipEqvSkipChopTrue RightChopEqvChop* **by** *fastforce*
have *7*: $\vdash$ ¬(¬ *f* ;(*skip*;#*True*)) = ¬((¬ *f* ;*skip*);#*True*)
  using *ChopAssoc* **by** *fastforce*
have *8*: $\vdash$ ¬((¬ *f* ;*skip*);#*True*) = ¬(*di* (¬ *f*;*skip*))
  by (*simp add*: *di-d-def*)
have *9*: $\vdash$ ¬(*di* (¬ *f*;*skip*)) = *bi* (¬(¬ *f* ;*skip*))
  using *NotDiEqvBiNot* **by** *blast*
have *10*: $\vdash$ *bi* (¬(¬ *f* ;*skip*)) = *bi*(*empty* ∨ (*f*;*skip*))
  using *NotNotChopSkip* **using** *BiEqvBi* **by** *blast*
have *11*: $\vdash$ *bi*(*empty* ∨ (*f*;*skip*)) = *bi*(¬ *more* ∨ (*f*;*skip*))
  by (*metis NotEmptyEqvMore int-eq intensional-simps*(*1*) *intensional-simps*(*4*))
have *12*: $\vdash$ (¬ *more* ∨ (*f*;*skip*)) = (*more* ⟶ *f*;*skip*) **by** *auto*
have *13*: $\vdash$ *bi*(¬ *more* ∨ (*f*;*skip*)) = *bi*(*more* ⟶ *f*;*skip*) **using** *12* **using** *BiEqvBi* **by** *blast*
have *14*: $\vdash$ *bs f* = ¬ ((¬ *f*;*skip*);#*True*) **using** *1 2 3 4 5 6 7* **by** *fastforce*
have *15*: $\vdash$ ¬ ((¬ *f*;*skip*);#*True*) = *bi*(*more* ⟶ *f*;*skip*) **using** *8 9 10 11 13* **by** *fastforce*
from *14 15* **show** *?thesis* **by** *fastforce*
qed

7.4.3 First occurrence

lemma *FstWithAndImp*:
$\vdash$ ▷*f* ∧ *g* ⟶ ▷ (*f* ∧ *g*)
proof −
have *1*: $\vdash$ (▷*f* ∧ *g*) = ((*f* ∧ (*bs* ¬*f*)) ∧ *g*)
  by (*simp add*: *first-d-def*)
have *2*: $\vdash$ ((*f* ∧ (*bs* ¬*f*)) ∧ *g*) = (*f* ∧ ¬(*ds f*) ∧ *g*)
  using *NotDsEqvBsNot* **by** *fastforce*
have *3*: $\vdash$ ¬(*ds f*) ⟶ ¬(*ds*(*f* ∧ *g*))
  using *DsAndImpElimL* **by** *fastforce*
hence *4*: $\vdash$ *f* ∧ ¬(*ds f*) ∧ *g* ⟶ *f* ∧ *g* ∧ ¬(*ds*(*f* ∧ *g*))
  by *auto*
have *5*: $\vdash$ (*f* ∧ *g* ∧ ¬(*ds*(*f* ∧ *g*))) = ((*f* ∧ *g*) ∧ (*bs* ¬(*f* ∧ *g*)))
  using *NotDsEqvBsNot* **by** *fastforce*
have *6*: $\vdash$ ((*f* ∧ *g*) ∧ (*bs* ¬(*f* ∧ *g*))) = ▷(*f* ∧ *g*)
  by (*simp add*: *first-d-def*)
from *1 2 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *FstWithOrEqv*:
$\vdash \rhd(f \vee g) = ((\rhd f \wedge bs\ \neg g) \vee (\rhd g \wedge bs\ \neg f))$
proof –
 have *1*: $\vdash \rhd(f \vee g) = ((f \vee g) \wedge bs\ \neg(f \vee g))$
   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*: $\vdash ((f \wedge bs\ \neg\ f) \wedge bs\ \neg\ g) = (\rhd f \wedge\ bs\ \neg\ g)$
   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*: $\vdash ((g \wedge bs\ \neg\ g) \wedge bs\ \neg\ f) = (\rhd g \wedge\ bs\ \neg\ f)$
   by (*simp add*: *first-d-def*)
 have *10*: $\vdash ((f \wedge bs\ \neg\ f) \wedge bs\ \neg\ g) \vee (g \wedge bs\ \neg\ f \wedge bs\ \neg\ g) =$
     $(\rhd f \wedge\ bs\ \neg\ g) \vee (\rhd g \wedge\ bs\ \neg\ f)$
   using *7 8 9* **by** *fastforce*
 from *1 5 6 10* **show** *?thesis* **by** (*metis 7 8 9 int-eq*)
qed

lemma *FstFstAndEqvFstAnd*:
$\vdash \rhd(\rhd f \wedge g) = (\rhd f \wedge g)$
proof –
  have *1*: $\vdash (\rhd f \wedge g) = ((f \wedge (bs\ \neg f)) \wedge g)$ **by** (*simp add*: *first-d-def*)
  hence *2*: $\vdash \rhd f \wedge g \longrightarrow (bs\ \neg f)$ **by** *auto*
  hence *3*: $\vdash \rhd f \wedge g \longrightarrow \rhd f \wedge g \wedge (bs\ \neg f)$ **by** *auto*
  have *4*: $\vdash \neg\ f \longrightarrow \neg\ f \vee \neg(bs\ \neg f) \vee \neg\ g$ **by** *auto*
  hence *5*: $\vdash bs\ (\neg\ f) \longrightarrow bs(\neg\ f \vee \neg(bs\ \neg f) \vee \neg\ g)$ **using** *BsImpBsRule* **by** *blast*
  have *6*: $\vdash (\neg\ f \vee \neg(bs\ \neg f) \vee \neg\ g) = \neg(f \wedge bs\ \neg\ f \wedge g)$ **by** *auto*
  hence *7*: $\vdash bs(\neg\ f \vee \neg(bs\ \neg f) \vee \neg\ g) = bs(\neg(f \wedge bs\ \neg\ f \wedge g))$ **using** *BsEqvRule* **by** *blast*
  have *8*: $\vdash ((f \wedge bs\ \neg\ f) \wedge g) = (\rhd f \wedge g)$ **by** (*simp add*: *first-d-def*)
  hence *9*: $\vdash \neg(f \wedge bs\ \neg\ f \wedge g) = \neg(\rhd f \wedge g)$ **by** *auto*
  hence *10*: $\vdash bs\ \neg(f \wedge bs\ \neg\ f \wedge g) = bs\ \neg(\rhd f \wedge g)$ **using** *BsEqvRule* **by** *blast*
  have *11*: $\vdash \rhd f \wedge g \longrightarrow (\rhd f \wedge g) \wedge\ bs\ \neg(\rhd f \wedge g)$ **using** *3 5 7 10* **by** *fastforce*
  hence *12*: $\vdash \rhd f \wedge g \longrightarrow \rhd(\rhd f \wedge g)$ **by** (*simp add*: *first-d-def*)
  have *13*: $\vdash \rhd(\rhd f \wedge g) = ((\rhd f \wedge g) \wedge\ bs\ \neg(\rhd f \wedge g))$ **by** (*simp add*: *first-d-def*)
  hence *14*: $\vdash \rhd(\rhd f \wedge g) \longrightarrow \rhd f \wedge g$ **by** *auto*
 from *12 14* **show** *?thesis* **by** *fastforce*
qed

lemma *FstTrue*:
$\vdash \rhd \#True = empty$

proof −
 have *1*: $\vdash \triangleright \# True = (\# True \wedge bs \neg \# True)$
  by (*simp add*: *first-d-def*)
 have *2*: $\vdash bs \neg \# True = (empty \vee (bi \neg \# True);skip)$
  by (*simp add*: *bs-d-def*)
 have *3*: $\vdash \neg(bi \neg \# True)$
  using *BiElim* **by** *fastforce*
 hence *4*: $\vdash \neg((bi \neg \# True);skip)$
  by (*metis BiEqvAndWprevBi MoreEqvSkipChopTrue NotChopSkipEqvMoreAndNotChopSkip SkipTrueEqvTrueSkip int-eq intensional-simps*(*19*) *intensional-simps*(*2*) *intensional-simps*(*21*))
 have *5*: $\vdash bs \neg \# True = empty$
  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*:
$\vdash \triangleright empty = empty$
proof −
 have *1*: $\vdash \triangleright empty = (empty \wedge bs \neg empty)$ **by** (*simp add*: *first-d-def*)
 have *2*: $\vdash bs \neg empty = (empty \vee bi \neg empty;skip)$ **by** (*simp add*: *bs-d-def*)
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndEmptyEqvAndEmpty*:
$\vdash (\triangleright f \wedge empty) = (f \wedge empty)$
proof −
 have *1*: $\vdash (\triangleright f \wedge empty) = ((f \wedge bs \neg f) \wedge empty)$ **by** (*simp add*: *first-d-def*)
 have *2*: $\vdash bs \neg f = (empty \vee bi \neg f;skip)$ **by** (*simp add*: *bs-d-def*)
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FstEmptyOrEqvEmpty*:
$\vdash \triangleright(empty \vee f) = empty$
proof −
 have *1*: $\vdash \triangleright(empty \vee f) = ((\triangleright empty \wedge bs \neg f) \vee (\triangleright f \wedge bs \neg empty))$ **using** *FstWithOrEqv* **by** *blast*
 have *2*: $\vdash \neg empty = more$ **by** (*simp add*: *empty-d-def*)
 hence *3*: $\vdash bs \neg empty = bs\ more$ **using** *BsEqvRule* **by** *blast*
 have *4*: $\vdash bs\ more = empty$ **using** *BsMoreEqvEmpty* **by** *blast*

have *5*: $\vdash$ ($\triangleright f$ $\wedge$ *bs* $\neg$ *empty*) = ($\triangleright f$ $\wedge$ *empty*) **using** *3 4* **by** *fastforce*
have *6*: $\vdash$ $\triangleright$*empty* = *empty* **using** *FstEmpty* **by** *blast*
hence *7*: $\vdash$ ($\triangleright$*empty* $\wedge$ *bs* $\neg$ *f*) = (*empty* $\wedge$ *bs* $\neg$ *f*) **by** *auto*
have *8*: $\vdash$ (*empty* $\wedge$ *bs* $\neg$ *f*) = (*empty* $\wedge$(*empty* $\vee$ *bi* $\neg$ *f*;*skip*)) **by** (*simp add*:*bs-d-def*)
have *9*: $\vdash$ (*empty* $\wedge$(*empty* $\vee$ *bi* $\neg$ *f*;*skip*)) = *empty* **by** *auto*
have *10*: $\vdash$ (*empty* $\wedge$ *bs* $\neg$ *f*) = *empty* **using** *8 9* **by** *auto*
have *11*: $\vdash$ (($\triangleright$*empty* $\wedge$ *bs* $\neg$ *f*) $\vee$ ($\triangleright f$ $\wedge$ *bs* $\neg$ *empty*)) =
(*empty* $\vee$ ($\triangleright f$ $\wedge$ *empty*)) **using** *7 10 5* **by** *fastforce*
have *12*: $\vdash$ (*empty* $\vee$ ($\triangleright f$ $\wedge$ *empty*)) = *empty* **by** *auto*
from *1 11 12* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopEmptyAndEmpty*:
$\vdash$ (*f*;*g* $\wedge$ *empty*) = (*f* $\wedge$ *g* $\wedge$ *empty*)
apply (*simp add*: *Valid-def chop-defs empty-defs*)
by (*metis interval-prefix-intlen interval-suffix-zero le-zero-eq*)

lemma *FstChopEmptyEqvFstChopFstEmpty*:
$\vdash$ ($\triangleright f$;*g* $\wedge$ *empty*) = ($\triangleright f$;$\triangleright g$ $\wedge$ *empty*)
proof −
have *1*: $\vdash$ ($\triangleright f$;*g* $\wedge$ *empty*) = ($\triangleright f$ $\wedge$ *g* $\wedge$ *empty*) **using** *ChopEmptyAndEmpty* **by** *blast*
have *2*: $\vdash$ ($\triangleright g$ $\wedge$ *empty*) = (*g* $\wedge$ *empty*) **using** *FstAndEmptyEqvAndEmpty* **by** *blast*
hence *3*: $\vdash$ ($\triangleright f$ $\wedge$ *g* $\wedge$ *empty*) = ($\triangleright f$ $\wedge$ $\triangleright g$ $\wedge$ *empty*) **by** *auto*
have *4*: $\vdash$ ($\triangleright f$;$\triangleright g$ $\wedge$ *empty*) = ($\triangleright f$ $\wedge$ $\triangleright g$ $\wedge$ *empty*) **using** *ChopEmptyAndEmpty* **by** *blast*
from *1 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopSkipImpMore*:
$\vdash$ *f*;*skip* $\longrightarrow$ *more*
using *ChopImpDiamond MoreEqvSkipChopTrue SkipTrueEqvTrueSkip TrueChopEqvDiamond* **by** *fastforce*

lemma *MoreEqvMoreChopTrue*:
$\vdash$ *more* = *more*;#*True*
proof −
have *1*: $\vdash$ *more* = *skip*;#*True*
using *MoreEqvSkipChopTrue* **by** *blast*
have *2*: $\vdash$ #*True* = #*True*;#*True*
by (*simp add*: *Valid-def chop-defs*, *auto*)
hence *3*: $\vdash$ *skip*;#*True* = *skip*;(#*True*;#*True*)
using *RightChopEqvChop* **by** *blast*
have *4*: $\vdash$ *skip*;(#*True*;#*True*) = (*skip*;#*True*);#*True*
using *ChopAssoc* **by** *blast*
have *5*: $\vdash$ (*skip*;#*True*);#*True* = *more*;#*True*
using *MoreEqvSkipChopTrue* **by** (*simp add*: *more-d-def next-d-def*)
from *1 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *BiEmptyEqvEmpty*:
$\vdash$ *bi empty* = *empty*

proof −
 have *1*: $\vdash$ *bi empty* = ¬ (*di* ¬ *empty*) **by** (*simp add*: *bi-d-def*)
 have *2*: $\vdash$ ¬ (*di* ¬ *empty*) = ¬ (¬ *empty*;#*True*) **by** (*simp add*: *di-d-def*)
 have *3*: $\vdash$ ¬ (¬ *empty*;#*True*) = ¬ (*more*;#*True*) **by** (*simp add*: *empty-d-def*)
 have *4*: $\vdash$ *more*;#*True* = *more* **using** *MoreEqvMoreChopTrue* **by** *auto*
 hence *5*: $\vdash$ ¬(*more*;#*True*) = ¬ *more* **by** *fastforce*
 from *1 2 3 5* **show** *?thesis* **using** *NotEmptyEqvMore* **by** *fastforce*
qed

lemma *FstMoreEqvSkip*:
$\vdash$ ▷ *more* = *skip*
proof −
 have *1*: $\vdash$ ▷ *more* = (*more* ∧ *bs* ¬ *more*) **by** (*simp add*: *first-d-def*)
 have *2*: $\vdash$ (*more* ∧ *bs* ¬ *more*) = (*more* ∧ (*empty* ∨ *bi* ¬ *more*;*skip*)) **by** (*simp add*:*bs-d-def*)
 have *3*: $\vdash$ (*more* ∧ (*empty* ∨ *bi* ¬ *more*;*skip*)) = (*more* ∧ *bi* ¬ *more*;*skip*) **using** *empty-d-def*
 using *MoreAndEmptyOrEqvMoreAnd* **by** *fastforce*
 have *4*: $\vdash$ (*more* ∧ ((*bi* ¬ *more*);*skip*)) = ((*bi* ¬ *more*);*skip*) **using** *ChopSkipImpMore* **by** *fastforce*
 have *5*: $\vdash$ ((*bi* ¬ *more*);*skip*) = *bi empty*;*skip* **by** (*simp add*: *empty-d-def*)
 have *6*: $\vdash$ *bi empty* = *empty* **using** *BiEmptyEqvEmpty* **by** *auto*
 hence *7*: $\vdash$ *bi empty*;*skip* = *empty*;*skip* **using** *LeftChopEqvChop* **by** *blast*
 have *8*: $\vdash$ *empty*;*skip* = *skip* **using** *EmptyChop* **by** *blast*
 from *1 2 3 4 5 7 8* **show** *?thesis* **by** (*metis int-eq*)
qed

lemma *FstEqvBsNotAndDi*:
$\vdash$ ▷*f* = (*bs* ¬ *f* ∧ *di f*)
proof −
 have *1*: $\vdash$ *bs* ¬ *f* = ¬(*ds f*) **by** (*simp add*: *ds-d-def*)
 hence *2*: $\vdash$ (*bs* ¬ *f* ∧ *di f*) = (¬(*ds f*) ∧ *di f*) **by** *auto*
 have *3*: $\vdash$ *di f* = (*ds f* ∨ *f*) **using** *OrDsEqvDi* **by** *fastforce*
 hence *4* : $\vdash$ (¬(*ds f*) ∧ *di f*) = (¬(*ds f*) ∧ (*ds f* ∨ *f*)) **by** *auto*
 have *5* : $\vdash$ (¬(*ds f*) ∧ (*ds f* ∨ *f*)) = (¬(*ds f*) ∧ *f*) **by** *auto*
 have *6* : $\vdash$ (¬(*ds f*) ∧ *f*) = (*f* ∧ *bs* ¬ *f*) **using** *1* **by** *auto*
 from *2 4 5 6* **show** *?thesis* **by** (*metis first-d-def int-eq*)
qed

lemma *FstOrDiEqvDi*:
$\vdash$ (▷*f* ∨ *di f*) = *di f*
proof −
 have *1*: $\vdash$ (▷*f* ∨ *di f*) = ((*f* ∧ *bs* ¬*f*) ∨ *di f*) **by** (*simp add*: *first-d-def*)
 have *2*: $\vdash$ ((*f* ∧ *bs* ¬*f*) ∨ *di f*) = ((*f* ∨ *di f*) ∧ (*bs* ¬ *f* ∨ *di f*)) **by** *auto*
 have *3*: $\vdash$ (*f* ∨ *di f*) = *di f*
 by (*metis 2 DiIntro RRDiamondEqvDi int-eq Prop02 Prop03 Prop11 Prop12*)
 hence *4*: $\vdash$ ((*f* ∨ *di f*) ∧ (*bs* ¬ *f* ∨ *di f*)) = (*di f* ∧ (*bs* ¬ *f* ∨ *di f*)) **by** *auto*
 have *5*: $\vdash$ (*di f* ∧ (*bs* ¬ *f* ∨ *di f*)) = *di f* **by** *auto*
 from *1 2 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndDiEqvFst*:
$\vdash$ (▷*f* ∧ *di f*) = ▷*f*

proof −
 have *1*: ⊢ (▷*f* ∧ *di f*) = ((*f* ∧ *bs* ¬*f*) ∧ *di f*) **by** (*simp add*: *first-d-def*)
 have *2*: ⊢ (*f* ∧ *di f*) = *f* **by** (*meson DiIntro Prop10 Prop11*)
 hence *3*: ⊢ (*f* ∧ *bs* ¬*f* ∧ *di f*) = (*f* ∧ *bs* ¬*f*) **by** *auto*
 from *1 3* **show** *?thesis* **by** (*metis first-d-def int-iffD2 int-iffI Prop12*)
qed

lemma *EmptyChopSkipInduct*:
 assumes ⊢ *empty* ⟶ *f*
 ⊢ *prev f* ⟶ *f*
 shows ⊢ *f*
proof −
 have *1*: ⊢ *empty* ⟶ *f* **using** *assms*(*1*) **by** *auto*
 have *2*: ⊢ *prev f* ⟶ *f* **using** *assms*(*2*) **by** *blast*
 have *3*: ⊢ (*empty* ∨ *prev f*) ⟶ *f* **using** *1 2* **by** *fastforce*
 have *4*: ⊢ *wprev f* = (*empty* ∨ *prev f*) **by** (*simp add*: *WprevEqvEmptyOrPrev*)
 hence *5*: ⊢ *wprev f* ⟶ *f* **using** *3* **by** *fastforce*
 hence *6*: ⊢ ¬*f* ⟶ ¬ (*wprev f*) **by** *fastforce*
 hence *7*: ⊢ ¬*f* ⟶ *prev* (¬ *f*) **by** (*simp add*: *wprev-d-def*)
 hence *8*: ⊢ ¬ ¬ *f* **by** (*rule PrevLoop*)
 from *8* **show** *?thesis* **by** *auto*
qed

lemma *DiAndEmptyEqvAndEmpty*:
⊢ (*di f* ∧ *empty*) = (*f* ∧ *empty*)
proof −
 have *1* : ⊢ *di f* = (*f* ∨ *di f*;*skip*) **using** *DiEqvOrDiChopSkipB* **by** *blast*
 hence *2*: ⊢ (*di f* ∧ *empty*) = ((*f* ∨ *di f*;*skip*) ∧ *empty*) **by** *fastforce*
 have *3* : ⊢ ((*f* ∨ *di f*;*skip*) ∧ *empty*) = ((*f* ∧ *empty*) ∨ (*di f*;*skip* ∧ *empty*)) **by** *auto*
 have *4*: ⊢ ¬(*di f*;*skip* ∧ *empty*) **using** *DsDi NotDsAndEmpty* **by** *fastforce*
 hence *5* : ⊢ ((*f* ∧ *empty*) ∨ (*di f*;*skip* ∧ *empty*)) = (*f* ∧ *empty*) **by** *auto*
 from *2 3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *MoreImpImpChopSkipEqv*:
⊢ *more* ⟶ ((*f*⟶*g*);*skip* = ((*f*;*skip*)⟶(*g*;*skip*)))
proof −
 have *01*: ⊢ (*f*⟶*g*) = (¬ *f* ∨ *g*) **by** *auto*
 hence *02*: ⊢ (*f*⟶*g*);*skip* = (¬ *f* ∨ *g*);*skip* **by** (*simp add*: *LeftChopEqvChop*)
 hence *1*: ⊢ (*more* ∧ (*f*⟶*g*);*skip*) = (*more* ∧ (¬ *f* ∨ *g*);*skip*) **by** *fastforce*
 have *2*: ⊢ (¬ *f* ∨ *g*);*skip* = (¬ *f*;*skip* ∨ *g*;*skip*)
 using *OrChopEqv* **by** *auto*
 hence *3*: ⊢ (*more* ∧ (¬ *f* ∨ *g*);*skip*) = (*more* ∧ (¬ *f*;*skip* ∨ *g*;*skip*))
 by *auto*
 have *4*: ⊢ ¬(¬ *f*;*skip*) = (*empty* ∨ (*f*;*skip*))
 using *NotNotChopSkip* **by** *blast*
 hence *5*: ⊢ (¬ *f*;*skip*) = ¬(*empty* ∨ (*f*;*skip*))
 by *fastforce*
 have *6*: ⊢ ¬(*empty* ∨ (*f*;*skip*)) = (*more* ∧ ¬(*f*;*skip*))

using *5 NotChopSkipEqvMoreAndNotChopSkip* **by** *fastforce*
have *7*: $\vdash (\neg\ f;skip \vee g;skip) = (\ (more \wedge \neg(f;skip)) \vee g;skip)$
using *5 6* **by** *fastforce*
hence *8*: $\vdash (more \wedge(\neg\ f;skip \vee g;skip)) = (more \wedge (\ (more \wedge \neg(f;skip)) \vee g;skip))$
by *auto*
have *9*: $\vdash (more \wedge (\ (more \wedge \neg(f;skip)) \vee g;skip)) = (more \wedge (\neg(f;skip) \vee g;skip))$
by *auto*
have *10*: $\vdash (more \wedge (\neg(f;skip) \vee g;skip)) = (more \wedge ((f;skip) \longrightarrow (g;skip)))$
by *auto*
have *11*: $\vdash (more \wedge (f \longrightarrow g);skip) = (more \wedge ((f;skip) \longrightarrow (g;skip)))$
using *1 2 3 8 9 10* **by** *fastforce*
from *11* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *MoreImpImpPrevEqv*:
$\vdash more \longrightarrow (\ prev(f \longrightarrow g) = (prev\ f \longrightarrow prev\ g)\ )$
by (*simp add*: *MoreImpImpChopSkipEqv prev-d-def*)

lemma *DiEqvDiFst*:
$\vdash di\ f = di\ (\triangleright\ f)$
proof −
have *1*: $\vdash\ di\ (\triangleright\ f) = di\ (f \wedge bs\ \neg f)$
by (*simp add*: *first-d-def*)
have *2*: $\vdash di\ (f \wedge bs\ \neg f) \longrightarrow di\ f \wedge di\ (bs\ \neg\ f)$
using *DiAndImpAnd* **by** *auto*
hence *3*: $\vdash di\ (f \wedge bs\ \neg f) \longrightarrow di\ f$
by *auto*
have *4*: $\vdash di\ (\triangleright\ f) \longrightarrow di\ f$ **using** *1 3*
by *fastforce*
have *5*: $\vdash (di\ f \wedge empty) = (f \wedge empty)$
using *DiAndEmptyEqvAndEmpty* **by** *blast*
have *6*: $\vdash (\triangleright\ f \wedge empty) = (f \wedge empty)$
using *FstAndEmptyEqvAndEmpty* **by** *auto*
have *7*: $\vdash di\ f \wedge empty \longrightarrow \triangleright f$
using *5 6* **by** *fastforce*
have *8*: $\vdash \triangleright f \longrightarrow di\ (\triangleright\ f)$
using *DiIntro* **by** *auto*
have *9*: $\vdash\ di\ f \wedge empty \longrightarrow di\ (\triangleright\ f)$
using *7 8* **using** *lift-imp-trans* **by** *blast*
hence *10*: $\vdash empty \longrightarrow (di\ f \longrightarrow di\ (\triangleright\ f))$
by *auto*
have *11*: $\vdash prev\ (\ di\ f \longrightarrow di\ (\triangleright\ f)) \longrightarrow more$
by (*simp add*: *ChopSkipImpMore prev-d-def*)
have *12*: $\vdash more \longrightarrow (\ prev\ (\ di\ f \longrightarrow di\ (\triangleright\ f)) = (prev(di\ f) \longrightarrow prev(di\ (\triangleright f))))$
using *MoreImpImpPrevEqv* **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*) = $\rhd$*f*
by (*simp add*: *first-d-def*)
have *24*: $\vdash$ ($\rhd$*f*) $\longrightarrow$ *di* ($\rhd$*f*)
using *DiIntro* **by** *auto*
have *25*: $\vdash$ (*f* $\wedge$ *bs* $\neg$ *f*) $\longrightarrow$ *di* ($\rhd$*f*)
using *23 24* **by** *fastforce*
have *26*: $\vdash$ (*di f* $\wedge$ *bs* $\neg$ *f*) $\longrightarrow$ *di* ($\rhd$*f*)
using *25 22* **by** *fastforce*
hence *27*: $\vdash$ (*di f* $\wedge$ *bs* $\neg$ *f* $\wedge$ (*prev* (*di f* $\longrightarrow$ *di* ($\rhd$ *f*)))) $\longrightarrow$ *di* ($\rhd$*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* ($\rhd$ *f*)))) =
(*di f* $\wedge$ *ds f* $\wedge$ (*prev* (*di f* $\longrightarrow$ *di* ($\rhd$ *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* ($\rhd$ *f*)))) =
(*di f* $\wedge$ *prev*(*di f*) $\wedge$ (*prev* (*di f* $\longrightarrow$ *di* ($\rhd$ *f*))))
by *auto*
have *32*: $\vdash$ *prev* (*di f* $\longrightarrow$ *di* ($\rhd$ *f*)) $\longrightarrow$ (*prev*(*di f*) $\longrightarrow$ *prev*(*di* ($\rhd$*f*)))
using *14* **by** *auto*
hence *33*: $\vdash$ *di f* $\wedge$ *prev*(*di f*) $\wedge$ *prev* (*di f* $\longrightarrow$ *di* ($\rhd$ *f*)) $\longrightarrow$
di f $\wedge$ *prev*(*di f*) $\wedge$ (*prev*(*di f*) $\longrightarrow$ *prev*(*di* ($\rhd$*f*)))
by *auto*
have *34*: $\vdash$ *di f* $\wedge$ *prev*(*di f*) $\wedge$ (*prev*(*di f*) $\longrightarrow$ *prev*(*di* ($\rhd$*f*))) $\longrightarrow$ *prev*(*di* ($\rhd$*f*))
by *auto*
have *35*: $\vdash$ *prev*(*di* ($\rhd$*f*))= (*di* ($\rhd$*f*));*skip*
by (*simp add*: *prev-d-def*)
have *36*: $\vdash$ (*di* ($\rhd$*f*));*skip* $\longrightarrow$ *di*(*di* ($\rhd$*f*))
using *ChopImpDi* **by** *auto*
have *37*: $\vdash$ *di*(*di* ($\rhd$*f*)) = *di* ($\rhd$*f*)
using *DiEqvDiDi* **by** *fastforce*
have *38*: $\vdash$ *di f* $\wedge$ *prev*(*di f*) $\wedge$ (*prev*(*di f*) $\longrightarrow$ *prev*(*di* ($\rhd$*f*))) $\longrightarrow$ *di* ($\rhd$*f*)
using *37 36 35 34* **by** *fastforce*

have *39*: $\vdash di\ f \wedge \neg(bs \neg\ f) \wedge (prev\ (di\ f \longrightarrow di\ (\rhd\ f))) \longrightarrow di\ (\rhd f)$
  using *29 31 33 38* **by** *fastforce*
hence *40*: $\vdash \neg(bs \neg\ f) \wedge (prev\ (di\ f \longrightarrow di\ (\rhd\ f))) \longrightarrow (di\ f \longrightarrow di\ (\rhd f))$
  by *fastforce*
have *41*: $\vdash bs \neg\ f \wedge (prev\ (di\ f \longrightarrow di\ (\rhd\ f))) \longrightarrow (di\ f \longrightarrow di\ (\rhd f))$
  using *27* **by** *fastforce*
have *42*: $\vdash (\neg(bs \neg\ f) \vee bs \neg\ f) \wedge (prev\ (di\ f \longrightarrow di\ (\rhd\ f))) \longrightarrow (di\ f \longrightarrow di\ (\rhd f))$
  using *40 41* **by** *fastforce*
have *43*: $\vdash (\neg(bs \neg\ f) \vee bs \neg\ f)$
  by *auto*
have *44*: $\vdash (prev\ (di\ f \longrightarrow di\ (\rhd\ f))) \longrightarrow (di\ f \longrightarrow di\ (\rhd f))$
  using *42 43* **by** *fastforce*
have *45*: $\vdash di\ f \longrightarrow di\ (\rhd f)$
  using *10 44 EmptyChopSkipInduct* **by** *blast*
from *4 45* **show** *?thesis* **by** *fastforce*
qed

lemma *FstDiEqvFst*:
$\vdash \rhd(di\ f) = \rhd f$
proof –
have *1*: $\vdash \rhd(di\ f) = (di\ f \wedge bs \neg\ (di\ f))$ **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*: $\vdash (f \wedge bs\ (\neg\ f)) = \rhd f$ **by** (*simp add*: *first-d-def*)
have *11*: $\vdash ((f \wedge bs\ (\neg\ f)) \vee (ds\ f \wedge bs\ (\neg\ f))) = \rhd f$ **using** *9 10* **by** *fastforce*
from *1 5 7 8 11* **show** *?thesis* **by** (*metis int-eq*)
qed

lemma *DiAndFstOrEqvFstOrDiAnd*:
$\vdash (di\ f \wedge (\rhd f \vee g)) = (\rhd f \vee (di\ f \wedge g))$
proof –
have *1*: $\vdash (di\ f \wedge (\rhd f \vee g)) = (\rhd f \wedge di\ f) \vee (di\ f \wedge g)$ **by** *auto*
have *2*: $\vdash (\rhd f \wedge di\ f) = \rhd f$ **using** *FstAndDiEqvFst* **by** *blast*
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *DiOrFstAndEqvDi*:
$\vdash di\ f \vee (\rhd f \wedge g) = di\ f$
proof –
have *1*: $\vdash (di\ f \vee (\rhd f \wedge g)) = ((\rhd f \vee di\ f) \wedge (di\ f \vee g))$ **by** *auto*
have *2*: $\vdash (\rhd f \vee di\ f) = di\ f$ **using** *FstOrDiEqvDi* **by** *blast*
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *FstDiAndDiEqv*:
$\vdash \rhd(di\ f \wedge di\ g) = ((\rhd f \wedge di\ g) \vee (\rhd g \wedge di\ f))$
proof −
have *1*: $\vdash \rhd(di\ f \wedge di\ g) = ((di\ f \wedge di\ g) \wedge bs\ \neg(di\ f \wedge di\ g))$ **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*: $\vdash \rhd f = (bs\ \neg f \wedge di\ f)$ **using** *FstEqvBsNotAndDi* **by** *blast*
hence *9*: $\vdash (bs\ \neg f \wedge di\ f \wedge di\ g) = (\rhd f \wedge di\ g)$ **by** *auto*
have *10*: $\vdash \rhd g = (bs\ \neg g \wedge di\ g)$ **using** *FstEqvBsNotAndDi* **by** *blast*
hence *11*: $\vdash (di\ f \wedge bs\ \neg g \wedge di\ g) = (di\ f \wedge \rhd g)$ **by** *auto*
have *12*: $\vdash (di\ f \wedge di\ g \wedge (bs\ \neg f \vee bs\ \neg g)) =$
$((\rhd f \wedge di\ g) \vee (di\ f \wedge \rhd g))$ **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*:
$\vdash bi\ \neg\ (\rhd f) = bi\ \neg\ f$
proof −
have *1*: $\vdash di\ f = di\ (\rhd f)$ **using** *DiEqvDiFst* **by** *blast*
hence *2*: $\vdash \neg(di\ f) = \neg(\ di\ (\rhd f))$ **by** *auto*
from *1 2* **show** *?thesis* **using** *NotDiEqvBiNot* **by** *fastforce*
qed

lemma *BsNotFstEqvBsNot*:
$\vdash bs\ \neg\ (\rhd f) = bs\ \neg\ f$
proof −
have *1*: $\vdash bs\ \neg\ (\rhd f) = (empty \vee bi\ \neg\ (\rhd f);skip)$ **by** (*simp add*: *bs-d-def*)
have *2*: $\vdash bi\ \neg\ (\rhd f) = bi\ \neg\ f$ **using** *BiNotFstEqvBiNot* **by** *blast*
hence *3*: $\vdash bi\ \neg\ (\rhd f);skip = bi\ \neg\ f;skip$ **using** *LeftChopEqvChop* **by** *blast*
hence *4*: $\vdash (empty \vee bi\ \neg\ (\rhd f);skip) = (empty \vee bi\ \neg\ f;skip)$ **by** *auto*
from *1 4* **show** *?thesis* **by** (*simp add*: *bs-d-def*)
qed

lemma *BsFalseEqvEmpty*:
$\vdash bs\ \#False = empty$
proof −
have *1*: $\vdash bs\ \#False = (empty \vee bi\ \#False;skip)$
by (*simp add*: *bs-d-def*)
have *2*: $\vdash \neg(bi\ \#False;skip)$
by (*metis 1 BiEqvAndEmptyOrBiChopSkip FstTrue NotNotChopSkip first-d-def int-eq intensional-simps*(*17*) *intensional-simps*(*19*) *intensional-simps*(*2*) *intensional-simps*(*29*) *intensional-simps*(*3*) *intensional-simps*(*8*))
from *1 2* **show** *?thesis* **by** *fastforce*

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** *fastforce*
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*: ⊢ (*empty* ∧ (*empty* ∨ *bi* ¬ *f*;*skip*) ∧ *init w*) =
(*empty* ∧ *bs* ¬ *f* ∧ *init w*)
by (*simp add*: *bs-d-def*)
have *4*: ⊢ (*empty* ∧ *bs* ¬ *f* ∧ *init w*) = (*empty* ∧ *init w* ∧ *bs* ¬ *f*)
by *auto*
have *5*: ⊢ (*empty* ∧ *init w*) = ⊳ (*init w*)
using *FstState* **by** *fastforce*
hence *6*: ⊢ (*empty* ∧ *init w* ∧ *bs* ¬ *f*) = (⊳ (*init w*) ∧ *bs* ¬ *f*)
by *auto*
have *7*: ⊢ ⊳ (*init w*) ∧ *bs* ¬ *f* ⟶ (⊳ (*init w*) ∧ *bs* ¬ *f*) ∨ (⊳*f* ∧ *bs* ¬(*init w*))
by *auto*
have *8*: ⊢ ⊳(*init w* ∨ *f*) = ((⊳ (*init w*) ∧ *bs* ¬ *f*) ∨ (⊳*f* ∧ *bs* ¬(*init w*)))
using *FstWithOrEqv* **by** *blast*
from *1 2 3 4 5 6 7 8* **show** *?thesis* **by** *fastforce*
qed

lemma *FstLenSame*:
(∀ σ. (σ ⊨ *di* (⊳*f* ∧ *len*(*i*)) ∧ *di* (⊳*f* ∧ *len*(*j*))) ⟶ (*i*=*j*))
by (*simp add*: *DiLenFstsem FstLenSamesem*)

lemma *FstLenSame-1*:
⊢ *di* (⊳*f* ∧ *len*(*i*)) ∧ *di* (⊳*f* ∧ *len*(*j*)) ⟶ (#*i*=#*j*)
using *FstLenSame Valid-def* **by** *fastforce*

lemma *FstAndLenSame*:
(∀ σ. (σ ⊨ *di* ((⊳*f* ∧ *g1*) ∧ *len*(*i*)) ∧ *di* ((⊳*f* ∧ *g2*) ∧ *len*(*j*))) ⟶ (*i*=*j*))
apply (*simp add*: *DiLenFstAndsem*)
using *linorder-neqE-nat* **by** *blast*

lemma *FstAndLenSame-1*:
⊢ *di* ((⊳*f* ∧ *g1*) ∧ *len*(*i*)) ∧ *di* ((⊳*f* ∧ *g2*) ∧ *len*(*j*)) ⟶ (#*i*=#*j*)
using *FstAndLenSame Valid-def* **by** *fastforce*

lemma *FstLenSameChop*:
(∀ σ. (σ ⊨ ((⊳*f* ∧ *g1*) ∧ *len*(*i*));*h1* ∧ ((⊳*f* ∧ *g2*) ∧ *len*(*j*));*h2*) ⟶ (*i*=*j*))
proof
fix σ
show (σ ⊨ ((⊳*f* ∧ *g1*) ∧ *len*(*i*));*h1* ∧ ((⊳*f* ∧ *g2*) ∧ *len*(*j*));*h2*) ⟶ (*i*=*j*)
proof
assume *0*: (σ ⊨ ((⊳*f* ∧ *g1*) ∧ *len*(*i*));*h1* ∧ ((⊳*f* ∧ *g2*) ∧ *len*(*j*));*h2*)
have *1*: (σ ⊨ ((⊳*f* ∧ *g1*) ∧ *len*(*i*));*h1*) **using** *0* **by** *auto*
have *2*: (σ ⊨ ((⊳*f* ∧ *g1*) ∧ *len*(*i*));*h1*) ⟶
(σ ⊨ ((⊳*f* ∧ *g1*) ∧ *len*(*i*));#*True*) **by** (*metis ChopImpDi Valid-def di-d-def unl-lift2*)
have *3*: (σ ⊨ *di*((⊳*f* ∧ *g1*) ∧ *len*(*i*))) **using** *1 2* **by** (*simp add*: *di-d-def*)
have *4*: (σ ⊨ ((⊳*f* ∧ *g2*) ∧ *len*(*j*));*h2*) **using** *0* **by** *auto*
have *5*: (σ ⊨ ((⊳*f* ∧ *g2*) ∧ *len*(*j*));*h2*) ⟶
(σ ⊨ ((⊳*f* ∧ *g2*) ∧ *len*(*j*));#*True*) **by** (*metis ChopImpDi Valid-def di-d-def unl-lift2*)

```
    have 6: (σ ⊨  di((▷f ∧ g2) ∧ len(j)))  using 4 5 by (simp add: di-d-def)
    have 7: (σ ⊨  di((▷f ∧ g1) ∧ len(i)) ∧ di((▷f ∧ g2) ∧ len(j)))  using 3 6 by auto
    thus (i=j) using FstAndLenSame by blast
    qed
qed

lemma FstLenSameChop-1:
 ⊢  ((▷f ∧ g1) ∧ len(i));h1 ∧  ((▷f ∧ g2) ∧ len(j));h2 ⟶ (#i=#j)
using FstLenSameChop Valid-def by fastforce

lemma DiImpExistsOneDiLenAndFst:
 (∀σ. (σ⊨ di f) ⟶ (∃! k. (σ ⊨ di(  ▷f ∧ len(k)))))
proof
 fix σ
 show (σ⊨ di f) ⟶ (∃! k. (σ ⊨ di(  ▷f ∧ len(k))))
 proof
  assume 0: (σ⊨ di f)
  have 1: (σ ⊨ di (▷ f))
       using 0 DiEqvDiFst Valid-def by force
  have 2: (σ ⊨ ▷ f) = ( (σ⊨▷ f) ∧ (∃k. (σ ⊨ len(k))))
       using AndExistsLen[of TEMP ▷ f] by (simp add: Valid-def)
  have 3: ((σ⊨▷ f) ∧ (∃k. (σ ⊨ len(k)))) =
           (∃k.  (σ⊨▷ f) ∧  (σ ⊨ len(k)))
       by auto
  have 4: (σ ⊨ di(▷ f)) = (∃k. (σ ⊨ di(▷f ∧ len(k))))
       using 2 3 by (metis 1 DiLensem di-defs)
  have 5: (∃k. (σ ⊨ di(▷f ∧ len(k))))
       using 1 using 4 by auto
  then obtain i where 6: (σ ⊨ di(▷f ∧ len(i)))  by blast
  from 5 obtain j where 7: (σ ⊨ di(▷f ∧ len(j)))  by blast
  have 8: (σ ⊨ di(▷f ∧ len(i))) ∧ (σ ⊨ di(▷f ∧ len(j)))
       using 6 7 by auto
  hence 9: (σ ⊨ di(▷f ∧ len(i)) ∧ di(▷f ∧ len(j)))
       by simp
  hence 10: i=j
       using FstLenSame by blast
  have 11: ⋀j. (σ ⊨ di(▷f ∧ len(j))) ⟶ (j=i)
       using 9 10 using FstLenSame by auto
  thus (∃! k. (σ ⊨ di(  ▷f ∧ len(k) )))
       using 11 5 by blast
  qed
qed

lemma DiImpExistsOneDiLenAndFst-1:
⊢ di f ⟶ (∃! k. ( di( ▷f ∧ len(k))))
using Valid-def DiImpExistsOneDiLenAndFst 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*:
$(\exists\ k.\ (\sigma \models ((\rhd f\ \wedge\ g1) \wedge len(k));h1\ \ \wedge\ ((\rhd f\ \wedge\ g2) \wedge len(k));h2)) =$
$(\exists\ k.\ (\sigma \models (((\rhd f\ \wedge\ g1) \wedge (\rhd f\ \wedge\ g2)) \wedge len(k));(h1\ \wedge\ h2)\ ))$
proof
assume *0*: $\exists k.\ \sigma \models\ ((\rhd\ f\ \wedge\ g1) \wedge len\ k)\ ;\ h1\ \ \wedge\ \ ((\rhd\ f\ \wedge\ g2) \wedge len\ k)\ ;\ h2$
obtain *k* **where** *1*: $\sigma \models\ ((\rhd\ f\ \wedge\ g1) \wedge len\ k)\ ;\ h1\ \ \wedge\ \ ((\rhd\ f\ \wedge\ g2) \wedge len\ k)\ ;\ h2$
using *0* **by** *auto*
hence *2*: $(\sigma \models (((\rhd f\ \wedge\ g1) \wedge (\rhd f\ \wedge\ g2)) \wedge len(k));(h1\ \wedge\ h2))$
using *LFstAndDist-help* **by** *blast*
show $(\exists\ k.\ (\sigma \models (((\rhd f\ \wedge\ g1) \wedge (\rhd f\ \wedge\ g2)) \wedge len(k));(h1\ \wedge\ h2)\ ))$
using *2* **by** *auto*
next
assume *3*: $(\exists\ k.\ (\sigma \models (((\rhd f\ \wedge\ g1) \wedge (\rhd f\ \wedge\ g2)) \wedge len(k));(h1\ \wedge\ h2)\ ))$
obtain *k* **where** *4*: $(\sigma \models (((\rhd f\ \wedge\ g1) \wedge (\rhd f\ \wedge\ g2)) \wedge len(k));(h1\ \wedge\ h2)\ )$
using *3* **by** *auto*
hence *5*: $(\sigma \models ((\rhd f\ \wedge\ g1) \wedge len(k));h1\ \ \wedge\ ((\rhd f\ \wedge\ g2) \wedge len(k));h2)$
using *LFstAndDist-help* **by** *blast*
show $(\exists\ k.\ (\sigma \models ((\rhd f\ \wedge\ g1) \wedge len(k));h1\ \ \wedge\ ((\rhd f\ \wedge\ g2) \wedge len(k));h2))$
using *5* **by** *auto*
qed

lemma *LFstAndDistrsem*:
$(\forall\ \sigma.\ (\sigma \models ((\rhd f\ \wedge\ g1);h1\ \wedge\ (\rhd f\ \wedge\ g2);h2) = (\rhd f\ \wedge\ g1\ \wedge\ g2);(h1\ \wedge\ h2)))$
proof
fix σ
show $(\sigma \models ((\rhd f\ \wedge\ g1);h1\ \wedge\ (\rhd f\ \wedge\ g2);h2) = (\rhd f\ \wedge\ g1\ \wedge\ g2);(h1\ \wedge\ h2))$
proof −
have *1*: $(\sigma \models (\rhd f\ \wedge\ g1);h1) = (\exists\ i.\ (\sigma \models ((\rhd f\ \wedge\ g1) \wedge len(i));h1)\ )$
using *AndExistsLenChop*[*of TEMP* $(\rhd\ f\ \wedge\ g1)$] **by** *fastforce*
have *2*: $(\sigma \models (\rhd f\ \wedge\ g2);h2) = (\exists\ j.\ (\sigma \models ((\rhd f\ \wedge\ g2) \wedge len(j));h2)\ )$
using *AndExistsLenChop*[*of TEMP* $(\rhd f\ \wedge\ g2)$] **by** *fastforce*
have *3*: $(\sigma \models (\rhd f\ \wedge\ g1);h1\ \wedge\ (\rhd f\ \wedge\ g2);h2) =$
$(\ (\exists\ i\,j.\ (\sigma \models ((\rhd f\ \wedge\ g1) \wedge len(i));h1\ \ \wedge$
$((\rhd f\ \wedge\ g2) \wedge len(j));h2)\ )$
$)$
using *1 2* **by** *auto*
have *4*: $(\ (\exists\ i\,j.\ (\sigma \models ((\rhd f\ \wedge\ g1) \wedge len(i));h1\ \ \wedge$
$((\rhd f\ \wedge\ g2) \wedge len(j));h2)\ )$
$) =$
$(\ (\exists\ k.\ (\sigma \models ((\rhd f\ \wedge\ g1) \wedge len(k));h1\ \ \wedge$
$((\rhd f\ \wedge\ g2) \wedge len(k));h2)\ )$
$)$
using *FstLenSameChop* **by** *blast*
have *5*: $(\exists\ k.\ (\sigma \models ((\rhd f\ \wedge\ g1) \wedge len(k));h1\ \ \wedge\ ((\rhd f\ \wedge\ g2) \wedge len(k));h2)) =$
$(\exists\ k.\ (\sigma \models (((\rhd f\ \wedge\ g1) \wedge (\rhd f\ \wedge\ g2)) \wedge len(k));(h1\ \wedge\ h2)\ ))$
using *LFstAndDist-help-1* **by** *blast*
have *6* : $(\exists\ k.\ (\sigma \models (((\rhd f\ \wedge\ g1) \wedge (\rhd f\ \wedge\ g2)) \wedge len(k));(h1\ \wedge\ h2)\ )) =$
$(\sigma \models ((\rhd f\ \wedge\ g1) \wedge (\rhd f\ \wedge\ g2));(h1\ \wedge\ h2))$

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*:
⊢ ((▷*f* ∧ *g1*);*h1* ∧ (▷*f* ∧ *g2*);*h2*) = (▷*f* ∧ *g1* ∧ *g2*);(*h1* ∧ *h2*)
using *LFstAndDistrsem* **by** *fastforce*

lemma *LFstAndDistrA*:
⊢ ((▷*f* ∧ *g1*);*h* ∧ (▷*f* ∧ *g2*);*h*) = (▷*f* ∧ *g1* ∧ *g2*);*h*
proof −
have *1*: ⊢ ((▷*f* ∧ *g1*);*h* ∧ (▷*f* ∧ *g2*);*h*) = (▷*f* ∧ *g1* ∧ *g2*);(*h* ∧ *h*) **using** *LFstAndDistr* **by** *blast*
have *2*: ⊢ (▷*f* ∧ *g1* ∧ *g2*);(*h* ∧ *h*) = (▷*f* ∧ *g1* ∧ *g2*);*h* **by** *auto*
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *LFstAndDistrB*:
⊢ ((▷*f* ∧ *g*);*h1* ∧ (▷*f* ∧ *g*);*h2*) = (▷*f* ∧ *g*);(*h1* ∧ *h2*)
proof −
have *1*: ⊢ ((▷*f* ∧ *g*);*h1* ∧ (▷*f* ∧ *g*);*h2*) = (▷*f* ∧ *g* ∧ *g*);(*h1* ∧ *h2*) **using** *LFstAndDistr* **by** *blast*
have *2*: ⊢ (▷*f* ∧ *g* ∧ *g*);(*h1* ∧ *h2*) = (▷*f* ∧ *g*);(*h1* ∧ *h2*) **by** *auto*
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *LFstAndDistrC*:
⊢ ((▷*f*);*h1* ∧ (▷*f*);*h2*) = (▷*f*);(*h1* ∧ *h2*)
proof −
have *1*: ⊢ ((▷*f* ∧ #*True*);*h1* ∧ (▷*f* ∧ #*True*);*h2*) = (▷*f* ∧ #*True* ∧ #*True*);(*h1* ∧ *h2*)
using *LFstAndDistr* **by** *blast*
have *2*: ⊢ (▷*f* ∧ #*True*);*h1* = (▷*f*);*h1*
by *auto*
have *3*: ⊢ (▷*f* ∧ #*True*);*h2* = (▷*f*);*h2*
by *auto*
have *4*: ⊢ (▷*f* ∧ #*True* ∧ #*True*);(*h1* ∧ *h2*) = (▷*f*);(*h1* ∧ *h2*)
by *auto*
from *1 2 3 4* **show** *?thesis* **by** *auto*
qed

lemma *LFstAndDistrD*:
⊢ (*di*(▷*f* ∧ *g1*) ∧ *di*(▷*f* ∧ *g2*)) = *di*(▷*f* ∧ *g1* ∧ *g2*)
proof −
have *1*: ⊢ ((▷*f* ∧ *g1*);#*True* ∧ (▷*f* ∧ *g2*);#*True*) = (▷*f* ∧ *g1* ∧ *g2*);(#*True* ∧ #*True*)
using *LFstAndDistr* **by** *blast*
have *2*: ⊢ (▷*f* ∧ *g1*);#*True* = *di*(▷*f* ∧ *g1*)
by (*simp add*: *di-d-def*)
have *3*: ⊢ (▷*f* ∧ *g2*);#*True* = *di*(▷*f* ∧ *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*: $\vdash (\triangleright(f^r))^r\ = \triangleleft\ (f)$
using *RRFirstEqvLast* **by** *blast*
have *13*: $\vdash ((\triangleright(f^r))^r\ \wedge (g1^r)^r) = (\triangleleft\ f \wedge g1)$
using *12 7* **by** *fastforce*
have *14*: $\vdash ((\triangleright(f^r))^r\ \wedge (g2^r)^r) = (\triangleleft\ f \wedge g2)$
using *12 8* **by** *fastforce*
have *15*: $\vdash (h1;(\triangleleft\ f \wedge g1) \wedge h2;(\triangleleft\ f \wedge g2)) =$
$((h1^r)^r;(\triangleright(f^r) \wedge g1^r)^r \wedge (h2^r)^r;(\triangleright(f^r) \wedge (g2^r))^r)$

using *14 13 10 11 5 6* **by** (*metis 4 int-eq*)
have *16*: $\vdash (((\triangleright(f^r) \wedge (g1^r) \wedge (g2^r));((h1^r) \wedge (h2^r)))^r) =$
$((h1^r) \wedge (h2^r))^r;((\triangleright\ (f^r)) \wedge (g1^r) \wedge (g2^r))^r$

by (*simp add*: *RevChop*)
have *17*: $\vdash ((\triangleright\ (f^r)) \wedge (g1^r) \wedge (g2)^r)^r = ((\triangleright\ (f^r))^r \wedge (g1^r)^r \wedge (g2^r)^r)$
by (*metis inteq-reflection rev-fun2*)
have *18*: $\vdash ((\triangleright\ (f^r))^r \wedge (g1^r)^r \wedge (g2^r)^r) = (\triangleleft\ f \wedge g1 \wedge g2)$
using *12 7 8* **by** *fastforce*
have *19*: $\vdash ((h1^r) \wedge (h2^r))^r = (h1 \wedge h2)$
using *RRAnd* **by** *auto*
have *20*: $\vdash ((h1^r) \wedge (h2^r))^r;((\triangleright\ (f^r)) \wedge (g1^r) \wedge (g2^r))^r =$
$(h1 \wedge h2);(\triangleleft\ f \wedge g1 \wedge g2)$
using *19 17 18* **using** *ChopEqvChop* **by** (*metis int-eq*)
from *15 4 3 2 16 20* **show** *?thesis* **using** *int-eq* **by** *metis*
qed

lemma *LstAndDistrA*:
$\vdash (h;(\triangleleft\ f \wedge g1) \wedge h;(\triangleleft\ f \wedge g2)) = h;(\triangleleft\ f \wedge g1 \wedge g2)$
proof −
have *1*: $\vdash (h;(\triangleleft\ f \wedge g1) \wedge h;(\triangleleft\ f \wedge g2)) = (h \wedge h);(\triangleleft\ f \wedge g1 \wedge g2)$
using *LstAndDistr* **by** *blast*
have *2*: $\vdash (h \wedge h);(\triangleleft\ f \wedge g1 \wedge g2) = h;(\triangleleft\ f \wedge g1 \wedge g2)$
by *auto*
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *LstAndDistrB*:
$\vdash\ (h1;(\triangleleft\ f \wedge g) \wedge h2;(\triangleleft\ f \wedge g)) = (h1 \wedge h2);(\triangleleft\ f \wedge g)$
proof −
have *1*: $\vdash (h1;(\triangleleft\ f \wedge g) \wedge h2;(\triangleleft\ f \wedge g)) = (h1 \wedge h2);(\triangleleft\ f \wedge g \wedge g)$
using *LstAndDistr* **by** *blast*
have *2*: $\vdash (h1 \wedge h2);(\triangleleft\ f \wedge g \wedge g) = (h1 \wedge h2);(\triangleleft\ f \wedge g)$
by *auto*
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *LstAndDistrC*:
$\vdash\ (h1;(\triangleleft\ f\ ) \wedge h2;(\triangleleft\ f\ )) = (h1 \wedge h2);(\triangleleft\ f\ )$
proof −
have *1*: $\vdash (h1;(\triangleleft\ f \wedge \#True) \wedge h2;(\triangleleft\ f \wedge \#True)) = (h1 \wedge h2);(\triangleleft\ f \wedge \#True \wedge \#True\ \ )$
using *LstAndDistr* **by** *blast*
have *2*: $\vdash (h1 \wedge h2);(\triangleleft\ f \wedge \#True \wedge \#True\ \ ) = (h1 \wedge h2);(\triangleleft\ f\ )$
by *auto*
have *3*: $\vdash h1;(\triangleleft\ f \wedge \#True) = h1;(\triangleleft\ f\ )$
by *auto*
have *4*: $\vdash h2;(\triangleleft\ f \wedge \#True) = h2;(\triangleleft\ f\ )$
by *auto*
from *1 2 3 4* **show** *?thesis* **by** *auto*
qed

lemma *LstAndDistrD*:
$\vdash\ (\Diamond(\triangleleft\ f \wedge g1) \wedge \Diamond(\triangleleft\ f \wedge g2)) = \Diamond(\triangleleft\ f \wedge g1 \wedge g2)$
proof −

have *1*: $\vdash$ ($\#True$;($\lhd$ f $\wedge$ $g1$) $\wedge$ $\#True$;($\lhd$ f $\wedge$ $g2$)) = ($\#True$ $\wedge$ $\#True$);($\lhd$ f $\wedge$ $g1$ $\wedge$ $g2$)
  using *LstAndDistr* **by** *blast*
have *2*: $\vdash$ ($\#True$ $\wedge$ $\#True$);($\lhd$ f $\wedge$ $g1$ $\wedge$ $g2$) = $\diamond$($\lhd$ f $\wedge$ $g1$ $\wedge$ $g2$)
  by (*simp add*: *sometimes-d-def*)
have *3*: $\vdash$ $\#True$;($\lhd$ f $\wedge$ $g1$) = $\diamond$($\lhd$ f $\wedge$ $g1$)
  by (*simp add*: *sometimes-d-def*)
have *4*: $\vdash$ $\#True$;($\lhd$ f $\wedge$ $g2$) = $\diamond$($\lhd$ f $\wedge$ $g2$)
  by (*simp add*: *sometimes-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *NotFstChop*:
$\vdash$ $\neg$($\rhd f$;g) = ($\neg$(di ($\rhd f$)) $\vee$ ($\rhd f$;$\neg g$))
proof −
have *1*: $\vdash$ g $\longrightarrow$ $\#True$ **by** *auto*
hence *2*: $\vdash$ $\rhd f$;g $\longrightarrow$ $\rhd f$;$\#True$ **using** *RightChopImpChop* **by** *blast*
hence *3*: $\vdash$ $\rhd f$;g $\longrightarrow$ di($\rhd f$) **by** (*simp add*:*di-d-def*)
hence *4*: $\vdash$ $\neg$(di($\rhd f$)) $\longrightarrow$ $\neg$($\rhd f$;g) **by** *auto*
have *5*: $\vdash$ ($\rhd f$; $\neg g$ $\longrightarrow$ $\neg$($\rhd f$;g)) = (($\rhd f$; $\neg g$) $\wedge$ ($\rhd f$;g) $\longrightarrow$ $\#False$) **by** *auto*
have *6*: $\vdash$ (($\rhd f$; $\neg g$) $\wedge$ ($\rhd f$;g)) = $\rhd f$;($\neg g$ $\wedge$ g) **using** *LFstAndDistrC* **by** *blast*
have *7*: $\vdash$ $\neg$($\rhd f$;($\neg g$ $\wedge$ g)) **by** (*simp add*: *FstChopFalseEqvFalse*)
have *8*: $\vdash$ $\rhd f$; $\neg g$ $\longrightarrow$ $\neg$($\rhd f$;g) **using** *5 6 7* **by** *fastforce*
have *9*: $\vdash$ $\neg$(di($\rhd f$)) $\vee$ ($\rhd f$; $\neg g$) $\longrightarrow$ $\neg$($\rhd f$;g) **using** *4 8* **by** *fastforce*
have *10*: $\vdash$ di($\rhd f$) $\vee$ $\neg$(di($\rhd f$)) **by** *auto*
hence *11*: $\vdash$ ($\rhd f$;$\#True$) $\vee$ $\neg$(di($\rhd f$)) **by** (*simp add*: *di-d-def*)
hence *12*: $\vdash$ ($\rhd f$;(g $\vee$ $\neg g$)) $\vee$ $\neg$(di($\rhd f$)) **by** *auto*
have *13*: $\vdash$ ($\rhd f$;(g $\vee$ $\neg g$)) = (($\rhd f$;g) $\vee$ ($\rhd f$;$\neg g$)) **using** *ChopOrEqv* **by** *fastforce*
have *14*: $\vdash$ (($\rhd f$;g) $\vee$ ($\rhd f$;$\neg g$)) $\vee$ $\neg$(di($\rhd f$)) **using** *12 13* **by** *fastforce*
hence *15*: $\vdash$ $\neg$($\rhd f$;g) $\longrightarrow$ $\neg$(di($\rhd f$)) $\vee$ ($\rhd f$; $\neg g$) **by** *auto*
from *9 15* **show** *?thesis* **by** *fastforce*
qed

lemma *BsNotFstChop*:
$\vdash$ bs($\neg$($\rhd f$;g)) = ($empty$ $\vee$ $\neg$(di($\rhd f$)) $\vee$ ($\rhd f$;$bs\neg g$))
proof −
have *1*: $\vdash$ bs($\neg$($\rhd f$;g))= ($empty$ $\vee$ $bi\neg$($\rhd f$;g);$skip$)
  by (*simp add*:*bs-d-def*)
have *2*: $\vdash$ ($empty$ $\vee$ $bi\neg$($\rhd f$;g);$skip$) = ($empty$ $\vee$ $\neg$(di($\rhd f$;g));$skip$)
  by (*metis 1 NotDiEqvBiNot int-eq*)
have *3*: $\vdash$ ($empty$ $\vee$ $\neg$(di($\rhd f$;g));$skip$) = ($empty$ $\vee$ $\neg$(($\rhd f$;g);$\#True$);$skip$)
  by (*simp add*: *di-d-def*)
have *4*: $\vdash$ $\neg$(($\rhd f$;g);$\#True$);$skip$ = $\neg$($\rhd f$;(g;$\#True$));$skip$
  by (*metis ChopAssocB int-eq intensional-simps*(*1*))
hence *5*: $\vdash$ ($empty$ $\vee$ $\neg$(($\rhd f$;g);$\#True$);$skip$) = ($empty$ $\vee$ $\neg$($\rhd f$;(g;$\#True$));$skip$)
  by *auto*
have *6*: $\vdash$ ($empty$ $\vee$ $\neg$($\rhd f$;(g;$\#True$));$skip$) = ($empty$ $\vee$ $\neg$($\rhd f$;di(g));$skip$)
  by (*simp add*: *di-d-def*)
have *7*: $\vdash$ ($empty$ $\vee$ $\neg$($\rhd f$;di(g));$skip$) = ($empty$ $\vee$ $\neg$($\neg$($\neg$($\rhd f$;di(g));$skip$)))
  by *auto*
have *8*: $\vdash$ $\neg$($\neg$($\neg$($\rhd f$;di(g));$skip$)) = $\neg$($empty$ $\vee$ ($\rhd f$;di(g));$skip$)

using *NotNotChopSkip* **by** *fastforce*
hence *9*: $\vdash (empty \lor \neg(\neg(\neg(\triangleright f;di(g));skip))) = (empty \lor \neg(empty \lor (\triangleright f;di(g));skip))$
by *auto*
have *10*: $\vdash (empty \lor \neg(empty \lor (\triangleright f;di(g));skip)) = (empty \lor (more \land \neg((\triangleright f;di(g));skip)))$
by (*meson 6 7 9 NotChopSkipEqvMoreAndNotChopSkip Prop04 Prop06*)
have *11*: $\vdash (empty \lor (more \land \neg((\triangleright f;di(g));skip))) = (empty \lor \neg((\triangleright f;di(g));skip))$
by (*simp add*: *empty-d-def*, *auto*)
have *12*: $\vdash (empty \lor \neg((\triangleright f;di(g));skip)) = (empty \lor \neg(\triangleright f;(di(g);skip)))$
using *ChopAssocB 11* **by** *fastforce*
have *13*: $\vdash \neg(\triangleright f;(di(g);skip)) = \neg(\triangleright f;(ds(g)))$
using *DsDi* **using** *RightChopEqvChop* **by** *fastforce*
hence *14*: $\vdash (empty \lor \neg(\triangleright f;(di(g);skip))) = (empty \lor \neg(\triangleright f;(ds(g))))$
by *auto*
have *15*: $\vdash (empty \lor \neg(\triangleright f;(ds(g)))) = (empty \lor \neg(di\ (\triangleright f)) \lor (\triangleright f;\neg(ds\ g)))$
using *NotFstChop* **by** *fastforce*
have *16*: $\vdash (\triangleright f;\neg(ds\ g)) = (\triangleright f;(bs\ \neg g))$
using *NotDsEqvBsNot RightChopEqvChop* **by** *blast*
hence *17*: $\vdash ((empty \lor \neg(di\ (\triangleright f))) \lor (\triangleright f;\neg(ds\ g))) = ((empty \lor \neg(di\ (\triangleright f))) \lor (\triangleright f;(bs\ \neg g)))$
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*: $\vdash \triangleright(\triangleright f;g) = ((\triangleright f;g) \land bs\ \neg(\triangleright f;g))$
by (*simp add*: *first-d-def*)
have *2*: $\vdash bs\ \neg(\triangleright f;g) = (empty \lor \neg(di(\triangleright f)) \lor (\triangleright f;bs \neg g))$
using *BsNotFstChop* **by** *auto*
hence *3*: $\vdash ((\triangleright f;g) \land bs\ \neg(\triangleright f;g)) = ((\triangleright f;g) \land (empty \lor \neg(di(\triangleright f)) \lor (\triangleright f;bs \neg g)))$
by *auto*
have *4*: $\vdash ((\triangleright f;g) \land (empty \lor \neg(di(\triangleright f)) \lor (\triangleright f;bs \neg g))) =$
$(((\triangleright f;g) \land empty) \lor ((\triangleright f;g) \land \neg(di(\triangleright f))) \lor ((\triangleright f;g) \land (\triangleright f;bs \neg g)))$
by *auto*
have *5*: $\vdash \neg((\triangleright f;g) \land \neg(di(\triangleright f)))$
using *ChopImpDi* **by** *fastforce*
hence *6*: $\vdash (((\triangleright f;g) \land empty) \lor ((\triangleright f;g) \land \neg(di(\triangleright f))) \lor ((\triangleright f;g) \land (\triangleright f;bs \neg g))) =$
$(((\triangleright f;g) \land empty) \lor ((\triangleright f;g) \land (\triangleright f;bs \neg g)))$
by *auto*
have *7*: $\vdash ((\triangleright f;g) \land (\triangleright f;(bs \neg g))) = ((\triangleright f;(g \land (bs \neg g))))$
using *LFstAndDistrC* **by** *blast*
hence *8*: $\vdash (((\triangleright f;g) \land empty) \lor ((\triangleright f;g) \land (\triangleright f;(bs \neg g)))) =$
$(((\triangleright f;g) \land empty) \lor ((\triangleright f;(g \land (bs \neg g)))))$
by *auto*
have *9*: $\vdash (((\triangleright f;g) \land empty) \lor ((\triangleright f;(g \land (bs \neg g))))) = (((\triangleright f;g) \land empty) \lor \triangleright f;\triangleright g)$
by (*simp add*: *first-d-def*)
have *10*: $\vdash ((\triangleright f;g) \land empty) = ((\triangleright f;\triangleright g) \land empty)$
using *FstChopEmptyEqvFstChopFstEmpty* **by** *blast*
hence *11*: $\vdash (((\triangleright f;g) \land empty) \lor \triangleright f;\triangleright g) = (((\triangleright f;\triangleright g) \land empty) \lor \triangleright f;\triangleright g)$
by *auto*

have *12*: $\vdash (((\rhd f;\rhd g) \wedge \mathit{empty}) \vee \rhd f;\rhd g) = \rhd f;\rhd g$
by *auto*
from *1 3 4 6 8 9 11 12* **show** *?thesis* **by** (*metis inteq-reflection*)
qed

lemma *FstFixFst*:
$\vdash \rhd(\rhd f) = \rhd f$
proof −
have *1*: $\vdash \rhd f = (\rhd f);\mathit{empty}$ **using** *ChopEmpty* **by** (*metis int-eq*)
hence *2*: $\vdash \rhd(\ \rhd f\ ) = \rhd((\rhd f);\mathit{empty})$ **using** *FstEqvRule* **by** *blast*
have *3*: $\vdash \rhd((\rhd f);\mathit{empty}) = \rhd f;\rhd \mathit{empty}$ **using** *FstFstChopEqvFstChopFst* **by** *auto*
have *4*: $\vdash \rhd f;\rhd \mathit{empty} = \rhd f;\mathit{empty}$ **using** *FstEmpty* **using** *RightChopEqvChop* **by** *blast*
have *5*: $\vdash \rhd f;\mathit{empty} = \rhd f$ **using** *ChopEmpty* **by** *blast*
from *2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *FstCSEqvEmpty*:
$\vdash \rhd(f^\star) = \mathit{empty}$
proof −
have *1*: $\vdash \rhd(f^\star) = \rhd(\mathit{empty} \vee ((f \wedge \mathit{more});f^\star))$ **using** *ChopstarEqv FstEqvRule* **by** *blast*
from *1* **show** *?thesis* **using** *FstEmptyOrEqvEmpty* **by** *fastforce*
qed

lemma *FstIterFixFst*:
$\vdash \mathit{power}\ (\rhd\ f)\ n = \rhd(\mathit{power}\ (\rhd\ f)\ n)$
proof
(*induct n*)
case *0*
then show *?case*
proof −
have *1*: $\vdash \mathit{power}\ (\rhd\ f)\ 0 = \mathit{empty}$ **by** *auto*
have *2*: $\vdash \mathit{empty} = \rhd\ \mathit{empty}$ **using** *FstEmpty* **by** *auto*
have *3*: $\vdash \rhd\ \mathit{empty} = \rhd(\mathit{power}\ (\rhd f)\ 0)$ **by** *auto*
from *1 2 3* **show** *?thesis* **by** *auto*
qed
next
case (*Suc n*)
then show *?case*
proof −
have *4*: $\vdash (\mathit{power}\ (\rhd\ f)\ (\mathit{Suc}\ n)) = (\rhd\ f)\ ;(\mathit{power}\ (\rhd f)\ n)$
by (*simp*)
have *5*: $\vdash (\rhd\ f)\ ;(\mathit{power}\ (\rhd f)\ n) = (\rhd\ f)\ ;\ \rhd\ (\mathit{power}\ (\rhd\ f)\ n)$
using *RightChopEqvChop Suc.hyps* **by** *blast*
have *6*: $\vdash (\rhd\ f)\ ;\ \rhd\ (\mathit{power}\ (\rhd\ f)\ n) = \rhd(\rhd f;(\mathit{power}\ (\rhd\ f)\ n))$
using *FstFstChopEqvFstChopFst* **by** *fastforce*
have *7*: $\vdash \rhd(\rhd f;(\mathit{power}\ (\rhd\ f)\ n)) = \rhd(\mathit{power}\ (\rhd\ f)\ (\mathit{Suc}\ n))$
by *simp*
from *4 5 6 7* **show** *?thesis* **by** *fastforce*
qed
qed

lemma *DsImpNotFst*:
$\vdash$ *ds f* $\longrightarrow$ $(\neg(\triangleright f))$
proof $-$
have *1*: $\vdash$ $($*ds f* $\wedge$ $\triangleright f)$ $=$ $($*ds f* $\wedge$ $(f \wedge$ *bs* $\neg$ $f))$ **by** (*simp add*: *first-d-def*)
have *2*: $\vdash$ $($*ds f* $\wedge$ $(f \wedge$ *bs* $\neg$ $f))$ $=$ $($*ds f* $\wedge$ $f \wedge \neg($*ds f*$))$ **using** *NotDsEqvBsNot* **by** *fastforce*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *LFixedAndDistrB1*:
$\vdash$ $(len(k);f \wedge len(k);g) = len(k);(f \wedge g)$
proof $-$
have *1*: $\vdash$ $len(k);f = (\#True \wedge len(k));f$
by *auto*
have *2*: $\vdash$ $len(k);g = (\#True \wedge len(k));g$
by *auto*
have *3*: $\vdash$ $(len(k);f \wedge len(k);g) = ((\#True \wedge len(k));f \wedge (\#True \wedge len(k));g)$
using *1 2* **by** *auto*
have *4*: $\vdash$ $((\#True \wedge len(k));f \wedge (\#True \wedge len(k));g) = (\#True \wedge len(k));(f \wedge g)$
using *LFixedAndDistrB* **by** *blast*
have *5*: $\vdash$ $(\#True \wedge len(k));(f \wedge g) = (len(k));(f \wedge g)$
by *auto*
from *1 2 3 4 5* **show** *?thesis* **by** *auto*
qed

lemma *FstLenAndEqvLenAnd*:
$\vdash$ $\triangleright(len(k) \wedge f) = (len(k) \wedge f)$
proof $-$
have *1*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow ds\ (len(k))$
using *DsAndImpElimL* **by** *fastforce*
hence *2*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (di(len(k)));skip$
using *DsDi* **by** *fastforce*
hence *3*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow ((len(k);\#True));skip$
by (*simp add*: *di-d-def*)
hence *4*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (len(k);(\#True;skip))$
using *ChopAssocB* **by** *fastforce*
hence *5*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (len(k);(skip;\#True))$
using *SkipTrueEqvTrueSkip TrueChopSkipEqvSkipChopTrue RightChopEqvChop* **by** *fastforce*
hence *6*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (len(k);(skip;\#True)) \wedge len(k)$
by *auto*
hence *7*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (len(k);(skip;\#True)) \wedge len(k);empty$
using *ChopEmpty* **by** (*metis int-eq*)
hence *8*: $\vdash$ $len(k) \wedge f \wedge ds(len(k) \wedge f) \longrightarrow (len(k);((skip;\#True) \wedge empty))$
using *LFixedAndDistrB1* **by** *fastforce*
have *9*: $\vdash$ $\neg(len(k);((skip;\#True) \wedge empty))$
by (*simp add*: *empty-d-def more-d-def next-d-def chop-defs Valid-def*)
have *10*: $\vdash$ $len(k) \wedge f \longrightarrow \neg(ds(len(k) \wedge f))$
using *8 9* **by** *fastforce*
hence *11*: $\vdash$ $len(k) \wedge f \longrightarrow bs\ \neg(len(k) \wedge f)$
using *NotDsEqvBsNot* **by** *fastforce*

hence *12*: $\vdash$ *len*(k) $\wedge$ f $\longrightarrow$ (*len*(k) $\wedge$ f) $\wedge$ *bs* $\neg$(*len*(k)$\wedge$ f)
  by *auto*
hence *13*: $\vdash$ *len*(k) $\wedge$ f $\longrightarrow$ $\triangleright$(*len*(k) $\wedge$ f)
  by (*simp add*: *first-d-def*)
have *14*: $\vdash$ $\triangleright$(*len*(k) $\wedge$ f) $\longrightarrow$ *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*:
$\vdash$ $\triangleright f$ $\longrightarrow$ $\neg$(*di* f;*skip*)
proof –
have *1*: $\vdash$ $\triangleright f$ $\longrightarrow$ *bs* $\neg$ f **by** (*simp add*: *first-d-def*,*auto*)
hence *2*: $\vdash$ $\triangleright f$ $\longrightarrow$ $\neg$(*ds* f) **using** *NotDsEqvBsNot* **by** *fastforce*
have *3*: $\vdash$ *ds* f = *di* f ; *skip* **using** *DsDi* **by** *blast*
hence *4*: $\vdash$ $\neg$(*ds* f) = $\neg$(*di* f;*skip*) **by** *auto*
from *2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *FstImpNotDiChopSkipB*:
$\vdash$ $\triangleright f$ $\longrightarrow$ $\neg$(*di* (f;*skip*))
proof –
have *1*: $\vdash$ $\triangleright f$ $\longrightarrow$ *bs* $\neg$ f
  by (*simp add*: *first-d-def*,*auto*)
hence *2*: $\vdash$ $\triangleright f$ $\longrightarrow$ $\neg$(*ds* f)
  using *NotDsEqvBsNot* **by** *fastforce*
have *3*: $\vdash$ *ds* f = *di* f ; *skip*
  using *DsDi* **by** *blast*
have *4*: $\vdash$ *di* f ; *skip* = (f;#*True*);*skip*
  by (*simp add*: *di-d-def*)
have *5*: $\vdash$(f;#*True*);*skip* = f;(#*True*;*skip*)
  using *ChopAssocB* **by** *blast*
have *6*: $\vdash$ f;(#*True*;*skip*) = f;(*skip*;#*True*)
  using *SkipTrueEqvTrueSkip* **using** *TrueChopSkipEqvSkipChopTrue RightChopEqvChop* **by** *blast*
have *7*: $\vdash$ f;(*skip*;#*True*) = (f;*skip*);#*True*
  using *ChopAssoc* **by** *blast*
have *8*: $\vdash$ (f;*skip*);#*True* = *di*(f;*skip*)
  by (*simp add*: *di-d-def*)
have *9*: $\vdash$ $\neg$(*ds* f) = $\neg$(*di*(f;*skip*))
  using *3 4 5 6 7 8* **by** *fastforce*
from *2 9* **show** *?thesis* **by** *fastforce*
qed

lemma *FstImpDiEqv*:
$\vdash$ $\triangleright f$ $\longrightarrow$ (*di* f = f)
proof –

have *1*: $\vdash \triangleright f \longrightarrow \neg(di\ f;skip)$ **using** *FstImpNotDiChopSkip* **by** *blast*
have *2*: $\vdash di\ f \longrightarrow f \vee (di\ f;skip)$ **using** *DiEqvOrDiChopSkipB* **by** *fastforce*
have *3*: $\vdash \triangleright f \wedge di\ f \longrightarrow (f \vee (di\ f;skip)) \wedge \neg(di\ f;skip)$ **using** *1 2* **by** *fastforce*
have *4*: $\vdash ((f \vee (di\ f;skip)) \wedge \neg(di\ f;skip)) = (f \wedge \neg(di\ f;skip))$ **by** *auto*
have *5*: $\vdash \triangleright f \wedge di\ f \longrightarrow f \wedge \neg(di\ f;skip)$ **using** *3 4* **by** *fastforce*
hence *6*: $\vdash \triangleright f \wedge di\ f \longrightarrow f$ **by** *fastforce*
hence *7*: $\vdash \triangleright f \longrightarrow (di\ f \longrightarrow f)$ **using** *FstAndElimL* **by** *fastforce*
have *8*: $\vdash f \longrightarrow di\ f$ **using** *DiIntro* **by** *auto*
hence *9*: $\vdash \triangleright f \longrightarrow (f \longrightarrow (di\ f))$ **by** *auto*
from *7 9* **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndDiFstAndEqvFstAnd*:
$\vdash (\triangleright f \wedge di(\triangleright f \wedge g)) = (\triangleright f \wedge g)$
proof −
have *1*: $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow \triangleright f$
  by *auto*
have *2*: $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow di(\triangleright f \wedge g)$
  by *auto*
have *3*: $\vdash di(\triangleright f \wedge g) = ((\triangleright f \wedge g) \vee di((\triangleright f \wedge g);skip))$
  using *DiEqvOrDiChopSkipA* **by** *blast*
have *4*: $\vdash di((\triangleright f \wedge g);skip) = ((\triangleright f \wedge g);skip);\#True$
  by (*simp add*: *di-d-def*)
have *5*: $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow (\triangleright f \wedge g) \vee ((\triangleright f \wedge g);skip);\#True$
  using *2 3 4* **by** *fastforce*
have *6*: $\vdash \triangleright f \wedge g \longrightarrow f$
  using *FstAndElimL* **by** *fastforce*
hence *7*: $\vdash ((\triangleright f \wedge g);skip);\#True \longrightarrow (f;skip);\#True$
  by (*simp add*: *LeftChopImpChop*)
hence *8*: $\vdash ((\triangleright f \wedge g);skip);\#True \longrightarrow di(f;skip)$
  by (*simp add*: *di-d-def*)
have *9*: $\vdash \triangleright f \longrightarrow \neg(di(f;skip))$
  using *FstImpNotDiChopSkipB* **by** *blast*
have *10*: $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow ((\triangleright f \wedge g) \vee di(f;skip))$
  using *5 8* **by** *fastforce*
have *11*: $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow \neg(di(f;skip)) \wedge ((\triangleright f \wedge g) \vee di(f;skip))$
  using *9 10 1* **by** *fastforce*
have *12*: $\vdash (\neg(di(f;skip)) \wedge ((\triangleright f \wedge g) \vee di(f;skip))) = (\neg(di(f;skip)) \wedge ((\triangleright f \wedge g)))$
  by *auto*
have *13*: $\vdash \triangleright f \wedge di(\triangleright f \wedge g) \longrightarrow (\triangleright f \wedge g)$
  using *11 12* **by** *auto*
have *14*: $\vdash (\triangleright f \wedge g) \longrightarrow \triangleright f$
  by *auto*
hence *15*: $\vdash (\triangleright f \wedge g) \longrightarrow di(\triangleright f \wedge g\ )$
  using *DiIntro* **by** *auto*
have *16*: $\vdash (\triangleright f \wedge g) \longrightarrow \triangleright f \wedge di(\triangleright f \wedge g)$
  using *14 15* **by** *auto*
from *13 16* **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndDiImpBsNotAndDi*:
$\vdash (\triangleright f \wedge di\ g) \longrightarrow (bs\ \neg(di\ f \wedge g))$
proof −
have *1*: $\vdash (\triangleright f \wedge di\ g) \wedge \neg(bs\ \neg(di\ f \wedge g)) \longrightarrow ds(di\ f \wedge g)$
by (*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*: $\vdash ((\triangleright f \vee g) \wedge bs(\neg(\triangleright f)) \wedge bs\ \neg\ g) =$

$((\triangleright f \lor g) \land bs\ (\neg(\triangleright f) \land \neg\ g))$
using *BsAndEqv* **by** *fastforce*
have *13*: $\vdash\ (\neg(\triangleright f) \land \neg\ g) = \neg(\triangleright f \lor g)$
by *auto*
hence *14*: $\vdash bs\ (\neg(\triangleright f) \land \neg\ g) = bs\ \neg(\triangleright f \lor g)$
using *BsEqvRule* **by** *blast*
hence *15*: $\vdash ((\triangleright f \lor g) \land bs\ (\neg(\triangleright f) \land \neg\ g)) = ((\triangleright f \lor g) \land bs\ \neg(\triangleright f \lor g))$
by *auto*
have *16*: $\vdash ((\triangleright f \lor g) \land bs\ \neg(\triangleright f \lor g)) = \triangleright(\triangleright f \lor g)$
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*:
$\vdash \triangleright(f \lor \triangleright g) = \triangleright(f \lor g)$
proof −
have *1*: $\vdash (f \lor \triangleright g) = (\triangleright g \lor f)$ **by** *auto*
hence *2*: $\vdash \triangleright\ (f \lor \triangleright g) = \triangleright(\triangleright g \lor f)$ **using** *FstEqvRule* **by** *blast*
have *3*: $\vdash \triangleright(\triangleright g \lor f) = \triangleright(g \lor f)$ **using** *FstFstOrEqvFstOrL* **by** *blast*
have *4*: $\vdash (g \lor f) = (f \lor g)$ **by** *auto*
hence *5*: $\vdash \triangleright(g \lor f) = \triangleright(f \lor g)$ **using** *FstEqvRule* **by** *blast*
from *2 3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *FstFstOrEqvFstOr*:
$\vdash \triangleright(\triangleright f \lor \triangleright g) = \triangleright(f \lor g)$
proof −
have *1*: $\vdash\ \triangleright(\triangleright f \lor \triangleright g) = \triangleright(f \lor \triangleright g)$ **using** *FstFstOrEqvFstOrL* **by** *blast*
have *2*: $\vdash \triangleright(f \lor \triangleright g) = \triangleright(f \lor g)$ **using** *FstFstOrEqvFstOrR* **by** *blast*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FstLenEqvLen*:
$\vdash \triangleright(\ len(k)) = len(k)$
proof −
have *1*: $\vdash \triangleright(\ len(k) \land \#True) = (len(k) \land \#True)$ **using** *FstLenAndEqvLenAnd* **by** *blast*
have *2*: $\vdash (len(k) \land \#True) = len(k)$ **by** *auto*
hence *3*: $\vdash \triangleright(\ len(k) \land \#True) = \triangleright(len(k))$ **using** *FstEqvRule* **by** *blast*
from *1 2 3* **show** *?thesis* **by** *auto*
qed

lemma *FstSkip*:
$\vdash \triangleright\ skip = skip$
proof −
have *1*: $\vdash skip = len(1)$ **using** *LenOneEqvSkip* **by** *fastforce*
hence *2*: $\vdash \triangleright skip = \triangleright\ (len(1))$ **using** *FstEqvRule* **by** *blast*
have *3*: $\vdash \triangleright(\ len(1)) = len(1)$ **using** *FstLenEqvLen* **by** *blast*
from *1 2 3* **show** *?thesis* **using** *LenOneEqvSkip* **by** *fastforce*
qed

lemma *NotChopNotSkip*:
$\vdash \neg(f;skip) = (empty \lor ((\neg\ f);skip))$
proof –
 have *1*: $\vdash \neg(\neg(\neg f);skip) = (empty \lor ((\neg\ f);skip))$ **using** *NotNotChopSkip* **by** *blast*
 have *2*: $\vdash \neg(\neg(\neg f);skip) = \neg(f;skip)$ **by** *auto*
 from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *BiBoxNotEqvNotTrueChopChopTrue*:
$\vdash bi(\Box\ \neg\ f) = \neg((\#True;f);\#True\ )$
by (*simp add*: *bi-d-def always-d-def di-d-def sometimes-d-def*)

lemma *BoxMoreStateEqvBsFinState*:
$\vdash \Box(more \longrightarrow \neg\ (init\ w)) = bs(\neg(fin(init\ w)))$
proof –
 have *1*: $\vdash \Box(more \longrightarrow \neg\ (init\ w)) = \neg(\Diamond(\neg(more \longrightarrow \neg\ (init\ w))))$
  by (*simp add*: *always-d-def*)
 have *01*: $\vdash \neg(more \longrightarrow \neg\ (init\ w)) = (init\ w \land\ more)$ **by** *auto*
 hence *2*: $\vdash \neg(\Diamond(\neg(more \longrightarrow \neg\ (init\ w)))) = \neg(\#True;(init\ w \land\ more))$
  using *TrueChopEqvDiamond int-eq intensional-simps*(*5*) **by** *force*
 have *3*: $\vdash more = \#True;\ skip$
  using *MoreEqvSkipChopTrue SkipTrueEqvTrueSkip* **by** *fastforce*
 have *4*: $\vdash (init\ w \land\ more) = (init\ w \land (\#True;\ skip))$
  using *3* **by** *auto*
 have *5*: $\vdash (init\ w \land (\#True;\ skip)) = ((init\ w \land empty);(\#True;skip))$
  using *StateAndEmptyChop* **by** *fastforce*
 have *6*: $\vdash (init\ w \land\ more) = ((init\ w \land empty);(\#True;skip))$
  using *4 5* **by** *fastforce*
 have *7*: $\vdash\ (\#True;(init\ w \land\ more)) = (\#True;((init\ w \land empty);(\#True;skip))\ )$
  using *6 RightChopEqvChop* **by** *blast*
 have *8*: $\vdash (\#True;((init\ w \land empty);(\#True;skip))\ ) =$
   $(((\#True;(init\ w \land empty));(\#True;skip))\ )$
  using *ChopAssoc* **by** *blast*
 have *9*: $\vdash (((\#True;(init\ w \land empty));(\#True;skip))\ ) =$
   $((((\#True;(init\ w \land empty));\#True);skip)\ )$
  using *ChopAssoc* **by** *blast*
 have *10*: $\vdash (\#True;(init\ w \land\ more)) =$
   $((((\#True;(init\ w \land empty));\#True);skip)\ )$
  using *7 8 9* **by** *fastforce*
 hence *11*: $\vdash \neg(\#True;(init\ w \land\ more)) =$
   $\neg((((\#True;(init\ w \land empty));\#True);skip)\ )$
  by *auto*
 have *12*: $\vdash \neg((((\#True;(init\ w \land empty));\#True);skip)\ ) =$
   $empty \lor (\neg((\#True;(init\ w \land empty));\#True);skip)$
  using *NotChopNotSkip* **by** *fastforce*
 have *13*: $\vdash (\neg((\#True;(init\ w \land empty));\#True)) = bi(\Box\ \neg(init\ w \land empty))$
  using *BiBoxNotEqvNotTrueChopChopTrue* **by** *fastforce*
 hence *14*: $\vdash (\neg((\#True;(init\ w \land empty));\#True));skip =$
   $(bi(\Box\ \neg(init\ w \land empty)));skip$
  using *RightChopEqvChop* **by** (*simp add*: *LeftChopEqvChop*)

hence *15*: $\vdash$ *empty* $\vee$ ($\neg$((#*True*;(*init w* $\wedge$ *empty*));#*True*));*skip* =
empty $\vee$(*bi*($\Box$ $\neg$(*init w* $\wedge$ *empty*)));*skip*
by *auto*
have *16*: $\vdash$ $\neg$((((#*True*;(*init w* $\wedge$ *empty*));#*True*);*skip*)) =
(*empty* $\vee$ (*bi*($\Box$ $\neg$(*init w* $\wedge$ *empty*));*skip*))
using *12 15* **using** *14 NotChopNotSkip int-eq* **by** *fastforce*
have *171*: $\vdash$ $\neg$(*init w* $\wedge$ *empty*) = ($\neg$(*init w*) $\vee$ $\neg$*empty*)
by *auto*
hence *172*: $\vdash$ $\Box\neg$(*init w* $\wedge$ *empty*) = $\Box$($\neg$(*init w*) $\vee$ $\neg$*empty*)
by (*simp add*: *BoxEqvBox*)
hence *173*: $\vdash$ *bi*($\Box\neg$(*init w* $\wedge$ *empty*)) = *bi*($\Box$($\neg$(*init w*) $\vee$ $\neg$*empty*))
by (*simp add*: *BiEqvBi*)
hence *174*: $\vdash$ *bi*($\Box\neg$(*init w* $\wedge$ *empty*));*skip* = *bi*($\Box$($\neg$(*init w*) $\vee$ $\neg$*empty*));*skip*
using *LeftChopEqvChop* **by** *blast*
hence *17*: $\vdash$ (*empty* $\vee$ (*bi*($\Box$ $\neg$(*init w* $\wedge$ *empty*));*skip*)) =
(*empty* $\vee$ (*bi*($\Box$ ($\neg$(*init w*) $\vee$ $\neg$*empty*));*skip*))
by *auto*
have *181*: $\vdash$ ($\neg$(*init w*) $\vee$ $\neg$*empty*) = ($\neg$*empty* $\vee$ $\neg$(*init w*))
by *auto*
hence *18*: $\vdash$ $\Box$ ($\neg$(*init w*) $\vee$ $\neg$*empty*) = $\Box$ ($\neg$*empty* $\vee$ $\neg$(*init w*))
by (*simp add*: *BoxEqvBox*)
have *191*: $\vdash$ ($\neg$*empty* $\vee$ $\neg$(*init w*)) = (*empty* $\longrightarrow$ $\neg$(*init w*))
by *auto*
hence *19*: $\vdash$ $\Box$ ($\neg$*empty* $\vee$ $\neg$(*init w*)) = $\Box$(*empty* $\longrightarrow$ $\neg$(*init w*))
by (*simp add*: *BoxEqvBox*)
have *20*: $\vdash$ $\Box$(*empty* $\longrightarrow$ $\neg$(*init w*)) = *fin* ($\neg$(*init w*))
by (*simp add*: *fin-d-def*)
have *21*: $\vdash$ *fin* ($\neg$(*init w*)) = $\neg$(*fin* (*init w*))
using *FinEqvFin FinNotStateEqvNotFinState Initprop*(*2*) **by** *fastforce*
have *22*: $\vdash$ *bi*($\Box$ ($\neg$(*init w*) $\vee$ $\neg$*empty*)) = *bi* ($\neg$(*fin* (*init w*)))
using *18 19 20 21 BiEqvBi* **by** (*metis int-eq*)
hence *23*: $\vdash$ (*bi*($\Box$ ($\neg$(*init w*) $\vee$ $\neg$*empty*)));*skip* = (*bi* ($\neg$(*fin* (*init w*))));*skip*
using *RightChopEqvChop* **by** (*simp add*: *LeftChopEqvChop*)
hence *24*: $\vdash$ (*empty* $\vee$ (*bi*($\Box$ ($\neg$(*init w*) $\vee$ $\neg$*empty*)));*skip*) =
(*empty* $\vee$ (*bi* ($\neg$(*fin* (*init w*))));*skip*)
by *auto*
hence *25*: $\vdash$ (*empty* $\vee$ (*bi* ($\neg$(*fin* (*init w*))));*skip*) = *bs*($\neg$(*fin* (*init w*)))
by (*simp add*:*bs-d-def*)
from *1 2 11 16 17 24 25* **show** *?thesis* **by** *fastforce*
qed

lemma *HaltStateEqvFstFinState*:
$\vdash$ *halt* (*init w*) = $\rhd$ (*fin* (*init w*))
proof –
have *1*: $\vdash$ *halt* (*init w*) = $\Box$(*empty* = (*init w*)) **by** (*simp add*: *halt-d-def*)
have *21*: $\vdash$ (*empty* = (*init w*)) = (((*empty* $\longrightarrow$ (*init w*)) $\wedge$ ((*init w*) $\longrightarrow$ *empty*)))
by *auto*
hence *2*: $\vdash$ $\Box$(*empty* = (*init w*)) = ($\Box$((*empty* $\longrightarrow$ (*init w*)) $\wedge$ ((*init w*) $\longrightarrow$ *empty*)))
by (*simp add*: *BoxEqvBox*)
have *3*: $\vdash$ ($\Box$((*empty* $\longrightarrow$ (*init w*)) $\wedge$ ((*init w*) $\longrightarrow$ *empty*))) =

$(\Box((empty \longrightarrow (init\ w))) \wedge \Box((init\ w) \longrightarrow empty))$
by (*metis 21 BoxAndBoxEqvBoxRule int-eq*)
have *4*: $\vdash ((init\ w) \longrightarrow empty) = (more \longrightarrow \neg(init\ w))$
by (*simp add*: *empty-d-def* ,*auto*)
hence *5*: $\vdash \Box\ ((init\ w) \longrightarrow empty) = \Box\ (more \longrightarrow \neg(init\ w))$ **using** *BoxEqvBox* **by** *blast*
have *6*: $\vdash \Box\ (more \longrightarrow \neg(init\ w)) = bs(\neg(fin(init\ w)))$ **using** *BoxMoreStateEqvBsFinState* **by** *blast*
have *7*: $\vdash \Box((empty \longrightarrow (init\ w))) = fin(init\ w)$ **by** (*simp add*: *fin-d-def*)
have *8*: $\vdash\ (\Box((empty \longrightarrow (init\ w))) \wedge \Box((init\ w) \longrightarrow empty)) =$
$(fin(init\ w) \wedge bs(\neg(fin(init\ w))))$ **using** *5 6 7* **by** *fastforce*
from *1 2 3 8* **show** *?thesis* **by** (*metis first-d-def inteq-reflection*)
qed

lemma *FstLenEqvLenFst*:
$\vdash \rhd(len\ k\ ;\ f) = len\ k\ ;\ \rhd\ f$
proof −
have *1*: $\vdash len\ k\ ;\ f = \rhd(len\ k)\ ;\ f$ **using** *FstLenEqvLen LeftChopEqvChop* **by** *fastforce*
have *2*: $\vdash \rhd(len\ k\ ;\ f) = \rhd\ (\rhd(len\ k)\ ;\ f)$ **using** *1 FstEqvRule* **by** *blast*
have *3*: $\vdash \rhd\ (\rhd(len\ k)\ ;\ f) = \rhd(len\ k)\ ;\ \rhd f$ **using** *FstFstChopEqvFstChopFst* **by** *blast*
have *4*: $\vdash \rhd(len\ k)\ ;\ \rhd f = len\ k\ ;\ \rhd f$ **using** *FstLenEqvLen LeftChopEqvChop* **by** *fastforce*
from *2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *FstNextEqvNextFst*:
$\vdash \rhd(\bigcirc\ f) = \bigcirc(\ \rhd\ f)$
proof −
have *1*: $\vdash \rhd(\bigcirc\ f) = \rhd(skip\ ;\ f)$ **using** *FstEqvRule* **by** (*simp add*: *next-d-def*)
have *2*: $\vdash skip\ ;\ f = \rhd skip\ ;\ f$ **using** *FstSkip* **using** *LeftChopEqvChop* **by** *fastforce*
have *3*: $\vdash \rhd(skip\ ;\ f) = \rhd\ (\rhd skip\ ;\ f)$ **using** *2 FstEqvRule LeftChopEqvChop* **by** *blast*
have *4*: $\vdash \rhd\ (\rhd skip\ ;\ f) = \rhd skip\ ;\ \rhd f$ **using** *3 FstFstChopEqvFstChopFst* **by** *blast*
have *5*: $\vdash \rhd skip\ ;\ \rhd f = skip\ ;\ \rhd f$ **using** *4 FstSkip LeftChopEqvChop* **by** *blast*
have *6*: $\vdash skip\ ;\ \rhd f\ =\ \bigcirc(\ \rhd\ f)$ **by** (*simp add*: *next-d-def*)
from *1 2 3 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *FstDiamondStateEqvHalt*:
$\vdash \rhd\ (\Diamond\ (init\ w)) = halt\ (init\ w)$
proof −
have *1*: $\vdash \Diamond\ (init\ w) = \Diamond\ ((init\ w) \wedge \#True)$ **by** *simp*
have *2*: $\vdash fin\ (init\ w)\ ;\ \#True = \Diamond\ ((init\ w) \wedge \#True)$ **using** *1 FinChopEqvDiamond* **by** *blast*
have *3*: $\vdash fin\ (init\ w)\ ;\ \#True = di\ (fin\ (init\ w))$ **by** (*simp add*: *di-d-def*)
have *4*: $\vdash (\Diamond\ (init\ w)) = (di\ (fin\ (init\ w)))$ **using** *1 2 3* **by** *fastforce*
have *5*: $\vdash \rhd\ (\Diamond\ (init\ w)) = \rhd\ (di\ (fin\ (init\ w)))$ **using** *4 FstEqvRule* **by** *blast*
hence *6*: $\vdash \rhd\ (\Diamond\ (init\ w)) = \rhd\ (fin\ (init\ w))$ **using** *FstDiEqvFst* **by** *fastforce*
hence *7*: $\vdash \rhd\ (\Diamond\ (init\ w)) = halt\ (init\ w)$ **using** *HaltStateEqvFstFinState* **by** *fastforce*
from *7* **show** *?thesis* **by** *simp*
qed

lemma *FstBoxStateEqvStateAndEmpty*:
$\vdash \rhd\ (\Box\ (init\ w)) = ((init\ w) \wedge empty)$
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*))* = $\rhd$ (*init w*) ; ($\Box$ (*init w*))*
using *FstState* **by** (*metis AndChopCommute int-eq*)
have *8*: $\vdash$ $\Box$ (*init w*) = $\rhd$ (*init w*) ; ($\Box$ (*init w*))*
using *6 7* **by** *fastforce*
have *9*: $\vdash$ $\rhd$ ($\Box$ (*init w*)) = $\rhd$ ($\rhd$ (*init w*) ; ($\Box$ (*init w*))*)
using *8 FstEqvRule* **by** *blast*
have *10*: $\vdash$ $\rhd$ ($\rhd$ (*init w*) ; ($\Box$ (*init w*))*) = $\rhd$ (*init w*) ; $\rhd$ (($\Box$ (*init w*))*)
using *FstFstChopEqvFstChopFst* **by** *blast*
have *11*: $\vdash$ $\rhd$ (*init w*) ; $\rhd$ (($\Box$ (*init w*))*) = $\rhd$ (*init w*) ; *empty*
using *RightChopEqvChop FstCSEqvEmpty* **by** *blast*
have *12*: $\vdash$ $\rhd$ (*init w*) ; *empty* = $\rhd$ (*init w*)
using *RightChopEqvChop ChopEmpty* **by** *blast*
have *13*: $\vdash$ $\rhd$ (*init w*) = ((*init w*) $\wedge$ *empty*)
using *FstState* **by** *fastforce*
from *9 10 11 12 13* **show** *?thesis* **by** *fastforce*
qed

end

theory *Monitor*
imports *First*

begin

8 Monitors

The RV monitors language is introduced plus the algebraic properties of the monitor operators.

8.1 Syntax

sledgehammer-params [*minimize=true*,*preplay-timeout=10*,*timeout=60*,*verbose=true*,
provers=cvc4 e z3 vampire spass]

declare [[*show-types*]]

datatype ('a ::world) monitor =
  mFIRST-d 'a formula ((FIRST -) [84] 83)
 | mUPTO-d 'a monitor 'a monitor ((- UPTO -) [84,84] 83)
 | mTHRU-d 'a monitor 'a monitor ((- THRU -) [84,84] 83)
 | mTHEN-d 'a monitor 'a monitor ((- THEN -) [84,84] 83)
 | mWITH-d 'a monitor 'a formula ((- WITH -) [84,84] 83)

fun MON :: ('a::world) monitor ⇒ 'a formula
where (MON (FIRST f)) = LIFT(▷ f)
  | (MON (a UPTO b)) = LIFT(▷((MON a) ∨ (MON b)))
  | (MON (a THRU b)) = LIFT(▷(di(MON a) ∧ di(MON b)))
  | (MON (a THEN b)) = LIFT((MON a);(MON b))
  | (MON (a WITH f)) = LIFT((MON a) ∧ f)

syntax
 -MON :: 'a monitor ⇒ lift (($\mathcal{M}$ -) [80] 80)

translations
 -MON == CONST MON

8.2 Derived Monitors

definition HALT-d :: ('a :: world) formula ⇒ 'a monitor
where HALT-d w ≡ FIRST(LIFT(fin (init w)))

definition LEN-d :: nat ⇒ ('a ::world) monitor
where
 LEN-d k ≡ FIRST (LIFT(len k))

definition EMPTY-d :: ('a:: world) monitor
where
 EMPTY-d ≡ FIRST (LIFT(empty))

definition SKIP-d :: ('a:: world) monitor
where
 SKIP-d ≡ FIRST (LIFT (skip))

syntax
 -HALT-d :: lift ⇒ 'a monitor ((HALT -) [84] 83)
 -LEN-d :: nat ⇒ 'a monitor ((LEN -) [84] 83)
 -EMPTY-d :: 'a monitor ((EMPTY))
 -SKIP-d :: 'a monitor ((SKIP))

syntax (ASCII)
 -HALT-d :: lift ⇒ 'a monitor ((HALT -) [84] 83)
 -LEN-d :: nat ⇒ 'a monitor ((LEN -) [84] 83)
 -EMPTY-d :: 'a monitor ((EMPTY))
 -SKIP-d :: 'a monitor ((SKIP))

translations
-HALT-d ⇌ *CONST HALT-d*
-LEN-d ⇌ *CONST LEN-d*
-EMPTY-d ⇌ *CONST EMPTY-d*
-SKIP-d ⇌ *CONST SKIP-d*

definition *GUARD-d* :: ('a::world) formula ⇒ 'a monitor
where
GUARD-d w ≡ (*EMPTY WITH LIFT*(*init w*))

primrec *TIMES-d* :: ('a :: world) monitor ⇒ nat ⇒ '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 ⇒ 'a monitor ((*GUARD* -) [84] 83)
-TIMES-d :: ['a monitor,nat] ⇒ 'a monitor ((- *TIMES* -) [84,84] 83)

syntax (*ASCII*)
-GUARD-d :: lift ⇒ 'a monitor ((*GUARD* -) [84] 83)
-TIMES-d :: ['a monitor,nat] ⇒ 'a monitor ((- *TIMES* -) [84,84] 83)

translations
-GUARD-d ⇌ *CONST GUARD-d*
-TIMES-d ⇌ *CONST TIMES-d*

definition *FAIL-d* :: ('a:: world) monitor
where
FAIL-d ≡ *GUARD* (#*False*)

definition *ALWAYS-d* :: ('a :: world) monitor ⇒ 'a formula ⇒ 'a monitor
where
ALWAYS-d a w ≡ (*a WITH LIFT*((*bi* (*fin* (*init w*)))))

definition *SOMETIME-d* :: ('a :: world) monitor ⇒ 'a formula ⇒ 'a monitor
where
SOMETIME-d a w ≡ (*a WITH LIFT*((*di* (*fin* (*init w*)))))

definition *LIMIT-d* :: ('a :: world) formula ⇒ 'a formula
where
LIMIT-d f ≡ *LIFT*(*bs* (¬ *f*))

definition *UNTIL-d* :: ('a :: world) formula ⇒ 'a formula ⇒ 'a monitor
where
UNTIL-d w1 w2 ≡ (*HALT w2*) *WITH* (*LIFT*(*bm w1*))

syntax

-FAIL-d :: 'a monitor (FAIL)
-ALWAYS-d :: ['a monitor,lift] ⇒ 'a monitor ((- ALWAYS -) [84,84] 83)
-SOMETIME-d :: ['a monitor,lift] ⇒ 'a monitor ((- SOMETIME -) [84,84] 83)
-LIMIT-d :: lift ⇒ lift ((Limit -) [84] 83)
-UNTIL-d :: [lift,lift] ⇒ 'a monitor ((- UNTIL -) [84,84] 83)

syntax (ASCII)
-FAIL-d :: 'a monitor (FAIL)
-ALWAYS-d :: ['a monitor,lift] ⇒ 'a monitor ((- ALWAYS -) [84,84] 83)
-SOMETIME-d :: ['a monitor,lift] ⇒ 'a monitor ((- SOMETIME -) [84,84] 83)
-LIMIT-d :: lift ⇒ lift ((Limit -) [84] 83)
-UNTIL-d :: [lift,lift] ⇒ 'a monitor ((- UNTIL -) [84,84] 83)

translations
-FAIL-d ⇌ CONST FAIL-d
-ALWAYS-d ⇌ CONST ALWAYS-d
-SOMETIME-d ⇌ CONST SOMETIME-d
-LIMIT-d ⇌ CONST LIMIT-d
-UNTIL-d ⇌ CONST UNTIL-d

definition WITHIN-d :: ('a :: world) monitor ⇒ 'a formula ⇒ 'a monitor
where
WITHIN-d a f ≡ (a WITH LIFT(Limit f))

syntax
-WITHIN-d :: ['a monitor,lift] ⇒ 'a monitor ((- WITHIN -) [84,84] 83)

syntax (ASCII)
-WITHIN-d :: ['a monitor,lift] ⇒ 'a monitor ((- WITHIN -) [84,84] 83)

translations
-WITHIN-d ⇌ CONST WITHIN-d

definition AND-d :: ('a :: world) monitor ⇒ 'a monitor ⇒ 'a monitor
where
AND-d a b ≡ (a WITH LIFT($\mathcal{M}$ b))

definition ITERATE-d :: ('a :: world) monitor ⇒ 'a monitor ⇒ 'a monitor
where
ITERATE-d a b ≡ (a WITH (LIFT ($\mathcal{M}$ b)*))

syntax
-AND-d :: ['a monitor,'a monitor] ⇒ 'a monitor ((- AND -) [84,84] 83)
-ITERATE-d :: ['a monitor,'a monitor] ⇒ 'a monitor ((- ITERATE -) [84,84] 83)

syntax (ASCII)
-AND-d :: ['a monitor,'a monitor] ⇒ 'a monitor ((- AND -) [84,84] 83)

-ITERATE-d :: ['*a monitor*,'*a monitor*] ⇒ '*a monitor* ((- *ITERATE* -) [*84*,*84*] *83*)

translations
-AND-d ⇌ *CONST AND-d*
-ITERATE-d ⇌ *CONST ITERATE-d*

definition *STAR-d* :: ('*a* :: *world*) *monitor* ⇒ '*a formula* ⇒ '*a monitor*
where
STAR-d a f ≡ ((*FIRST LIFT*(◇ *f*)) *ITERATE* (*a*))

definition *REPEAT-d* :: ('*a* :: *world*) *monitor* ⇒ '*a formula* ⇒ '*a monitor*
where
REPEAT-d a w ≡ (((*HALT w*) *ITERATE* (*a WITH LIFT*(*keep*(¬ (*init w*))))) *THEN a*)

syntax
-STAR-d :: ['*a monitor*,*lift*] ⇒ '*a monitor* ((- *STAR* -) [*84*,*84*] *83*)
-REPEAT-d :: ['*a monitor*,*lift*] ⇒ '*a monitor* ((- *REPEATUNTIL* -) [*84*,*84*] *83*)

syntax (*ASCII*)
-STAR-d :: ['*a monitor*,*lift*] ⇒ '*a monitor* ((- *STAR* -) [*84*,*84*] *83*)
-REPEAT-d :: ['*a monitor*,*lift*] ⇒ '*a monitor* ((- *REPEATUNTIL* -) [*84*,*84*] *83*)

translations
-STAR-d ⇌ *CONST STAR-d*
-REPEAT-d ⇌ *CONST REPEAT-d*

8.3 Monitor Laws

lemma *MFixFst*:
⊢ ($\mathcal{M}$ *a*) = ▷ ($\mathcal{M}$ *a*)
proof
(*induct a*)
case (*mFIRST-d x*)
then show *?case*
proof −
have *1*: ⊢ ($\mathcal{M}$ (*FIRST x*)) = ▷ *x* **by** *simp*
have *2*: ⊢ ▷ *x* = ▷ (▷ *x*) **using** *FstFixFst* **by** *fastforce*
have *3*: ⊢ ▷ (▷ *x*) = ▷($\mathcal{M}$ (*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*: ⊢ ($\mathcal{M}$ (*a1 UPTO a2*)) = ▷(($\mathcal{M}$ *a1*) ∨ ($\mathcal{M}$ *a2*))
by (*simp*)
have *2*: ⊢ ▷(($\mathcal{M}$ *a1*) ∨ ($\mathcal{M}$ *a2*)) = ▷(▷(($\mathcal{M}$ *a1*) ∨ ($\mathcal{M}$ *a2*)))
using *FstFixFst* **by** *fastforce*
have *3*: ⊢ ▷(▷(($\mathcal{M}$ *a1*) ∨ ($\mathcal{M}$ *a2*))) = ▷($\mathcal{M}$ (*a1 UPTO a2*))

using *2* **by** *simp*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed
next
case (*mTHRU-d a1 a2*)
then show *?case*
proof −
have *1*: $\vdash (\mathcal{M}\ (a1\ THRU\ a2)) = \triangleright(\ di\ (\mathcal{M}\ a1) \wedge di(\mathcal{M}\ a2))$
by (*simp*)
have *2*: $\vdash \triangleright(\ di\ (\mathcal{M}\ a1) \wedge di(\mathcal{M}\ a2)) = \triangleright(\triangleright(di(\mathcal{M}\ a1) \wedge di(\mathcal{M}\ a2)))$
using *FstFixFst* **by** *fastforce*
have *3*: $\vdash \triangleright(\triangleright(\ di\ (\mathcal{M}\ a1) \wedge di(\mathcal{M}\ a2))) = \triangleright(\mathcal{M}\ (a1\ THRU\ a2))$
using *2* **by** *simp*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed
next
case (*mTHEN-d a1 a2*)
then show *?case*
proof −
have *1*: $\vdash (\mathcal{M}\ (a1\ THEN\ a2)) = (\mathcal{M}\ a1)\ ;\ (\mathcal{M}\ a2)$
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*: $\vdash \triangleright((\mathcal{M}\ a1)\ ;\ (\mathcal{M}\ a2)) = \triangleright(\mathcal{M}\ (a1\ THEN\ a2))$
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*: $\vdash (\mathcal{M}\ (a\ WITH\ x2)) = ((\mathcal{M}\ a) \wedge (\ x2))$
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*: $\vdash \triangleright((\mathcal{M}\ a) \wedge (\ x2)) = \triangleright(\mathcal{M}\ (a\ WITH\ x2))$
using *4* **by** *simp*
from *1 2 3 4 5* **show** *?thesis* **by** (*metis inteq-reflection*)
qed
qed

lemma *MGuardFalseEqvFalse*:

$\vdash\ \mathcal{M}(GUARD\ \#False) = \#False$
proof −
  have *1*: $\vdash \mathcal{M}(GUARD\ \#False) = \mathcal{M}(EMPTY\ WITH\ LIFT(init\ \#False))$ **by** (*simp add*: *GUARD-d-def*)
  have *2*: $\vdash \mathcal{M}(EMPTY\ WITH\ LIFT(init\ \#False)) = (\mathcal{M}(EMPTY) \wedge (init\ \#False))$ **by** (*simp*)
  have *3*: $\vdash \#False = (init\ \#False)$ **by** (*simp add*:*init-defs Valid-def*)
  have *4*: $\vdash (\mathcal{M}(EMPTY) \wedge (init\ \#False)) = (\mathcal{M}(EMPTY) \wedge \#False)$ **using** *3* **by** *auto*
  have *5*: $\vdash (\mathcal{M}(EMPTY) \wedge \#False) = \#False$ **by** *simp*
  have *6*: $\vdash (\mathcal{M}(EMPTY) \wedge (init\ \#False)) = \#False$ **using** *4 5* **by** *simp*
  have *7*: $\vdash \mathcal{M}(EMPTY\ WITH\ LIFT(init\ \#False)) = \#False$ **using** *2 6* **by** *fastforce*
  have *8*: $\vdash \mathcal{M}(GUARD\ \#False) = \#False$ **using** *1 7* **by** *fastforce*
  from *8* **show** *?thesis* **by** *auto*
qed

lemma *MFirstFalseEqvFalse*:
$\vdash\ \mathcal{M}(FIRST\ LIFT\ \#False) = \#False$
proof −
  have *1*: $\vdash \mathcal{M}(FIRST\ LIFT\ \#False) = \rhd\ \#False$ **by** (*simp*)
  have *2*: $\vdash \mathcal{M}(FIRST\ LIFT\ \#False) = \#False$ **using** *FstFalse* **by** *fastforce*
 from *2* **show** *?thesis* **by** *auto*
qed

lemma *MFailAlt*:
$\vdash\ \mathcal{M}\ FAIL = \#False$
proof −
  have *1*: $\vdash\ \mathcal{M}\ FAIL =\ \mathcal{M}\ (GUARD\ (\#False))$ **by** (*simp add*: *FAIL-d-def*)
  have *2*: $\vdash \mathcal{M}(GUARD\ (\#False)) = \#False$ **using** *MGuardFalseEqvFalse* **by** *auto*
 from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *MFailEqvFirstFalseWithinEmpty*:
$\vdash \mathcal{M}\ FAIL = \mathcal{M}\ ((FIRST\ LIFT\ \#False)\ WITHIN\ empty)$
proof −
  have *1*: $\vdash \mathcal{M}\ (\ (FIRST\ LIFT\ \#False)\ WITHIN\ (\ empty\ )) =$
    $\mathcal{M}((FIRST\ LIFT\ \#False)\ WITH\ LIFT(Limit\ empty)\ )$
    by (*simp add*: *WITHIN-d-def*)
  have *2*: $\vdash\ \mathcal{M}((FIRST\ LIFT\ \#False)\ WITH\ LIFT(Limit\ empty)\ ) =$
    $(\mathcal{M}(FIRST\ LIFT\ \#False) \wedge (Limit\ empty\ ))$
    by (*simp*)
  have *3*: $\vdash\ \mathcal{M}((FIRST\ LIFT\ \#False)\ WITH\ LIFT(Limit\ empty)\ ) = \#False$
    using *MFirstFalseEqvFalse* **by** *auto*
  have *4*: $\vdash \mathcal{M}(\ (FIRST\ LIFT\ \#False)\ WITHIN\ (\ empty\ )) = \#False$
    using *1 3* **by** *fastforce*
  have *5*: $\vdash \mathcal{M}(\ FAIL) = \#False$
    using *MFailAlt* **by** *simp*
 from *4 5* **show** *?thesis* **by** *fastforce*
 qed

lemma *MEmptyAlt*:
$\vdash\ \mathcal{M}\ EMPTY = empty$
proof −

have *1*: $\vdash \mathcal{M}$ (*EMPTY*) = $\mathcal{M}$((*FIRST LIFT empty*)) **by** (*simp add*: *EMPTY-d-def*)
have *2*: $\vdash \mathcal{M}$ ((*FIRST LIFT empty*)) = $\triangleright$ *empty* **by** (*simp*)
have *3*: $\vdash \triangleright$ *empty* = *empty* **using** *FstEmpty* **by** *auto*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MSkipAlt*:
$\vdash$ $\mathcal{M}$ *SKIP* = *skip*
proof −
have *1*: $\vdash$ $\mathcal{M}$ *SKIP* = $\mathcal{M}$ (*FIRST LIFT skip*) **by** (*simp add*: *SKIP-d-def*)
have *2*: $\vdash \mathcal{M}$ (*FIRST LIFT skip*) = $\triangleright$ *skip* **by** (*simp*)
have *3*: $\vdash \triangleright$ *skip* = *skip* **using** *FstSkip* **by** *simp*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MGuardAlt*:
$\vdash$ $\mathcal{M}$ (*GUARD*(*w*)) = (*empty* $\wedge$ *init w*)
proof −
have *1*: $\vdash$ $\mathcal{M}$(*GUARD*(*w*)) = $\mathcal{M}$(*EMPTY WITH* (*LIFT* (*init w*))) **by** (*simp add*:*GUARD-d-def*)
have *2*: $\vdash \mathcal{M}$(*EMPTY WITH* (*LIFT*(*init w*))) = ($\mathcal{M}$(*EMPTY*) $\wedge$ (*init w*)) **by** (*simp*)
have *3*: $\vdash$ ($\mathcal{M}$(*EMPTY*) $\wedge$ (*init w*)) = (*empty* $\wedge$ (*init w*)) **using** *MEmptyAlt* **by** *fastforce*
have *4*: $\vdash$ (*empty* $\wedge$ (*init w*)) = (*empty* $\wedge$ *init w*) **by** *simp*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *MLengthAlt*:
$\vdash \mathcal{M}$ (*LEN*(*k*)) = *len*(*k*)
proof −
have *1*: $\vdash \mathcal{M}$(*LEN*(*k*)) = $\mathcal{M}$(*FIRST LIFT*(*len*(*k*))) **by** (*simp add*:*LEN-d-def*)
have *2*: $\vdash \mathcal{M}$(*FIRST LIFT*(*len*(*k*))) = $\triangleright$(*len*(*k*)) **by** (*simp*)
have *3*: $\vdash \triangleright$(*len*(*k*)) = *len*(*k*) **using** *FstLenEqvLen* **by** *blast*
from *1 2 3* **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*: ⊢ *init* (¬ *w*) = ¬ (*init w*)
using *Initprop*(*2*) **by** *fastforce*
have *9*: ⊢ *fin* (*init* (¬ *w*)) = *fin* (¬ (*init w*))
using *8 FinEqvFin* **by** *blast*
have *10*: ⊢ *fin* (*init* (¬ *w*)) = ¬ (*fin* (*init w*))
using *8 FinNotStateEqvNotFinState FinEqvFin* **by** *blast*
have *11*: ⊢ ¬ (*fin* (*init* (¬ *w*))) = (*fin* (*init w*))
using *10* **by** *fastforce*
have *12*: ⊢ *bi* (¬ (*fin* (*init* (¬ *w*)))) = *bi* (*fin* (*init w*))
using *11* **by** (*simp add*: *BiEqvBi*)
have *13*: ⊢ □ (*init w*) = *bi* (*fin* (*init w*))
using *7 12* **by** *fastforce*
from *13* **show** *?thesis* **by** *simp*
qed

lemma *MAlwaysAlt*:
⊢ $\mathcal{M}$(*a ALWAYS w*) = ($\mathcal{M}$(*a*) ∧ □ (*init w*))
proof −
have *1*: ⊢ $\mathcal{M}$(*a ALWAYS w*) = $\mathcal{M}$(*a WITH LIFT*(*bi* (*fin* (*init w*))))
by (*simp add*: *ALWAYS-d-def*)
have *2*: ⊢ $\mathcal{M}$(*a WITH LIFT*(*bi* (*fin* (*init w*)))) = ($\mathcal{M}$(*a*) ∧ (*bi* (*fin* (*init w*))))
by (*simp*)
have *3*: ⊢ ($\mathcal{M}$(*a*) ∧ (*bi* (*fin* (*init w*)))) = ($\mathcal{M}$(*a*) ∧ □ (*init w*))
using *BoxStateEqvBiFinState* **by** *fastforce*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DiamondStateEqvDiFinState*:
⊢ ◇ (*init w*) = *di* (*fin* (*init w*))
proof −
have *1*: ⊢ □ (*init* (¬ *w*)) = *bi* (*fin* (*init* (¬ *w*)))
using *BoxStateEqvBiFinState* **by** *blast*
have *2*: ⊢ ¬ (□ (*init* (¬ *w*))) = ¬ (*bi* (*fin* (*init* (¬ *w*))))
using *1* **by** *auto*
have *3*: ⊢ ◇ (¬ (*init* (¬ *w*))) = *di* (¬ (*fin* (*init* (¬ *w*))))
using *2* **by** (*simp add*: *always-d-def bi-d-def*)
have *4*: ⊢ ◇ (*init w*) = *di* (¬ (*fin* (*init* (¬ *w*))))
using *3 Initprop*(*2*) **by** (*metis int-eq intensional-simps*(*4*))
have *5*: ⊢ ◇ (*init w*) = *di* (*fin* (*init w*)) **using** *4 FinNotStateEqvNotFinState*
by (*metis DiEqvNotBiNot DiNotEqvNotBi inteq-reflection*)
from *1 2 3 4 5* **show** *?thesis* **by** *simp*
qed

lemma *MSometimeAlt*:
⊢ $\mathcal{M}$(*a SOMETIME w*) = ($\mathcal{M}$(*a*) ∧ ◇ (*init w*))
proof −
have *1*: ⊢ $\mathcal{M}$(*a SOMETIME w*) = $\mathcal{M}$(*a WITH LIFT*(*di* (*fin* (*init w*))))
by (*simp add*: *SOMETIME-d-def*)
have *2*: ⊢ $\mathcal{M}$(*a WITH LIFT*(*di* (*fin* (*init w*)))) = ($\mathcal{M}$(*a*) ∧ (*di* (*fin* (*init w*))))

```
      by (simp)
  have 3: ⊢ 𝓜(a WITH LIFT(di (fin (init w)))) = (𝓜(a) ∧  ◇ (init w))
      using DiamondStateEqvDiFinState  by fastforce
  from 1 2 3 show ?thesis by fastforce
qed

lemma MWithinAlt:
⊢ 𝓜(a WITHIN f) = (𝓜(a) ∧ (bs (¬ f)))
proof −
  have 1: ⊢ 𝓜(a WITHIN f) = 𝓜(a WITH LIFT(bs (¬ f)))
      by (simp add: WITHIN-d-def LIMIT-d-def)
  have 2: ⊢ 𝓜(a WITH LIFT(bs (¬ f))) = (𝓜(a) ∧ (bs (¬ f)))
      by (simp)
  from 1 2 show ?thesis by fastforce
qed

lemma MTimesAlt:
⊢ 𝓜(a TIMES k) = power (𝓜(a)) k
proof
 (induct k)
 case 0
 then show ?case
  proof −
   have 1: ⊢  𝓜 (a TIMES 0) =  𝓜 EMPTY by simp
   have 2: ⊢  𝓜 EMPTY = empty  using MEmptyAlt by simp
   have 3: ⊢ empty = power (𝓜 a) 0 by simp
   from 1 2 3 show ?thesis by auto
  qed
 next
 case (Suc k)
 then show ?case
  proof −
   have 1: ⊢  𝓜( a TIMES Suc k) = 𝓜(a THEN (a TIMES k) )
       by simp
   have 2: ⊢ 𝓜(a THEN (a TIMES k) ) = (𝓜 a);(𝓜 (a TIMES k))
       by (simp)
   have 3: ⊢ (𝓜 a);(𝓜(a TIMES k)) = (𝓜 a);(power (𝓜 a) k)
       using RightChopEqvChop Suc.hyps by blast
   have 4: ⊢ (𝓜 a);(power (𝓜 a) k) = power (𝓜 a) (Suc k)
       by simp
   from 1 2 3 4 show ?thesis by fastforce
  qed
 qed

lemma MUptoAlt:
⊢ 𝓜(a UPTO b) = (( (𝓜 a) ∧ bi ¬(𝓜 b)) ∨ ((𝓜 b) ∧ bi ¬(𝓜 a)) ∨ ((𝓜 a) ∧ (𝓜 b)))
proof −
  have 1: ⊢ 𝓜 (a UPTO b) = ▷((𝓜 a) ∨ (𝓜 b))
      by (simp)
  have 2: ⊢ ▷((𝓜 a) ∨ (𝓜 b)) = ((▷(𝓜 a) ∧  (bs ¬(𝓜 b))) ∨ (▷(𝓜 b) ∧  (bs ¬(𝓜 a))))
```

using *FstWithOrEqv* **by** *blast*
have *3*: $\vdash ((\triangleright(\mathcal{M}\ a) \land\ (bs\ \neg(\mathcal{M}\ b))) \lor (\triangleright(\mathcal{M}\ b) \land\ (bs\ \neg(\mathcal{M}\ a)))) =$
$(((\mathcal{M}\ a) \land ((\mathcal{M}\ b) \lor \neg(\mathcal{M}\ b)) \land\ (bs\ \neg(\mathcal{M}\ b))) \lor$
$((\mathcal{M}\ b) \land ((\mathcal{M}\ a) \lor \neg(\mathcal{M}\ a)) \land\ (bs\ \neg(\mathcal{M}\ a))))$
using *MFixFst* **by** *fastforce*
have *4*: $\vdash (((\mathcal{M}\ a) \land ((\mathcal{M}\ b) \lor \neg(\mathcal{M}\ b)) \land\ (bs\ \neg(\mathcal{M}\ b))) \lor$
$((\mathcal{M}\ b) \land ((\mathcal{M}\ a) \lor \neg(\mathcal{M}\ a)) \land\ (bs\ \neg(\mathcal{M}\ a)))) =$
$(((\mathcal{M}\ a) \land (\ ((\mathcal{M}\ b) \land bs\neg(\mathcal{M}\ b)) \lor (\neg(\mathcal{M}\ b) \land bs\neg(\mathcal{M}\ b)))\ ) \lor$
$((\mathcal{M}\ b) \land (\ ((\mathcal{M}\ a) \land bs\neg(\mathcal{M}\ a)) \lor (\neg(\mathcal{M}\ a) \land bs\neg(\mathcal{M}\ a)))\ ))$
by *auto*
have *5*: $\vdash (((\mathcal{M}\ a) \land (\ ((\mathcal{M}\ b) \land bs\neg(\mathcal{M}\ b)) \lor (\neg(\mathcal{M}\ b) \land bs\neg(\mathcal{M}\ b)))\ ) \lor$
$((\mathcal{M}\ b) \land (\ ((\mathcal{M}\ a) \land bs\neg(\mathcal{M}\ a)) \lor (\neg(\mathcal{M}\ a) \land bs\neg(\mathcal{M}\ a)))\ )) =$
$(((\mathcal{M}\ a) \land (\ (\triangleright(\mathcal{M}\ b)) \lor (\neg(\mathcal{M}\ b) \land bs\neg(\mathcal{M}\ b)))\ ) \lor$
$((\mathcal{M}\ b) \land (\ (\triangleright(\mathcal{M}\ a)) \lor (\neg(\mathcal{M}\ a) \land bs\neg(\mathcal{M}\ a)))\ ))$
by (*simp add*: *first-d-def*)
have *6*: $\vdash (((\mathcal{M}\ a) \land (\ (\triangleright(\mathcal{M}\ b)) \lor (\neg(\mathcal{M}\ b) \land bs\neg(\mathcal{M}\ b)))\ ) \lor$
$((\mathcal{M}\ b) \land (\ (\triangleright(\mathcal{M}\ a)) \lor (\neg(\mathcal{M}\ a) \land bs\neg(\mathcal{M}\ a)))\ )) =$
$(((\mathcal{M}\ a) \land (\ ((\mathcal{M}\ b)) \lor (\neg(\mathcal{M}\ b) \land bs\neg(\mathcal{M}\ b)))\ ) \lor$
$((\mathcal{M}\ b) \land (\ ((\mathcal{M}\ a)) \lor (\neg(\mathcal{M}\ a) \land bs\neg(\mathcal{M}\ a)))\ ))$
using *MFixFst* **by** *fastforce*
have *7*: $\vdash (\neg(\mathcal{M}\ b) \land bs\neg(\mathcal{M}\ b)) = bi(\neg(\mathcal{M}\ b))$
using *AndBsEqvBi* **by** *blast*
have *8*: $\vdash (\neg(\mathcal{M}\ a) \land bs\neg(\mathcal{M}\ a)) = bi(\neg(\mathcal{M}\ a))$
using *AndBsEqvBi* **by** *blast*
have *9*: $\vdash (((\mathcal{M}\ a) \land (\ ((\mathcal{M}\ b)) \lor ((\neg(\mathcal{M}\ b)) \land bs(\neg(\mathcal{M}\ b))))\ ) \lor$
$((\mathcal{M}\ b) \land (\ ((\mathcal{M}\ a)) \lor ((\neg(\mathcal{M}\ a)) \land bs(\neg(\mathcal{M}\ a))))\ )) =$
$(((\mathcal{M}\ a) \land (\ ((\mathcal{M}\ b)) \lor (\ bi(\neg(\mathcal{M}\ b))))\ ) \lor$
$((\mathcal{M}\ b) \land (\ ((\mathcal{M}\ a)) \lor (\ bi(\neg(\mathcal{M}\ a))))\ ))$
using *7 8* **by** *fastforce*
have *10*: $\vdash(((\mathcal{M}\ a) \land (\ ((\mathcal{M}\ b)) \lor (\ bi(\neg(\mathcal{M}\ b))))\ ) \lor$
$((\mathcal{M}\ b) \land (\ ((\mathcal{M}\ a)) \lor (\ bi(\neg(\mathcal{M}\ a))))\ )) =$
$(((\ (\mathcal{M}\ a) \land (\mathcal{M}\ b)) \lor (\ (\mathcal{M}\ a) \land bi(\neg(\mathcal{M}\ b)))) \lor$
$((\ (\mathcal{M}\ b) \land (\mathcal{M}\ a)) \lor (\ (\mathcal{M}\ b) \land bi(\neg(\mathcal{M}\ a)))))$
by *auto*
have *11*: $\vdash (((\ (\mathcal{M}\ a) \land (\mathcal{M}\ b)) \lor (\ (\mathcal{M}\ a) \land bi(\neg(\mathcal{M}\ b)))) \lor$
$((\ (\mathcal{M}\ b) \land (\mathcal{M}\ a)) \lor (\ (\mathcal{M}\ b) \land bi(\neg(\mathcal{M}\ a))))) =$
$((\ (\mathcal{M}\ a) \land bi\ \neg(\mathcal{M}\ b)) \lor ((\mathcal{M}\ b) \land bi\ \neg(\mathcal{M}\ a)) \lor ((\mathcal{M}\ a) \land (\mathcal{M}\ b)))$
by *auto*
from *1 2 3 4 5 6 9 10 11* **show** *?thesis* **by** (*metis int-eq*)
qed

lemma *MThruAlt*:
$\vdash \mathcal{M}(a\ THRU\ b) = (((\mathcal{M}\ a) \land di(\mathcal{M}\ b)) \lor ((\mathcal{M}\ b) \land di(\mathcal{M}\ a)))$
proof $-$
have *1*: $\vdash \mathcal{M}(a\ THRU\ b) = \triangleright(di(\mathcal{M}\ a) \land di(\mathcal{M}\ b))$
by (*simp*)
have *2*: $\vdash \triangleright(di(\mathcal{M}\ a) \land di(\mathcal{M}\ b)) = ((\triangleright(\mathcal{M}\ a) \land di(\mathcal{M}\ b)) \lor (\triangleright(\mathcal{M}\ b) \land di(\mathcal{M}\ a)))$
using *FstDiAndDiEqv* **by** *auto*
have *3*: $\vdash ((\triangleright(\mathcal{M}\ a) \land di(\mathcal{M}\ b)) \lor (\triangleright(\mathcal{M}\ b) \land di(\mathcal{M}\ a))) =$
$(((\mathcal{M}\ a) \land di(\mathcal{M}\ b)) \lor ((\mathcal{M}\ b) \land di(\mathcal{M}\ a)))$

using *MFixFst* **by** *fastforce*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MHaltAlt*:
$\vdash \mathcal{M}(HALT\ w) = halt(init\ w)$
proof $-$
have *1*: $\vdash \mathcal{M}(HALT\ w) = \mathcal{M}(FIRST\ LIFT(fin\ (init\ \ w)))$ **by** (*simp add*: *HALT-d-def*)
have *2*: $\vdash \mathcal{M}(FIRST\ LIFT(fin\ (init\ \ w))) = \rhd\ (fin\ (init\ \ w))$ **by** (*simp*)
have *3*: $\vdash \rhd\ (fin\ (init\ \ w)) = halt(init\ w)$ **using** *HaltStateEqvFstFinState* **by** *fastforce*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MFailUpto*:
$\vdash\ \mathcal{M}(FAIL\ UPTO\ a) = (\mathcal{M}\ a)$
proof $-$
have *1*: $\vdash\ \mathcal{M}(FAIL\ UPTO\ a) = \rhd(\ (\mathcal{M}\ FAIL) \vee (\mathcal{M}\ a))$ **by** (*simp*)
have *2*: $\vdash\ (\mathcal{M}\ FAIL \vee \mathcal{M}\ a) = (\#False \vee \mathcal{M}\ a)$ **using** *MFailAlt* **by** *auto*
have *3*: $\vdash \rhd(\mathcal{M}\ FAIL \vee (\mathcal{M}\ a)) = \rhd(\#False \vee (\mathcal{M}\ a))$ **using** *2 FstEqvRule* **by** *blast*
have *4*: $\vdash (\#False \vee (\mathcal{M}\ a)) = \mathcal{M}\ a$ **by** *simp*
have *5*: $\vdash \rhd(\#False \vee (\mathcal{M}\ a)) =\ \rhd(\mathcal{M}\ a)$ **using** *4 FstEqvRule* **by** *blast*
have *6*: $\vdash \rhd(\mathcal{M}\ a) = \mathcal{M}\ a$ **using** *MFixFst* **by** *fastforce*
from *1 2 3 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *MFailThru*:
$\vdash \mathcal{M}\ (FAIL\ THRU\ (\ a)) =\ \mathcal{M}\ FAIL$
proof $-$
have *1*: $\vdash \mathcal{M}\ (FAIL\ THRU\ (\ a)) = \rhd(di(\mathcal{M}\ FAIL) \wedge di(\mathcal{M}\ a))$
by (*simp*)
have *2*: $\vdash \rhd(di\ (\mathcal{M}\ FAIL) \wedge di(\mathcal{M}\ a)) = \rhd(di\ (\#False) \wedge di(\mathcal{M}\ a))$
using *MFailAlt* **by** (*metis 1 int-eq*)
have *3*: $\vdash di\ \#False = \#False$
by (*simp add*: *di-defs Valid-def*)
hence *4*: $\vdash \rhd(di\ (\#False) \wedge di(\mathcal{M}\ a)) = \rhd(\ (\#False) \wedge di(\mathcal{M}\ a))$
using *FstEqvRule* **by** (*metis int-eq intensional-simps*(*19*))
have *5*: $\vdash \rhd(\ (\#False) \wedge di(\mathcal{M}\ a)) = \rhd\#False$
using *FstEqvRule* **by** *fastforce*
have *6*: $\vdash \rhd\#False = \#False$ **using** *FstFalse*
by *auto*
have *7*: $\vdash \#False = \mathcal{M}\ FAIL$
using *MFailAlt* **by** *auto*
from *1 2 4 5 6 7* **show** *?thesis* **by** *fastforce*
qed

lemma *MFailAnd*:
$\vdash\ \mathcal{M}\ (FAIL\ AND\ a) =\ \mathcal{M}\ FAIL$
proof $-$
have *1*: $\vdash\ \mathcal{M}\ (FAIL\ AND\ a) = (\mathcal{M}\ FAIL \wedge (\mathcal{M}\ a))$ **by** (*simp add*: *AND-d-def*)
have *2*: $\vdash (\mathcal{M}\ FAIL \wedge (\mathcal{M}\ a)) = (\#False \wedge (\mathcal{M}\ a))$ **using** *MFailAlt* **by** *fastforce*

have *3*: $\vdash$ (#*False* $\wedge$ ($\mathcal{M}$ *a*)) = #*False* **by** *auto*
have *4*: $\vdash$ $\mathcal{M}$(*FAIL AND a*) = #*False* **using** *1 2 3* **by** *fastforce*
have *5*: $\vdash$ #*False* = $\mathcal{M}$ *FAIL* **using** *MFailAlt* **by** *auto*
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *MThenFail*:
$\vdash$ $\mathcal{M}$ (*a THEN FAIL*) = $\mathcal{M}$ *FAIL*
proof –
have *1*: $\vdash$ $\mathcal{M}$ (*a THEN FAIL*) = ($\mathcal{M}$ *a*);($\mathcal{M}$ *FAIL*) **by** (*simp*)
have *2*: $\vdash$ ($\mathcal{M}$ *a*);($\mathcal{M}$ *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* = $\mathcal{M}$ *FAIL* **using** *MFailAlt* **by** *auto*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *MFailThen*:
$\vdash$ $\mathcal{M}$ (*FAIL THEN a*) = $\mathcal{M}$ *FAIL*
proof –
have *1*: $\vdash$ $\mathcal{M}$(*FAIL THEN a*) = ($\mathcal{M}$ *FAIL*);($\mathcal{M}$ *a*) **by** (*simp*)
have *2*: $\vdash$ ($\mathcal{M}$ *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* = $\mathcal{M}$ *FAIL* **using** *MFailAlt* **by** *auto*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *MFailWith*:
$\vdash$ $\mathcal{M}$ (*FAIL WITH f*) = $\mathcal{M}$ *FAIL*
proof –
have *1*: $\vdash$ $\mathcal{M}$ (*FAIL WITH f*) = (($\mathcal{M}$ *FAIL*) $\wedge$ *f*) **by** (*simp*)
have *2*: $\vdash$ (($\mathcal{M}$ *FAIL*) $\wedge$ *f*) = (#*False* $\wedge$ *f*) **using** *MFailAlt* **by** *auto*
have *3*: $\vdash$ (#*False* $\wedge$ *f*) = #*False* **by** *simp*
have *4*: $\vdash$ #*False* = $\mathcal{M}$ *FAIL* **using** *MFailAlt* **by** *auto*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *MEmptyUpto*:
$\vdash$ $\mathcal{M}$ (*EMPTY UPTO a*) = $\mathcal{M}$ *EMPTY*
proof –
have *1*: $\vdash$ $\mathcal{M}$ (*EMPTY UPTO a*) = $\triangleright$($\mathcal{M}$ *EMPTY* $\vee$ ($\mathcal{M}$ *a*)) **by** (*simp*)
have *2*: $\vdash$ $\mathcal{M}$ *EMPTY* = *empty* **using** *MEmptyAlt* **by** *auto*
hence *3*: $\vdash$ ($\mathcal{M}$ *EMPTY* $\vee$ ($\mathcal{M}$ *a*)) = (*empty* $\vee$ ($\mathcal{M}$ *a*)) **by** *auto*
hence *4*: $\vdash$ $\triangleright$($\mathcal{M}$ *EMPTY* $\vee$ $\mathcal{M}$ *a*) = $\triangleright$(*empty* $\vee$ $\mathcal{M}$ *a*) **using** *FstEqvRule* **by** *blast*
have *5*: $\vdash$ $\triangleright$(*empty* $\vee$ $\mathcal{M}$ *a*) = *empty* **using** *FstEmptyOrEqvEmpty* **by** *blast*
have *6*: $\vdash$ *empty* = $\mathcal{M}$ *EMPTY* **using** *MEmptyAlt* **by** *auto*
from *1 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *MEmptyThru*:
$\vdash$ $\mathcal{M}$ (*EMPTY THRU a*) = ($\mathcal{M}$ *a*)

proof −
 have *1*: $\vdash$ $\mathcal{M}$(*EMPTY THRU a*) = $\triangleright$(*di*($\mathcal{M}$ *EMPTY*) $\wedge$ *di*($\mathcal{M}$ *a*)) **by** (*simp*)
 have *2*: $\vdash$ *di*($\mathcal{M}$ *EMPTY*) = *di empty* **using** *MEmptyAlt DiEqvDi* **by** *blast*
 hence *3*: $\vdash$ (*di*($\mathcal{M}$ *EMPTY*) $\wedge$ *di*($\mathcal{M}$ *a*)) = (*di empty* $\wedge$ *di*($\mathcal{M}$ *a*)) **by** *auto*
 hence *4*: $\vdash$ (*di empty* $\wedge$ *di*($\mathcal{M}$ *a*)) = *di*($\mathcal{M}$ *a*) **using** *DiEmpty* **by** *auto*
 have *5*: $\vdash$ (*di*($\mathcal{M}$ *EMPTY*) $\wedge$ *di*($\mathcal{M}$ *a*)) = *di*($\mathcal{M}$ *a*) **using** *3 4* **by** *fastforce*
 hence *6*: $\vdash$ $\triangleright$(*di*($\mathcal{M}$ *EMPTY*) $\wedge$ *di*($\mathcal{M}$ *a*)) = $\triangleright$(*di*($\mathcal{M}$ *a*)) **using** *FstEqvRule* **by** *blast*
 have *7*: $\vdash$ $\triangleright$(*di*($\mathcal{M}$ *a*)) = $\triangleright$($\mathcal{M}$ *a*) **using** *FstDiEqvFst* **by** *blast*
 have *8*: $\vdash$ $\triangleright$($\mathcal{M}$ *a*) = ($\mathcal{M}$ *a*) **using** *MFixFst* **by** *fastforce*
 from *1 6 7 8* **show** *?thesis* **by** *fastforce*
qed

lemma *MThenEmpty*:
$\vdash$ $\mathcal{M}$(*a THEN EMPTY*) = ($\mathcal{M}$ *a*)
proof −
 have *1*: $\vdash$ $\mathcal{M}$(*a THEN EMPTY*) = ($\mathcal{M}$ *a*); ($\mathcal{M}$ *EMPTY*) **by** (*simp*)
 have *2*: $\vdash$ ($\mathcal{M}$ *a*); ($\mathcal{M}$ *EMPTY*) = ($\mathcal{M}$ *a*); *empty* **by** (*simp add*: *MEmptyAlt RightChopEqvChop*)
 have *3*: $\vdash$ ($\mathcal{M}$ *a*); *empty* = ($\mathcal{M}$ *a*) **using** *ChopEmpty* **by** *auto*
 from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MEmptyThen*:
$\vdash$ $\mathcal{M}$(*EMPTY THEN a*) = $\mathcal{M}$ *a*
proof −
 have *1*: $\vdash$ $\mathcal{M}$(*EMPTY THEN a*) = ($\mathcal{M}$ *EMPTY*);($\mathcal{M}$ *a*) **by** (*simp*)
 have *2*: $\vdash$ ($\mathcal{M}$ *EMPTY*);($\mathcal{M}$ *a*) = *empty*;($\mathcal{M}$ *a*) **by** (*simp add*: *MEmptyAlt LeftChopEqvChop*)
 have *3*: $\vdash$ *empty*;($\mathcal{M}$ *a*) = ($\mathcal{M}$ *a*) **by** (*simp add*: *EmptyChop*)
 from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MEmptyIterate*:
$\vdash$ $\mathcal{M}$(*EMPTY ITERATE b*) = $\mathcal{M}$ *EMPTY*
proof −
 have *1*: $\vdash$ $\mathcal{M}$(*EMPTY ITERATE b*) = $\mathcal{M}$(*EMPTY WITH LIFT*($\mathcal{M}$ *b*)*)
  by (*simp add*: *ITERATE-d-def*)
 have *2*: $\vdash$ $\mathcal{M}$ (*EMPTY WITH LIFT*($\mathcal{M}$ *b*)*) = ($\mathcal{M}$ *EMPTY* $\wedge$ ($\mathcal{M}$ *b*)*)
  by (*simp*)
 have *3*: $\vdash$ ($\mathcal{M}$ *EMPTY* $\wedge$ ($\mathcal{M}$ *b*)*) = (*empty* $\wedge$ ($\mathcal{M}$ *b*)*)
  using *MEmptyAlt* **by** *auto*
 have *4*: $\vdash$ (*empty* $\wedge$ ($\mathcal{M}$ *b*)*) = (*empty* $\wedge$ (*empty* $\vee$ ((($\mathcal{M}$ *b*) $\wedge$ *more*);($\mathcal{M}$ *b*)*)))
  using *ChopstarEqv* **by** *fastforce*
 have *5*: $\vdash$ (*empty* $\wedge$ (*empty* $\vee$ ((($\mathcal{M}$ *b*) $\wedge$ *more*);($\mathcal{M}$ *b*)*))) = *empty*
  by *auto*
 have *6*: $\vdash$ $\mathcal{M}$(*EMPTY ITERATE b*) = $\mathcal{M}$ *EMPTY*
  using *1 2 3 4 5 MEmptyAlt* **by** *fastforce*
 from *6* **show** *?thesis* **by** *simp*
qed

lemma *FstAndFstStarEqvFst*:
$\vdash$ ($\triangleright$*f* $\wedge$ ($\triangleright$*f*)*) = $\triangleright$*f*

proof −
  have *1*: $\vdash (\triangleright f)^\star = (empty \vee (\triangleright f);(\triangleright f)^\star)$
    using *CSEqvOrChopCS* **by** *fastforce*
  have *2*: $\vdash ((\triangleright f)^\star \wedge \triangleright f) = ((empty \vee (\triangleright f);(\triangleright f)^\star) \wedge \triangleright f)$
    using *1* **by** *fastforce*
  have *3*: $\vdash ((empty \vee (\triangleright f);(\triangleright f)^\star) \wedge \triangleright f) = ((empty \wedge \triangleright f) \vee ((\triangleright f);(\triangleright f)^\star \wedge \triangleright f))$
    by *auto*
  have *4*: $\vdash ((\triangleright f)^\star \wedge \triangleright f) = ((empty \wedge \triangleright f) \vee ((\triangleright f);(\triangleright f)^\star \wedge \triangleright f))$
    using *2 3* **by** *fastforce*
  have *5*: $\vdash ((\triangleright f);(\triangleright f)^\star \wedge \triangleright f) = ((\triangleright f);(\triangleright f)^\star \wedge \triangleright f;empty)$
    using *ChopEmpty* **by** (*metis inteq-reflection*)
  have *6*: $\vdash ((\triangleright f);(\triangleright f)^\star \wedge \triangleright f;empty) = (\triangleright f);((\triangleright f)^\star \wedge empty)$
    using *LFstAndDistrC* **by** *blast*
  have *7*: $\vdash ((\triangleright f)^\star \wedge empty) = empty$
    using *EmptyImpCS* **by** *fastforce*
  have *8*: $\vdash (\triangleright f);((\triangleright f)^\star \wedge empty) = \triangleright f$
    using *7 ChopEmpty* **by** (*metis inteq-reflection*)
  have *9*: $\vdash ((\triangleright f);(\triangleright f)^\star \wedge \triangleright f) = \triangleright f$
    using *5 6 8* **by** *fastforce*
  have *10*: $\vdash ((\triangleright f)^\star \wedge \triangleright f) = ((empty \wedge \triangleright f) \vee \triangleright f)$
    using *4 9* **by** *fastforce*
  have *11*: $\vdash ((empty \wedge \triangleright f) \vee \triangleright f) = \triangleright f$
    by *auto*
  have *12*: $\vdash ((\triangleright f)^\star \wedge \triangleright f) = \triangleright f$
    using *10 11* **by** *fastforce*
  from *12* **show** *?thesis* **by** *auto*
qed

lemma *MIterateIdemp*:
$\vdash \mathcal{M}(a\ ITERATE\ a) = (\mathcal{M}\ a)$
proof −
  have *1*: $\vdash \mathcal{M}(a\ ITERATE\ a) = \mathcal{M}\ (a\ WITH\ LIFT(\mathcal{M}\ a)^\star)$ **by** (*simp add*: *ITERATE-d-def*)
  have *2*: $\vdash \mathcal{M}(a\ WITH\ LIFT(\mathcal{M}\ a)^\star) = ((\mathcal{M}\ a) \wedge (\mathcal{M}\ a)^\star)$ **by** (*simp*)
  have *3*: $\vdash ((\mathcal{M}\ a) \wedge (\mathcal{M}\ a)^\star) = (\triangleright(\mathcal{M}\ a) \wedge (\triangleright(\mathcal{M}\ a))^\star)$ **using** *MFixFst*
  by (*metis ImpCS inteq-reflection Prop10*)
  have *4*: $\vdash (\triangleright(\mathcal{M}\ a) \wedge (\triangleright(\mathcal{M}\ a))^\star) = \triangleright(\mathcal{M}\ a)$ **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* **by** *fastforce*
qed

lemma *MUptoIdemp*:
$\vdash \mathcal{M}(a\ UPTO\ a) = (\mathcal{M}\ a)$
proof −
  have *1*: $\vdash \mathcal{M}(a\ UPTO\ a) = \triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ a))$ **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* **by** *fastforce*
qed

lemma *MThruIdemp*:

$\vdash\ \mathcal{M}(\ a\ THRU\ a) = (\mathcal{M}\ a)$
proof −
 have *1*: $\vdash\ \mathcal{M}(\ a\ THRU\ a) = \rhd(\ di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ a))$ **by** *auto*
 have *2*: $\vdash \rhd(\ di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ a)) = \rhd(di\ (\mathcal{M}\ a))$ **using** *FstEqvRule* **by** *fastforce*
 have *3*: $\vdash \rhd(di\ (\mathcal{M}\ a)) = \rhd(\mathcal{M}\ a)$ **using** *FstDiEqvFst* **by** *blast*
 have *4*: $\vdash \rhd(\mathcal{M}\ a) = (\mathcal{M}\ a)$ **using** *MFixFst* **by** *fastforce*
 from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *MAndIdemp*:
$\vdash \mathcal{M}(a\ AND\ a) = (\mathcal{M}\ a)$
proof −
 have *1*: $\vdash \mathcal{M}(a\ AND\ a) = ((\mathcal{M}\ a) \wedge (\mathcal{M}\ a))$ **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* **by** *auto*
qed

lemma *MWithIdemp*:
$\vdash\ \mathcal{M}(\ (a\ WITH\ f)\ \ WITH\ f) = \mathcal{M}(a\ WITH\ f)$
proof −
 have *1*: $\vdash \mathcal{M}(\ (a\ WITH\ f)\ \ WITH\ f) = (((\mathcal{M}\ a) \wedge (\ f)) \wedge (\ f))$ **by** *auto*
 have *2*: $\vdash (((\mathcal{M}\ a) \wedge (\ f)) \wedge (\ f)) = ((\mathcal{M}\ a) \wedge (\ f))$ **by** *fastforce*
 have *3*: $\vdash ((\mathcal{M}\ a) \wedge (\ f)) = \mathcal{M}(a\ WITH\ f)$ **by** *auto*
 from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MUptoCommut*:
$\vdash \mathcal{M}(a\ UPTO\ b) = \mathcal{M}(b\ UPTO\ a)$
proof −
 have *1*: $\vdash \mathcal{M}(a\ UPTO\ b) = \rhd((\mathcal{M}\ a) \vee (\mathcal{M}\ b))$ **by** (*simp*)
 have *2*: $\vdash ((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) = ((\mathcal{M}\ b) \vee (\mathcal{M}\ a))$ **by** *auto*
 hence *3*: $\vdash \rhd((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) = \rhd((\mathcal{M}\ b) \vee (\mathcal{M}\ a))$ **using** *FstEqvRule* **by** *blast*
 have *4*: $\vdash \rhd((\mathcal{M}\ b) \vee (\mathcal{M}\ a)) = \mathcal{M}(b\ UPTO\ a)$ **by** *auto*
 from *1 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *MThruCommut*:
$\vdash \mathcal{M}(a\ THRU\ b) = \mathcal{M}(b\ THRU\ a)$
proof −
 have *1*: $\vdash \mathcal{M}(a\ THRU\ b) = \rhd(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))$ **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*: $\vdash \rhd(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) = \rhd(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ a))$ **using** *FstEqvRule* **by** *blast*
 have *4*: $\vdash \rhd(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ a)) = \mathcal{M}(b\ THRU\ a)$ **by** *auto*
 from *1 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *MAndCommut*:
$\vdash \mathcal{M}(a\ AND\ b) = \mathcal{M}(b\ AND\ a)$
proof −
 have *1*: $\vdash \mathcal{M}(a\ AND\ b) = ((\mathcal{M}\ a) \wedge (\mathcal{M}\ b))$ **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*: $\vdash ((\mathcal{M}\ b) \wedge (\mathcal{M}\ a)) = \mathcal{M}(b\ AND\ a)$ **by** (*simp add*: *AND-d-def*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MWithCommut*:
$\vdash \mathcal{M}((a\ WITH\ f)\ WITH\ g) = \mathcal{M}((a\ WITH\ g)\ WITH\ f)$
proof −
have *1*: $\vdash \mathcal{M}((a\ WITH\ f)\ WITH\ g) = (((\mathcal{M}\ a) \wedge (f)) \wedge (g))$ **by** *auto*
have *2*: $\vdash (((\mathcal{M}\ a) \wedge (f)) \wedge (g)) = (((\mathcal{M}\ a) \wedge (g)) \wedge (f))$ **by** *auto*
have *2*: $\vdash (((\mathcal{M}\ a) \wedge (g)) \wedge (f)) = \mathcal{M}((a\ WITH\ g)\ WITH\ f)$ **by** *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *MWithAbsorp*:
$\vdash \mathcal{M}((a\ WITH\ f)\ WITH\ g) = \mathcal{M}(a\ WITH\ LIFT(f \wedge g))$
proof −
have *1*: $\vdash \mathcal{M}((a\ WITH\ f)\ WITH\ g) = (((\mathcal{M}\ a) \wedge (f)) \wedge (g))$ **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** *auto*
qed

lemma *MUptoAssoc*:
$\vdash \mathcal{M}((a\ UPTO\ b)\ UPTO\ c) = \mathcal{M}(a\ UPTO\ (b\ UPTO\ c))$
proof −
have *1*: $\vdash \mathcal{M}((a\ UPTO\ b)\ UPTO\ c) = \rhd(\mathcal{M}(a\ UPTO\ b) \vee (\mathcal{M}\ c))$
by (*simp*)
have *2*: $\vdash \rhd(\mathcal{M}(a\ UPTO\ b) \vee (\mathcal{M}\ c)) = \rhd(\rhd((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \vee (\mathcal{M}\ c))$
by *auto*
have *3*: $\vdash \rhd(\rhd((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \vee (\mathcal{M}\ c)) = \rhd(((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \vee (\mathcal{M}\ c))$
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*: $\vdash \rhd(((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \vee (\mathcal{M}\ c)) = \rhd((\mathcal{M}\ a) \vee ((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))$
using *FstEqvRule* **by** *blast*
have *6*: $\vdash \rhd((\mathcal{M}\ a) \vee ((\mathcal{M}\ b) \vee (\mathcal{M}\ c))) = \rhd((\mathcal{M}\ a) \vee \rhd((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))$
using *FstFstOrEqvFstOrR* **by** *fastforce*
have *7*: $\vdash \rhd((\mathcal{M}\ a) \vee \rhd((\mathcal{M}\ b) \vee (\mathcal{M}\ c))) = \rhd((\mathcal{M}\ a) \vee \mathcal{M}(b\ UPTO\ c))$
by *auto*
have *8*: $\vdash \rhd((\mathcal{M}\ a) \vee \mathcal{M}(b\ UPTO\ c)) = \mathcal{M}(a\ UPTO\ (b\ UPTO\ c))$
by *auto*
from *1 2 3 5 6 7 8* **show** *?thesis* **by** *fastforce*
qed

lemma *MThruAssoc*:
$\vdash \mathcal{M}((a\ THRU\ b)\ THRU\ c) = \mathcal{M}(a\ THRU\ (b\ THRU\ c))$
proof −
have *1*: $\vdash \mathcal{M}((a\ THRU\ b)\ THRU\ c) = \rhd(di(\rhd(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) \wedge di(\mathcal{M}\ c))$
by *auto*
have *2*: $\vdash di(\rhd(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) = di((di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))$

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*: $\vdash \triangleright(di(\mathcal{M}\ a) \wedge di\ (\triangleright(di(\mathcal{M}\ b) \ \wedge di(\mathcal{M}\ c)))) = \mathcal{M}(a\ THRU\ (b\ THRU\ c))$
by *auto*
from *1 11 12* **show** *?thesis* **by** *fastforce*
qed

lemma *MAndAssoc*:
$\vdash \mathcal{M}((a\ AND\ b)\ AND\ c) = \mathcal{M}(a\ AND\ (b\ AND\ c))$
proof −
have *1*: $\vdash \mathcal{M}((a\ AND\ b)\ AND\ c) = ((\mathcal{M}\ a) \wedge (\mathcal{M}\ b) \wedge (\mathcal{M}\ c))$
using *AND-d-def* **by** (*metis MON.simps*(*5*) *MWithAbsorp*)
have *2*: $\vdash ((\mathcal{M}\ a) \wedge (\mathcal{M}\ b) \wedge (\mathcal{M}\ c)) = \mathcal{M}\ (a\ AND\ (b\ AND\ c))$
using *AND-d-def* **by** (*simp add*: *AND-d-def*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *MThenAssoc*:
$\vdash \mathcal{M}((a\ THEN\ b)\ THEN\ c) = \mathcal{M}(a\ THEN\ (b\ THEN\ c))$
proof −
have *1*: $\vdash \mathcal{M}((a\ THEN\ b)\ THEN\ c) = ((\mathcal{M}\ a);(\mathcal{M}\ b));(\mathcal{M}\ c)$ **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*: $\vdash (\mathcal{M}\ a);((\mathcal{M}\ b);(\mathcal{M}\ c)) = \mathcal{M}(a\ THEN\ (b\ THEN\ c))$ **by** *auto*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *OrDiEqvDi*:
$\vdash (f \vee di\ f) = di\ f$
proof −
have *1*: $\vdash f \longrightarrow di\ f$ **using** *DiIntro* **by** *blast*
from *1* **show** *?thesis* **by** *auto*

qed

lemma *AndDiEqv*:
$\vdash (f \wedge di\ f) = f$
proof $-$
  have *1*: $\vdash f \longrightarrow di\ f$ **using** *DiIntro* **by** *blast*
 from *1* **show** *?thesis* **by** *auto*
qed

lemma *MUptoThruAbsorp*:
$\vdash \mathcal{M}(a\ UPTO\ (a\ THRU\ b)) = \mathcal{M}\ a$
proof $-$
  have *1*: $\vdash \mathcal{M}(a\ UPTO\ (a\ THRU\ b)) = \triangleright((\mathcal{M}\ a) \vee \triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ ))$
      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*: $\vdash (\triangleright(\mathcal{M}\ a) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b)))) =$

```
        ((𝓜 a) ∧ bs ¬(𝓜 a) ∧ (bs (¬(𝓜 a)) ∨ bs (¬(𝓜 b))))
     by (simp add: first-d-def, auto)
  have 29: ⊢ ((𝓜 a) ∧ bs ¬(𝓜 a) ∧ (bs (¬(𝓜 a)) ∨ bs (¬(𝓜 b)))) =
        ((𝓜 a) ∧ bs ¬(𝓜 a))
     by auto
  have 30: ⊢ ((𝓜 a) ∧ bs ¬(𝓜 a)) = ▷(𝓜 a)
     by (simp add: first-d-def)
  have 31: ⊢ ▷(𝓜 a) = (𝓜 a)
     using MFixFst by fastforce
  have 32: ⊢ 𝓜(a UPTO (a THRU b)) =
        ((di(𝓜 a)) ∧ ((𝓜 a) ∨ di(𝓜 b)) ∧
        bs ¬( (di(𝓜 a)) ∧ ((𝓜 a) ∨ di(𝓜 b))))
     using 1 2 6 7 by fastforce
  have 33: ⊢ ((di(𝓜 a)) ∧ ((𝓜 a) ∨ di(𝓜 b)) ∧
        bs ¬( (di(𝓜 a)) ∧ ((𝓜 a) ∨ di(𝓜 b)))) =
        (((𝓜 a)) ∧  bs(¬(di(𝓜 a) ∧ di(𝓜 b))))
     using 15 16 17 18 19 by (metis int-eq)
  have 34: ⊢ (((𝓜 a)) ∧  bs(¬(di(𝓜 a) ∧ di(𝓜 b)))) = (𝓜 a)
     using 26 27 28 29 30 31 by (metis int-eq)
 from 32 33 34 show ?thesis by fastforce
qed

lemma MThruUptoAbsorp:
⊢ 𝓜(a THRU (a UPTO b)) = (𝓜 a)
proof −
  have 1: ⊢ 𝓜(a THRU (a UPTO b)) = ▷(di(𝓜 a) ∧ di(▷((𝓜 a) ∨ (𝓜 b))))
     by simp
  have 2: ⊢  ▷(di(𝓜 a) ∧ di(▷((𝓜 a) ∨ (𝓜 b)))) =
         ▷(di(𝓜 a) ∧ di(((𝓜 a) ∨ (𝓜 b))))
    by (metis DiEqvDiFst FstEqvRule inteq-reflection lift-and-com)
  have 3: ⊢ ▷(di(𝓜 a) ∧ di(((𝓜 a) ∨ (𝓜 b)))) =
        ▷(di(𝓜 a) ∧ (di(𝓜 a) ∨ di(𝓜 b)))
    by (metis DiOrEqv FstEqvRule inteq-reflection lift-and-com)
  have 4: ⊢ (di(𝓜 a) ∧ (di(𝓜 a) ∨ di(𝓜 b))) = (di(𝓜 a))
      by auto
  hence 5: ⊢ ▷(di(𝓜 a) ∧ (di(𝓜 a) ∨ di(𝓜 b))) = ▷(di(𝓜 a))
     using FstEqvRule by blast
  have 6: ⊢ ▷(di(𝓜 a)) = ▷(𝓜 a)
     using FstDiEqvFst by blast
  have 7: ⊢ ▷(𝓜 a) = (𝓜 a)
     using MFixFst by fastforce
 from 1 2 3 5 6 7 show ?thesis by fastforce
qed

lemma MUptoThruDistrib:
⊢ 𝓜(a UPTO (b THRU c)) = 𝓜((a UPTO b) THRU (a UPTO c))
proof −
  have 1: ⊢ 𝓜((a UPTO b) THRU (a UPTO c)) =
        ▷( di(▷((𝓜 a) ∨ (𝓜 b))) ∧ di(▷((𝓜 a) ∨ (𝓜 c))) )
     by simp
```

have *2*: $\vdash (\ di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ ) =$
$(\ di(((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ )$
using *DiEqvDiFst* **by** *fastforce*
have *3*: $\vdash (\ di(((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ ) =$
$((\ di(\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge (di(\mathcal{M}\ a) \vee di(\mathcal{M}\ c)))$
using *DiOrEqv* **by** *fastforce*
have *4*: $\vdash ((\ di(\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge (di(\mathcal{M}\ a) \vee di(\mathcal{M}\ c))) =$
$(di(\mathcal{M}\ a) \vee (di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)))$
by *auto*
have *5*: $\vdash (\ di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ ) =$
$(di(\mathcal{M}\ a) \vee (di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)))$
using *2 3 4* **by** *fastforce*
hence *6*: $\vdash \triangleright(\ di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ ) =$
$\triangleright(\ di(\mathcal{M}\ a) \vee (di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))\ )$
using *FstEqvRule* **by** *blast*
have *7*: $\vdash \triangleright(\ di(\mathcal{M}\ a) \vee (di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))\ ) =$
$\triangleright(\ \triangleright(di(\mathcal{M}\ a)) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))\ )$
using *FstFstOrEqvFstOr* **by** *fastforce*
have *8*: $\vdash \triangleright(di(\mathcal{M}\ a)) = \triangleright((\mathcal{M}\ a))$
using *FstDiEqvFst* **by** *blast*
have *9*: $\vdash \triangleright((\mathcal{M}\ a)) = (\mathcal{M}\ a)$
using *MFixFst* **by** *fastforce*
have *10*: $\vdash \triangleright(di(\mathcal{M}\ a)) = (\mathcal{M}\ a)$
using *8 9* **by** *fastforce*
hence *11*: $\vdash (\triangleright(di(\mathcal{M}\ a)) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))) =$
$((\mathcal{M}\ a) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)))$
by *auto*
hence *12*: $\vdash \triangleright(\triangleright(di(\mathcal{M}\ a)) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))) =$
$\triangleright((\mathcal{M}\ a) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)))$
using *FstEqvRule* **by** *blast*
have *13*: $\vdash \triangleright((\mathcal{M}\ a) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))) = \mathcal{M}(a\ UPTO\ (b\ THRU\ c))$
by *simp*
from *1 6 7 12 13* **show** *?thesis* **by** *fastforce*
qed

lemma *MThruUptoDistrib*:
$\vdash \mathcal{M}(a\ THRU\ (b\ UPTO\ c)) = \mathcal{M}((a\ THRU\ b)\ UPTO\ (a\ THRU\ c))$
proof −
have *1*: $\vdash \mathcal{M}((a\ THRU\ b)\ UPTO\ (a\ THRU\ c)) =$
$\triangleright(\triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee \triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ c)))$
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*: $\vdash \triangleright(di(\mathcal{M}\ a) \wedge di(\triangleright(\ (\mathcal{M}\ b) \vee (\mathcal{M}\ c)))) =$
$\mathcal{M}(a\ THRU\ (b\ UPTO\ c))$ **by** *simp*
from *1 2 7 8* **show** *?thesis* **by** *fastforce*
qed

lemma *MWithAndDistrib*:
$\vdash \mathcal{M}((a\ AND\ b)\ WITH\ f) = \mathcal{M}((a\ WITH\ f)\ AND\ (b\ WITH\ f))$
proof $-$
have *1*: $\vdash \mathcal{M}((a\ AND\ b)\ WITH\ f) = (\mathcal{M}(a\ AND\ b) \wedge f)$
by (*simp*)
have *2*: $\vdash \mathcal{M}(a\ AND\ b) = \mathcal{M}(a\ WITH\ LIFT(\mathcal{M}\ b))$
by (*simp add*: *AND-d-def*)
have *3*: $\vdash (\mathcal{M}(a\ AND\ b) \wedge f) = (\mathcal{M}(a\ WITH\ LIFT(\mathcal{M}\ b)) \wedge f)$
using *2* **by** *auto*
have *4*: $\vdash \mathcal{M}(a\ WITH\ (LIFT(\ (\mathcal{M}\ b) \wedge f))) = (\mathcal{M}(a) \wedge \mathcal{M}(b) \wedge f)$
by *simp*
have *5*: $\vdash (\mathcal{M}(a) \wedge \mathcal{M}(b) \wedge f) = ((\mathcal{M}(a) \wedge f) \wedge (\mathcal{M}(b) \wedge f))$
by *auto*
have *6*: $\vdash ((\mathcal{M}(a) \wedge f) \wedge (\mathcal{M}(b) \wedge f)) = (\mathcal{M}(a\ WITH\ f) \wedge \mathcal{M}(\ b\ WITH\ f))$
by *simp*
have *7*: $\vdash (\mathcal{M}(a\ WITH\ f) \wedge \mathcal{M}(b\ WITH\ f)) = \mathcal{M}((a\ WITH\ f)\ WITH\ LIFT(\mathcal{M}(b\ WITH\ f)))$
by *simp*
have *8*: $\vdash \mathcal{M}((a\ WITH\ f)\ WITH\ LIFT(\mathcal{M}(b\ WITH\ f))) = \mathcal{M}((a\ WITH\ f)\ AND\ (b\ WITH\ f))$
by (*simp add*: *AND-d-def*)
from *1 2 3 4 5 6 7 8* **show** *?thesis* **by** (*metis AND-d-def MWithAbsorp int-eq*)
qed

lemma *MHaltWithAndDistrib*:
$\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g)) = \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g))$
proof $-$
have *1*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g)) =$
$\mathcal{M}(((HALT\ w)\ WITH\ f)\ WITH\ LIFT(\mathcal{M}((HALT\ w)\ WITH\ g)))$
by (*simp add*: *AND-d-def*)
have *2*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ WITH\ LIFT(\mathcal{M}((HALT\ w)\ WITH\ g))) =$
$(\mathcal{M}(HALT\ w) \wedge f \wedge \mathcal{M}(HALT\ w) \wedge g)$
by *auto*
have *3*: $\vdash (\mathcal{M}(HALT\ w) \wedge f \wedge \mathcal{M}(HALT\ w) \wedge g) = (\mathcal{M}(HALT\ w) \wedge f \wedge g)$
by *auto*
have *4*: $\vdash (\mathcal{M}(HALT\ w) \wedge f \wedge g) = \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g))$
by *auto*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *MHaltWithUptoHaltWithEqvHaltWithOr*:
$\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ UPTO\ ((HALT\ w)\ WITH\ g)) = \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \vee g))$
proof $-$

have *1*: $\vdash \mathcal{M}(((\mathit{HALT}\ w)\ \mathit{WITH}\ f)\ \mathit{UPTO}\ ((\mathit{HALT}\ w)\ \mathit{WITH}\ g)) =$
$\triangleright(\mathcal{M}((\mathit{HALT}\ w)\ \mathit{WITH}\ f) \lor \mathcal{M}((\mathit{HALT}\ w)\ \mathit{WITH}\ g))$
by (*simp*)
have *2*: $\vdash \triangleright(\mathcal{M}((\mathit{HALT}\ w)\ \mathit{WITH}\ f) \lor \mathcal{M}((\mathit{HALT}\ w)\ \mathit{WITH}\ g)) =$
$\triangleright((\mathcal{M}(\mathit{HALT}\ w) \land f) \lor (\mathcal{M}(\mathit{HALT}\ w) \land g))$
by *auto*
have *3*: $\vdash ((\mathcal{M}(\mathit{HALT}\ w) \land f) \lor (\mathcal{M}(\mathit{HALT}\ w) \land g)) = (\mathcal{M}(\mathit{HALT}\ w) \land (f \lor g))$
by *auto*
have *4*: $\vdash \triangleright((\mathcal{M}(\mathit{HALT}\ w) \land f) \lor (\mathcal{M}(\mathit{HALT}\ w) \land g)) = \triangleright(\mathcal{M}(\mathit{HALT}\ w) \land (f \lor g))$
using *3 FstEqvRule* **by** *fastforce*
have *5*: $\vdash \triangleright(\mathcal{M}(\mathit{HALT}\ w) \land (f \lor g)) = \triangleright(\mathcal{M}((\mathit{HALT}\ w)\ \mathit{WITH}\ \mathit{LIFT}(f \lor g)))$
by *simp*
have *6*: $\vdash \mathcal{M}(((\mathit{HALT}\ w)\ \mathit{WITH}\ \mathit{LIFT}(f \lor g))) = \triangleright(\mathcal{M}((\mathit{HALT}\ w)\ \mathit{WITH}\ \mathit{LIFT}(f \lor g)))$
using *MFixFst* **by** *blast*
from *1 2 3 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *DiHaltAndDiHaltAndEqvDiHaltAndAnd*:
$\vdash (\mathit{di}(\mathit{halt}\ (\mathit{init}\ w) \land f) \land \mathit{di}(\mathit{halt}\ (\mathit{init}\ w) \land g)) = \mathit{di}(\mathit{halt}\ (\mathit{init}\ w) \land f \land g)$
proof −
have *1*: $\vdash (\mathit{di}(\mathit{halt}\ (\mathit{init}\ w) \land f) \land \mathit{di}(\mathit{halt}\ (\mathit{init}\ w) \land g)) =$
$(\mathit{di}(\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \land f) \land \mathit{di}\ (\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \land g))$
using *HaltStateEqvFstFinState* **by** (*metis LFstAndDistrD inteq-reflection*)
have *2*: $\vdash (\mathit{di}(\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \land f) \land \mathit{di}(\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \land g)) =$
$\mathit{di}(\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \land f \land g)$
using *LFstAndDistrD* **by** *fastforce*
have *3*: $\vdash \mathit{di}(\triangleright(\mathit{fin}\ (\mathit{init}\ w)) \land f \land g) = \mathit{di}(\mathit{halt}\ (\mathit{init}\ w) \land f \land g)$
using *HaltStateEqvFstFinState* **by** (*metis DiEqvDi int-eq lift-and-com*)
from *1 2 3* **show** *?thesis* **using** *int-eq* **by** *metis*
qed

lemma *MHaltWithThruHaltWithEqvHaltWithAndHaltWith*:
$\vdash \mathcal{M}(((\mathit{HALT}\ w)\ \mathit{WITH}\ f)\ \mathit{THRU}\ ((\mathit{HALT}\ w)\ \mathit{WITH}\ g)) = \mathcal{M}(((\mathit{HALT}\ w)\ \mathit{WITH}\ f)\ \mathit{AND}\ ((\mathit{HALT}\ w)\ \mathit{WITH}$
$g))$
proof −
have *1*: $\vdash \mathcal{M}(((\mathit{HALT}\ w)\ \mathit{WITH}\ f)\ \mathit{THRU}\ ((\mathit{HALT}\ w)\ \mathit{WITH}\ g)) =$
$\triangleright(\ \mathit{di}(\mathcal{M}(\mathit{HALT}\ w) \land f) \land \mathit{di}(\mathcal{M}(\mathit{HALT}\ w) \land g)\ )$
by *simp*
have *2*: $\vdash (\mathit{di}(\mathcal{M}(\mathit{HALT}\ w) \land f) \land \mathit{di}(\mathcal{M}(\mathit{HALT}\ w) \land g)) =$
$(\mathit{di}(\mathit{halt}(\mathit{init}\ w) \land f) \land \mathit{di}(\mathit{halt}(\mathit{init}\ w) \land g))$
using *MHaltAlt DiEqvDi*
by (*metis* (*no-types*, *lifting*) *inteq-reflection lift-and-com*)
have *3*: $\vdash (\mathit{di}(\mathit{halt}(\mathit{init}\ w) \land f) \land \mathit{di}(\mathit{halt}(\mathit{init}\ w) \land g)) =$
$\mathit{di}(\mathit{halt}(\mathit{init}\ w) \land f \land g)$
using *DiHaltAndDiHaltAndEqvDiHaltAndAnd* **by** *fastforce*
have *4*: $\vdash \mathit{di}(\ \mathit{halt}(\mathit{init}\ w) \land f \land g) = \mathit{di}(\mathcal{M}(\mathit{HALT}\ w) \land f \land g)$
by (*metis DiEqvDi MHaltAlt inteq-reflection lift-and-com*)
have *5*: $\vdash (\mathit{di}(\mathcal{M}(\mathit{HALT}\ w) \land f) \land \mathit{di}(\mathcal{M}(\mathit{HALT}\ w) \land g)) = \mathit{di}(\mathcal{M}(\mathit{HALT}\ w) \land f \land g)$
using *2 3 4* **by** *fastforce*
have *6*: $\vdash \triangleright(\mathit{di}(\mathcal{M}(\mathit{HALT}\ w) \land f) \land \mathit{di}(\mathcal{M}(\mathit{HALT}\ w) \land g)) = \triangleright(\mathit{di}(\mathcal{M}(\mathit{HALT}\ w) \land f \land g))$

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* **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* **by** *fastforce*
qed

end

theory *Example*
imports
FOTheorems
begin

9 Example

sledgehammer-params [*minimize=true,preplay-timeout=10,timeout=60,verbose=true,*
provers=cvc4 e z3 vampire spass]

declare [[*show-types*]]

locale *Test* =
fixes *v* :: *state* ⇒ *nat*
fixes *v1* :: *state* ⇒ *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 $F2 \equiv (\lambda\ v.\ TEMP\ \square\ (\ \#0 \le \$v\ ))$
defines $F3 \equiv (\lambda\ p.\ TEMP\ \square\ (\ \$p \vee \neg\ \$p))$
defines $F4 \equiv (\lambda\ z.\ TEMP\ \square\ (\ \#0 \le \$z \vee \$z < \#0))$
defines $F5 \equiv (\lambda\ v.\ TEMP\ \$v{=}\#0 \wedge v\ gets\ \$v{+}\#1)$
defines $Init2 \equiv (\lambda\ v.\ TEMP\ \$v = \#0)$

defines *Init3* $\equiv$ (λ *p*. *TEMP* \$*p*)

locale *Test1* =
fixes *v* :: *state* $\Rightarrow$ *nat*
fixes *F5* :: *nat statefun* $\Rightarrow$ *nat* $\Rightarrow$ *temporal*
defines *F5* $\equiv$ (λ *v n*. *TEMP* \$*v*=#*0* $\wedge$ *v gets* \$*v*+#*1* $\wedge$ *fin*(\$*v*=#*n*))

definition *F1* :: *nat statefun* $\Rightarrow$ *temporal*
where *F1 w* $\equiv$ *TEMP* $\Box$ (#*0* $\leq$ \$*w*)

definition *Init1* :: *nat statefun* $\Rightarrow$ *temporal*
where *Init1 w* $\equiv$ *TEMP* \$*w* = #*0*

lemma (**in** *Test*) *currentval-test* :
(*s* $\models$ (\$*v* = #*0*)) = ((*v* (*nth s 0*)) = *0*)
by (*simp add*: *current-val-d-def*)

lemma (**in** *Test*) *nextempty-test* :
($\langle$*s0*$\rangle$ $\models$ *v*\$) = (ϵ *x*. *x*=*x*)
by (*simp add*: *next-val-d-def*)

lemma (**in** *Test*) *nextempty-test-1* :
($\langle$*s0*$\rangle$ $\models$ *v*\$ = *v*\$)
by *simp*

lemma (**in** *Test*) *nextempty-test-2* :
($\langle$*s0*$\rangle$ $\models$ *v*\$ = *v1*\$)
by (*simp add*: *Test.nextempty-test*)

lemma (**in** *Test*) *nextcurrent-test*:
($\langle$*s0*,*s1*$\rangle$ $\models$ *skip* $\wedge$ (\$*v*=#*0*) $\wedge$ (*v*\$=\$*v*+#*1*)) = (((*v s0*) = *0*) $\wedge$ ((*v s1*) = *1*))
unfolding *current-val-d-def next-val-d-def skip-defs* **by** *auto*

lemma (**in** *Test*) *nextcurrentfinpenult-test*:
($\langle$*s0*,*s1*,*s2*,*s3*$\rangle$ $\models$ *len*(*3*) $\wedge$ *v* =: !*v*−#*1* $\wedge$ *v* $\leftarrow$ #*3* $\wedge$ \$*v*=#*0* $\wedge$ *v* := \$*v*+#*1*) =
(((*v s0*) = *0*) $\wedge$ ((*v s1*) = *1* $\wedge$ (*v* (*s2*)) = *2* $\wedge$ ((*v s3*) = *3*)))
unfolding *current-val-d-def next-val-d-def fin-val-d-def penult-val-d-def*
assign-d-def prev-assign-d-def temporal-assign-d-def len-defs **by** *auto*

lemma (**in** *Test*) *stable-test*:
($\langle$*s0*,*s1*,*s2*,*s3*$\rangle$ $\models$ *len*(*3*) $\wedge$ *stable v* $\wedge$ \$*v*=#*0*) =
((*v s0*) = *0* $\wedge$ (*v s1*) = *0* $\wedge$ (*v s2*) = *0* $\wedge$ (*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*$\rangle$ $\models$ (*len 3* $\wedge$ *v*! = !*v*−#*1* $\wedge$!*v* = #*3* $\wedge$ \$*v*=#*0* $\wedge$ *v*\$=\$*v*+#*1*)^r) =
(((*v s3*) = *0*) $\wedge$ ((*v s2*) = *1* $\wedge$ (*v* (*s1*)) = *2* $\wedge$ ((*v s0*) = *3*)))

unfolding *reverse-d-def len-defs current-val-d-def next-val-d-def*
penult-val-d-def fin-val-d-def **by** *auto*

lemma *revnextcurrentfinpenult*:
$\vdash$ *more* $\longrightarrow (v\$ = \$v)^r = (v!{=}!v)$
proof −
have *1*: $\vdash$ *more* $\longrightarrow (v\$ = \$v)^r = (\ (v\$)^r = (\$v)^r)$
by (*metis* (*mono-tags*, *lifting*) *intI inteq-reflection rev-fun2 unl-lift2*)
have *2*: $\vdash$ *more* $\longrightarrow (\ (v\$)^r = \ (v!)\ )$ **by** (*simp add*: *rev-next*)
have *3*: $\vdash$ *more* $\longrightarrow (\ (\$v)^r = \ (!v))$ **using** *rev-current* **by** *fastforce*
have *4*: $\vdash$ *more* $\longrightarrow ((\ (v\$)^r = (\$v)^r) \longrightarrow (\ (v!) = (!v)\ ))$ **using** *2 3* **by** *fastforce*
have *5*: $\vdash$ *more* $\longrightarrow ((\ (v!) = (!v)\ ) \longrightarrow \ (\ (v\$)^r = (\$v)^r)\ )$ **using** *2 3* **by** *fastforce*
have *6*: $\vdash$ *more* $\longrightarrow ((\ (v\$)^r = (\$v)^r) = (\ (v!) = (!v)\ ))$ **using** *4 5* **by** *fastforce*
from *1 6* **show** *?thesis* **by** *fastforce*
qed

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* **using** *EExI*[*unlift-rule*] **by** *blast*
qed

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* **using** *EExI*[*unlift-rule*] **by** *blast*
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* **using** *EExI*[*unlift-rule*] **by** *blast*
qed

lemma (**in** *Test1*) *test-E-F5-1*:
(
$x\ (Interval.nth\ w\ (0{::}nat)) = (0{::}nat) \wedge$
$(\forall i{<}intlen\ w.\ x\ (Interval.nth\ w\ (Suc\ i)) = Suc\ (x\ (Interval.nth\ w\ i))) \wedge$
$x\ (Interval.nth\ w\ (intlen\ w)) = n) \longrightarrow$
(
$x\ (Interval.nth\ w\ (0{::}nat)) = (0{::}nat) \wedge$
$(\forall i{\le}intlen\ w.\ x\ (Interval.nth\ w\ (i)) = i) \wedge$
$x\ (Interval.nth\ w\ (intlen\ w)) = n)$

```
apply simp
proof
  assume 0: 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
  have 1: x (Interval.nth w (0::nat)) = (0::nat)  using 0 by auto
  have 2: x (Interval.nth w (intlen w)) = n  using 0 by auto
  have 3: (∀ i<intlen w. x (Interval.nth w (Suc i)) = Suc (x (Interval.nth w i))) using 0 by auto
  show ∀ i≤intlen w. x (Interval.nth w i) = i
  proof
    fix i
    show i ≤ intlen w ⟶ x (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:
⊢ (∃∃ v. (F5 v n)) ⟶ (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


end


theory MonitorExample
imports
   FOTheorems Monitor
begin
```

10 Example

```
locale Test =
 fixes v :: state ⇒ nat
 fixes y :: state ⇒ bool
 fixes z :: state ⇒ nat
 fixes F2 :: nat statefun ⇒ temporal
```

fixes *F3* :: *bool statefun* ⇒ *temporal*
fixes *F4* :: *nat statefun* ⇒ *temporal*
fixes *F5* :: *nat statefun* ⇒ *temporal*
fixes *Init2* :: *nat statefun* ⇒ *temporal*
fixes *Init3* :: *bool statefun* ⇒ *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* ≡ (λ *v*. *TEMP* □ (#*0* ≤ $*v*))
defines *F3* ≡ (λ *p*. *TEMP* □ ($*p* ∨ ¬ $*p*))
defines *F4* ≡ (λ *z*. *TEMP* $*z*=#*0* ∧ *z gets* $*z*+#*1*)
defines *F5* ≡ (λ *z*. *TEMP fin*($*z*=#*4*))
defines *Init2* ≡ (λ *v*. *TEMP* $*v* = #*0*)
defines *Init3* ≡ (λ *p*. *TEMP* $*p*)
defines *Mon1* ≡ *FIRST*(*F2 v*)
defines *Mon2* ≡ *EMPTY UPTO Mon1*
defines *Mon3* ≡ *Mon1 WITH* (*F2 v*)
defines *Mon4* ≡ *Mon2 THEN Mon1*
defines *Mon5* ≡ *Mon3 THRU Mon4*
defines *Mon6* ≡ (*FIRST F4 z*) *WITH* (*F5 z*)

lemma (**in** *Test*) *test*:
⊢ $\mathcal{M}$(*Mon1*) = *empty*
proof −
have *1*: ⊢ $\mathcal{M}$(*Mon1*) = ▷(□ (#*0* ≤ $*v*))
    using *F2-def Mon1-def* **by** *fastforce*
have *2*: ⊢ □ (#*0* ≤ $*v*)
    by (*simp add*: *Valid-def always-defs current-val-d-def*)
have *3*: ⊢ ▷(□ (#*0* ≤ $*v*)) = *empty*
    using *2* **by** (*metis FstTrue int-eq int-eq-true*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test1*:
⊢ $\mathcal{M}$(*Mon2*) = *empty*
proof −
have *1*: ⊢ $\mathcal{M}$(*Mon2*) = $\mathcal{M}$(*EMPTY UPTO Mon1*)
    using *Mon2-def* **by** *fastforce*
have *2*: ⊢ $\mathcal{M}$(*EMPTY UPTO Mon1*) = ▷($\mathcal{M}$(*EMPTY*) ∨ $\mathcal{M}$(*Mon1*))
    by *fastforce*
have *3*: ⊢ ▷($\mathcal{M}$(*EMPTY*) ∨ $\mathcal{M}$(*Mon1*)) = ▷(*empty* ∨ *empty*)
    using *test* **by** (*metis FstEqvRule MEmptyAlt intensional-simps*(*27*) *inteq-reflection*)
have *4*: ⊢ ▷(*empty* ∨ *empty*) = *empty*
    using *FstEmptyOrEqvEmpty* **by** *blast*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test2*:
$\vdash \mathcal{M}(Mon3) = empty$
proof −
have *1*: $\vdash \mathcal{M}(Mon3) = \mathcal{M}(Mon1\ WITH\ (F2\ v))$ **using** *Mon3-def* **by** *fastforce*
have *2*: $\vdash \mathcal{M}(Mon1\ WITH\ (F2\ v)) = (\mathcal{M}(Mon1) \wedge (F2\ v))$ **by** *fastforce*
have *3*: $\vdash (\mathcal{M}(Mon1) \wedge (F2\ v)) = (empty \wedge (F2\ v))$ **using** *test* **by** *fastforce*
have *4*: $\vdash (F2\ v)$ **by** (*simp add*: *F2-def Valid-def always-defs current-val-d-def*)
have *5*: $\vdash (empty \wedge (F2\ v)) = empty$ **using** *4* **by** *fastforce*
from *1 2 3 5* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test3*:
$\vdash \mathcal{M}(Mon4) = empty$
proof −
have *1*: $\vdash \mathcal{M}(Mon4) = \mathcal{M}(Mon2\ THEN\ Mon1)$
using *Mon4-def* **by** *fastforce*
have *2*: $\vdash \mathcal{M}(Mon2\ THEN\ Mon1) = (\ \mathcal{M}(Mon2)\ ) ;(\ \mathcal{M}(Mon1)\ )$
by *fastforce*
have *3*: $\vdash (\ \mathcal{M}(Mon2)\ ) ;(\ \mathcal{M}(Mon1)\ ) = empty;empty$
using *test test1* **using** *ChopEqvChop* **by** *blast*
have *4*: $\vdash empty;\ empty = empty$
by (*simp add*: *ChopEmpty*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test4*:
$\vdash \mathcal{M}(Mon5) = empty$
proof −
have *1*: $\vdash \mathcal{M}(Mon5) = \mathcal{M}(Mon3\ THRU\ Mon4)$
using *Mon5-def* **by** *fastforce*
have *2*: $\vdash \mathcal{M}(Mon3\ THRU\ Mon4) = \rhd(di(\mathcal{M}(Mon3)) \wedge di(\mathcal{M}(Mon4)))$
by *fastforce*
have *3*: $\vdash (di(\mathcal{M}(Mon3)) \wedge di(\mathcal{M}(Mon4))) = (di(empty) \wedge di(empty))$
using *test3 test2* **by** (*metis inteq-reflection lift-and-com*)
hence *4*: $\vdash \rhd(di(\mathcal{M}(Mon3)) \wedge di(\mathcal{M}(Mon4))) = \rhd(di(empty) \wedge di(empty))$
by (*simp add*: *FstEqvRule*)
have *5*: $\vdash \rhd(di(empty) \wedge di(empty)) = \rhd(di(empty))$
by *simp*
have *6*: $\vdash \rhd(di(empty)) = empty$
using *FstDiEqvFst FstEmpty* **by** *fastforce*
from *6 5 4 2 1* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test5*:
$\vdash \mathcal{M}\ (Mon6) = (\rhd(\$z{=}\#0\ \wedge z\ gets\ \$z{+}\#1) \wedge fin(\$z{=}\#4)\ )$
proof −
have *1*: $\vdash \mathcal{M}(Mon6) = (\mathcal{M}(FIRST\ F4\ z) \wedge (F5\ z))$
using *Mon6-def* **by** *fastforce*
have *2*: $\vdash (\mathcal{M}(FIRST\ F4\ z) \wedge (F5\ z)) = (\rhd(F4\ z) \wedge fin(\$z{=}\#4))$
using *F5-def* **by** *fastforce*

```
 have 3: ⊢ (▷(F4 z) ∧ fin($z=#4)) = (▷($z=#0 ∧  z gets $z+#1) ∧ fin($z=#4) )
     using F4-def by fastforce
 from 1 2 3 show ?thesis by fastforce
qed

lemma (in Test) test5-1:
⊢ ▷($z=#0 ∧  z gets $z+#1) ∧ fin($z=#4)  ⟶
   ▷(($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4))

using FstWithAndImp by blast

lemma (in Test) test5-2:
 (s ⊨ ($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4)) =
  (z (nth s 0) =0  ∧ (∀  i< intlen s. z (nth s (Suc i)) = Suc(z (nth s i))) ∧
   z (nth s (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  ∧ (∀  i< intlen s. z (nth s (Suc i)) = Suc(z (nth s i))) ∧
   z (nth s (intlen s)) = 4)
  ⟹
 (z (nth s 0) =0  ∧ (∀  i≤ intlen s. z (nth s i) = i)
  ∧ z (nth s (intlen s)) = 4)

proof −
  assume 0: (z (nth s 0) =0  ∧ (∀  i< intlen s. z (nth s (Suc i)) = Suc(z (nth s i))) ∧
   z (nth s (intlen s)) = 4)
  show (z (nth s 0) =0  ∧ (∀  i≤ intlen s. z (nth s i) = i)
  ∧ 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: (∀  i≤ intlen s. z (nth s i) = i)
    proof
      fix i
      show i ≤ intlen s ⟶ 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 ∧ (∀ i≤ intlen s. z (nth s i) = i)
 ∧ z (nth s (intlen s)) = 4) ⟹
   (z (nth s 0) =0 ∧ (∀ i< intlen s. z (nth s (Suc i)) = Suc(z (nth s i))) ∧
  z (nth s (intlen s)) = 4)

proof −
 assume 0: (z (nth s 0) =0 ∧ (∀ i≤ intlen s. z (nth s i) = i)
  ∧ z (nth s (intlen s)) = 4)
 show (z (nth s 0) =0 ∧ (∀ i< intlen s. z (nth s (Suc i)) = Suc(z (nth s i))) ∧
   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: (∀ 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 (nth s 0) =0 ∧ (∀ i< intlen s. z (nth s (Suc i)) = Suc(z (nth s i))) ∧
   z (nth s (intlen s)) = 4)
  =
 (z (nth s 0) =0 ∧ (∀ i≤ intlen s. z (nth s i) = i)
  ∧ z (nth s (intlen s)) = 4)

using   test5-3 test5-4 by blast

lemma (in Test) test5-6 :
 (z (nth s 0) =0 ∧ (∀ i≤ intlen s. z (nth s i) = i)
  ∧ z (nth s (intlen s)) = 4) =
 (intlen s =4 ∧ (∀ i≤ intlen s. z (nth s i) = i) )

by auto

lemma (in Test) test5-7 :
 (s ⊨ ($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4)) =
  (intlen s =4 ∧ (∀ i≤ intlen s. z (nth s i) = i) )
using test5-6 test5-5 test5-2 by fastforce

lemma (in Test) test5-8 :
 (s ⊨ ▷(($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4))) =
  (
   ( (s ⊨ ($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4)) ∧ intlen s =0) ∨
   ( 0 < intlen s ∧ (s ⊨ $z=#0 ∧ z gets $z+#1 ∧ fin($z=#4)) ∧
     (∀ ia < intlen s. (prefix ia s ⊨ ¬(($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4)))))
  )

using Fstsem[of TEMP ($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4)]
by simp
```

```
lemma (in Test) test5-9 :
 ¬( (s ⊨ ($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4)) ∧ intlen s =0)
using test5-7 by simp

lemma (in Test) test5-10:
   (s ⊨ ($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4))
   ⟹
    0 < intlen s  ∧
   (∀ ia < intlen s. (prefix ia s ⊨ ¬(($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4))))

proof −
 assume 0: s ⊨ ($z=#0 ∧ z gets $z+#1) ∧ fin($z=#4)
 show 0 < intlen s  ∧
      (∀ ia < intlen s. (prefix ia s ⊨ ¬(($z=#0 ∧ z gets $z+#1) ∧ fin($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/itlhomepagese6.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] L. Lamport. *Specifying Systems — The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, Reading, Massachusetts, 2002.

[4] S. Merz. An Encoding of TLA in Isabelle. http://www.pst.informatik.uni-muenchen.de/˜merz/isabelle/. Part of the Isabelle distribution., 1998.

[5] B. Moszkowski. A Hierarchical Completeness Proof for Propositional Interval Temporal Logic with Finite Time. *Journal of Applied Non-Classical Logics*, 14(1–2):55–104, 2004.

[6] B. C. Moszkowski. Imperative reasoning in interval temporal logic. Technical report, University of Newcastle upon Tyne, 1996.

[7] M. Wenzel. Using Axiomatic Type Classes in Isabelle, May 2000.

[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.