An encoding of Interval Temporal Logic in Isabelle/HOL

Antonio Cau Ben Moszkowski David Smallwood

March 16, 2019

Abstract

These Isabelle theories introduce the semantics and syntax of Interval Temporal Logic (ITL). The ITL proof system, as introduced in [4], has been encoded and its soundness has been checked. The encoding is shallow using the Intensional Logic technique of [3]. An extensive library of ITL theorems, taken from [5], 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 ∧ (*nth l* (*intlen l*)) = (*intlen xs*) ∧ 0≤*i* ∧ *i*< (*intlen l*)
⟹ 0≤ (*nth l i*) ∧ (*nth l i*) ≤ (*nth l* (*i*+1)) ∧ (*nth l* (*i*+1)) ≤ (*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*) ∧ *ls* = *map* (*shiftm k*) *lsk*) ⟹
index-sequence 0 (*ls*) ∧ (*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*) ∧ *lsk* = *map* (*shift k*) *ls* ∧ *index-sequence 0* (*ls*)) =
(*index-sequence k* (*lsk*) ∧ *ls* = *map* (*shiftm k*) *lsk* ∧ *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*) ∧ *ls* = *map* (*shiftm k*) *lsk*) =
(*index-sequence k* (*lsk*) ∧ *ls* = *map* (*shiftm k*) *lsk* ∧
index-sequence 0 (*ls*) ∧ (*intlen ls*) =(*intlen lsk*))
using *interval-idx-shiftm* **by** *blast*

lemma *interval-idx-link*:
(*lsk* = *map* (*shift k*) *ls* ∧ *index-sequence 0* (*ls*)) =
(*lsk* = *map* (*shift k*) *ls* ∧ *index-sequence k* (*lsk*) ∧ *index-sequence 0* (*ls*)∧
(*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* ∧ *Interval.nth ls* (*intlen ls*) = *intlen* (*suffix k xs*)
shows ((*i*≤*intlen ls*) ⟶ ((*nth ls* (*i*)) ≤ (*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-sequence 0* (*ls*) ∧ (*nth* (*ls*) (*intlen* (*ls*))) = (*intlen* (*suffix k xs*))) ⟷
(*index-sequence 0* (*ls*) ∧ (*nth* (*ls*) (*intlen* (*ls*))) = (*intlen* (*suffix k xs*)) ∧
(∀ *i*. (*i*≤*intlen ls*) ⟶ ((*nth ls* (*i*)) ≤ (*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*⊙*xs*) = *x* ⊙ (*prefix m xs*)
by *auto*

lemma *interval-prefix-zero* [*simp*]:
prefix 0 (*x*⊙*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≤*intlen xs* ⟹ *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*⊙*xs*)) = *1* + *intlen*(*prefix i xs*)
using *interval-intlen-cons* **by** *auto*

lemma *interval-prefix-length* :
intlen (*prefix i xs*) = (*if i*≤ *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*≤ *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 $i \leq (intlen\ xs)$
shows $intlen\ (prefix\ i\ xs) \leq intlen\ xs$
using *assms* **by** (*induct xs*, *simp*) (*metis interval-prefix-length*)

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

lemma *interval-suffix-length-good* [*simp*]:
assumes $i \leq intlen\ xs$
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*]:
$i \leq intlen\ xs \wedge k \leq i \Longrightarrow nth\ (prefix\ i\ xs)\ k = nth\ xs\ k$
apply (*induct xs arbitrary*: *i k*, *auto*)
apply (*case-tac i*, *auto*)
apply (*case-tac k*, *auto*)
done

lemma *interval-nth-suffix* [*simp*]:
$i \leq intlen\ xs \wedge k \leq intlen\ xs - i \Longrightarrow nth\ (suffix\ i\ xs)\ k = nth\ xs\ (i+k)$
by (*induct xs arbitrary*: *i k*, *auto*) (*case-tac i*, *auto*)

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

lemma *interval-suffix-prefix-help-3*:
assumes $ia+i \leq intlen\ xs$
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 $ia+i \leq intlen\ xs$
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 $i \leq intlen\ (prefix\ (intlen\ xs - Suc\ 0)\ xs)$
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 $intlen\ xs > 0 \wedge ia < intlen\ (xs)$
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 $i > 0 \wedge i \leq intlen\ xs$
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 $0 \leq i \wedge i \leq j \wedge j \leq$ *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 $0 \leq i \wedge i \leq j \wedge j \leq$ *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 $0 \leq i \wedge i \leq j \wedge j \leq$ *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 $0 \leq i \wedge i \leq j \wedge j \leq$ *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-sequence 0 l* ∧ (*nth l* (*intlen l*)) = (*intlen xs*) ∧
$0 \leq i \wedge i <$ (*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-sequence 0 l* ∧ (*nth l* (*intlen l*)) = (*intlen xs*) ∧
ls = *map* (λ *x*. (*intlen xs*) $-x$) (*intrev l*)
shows (*nth ls 0*) = 0 ∧ (*nth ls* (*intlen ls*)) = (*intlen xs*) ∧ *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-sequence 0 l ∧ (*nth l* (*intlen l*)) = (*intlen xs*) ∧
ls = *map* (λ *x*. (*intlen xs*) $-x$) (*intrev l*)
⟹ $i \leq$ *intlen ls* ⟶ (*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-sequence 0 l* ∧ (*nth l* (*intlen l*)) = (*intlen xs*))

shows $(i < intlen\ l \longrightarrow$
$(intlen\ xs) - (nth\ l\ ((intlen\ l) - i)) < (intlen\ xs) - (nth\ l\ ((intlen\ l) - (i+1))))$
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 $(index\text{-}sequence\ 0\ l \wedge (nth\ l\ (intlen\ l)) = (intlen\ xs) \wedge$
$ls = map\ (\lambda\ x.\ (intlen\ xs) - x)\ (intrev\ l))$
shows $(i < intlen\ ls \longrightarrow$
$((nth\ ls\ i) = (intlen\ xs) - (nth\ l\ ((intlen\ l) - i)) \wedge$
$(nth\ ls\ (i+1)) = (intlen\ xs) - (nth\ l\ ((intlen\ l) - (i+1))) \wedge$
$(nth\ ls\ i) < (nth\ ls\ (i+1))))$
proof −
have *1*: $(i < intlen\ ls \longrightarrow (nth\ ls\ i) = (intlen\ xs) - (nth\ l\ ((intlen\ l) - i)))$
using *assms interval-intrev-idx-4 less-imp-le-nat* **by** *blast*
have *2*: $(i < intlen\ ls \longrightarrow (nth\ ls\ (i+1)) = (intlen\ xs) - (nth\ l\ ((intlen\ l) - (i+1))))$
using *assms* **by** (*simp add*: *interval-intrev-idx-4*)
have *3*: $(\ i < intlen\ ls \longrightarrow$
$((nth\ ls\ i) = (intlen\ xs) - (nth\ l\ ((intlen\ l) - i)) \wedge$
$(nth\ ls\ (i+1)) = (intlen\ xs) - (nth\ l\ ((intlen\ l) - (i+1)))))$
using *1 2* **by** *auto*
have *4*: $(i < intlen\ ls \longrightarrow$
$((nth\ ls\ i) = (intlen\ xs) - (nth\ l\ ((intlen\ l) - i)) \wedge$
$(nth\ ls\ (i+1)) = (intlen\ xs) - (nth\ l\ ((intlen\ l) - (i+1))) \wedge$
$(nth\ ls\ i) < (nth\ ls\ (i+1))))$
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 $(index\text{-}sequence\ 0\ l \wedge (nth\ l\ (intlen\ l)) = (intlen\ xs) \wedge$
$ls = map\ (\lambda\ x.\ (intlen\ xs) - x)\ (intrev\ l))$
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 $index\text{-}sequence\ 0\ l \wedge (nth\ l\ (intlen\ l)) = (intlen\ xs) \wedge$
$ls = map\ (\lambda\ x.\ (intlen\ xs) - x)\ (intrev\ l) \wedge index\text{-}sequence\ 0\ ls$
shows $i < intlen\ ls \longrightarrow$
$(intlen\ xs) - (nth\ l\ (i+1)) = nth\ ls\ ((intlen\ ls) - (i+1)) \wedge$
$(intlen\ xs) - (nth\ l\ i) \quad = (nth\ ls\ ((intlen\ ls) - i))$
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 Semantics

theory *Semantics*
imports *Interval HOL−TLA.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., [6] for a discussion about deep vs. shallow embeddings in Isabelle/HOL. The choice of a shallow over a deep embedding is motivated [3, 2] by the following factors: a shallow embedding is usually less involved, and existing Isabelle theories and tools can be applied more directly to enhance automation; due to the lifting in the *Intensional* theory, a shallow embedding can reuse standard logical operators, whilst a deep embedding requires a different set of operators for formulas. Finally, since our target is system verification rather than proving meta-properties of the logic, which requires a deep embedding, a shallow embedding is more fit for purpose.

2.1 Types of Formulas

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

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

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

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

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

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

2.2 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*)

2.2.1 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*) ⇒ -)

2.3 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*)

2.3.1 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 *initonly-d* :: ('*a*::*world*) *formula* ⇒ '*a formula*

where *initonly-d F* ≡ *LIFT*(*bi*(*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*)
-initonly-d :: *lift* ⇒ *lift* ((*initonly -*) [*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*)
-initonly-d :: *lift* ⇒ *lift* ((*initonly -*) [*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 $\rightleftharpoons$ *CONST bm-d*
-init-d $\rightleftharpoons$ *CONST init-d*
-fin-d $\rightleftharpoons$ *CONST fin-d*
-halt-d $\rightleftharpoons$ *CONST halt-d*
-initonly-d $\rightleftharpoons$ *CONST initonly-d*
-keep-d $\rightleftharpoons$ *CONST keep-d*
-yields-d $\rightleftharpoons$ *CONST yields-d*
-ifthenelse-d $\rightleftharpoons$ *CONST ifthenelse-d*
-len-d $\rightleftharpoons$ *CONST len-d*
-power-chop-d $\rightleftharpoons$ *CONST power-chop-d*
-power-d $\rightleftharpoons$ *CONST power-d*

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

definition *while-d* :: (*'a*::*world*) *formula* $\Rightarrow$ *'a formula* $\Rightarrow$ *'a formula*
where *while-d F G* $\equiv$ *LIFT*((*F* $\wedge$ *G*)* $\wedge$ (*fin* ((¬*F*))))

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

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

translations
-ifthen-d $\rightleftharpoons$ *CONST ifthen-d*
-while-d $\rightleftharpoons$ *CONST while-d*

definition *repeat-d* :: (*'a*::*world*) *formula* $\Rightarrow$ *'a formula* $\Rightarrow$ *'a formula*
where *repeat-d F G* $\equiv$ *LIFT*(*F*;*while* (¬ *G*) *do F*)

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

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

translations
-repeat-d $\rightleftharpoons$ *CONST repeat-d*

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

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

where *prev-assign-d v e* ≡ *LIFT*(*v*! = *e*)

definition *always-eq-d* :: (*'a*::*world*,*'b*) *stfun* ⇒ (*'a*,*'b*) *formfun* ⇒ *'a formula*
where *always-eq-d v e* ≡ λ *s*. *s* ⊨ □($*v* = *e*)

definition *temporal-assign-d* :: (*'a*::*world*,*'b*) *stfun* ⇒ (*'a*,*'b*) *formfun* ⇒ *'a formula*
where *temporal-assign-d v e* ≡ λ *s*. *s* ⊨ !*v* = *e*

definition *gets-d* :: (*'a*::*world*,*'b*) *stfun* ⇒ (*'a*,*'b*) *formfun* ⇒ *'a formula*
where *gets-d v e* ≡ λ *s*. *s* ⊨ *keep*(*temporal-assign-d v e*)

definition *stable-d* :: (*'a*::*world*,*'b*) *stfun* ⇒ *'a formula*
where *stable-d v* ≡ λ *s*. *s* ⊨ *gets-d v* (*current-val-d v*)

definition *padded-d* :: (*'a*::*world*,*'b*) *stfun* ⇒ *'a formula*
where *padded-d v* ≡ λ *s*. *s* ⊨ (*stable-d v*);*skip* ∨ *empty*

definition *padded-temp-assign-d* :: (*'a*::*world*,*'b*) *stfun* ⇒ (*'a*,*'b*) *formfun* ⇒ *'a formula*
where *padded-temp-assign-d v e* ≡ λ *s*. *s* ⊨ (*temporal-assign-d v e*) ∧ (*padded-d v*)

syntax
-next-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- := -) [*50*,*51*] *50*)
-prev-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- =: -) [*50*,*51*] *50*)
-always-eq-d :: [*lift*,*lift*] ⇒ *lift* ((- ≈ -) [*50*,*51*] *50*)
-temporal-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- ← -) [*50*,*51*] *50*)
-gets-d :: [*lift*,*lift*] ⇒ *lift* ((- *gets* -) [*50*,*51*] *50*)
-stable-d :: *lift* ⇒ *lift* ((*stable* -) [*51*] *50*)
-padded-d :: *lift* ⇒ *lift* ((*padded* -) [*51*] *50*)
-padded-temp-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- <∼ -) [*50*,*51*] *50*)

syntax (*ASCII*)
-next-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- := -) [*50*,*51*] *50*)
-prev-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- =: -) [*50*,*51*] *50*)
-always-eq-d :: [*lift*,*lift*] ⇒ *lift* ((- *alweqv* -) [*50*,*51*] *50*)
-temporal-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- <−− -) [*50*,*51*] *50*)
-gets-d :: [*lift*,*lift*] ⇒ *lift* ((- *gets* -) [*50*,*51*] *50*)
-stable-d :: *lift* ⇒ *lift* ((*stable* -) [*51*] *50*)
-padded-d :: *lift* ⇒ *lift* ((*padded* -) [*51*] *50*)
-padded-temp-assign-d :: [*lift*,*lift*] ⇒ *lift* ((- <~ -) [*50*,*51*] *50*)

translations
-next-assign-d ⇌ *CONST next-assign-d*
-prev-assign-d ⇌ *CONST prev-assign-d*
-always-eq-d ⇌ *CONST always-eq-d*
-temporal-assign-d ⇌ *CONST temporal-assign-d*
-gets-d ⇌ *CONST gets-d*
-stable-d ⇌ *CONST stable-d*
-padded-d ⇌ *CONST padded-d*
-padded-temp-assign-d ⇌ *CONST padded-temp-assign-d*

2.4 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* :
$(w \models F1\ ;\ F2) = (\exists\ n\ .\ 0 \leq n \wedge n \leq intlen\ w \wedge ((prefix\ n\ w) \models F1) \wedge (\ (suffix\ n\ w) \models F2)\ \ )$
by (*simp add*: *chop-d-def*)

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

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

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

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

lemma *da-defs* :
$(w \models da\ F) = (\exists\ n\ na.\ 0 \leq n \wedge\ \ na+n \leq intlen\ w \wedge ((sub\ n\ (na+n)\ w) \models F))$
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* :
$(w \models ba\ F) = (\forall\ n\ na.\ 0 \leq n \wedge\ \ na+n \leq intlen\ w \longrightarrow ((sub\ n\ (na+n)\ w) \models F))$
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 *initalt-defs* :
$(w \models bi(\ empty \longrightarrow F)) = (\ (Interval.prefix\ 0\ w) \models F\ )$
by (*simp add*: *bi-defs empty-defs*)

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

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

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

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

lemma *prev-assign-defs* :
$intlen\ s > 0 \Longrightarrow (s \models v =: e) = v\ (Interval.nth\ s\ ((intlen\ s)-1)) = e\ s$
by (*auto simp*: *prev-assign-d-def penult-val-d-def*)

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

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

lemma *gets-defs* :
$(s \models v\ gets\ e) = (\forall\ i < intlen\ s.\ v\ (Interval.nth\ s\ (Suc\ i)) = e\ (sub\ i\ (i+1)\ s)\ )$
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*:
$(\forall i < intlen\ s.\ v\ (Interval.nth\ s\ (Suc\ i)) = v\ (Interval.nth\ s\ i)) =$
$(\forall i \leq intlen\ s.\ v\ (Interval.nth\ s\ i) = v\ (Interval.nth\ s\ 0))$
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*:
$(s \models \mathit{stable}\ v) = (\forall\ i{\leq}\mathit{intlen}\ s.\ (v\ (\mathit{nth}\ s\ i)) = (v\ (\mathit{nth}\ s\ 0)))$
by (*simp add*: *stable-d-def gets-defs current-val-d-def sub-def stable-defs-help*)

lemma *padded-defs* :
$(s \models \mathit{padded}\ v) = ((\forall\ i{<}\ \mathit{intlen}\ s.\ (v\ (\mathit{nth}\ s\ i)) = (v\ (\mathit{nth}\ s\ 0))) \vee \mathit{intlen}\ s = 0)$
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* :
$(s \models v <\sim e) =$
$((s \models \mathit{padded}\ v) \wedge$
$(v\ (\mathit{Interval.nth}\ s\ (\mathit{intlen}\ s)) = e\ s\ ))$
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)

2.5 Soundness Axioms

2.5.1 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 ... =
$(\exists\, i{\leq}\mathit{intlen}\ \sigma.\ (\exists\, ia{\leq}\mathit{intlen}\ (\mathit{prefix}\ i\ \sigma).\ \ (\mathit{prefix}\ ia\ (\mathit{prefix}\ i\ \sigma) \models f) \wedge$

(*suffix ia* (*prefix i* σ) $\models$ *g*)) $\wedge$ (*suffix i* σ $\models$ *h*))
by *fastforce*
also have ... =
(σ $\models$ (*f*;*g*);*h*) **by** (*simp add*: *chop-defs*)
finally show (σ $\models$ *f* ; (*g* ; *h*)) = (σ $\models$ (*f*;*g*);*h*) **.**
qed

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

2.5.2 OrChopImp

lemma *OrChopImpSem*:
(σ $\models$ (*f* $\vee$ *g*);*h* $\longrightarrow$ *f*;*h* $\vee$ *g*;*h*)
by (*simp add*: *chop-defs*) *blast*

2.5.3 ChopOrImp

lemma *ChopOrImpSem*:
(σ $\models$ *f*;(*g* $\vee$ *h*) $\longrightarrow$ *f*;*g* $\vee$ *f*;*h*)
by (*simp add*: *chop-defs*) *blast*

2.5.4 EmptyChop

lemma *EmptyChopSem*:
(σ $\models$ *empty* ; *f* = *f*)
by (*simp add*: *empty-defs chop-defs*) *auto*

2.5.5 ChopEmpty

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

2.5.6 StateImpBi

lemma *StateImpBiSem*:
(σ $\models$ *init f* $\longrightarrow$ *bi* (*init f*))
by (*simp add*: *init-defs bi-defs*)

2.5.7 NextImpNotNextNot

lemma *NextImpNotNextNotSem*:
(σ $\models$ $\bigcirc$ *f* $\longrightarrow$ $\neg$ ($\bigcirc$ ($\neg$ *f*)))
by (*simp add*: *next-defs*)

2.5.8 BiBoxChopImpChop

lemma *BiBoxChopImpChopSem*:
(σ $\models$ *bi* (*f* $\longrightarrow$ *f1*) $\wedge$ $\Box$(*g* $\longrightarrow$ *g1*) $\longrightarrow$ *f*;*g* $\longrightarrow$ *f1*;*g1*)
by (*simp add*: *bi-defs always-defs chop-defs*) *fastforce*

2.5.9 BoxInduct

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

2.5.10 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*:
$(\exists\ (l::index).\ (intlen\ l) = (Suc\ n) \wedge index\text{-}sequence\ 0\ l \wedge (nth\ l\ (intlen\ l)) = (intlen\ \sigma) \wedge$
$(\forall i.\ (0 \leq i \wedge i < (intlen\ l)) \longrightarrow$
$((sub\ (nth\ l\ i)\ (nth\ l\ (i+1))\ \sigma) \models f)$

```
                    )
              ) =
  (∃ 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)
               ))
       )
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  ... =
```

$$
\begin{array}{l}
\quad (\exists\ x\ ls\ .\ (intlen\ ls) = n \wedge x = 0 \wedge index\text{-}sequence\ (nth\ ls\ 0)\ (ls) \wedge \\
\qquad (nth\ (ls)\ (intlen\ (ls))) = (intlen\ \sigma) \wedge \\
\qquad (x < (nth\ ls\ 0) \wedge \\
\qquad ((\forall\, i.\ (0 \leq i \wedge i < (intlen\ (x \odot ls))) \longrightarrow \\
\qquad\quad ((sub\ (nth\ (x \odot ls)\ i)\ (nth\ (x \odot ls)\ (i+1))\ \sigma) \models f)) \\
\qquad\ ) \\
\qquad) \\
\quad)
\end{array}
$$

using *interval-idx-cons* **by** *auto*
also have ... =

$$
\begin{array}{l}
\quad (\exists\ x\ ls\ .\ (intlen\ ls) = n \wedge x = 0 \wedge index\text{-}sequence\ (nth\ ls\ 0)\ (ls) \wedge \\
\qquad (nth\ (ls)\ (intlen\ (ls))) = (intlen\ \sigma) \wedge \\
\qquad (x < (nth\ ls\ 0) \wedge \\
\qquad ((sub\ (nth\ (x \odot ls)\ 0)\ (nth\ (x \odot ls)\ (1))\ \sigma) \models f) \\
\qquad \wedge \\
\qquad ((\forall\, i.\ (0 < i \wedge i < 1 + (intlen\ (ls))) \longrightarrow \\
\qquad\quad ((sub\ (nth\ (x \odot ls)\ i)\ (nth\ (x \odot ls)\ (i+1))\ \sigma) \models f)) \\
\qquad\ ) \\
\qquad) \\
\quad)
\end{array}
$$

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 ... =

$$
\begin{array}{l}
\quad (\exists\ x\ ls\ .\ (intlen\ ls) = n \wedge x = 0 \wedge index\text{-}sequence\ (nth\ ls\ 0)\ (ls) \wedge \\
\qquad (nth\ (ls)\ (intlen\ (ls))) = (intlen\ \sigma) \wedge \\
\qquad (x < (nth\ ls\ 0) \wedge (nth\ (x \odot ls)\ 0) = x \wedge (nth\ (x \odot ls)\ (1)) = (nth\ ls\ 0) \wedge \\
\qquad ((sub\ (nth\ (x \odot ls)\ 0)\ (nth\ (x \odot ls)\ (1))\ \sigma) \models f) \\
\qquad \wedge \\
\qquad ((\forall\, i.\ (0 < i \wedge i < 1 + (intlen\ (ls))) \longrightarrow \\
\qquad\quad ((sub\ (nth\ (x \odot ls)\ i)\ (nth\ (x \odot ls)\ (i+1))\ \sigma) \models f)) \\
\qquad\ ) \\
\qquad) \\
\quad)
\end{array}
$$

by *auto*
also have ... =

$$
\begin{array}{l}
\quad (\exists\ x\ ls\ .\ (intlen\ ls) = n \wedge x = 0 \wedge index\text{-}sequence\ (nth\ ls\ 0)\ (ls) \wedge \\
\qquad (nth\ (ls)\ (intlen\ (ls))) = (intlen\ \sigma) \wedge \\
\qquad (x < (nth\ ls\ 0) \wedge (nth\ (x \odot ls)\ 0) = x \wedge (nth\ (x \odot ls)\ (1)) = (nth\ ls\ 0) \wedge \\
\qquad ((sub\ x\ (nth\ ls\ 0)\ \sigma) \models f) \\
\qquad \wedge \\
\qquad ((\forall\, i.\ (0 < i \wedge i < 1 + (intlen\ (ls))) \longrightarrow \\
\qquad\quad ((sub\ (nth\ (x \odot ls)\ i)\ (nth\ (x \odot ls)\ (i+1))\ \sigma) \models f)) \\
\qquad\ ) \\
\qquad) \\
\quad)
\end{array}
$$

by *auto*
also have ... =

$$
\begin{array}{l}
\quad (\exists\ x\ ls\ .\ (intlen\ ls) = n \wedge x = 0 \wedge index\text{-}sequence\ (nth\ ls\ 0)\ (ls) \wedge \\
\qquad (nth\ (ls)\ (intlen\ (ls))) = (intlen\ \sigma) \wedge \\
\qquad (x < (nth\ ls\ 0) \wedge
\end{array}
$$

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

by *auto*

also have $\ldots =$

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

using *interval-nth-cons* **by** *metis*

also have $\ldots =$

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

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

also have $\ldots =$

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

by *simp*

also have $\ldots =$

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

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 $\ldots =$

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

$(\exists\ k\ .\ 0 \leq k \wedge k \leq intlen\ \sigma \wedge k > 0 \wedge$
$(sub\ 0\ k\ \sigma \models f) \wedge$
$(\exists\ ls.\ (intlen\ ls) = n\ \wedge index\text{-}sequence\ 0\ (ls) \wedge$
$(nth\ (ls)\ (intlen\ (ls))) = (intlen\ (suffix\ k\ \sigma)) \wedge$
$(\forall i{\leq}intlen\ ls.\ Interval.nth\ ls\ i \leq intlen\ (suffix\ k\ \sigma)) \wedge$
$(\forall i.\ (0{\leq}i \wedge i{<}\ (intlen\ ls)) \longrightarrow$
$((sub\ ((nth\ ls\ (i))){+}k)\ ((nth\ ls\ ((i){+}1)){+}k)\ (\sigma)) \models f)$
$)$
$)$
$)$
using *interval-idx-bound-1* **by** *blast*
also have $... =$
$(\exists\ k.\ 0 \leq k \wedge k \leq intlen\ \sigma \wedge k > 0 \wedge$
$(sub\ 0\ k\ \sigma \models f) \wedge$
$(\exists\ ls.\ (intlen\ ls) = n\ \wedge index\text{-}sequence\ 0\ (ls) \wedge$
$(nth\ (ls)\ (intlen\ (ls))) = (intlen\ (suffix\ k\ \sigma)) \wedge$
$(\forall i{\leq}intlen\ ls.\ Interval.nth\ ls\ i \leq intlen\ (suffix\ k\ \sigma))$
$\wedge\ (\forall i.\ (0{\leq}i \wedge i{<}\ (intlen\ ls)) \longrightarrow$
$((sub\ (nth\ ls\ (i))\ (nth\ ls\ ((i){+}1))\ (suffix\ k\ \sigma)) \models f)$
$)$
$)$
$)$
by (*smt add.commute index-sequence-def interval-idx-expand interval-sub-suffix*
interval-suffix-length-good plus-1-eq-Suc)
also have $... =$
$(\exists\ k.\ 0 \leq k \wedge k \leq intlen\ \sigma \wedge k > 0 \wedge$
$(sub\ 0\ k\ \sigma \models f) \wedge$
$(\exists\ ls.\ (intlen\ ls) = n\ \wedge index\text{-}sequence\ 0\ (ls) \wedge$
$(nth\ (ls)\ (intlen\ (ls))) = (intlen\ (suffix\ k\ \sigma))$
$\wedge\ (\forall i.\ (0{\leq}i \wedge i{<}\ (intlen\ ls)) \longrightarrow$
$((sub\ (nth\ ls\ (i))\ (nth\ ls\ ((i){+}1))\ (suffix\ k\ \sigma)) \models f)$
$))$
$)$
using *interval-idx-bound-1* **by** *blast*
finally show $(\exists\ (l{::}index).\ (intlen\ l) = (Suc\ n) \wedge index\text{-}sequence\ 0\ l \wedge$
$(nth\ l\ (intlen\ l)) = (intlen\ \sigma) \wedge$
$(\forall i.\ (0{\leq}i \wedge i{<}\ (intlen\ l)) \longrightarrow$
$((sub\ (nth\ l\ i)\ (nth\ l\ (i{+}1))\ \sigma) \models f)$
$)$
$) =$
$(\exists\ k.\ 0 \leq k \wedge k \leq intlen\ \sigma \wedge k > 0 \wedge (sub\ 0\ k\ \sigma \models f) \wedge$
$(\exists\ ls.\ (intlen\ ls) = n \wedge index\text{-}sequence\ 0\ (ls) \wedge$
$(nth\ (ls)\ (intlen\ (ls))) = (intlen\ (suffix\ k\ \sigma)) \wedge$
$(\forall i.\ (0{\leq}i \wedge i{<}\ (intlen\ ls)) \longrightarrow$
$((sub\ (nth\ ls\ (i))\ (nth\ ls\ ((i){+}1))\ (suffix\ k\ \sigma)) \models f)$
$)$
$)$
$)$
.
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)
```

2.6 Quantification over State (Flexible) Variables

The hidden state approach, as used in the embedding of TLA in Isabelle/HOL TLA embedding [3, 2], is used. Here [3, 2], a state space is defined by its projections, and everything else is unknown. Thus, a variable is a projection of the state space, and has the same type as a state function. Moreover, strong typing is achieved, since the projection function may have any result type. To achieve this, the state space is represented by an undefined type, which is an instance of the *world* class to enable use with the *Intensional* theory.

```
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 [3, 2] we define a state to be an anonymous type whose only purpose is to provide Skolem constants. Similarly, we do not define a type of state variables separate from that of arbitrary state functions, again in order to simplify the definition of flexible quantification later on. Nevertheless, we need to distinguish state variables. Note we deviate from [3, 2] in that we do not use axioms but use definitions and lemmas.

2.7 Temporal Quantifiers

definition *exist-state-d* :: (*'a statefun* ⇒ *temporal*)⇒ *temporal* (**binder** *Eex* *10*)
where *exist-state-d* *F* ≡ (λ*s*. (∃ *x*. *s* ⊨ *F x*))

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

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

definition *forall-state-d* :: (*'a statefun* ⇒ *temporal*)⇒ *temporal* (**binder** *Aall* *10*)
where *forall-state-d F* ≡ *LIFT*(¬(∃∃ *x*. ¬(*F x*)))

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

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

2.8 Unlifting attributes and methods

The following is again from [3, 2] but adapted for our need.

lemma *int-eq-true*: ⊢ *P* ⟹ ⊢ *P* = #*True*
by *auto*

lemma *int-eq*: ⊢ *X* = *Y* ⟹ *X* = *Y*
by (*auto simp*: *inteq-reflection*)

lemma *int-iffI*:
assumes ⊢ *F* ⟶ *G* **and** ⊢ *G* ⟶ *F*
shows ⊢ *F* = *G*
using *assms* **by** *force*

lemma *int-iffD1*: **assumes** *h*: ⊢ *F* = *G* **shows** ⊢ *F* ⟶ *G*
using *h* **by** *auto*

lemma *int-iffD2*: **assumes** *h*: ⊢ *F* = *G* **shows** ⊢ *G* ⟶ *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*

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

3 Axioms and Rules

The ITL axiom and proof rules are introduced (taken from [4]) together with the validity operation. The soundness of the rules and axioms are checked using the lemmas of Semantics.thy.

3.1 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*)

3.2 Axioms

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

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

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

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

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

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

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

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

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

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

3.3 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*

3.4 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-forall*: $\vdash \$(\forall\ x.\ P\ x) = (\forall\ x.\ \$(P\ x))$
by (*auto simp*: *current-val-d-def*)

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

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

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

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

3.5 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-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-forall next-exists next-exists1

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

3.6 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-forall*: $\vdash\ !(\forall\ x.\ P\ x) = (\forall\ x.\ !(P\ x))$
by (*auto simp*: *fin-val-d-def*)

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

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

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

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

3.7 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-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-forall penult-exists penult-exists1

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

3.8 Basic temporal variables properties

lemma *empty-imp-fin-eqv-curr*:
$\vdash$ *empty* $\longrightarrow$ $!v = \$v$
by (*simp add*: *Valid-def current-val-d-def empty-defs finval-defs*)

lemma *skip-imp-fin-eqv-next*:
$\vdash$ *skip* $\longrightarrow$ $!v = v\$$
by (*simp add*: *Valid-def skip-defs next-val-d-def finval-defs*)

lemma *skip-imp-penult-eqv-curr*:
$\vdash$ *skip* $\longrightarrow$ $v! = \$v$
by (*simp add*: *Valid-def skip-defs penultval-defs current-val-d-def*)

3.9 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-forall*:

$\vdash (\forall\ x.\ P\ x)^r = (\forall\ x.\ (P\ x)^r)$
by (*auto simp*: *reverse-d-def*)

lemma *rev-exists*:
$\vdash (\exists\ x.\ P\ x)^r = (\exists\ x.\ (P\ x)^r)$
by (*auto simp*: *reverse-d-def*)

lemma *rev-exists1*:
$\vdash (\exists!\ x.\ P\ x)^r = (\exists!\ x.\ (P\ x)^r)$
by (*auto simp*: *reverse-d-def*)

lemma *rev-current*:
$\vdash (\$v)^r = (!v)$
by (*auto simp*: *interval-intrev-nth current-val-d-def fin-val-d-def reverse-d-def*)

lemma *rev-next*:
$\vdash\ (v\$)^r = (v!)$
by (*auto simp*: *interval-intrev-nth next-val-d-def penult-val-d-def reverse-d-def*)

lemma *rev-penult*:
$\vdash\ (v!)^r = (v\$)$
by (*auto simp*: *interval-intrev-nth next-val-d-def penult-val-d-def reverse-d-def*)

lemma *rev-fin*:
$\vdash (!v)^r = (\$v)$
by (*auto simp*: *interval-intrev-nth fin-val-d-def current-val-d-def reverse-d-def*)

lemma *EqvReverseReverse*:
$\vdash (f^r)^r = f$
by (*simp add*: *Valid-def reverse-d-def*)

lemma *ReverseEqv*:
$(\vdash f) \longleftrightarrow (\vdash f^r)$
by (*metis Valid-def interval-rev-swap reverse-d-def*)

lemma *RevSkip*:
$\vdash skip^r = skip$
by (*simp add*: *Valid-def reverse-d-def skip-defs*)

lemma *RevChop*:
$\vdash (f;g)^r = (g^r;f^r)$
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-forall rev-exists*
rev-exists1 rev-current rev-next rev-penult rev-fin RevSkip RevChop RevChopstar

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

end

theory *Theorems*
imports
ITL
begin

4 ITL theorems

We give the proofs of a list of ITL theorems. These proofs and theorems were from [5].

4.1 Propositional reasoning

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

lemma *IfThenElseImp*:
$\vdash (\mathit{if}_i\ \ g\ \ \mathit{then}\ \ f\ \ \mathit{else}\ f1) \longrightarrow ((\ g \longrightarrow f\ ) \wedge (\neg g \longrightarrow f1))$
by (*simp add*: *ifthenelse-defs Valid-def*)

lemma *Prop01*:
assumes $\vdash f \longrightarrow \neg\ g \vee h$
shows $\vdash g \wedge f \longrightarrow h$
using *assms* **by** *auto*

lemma *Prop02*:
assumes $\vdash f \longrightarrow g$
$\vdash f1 \longrightarrow g$
shows $\vdash f \vee f1 \longrightarrow g$
using *assms*(*1*) *assms*(*2*) **by** *fastforce*

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

lemma *Prop04*:
assumes $\vdash f = h$
$\vdash f = h1$
shows $\vdash h1 = h$
using *assms*(*1*) *assms*(*2*) **using** *int-eq* **by** *auto*

lemma *Prop05*:
assumes $\vdash f \longrightarrow g$
shows $\vdash f \longrightarrow h \vee g$
using *assms* **by** *auto*

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

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

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

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

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

lemma *Prop11*:
$(\vdash f = f1) = (\ (\vdash f \longrightarrow f1) \wedge\ (\vdash f1 \longrightarrow f)\ )$
by (*auto simp*: *Valid-def*)

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

4.2 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*)

4.3 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*: $\vdash g \longrightarrow g \vee g1$ **by** *auto*
hence *2*: $\vdash f;g \longrightarrow f;(g \vee g1)$ **by** (*rule RightChopImpChop*)
have *3*: $\vdash g1 \longrightarrow g \vee g1$ **by** *auto*
hence *4*: $\vdash f;g1 \longrightarrow f;(g \vee g1)$ **by** (*rule RightChopImpChop*)
from *2 4* **show** *?thesis* **by** (*meson ChopOrImp Prop02 Prop11*)
qed

lemma *OrChopEqv*:
$\vdash (f \vee f1);g = (f;g \vee f1;g)$
proof −
have *1*: $\vdash f \longrightarrow f \vee f1$ **by** *auto*
hence *2*: $\vdash f;g \longrightarrow (f \vee f1);g$ **by** (*rule LeftChopImpChop*)
have *3*: $\vdash f1 \longrightarrow f \vee f1$ **by** *auto*
hence *4*: $\vdash f1;g \longrightarrow (f \vee f1);g$ **by** (*rule LeftChopImpChop*)

from *2 4* **show** *?thesis*
by (*meson OrChopImp int-iffI Prop02*)
qed

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

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

lemma *OrChopEqvRule*:
assumes $\vdash f = (f1 \vee f2)$
shows $\vdash f;g = ((f1;g) \vee (f2;g))$
proof −
have *1*: $\vdash f = (f1 \vee f2)$ **using** *assms* **by** *auto*
hence *2*: $\vdash f;g = ((f1 \vee f2);g)$ **by** (*rule LeftChopEqvChop*)
have *3*: $\vdash (f1 \vee f2);g = (f1;g \vee f2;g)$ **by** (*rule OrChopEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *NextImpNext*:
assumes $\vdash f \longrightarrow g$
shows $\vdash \bigcirc f \longrightarrow \bigcirc g$
proof −
have *1*: $\vdash f \longrightarrow g$ **using** *assms* **by** *auto*
hence *2*: $\vdash \Box\ (f \longrightarrow g)$ **by** (*rule BoxGen*)
have *3*: $\vdash \Box\ (f \longrightarrow g) \longrightarrow (skip;f) \longrightarrow (skip;g)$ **by** (*rule BoxChopImpChop*)
have *4*: $\vdash (skip;f) \longrightarrow (skip;g)$ **by** (*metis 2 3 MP*)
from *4* **show** *?thesis* **by** (*metis next-d-def*)
qed

lemma *ChopOrImpRule*:
assumes $\vdash g \longrightarrow g1 \vee g2$
shows $\vdash f;g \longrightarrow (f;g1) \vee (f;g2)$

proof −
have *1*: ⊢ *g* ⟶ *g1* ∨ *g2* **using** *assms* **by** *auto*
hence *2*: ⊢ *f*;*g* ⟶ *f*;(*g1* ∨ *g2*) **by** (*rule RightChopImpChop*)
have *3*: ⊢ *f*;(*g1* ∨ *g2*) = (*f*;*g1* ∨ *f*;*g2*) **by** (*rule ChopOrEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *NextImpDist*:
⊢ ○ (*f* ⟶ *g*) ⟶ ○ *f* ⟶ ○ *g*
proof −
have *1*: ⊢ (¬ (*f* ⟶ *g*)) = (*f* ∧ ¬ *g*) **by** *auto*
hence *2*: ⊢ *skip*;(¬ (*f* ⟶ *g*)) = *skip*;(*f* ∧ ¬ *g*) **by** (*rule RightChopEqvChop*)
have *3*: ⊢ *f* ⟶ *g* ∨ (*f* ∧ ¬ *g*) **by** *auto*
hence *4*: ⊢ *skip*;*f* ⟶ (*skip*;*g*) ∨ (*skip*;(*f* ∧ ¬ *g*)) **by** (*rule ChopOrImpRule*)
hence *5*: ⊢ ¬ (*skip*;(*f* ∧ ¬ *g*)) ⟶ (*skip*;*f*) ⟶ (*skip*;*g*) **by** *auto*
have *6*: ⊢ ¬ (*skip*;(¬(*f* ⟶ *g*))) ⟶ (*skip*;*f*) ⟶ (*skip*;*g*) **using** *2 5* **by** *fastforce*
hence *7*: ⊢ ¬ (○(¬(*f* ⟶ *g*))) ⟶ (○ *f*) ⟶ (○ *g*) **by** (*simp add*: *next-d-def*)
have *8*: ⊢ ○(*f* ⟶ *g*) ⟶ ¬ (○(¬(*f* ⟶ *g*))) **by** (*rule NextImpNotNextNot*)
from *7 8* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *ChopImpDiamond*:
⊢ *f*;*g* ⟶ ◇ *g*
proof −
have *1*: ⊢ *f* ⟶ #*True* **by** *auto*
hence *2*: ⊢ *f*;*g* ⟶ #*True*;*g* **by** (*rule LeftChopImpChop*)
from *2* **show** *?thesis* **by** (*simp add*: *sometimes-d-def*)
qed

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

lemma *BoxElim*:
⊢ □ *f* ⟶ *f*
proof −
have *1*: ⊢ ¬ *f* ⟶ ◇ (¬ *f*) **by** (*rule NowImpDiamond*)
hence *2*: ⊢ ¬ (◇ (¬ *f*)) ⟶ *f* **by** *auto*
from *2* **show** *?thesis* **by** (*metis always-d-def*)
qed

lemma *NextDiamondImpDiamond*:
⊢ ○ (◇ *f*) ⟶ ◇ *f*

proof −
have *1*: ⊢ *skip*;(#*True*;*f*) = ((*skip*;#*True*);*f*) **by** (*rule ChopAssoc*)
hence *2*: ⊢ (*skip*;#*True*);*f* = *skip*;(#*True*;*f*) **by** *auto*
hence *3*: ⊢ (*skip*;#*True*);*f* = ○(◇*f*) **by** (*simp add*: *next-d-def sometimes-d-def*)
have *4*: ⊢ (*skip*;#*True*);*f* ⟶ ◇ *f* **by** (*rule ChopImpDiamond*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxImpNowAndWeakNext*:
⊢ □ *f* ⟶ (*f* ∧ *wnext* (□ *f*))
proof −
have *1*: ⊢ ¬ *f* ⟶ ◇ (¬ *f*) **by** (*rule NowImpDiamond*)
hence *2*: ⊢ ¬ (◇ (¬ *f*)) ⟶ *f* **by** *auto*
hence *3*: ⊢ □ *f* ⟶ *f* **by** (*metis always-d-def*)
have *4*: ⊢ ○ (◇ (¬ *f*)) ⟶ ◇ (¬ *f*) **by** (*rule NextDiamondImpDiamond*)
have *5*: ⊢ ¬ ¬ (◇ (¬ *f*)) ⟶ ◇(¬ *f*) **by** *auto*
hence *6*: ⊢ ○ (¬ ¬ (◇ (¬ *f*))) ⟶ ○ (◇(¬ *f*)) **by** (*rule NextImpNext*)
have *7*: ⊢ ○ (¬ ¬ (◇ (¬ *f*))) ⟶ ◇ (¬ *f*) **using** *4 6* **by** *auto*
hence *8*: ⊢ ○ (¬(□ *f*)) ⟶ ◇ (¬ *f*) **by** (*simp add*: *always-d-def*)
hence *9*: ⊢ ¬ (◇ (¬ *f*)) ⟶ ¬ (○ (¬(□ *f*))) **by** *auto*
hence *10*:⊢ □*f* ⟶ *wnext* (□ *f*) **by** (*simp add*: *always-d-def wnext-d-def*)
from *3 10* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxImpBoxRule*:
assumes ⊢ *f* ⟶ *g*
shows ⊢ □ *f*⟶ □ *g*
proof −
have *1*: ⊢ *f* ⟶ *g* **using** *assms* **by** *auto*
hence *2*: ⊢ ¬ *g* ⟶ ¬ *f* **by** *auto*
hence *3*: ⊢ □(¬ *g* ⟶ ¬ *f*) **by** (*rule BoxGen*)
have *4*: ⊢ □(¬ *g* ⟶ ¬ *f*) ⟶ (#*True*;(¬*g*)) ⟶ (#*True*;(¬*f*)) **by** (*rule BoxChopImpChop*)
have *5*: ⊢ (#*True*;(¬*g*)) ⟶ (#*True*;(¬*f*)) **using** *3 4 MP* **by** *blast*
hence *6*: ⊢ ◇ (¬*g*) ⟶ ◇(¬*f*) **by** (*simp add*: *sometimes-d-def*)
hence *7*: ⊢ ¬ (◇(¬*f*)) ⟶ ¬(◇ (¬*g*)) **by** *auto*
from *7* **show** *?thesis* **by** (*simp add*: *always-d-def*)
qed

lemma *BoxImpDist*:
⊢ □(*f* ⟶ *g*) ⟶ □ *f* ⟶ □ *g*
proof −
have *1*: ⊢ (*f* ⟶ *g*) ⟶ (¬ *g* ⟶ ¬ *f*) **by** *auto*
hence *2*: ⊢ □(*f* ⟶ *g*) ⟶ □(¬ *g* ⟶ ¬ *f*) **by** (*rule BoxImpBoxRule*)
have *3*: ⊢ □((¬ *g*) ⟶ ¬ *f*) ⟶ (#*True*; (¬ *g*)) ⟶ (#*True*; (¬ *f*)) **by** (*rule BoxChopImpChop*)
have *4*: ⊢ □(*f* ⟶ *g*) ⟶ (#*True*; (¬ *g*)) ⟶ (#*True*; (¬ *f*)) **using** *2 3 lift-imp-trans* **by** *blast*
hence *5*: ⊢ □(*f* ⟶ *g*) ⟶ ◇(¬ *g*) ⟶ ◇(¬ *f*) **by** (*simp add*: *sometimes-d-def*)
hence *6*: ⊢ □(*f* ⟶ *g*) ⟶ ¬(◇(¬ *f*)) ⟶ ¬(◇(¬ *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*: $\vdash f \longrightarrow \bigcirc\ f$ **using** *assms* **by** *auto*
hence *2*: $\vdash f \longrightarrow (more \wedge wnext\ f)$ **by** (*auto simp*: *more-defs wnext-defs next-defs*)
hence *3*: $\vdash f \longrightarrow wnext\ f$ **by** *auto*
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*: $\vdash f \longrightarrow more$ **using** *2* **by** *auto*
hence *11*: $\vdash \Box\ f \longrightarrow \Box\ more$ **by** (*rule ImpBoxRule*)
have *12*: $\vdash \neg(\Box\ more)$ **by** (*auto simp*: *always-defs more-defs*)
from *7 9 11 12* **show** *?thesis* **by** *fastforce*
qed

lemma *WnextEqvEmptyOrNext*:
$\vdash$ *wnext f* = (*empty* $\vee$ $\bigcirc$ *f*)
by (*auto simp*: *empty-defs wnext-defs next-defs*)

lemma *NotEmptyAndNext*:
$\vdash \neg$(*empty* $\wedge$ $\bigcirc$ *f*)
by (*auto simp*: *empty-defs next-defs*)

lemma *BoxEqvAndWnextBox*:
$\vdash \Box$ *f* = (*f* $\wedge$ *wnext* ($\Box$ *f*))
proof −
have *1*: $\vdash \Box$ *f* $\longrightarrow$ *f* $\wedge$ *wnext* ($\Box$ *f*)
using *BoxImpNowAndWeakNext* **by** *blast*
have *2*: $\vdash$ *f* $\wedge$ *wnext* ($\Box$ *f*) $\longrightarrow$ *f*
by *auto*
have *3*: $\vdash$ *more* $\wedge$ (*f* $\wedge$ *wnext* ($\Box$ *f*)) $\longrightarrow$ $\bigcirc$ (*f* $\wedge$ *wnext* ($\Box$ *f*))
using *1 NextImpNext WnextEqvEmptyOrNext empty-d-def int-iffD1*
by (*metis Prop01 Prop05 Prop08*)
have *4*: $\vdash$ *f* $\wedge$ *wnext* ($\Box$ *f*) $\longrightarrow$ $\Box$ *f*
using *2 3 BoxIntro* **by** *blast*
from *1 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxEqvAndEmptyOrNextBox*:
$\vdash \Box f$ = (*f* $\wedge$ (*empty* $\vee$ $\bigcirc$($\Box$ *f*)))
using *BoxEqvAndWnextBox WnextEqvEmptyOrNext* **by** (*metis int-eq*)

lemma *BoxEqvBoxBox*:
$\vdash \Box f = \Box$ ($\Box$ *f*)
using *BoxGen BoxInduct*
by (*metis BoxImpNowAndWeakNext MP int-iffI Prop09 Prop12*)

lemma *BoxBoxImpBox*:
$\vdash \Box(\Box h) \longrightarrow \Box$ *h*
by (*simp add*: *BoxElim*)

lemma *BoxImpBoxBox*:
$\vdash \Box$ *h* $\longrightarrow$ $\Box(\Box h)$
by (*auto simp*: *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*: $\vdash \Box\ (\neg\ g) = ((\neg\ g) \wedge \mathit{wnext}(\Box\ (\neg\ g)))$
using *BoxEqvAndWnextBox* **by** *metis*
have *7*: $\vdash \bigcirc\ f \wedge \Box\ (\neg\ g) \longrightarrow\ \bigcirc\ f \wedge \mathit{wnext}(\Box\ (\neg\ g))$
using *6* **by** *auto*
have *8*: $\vdash f \wedge (\Box\ (\neg\ g)) \longrightarrow \bigcirc\ f \wedge \mathit{wnext}(\Box\ (\neg\ g))$
using *5 7* **using** *lift-imp-trans* **by** *blast*
hence *9*: $\vdash f \wedge (\Box\ (\neg\ g)) \longrightarrow \mathit{more} \wedge \mathit{wnext}\ f \wedge \mathit{wnext}(\Box\ (\neg\ g))$
by (*auto simp*: *always-defs more-defs next-defs wnext-defs*)
hence *10*: $\vdash f \wedge (\Box\ (\neg\ g)) \longrightarrow \mathit{wnext}\ f \wedge \mathit{wnext}(\Box\ (\neg\ g))$
by *auto*
hence *11*: $\vdash f \wedge (\Box\ (\neg\ g)) \longrightarrow \mathit{wnext}\ (f \wedge \Box\ (\neg\ g))$
by (*auto simp*: *wnext-defs always-defs next-defs*)
hence *12*: $\vdash \Box(f \wedge (\Box\ (\neg\ g)) \longrightarrow \mathit{wnext}\ (f \wedge \Box\ (\neg\ g)))$
by (*rule BoxGen*)
have *13*: $\vdash \Box(f \wedge (\Box\ (\neg\ g)) \longrightarrow \mathit{wnext}\ (f \wedge \Box\ (\neg\ g))) \wedge f \wedge (\Box\ (\neg\ g)) \longrightarrow \Box(f \wedge (\Box\ (\neg\ g)))$
by (*rule BoxInduct*)
hence *14*: $\vdash \Box(f \wedge (\Box\ (\neg\ g)) \longrightarrow \mathit{wnext}\ (f \wedge \Box\ (\neg\ g))) \longrightarrow ((f \wedge (\Box\ (\neg\ g))) \longrightarrow \Box(f \wedge (\Box\ (\neg\ g))))$
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*: $\vdash (f \wedge (\Box\ (\neg\ g))) \longrightarrow \mathit{more}$
using *9* **by** *auto*
hence *19*: $\vdash \Box(f \wedge (\Box\ (\neg\ g))) \longrightarrow \Box\ \mathit{more}$
by (*rule ImpBoxRule*)
have *20*: $\vdash \neg(\Box\ \mathit{more})$
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*: $\vdash (\mathit{empty} \vee \bigcirc(f \longrightarrow g)) \wedge f \longrightarrow g$ **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** (*simp add*: *FinImpFin Prop11*)

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*: ⊢ *f* = *g* **using** *assms* **by** *auto*
hence *2*: ⊢ (*empty* = *f*) = (*empty* = *g*) **by** *auto*
hence *3*: ⊢ □(*empty* = *f*) = □ (*empty* = *g*) **by** (*rule BoxEqvBox*)
from *3* **show** *?thesis* **by** (*simp add*: *halt-d-def*)
qed

lemma *BiImpDiImpDi*:
⊢ *bi* (*f* ⟶ *g*) ⟶ *di f* ⟶ *di g*
proof −
have *1*: ⊢ *bi* (*f* ⟶ *g*) ⟶ (*f*; #*True*) ⟶ (*g*; #*True*) **by** (*rule BiChopImpChop*)
from *1* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiImpDi*:
assumes ⊢ *f* ⟶ *g*
shows ⊢ *di f* ⟶ *di g*
proof −
have *1*: ⊢ *f* ⟶ *g* **using** *assms* **by** *auto*
hence *2*: ⊢ *f*; #*True* ⟶ *g*; #*True* **by** (*rule LeftChopImpChop*)
from *2* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *BiImpBiRule*:
assumes ⊢ *f* ⟶ *g*
shows ⊢ *bi f* ⟶ *bi g*
proof −
have *1*: ⊢ *f* ⟶ *g* **using** *assms* **by** *auto*
hence *2*: ⊢ ¬ *g* ⟶ ¬ *f* **by** *auto*
hence *3*: ⊢ *di* (¬ *g*) ⟶ *di* (¬ *f*) **by** (*rule DiImpDi*)
hence *4*: ⊢ ¬ (*di* (¬ *f*)) ⟶ ¬ (*di* (¬ *g*)) **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *bi-d-def*)
qed

lemma *DiEqvDi*:
assumes ⊢ *f* = *g*
shows ⊢ *di f* = *di g*
proof −
have *1*: ⊢ *f* = *g* **using** *assms* **by** *auto*
hence *2*: ⊢ *f*; #*True* = *g*; #*True* **by** (*rule LeftChopEqvChop*)
from *2* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *BiEqvBi*:
assumes ⊢ *f* = *g*
shows ⊢ *bi f* = *bi g*
proof −
have *1*: ⊢ *f* = *g* **using** *assms* **by** *auto*
hence *2*: ⊢ (¬ *f*) = (¬ *g*) **by** *auto*
hence *3*: ⊢ *di* (¬ *f*) = *di* (¬ *g*) **by** (*rule DiEqvDi*)

hence *4*: $\vdash (\neg\ (di\ (\neg\ f))) = (\neg\ (\ di\ (\neg\ g)))$ **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *bi-d-def*)
qed

lemma *LeftChopChopImpChopRule*:
assumes $\vdash (f; g) \longrightarrow g$
shows $\vdash (f; g); h \longrightarrow (g; h)$
proof −
have *1*: $\vdash (f; g) \longrightarrow g$ **using** *assms* **by** *blast*
hence *2*: $\vdash (f; g); h \longrightarrow g; h$ **by** (*rule LeftChopImpChop*)
have *3*: $\vdash f; (g; h) = (f; g); h$ **by** (*rule ChopAssoc*)
from *2 3* **show** *?thesis* **by** *auto*
qed

lemma *AndChopCommute* :
$\vdash (f \wedge f1); g = (f1 \wedge f); g$
proof −
have *1*: $\vdash (f \wedge f1) = (f1 \wedge f)$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule LeftChopEqvChop*)
qed

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

lemma *StateAndChopImport*:
$\vdash (init\ w) \wedge (f; g) \longrightarrow ((init\ w) \wedge f); g$
proof −
have *1*: $\vdash (init\ w) \longrightarrow bi\ \ (init\ w)$ **by** (*rule StateImpBi*)
hence *2*: $\vdash (init\ w) \wedge (f; g) \longrightarrow bi\ (init\ w) \wedge (f; g)$ **by** *auto*
have *3*: $\vdash bi\ \ (init\ w) \wedge (f; g) \longrightarrow ((init\ w) \wedge f); g$ **by** (*rule BiAndChopImport*)
from *2 3* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

4.4 Further Properties Di and Bi

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

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

lemma *StateChop*:
$\vdash$ (*init w*); *f* $\longrightarrow$ (*init w*)
using *DiState* **by** (*auto simp*: *di-defs init-defs chop-defs*)

lemma *StateChopExportA*:
$\vdash$ ((*init w*) $\wedge$ *f*); *g* $\longrightarrow$ (*init w*)
using *DiState* **by** (*auto simp*: *init-defs chop-defs*)

lemma *StateAndChop*:
$\vdash$ ((*init w*) $\wedge$ *f*); *g* = ((*init w*) $\wedge$ (*f*; *g*))
by (*simp add*: *AndChopB StateAndChopImport StateChopExportA Prop11 Prop12*)

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

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

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

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

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

lemma *BiContraPosImpDist*:
$\vdash bi\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow (bi\ f) \longrightarrow (bi\ g)$
proof −
have *1*: $\vdash bi\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow (\ di\ (\neg\ g)) \longrightarrow (\ di\ (\neg\ f))$ **by** (*rule BiImpDiImpDi*)
hence *2*: $\vdash bi\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow (\neg\ (\ di\ (\neg\ f))) \longrightarrow (\neg\ (\ di\ (\neg\ g)))$ **by** *auto*
from *2* **show** *?thesis* **by** (*metis bi-d-def*)
qed

lemma *BiImpDist*:
$\vdash bi\ (f \longrightarrow g) \longrightarrow (bi\ f) \longrightarrow (bi\ g)$
proof −
have *1*: $\vdash (f \longrightarrow g) \longrightarrow (\neg\ g \longrightarrow \neg\ f)$ **by** *auto*
hence *2*: $\vdash \neg\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow \neg\ (f \longrightarrow g)$ **by** *auto*
hence *3*: $\vdash bi\ (\neg\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow \neg\ (f \longrightarrow g))$ **by** (*rule BiGen*)
have *4*: $\vdash bi\ (\neg\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow \neg\ (f \longrightarrow g))$
$\longrightarrow$
$bi\ (f \longrightarrow g) \longrightarrow bi\ (\neg\ g \longrightarrow \neg\ f)$ **by** (*rule BiContraPosImpDist*)
have *5*: $\vdash bi\ (f \longrightarrow g) \longrightarrow bi\ (\neg\ g \longrightarrow \neg\ f)$ **using** *3 4 MP* **by** *blast*
have *6*: $\vdash bi\ (\neg\ g \longrightarrow \neg\ f) \longrightarrow (bi\ f) \longrightarrow (bi\ g)$ **by** (*rule BiContraPosImpDist*)

from *5 6* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *IfChopEqvRule*:
assumes $\vdash f = \mathit{if}_i\ (\mathit{init}\ w)\ \mathit{then}\ f1\ \mathit{else}\ f2$
shows $\vdash f; g = \mathit{if}_i\ (\mathit{init}\ w)\ \mathit{then}\ (f1; g)\ \mathit{else}\ (f2; g)$
proof −
have *1*: $\vdash f = \mathit{if}_i\ (\mathit{init}\ w)\ \mathit{then}\ f1\ \mathit{else}\ f2$
using *assms* **by** *auto*
hence *2*: $\vdash f = (((\mathit{init}\ w) \wedge f1) \vee\ (\ (\mathit{init}\ (\neg\ w)) \wedge f2))$
by (*simp add*: *ifthenelse-d-def init-defs Valid-def*)
hence *3*: $\vdash f; g = (((\mathit{init}\ w) \wedge f1); g \vee\ (\ \ (\mathit{init}\ (\neg\ w)) \wedge f2); g)$
by (*rule OrChopEqvRule*)
have *4*: $\vdash ((\mathit{init}\ w) \wedge f1); g = ((\mathit{init}\ w) \wedge (f1; g))$
by (*rule StateAndChop*)
have *5*: $\vdash (\ \ (\mathit{init}\ (\neg\ w)) \wedge f2); g = \ \ ((\mathit{init}\ (\neg\ w)) \wedge (f2; g))$
by (*rule StateAndChop*)
have *6*: $\vdash f; g = (((\mathit{init}\ w) \wedge f1; g)\ \vee\ (\ \ (\mathit{init}\ (\neg\ w)) \wedge f2; g))$
using *3 4 5* **by** *fastforce*
from *6* **show** *?thesis* **by** (*simp add*: *ifthenelse-d-def init-defs Valid-def*)
qed

lemma *ChopOrEqvRule*:
assumes $\vdash g = (g1 \vee\ g2)$
shows $\vdash f; g = ((f; g1) \vee\ (f; g2))$
proof −
have *1*: $\vdash g = (g1 \vee\ g2)$ **using** *assms* **by** *auto*
hence *2*: $\vdash f; g = (f; (g1 \vee\ g2))$ **by** (*rule RightChopEqvChop*)
have *3*: $\vdash f; (g1 \vee\ g2) = (f; g1 \vee\ f; g2)$ **by** (*rule ChopOrEqv*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

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

lemma *EmptyOrNextChopEqv*:
$\vdash (\ \mathit{empty}\ \vee\ \bigcirc f); g = (g \vee\ \bigcirc(f; g))$
proof −
have *1*: $\vdash (\ \mathit{empty}\ \vee\ \bigcirc f); g = (g \vee\ ((\bigcirc f); g))$ **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 $\vdash\ \ f \longrightarrow\ \mathit{empty}\ \vee\ f1$

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*: $\vdash (\neg(f \wedge (\bigcirc\ (\Box\ f))\ )) = (\neg f \vee \neg(\ \bigcirc\ (\Box\ f)\ )\quad)$ **by** *auto*
from *2 3* **show** *?thesis* **by** *auto*
qed

lemma *BoxStateChopBoxEqvBox*:
$\vdash\quad \Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w) = \Box\ (\mathit{init}\ w)$
proof −
have *1*: $\vdash (\Box\ (\mathit{init}\ w)) = ((\mathit{init}\ w) \wedge (\ \mathit{empty}\ \vee\ \bigcirc(\Box\ (\mathit{init}\ w))))$
by (*rule BoxEqvAndEmptyOrNextBox*)
hence *2*: $\vdash (\Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w)) =$
$((\mathit{init}\ w) \wedge ((\ \mathit{empty}\ \vee\ \bigcirc(\Box\ (\mathit{init}\ w)));\ \Box\ (\mathit{init}\ w)))$
by (*metis StateAndChop inteq-reflection*)
have *3*: $\vdash ((\ \mathit{empty}\ \vee\ \bigcirc(\Box\ (\mathit{init}\ w)));\ \Box\ (\mathit{init}\ w)) =$
$(\Box\ (\mathit{init}\ w) \vee\ \bigcirc(\Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w)))$
by (*rule EmptyOrNextChopEqv*)
have *4*: $\vdash (\Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w)) =$
$((\mathit{init}\ w) \wedge (\Box\ (\mathit{init}\ w) \vee\ \bigcirc(\Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w))))$
using *2 3* **by** *fastforce*
have *5*: $\vdash \neg\ (\Box\ (\mathit{init}\ w)) \longrightarrow \neg\ \ (\mathit{init}\ w) \vee\ \neg(\ \bigcirc(\Box\ (\mathit{init}\ w)))$
by (*rule NotBoxImpNotOrNotNextBox*)
have *6*: $\vdash (\Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w)) \wedge \neg(\ \Box\ (\mathit{init}\ w)) \longrightarrow$
$\bigcirc(\Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w)) \wedge \neg(\ \bigcirc(\Box\ (\mathit{init}\ w)))$
using *4 5* **by** *fastforce*
hence *7*: $\vdash \Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w) \longrightarrow \Box\ (\mathit{init}\ w)$
by (*rule NextContra*)
have *11*: $\vdash \Box\ (\mathit{init}\ w) = ((\mathit{init}\ w) \wedge \Box\ (\mathit{init}\ w))$
by (*rule BoxEqvAndBox*)
have *12*: $\vdash \mathit{empty}\ ;\ \Box\ (\mathit{init}\ w) = \Box\ (\mathit{init}\ w)$
by (*rule EmptyChop*)
have *13*: $\vdash ((\mathit{init}\ w) \wedge\ \mathit{empty}\ );\ \Box\ (\mathit{init}\ w) = ((\mathit{init}\ w) \wedge (\ \mathit{empty}\ ;\ \Box\ (\mathit{init}\ w)))$
by (*rule StateAndChop*)
have *14*: $\vdash \Box\ (\mathit{init}\ w) = ((\mathit{init}\ w) \wedge\ \mathit{empty}\ );\ \Box\ (\mathit{init}\ w)$
using *11 12 13* **by** *fastforce*
have *15*: $\vdash (\mathit{init}\ w) \wedge\ \mathit{empty}\ \longrightarrow \Box\ (\mathit{init}\ w)$
by (*rule StateAndEmptyImpBoxState*)
hence *16*: $\vdash ((\mathit{init}\ w) \wedge\ \mathit{empty}\ );\ \Box\ (\mathit{init}\ w) \longrightarrow \Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w)$
by (*rule LeftChopImpChop*)
have *17*: $\vdash \Box\ (\mathit{init}\ w) \longrightarrow \Box\ (\mathit{init}\ w);\ \Box\ (\mathit{init}\ w)$

using *14 16* **by** *fastforce*
from *7 17* **show** *?thesis* **by** *fastforce*
qed

lemma *NotBoxStateImpBoxYieldsNotBox*:
$\vdash \neg(\ \Box\ (\mathit{init}\ w)) \longrightarrow (\Box\ (\mathit{init}\ w))\ \mathit{yields}\ (\neg(\ \Box\ (\mathit{init}\ w)))$
proof −
have *1*: $\vdash \Box\ (\mathit{init}\ w); \Box\ (\mathit{init}\ w) = \Box\ (\mathit{init}\ w)$ **by** (*rule BoxStateChopBoxEqvBox*)
have *2*: $\vdash \Box\ (\mathit{init}\ w) = (\neg\ \neg(\ \Box\ (\mathit{init}\ w)))$ **by** *auto*
hence *3*: $\vdash \Box\ (\mathit{init}\ w); \Box\ (\mathit{init}\ w) = \Box\ (\mathit{init}\ w); (\neg\ \neg(\ \Box\ (\mathit{init}\ w)))$ **by** (*rule RightChopEqvChop*)
have *4*: $\vdash \neg(\ \Box\ (\mathit{init}\ w)) \longrightarrow \neg\ (\Box\ (\mathit{init}\ w); (\neg\ \neg\ (\Box\ (\mathit{init}\ w))))$ **using** *1 3* **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *StateEqvBi*:
$\vdash\ (\mathit{init}\ w) = \mathit{bi}\ \ (\mathit{init}\ w)$
proof −
have *1*: $\vdash (\mathit{init}\ w) \longrightarrow \mathit{bi}\ \ (\mathit{init}\ w)$ **by** (*rule StateImpBi*)
have *2*: $\vdash \mathit{bi}\ \ (\mathit{init}\ w) \longrightarrow (\mathit{init}\ w)$ **by** (*rule BiElim*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *TrueChopEqvDiamond*:
$\vdash\ \#\mathit{True}; f = \Diamond\ f$
by (*simp add*: *sometimes-d-def*)

4.5 Properties of Da and Ba

lemma *DaEqvDtDi*:
$\vdash\ \ \mathit{da}\ \ f = \Diamond\ (\mathit{di}\ \ f)$
proof −
have *1*: $\vdash \#\mathit{True}; (f; \#\mathit{True}) = \#\mathit{True}; (f; \#\mathit{True})$ **by** *auto*
hence *2*: $\vdash \#\mathit{True}; (f; \#\mathit{True}) = \#\mathit{True};\ \ \mathit{di}\ \ f$ **by** (*simp add*: *di-d-def*)
have *3*: $\vdash \#\mathit{True};\ \ \mathit{di}\ \ f = \Diamond(\ \mathit{di}\ \ f)$ **by** (*rule TrueChopEqvDiamond*)
have *4*: $\vdash \#\mathit{True}; (f; \#\mathit{True}) = \Diamond(\ \mathit{di}\ \ f)$ **using** *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*:*da-d-def*)
qed

lemma *DaEqvDiDt*:
$\vdash\ \ \mathit{da}\ \ f =\ \ \mathit{di}\ (\Diamond\ f)$
proof −
have *1*: $\vdash \#\mathit{True}; f = \Diamond\ f$ **by** (*rule TrueChopEqvDiamond*)
hence *2*: $\vdash (\#\mathit{True}; f); \#\mathit{True} = (\Diamond\ f); \#\mathit{True}$ **by** (*rule LeftChopEqvChop*)
hence *3*: $\vdash (\#\mathit{True}; f); \#\mathit{True} =\ \ \mathit{di}(\ \Diamond\ f)$ **by** (*simp add*: *di-d-def*)
have *4*: $\vdash \#\mathit{True}; (f; \#\mathit{True}) = (\#\mathit{True}; f); \#\mathit{True}$ **by** (*rule ChopAssoc*)
have *5*: $\vdash \#\mathit{True}; (f; \#\mathit{True}) =\ \ \mathit{di}\ (\Diamond\ f)$ **using** *3 4* **by** *fastforce*
from *5* **show** *?thesis* **by** (*simp add*: *da-d-def*)
qed

lemma *DtDiEqvDiDt*:
$\vdash \quad \Diamond\ (di\ \ f) =\ \ di\ (\Diamond\ f)$
by (*metis ChopAssoc di-d-def sometimes-d-def*)

lemma *DiamondNotEqvNotBox*:
$\vdash \Diamond\ (\neg\ \ f) = (\neg\ (\Box\ f))$
by (*simp add*: *always-d-def*)

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

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

lemma *NotDiamondNotEqvBox*:
$\vdash (\neg\ (\Diamond(\neg\ f))) = \Box\ f$
by (*simp add*: *always-d-def*)

lemma *BaEqvBtBi*:
$\vdash \quad ba\ \ f = \Box\ (bi\ \ f)$
proof −
have *1*: $\vdash\ \ da\ (\neg\ \ f) = \Diamond\ (di\ (\neg\ \ f))$ **by** (*rule DaEqvDtDi*)
have *2*: $\vdash\ \ di\ (\neg\ \ f) = (\neg\ (bi\ \ f))$ **by** (*rule DiNotEqvNotBi*)
hence *3*: $\vdash \Diamond\ (di\ (\neg\ \ f)) = \Diamond(\neg\ (bi\ \ f))$ **by** (*rule DiamondEqvDiamond*)
have *4*: $\vdash (\neg\ (\Diamond(\neg\ (bi\ \ f)))) = \Box(bi\ \ f)$ **by** (*rule NotDiamondNotEqvBox*)
have *5*: $\vdash (\neg\ (\ da\ (\neg\ \ f))) = \Box(bi\ \ f)$ **using** *1 2 3 4* **by** *fastforce*
from *5* **show** *?thesis* **by** (*simp add*: *ba-d-def*)
qed

lemma *BtBiEqvBiBt*:
$\vdash \quad \Box\ (bi\ \ f) = bi(\ \Box\ f)$
proof −
have *1*: $\vdash \quad ba\ \ f = \Box\ (bi\ \ f)$ **by** (*rule BaEqvBtBi*)
have *2*: $\vdash \quad ba\ \ f = bi(\ \Box\ f)$ **by** (*rule BaEqvBiBt*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxStateEqvBaBoxState*:
$\vdash \quad \Box\ (init\ w) = \ ba\ (\Box\ (init\ w))$
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*:
$\vdash \quad ba\ \ f \longrightarrow bi\ \ f$
proof −
have *1*: $\vdash \ ba\ \ f = \Box(bi\ \ f)$ **by** (*rule BaEqvBtBi*)
have *2*: $\vdash \Box(bi\ \ f) \longrightarrow bi\ \ f$ **by** (*rule BoxElim*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *fastforce*
qed

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

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

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

lemma *BoxAndChopImport*:
$\vdash \quad \Box\ h \wedge f; g \longrightarrow f; (h \wedge g)$
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*:
$\vdash \quad ba\ \ f \wedge (g; g1) \longrightarrow (f \wedge g); (f \wedge g1)$
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*:
$\vdash \quad f; (g \wedge g1) = f; (g1 \wedge g)$
proof −
have *1*: $\vdash (g \wedge g1) = (g1 \wedge g)$ **by** *auto*
from *1* **show** *?thesis* **by** (*rule RightChopEqvChop*)
qed

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

lemma *ChopAndB*:
$\vdash \quad f; (g \wedge g1) \longrightarrow f; g1$
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*:
$\vdash \quad di\ \ f = (\neg(\ bi\ (\neg\ \ f)))$
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*:
$\vdash \quad f;\ g \wedge \Box\ h \longrightarrow f;\ (g \wedge h)$
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*:
$\vdash \quad (f \wedge g);\ (f1 \wedge g1) = (g \wedge f);\ (g1 \wedge f1)$
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 $\vdash \quad f \longrightarrow f1\ \vdash g \longrightarrow g1$
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 $\vdash \quad f = f1\ \vdash g = g1$
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*:
$\vdash \quad \Box\ h \longrightarrow f;\ g \longrightarrow f;\ (\Box\ h \wedge g)$
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*:
$\vdash \quad (\neg\ (f;\ g)) = f\ yields\ \ (\neg\ \ g)$
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*:
$\vdash \quad \neg\ (\ di\ \ \#False)$
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*:
$\vdash \quad ((init\ w) \wedge\ \ empty\ );\ f = ((init\ w) \wedge f)$
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*:
$\vdash \quad ((init\ w) \wedge \bigcirc\ f);\ g = ((init\ w) \wedge \bigcirc(f;\ g))$
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*:
$\vdash \quad \bigcirc(((init\ w) \wedge f);\ g) = (\bigcirc\ (init\ w) \wedge \bigcirc(f;\ g))$
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*:
$\vdash \quad ((init\ w) \longrightarrow (f\ yields\ \ g)) = ((init\ w) \wedge f)\ yields\ \ g$
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*:
$\vdash \quad ((init\ w) \wedge\ \ di\ \ f) =\ \ di\ ((init\ w) \wedge f)$
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*: ⊢ (*init w*) ⟶ *f* **using** *assms* **by** *auto*
hence *2*: ⊢ ¬ *f* ⟶ ¬ (*init w*) **by** *auto*
hence *3*: ⊢ *di* (¬ *f*) ⟶ *di* (¬ (*init w*)) **by** (*rule DiImpDi*)
hence *4*: ⊢ *di* (¬ *f*) ⟶ *di* (*init* (¬*w*)) **by** (*metis Initprop*(*2*) *inteq-reflection*)
have *5*: ⊢ *di* (*init* (¬ *w*)) = (*init* (¬ *w*)) **by** (*rule DiState*)
have *6*: ⊢ *di* (¬ *f*) ⟶ ¬ (*init w*) **using** *4 5* **using** *Initprop*(*2*) **by** *fastforce*
hence *7*: ⊢ (*init w*) ⟶ ¬ (*di* (¬ *f*)) **by** *auto*
from *7* **show** *?thesis* **by** (*simp add*: *bi-d-def*)
qed

lemma *ChopAndNotChopImp*:
⊢ *f*; *g* ∧ ¬ (*f*; *g1*) ⟶ *f*; (*g* ∧ ¬ *g1*)
proof −
have *1*: ⊢ *g* ⟶ (*g*∧ ¬ *g1*) ∨ *g1* **by** *auto*
hence *2*: ⊢ *f*; *g* ⟶ *f*; ((*g*∧ ¬ *g1*) ∨ *g1*) **by** (*rule RightChopImpChop*)
have *3*: ⊢ *f*; ((*g*∧ ¬ *g1*) ∨ *g1*) ⟶ (*f*; (*g*∧ ¬ *g1*)) ∨ (*f*; *g1*) **by** (*rule ChopOrImp*)
have *4*: ⊢ *f*; *g* ⟶ *f*; (*g*∧ ¬ *g1*) ∨ *f*; *g1* **using** *2 3 MP* **by** *fastforce*
from *4* **show** *?thesis* **by** *auto*
qed

lemma *ChopAndYieldsImp*:
⊢ *f*; *g* ∧ *f yields g1* ⟶ *f*; (*g* ∧ *g1*)
proof −
have *1*: ⊢ *g* ⟶ (*g*∧ *g1*) ∨ ¬ *g1* **by** *auto*
hence *2*: ⊢ *f*; *g* ⟶ *f*; ((*g*∧ *g1*) ∨ ¬ *g1*) **by** (*rule RightChopImpChop*)
have *3*: ⊢ *f*; ((*g*∧ *g1*) ∨ ¬ *g1*) ⟶ (*f*; (*g*∧ *g1*)) ∨ (*f*; (¬ *g1*)) **by** (*rule ChopOrImp*)
have *4*: ⊢ *f*; *g* ⟶ *f*; (*g*∧ *g1*) ∨ *f*; (¬ *g1*) **using** *2 3 MP* **by** *fastforce*
hence *5*: ⊢ *f*; *g* ∧ ¬ (*f*; (¬ *g1*)) ⟶ *f*; (*g* ∧ *g1*) **by** *auto*
from *5* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

lemma *ChopAndYieldsMP*:
⊢ *f*; *g* ∧ *f yields* (*g*⟶ *g1*) ⟶ *f*; *g1*
proof −
have *1*: ⊢ *f*; *g* ∧ *f yields* (*g*⟶ *g1*) ⟶ *f*; (*g* ∧ (*g* ⟶ *g1*)) **by** (*rule ChopAndYieldsImp*)
have *2*: ⊢ *g* ∧ (*g* ⟶ *g1*) ⟶ *g1* **by** *auto*
hence *3*: ⊢ *f*; (*g* ∧ (*g* ⟶ *g1*)) ⟶ *f*; *g1* **by** (*rule RightChopImpChop*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *OrYieldsImp*:
⊢ (*f* ∨ *f1*) *yields g* = ((*f yields g*) ∧ (*f1 yields g*))
proof −
have *1*: ⊢ ((*f*∨ *f1*); (¬ *g*)) = ((*f*; (¬ *g*)) ∨ (*f1*; (¬ *g*))) **by** (*rule OrChopEqv*)
hence *2*: ⊢ (¬ ((*f*∨ *f1*); (¬ *g*))) = (¬ (*f*; (¬ *g*)) ∧ ¬(*f1*; (¬ *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

4.6 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*: $\vdash \neg(\Diamond(\neg(empty \longrightarrow f \ ))) = (\neg(\Diamond(\neg \ f \wedge empty)))$
by *auto*
have *6*: $\vdash (\neg(\Diamond(\neg \ f \wedge empty))) = \#True;(f \wedge empty)$
using *Finprop*(*4*) *sometimes-d-def* **by** (*metis int-eq int-simps*(*4*))
from *1 2 5 6* **show** *?thesis* **by** *fastforce*
qed

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

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

lemma *FinImpBox*:

$\vdash$ *fin w* $\longrightarrow$ $\Box$(*fin w*)
by (*metis BoxImpBoxBox fin-d-def*)

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

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

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

lemma *FinAndEmpty*:
$\vdash$ ((*fin w*) $\wedge$ *empty*) = (*w* $\wedge$ *empty*)
proof −
have *1*: $\vdash$ ((*fin w*) $\wedge$ *empty*) = (#*True*;(*w* $\wedge$ *empty*) $\wedge$ *empty*)
using *FinEqvTrueChopAndEmpty* **by** *fastforce*
have *2*: $\vdash$ (#*True*;(*w* $\wedge$ *empty*) $\wedge$ *empty*) = ((#*True* $\wedge$ *empty*);(*w* $\wedge$ *empty*))
using *ChopAndEmptyEqvEmptyChopEmpty*
by (*smt int-eq int-iffD2 lift-and-com Prop10 Prop12*)
have *3*: $\vdash$ (#*True* $\wedge$ *empty*);(*w* $\wedge$ *empty*) = (*empty*;(*w* $\wedge$ *empty*))
using *LeftChopEqvChop* **by** *fastforce*
have *4*: $\vdash$ (*empty*;(*w* $\wedge$ *empty*)) = (*w* $\wedge$ *empty*)
using *EmptyChop* **by** *blast*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *AndFinEqvChopAndEmpty*:
$\vdash$ (*f* $\wedge$ *fin g*) = *f*; (*g* $\wedge$ *empty*)
proof −
have *1*: $\vdash$ (*f* $\wedge$ *fin g*) = (*f*;*empty* $\wedge$ *fin g*)
using *ChopEmpty* **by** (*metis int-eq*)
have *2*: $\vdash$ (*fin g* $\wedge$ *f*;*empty*) = (*f*;(*empty* $\wedge$ *fin g*))
using *FinAndChop* **by** *fastforce*
have *3*: $\vdash$ (*empty* $\wedge$ *fin g*) = (*fin g* $\wedge$ *empty*)
by *auto*
have *4*: $\vdash$ (*fin g* $\wedge$ *empty*) = (*g* $\wedge$ *empty*)
using *FinAndEmpty* **by** *metis*
have *5*: $\vdash$ (*empty* $\wedge$ *fin g*) = (*g* $\wedge$ *empty*)
using *3 4* **by** *auto*
hence *6*: $\vdash$ *f*;(*empty* $\wedge$ *fin g*) = *f*;(*g* $\wedge$ *empty*)
using *RightChopEqvChop* **by** *blast*

from *1 2 5* **show** *?thesis* **by** (*metis inteq-reflection lift-and-com*)
qed

lemma *AndFinEqvChopStateAndEmpty*:
$\vdash \ (f \wedge \ fin \ (init\ w)) = f;((init\ w) \wedge \ empty\ )$
using *AndFinEqvChopAndEmpty* **by** *blast*

lemma *FinStateEqvStateAndEmptyOrNextFinState*:
$\vdash \ fin \ (init\ w) = (((init\ w) \wedge \ empty\ ) \vee \ \bigcirc(\ fin \ (init\ w)))$
proof −
have *1*: $\vdash fin\ (init\ w) = \Box(\ empty \longrightarrow init\ w)$
by (*simp add*: *fin-d-def*)
have *2* : $\vdash \Box(empty \longrightarrow init\ w) =$
$((empty \longrightarrow init\ w) \wedge wnext\ (\Box\ (empty \longrightarrow init\ w)))$
by (*rule BoxEqvAndWnextBox*)
have *3*: $\vdash fin\ (init\ w) = \ ((empty \longrightarrow init\ w) \wedge wnext\ (fin\ (init\ w)))$
using *1 2* **by** (*simp add*: *fin-d-def*)
have *4*: $\vdash wnext\ (fin\ (init\ w)) = (empty \vee \bigcirc\ (fin\ (init\ w)))$
by (*rule WnextEqvEmptyOrNext*)
have *5*: $\vdash fin\ (init\ w) = \ ((empty \longrightarrow init\ w) \wedge (empty \vee \bigcirc\ (fin\ (init\ w))))$
using *3 4* **by** *fastforce*
have *6*: $\vdash ((empty \longrightarrow init\ w) \wedge (empty \vee \bigcirc\ (fin\ (init\ w)))) =$
$(((empty \longrightarrow init\ w) \wedge empty) \vee ((empty \longrightarrow init\ w) \wedge \bigcirc\ (fin\ (init\ w))))$
by *auto*
have *7*: $\vdash ((empty \longrightarrow init\ w) \wedge empty) = ((init\ w) \wedge empty)$
by *auto*
have *8*: $\vdash ((empty \longrightarrow init\ w) \wedge \bigcirc\ (fin\ (init\ w))) = \ \bigcirc\ (fin\ (init\ w))$
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))
proof −
have *1*: $\vdash$ (¬($\Diamond$ (f $\wedge$ ¬ f))) = $\Box$(¬(f $\wedge$ ¬f)) **using** *NotDiamondNotEqvBox* **by** *fastforce*
have *2*: $\vdash$ ¬(f $\wedge$ ¬f) **by** *simp*
have *3*: $\vdash$ $\Box$(¬(f $\wedge$ ¬f)) **using** *2* **by** (*simp add*: *BoxGen*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *FinYields*:
$\vdash$ (*fin* (*init w*)) *yields* (*init w*)
proof −
have *1*: $\vdash$ (*fin* (*init w*)); (¬(*init w*)) = $\Diamond$((*init w*) $\wedge$ ¬(*init w*)) **by** (*rule FinChopEqvDiamond*)
have *2*: $\vdash$ ¬($\Diamond$((*init w*) $\wedge$ ¬ (*init w*))) **by** (*rule NotDiamondAndNot*)
have *3*: $\vdash$ ¬ ((*fin* (*init w*)); (¬ (*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* (¬ (*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-simps*(*14*) *inteq-reflection*)
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*: ⊢ (*f* ∧ *fin*(*init w*)); #*True* = *f*;((*init w*) ∧ #*True*) **by** (*rule AndFinChopEqvStateAndChop*)
have *2*: ⊢ ((*init w*) ∧ #*True*) = (*init w*) **by** *auto*
hence *3*: ⊢ (*f*; ((*init w*) ∧ #*True*)) = (*f*; (*init w*)) **by** (*rule RightChopEqvChop*)
have *4*: ⊢ (*f* ∧ *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*:
⊢ *fin* (*init* (¬ *w*)) = (¬(*fin* (*init w*)))
using *FinEqvTrueChopAndEmpty*
by (*metis* (*no-types*, *hide-lams*) *Finprop*(*4*) *Initprop*(*2*) *int-eq int-simps*(*4*) *int-simps*(*7*) *sometimes-d-def*)

lemma *BiImpFinEqvYieldsState*:
⊢ *bi* (*f* ⟶ *fin* (*init w*)) = *f yields* (*init w*)
proof −
have *1*: ⊢ *di* (*f* ∧ *fin* (*init* (¬ *w*))) = *f*; (*init* (¬ *w*))
by (*rule DiAndFinEqvChopState*)
have *2*: ⊢ (*f* ∧ *fin*(*init* (¬ *w*))) = (*f* ∧ ¬(*fin*(*init w*)))
using *FinNotStateEqvNotFinState* **by** *fastforce*
have *3*: ⊢ (*f* ∧ ¬ (*fin*(*init w*))) = (¬ (*f* ⟶ *fin* (*init w*)))
by *auto*
have *4*: ⊢ (*f* ∧ *fin*(*init* (¬ *w*))) = (¬ (*f* ⟶ *fin*(*init w*)))
using *2 3* **by** *fastforce*
hence *5*: ⊢ *di* (*f* ∧ *fin* (*init* (¬ *w*))) = *di* (¬ (*f* ⟶ *fin*(*init w*)))
by (*rule DiEqvDi*)
have *6*: ⊢ *di* (¬ (*f* ⟶ *fin* (*init w*))) = (¬(*bi* (*f* ⟶ *fin*(*init w*))))
by (*rule DiNotEqvNotBi*)
have *7*: ⊢ ¬ (*bi* (*f* ⟶ *fin* (*init w*))) = *f*;(*init* (¬ *w*))
using *1 5 6 Initprop* **by** *fastforce*
hence *8*: ⊢ *bi* (*f* ⟶ *fin* (*init w*)) = (¬ (*f*; (¬ (*init w*))))
by (*metis Initprop*(*2*) *int-eq int-simps*(*7*))
from *8* **show** *?thesis* **by** (*simp add*: *yields-d-def*)
qed

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

lemma *StateAndYieldsImpYields*:
assumes ⊢ (*init w*) ∧ *f* ⟶ *f1*
shows ⊢ (*init w*) ∧ (*f1 yields g*) ⟶ (*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*:
$\vdash \ \ (f\ yields\ \ g) \wedge (f1\ yields\ \ g1) \longrightarrow (f \wedge f1)\ yields\ (g \wedge g1)$
proof −
have *1*: $\vdash f\ yields\ \ g \longrightarrow (f \wedge f1)\ yields\ \ g$
by (*rule AndYieldsA*)
have *2*: $\vdash f1\ yields\ \ g1 \longrightarrow (f \wedge f1)\ yields\ \ g1$
by (*rule AndYieldsB*)
have *3*: $\vdash ((f \wedge f1)\ yields\ \ g \wedge (f \wedge f1)\ yields\ \ g1) = (f \wedge f1)\ yields\ \ (g \wedge g1)$
by (*rule YieldsAndYieldsEqvYieldsAnd*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

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

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

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

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

lemma *SkipYieldsEqvWeakNext*:
$\vdash\ \ skip\ \ yields\ \ f = wnext\ \ f$
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*: ⊢ *skip* ; #*True* = ○#*True* **by** (*rule SkipChopEqvNext*)
hence *2*: ⊢ ○#*True* = *skip* ; #*True* **by** *auto*
from *2* **show** *?thesis* **by** (*simp add*: *more-d-def*)
qed

lemma *MoreChopImpMore*:
⊢ *more* ; *f* ⟶ *more*
proof −
have *1*: ⊢ (○#*True*); *f* = ○(#*True*; *f*) **by** (*rule NextChop*)
have *2*: ⊢ ○(#*True*; *f*) ⟶ *more* **by** (*auto simp*: *more-defs next-defs*)
have *3*: ⊢ (○#*True*; *f*) ⟶ *more* **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*metis more-d-def*)
qed

lemma *ChopMoreImpMore*:
⊢ *f*; *more* ⟶ *more*
proof −
have *1*: ⊢ *f*; *more* ⟶ ◇ *more* **by** (*rule ChopImpDiamond*)
have *2*: ⊢ ◇ *more* ⟶ *more* **by** (*auto simp*: *more-defs sometimes-defs*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *MoreChopEqvNextDiamond*:
⊢ *more* ; *f* = ○(◇ *f*)
proof −
have *1*: ⊢ *more* ; *f* = (○ #*True*); *f* **by** (*simp add*: *more-d-def*)
have *2*: ⊢ (○#*True*); *f* = ○(#*True*; *f*) **by** (*rule NextChop*)
have *3*: ⊢ *more* ; *f* = ○(#*True*; *f*) **using** *1 2* **by** *fastforce*
from *3* **show** *?thesis* **by** (*simp add*: *sometimes-d-def*)
qed

lemma *WeakNextBoxImpMoreYields*:
⊢ *more yields f* = *wnext*(□ *f*)
proof −
have *1*: ⊢ *more* ; (¬ *f*) = ○(◇ (¬*f*)) **by** (*rule MoreChopEqvNextDiamond*)
have *2*: ⊢ ○(◇ (¬*f*)) = ○(¬(□*f*)) **by** (*auto simp*: *always-d-def*)
have *3*: ⊢ ○(¬(□*f*)) = (¬ (*wnext*(□ *f*))) **by** (*auto simp*: *wnext-d-def*)
have *4*: ⊢ *more* ; (¬ *f*) = (¬(*more yields f*)) **by** (*simp add*: *yields-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *NotEqvYieldsMore*:
⊢ (¬ *f*) = *f yields more*
proof −
have *1*: ⊢ *f*; *empty* = *f* **by** (*rule ChopEmpty*)
hence *2*: ⊢ (¬ (*f*; *empty*)) = (¬ *f*) **by** *auto*
have *3*: ⊢ *empty* = (¬ *more*) **by** (*auto simp*: *empty-d-def*)
hence *4*: ⊢ *f*; *empty* = *f*; (¬ *more*) **by** (*rule RightChopEqvChop*)
hence *5*: ⊢ (¬ (*f*; *empty*)) = (¬ (*f*; (¬ *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*:
$\vdash \quad di\ \ f = \ di\ (\ \ di\ \ f)$
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*:
$\vdash \quad bi\ \ f = bi(\ bi\ \ f)$
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*:
$\vdash \quad di\ (f \vee\ \ g) = \ (di\ \ f \vee\ \ \ di\ \ g)$
proof −
have *1*: $\vdash (f \vee\ \ g); \#True = (f; \#True \vee\ \ g; \#True)$ **by** (*rule OrChopEqv*)
from *1* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiAndA*:
$\vdash \quad di\ (f \wedge g) \longrightarrow\ \ di\ \ f$
proof −
have *1*: $\vdash (f \wedge g); \#True \longrightarrow f; \#True$ **by** (*rule AndChopA*)
from *1* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiAndB*:
$\vdash \quad di\ (f \wedge g) \longrightarrow\ \ di\ \ g$
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*:
$\vdash \quad di\ (f \wedge g) \longrightarrow\ \ di\ \ f \wedge\ \ di\ \ g$
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* = ○#*True* **by** (*rule SkipChopEqvNext*)
have *2*: $\vdash$ ○#*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* (○ #*True*) = ○(*di* #*True*) **by** (*rule DiNext*)
have *2*: $\vdash$ ○(*di* #*True*) ⟶ *more* **by** (*auto simp*: *next-defs di-defs more-defs*)
have *3*: $\vdash$ *di*(○ #*True*) ⟶ *more* **using** *1 2* **by** *fastforce*
hence *4*: $\vdash$ *di more* ⟶ *more* **by** (*simp add*: *more-d-def*)
have *5*: $\vdash$ *more* ⟶ *di more* **by** (*rule ImpDi*)
from *4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *DiIfEqvRule*:
assumes $\vdash$ *f* = *if*$_i$ (*init w*) *then g else h*
shows $\vdash$ *di f* = *if*$_i$ (*init w*) *then* (*di g*) *else* (*di h*)
proof −
have *1*: $\vdash$ *f* = *if*$_i$ (*init w*) *then g else h* **using** *assms* **by** *auto*
hence *2*: $\vdash$ *f*; #*True* = *if*$_i$ (*init w*) *then* (*g*; #*True*) *else* (*h*; #*True*) **by** (*rule IfChopEqvRule*)
from *2* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

lemma *DiEmpty*:
$\vdash$ *di empty*
proof −
have *1*: $\vdash$ #*True* **by** *auto*
have *2*: $\vdash$ *empty* ; #*True* = #*True* **by** (*rule EmptyChop*)
have *3*: $\vdash$ *empty* ; #*True* **using** *1 2* **by** *auto*
from *3* **show** *?thesis* **by** (*simp add*: *di-d-def*)
qed

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

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

lemma *DaEqvNotBaNot*:
$\vdash \quad da\ \ f = (\neg\ (\ ba\ (\neg\ \ f)))$
proof −
have *1*: $\vdash\ \ ba\ (\neg\ \ f) = (\neg\ (\ da\ (\neg\ \neg\ \ f)))$ **by** (*simp add*: *ba-d-def*)
hence *2*: $\vdash\ \ da\ (\neg\ \neg\ \ f) = (\neg(\ \ ba\ (\neg\ \ f)))$ **by** *fastforce*
have *3*: $\vdash f = (\neg\ \neg\ \ f)$ **by** *simp*
hence *4*: $\vdash\ \ da\ \ f = \ \ da\ \ (\neg\ \neg\ \ f)$ **by** (*rule DaEqvDa*)
from *2 4* **show** *?thesis* **by** *simp*
qed

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

lemma *DaIntro*:
$\vdash \quad f \longrightarrow\ \ da\ \ f$
proof −
have *1*: $\vdash\ \ ba\ \ (\neg\ \ f) \longrightarrow (\neg\ \ f)$ **by** (*rule BaElim*)
hence *2*: $\vdash \neg\ \neg\ \ f \longrightarrow \neg\ (\ \ ba\ \ (\neg\ \ f))$ **by** *fastforce*
have *3*: $\vdash f = (\neg\ \neg\ \ f)$ **by** *simp*
have *4*: $\vdash\ \ da\ \ f = (\neg\ \ (\ ba\ \ (\neg\ \ f)))$ **by** (*rule DaEqvNotBaNot*)
from *2 3 4* **show** *?thesis* **by** *fastforce*
qed

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

lemma *BaImpDist*:
$\vdash\ \ ba\ (f \longrightarrow g) \longrightarrow\ \ ba\ \ f \longrightarrow\ \ ba\ \ g$
proof −
have *1*: $\vdash\ bi\ (f \longrightarrow g) \longrightarrow (bi\ \ f \longrightarrow bi\ \ g)$ **by** (*rule BiImpDist*)
hence *2*: $\vdash \Box(bi\ (f \longrightarrow g) \longrightarrow (bi\ \ f \longrightarrow bi\ \ g))$ **by** (*rule BoxGen*)
have *3*: $\vdash \Box(bi\ (f \longrightarrow g) \longrightarrow (bi\ \ f \longrightarrow bi\ \ g))$

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

lemma *BaAndEqv*:
$\vdash\ \ \ ba\ (f \wedge g) = \ (ba\ \ f \wedge\ \ ba\ \ g)$
proof −
have *1*: $\vdash\ \ \ ba\ (f \wedge g) = \ \ \Box(bi\ (f \wedge g))$
by (*rule BaEqvBtBi*)
have *2*: $\vdash\ \ bi\ (f \wedge g) = (bi\ f \wedge bi\ g)$
by (*auto simp*: *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*) $\longrightarrow$ $\Box$(*g* $\longrightarrow$ *g1*) **by** (*rule BaImpBt*)
have *2*: $\vdash$ $\Box$(*g* $\longrightarrow$ *g1*) $\longrightarrow$ *f*; *g* $\longrightarrow$ *f*; *g1* **by** (*rule BoxChopImpChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopAndBaImport*:
$\vdash$ (*f*; *f1*) $\wedge$ *ba g* $\longrightarrow$ (*f* $\wedge$ *g*); (*f1* $\wedge$ *g*)
proof −
have *1*: $\vdash$ *ba g* $\wedge$ (*f*; *f1*) $\longrightarrow$ (*g* $\wedge$ *f*); (*g* $\wedge$ *f1*) **by** (*rule BaAndChopImport*)
have *2*: $\vdash$ (*g* $\wedge$ *f*); (*g* $\wedge$ *f1*) = (*f* $\wedge$ *g*); (*f1* $\wedge$ *g*) **by** (*rule AndChopAndCommute*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *BaImpBaImpBaAnd*:
$\vdash$ *ba h* $\longrightarrow$ *ba*(*g* $\longrightarrow$ *ba h* $\wedge$ *g*)
proof −
have *1*: $\vdash$ *ba h* $\longrightarrow$ (*g* $\longrightarrow$ *ba h* $\wedge$ *g*) **by** *fastforce*
hence *2*: $\vdash$ *ba*(*ba h*) $\longrightarrow$ *ba*(*g* $\longrightarrow$ *ba h* $\wedge$ *g*) **by** (*rule BaImpBa*)
have *3*: $\vdash$ *ba h* = *ba*(*ba h*) **by** (*rule BaEqvBaBa*)
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BaChopImpChopBa*:
$\vdash$ *ba f* $\longrightarrow$ *g*; *g1* $\longrightarrow$ *g*; ((*ba f*) $\wedge$ *g1*)
proof −
have *1*: $\vdash$ *ba f* $\longrightarrow$ *ba* (*g1* $\longrightarrow$ (*ba f*) $\wedge$ *g1*) **by** (*rule BaImpBaImpBaAnd*)
have *2*: $\vdash$ *ba* (*g1* $\longrightarrow$ *ba f* $\wedge$ *g1*) $\longrightarrow$ *g*; *g1* $\longrightarrow$ *g*; (*ba f* $\wedge$ *g1*) **by** (*rule BaRightChopImpChop*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *DiNotBaImpNotBa*:
$\vdash$ *di* ($\neg$ (*ba f*)) $\longrightarrow$ $\neg$ (*ba f*)
proof −

have *1*: ⊢ *ba f* = *ba*(*ba f*) **by** (*rule BaEqvBaBa*)
have *2*: ⊢ *ba* (*ba f*) ⟶ *bi* (*ba f*) **by** (*rule BaImpBi*)
have *3*: ⊢ *ba f* ⟶ *bi* (*ba f*) **using** *1 2* **by** *fastforce*
hence *4*: ⊢ *ba f* ⟶ ¬ (*di* (¬ (*ba f*))) **by** (*simp add*: *bi-d-def*)
from *4* **show** *?thesis* **by** *fastforce*
qed

lemma *NotBaChopImpNotBa*:
⊢ (¬ (*ba f*)); *g* ⟶ ¬ (*ba f*)
proof −
have *1*: ⊢ (¬ (*ba f*)); *g* ⟶ *di* (¬ (*ba f*)) **by** (*rule ChopImpDi*)
have *2*: ⊢ *di* (¬ (*ba f*)) ⟶ ¬ (*ba f*) **by** (*rule DiNotBaImpNotBa*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *DiamondFinImpFin*:
⊢ ◇ (*fin f*) ⟶ *fin f*
proof −
have *1*: ⊢ *fin f* = #*True*;(*f* ∧ *empty*)
by (*rule FinEqvTrueChopAndEmpty*)
hence *2*: ⊢ ◇ (*fin f*) = #*True*;(#*True*;(*f* ∧ *empty*))
by (*metis ChopEqvChop TrueEqvTrueChopTrue inteq-reflection sometimes-d-def*)
have *3*: ⊢ #*True*;(#*True*;(*f* ∧ *empty*)) = (#*True*;#*True*);(*f* ∧ *empty*)
by (*rule ChopAssoc*)
have *4*: ⊢ (#*True*;#*True*);(*f* ∧ *empty*) = #*True*;(*f* ∧ *empty*)
using *TrueEqvTrueChopTrue* **using** *LeftChopEqvChop* **by** (*metis int-eq*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopFinImpFin*:
⊢ *f*; *fin* (*init w*) ⟶ *fin* (*init w*)
proof −
have *1*: ⊢ *f*; *fin* (*init w*) ⟶ ◇ (*fin* (*init w*)) **by** (*rule ChopImpDiamond*)
have *2*: ⊢ ◇ (*fin* (*init w*)) ⟶ *fin* (*init w*) **by** (*rule DiamondFinImpFin*)
from *1 2* **show** *?thesis* **using** *lift-imp-trans* **by** *blast*
qed

lemma *FinImpYieldsFin*:
⊢ *fin* (*init w*) ⟶ *f yields* (*fin* (*init w*))
proof −
have *1*: ⊢ *f*; *fin* (*init* (¬ *w*)) ⟶ *fin* (*init* (¬ *w*))
by (*rule ChopFinImpFin*)
have *2*: ⊢ *fin* (*init* (¬ *w*)) = (¬ (*fin* (*init w*)))
using *FinNotStateEqvNotFinState* **by** *blast*
hence *3*: ⊢ *f*; *fin* (*init* (¬ *w*)) = *f*; (¬ (*fin* (*init w*)))
by (*rule RightChopEqvChop*)
have *4*: ⊢ *f*; (¬ (*fin* (*init w*))) ⟶ ¬ (*fin* (*init w*))
using *1 2 3* **by** *fastforce*
hence *5*: ⊢ *fin* (*init w*) ⟶ ¬ (*f*; (¬ (*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: ⊢ (init w) ∧ ((init w) ⟶ fin (init w1)) ⟶ fin (init w1)
    by auto
 have 3: ⊢ ((init w) ∧ ((init w)⟶ fin (init w1))); ((init w1) ⟶ fin (init w2))
       ⟶
       ( fin (init w1)); ((init w1) ⟶ fin (init w2))
    using 2 by (rule LeftChopImpChop)
 have 4: ⊢ ( fin (init w1)); ((init w1) ⟶ fin (init w2)) =
        ◇((init w1) ∧ ((init w1) ⟶ fin (init w2)))
    by (rule FinChopEqvDiamond)
 have 41: ⊢ ((init w1) ∧ ((init w1) ⟶ fin (init w2))) ⟶ fin (init w2)
    by auto
 have 42: ⊢ ◇((init w1) ∧ ((init w1) ⟶ fin (init w2))) ⟶ ◇ ( fin (init w2))
    using 41 DiamondImpDiamond by blast
 have 5: ⊢ ◇( fin( init w2)) ⟶ fin (init w2)
    using DiamondFinImpFin by blast
 have 6: ⊢ (init w) ∧ ((init w)⟶ fin (init w1)); ((init w1) ⟶ fin (init w2))
          ⟶ fin (init w2)
    using 1 3 4 5 42 by fastforce
 from 6 show ?thesis by fastforce
qed

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

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

lemma *MoreChopLoop*:
assumes $\vdash\ f \longrightarrow\ more\ ;\ f$
shows $\vdash \neg\ f$
proof −
have *1*: $\vdash f \longrightarrow\ more\ ;\ f$
using *assms* **by** *auto*
hence *11*: $\vdash \Diamond\ f \longrightarrow \Diamond\ (more;f)$
by (*rule DiamondImpDiamond*)
have *12*: $\vdash \Diamond\ (more;f) = \#True;(more;f)$
by (*simp add*: *sometimes-d-def*)
have *13*: $\vdash \#True;(more;f) = (\#True;more);f$
by (*rule ChopAssoc*)
have *14*: $\vdash\ \Diamond\ (more;f) = more;f$
using *TrueChopMoreEqvMore 12 13* **by** (*metis int-eq*)
have *2*: $\vdash\ more\ ;\ f = \bigcirc(\Diamond\ f)$
by (*rule MoreChopEqvNextDiamond*)
have *3*: $\vdash\ \Diamond\ f \longrightarrow \bigcirc(\Diamond\ f)$
using *11 14 2* **by** *fastforce*
hence *4*: $\vdash \neg\ (\Diamond\ f)$
by (*rule NextLoop*)
have *5*: $\vdash \neg\ (\Diamond\ f) \longrightarrow \neg\ f$
using *NowImpDiamond* **by** *fastforce*
from *4 5* **show** *?thesis* **using** *MP* **by** *blast*
qed

lemma *MoreChopContra*:
assumes $\vdash\ f \wedge \neg\ g \longrightarrow (\ more\ ;\ (f \wedge \neg\ g))$
shows $\vdash f \longrightarrow g$
proof −
have *1*: $\vdash f \wedge \neg\ g \longrightarrow (\ more\ ;\ (f \wedge \neg\ g))$ **using** *assms* **by** *auto*

hence *2*: $\vdash \neg\ (f \wedge \neg\ g)$ **by** (*rule MoreChopLoop*)
from *2* **show** *?thesis* **by** *auto*
qed

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

lemma *ChopContra*:
assumes $\vdash\ f \wedge \neg\ g \longrightarrow h;\ f \wedge \neg\ (h;\ g)$
$\vdash h \longrightarrow$ *more*
shows $\vdash f \longrightarrow g$
proof −
have *1*: $\vdash f \wedge \neg\ g \longrightarrow h;\ f \wedge \neg\ (h;\ g)$ **using** *assms* **by** *auto*
have *2*: $\vdash h \longrightarrow$ *more* **using** *assms* **by** *auto*
have *3*: $\vdash h;\ f \wedge \neg\ (h;\ g) \longrightarrow h;\ (f \wedge \neg\ g)$ **by** (*rule ChopAndNotChopImp*)
have *4*: $\vdash h;\ (f \wedge \neg\ g) \longrightarrow$ *more* ; $(f \wedge \neg\ g)$ **using** *2* **by** (*rule LeftChopImpChop*)
have *5*: $\vdash f \wedge \neg\ g \longrightarrow$ *more* ; $(f \wedge \neg\ g)$ **using** *1 3 4* **by** *fastforce*
from *5* **show** *?thesis* **using** *MoreChopContra* **by** *auto*
qed

4.7 Properties of Chopstar and Chopplus

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

lemma *CSEqvOrChopCS*:
$\vdash\ f^\star =\ (\mathit{empty}\ \vee\ (f;\ f^\star))$
proof −
have *1*: $\vdash f^\star = (\mathit{empty} \vee (f \wedge \mathit{more});f^\star)$ **by** (*rule ChopstarEqv*)
have *2*: $\vdash (f \wedge \mathit{more});f^\star \longrightarrow f;f^\star$ **by** (*rule AndChopA*)
have *3*: $\vdash f^\star \longrightarrow$ *empty* $\vee$ f; $f^\star$ **using** *1 2* **by** (*metis int-iffD1 Prop08*)
have *4*: $\vdash \mathit{empty} \longrightarrow f^\star$ **by** (*rule EmptyImpCS*)
have *5*: $\vdash f \longrightarrow \mathit{empty} \vee (f \wedge \mathit{more})$ **by** (*auto simp*: *empty-d-def*)
have *6*: $\vdash f;\ f^\star \longrightarrow f^\star \vee\ (f \wedge$ *more* $);\ f^\star$ **using** *5* **by** (*rule EmptyOrChopImpRule*)
have *7*: $\vdash f^\star \longrightarrow \mathit{empty} \vee (f \wedge \mathit{more});f^\star$ **using** *1* **by** *fastforce*
have *8*: $\vdash f;\ f^\star \longrightarrow \mathit{empty} \vee\ (f \wedge$ *more* $);\ f^\star$ **using** *6 7* **by** *fastforce*

hence *9*: $\vdash f;\ f^\star \longrightarrow f^\star$ **using** *1* **by** *fastforce*
have *10*: $\vdash$ *empty* $\vee\ f;\ f^\star \longrightarrow f^\star$ **using** *9 4* **by** *fastforce*
from *3 10* **show** *?thesis* **by** *fastforce*
qed

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

lemma *CSAndMoreImpChopCS*:
$\vdash\ \ f^\star \wedge\ more \longrightarrow f;\ f^\star$
proof −
have *1*: $\vdash (f^\star \wedge\ more) = (f \wedge\ more);\ f^\star$ **by** (*rule CSAndMoreEqvAndMoreChop*)
have *2*: $\vdash (f \wedge\ more);\ f^\star \longrightarrow f;\ f^\star$ **by** (*rule AndChopA*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

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

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

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

by *auto*
hence *4*: $\vdash (f \wedge\ more\ ); f^\star \longrightarrow (f \wedge\ more\ ) \vee\ ((f \wedge\ more\ ); (f^\star \wedge\ more\ ))$
by (*rule ChopEmptyOrImpRule*)
hence *5*: $\vdash (f \wedge\ more\ ); f^\star \wedge \neg(f \wedge more) \longrightarrow\ \ ((f \wedge\ more\ ); (f^\star \wedge\ more\ ))$
by *fastforce*
have *6*: $\vdash (f \wedge\ more\ ); f^\star =\ ((f \wedge\ more\ ); f^\star \wedge more)$ **using** *1*
by *auto*
have *7*: $\vdash ((f \wedge\ more\ ); f^\star \wedge \neg(f \wedge more)) = ((f \wedge\ more\ ); f^\star \wedge more \wedge \neg(f \wedge more))$
using *6* **by** *auto*
have *8*: $\vdash (f \wedge\ more\ ); f^\star \wedge\ more\ \wedge \neg\ f \longrightarrow (f \wedge\ more\ ); (f^\star \wedge\ more\ )$
using *5 7* **by** *auto*
have *9*: $\vdash (f^\star \wedge\ more\ \wedge \neg\ f) = ((f^\star \wedge\ more) \wedge (more\ \wedge \neg\ f))$
by *auto*
have *10*: $\vdash ((f^\star \wedge\ more) \wedge (more\ \wedge \neg\ f)) = ((f \wedge\ more\ ); f^\star \wedge\ (more\ \wedge \neg\ f))$
using *1* **by** *fastforce*
from *1 8 9 10* **show** *?thesis* **by** *fastforce*
qed

lemma *CSAndMoreImpCSChop*:
$\vdash\ \ f^\star \wedge\ more\ \longrightarrow f^\star; f$
proof −
have *1*: $\vdash (f^\star \wedge\ more) =\ (f \wedge\ more\ ); f^\star$
by (*rule CSAndMoreEqvAndMoreChop*)
have *2*: $\vdash\ empty\ \vee\ \ more$
by (*auto simp*: *empty-d-def*)
hence *3*: $\vdash f^\star \longrightarrow\ empty\ \vee\ (f^\star \wedge\ more\ )$
by *auto*
hence *4*: $\vdash (f \wedge\ more\ ); f^\star \longrightarrow$
$(f \wedge\ more\ ) \vee\ ((f \wedge\ more\ ); (f^\star \wedge more\ ))$
by (*rule ChopEmptyOrImpRule*)
have *5*: $\vdash f^\star \wedge\ more\ \wedge \neg\ f \longrightarrow (f \wedge\ more\ ); (f^\star \wedge\ more\ )$
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*:
$\vdash \quad \neg\ (f^\star) \longrightarrow\ more$
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*: $\vdash f^\star = \ (empty \ \vee \ (f \wedge \ more \ ); f^\star)$
by (*rule ChopstarEqv*)
have *2*: $\vdash \ empty \longrightarrow g$
using *assms* **by** *blast*
have *3*: $\vdash (f \wedge \ more \ ); g \longrightarrow g$
using *assms* **by** *blast*
have *31*: $\vdash \ \neg \ g \longrightarrow more$
using *2* **by** (*auto simp*: *empty-d-def*)
have *32*: $\vdash \neg \ g \longrightarrow \neg \ ((f \wedge \ more \ ); g)$
using *3* **by** *fastforce*
have *33*: $\vdash f^\star \wedge more \longrightarrow (f \wedge \ more \ ); f^\star$
using *1* **using** *CSAndMoreEqvAndMoreChop* **by** *fastforce*
have *34*: $\vdash f^\star \wedge \neg \ g \longrightarrow f^\star \wedge more$
using *31* **by** *auto*
have *35*: $\vdash f^\star \wedge \neg \ g \longrightarrow (f \wedge \ more \ ); f^\star$
using *33 34* **by** *fastforce*
have *36*: $\vdash f^\star \wedge \neg \ g \longrightarrow \neg \ ((f \wedge \ more \ ); g)$

using *32* **by** *auto*
have *4*: $\vdash f^\star \wedge \neg\ g \longrightarrow (f \wedge\ more\ );\ f^\star \wedge \neg\ ((f \wedge more\ );\ g)$
using *35 36* **by** *fastforce*
have *5*: $\vdash f \wedge\ more \longrightarrow\ more$
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*: $\vdash (f^\star \wedge\ more\ );\ f^\star \longrightarrow f^\star;\ f^\star$ **by** (*rule AndChopA*)
have *3*: $\vdash f^\star;\ f^\star \longrightarrow f^\star$ **by** (*rule CSChopCSImpCS*)
have *4*: $\vdash (f^\star \wedge\ more\ );\ f^\star \longrightarrow f^\star$ **using** *2 3 lift-imp-trans* **by** *blast*
from *1 4* **show** *?thesis* **using** *CSElim* **by** *blast*
qed

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

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

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

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

lemma *NotAndMoreChopAndEmpty*:
$\vdash \neg(((f \wedge more);g) \wedge empty)$

by (*metis AndChopA ChopEmpty LeftChopImpMoreRule Prop01 empty-d-def int-simps*(*14*)
int-simps(*25*) *int-simps*(*4*) *inteq-reflection lift-and-com*)

lemma *NotChopAndMoreAndEmpty*:
$\vdash \neg((f;(g \wedge more)) \wedge empty)$
by (*metis* (*no-types*, *lifting*) *NotAndMoreChopAndEmpty REmptyEqvEmpty RMoreEqvMore RevChop*
ReverseEqv inteq-reflection rev-fun1 rev-fun2)

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

lemma *AndMoreChopAndMoreEqvAndMoreChop*:
$\vdash ((f \wedge more);g \wedge more) = (f \wedge more);g$
using *ChopImpDi DiAndB DiMoreEqvMore* **by** *fastforce*

lemma *ChopPlusEqv*:
$\vdash (f;f^\star) = (f \vee (f \wedge more); (f;f^\star))$
proof −
have *1*: $\vdash f^\star = (empty \vee (f \wedge more); f^\star)$
by (*rule ChopstarEqv*)
have *2*: $\vdash f^\star = (empty \vee f;f^\star)$
by (*rule CSEqvOrChopCS*)
hence *3*: $\vdash (empty \vee f;f^\star) = (empty \vee (f \wedge more);f^\star)$
using *1 2* **by** *fastforce*
have *4*: $\vdash (f \wedge more);(f^\star) = (f \wedge more);(empty \vee f;f^\star)$
using *2* **using** *RightChopEqvChop* **by** *blast*
hence *5*: $\vdash empty \vee f;f^\star = empty \vee (f \wedge more);(empty \vee f;f^\star)$
using *3 4* **by** *fastforce*
have *6*: $\vdash (f \wedge more); (empty \vee f;f^\star) =$
$((f \wedge more); empty \vee (f \wedge more); (f;f^\star))$
using *ChopOrEqv* **by** *blast*
have *7*: $\vdash (f \wedge more); empty = (f \wedge more)$
using *ChopEmpty* **by** *blast*
have *8*: $\vdash (empty \vee f;f^\star) =$
$(empty \vee (f \wedge more) \vee (f \wedge more); (f;f^\star))$
using *5 6 7* **by** (*metis 2 3 inteq-reflection*)
have *9*: $\vdash ((empty \vee f;f^\star) \wedge more) = (f;f^\star \wedge more)$
by (*auto simp*: *empty-d-def*)
have *10*: $\vdash ((empty \vee (f \wedge more) \vee (f \wedge more); (f;f^\star)) \wedge more) =$
$(((f \wedge more) \vee (f \wedge more); (f;f^\star)) \wedge more)$
by (*auto simp*: *empty-d-def*)

have *11*: $\vdash (((f \wedge\ more\ ) \vee\ (f \wedge\ more\ );(f;f^\star)) \wedge more) =$
$((f \wedge\ more\ ) \vee\ (f \wedge\ more\ );(f;f^\star))$
using *10 6 7 int-eq*
using *AndMoreChopAndMoreEqvAndMoreChop* **by** *fastforce*
have *12*: $\vdash (f;f^\star \wedge more) = ((f \wedge\ more\ ) \vee\ (f \wedge\ more\ );(f;f^\star))$
using *8 9 10 11* **by** *fastforce*
have *13*: $\vdash (f;f^\star \wedge empty) = (f \wedge empty)$
by (*rule ChopCsAndEmptyEqvAndEmpty*)
have *14*: $\vdash ((f \wedge\ more\ ) \vee\ (f \wedge\ more\ );(f;f^\star) \vee (f \wedge empty)) =$
$(f \vee\ (f \wedge\ more\ );(f;f^\star))$
by (*auto simp*: *empty-d-def*)
have *15*: $\vdash f;f^\star = ((\ f;f^\star \wedge empty) \vee (\ f;f^\star \wedge more))$
by (*auto simp*: *empty-d-def*)
from *12 13 14 15* **show** *?thesis* **by** *fastforce*
qed

lemma *ChopPlusImpChopPlus*:
assumes $\vdash\ \ f \longrightarrow g$
shows $\vdash\ \ f;f^\star \longrightarrow g;g^\star$
proof −
have *1*: $\vdash f \longrightarrow g$
using *assms* **by** *auto*
have *2*: $\vdash f;f^\star\ = (f \vee\ (f \wedge\ more\ );(f;f^\star))$
by (*rule ChopPlusEqv*)
have *3*: $\vdash g;g^\star\ = (g \vee\ (g \wedge\ more\ );(g;g^\star))$
by (*rule ChopPlusEqv*)
have *4*: $\vdash\ \ f;f^\star\ \wedge \neg\ (g;g^\star\ ) \longrightarrow ((f \wedge\ more\ );(f;f^\star)\ ) \wedge \neg\ ((g \wedge\ more\ );(g;g^\star)\ )$
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*)
by (*simp add*: *DiamondAndEmptyEqvAndEmpty*)
have *4*: $\vdash$ ($\neg$ (*init w*)) = (*init* ($\neg$ *w*)) **using** *Initprop*(*2*) **by** *blast*
have *5*: $\vdash$ ($\neg$ (*init w*) $\wedge$ *empty*) = ($\neg$ *w* $\wedge$ *empty*)
using *4 InitAndEmptyEqvAndEmpty* **by** (*metis inteq-reflection*)
have *6*: $\vdash$ ($\neg$($\Box$ (*init w*)) $\wedge$ *empty*) = ($\neg$ *w* $\wedge$ *empty*)
using *2 3 5* **by** *fastforce*
have *7*: $\vdash$ $\neg$(*init w* $\wedge$ $\neg$($\Box$ (*init w*)) $\wedge$ *empty*)
using *1 6* **by** *fastforce*
from *7* **show** *?thesis* **by** *auto*
qed

lemma *BoxImpTrueChopAndEmpty*:
$\vdash$ $\Box$ *f* $\longrightarrow$ #*True*;(*f* $\wedge$ *empty*)
using *BoxAndChopImport Finprop*(*3*) **by** *fastforce*

lemma *BoxInitAndMoreImpBoxInitAndMoreAndFinInit*:
$\vdash$ $\Box$(*init w*) $\wedge$ *more* $\longrightarrow$ ($\Box$ (*init w*) $\wedge$ *more*) $\wedge$ *fin* (*init w*)
proof −
have *1*: $\vdash$ *fin* (*init w*) = #*True* ; (*init w* $\wedge$ *empty*) **using** *FinEqvTrueChopAndEmpty* **by** *blast*
have *2*: $\vdash$ $\Box$(*init w*) $\longrightarrow$ #*True*;(*init w* $\wedge$ *empty*) **by** (*rule BoxImpTrueChopAndEmpty*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *CSImpBox*:
assumes $\vdash$ *f* $\longrightarrow$ *empty* $\vee$ ($\Box$ (*init w*) $\wedge$ *more*); *f*
shows $\vdash$ *init w* $\wedge$ *f* $\longrightarrow$ $\Box$ (*init w*)
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ *empty* $\vee$ ($\Box$(*init w*) $\wedge$ *more*); *f*
using *assms* **by** *auto*
have *2*: $\vdash$ *init w* $\wedge$ $\neg$($\Box$ (*init w*)) $\longrightarrow$ $\neg$ *empty*
by (*rule InitAndNotBoxInitImpNotEmpty*)
have *3*: $\vdash$ *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*:
$\vdash\ \ (init\ w \wedge (\Box(\ init\ w))^\star) = \Box\ (init\ w)$
proof −
have *1*: $\vdash (\Box\ (init\ w))^\star =\ (empty\ \vee\ (\Box\ (init\ w) \wedge\ more\ );\ (\Box\ (init\ w))^\star)$
by (*rule ChopstarEqv*)
hence *2*: $\vdash (\Box\ (init\ w))^\star \longrightarrow\ empty\ \vee\ (\Box\ (init\ w) \wedge\ more\ );\ (\Box\ (init\ w))^\star$
by *fastforce*
hence *3*: $\vdash init\ w \wedge (\Box\ (init\ w))^\star \longrightarrow \Box\ (init\ w)$
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*:
$\vdash\quad ba\ (f \longrightarrow g) \longrightarrow f^\star \longrightarrow g^\star$
proof −
have *1*: $\vdash f^\star =\ (empty\ \vee\ (f \wedge\ more\ );f^\star)$
by (*rule ChopstarEqv*)
have *2*: $\vdash g^\star =\ (empty\ \vee\ (g \wedge\ more\ );g^\star)$
by (*rule ChopstarEqv*)
have *21*: $\vdash \neg(g^\star) =\ (\neg empty\ \wedge \neg(\ (g \wedge\ more\ );g^\star))$
using *2* **by** *fastforce*
hence *22*: $\vdash \neg(g^\star) =\ (more\ \wedge \neg(\ (g \wedge\ more\ );g^\star))$
using *NotEmptyEqvMore* **by** *fastforce*
have *3*: $\vdash f^\star \wedge \neg\ (g^\star) \longrightarrow$
$(empty\ \vee (f \wedge\ more\ );f^\star) \wedge more \wedge \neg\ ((g \wedge\ more\ );g^\star)$
using *1 22* **by** *fastforce*
have *31*: $\vdash ((empty\ \vee (f \wedge\ more\ );f^\star) \wedge more) = ((f \wedge\ more\ );f^\star \wedge more)$
by (*auto simp*: *empty-d-def*)
have *32*: $\vdash f^\star \wedge \neg\ (g^\star) \longrightarrow (f \wedge\ more\ );f^\star \wedge \neg\ ((g \wedge\ more\ );g^\star)$
using *3 31* **by** *fastforce*
have *4*: $\vdash (f \longrightarrow g) \longrightarrow (f \wedge\ more\ \longrightarrow g \wedge\ more\ )$
by *auto*
hence *5*: $\vdash\ ba\ (f \longrightarrow g) \longrightarrow\ ba\ (f \wedge\ more\ \longrightarrow g \wedge\ more\ )$
by (*rule BaImpBa*)
have *6*: $\vdash\ ba\ (f \wedge\ more\ \longrightarrow g \wedge\ more\ ) \longrightarrow$

$(f \wedge \ more\ ); f^\star \longrightarrow \ (g \wedge \ more\ ); f^\star$
by (*rule BaLeftChopImpChop*)
have *7*: $\vdash \ ba\ (f \longrightarrow g) \wedge (f \wedge more\ ); f^\star \longrightarrow (g \wedge \ more\ ); f^\star$
using *5 6* **by** *fastforce*
have *8*: $\vdash (g \wedge \ more\ ); f^\star \wedge \neg\ ((g \wedge \ more\ ); g^\star)$
$\longrightarrow \ (g \wedge \ more\ ); (f^\star \wedge \neg\ (g^\star))$
by (*rule ChopAndNotChopImp*)
have *9*: $\vdash (g \wedge \ more\ ); (f^\star \wedge \neg\ (g^\star)) \longrightarrow \ more\ ; (f^\star \wedge \neg\ (g^\star))$
by (*rule AndChopB*)
have *10*: $\vdash \ ba\ (f \longrightarrow g) \longrightarrow more\ ; (f^\star \wedge \neg\ (g^\star)) \longrightarrow$
$more\ ; (\ ba\ (f \longrightarrow g) \wedge f^\star \wedge \neg\ (g^\star))$
by (*rule BaChopImpChopBa*)
have *11*: $\vdash \ ba\ (f \longrightarrow g) \wedge f^\star \wedge \neg\ (g^\star) \longrightarrow$
$more\ ; (\ ba\ (f \longrightarrow g) \wedge f^\star \wedge \neg\ (g^\star))$
using *32 7 8 9 10* **by** *fastforce*
hence *12*: $\vdash \neg\ (\ (ba\ (f \longrightarrow g)) \wedge (f^\star) \wedge (\neg\ (g^\star)))$
using *MoreChopLoop* **by** *blast*
from *12* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

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

lemma *BaAndCSImport*:
$\vdash \quad ba\ \ f \wedge g^\star \longrightarrow (f \wedge g)^\star$
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*:
$\vdash skip^\star$
by (*metis ChopPlusImpCS EmptyImpCS EmptyNextInducta next-d-def*)

4.8 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*: $\vdash \ while\ \ (init\ w)\ \ do\ \ f = (((init\ w) \wedge f)^\star \wedge \ fin\ (\neg\ \ (init\ w)))$

by (*simp add*: *while-d-def*)
have *2*: $\vdash (init\ w \wedge f)^\star = (empty \vee ((init\ w \wedge f); (init\ w \wedge f)^\star))$
by (*rule CSEqvOrChopCS*)
have *21*: $\vdash (((init\ w) \wedge f)^\star \wedge fin\ (\neg\ (init\ w))) =$
$((empty \vee ((init\ w \wedge f); (init\ w \wedge f)^\star)) \wedge fin\ (\neg\ (init\ w)))$
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*: $\vdash (init\ w \wedge (f; ((init\ w \wedge f)^\star \wedge fin\ (\neg\ (\ init\ \ w))))) =$
$(init\ w \wedge (f;\ while\ (init\ w)\ do\ f))$ **using** *6*
by *auto*
have *62*: $\vdash (\ empty \wedge fin\ (\neg\ (init\ w))) \vee (\ ((init\ w \wedge f); (init\ w \wedge f)^\star) \wedge fin\ (\neg\ (init\ w)))$
$= (\neg\ (\ init\ w) \wedge empty\ ) \vee (init\ w \wedge (f;\ while\ (init\ w)\ do\ f))$
using *21 22 3 4 52 61* **by** *fastforce*
have *7*: $\vdash while\ (init\ w)\ do\ f$
$= ((\neg\ (\ init\ w) \wedge empty\ ) \vee (init\ w \wedge (f;\ while\ (init\ w)\ do\ f)))$
using *1 21 22 62*
by (*metis 3 41 42 5 51 inteq-reflection*)
have *71*: $\vdash if_i\ (init\ w)\ then\ (f; (\ while\ (init\ w)\ do\ f))\ else\ empty =$
$((\neg\ (\ init\ w) \wedge empty\ ) \vee (init\ w \wedge (f;\ while\ (init\ w)\ do\ f)))$
by (*auto simp*: *ifthenelse-d-def*)
from *7 71* **show** *?thesis* **by** *fastforce*
qed

lemma *WhileChopEqvIf*:
$\vdash\ (\ while\ (\ init\ w)\ do\ f); g = if_i(init\ w)\ then\ (f; ((while\ (\ init\ w)\ do\ f); g))\ else\ g$
proof −
have *1*: $\vdash while\ (init\ w)\ do\ f =$

$if_i\ (init\ w)\ then\ (f;\ (\ while\ (init\ w)\ do\ f))\ else\ \ empty$
by (*rule WhileEqvIf*)
hence *2*: $\vdash\ (\ while\ (init\ w)\ do\ f);\ g =$
$if_i\ (init\ w)\ then\ ((f;\ while\ (init\ w)\ do\ f);\ g)\ else\ \ (empty\ ;\ g)$
by (*rule IfChopEqvRule*)
have *3*: $\vdash\ empty\ ;\ g = g$
by (*rule EmptyChop*)
have *4*: $\vdash if_i\ (init\ w)\ then\ ((f;\ while\ (init\ w)\ do\ f);\ g)\ else\ \ (empty\ ;\ g) =$
$if_i\ (init\ w)\ then\ ((f;\ while\ (init\ w)\ do\ f);\ g)\ else\ \ g$
using *3* **using** *inteq-reflection* **by** *fastforce*
have *5*: $\vdash ((f;\ while\ (init\ w)\ do\ f);\ g) = (f;\ (while\ (init\ w)\ do\ f;\ g))$
by (*rule ChopAssocB*)
have *6*: $\vdash if_i\ (init\ w)\ then\ ((f;\ while\ (init\ w)\ do\ f);\ g)\ else\ \ g =$
$if_i\ (init\ w)\ then\ (f;\ ((while\ (init\ w)\ do\ f);\ g))\ else\ \ g$
using *5* **using** *inteq-reflection* **by** *fastforce*
from *1 2 4 6* **show** *?thesis* **by** *fastforce*
qed

lemma *WhileChopEqvIfRule*:
assumes $\vdash\ f = (\ while\ (init\ w)\ do\ g);\ h$
shows $\vdash f =\ if_i\ (init\ w)\ then\ (g;\ f)\ else\ h$
proof −
have *1*: $\vdash f = (\ while\ (init\ w)\ do\ g);\ h$
using *assms* **by** *auto*
have *2*: $\vdash (\ while\ (init\ w)\ do\ g);\ h =$
$if_i\ (init\ w)\ then\ (g;\ ((\ while\ (init\ w)\ do\ g);\ h))\ else\ h$
by (*rule WhileChopEqvIf*)
have *3*: $\vdash (g;\ f) = (g;\ ((\ while\ (init\ w)\ do\ g);\ h))$
using *1* **by** (*rule RightChopEqvChop*)
have *4*: $\vdash (g;\ ((\ while\ (init\ w)\ do\ g);\ h)) = (g;\ f)$
using *3* **by** *auto*
have *5*: $\vdash if_i\ (init\ w)\ then\ (g;\ ((\ while\ (init\ w)\ do\ g);\ h))\ else\ h =$
$if_i\ (init\ w)\ then\ (g;\ f)\ else\ h$
using *4* **using** *inteq-reflection* **by** *fastforce*
from *1 2 5* **show** *?thesis* **by** *fastforce*
qed

lemma *WhileImpFin*:
$\vdash\ \ while\ (init\ w)\ do\ f \longrightarrow fin\ (\neg\ (\ init\ w))$
proof −
have *1*: $\vdash (init\ w \wedge f)^\star \wedge\ fin\ (\neg\ (\ init\ w)) \longrightarrow\ fin\ (\neg\ (\ init\ w))$ **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*: $\vdash (init\ w \wedge f)^\star\ =\ (empty\ \vee\ ((init\ w \wedge f) \wedge\ more\ );\ (init\ w \wedge f)^\star)$
by (*rule ChopstarEqv*)
have *2*: $\vdash ((init\ w \wedge f) \wedge\ more)\ = (init\ w \wedge (f \wedge\ more\ ))$

by *auto*
hence *3*: $\vdash ((init\ w \wedge f) \wedge\ more\ );\ (init\ w \wedge f)^\star\ = (init\ w \wedge f \wedge\ more\ );\ (init\ w \wedge f)^\star$
by (*rule LeftChopEqvChop*)
have *4*: $\vdash (init\ w \wedge f)^\star\ =\ (empty\ \vee\ (init\ w \wedge f \wedge\ more\ );\ (init\ w \wedge f)^\star)$
using *1 3* **by** *fastforce*
have *5*: $\vdash ((init\ w \wedge f)^\star \wedge\ fin\ (\neg\ (\ init\ w))) =$
$((\ empty\ \wedge\ fin\ (\neg\ (init\ w))) \vee$
$((init\ w \wedge f \wedge\ more\ );\ (init\ w \wedge f)^\star \wedge fin\ (\neg\ (\ init\ w))))$
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*: $\vdash (init\ w \wedge f \wedge\ more\ );\ (init\ w \wedge f)^\star = (init\ w \wedge (f \wedge\ more\ );\ (init\ w \wedge f)^\star)$
by (*rule StateAndChop*)
have *8*: $\vdash (((f \wedge\ more\ );\ (init\ w \wedge f)^\star) \wedge\ fin\ (\ init\ (\neg\ w))) =$
$((f \wedge\ more\ );\ ((init\ w \wedge f)^\star \wedge\ fin\ (\ \ init\ (\neg\ w))))$
by (*rule ChopAndFin*)
have *81*: $\vdash fin\ (\ \ init\ (\neg\ w)) = fin\ (\neg\ (\ init\ w))$
using *FinEqvFin Initprop*(*2*) **by** *fastforce*
have *82*: $\vdash ((f \wedge\ more\ );\ (init\ w \wedge f)^\star \wedge\ fin\ (\neg\ (\ init\ w))) =$
$((f \wedge\ more\ );\ ((init\ w \wedge f)^\star \wedge\ fin\ (\neg\ (\ init\ w))))$
using *8 81*
by (*metis inteq-reflection*)
have *9*: $\vdash ((init\ w \wedge f)^\star \wedge\ fin\ (\neg\ (\ init\ w)))\ =$
$((\neg\ (\ init\ w) \wedge\ empty\ ) \vee$
$(init\ w \wedge (f \wedge\ more\ );\ ((init\ w \wedge f)^\star \wedge\ fin\ (\neg\ (\ init\ w)))))$
using *5 6 7 82* **by** *fastforce*
from *9* **show** *?thesis* **by** (*simp add*: *while-d-def*)
qed

lemma *WhileIntro*:
assumes $\vdash\ \neg\ (\ init\ w) \wedge f \longrightarrow\ empty$
$\vdash init\ w \wedge f \longrightarrow (g \wedge\ more\ );\ f$
shows $\vdash f \longrightarrow\ while\ (\ init\ w)\ do\ \ g$
proof −
have *1*: $\vdash\ \neg\ (\ init\ w) \wedge f \longrightarrow\ empty$
using *assms* **by** *blast*
have *2*: $\vdash init\ w \wedge f \longrightarrow (g \wedge\ more\ );\ f$
using *assms* **by** *blast*
have *3*: $\vdash\ while\ (\ init\ w)\ \ do\ \ g =$
$((\neg\ \ (init\ w) \wedge\ \ empty\ ) \vee\ \ (init\ w \wedge (g \wedge\ \ more\ );\ while\ \ (init\ w)\ \ do\ \ g))$
by (*rule WhileEqvEmptyOrChopWhile*)
hence *31*: $\vdash \neg\ (\ while\ \ (init\ w)\ \ do\ \ g) =$
$(\neg(\ (\neg\ \ (init\ w) \wedge\ \ empty\ ) \vee\ \ (init\ w \wedge (g \wedge\ \ more\ );\ \ while\ \ (init\ w)\ \ do\ \ g)))$
by *fastforce*
hence *32*: $\vdash (f \wedge \neg\ (\ while\ \ (init\ w)\ \ do\ \ g)) =$
$(f \wedge \neg(\ (\neg(init\ w) \wedge empty) \vee\ \ (init\ w \wedge (g \wedge\ \ more\ );\ \ while\ \ (init\ w)\ \ do\ \ g)))$
by *fastforce*
have *33*: $\vdash (f \wedge \neg(\ (\neg(init\ w) \wedge empty) \vee (init\ w \wedge (g \wedge more);\ \ while\ \ (init\ w)\ \ do\ \ g))) =$
$(f \wedge \neg(\neg(init\ w) \wedge\ \ empty\ ) \wedge \neg(init\ w \wedge (g \wedge\ \ more\ );\ \ while\ \ (init\ w)\ \ do\ \ g))$
by *auto*

have *34*: $\vdash (f \wedge \neg(\neg(init\ w) \wedge empty) \wedge \neg((init\ w) \wedge ((g \wedge more);\ while\ (init\ w)\ do\ g))) =$
$(f \wedge (\ (init\ w) \vee more\ ) \wedge (\neg(init\ w) \vee \neg((g \wedge more);\ while\ (init\ w)\ do\ g)))$
by (*auto simp*: *empty-d-def*)
have *35*: $\vdash (f \wedge ((init\ w) \vee more) \wedge (\neg(init\ w) \vee \neg((g \wedge more);while\ (init\ w)\ do\ g))) =$
$((f \wedge (init\ w) \wedge \neg\ ((g \wedge more);\ while\ (\ init\ w)\ do\ g)) \vee$
$(f \wedge (init\ w) \wedge \neg(init\ w)) \vee$
$(f \wedge more \wedge \neg\ ((g \wedge more);\ while\ (\ init\ w)\ do\ g)) \vee$
$(f \wedge more \wedge \neg(init\ w)))$
by *auto*
have *36*: $\vdash (f \wedge \neg\ (\ while\ (init\ w)\ do\ g)) =$
$((f \wedge (init\ w) \wedge \neg\ ((g \wedge more);\ while\ (\ init\ w)\ do\ g)) \vee$
$(f \wedge (init\ w) \wedge \neg(init\ w)) \vee$
$(f \wedge more \wedge \neg\ ((g \wedge more);\ while\ (\ init\ w)\ do\ g)) \vee$
$(f \wedge more \wedge \neg(init\ w)))$ **using** *32 33 34 35* **by** *fastforce*
have *37*: $\vdash \neg(f \wedge more \wedge \neg(init\ w))$
using *1* **by** (*auto simp*: *empty-d-def*)
have *38*: $\vdash (f \wedge more \wedge \neg\ ((g \wedge more);\ while\ (\ init\ w)\ do\ g)) \longrightarrow$
$((g \wedge more);\ f \wedge \neg\ ((g \wedge more);\ while\ (\ init\ w)\ do\ g))$
using *1 2* **by** (*auto simp*: *empty-d-def Valid-def*)
have *39*: $\vdash (f \wedge (init\ w) \wedge \neg\ ((g \wedge more);\ while\ (\ init\ w)\ do\ g)) \longrightarrow$
$((g \wedge more);\ f \wedge \neg\ ((g \wedge more);\ while\ (\ init\ w)\ do\ g))$
using *2* **by** *auto*
have *40*: $\vdash ((f \wedge (init\ w) \wedge \neg\ ((g \wedge more);\ while\ (\ init\ w)\ do\ g)) \vee$
$(f \wedge (init\ w) \wedge \neg(init\ w)) \vee$
$(f \wedge more \wedge \neg\ ((g \wedge more);\ while\ (\ init\ w)\ do\ g)) \vee$
$(f \wedge more \wedge \neg(init\ w))) \longrightarrow$
$(g \wedge more);\ f \wedge \neg\ ((g \wedge more);\ while\ (\ init\ w)\ do\ g)$
using *39 38 37 38* **by** *fastforce*
have *4*: $\vdash f \wedge \neg\ (\ while\ (init\ w)\ do\ g) \longrightarrow$
$(g \wedge more);\ f \wedge \neg\ ((g \wedge more);\ while\ (\ init\ w)\ do\ g)$
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*: $\vdash\ while\ (\ init\ w)\ do\ f =$
$((\neg\ (\ init\ w) \wedge\ empty\ ) \vee\ (init\ w \wedge (f \wedge more);\ while\ (\ init\ w)\ do\ f))$
by (*rule WhileEqvEmptyOrChopWhile*)
hence *11*: $\vdash ((while\ (\ init\ w)\ do\ f) \wedge \neg\ g) =$
$(((\neg(init\ w) \wedge empty) \vee\ (init\ w \wedge (f \wedge more);while\ (init\ w)\ do\ f)) \wedge \neg\ g)$
by *auto*
have *2*: $\vdash \neg\ (\ init\ w) \wedge\ empty \longrightarrow g$
using *assms* **by** *blast*
hence *21*: $\vdash \neg\ g \longrightarrow \neg(\neg\ (\ init\ w) \wedge\ empty)$

by *auto*
have *22*: $\vdash$ $((\neg\ (init\ w) \wedge empty) \vee (init\ w \wedge (f \wedge more);while\ (init\ w)\ do\ f)) \wedge \neg\ g \longrightarrow$
$(init\ w \wedge (f \wedge more);\ while\ (init\ w)\ do\ f)$
using *21* **by** *auto*
have *23*: $\vdash (while\ (init\ w)\ do\ f) \wedge \neg\ g \longrightarrow$
$(init\ w \wedge (f \wedge more);\ while\ (init\ w)\ do\ f) \wedge \neg\ g$
using *11 21* **by** *fastforce*
have *3*: $\vdash (init\ w) \wedge ((f \wedge more);\ g) \longrightarrow g$
using *assms* **by** *blast*
hence *31*: $\vdash \neg\ g \longrightarrow \neg((init\ w) \wedge ((f \wedge more);\ g))$
by *fastforce*
have *32*: $\vdash (init\ w \wedge (f \wedge more);\ while\ (init\ w)\ do\ f) \wedge \neg\ g \longrightarrow$
$(((f \wedge more);\ (while\ (init\ w)\ do\ f)) \wedge \neg\ ((f \wedge more);\ g)) \wedge \neg g$
using *31* **by** *auto*
have *4*: $\vdash\ (while\ (init\ w)\ do\ f) \wedge \neg\ g \longrightarrow$
$((f \wedge more);\ (while\ (init\ w)\ do\ f)) \wedge \neg\ ((f \wedge more);\ g)$
using *23 32* **by** *fastforce*
have *5*: $\vdash f \wedge more \longrightarrow more$
by *auto*
from *4 5* **show** *?thesis* **using** *ChopContra* **by** *blast*
qed

lemma *BaWhileImpWhile*:
$\vdash\ \ ba\ (f \longrightarrow g) \longrightarrow (\ while\ (init\ w)\ do\ f) \longrightarrow (\ while\ (init\ w)\ do\ g)$
proof −
have *1*: $\vdash (f \longrightarrow g) \longrightarrow ((init\ w \wedge f) \longrightarrow (init\ w \wedge g))$
by *auto*
hence *2*: $\vdash\ ba\ (f \longrightarrow g) \longrightarrow\ ba\ ((init\ w \wedge f) \longrightarrow (init\ w \wedge g))$
by (*rule BaImpBa*)
have *3*: $\vdash\ ba\ ((init\ w \wedge f) \longrightarrow (init\ w \wedge g)) \longrightarrow ((init\ w \wedge f)^\star \longrightarrow (init\ w \wedge g)^\star)$
by (*rule BaCSImpCS*)
have *4*: $\vdash\ ba\ (f \longrightarrow g) \longrightarrow ((init\ w \wedge f)^\star \wedge fin\ (\neg\ (\ init\ w)) \longrightarrow (init\ w \wedge g)^\star \wedge fin\ (\neg\ (init\ w)))$
using *2 3* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*: *while-d-def*)
qed

lemma *WhileImpWhile*:
assumes $\vdash\ \ f \longrightarrow g$
shows $\vdash (\ while\ (init\ w)\ do\ f) \longrightarrow (\ while\ (init\ w)\ do\ g)$
proof −
have *1*: $\vdash f \longrightarrow g$
using *assms* **by** *auto*
hence *2*: $\vdash\ ba\ (f \longrightarrow g)$
by (*rule BaGen*)
have *3*: $\vdash\ ba\ (f \longrightarrow g) \longrightarrow (\ while\ (init\ w)\ do\ f) \longrightarrow (\ while\ (init\ w)\ do\ g)$
by (*rule BaWhileImpWhile*)
from *2 3* **show** *?thesis* **using** *MP* **by** *blast*
qed

4.9 Properties of Halt

lemma *WnextAndMoreEqvNext*:
$\vdash (wnext\ f \wedge more) = \bigcirc f$
by (*auto simp*: *wnext-defs more-defs next-defs*)

lemma *BoxStateAndEmptyEqvStateAndEmpty*:
$\vdash (\Box(empty = (init\ w)) \wedge empty) = ((init\ w) \wedge empty)$
by (*auto simp*: *always-defs init-defs empty-defs*)

lemma *BoxEmptyEqvIStateqvEmptyAndStateOrNotStateNext*:
$\vdash \Box(empty = (init\ w)) = ((empty \wedge init\ w) \vee (\neg(init\ w) \wedge \bigcirc(\Box(empty = (init\ w)))))$
proof −
have *1*: $\vdash \Box(empty = (init\ w)) =$
$((\Box(empty = (init\ w)) \wedge empty) \vee (\Box(empty = (init\ w)) \wedge more))$
by (*auto simp*: *empty-d-def*)
have *2*: $\vdash (\Box(empty = (init\ w)) \wedge empty) = ((init\ w) \wedge empty)$
using *BoxStateAndEmptyEqvStateAndEmpty* **by** *blast*
have *3*: $\vdash \Box(empty = (init\ w)) = ((empty = (init\ w)) \wedge wnext(\Box(empty = (init\ w))))$
using *BoxEqvAndWnextBox* **by** *blast*
hence *4*: $\vdash (\Box(empty = (init\ w)) \wedge more) =$
$(((empty = (init\ w)) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more))$
by *auto*
have *5*: $\vdash ((empty = (init\ w)) \wedge more) = (\neg(init\ w) \wedge more)$
by (*auto simp*: *empty-d-def*)
have *6*: $\vdash (wnext(\Box(empty = (init\ w))) \wedge more) = \bigcirc(\Box(empty = (init\ w)))$
using *WnextAndMoreEqvNext* **by** *metis*
have *7*: $\vdash (\Box(empty = (init\ w)) \wedge more) =$
$((\neg\ (init\ w) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)\ )$
using *4 5* **by** *fastforce*
have *8*: $\vdash ((\neg\ (init\ w) \wedge more) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)\ ) =$
$((\neg\ (init\ w)) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)\ )$ **by** *auto*
have *9*: $\vdash ((\neg\ (init\ w)) \wedge (wnext(\Box(empty = (init\ w))) \wedge more)\ ) =$
$((\neg\ (init\ w)) \wedge \bigcirc(\Box(empty = (init\ w))))$ **using** *8 6* **by** *auto*
have *10*: $\vdash \Box(empty = (init\ w)) = (((init\ w) \wedge empty) \vee (\Box(empty = (init\ w)) \wedge more)\ )$
using *1 2* **by** *fastforce*
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*:
$\vdash ((\#True;(f \wedge empty)) \wedge g) = (g;(f \wedge empty))$
using *AndFinEqvChopAndEmpty FinEqvTrueChopAndEmpty* **by** (*metis int-eq lift-and-com*)

lemma *WprevEqvEmptyOrPrev*:
$\vdash wprev\ f = (empty \vee prev\ f)$
by (*auto simp*: *wprev-defs empty-defs prev-defs*)

lemma *NotChopSkipEqvMoreAndNotChopSkip*:
$\vdash (\neg\ f);skip = (more \wedge \neg(f;skip))$
proof −
have *1*: $\vdash wprev\ f = (empty \vee prev\ f)$ **using** *WprevEqvEmptyOrPrev* **by** *auto*
hence *2*: $\vdash (\neg(wprev\ f)) = (\neg(empty \vee prev\ f))$ **by** *auto*
have *3*: $\vdash \neg(wprev\ f) = ((\neg\ f);skip)$ **by** (*simp add*: *wprev-d-def prev-d-def*)
have *31*: $\vdash (empty \vee prev\ f) = (empty \vee (f;skip))$ **by** (*simp add*: *prev-d-def*)
have *32*: $\vdash (empty \vee (f;skip)) = (\neg more \vee \neg\neg(f;skip))$ **by** (*simp add*: *empty-d-def*)
have *33*: $\vdash (\neg more \vee \neg\neg(f;skip)) = (\neg(more \wedge \neg(f;skip)))$ **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

4.10 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

4.11 Properties of Time Reversal

lemma *RNot*:
$\vdash (\neg f)^r = (\neg\ f^r)$
by (*simp add*: *rev-fun1*)

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

lemma *RTrue*:
$\vdash (\#True)^r = \#True$
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*)

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

lemma *RAndEmptyEqvAndEmpty*:
$\vdash (f \wedge empty)^r = (f \wedge empty)$
apply (*simp add*: *Valid-def empty-defs reverse-d-def*)
by (*metis interval-st-intlen intrev*.*simps*(*1*))

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 inteq-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*:
$\vdash$ $(init\ f)^r = fin(f)$
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*: $\vdash$ $\#True;(f^r \wedge empty) = \#True;(f \wedge empty)$
using *RAndEmptyEqvAndEmpty*
by (*metis REmptyEqvEmpty RightChopEqvChop all-rev-eq*(*3*) *int-eq*)
have *5*: $\vdash$ $\#True;(f \wedge empty) = fin(f)$
using *FinEqvTrueChopAndEmpty* **by** *fastforce*
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *RFinEqvInit*:
$\vdash$ $(fin\ f)^r = init\ (f)$
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*: $\vdash$ $(f\wedge empty)^r;\#True = (f\wedge empty);\#True$
by (*simp add*: *RAndEmptyEqvAndEmpty LeftChopEqvChop*)
have *6*: $\vdash$ $(f \wedge empty);\#True = init(f)$
by (*simp add*: *AndChopCommute init-d-def*)
from *1 2 3 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *RHaltEqvInitonly*:
$\vdash$ $(halt\ f)^r = initonly\ (f^r)$
proof −
have *1*: $\vdash$ $(halt\ f)^r = (\Box\ (\ empty = f\ ))^r$ **by** (*simp add*: *halt-d-def*)
have *2*: $\vdash$ $(\Box\ (\ empty = f\ ))^r = bi\ (\ (empty = f)^r)$ **by** (*simp add*: *RBoxEqvBi*)

have *3*: $\vdash (empty = f)^r = (empty = f^r)$ **by** (*metis REmptyEqvEmpty inteq-reflection rev-fun2*)
hence *4*: $\vdash bi\ (\ (empty = f)^r) = bi(empty = f^r)$ **by** (*simp add*: *BiEqvBi*)
have *5*: $\vdash bi(empty = f^r) = initonly(f^r)$ **by** (*simp add*: *initonly-d-def*)
from *1 2 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *RInitonlyEqvHalt*:
$\vdash (initonly\ f)^r = halt(f^r)$
proof −
have *1*: $\vdash (initonly\ f)^r = (bi\ (empty = f))^r$ **by** (*simp add*: *initonly-d-def*)
have *2*: $\vdash (bi\ (empty = f))^r = \Box((empty = f)^r)$ **by** (*simp add*: *RBiEqvBox*)
have *3*: $\vdash (empty = f)^r = (empty = f^r)$ **by** (*metis REmptyEqvEmpty inteq-reflection rev-fun2*)
hence *4*: $\vdash \Box\ ((empty = f)^r) = \Box(empty = f^r)$ **by** (*simp add*: *BoxEqvBox*)
have *5*: $\vdash \Box(empty = f^r) = halt(f^r)$ **by** (*simp add*: *halt-d-def*)
from *1 2 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *RRHaltEqvInitonly*:
$\vdash (halt\ (f^r))^r = initonly\ (f)$
proof −
have *1*: $\vdash (halt\ (f^r))^r = initonly\ ((f^r)^r)$ **using** *RHaltEqvInitonly* **by** *blast*
have *2*: $\vdash initonly\ ((f^r)^r) = initonly(f)$ **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RRInitonlyEqvHalt* :
$\vdash (initonly\ (f^r))^r = halt(f)$
proof −
have *1*: $\vdash (initonly\ (f^r))^r = halt((f^r)^r)$ **using** *RInitonlyEqvHalt* **by** *blast*
have *2*: $\vdash halt((f^r)^r) = halt(f)$ **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RKeepEqvKeep* :
$\vdash (keep\ f)^r = keep(f^r)$
proof −
have *1*: $\vdash (keep\ f)^r = (ba(skip \longrightarrow f))^r$ **by** (*simp add*: *keep-d-def*)
have *2*: $\vdash (ba(skip \longrightarrow f))^r = ba((skip \longrightarrow f)^r)$ **by** (*simp add*:*RBaEqvBa*)
have *3*: $\vdash (skip \longrightarrow f)^r = (skip \longrightarrow f^r)$ **by** (*metis all-rev-eq*(*12*) *rev-fun2*)
hence *4*: $\vdash ba((skip \longrightarrow f)^r) = ba(skip \longrightarrow f^r)$ **by** (*simp add*: *BaEqvBa*)
have *5*: $\vdash ba(skip \longrightarrow f^r) = keep(f^r)$ **by** (*simp add*: *keep-d-def*)
from *1 2 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *RRKeepEqvKeep* :
$\vdash (keep\ (f^r))^r = keep(f)$
proof −
have *1*: $\vdash (keep\ (f^r))^r = keep(\ (f^r)^r\ )$ **using** *RKeepEqvKeep* **by** *blast*
have *2*: $\vdash keep(\ (f^r)^r\ ) = keep(f)$ **using** *EqvReverseReverse* **by** (*metis inteq-reflection*)
from *1 2* **show** *?thesis* **by** *fastforce*

qed

lemma *NextDiamondEqvDiamondNext*:
$\vdash \bigcirc(\ \Diamond\ f) = \Diamond(\bigcirc f)$
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

lemma *RassignEqvTAssign*:
$\vdash (\$v = e)^r = (v \leftarrow e^r)$
proof −
have *1*: $\vdash (\$v = e)^r = ((\$v)^r = e^r)$ **by** (*simp add*: *rev-fun2*)
have *2*: $\vdash ((\$v)^r = e^r) = ((!v) = e^r)$ **by** (*simp add*: *all-rev-eq*(*8*))
have *3*: $\vdash ((!v) = e^r) = (v \leftarrow e^r)$ **by** (*simp add*: *intI temporal-assign-d-def*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *RTAssignEqvAssign*:
$\vdash (v \leftarrow e)^r = (\$v = e^r)$
proof −
have *1*: $\vdash (v \leftarrow e)^r = (!v = e)^r$ **by** (*simp add*: *REqvRule intI temporal-assign-d-def*)
have *2*: $\vdash (!v = e)^r = (\$v = e^r)$ **by** (*metis all-rev-eq*(*11*) *rev-fun2*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RNextAssignEqvPrevAssign*:
$\vdash (v := e)^r = (v =: e^r)$
proof −
have *1*: $\vdash (v := e)^r = (v\$ = e)^r$ **by** (*simp add*: *REqvRule intI next-assign-d-def*)
have *2*: $\vdash (v\$ = e)^r = (v! = e^r)$ **by** (*metis all-rev-eq*(*9*) *rev-fun2*)
have *3*: $\vdash (v! = e^r) = (v =: e^r)$ **by** (*simp add*: *prev-assign-d-def*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *RPrevAssignEqvNextAssign*:
$\vdash (v =: e)^r = (v := e^r)$
proof −
have *1*: $\vdash (v =: e)^r = (v! = e)^r$ **by** (*simp add*: *REqvRule intI prev-assign-d-def*)
have *2*: $\vdash (v! = e)^r = (v\$ = e^r)$ **by** (*metis all-rev-eq*(*10*) *rev-fun2*)
have *3*: $\vdash (v\$ = e^r) = (v := e^r)$ **by** (*simp add*: *next-assign-d-def*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *RGetsEqvBaSkipImp*:
$\vdash (v \ gets \ e)^r = ba(skip \longrightarrow (\$v = e^r))$
proof −
have *1*: $\vdash (v \ gets \ e)^r = (ba(skip \longrightarrow (!v = e)))^r$
using *gets-d-def temporal-assign-d-def keep-d-def REqvRule*
by (*metis Prop04 ba-d-def int-simps*(*15*))
have *2*: $\vdash (ba(skip \longrightarrow (!v = e)))^r = ba\ (\ (skip \longrightarrow (!v = e))^r\ )$
by (*simp add*: *RBaEqvBa*)
have *3*: $\vdash (skip \longrightarrow (!v = e))^r = (skip \longrightarrow (\$v = e^r))$
by (*simp add*: *all-rev-eq*(*11*) *all-rev-eq*(*12*) *all-rev-eq*(*3*))
hence *4*: $\vdash ba\ ((skip \longrightarrow (!v = e))^r) = ba\ (skip \longrightarrow (\$v = e^r))$
by (*simp add*: *BaEqvBa*)
from *1 2 4* **show** *?thesis* **by** *fastforce*
qed

lemma *RIfThenElse*:
$\vdash (if_i \ f0 \ then \ f1 \ else \ f2)^r = if_i \ (f0^r) \ then \ (f1^r) \ else \ (f2^r)$
by (*simp add*: *all-rev-eq*(*2*) *all-rev-eq*(*3*) *ifthenelse-d-def*)

lemma *RWhile*:
$\vdash (init \ f \wedge while \ f0 \ do \ f1)^r = (\ fin(f) \wedge ((f0^r) \wedge (f1^r))^\star \wedge init\ (\neg(f0))\ )$
proof −
have *1*: $\vdash (init \ f \wedge while \ f0 \ do \ f1)^r = (\ init \ f \wedge (f0 \wedge f1)^\star \wedge fin\ (\neg f0)\ )^r$
by (*simp add*: *while-d-def*)
have *2*: $\vdash (\ init \ f \wedge (f0 \wedge f1)^\star \wedge fin\ (\neg f0)\ )^r = ((init \ f)^r \wedge ((f0 \wedge f1)^\star)^r \wedge (fin\ (\neg f0))^r)$
by (*simp add*: *all-rev-eq*(*3*))
have *3*: $\vdash (init \ f)^r = fin(f)$
by (*simp add*: *RInitEqvFin*)
have *4*: $\vdash ((f0 \wedge f1)^\star)^r = ((f0^r) \wedge (f1^r))^\star$
by (*metis RevChopstar all-rev-eq*(*3*))
have *5*: $\vdash (fin\ (\neg f0))^r = init\ (\neg(f0))$

by (*metis RFinEqvInit*)
have *6*: $\vdash ((init\ f)^r \wedge ((f0 \wedge f1)^\star)^r \wedge (fin\ (\neg f0))^r) =$
$(\ fin(f) \wedge ((f0^r) \wedge (f1^r))^\star \wedge init\ (\neg(f0))\ )$ **using** *3 4 5* **by** *fastforce*
from *1 2 6* **show** *?thesis* **by** *fastforce*
qed

end

theory *FOTheorems*
imports
Theorems
begin

5 First Order ITL theorems

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

lemmas *EExI-unI* = *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 AAxDef EExE EExI*
by (*smt Valid-def exist-state-d-def intensional-rews*(*2*) *intensional-rews*(*3*))

lemma *EExEqvRule*:
assumes $\bigwedge x. \vdash F\ x = G\ x$
shows $\vdash (\exists\exists\ x.\ F\ x) = (\exists\exists\ x.\ G\ x)$
by (*metis EExE EExI assms int-iffD1 int-iffD2 int-iffI lift-imp-trans*)

lemma *AAxImpEEx*:
$\vdash (\forall\forall\ x.\ F\ x) \longrightarrow (\exists\exists\ x.\ F\ x)$
by (*simp add*: *exist-state-d-def forall-state-d-def intI*)

lemma *EExImpRule*:
assumes $\vdash F\ x \longrightarrow G\ x$
shows $\vdash (\exists\exists\ x.\ F\ x \longrightarrow\ G\ x)$
using *assms* **by** (*meson MP EExI*)

lemma *EExImpRuleDist*:

assumes $\vdash F\ x \longrightarrow G\ x$
shows $\vdash (\forall\forall\ x.\ F\ x) \longrightarrow (\exists\exists\ x.\ G\ x)$
proof −
have *1*: $\vdash (F\ x) \longrightarrow (\exists\exists\ x.\ G\ x)$ **using** *EExI assms lift-imp-trans* **by** *blast*
have *2*: $\vdash \neg(F\ x) \vee (\exists\exists\ x.\ G\ x)$ **using** *1* **by** *auto*
have *3*: $\vdash \neg(F\ x) \longrightarrow (\exists\exists\ x.\ \neg(F\ x))$ **by** (*meson EExI*)
have *4*: $\vdash (\exists\exists\ x.\ \neg(F\ x)) = (\neg(\forall\forall\ x.\ F\ x))$ **using** *AAxDef* **by** *fastforce*
from *2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *EExImpNoDepDist*:
assumes $\vdash R \longrightarrow G\ x$
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 (*simp add*: *EExI-unI intI*)
from *1* **show** *?thesis* **by** (*simp add*: *EExE*)
qed

lemma *EExOrDist*:
$\vdash ((\exists\exists\ x.\ F\ x) \vee (\exists\exists\ x.\ H\ x)) = (\exists\exists\ x.\ (F\ x) \vee (H\ x))$
using *EExOrDist-3 EExOrDist-4* **by** *fastforce*

lemma *EExOrImport-1*:

$\vdash G \longrightarrow (\exists\exists\ x.\ G \vee (F\ x))$
by (*simp add*: *EExI-unl Valid-def*)

lemma *EExOrImport-2*:
$\vdash (\exists\exists\ x.\ F\ x) \longrightarrow (\exists\exists\ x.\ G \vee (F\ x))$
by (*simp add*: *EExOrDist-1*)

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

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

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

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

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

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

lemma *EExAndDist*:
assumes $\vdash F\ x \wedge G\ x$
shows $\vdash (\exists\exists\ x.\ F\ x) \wedge (\exists\exists\ x.\ G\ x)$
proof −
have *1*: $\vdash F\ x$ **using** *assms* **by** *fastforce*

have *2*: $\vdash G\ x$ **using** *assms* **by** *fastforce*
have *3*: $\vdash (\exists\exists\ x.\ F\ x)$ **using** *1* **by** (*meson EExI MP*)
have *4*: $\vdash (\exists\exists\ x.\ G\ x)$ **using** *2* **by** (*meson EExI MP*)
from *3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *EExAndNoDepDist*:
assumes $\vdash R \wedge G\ x$
shows $\vdash R \wedge (\exists\exists\ x.\ G\ x)$
proof −
have *1*: $\vdash R$ **using** *assms* **by** *fastforce*
have *2*: $\vdash G\ x$ **using** *assms* **by** *fastforce*
have *3*: $\vdash (\exists\exists\ x.\ G\ x)$ **using** *2* **by** (*meson EExI MP*)
from *1 3* **show** *?thesis* **by** *fastforce*
qed

lemma *Spec*:
$\vdash (\forall\forall\ x.\ F\ x) \longrightarrow F\ x$
proof −
have *1*: $\vdash \neg(F\ x) \longrightarrow (\exists\exists\ x.\ \neg(F\ x))$ **by** (*meson EExI*)
have *2*: $\vdash \neg(\exists\exists\ x.\ \neg(F\ x)) \longrightarrow F\ x$ **using** *1* **by** *auto*
from *2* **show** *?thesis* **using** *AAxDef* **by** *fastforce*
qed

lemma *AAxE*:
assumes $\vdash (\forall\forall\ x.\ F\ x)$
$\vdash F\ x \longrightarrow R$
shows $\vdash R$
using *MP Spec assms*(*1*) *assms*(*2*) **by** *blast*

lemma *AAxI*:
assumes $\bigwedge x.\ \vdash F\ x$
shows $\vdash (\forall\forall\ x.\ F\ x)$
unfolding *AAxDef*
using *AAxDef EExE assms*
by (*smt Valid-def int-simps*(*15*) *unl-lift unl-lift2*)

lemma *AAxEqvRule*:
assumes $\bigwedge x.\ \vdash F\ x = G\ x$
shows $\vdash (\forall\forall\ x.\ F\ x) = (\forall\forall\ x.\ G\ x)$
by (*metis* (*mono-tags*, *lifting*) *AAxDef EExEqvRule assms int-iffD1 int-iffI*
inteq-reflection lift-imp-neg)

lemma *AAxAndDist*:
$\vdash (\forall\forall\ x.\ (F\ x) \wedge (G\ x)) = ((\forall\forall\ x.\ F\ x) \wedge (\forall\forall\ x.\ G\ x))$
proof −
have *1*: $\vdash ((\exists\exists\ x.\ \neg(F\ x)) \vee (\exists\exists\ x.\ \neg(G\ x))) = (\exists\exists\ x.\ \neg(F\ x) \vee \neg(G\ x))$ **by** (*simp add*:*EExOrDist*)
have *2*: $\vdash ((\exists\exists\ x.\ \neg(F\ x))) = (\neg(\forall\forall\ x.\ F\ x)\ )$ **using** *AAxDef* **by** *fastforce*
have *3*: $\vdash ((\exists\exists\ x.\ \neg(G\ x))) = (\neg(\forall\forall\ x.\ G\ x)\ )$ **using** *AAxDef* **by** *fastforce*

have *4*: $\vdash ((\exists\exists\ x.\ \neg(F\ x)) \vee (\exists\exists\ x.\ \neg(G\ x))) = (\neg(\forall\forall\ x.\ F\ x) \vee \neg(\forall\forall\ x.\ G\ x))$
using *2 3* **by** *fastforce*
have *5*: $\bigwedge x\ . \vdash (\neg(F\ x) \vee \neg(G\ x)) = (\neg((F\ x) \wedge (G\ x)))$ **by** *auto*
have *6*: $\vdash (\exists\exists\ x.\ \neg(F\ x) \vee \neg(G\ x)) = (\exists\exists\ x.\ \neg((F\ x) \wedge (G\ x)))$ **using** *5* **by** (*simp add*: *EExEqvRule*)
have *7*: $\vdash (\exists\exists\ x.\ \neg((F\ x) \wedge (G\ x))) = (\neg(\forall\forall\ x.\ (F\ x) \wedge (G\ x)))$ **using** *AAxDef* **by** *fastforce*
have *8*: $\vdash (\neg(\forall\forall\ x.\ F\ x) \vee \neg(\forall\forall\ x.\ G\ x)\ ) = (\neg(\ (\forall\forall\ x.\ F\ x) \wedge (\forall\forall\ x.\ G\ x)))$ **by** *fastforce*
have *9*: $\vdash (\neg(\ (\forall\forall\ x.\ F\ x) \wedge (\forall\forall\ x.\ G\ x))) = (\neg(\forall\forall\ x.\ (F\ x) \wedge (G\ x)))$
using *1 4 6 7 8* **by** *fastforce*
from *9* **show** *?thesis* **by** *fastforce*
qed

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

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

lemma *EExImpChopRule*:
assumes $\vdash F\ x \longrightarrow G\ x$
shows $\vdash (\exists\exists\ x.\ H;(F\ x) \longrightarrow H;(G\ x))$
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

6 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.

6.1 Definitions

6.1.1 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*

6.1.2 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*

6.2 First and Time Reversal

lemma *BsEqvRule*:
assumes ⊢ *f* = *g*
shows ⊢ *bs f* = *bs g*
proof −
have *1*: ⊢ *f* = *g* **using** *assms* **by** *auto*
hence *2*: ⊢ *bi*(*f*) = *bi*(*g*) **by** (*simp add*: *BiEqvBi*)
hence *3*: ⊢ *bi*(*f*);*skip* = *bi*(*g*);*skip* **by** (*simp add*: *LeftChopEqvChop*)
hence *4*: ⊢ (*empty* ∨ *bi*(*f*);*skip*) = (*empty* ∨ *bi*(*g*);*skip*) **by** *auto*
hence *5*: ⊢ *bs*(*f*) = *bs*(*g*) **by** (*simp add*: *bs-d-def*)
from *1 2 3 4 5* **show** *?thesis* **by** *auto*
qed

lemma *BtEqvRule*:
assumes ⊢ *f* = *g*
shows ⊢ *bt f* = *bt g*
proof −
have *1*: ⊢ *f* = *g* **using** *assms* **by** *auto*
hence *2*: ⊢ □(*f*) = □(*g*) **by** (*simp add*: *BoxEqvBox*)
hence *3*: ⊢ *skip*;□(*f*) = *skip*;□(*g*) **using** *RightChopEqvChop* **by** *blast*
hence *4*: ⊢ (*empty* ∨ *skip*;□(*f*)) = (*empty* ∨ *skip*;□(*g*)) **by** *auto*
hence *5*: ⊢ *bt*(*f*) = *bt*(*g*) **by** (*simp add*: *bt-d-def*)
from *1 2 3 4 5* **show** *?thesis* **by** *auto*
qed

lemma *FstEqvRule*:
assumes ⊢ *f* = *g*
shows ⊢ ▷*f* = ▷*g*
proof −
have *1*: ⊢ *f* = *g* **using** *assms* **by** *auto*
hence *2*: ⊢ (¬ *f*) = (¬ *g*) **by** *auto*
hence *3*: ⊢ *bs*(¬ *f*) = *bs*(¬ *g*) **by** (*simp add*: *BsEqvRule*)
hence *4*: ⊢ (*f* ∧ *bs*(¬ *f*)) = (*g* ∧ *bs*(¬ *g*)) **using** *1* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*:*first-d-def*)
qed

lemma *LstEqvRule*:

assumes $\vdash f = g$
shows $\vdash \lhd f = \lhd g$
proof −
have *1*: $\vdash f = g$ **using** *assms* **by** *auto*
hence *2*: $\vdash (\neg f) = (\neg g)$ **by** *auto*
hence *3*: $\vdash bt(\neg f) = bt(\neg g)$ **by** (*simp add*: *BtEqvRule*)
hence *4*: $\vdash (f \wedge bt(\neg f)) = (g \wedge bt(\neg g))$ **using** *1* **by** *fastforce*
from *4* **show** *?thesis* **by** (*simp add*:*last-d-def*)
qed

lemma *RBsEqvBt*:
$\vdash (bs\ f)^r = (bt\ (f^r))$
proof −
have *1*: $\vdash (bs\ f)^r = (empty \vee ((bi\ f)\ ;\ skip))^r$
by (*simp add*: *bs-d-def*)
have *2*: $\vdash (empty \vee ((bi\ f)\ ;\ skip))^r = (empty^r \vee ((bi\ f)\ ;\ skip)^r)$
using *ROr* **by** *blast*
have *3*: $\vdash (empty^r \vee ((bi\ f)\ ;\ skip)^r) = (empty \vee (skip^r;(bi\ f)^r))$
using *REmptyEqvEmpty RevChop* **by** *fastforce*
have *4*: $\vdash (empty \vee (skip^r;(bi\ f)^r)) = (empty \vee (skip;\Box\ (f^r)))$
by (*metis 3 RBiEqvBox RevSkip int-eq*)
have *5*: $\vdash (empty \vee (skip;\Box\ (f^r))) = (bt\ (f^r))$
by (*simp add*: *bt-d-def*)
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *RRBsEqvBt*:
$\vdash (bs\ (f^r))^r = (bt\ (f))$
proof −
have *1*: $\vdash (bs\ (f^r))^r = bt\ ((f^r)^r)$ **using** *RBsEqvBt* **by** *blast*
have *2*: $\vdash bt\ ((f^r)^r) = bt\ f$ **using** *EqvReverseReverse* **using** *BtEqvRule* **by** *blast*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RBtEqvBs*:
$\vdash (bt\ f)^r = (bs\ (f^r))$
proof −
have *1*: $\vdash (bt\ f)^r = (empty \vee (skip;\Box\ f))^r$
by (*simp add*: *bt-d-def*)
have *2*: $\vdash (empty \vee (skip;\Box\ f))^r = (empty^r \vee (skip;\Box\ f)^r)$
using *ROr* **by** *blast*
have *3*: $\vdash (empty^r \vee (skip;\Box\ f)^r) = (empty \vee (\Box\ f)^r;skip^r)$
using *REmptyEqvEmpty RevChop* **by** *fastforce*
have *4*: $\vdash (empty \vee (\Box\ f)^r;skip^r) = (empty \vee (bi\ (f^r));skip)$
by (*metis 3 RBoxEqvBi RevSkip int-eq*)
have *5*: $\vdash (empty \vee (bi\ (f^r));skip) = (bs\ (f^r))$
by (*simp add*: *bs-d-def*)
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *RRBtEqvBs*:
$\vdash (bt\ (f^r))^r = (bs\ (f))$
proof −
have *1*: $\vdash (bt\ (f^r))^r = bs\ ((f^r)^r)$ **using** *RBtEqvBs* **by** *blast*
have *2*: $\vdash bs\ ((f^r)^r) = bs\ f$ **using** *EqvReverseReverse* **using** *BsEqvRule* **by** *blast*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *RFirstEqvLast*:
$\vdash (\triangleright\ f)^r = (\triangleleft\ (f^r))$
proof −
have *1*: $\vdash (\triangleright\ f)^r = (f \wedge bs(\neg\ f))^r$ **by** (*simp add*: *first-d-def*)
have *2*: $\vdash (f \wedge bs(\neg\ f))^r = (f^r \wedge (bs\ (\neg\ f))^r)$ **using** *RAnd* **by** *blast*
have *3*: $\vdash (f^r \wedge (bs\ (\neg\ f))^r) = (f^r \wedge bt\ ((\neg\ f)^r))$ **using** *RBsEqvBt* **by** *fastforce*
have *4*: $\vdash (f^r \wedge bt\ ((\neg\ f)^r)) = (f^r \wedge bt\ (\neg(f^r)))$ **using** *RNot int-eq* **by** *fastforce*
have *5*: $\vdash (f^r \wedge bt\ (\neg(f^r))) = (\triangleleft\ (f^r))$ **by** (*simp add*: *last-d-def*)
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

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

lemma *RLastEqvFirst*:
$\vdash (\triangleleft\ f)^r = (\triangleright\ (f^r))$
proof −
have *1*: $\vdash (\triangleleft\ f)^r = (f \wedge bt(\neg\ f))^r$ **by** (*simp add*: *last-d-def*)
have *2*: $\vdash (f \wedge bt(\neg\ f))^r = (f^r \wedge (bt\ (\neg f))^r)$ **using** *RAnd* **by** *blast*
have *3*: $\vdash (f^r \wedge (bt\ (\neg f))^r) = (f^r \wedge bs\ (\ (\neg f)^r))$ **using** *RBtEqvBs* **by** *fastforce*
have *4*: $\vdash (f^r \wedge bs\ (\ (\neg f)^r)) = (f^r \wedge bs(\neg(f^r)))$ **using** *RNot int-eq* **by** *fastforce*
have *5*: $\vdash (f^r \wedge bs(\neg(f^r))) = (\triangleright\ (f^r))$ **by** (*simp add*: *first-d-def*)
from *1 2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

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

6.3 Semantic Theorems

6.3.1 Semantics First and Last Operators

lemma *FstAndBisem*:

```
 (intlen σ >0 ∧ (σ ⊨ f) ∧ ( σ ⊨  bi (¬f);skip)) =
  (intlen σ >0 ∧ (σ ⊨ f) ∧ (∀ ia<intlen (σ). (prefix ia  σ ⊨ ¬f)) )
apply (simp add: chop-defs bi-defs skip-defs)
apply (simp add: interval-prefix-length interval-suffix-length)
proof −
 have 1: (0 < intlen σ ∧ (σ ⊨ f) ∧
         (∃ i. (i ≤ intlen σ ⟶ (∀ ia≤i. ¬ (prefix ia (prefix i σ) ⊨ f)) ∧
          intlen σ − i = Suc 0) ∧ i ≤ intlen σ)
         ) =
         (0 < intlen σ ∧ (σ ⊨ f) ∧
          (∃ i. (i ≤ intlen σ ⟶ (∀ ia≤i. ¬ (prefix ia (prefix i σ) ⊨ f)) ∧
          i = intlen σ − Suc 0) ∧ i ≤ intlen σ)
         )
     by auto
 also have ... =
         (0 < intlen σ ∧ (σ ⊨ f) ∧
          (∀ ia≤(intlen σ − Suc 0). ¬ (prefix ia (prefix (intlen σ − Suc 0) σ) ⊨ f) )
         )
     using diff-le-self by blast
 also have ... =
         (intlen σ >0 ∧ (σ ⊨ f) ∧
           (∀ ia<intlen ( σ). ¬ (prefix ia (prefix (intlen σ − Suc 0) σ) ⊨ f))
         )  by (metis Suc-pred less-Suc-eq-le)
 also have ... =
         (intlen σ >0 ∧ (σ ⊨ f) ∧
          (∀ ia<intlen ( σ). (prefix ia (prefix (intlen σ − Suc 0) σ) ⊨ ¬f))
         )
     by auto
 also have ... =
         (intlen σ >0 ∧ (σ ⊨ f) ∧ (∀ ia<intlen ( σ). ¬ (prefix ia  σ ⊨ f)))
         by (simp add: interval-pref-pref-help)
 finally show (0 < intlen σ ∧ (σ ⊨ f) ∧
              (∃ i. (i ≤ intlen σ ⟶ (∀ ia≤i. ¬ (prefix ia (prefix i σ) ⊨ f)) ∧
                           intlen σ − i = Suc 0) ∧ i ≤ intlen σ)
             ) =
             (0 < intlen σ ∧ (σ ⊨ f) ∧ (∀ ia<intlen σ. ¬ (prefix ia σ ⊨ f))) .
qed

lemma Fstsem-0:
 (σ ⊨ ▷f) =
  (
   ( σ ⊨ f ∧ empty) ∨ (intlen σ >0  ∧ (σ ⊨ f) ∧ ( σ ⊨  bi (¬f);skip))
  )
apply (simp add: first-d-def bs-d-def ) using empty-defs by auto

lemma Emptysem:
 (σ ⊨ f ∧ empty) = ((σ ⊨ f) ∧ intlen σ = 0)
using empty-defs by auto

lemma Fstsem:
```

$(\sigma \models \rhd f) =$
$($
$(\ (\sigma \models f) \wedge \mathit{intlen}\ \sigma = 0) \vee$
$(\ \mathit{intlen}\ \sigma > 0 \wedge (\sigma \models f) \wedge (\forall\, \mathit{ia} < \mathit{intlen}\ (\sigma).\ (\mathit{prefix}\ \mathit{ia}\ \ \sigma \models \neg f)))$
$)$
using *Fstsem-0 Emptysem FstAndBisem* **by** *metis*

lemma *Lstsem*:
$(\sigma \models \lhd f) =$
$(\ (\ (\sigma \models f) \wedge \mathit{intlen}\ \sigma = 0) \vee$
$(\ \mathit{intlen}\ \sigma > 0 \wedge (\sigma \models f) \wedge (\forall\, \mathit{ia} < \mathit{intlen}\ \sigma.\ (\mathit{suffix}\ ((\mathit{intlen}\ \sigma) - \mathit{ia})\ \ \sigma \models \neg f))\ )$
$)$

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

```
    (∀ ia < intlen σ. (suffix ((intlen σ) −ia)  σ ⊨ ¬f)) )
  )  .
qed
```

6.3.2 Various Semantic Lemmas

```
lemma DiLensem:
 (σ ⊨ di (f ∧ len(i))) =
   ( (prefix i σ ⊨ f) ∧ i≤intlen σ)
apply (simp add: di-defs len-defs)
using interval-prefix-length-good by auto

lemma PrefixFstsem:
 ( (prefix i σ ⊨ ▷f) ∧ i≤intlen σ) =
  ( i≤intlen σ ∧
   (
    ( (prefix i σ ⊨ f) ∧ i = 0) ∨
    ( i>0 ∧ (prefix i σ ⊨ f) ∧ (∀ ia<i. (prefix ia σ ⊨ ¬f)))
   )
 )
proof −
 have 1: ( ((prefix i σ) ⊨ ▷f)) =
  (
   ( ((prefix i σ) ⊨ f) ∧ intlen (prefix i σ) = 0) ∨
   ( intlen (prefix i σ)>0 ∧ ((prefix i σ) ⊨ f) ∧
     (∀ ia<intlen (prefix i σ). (prefix ia  (prefix i σ) ⊨ ¬f)) )
  )
      using Fstsem by blast
 hence 2: ( ((prefix i σ) ⊨ ▷f) ∧ i≤intlen σ) =
  (  i≤intlen σ ∧ (
   ( ((prefix i σ) ⊨ f) ∧ intlen (prefix i σ) = 0) ∨
   ( intlen (prefix i σ)>0 ∧ ((prefix i σ) ⊨ f) ∧
     (∀ ia<intlen (prefix i σ). (prefix ia  (prefix i σ) ⊨ ¬f)) )
    )
  )
      by auto
 hence 3: ( ((prefix i σ) ⊨ ▷f) ∧ i≤intlen σ) =
  (  i≤intlen σ ∧ (
   ( ((prefix i σ) ⊨ f) ∧ i = 0) ∨
   ( i>0 ∧ ((prefix i σ) ⊨ f) ∧ (∀ ia<i. (prefix ia  (prefix i σ) ⊨ ¬f)))
    )
  )
      by auto
 hence 4: ( ((prefix i σ) ⊨ ▷f) ∧ i≤intlen σ) =
  (  i≤intlen σ ∧ (
   ( ((prefix i σ) ⊨ f) ∧ i = 0) ∨
   ( i>0 ∧ ((prefix i σ) ⊨ f) ∧ (∀ ia<i. (prefix ia σ ⊨ ¬f)))
    )
  )
     using interval-pref-pref-3 using less-imp-add-positive by fastforce
```

from *4* **show** *?thesis* **by** *auto*
qed

lemma *PrefixFstAndsem*:
((*prefix i σ* ⊨ ▷*f* ∧ *g*) ∧ *i*≤*intlen σ*) =
(*i*≤*intlen σ* ∧
(
((*prefix i σ* ⊨ *f* ∧ *g*) ∧ *i* = *0*) ∨
(*i*>*0* ∧ (*prefix i σ* ⊨ *f* ∧ *g*) ∧ (∀ *ia*<*i*. (*prefix ia σ* ⊨ ¬*f*)))
)
)
using *PrefixFstsem* **by** (*metis unl-lift2*)

lemma *DiLenFstsem*:
(*σ* ⊨ *di* (▷*f* ∧ *len*(*i*))) =
(*i*≤*intlen σ* ∧
(
((*prefix i σ* ⊨ *f*) ∧ *i* = *0*) ∨
(*i*>*0* ∧ (*prefix i σ* ⊨ *f*) ∧ (∀ *ia*<*i*. (*prefix ia σ* ⊨ ¬*f*)))
)
)
by (*simp add*: *DiLensem PrefixFstsem*)

lemma *DiLenFstAndsem*:
(*σ* ⊨ *di* ((▷*f* ∧ *g*) ∧ *len*(*i*))) =
(*i*≤*intlen σ* ∧
(
((*prefix i σ* ⊨ *f* ∧ *g*) ∧ *i* = *0*) ∨
(*i*>*0* ∧ (*prefix i σ* ⊨ *f* ∧ *g*) ∧ (∀ *ia*<*i*. (*prefix ia σ* ⊨ ¬*f*)))
)
)
using *DiLensem PrefixFstAndsem* **by** *metis*

lemma *FstLenSamesem*:
((*i*≤*intlen σ* ∧
(
((*prefix i σ* ⊨ *f*) ∧ *i* = *0*) ∨
(*i*>*0* ∧ (*prefix i σ* ⊨ *f*) ∧ (∀ *ia*<*i*. (*prefix ia σ* ⊨ ¬*f*)))
)
) ∧
(*j*≤*intlen σ* ∧
(
((*prefix j σ* ⊨ *f*) ∧ *j* = *0*) ∨
(*j*>*0* ∧ (*prefix j σ* ⊨ *f*) ∧ (∀ *ia*<*j*. (*prefix ia σ* ⊨ ¬*f*)))
)
)
) ⟶ (*i*=*j*)

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

6.4 Theorems

6.4.1 Fixed length intervals

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

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

lemma *LenNPlusOneA*:
$\vdash$ *len*(*n*+*1*) = *skip*;*len*(*n*)
by *simp*

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

lemma *LenCommute*:
$\vdash$ (*skip*;(*len n*)) = (*len n*);*skip*
proof
(*induct n*)
case *0*
then show *?case* **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 *SkipTrueEqvTrueSkip*:
$\vdash$ *skip*;#*True* = #*True*;*skip*

using *TrueChopSkipEqvSkipChopTrue* **by** *fastforce*

lemma *PowerCommute*:
$\vdash (f;(power\ f\ n)) = ((power\ f\ n);f)$
proof
(*induct n*)
case *0*
then show *?case* **using** *EmptyChop ChopEmpty pow-0* **by** (*metis int-eq*)
next
case (*Suc n*)
then show *?case* **using** *ChopAssoc pow-Suc* **by** (*metis inteq-reflection*)
qed

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

lemma *RLenEqvLen*:
$\vdash (len\ k)^r = (len\ k)$
proof
(*induct k*)
case *0*
then show *?case* **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 LenEqvLenChopLen Suc-eq-plus1 int-eq 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 *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 *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 *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 *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 *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*)

lemma *NotChopNotSkip*:
$\vdash (\neg(f;skip)) = (empty \vee ((\neg\ f);skip))$
proof −
have *1*: $\vdash (\neg((\neg(\neg f));skip)) = (empty \vee ((\neg\ f);skip))$ **using** *NotNotChopSkip* **by** *blast*
have *2*: $\vdash (\neg((\neg(\neg f));skip)) = (\neg(f;skip))$ **by** *auto*
from *1 2* **show** *?thesis* **by** *auto*
qed

6.4.2 Additional ITL theorems

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

lemma *MoreAndBiImpBiChopSkip*:
$\vdash more \wedge bi\ f \longrightarrow (bi\ f);skip$
proof −
have *1*: $\vdash\ (bi\ f);skip = ((\neg(di\ (\neg\ f)));skip)$ **by** (*simp add*: *bi-d-def*)
have *2*: $\vdash (\neg(\ (\neg(di\ (\neg\ f)));skip)) = (empty \vee (di\ (\neg\ f));skip)$ **by** (*rule NotNotChopSkip*)
have *3*: $\vdash empty \longrightarrow empty \vee di\ (\neg\ f)$ **by** *auto*
have *4*: $\vdash (di\ (\neg\ f));skip \longrightarrow di\ (\neg\ f)$ **using** *ChopImpDi DiEqvDiDi* **by** *fastforce*
hence *5*: $\vdash (di\ (\neg\ f));skip \longrightarrow empty \vee di\ (\neg\ f)$ **by** (*rule Prop05*)
have *6*: $\vdash \neg(\ (\neg(di\ (\neg\ f)));skip) \longrightarrow empty \vee di\ (\neg\ f)$ **using** *2 3 5* **by** *fastforce*
hence *7*: $\vdash \neg(empty \vee di\ (\neg\ f)) \longrightarrow \neg(\neg(\ (\neg(di\ (\neg\ f)));skip))$ **by** *fastforce*
have *8*: $\vdash (\neg(\ \neg(\ (\neg(di\ (\neg\ f)));skip))) = ((\neg(di\ (\neg\ f)));skip)$ **by** *auto*
have *9*: $\vdash (\neg(empty \vee di\ (\neg\ f))) = (more \wedge \neg(\ di\ (\neg\ f)))$
using *NotAndMoreEqvEmptyOr* **by** *fastforce*
have *10*: $\vdash (more \wedge \neg(\ di\ (\neg\ f))) = (more \wedge bi\ f)$ **by** (*simp add*: *bi-d-def*)
from *1 6 7 8 9 10* **show** *?thesis* **by** (*metis int-eq*)
qed

lemma *DiChopImpDiB*:
$\vdash di(f;g) \longrightarrow di\ f$
proof −
have *1*: $\vdash f\ ;\ (g;\#True) \longrightarrow di\ f$ **by** (*rule ChopImpDi*)

have *2*: $\vdash f\ ;\ (g;\#True) = (f;g);\#True$ **by** (*rule ChopAssoc*)
from *1 2* **show** *?thesis* **by** (*metis di-d-def int-eq*)
qed

lemma *BiBiOrImpBi*:
$\vdash bi\ (\ bi\ f \vee bi\ g) \longrightarrow bi\ f \vee bi\ g$
using *BiElim* **by** *auto*

lemma *BiImpBiBiOr*:
$\vdash bi\ f \longrightarrow bi\ (\ bi\ f \vee bi\ g)$
proof −
have *1*: $\vdash bi\ f \longrightarrow bi\ f \vee bi\ g$ **by** *auto*
hence *2*: $\vdash bi\ (bi\ f) \longrightarrow bi(bi\ f \vee bi\ g)$ **using** *BiImpBiRule* **by** *blast*
have *3*: $\vdash bi\ (bi\ f) = bi\ f$ **using** *BiEqvBiBi* **by** *fastforce*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BiImpBiBiOrB*:
$\vdash bi\ g \longrightarrow bi\ (\ bi\ f \vee bi\ g)$
proof −
have *1*: $\vdash bi\ g \longrightarrow bi\ f \vee bi\ g$ **by** *auto*
hence *2*: $\vdash bi\ (bi\ g) \longrightarrow bi(bi\ f \vee bi\ g)$ **using** *BiImpBiRule* **by** *blast*
have *3*: $\vdash bi\ (bi\ g) = bi\ g$ **using** *BiEqvBiBi* **by** *fastforce*
from *2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *BiBiOrEqvBi*:
$\vdash bi\ (\ bi\ f \vee bi\ g) = bi\ f \vee bi\ g$
proof −
have *1*: $\vdash bi\ (\ bi\ f \vee bi\ g) \longrightarrow bi\ f \vee bi\ g$ **by** (*rule BiBiOrImpBi*)
have *2*: $\vdash bi\ f \longrightarrow bi\ (\ bi\ f \vee bi\ g)$ **by** (*rule BiImpBiBiOr*)
have *3*: $\vdash bi\ g \longrightarrow bi\ (\ bi\ f \vee bi\ g)$ **by** (*rule BiImpBiBiOrB*)
have *4*: $\vdash bi\ f \vee bi\ g \longrightarrow bi\ (\ bi\ f \vee bi\ g)$ **using** *2 3* **by** *fastforce*
from *1 4* **show** *?thesis* **by** *fastforce*
qed

lemma *DiEqvOrDiChopSkipA*:
$\vdash di\ f = (f \vee di(f;skip))$
proof −
have *1*: $\vdash di\ f = f\ ;\#True$ **by** (*simp add*: *di-d-def*)
hence *2*: $\vdash di\ f = f;\ (\ empty \vee more)$ **by** (*simp add*: *empty-d-def*)
hence *3*: $\vdash f;\ (\ empty \vee more) = (f;empty \vee f;more)$ **using** *ChopOrEqv* **by** *blast*
have *4*: $\vdash f;empty = f$ **by** (*rule ChopEmpty*)
have *5*: $\vdash more = skip;\#True$ **using** *MoreEqvSkipChopTrue* **by** *blast*
hence *6*: $\vdash f;more = f;(skip;\#True)$ **using** *RightChopEqvChop* **by** *blast*
have *7*: $\vdash f;(skip;\#True) = (f;skip);\#True$ **by** (*rule ChopAssoc*)
from *2 3 4 6 7* **show** *?thesis* **by** (*metis di-d-def int-eq*)
qed

lemma *DiEqvOrDiChopSkipB*:

$\vdash$ *di f* = (*f* $\vee$ (*di f*);*skip*)
proof −
have *1*: $\vdash$ (*di f*) = (*f* $\vee$ *di*(*f*;*skip*)) **by** (*rule DiEqvOrDiChopSkipA*)
have *2*: $\vdash$ *di*(*f*;*skip*) = (*f*;*skip*);#*True* **by** (*simp add*: *di-d-def*)
have *3*: $\vdash$ (*f*;*skip*);#*True* = *f*;(*skip*;#*True*) **by** (*rule ChopAssocB*)
have *4*: $\vdash$ *di*(*f*;*skip*) = *f*;(*skip*;#*True*) **using** *2 3* **by** *fastforce*
have *5*: $\vdash$ *skip*;#*True* = #*True*;*skip* **by** (*rule SkipTrueEqvTrueSkip*)
hence *6*: $\vdash$ *f*;(*skip*;#*True*) = *f*;(#*True*;*skip*) **using** *RightChopEqvChop* **by** *blast*
have *7*: $\vdash$ *di*(*f*;*skip*) = *f*;(#*True*;*skip*) **using** *4 6* **by** *fastforce*
have *8*: $\vdash$ *f*;(#*True*;*skip*) = (*f*;#*True*);*skip* **by** (*rule ChopAssoc*)
have *9*: $\vdash$ (*f*;#*True*);*skip* = (*di f*);*skip* **by** (*simp add*: *di-d-def*)
have *10* : $\vdash$ *di*(*f*;*skip*) =(*di f*);*skip* **using** *7 8 9* **by** *fastforce*
hence *11*: $\vdash$ (*f* $\vee$ *di*(*f*;*skip*)) = (*f* $\vee$ (*di f*);*skip*) **by** *auto*
from *1 11* **show** *?thesis* **by** *fastforce*
qed

lemma *BiEqvAndEmptyOrBiChopSkip*:
$\vdash$ *bi f* = (*f* $\wedge$ (*empty* $\vee$ (*bi f*);*skip*))
proof −
have *1*: $\vdash$ *di* (¬ *f*) = (¬ *f* $\vee$ (*di* (¬ *f*);*skip*)) **by** (*rule DiEqvOrDiChopSkipB*)
have *2*: $\vdash$ *di* (¬ *f*) = (¬(*bi f*)) **by** (*rule DiNotEqvNotBi*)
have *3*: $\vdash$ (¬(*bi f*)) = (¬ *f* $\vee$ (*di* (¬ *f*);*skip*)) **using** *1 2* **by** *fastforce*
hence *4*: $\vdash$ *bi f* = (¬(¬ *f* $\vee$ (*di* (¬ *f*);*skip*))) **by** *auto*
have *5*: $\vdash$ (¬(¬ *f* $\vee$ (*di* (¬ *f*);*skip*))) = (*f* $\wedge$ ¬(*di* (¬ *f*);*skip*)) **by** *auto*
have *6*: $\vdash$ *di* (¬ *f*);*skip* = ((¬(*bi f*));*skip*) **by** (*simp add*: *2 LeftChopEqvChop*)
hence *7*: $\vdash$ (¬(*di* (¬ *f*);*skip*)) = (¬((¬(*bi f*));*skip*)) **by** *auto*
have *8*: $\vdash$ (¬((¬(*bi f*));*skip*)) = (*empty* $\vee$ (*bi f*);*skip*) **using** *NotNotChopSkip* **by** *blast*
hence *9*: $\vdash$ (*f* $\wedge$ ¬(*di* (¬ *f*);*skip*)) = (*f* $\wedge$ (*empty* $\vee$ (*bi f*);*skip*)) **using** *7 8* **by** *fastforce*
from *4 5 9* **show** *?thesis* **by** *fastforce*
qed

lemma *DiDiAndEqvDi*:
$\vdash$ *di* (*di f* $\wedge$ *di g*) = (*di f* $\wedge$ *di g*)
proof −
have *1*: $\vdash$ *bi* (*bi* (¬ *f*) $\vee$ *bi* (¬ *g*)) = (*bi* (¬*f*) $\vee$ *bi* (¬ *g*))
by (*meson BiBiOrImpBi BiImpBiBiOr BiImpBiBiOrB Prop02 int-iffI*)
have *2*: $\vdash$ *bi* (¬*f*) = (¬ (*di f*))
by (*simp add*: *bi-d-def*)
have *3*: $\vdash$ *bi* (¬*g*) = (¬ (*di g*))
by (*simp add*: *bi-d-def*)
have *4*: $\vdash$ (*bi* (¬*f*) $\vee$ *bi* (¬ *g*)) = (¬ (*di f*) $\vee$ ¬ (*di g*))
using *2 3* **by** *fastforce*
have *5*: $\vdash$ (¬ (*di f*) $\vee$ ¬ (*di g*)) = (¬(*di f* $\wedge$ *di g*))
by *auto*
have *6*: $\vdash$ *bi* (*bi* (¬ *f*) $\vee$ *bi* (¬ *g*)) = (¬(*di f* $\wedge$ *di g*))
using *1 5 4* **by** *fastforce*
hence *7*: $\vdash$ (¬(*bi* (*bi* (¬ *f*) $\vee$ *bi* (¬ *g*)))) = (*di f* $\wedge$ *di g*)
by *auto*
have *8* : $\vdash$ (¬(*bi* (*bi* (¬ *f*) $\vee$ *bi* (¬ *g*)))) = *di* (¬(*bi* (¬ *f*) $\vee$ *bi* (¬ *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* $\vee$ *f*;*more*)
proof −
have *1*: $\vdash$ *di f* = *f*;#*True* **by** (*simp add*: *di-d-def*)
hence *2*: $\vdash$ *di f* = *f*; (*empty* $\vee$ *more*) **by** (*simp add*: *empty-d-def*)
have *3*: $\vdash$ *f*; (*empty* $\vee$ *more*) = (*f*;*empty* $\vee$ *f*;*more*) **by** (*simp add*: *ChopOrEqv*)
have *4*: $\vdash$ *f*;*empty* = *f* **by** (*rule ChopEmpty*)
from *2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *DiAndDiEqvDiAndDiOrDiAndDi*:
$\vdash$ (*di f* $\wedge$ *di g*) = (*di*(*f* $\wedge$ *di g*) $\vee$ *di*(*g* $\wedge$ *di f*))
proof −
have *1*: $\vdash$ *di f* = (*f* $\vee$ *f*;*more*)
using *DiEqvOrChopMore* **by** *blast*
have *2*: $\vdash$ *di g* = (*g* $\vee$ *g*;*more*)
using *DiEqvOrChopMore* **by** *blast*
have *3*: $\vdash$ (*di f* $\wedge$ *di g*) = ((*f* $\vee$ *f*;*more*) $\wedge$ (*g* $\vee$ *g*;*more*))
using *1 2* **by** *fastforce*
have *4*: $\vdash$ ((*f* $\vee$ *f*;*more*) $\wedge$ (*g* $\vee$ *g*;*more*)) =
((*f* $\wedge$ *g*) $\vee$ (*f* $\wedge$ *g*;*more*) $\vee$ (*g* $\wedge$ *f*;*more*) $\vee$ (*f*;*more* $\wedge$ *g*;*more*))
by *auto*
have *5*: $\vdash$ *more* = #*True*;*skip*
using *MoreEqvSkipChopTrue SkipTrueEqvTrueSkip* **by** *fastforce*
hence *6*: $\vdash$ *f*;*more* = *f*;(#*True*;*skip*)
using *RightChopEqvChop* **by** *blast*
have *7*: $\vdash$ *f*;(#*True*;*skip*) = (*f*;#*True*);*skip*
by (*rule ChopAssoc*)
have *8*: $\vdash$ *f*;*more* = *prev* (*di f*)
using *6 7* **by** (*metis di-d-def int-eq prev-d-def*)
have *9*: $\vdash$ *g*;*more* = *g*;(#*True*;*skip*)
using *5 RightChopEqvChop* **by** *blast*
have *10*: $\vdash$ *g*;(#*True*;*skip*) = (*g*;#*True*);*skip*

by (*rule ChopAssoc*)
have *11*: $\vdash g;more = prev\ (di\ g)$
using *9 10* **by** (*metis di-d-def int-eq prev-d-def*)
have *12*: $\vdash (f;more \wedge g;more) = (prev\ (di\ f) \wedge prev\ (di\ g))$
using *8 11* **by** *fastforce*
hence *13*: $\vdash (f;more \wedge g;more) = prev\ (di\ f \wedge di\ g)$
by (*metis ChopSkipAndChopSkip int-eq prev-d-def*)
have *14*: $\vdash (di\ f \wedge di\ g) =$
$((f \wedge g) \vee (\ f \wedge g;more) \vee (g \wedge f;more)) \vee (f;more \wedge g;more)$
using *3 4* **by** *auto*
have *15*: $\vdash (di\ f \wedge di\ g) =$
$((f \wedge g) \vee (\ f \wedge g;more) \vee (g \wedge f;more)) \vee prev\ (di\ f \wedge di\ g)$
using *13 14* **by** *fastforce*
hence *16*: $\vdash (di\ f \wedge di\ g) \longrightarrow$
$((f \wedge g) \vee (\ f \wedge g;more) \vee (g \wedge f;more)) \vee prev\ (di\ f \wedge di\ g)$
by *fastforce*
hence *17*: $\vdash (di\ f \wedge di\ g) \wedge \neg((f \wedge g) \vee (\ f \wedge g;more) \vee (g \wedge f;more)) \longrightarrow$
$prev\ (di\ f \wedge di\ g)$
by *fastforce*
hence *18*: $\vdash (di\ f \wedge di\ g) \longrightarrow di((f \wedge g) \vee (\ f \wedge g;more) \vee (g \wedge f;more))$
using *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*: $\vdash di(f \wedge g;more) \longrightarrow \quad di(f \wedge di\ g) \vee di(g \wedge di\ f)$
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: ⊢ di((f ∧ g) ∨ ( f ∧ g;more) ∨ (g ∧ f;more)) ⟶
           di(f ∧ di g) ∨  di(g ∧ di f)
    using 19 32 by fastforce
 have  34: ⊢ (di f ∧ di g) ⟶ di(f ∧ di g) ∨  di(g ∧ di f)
    using 18 33 by fastforce
 have  35: ⊢  f  ⟶ di f
    using DiIntro by blast
 hence 36: ⊢ f ∧ di g ⟶ di f ∧ di g
    by auto
 hence 37: ⊢ di (f ∧ di g) ⟶ di (di f ∧ di g)
    using DiImpDi by blast
 have  38: ⊢ di (di f ∧ di g) = (di f ∧ di g)
    using DiDiAndEqvDi by blast
 have  39: ⊢ di (f ∧ di g) ⟶ di f ∧ di g
    using 37 38 by fastforce
 have  40: ⊢  g  ⟶ di g
    using DiIntro by blast
 hence 41: ⊢   g ∧ di f ⟶ di f ∧ di g
    by auto
 hence 42: ⊢ di (g ∧ di f) ⟶ di (di f ∧ di g)
    using DiImpDi by blast
 have  43: ⊢ di (di f ∧ di g) = (di f ∧ di g)
    using DiDiAndEqvDi by fastforce
 have  44: ⊢ di (g ∧ di f) ⟶ di f ∧ di g
    using 42 43 by fastforce
 have  45: ⊢ di (f ∧ di g) ∨ di (g ∧ di f) ⟶ di f ∧ di g
    using 39 44  by fastforce
 from 34 45 show ?thesis by fastforce
qed

lemma BoxStateEqvBiFinState:
 ⊢ □ (init w) = bi (fin (init w))
proof −
  have 1: ⊢ ◇ (¬ (init w)) = #True ; (¬(init w))
     by (simp add: sometimes-d-def)
  have 2: ⊢ ◇ (init(¬ w)) = #True ; init (¬ w)
     by (simp add: sometimes-d-def)
  have 3: ⊢ di (#True ∧ fin (init (¬ w))) = #True ; init (¬ w)
     using DiAndFinEqvChopState by blast
  have 4: ⊢ ◇ (init(¬ w)) = di (#True ∧ fin (init (¬ w)))
     using 1 2 3 by fastforce
  have 5: ⊢ (¬ (◇ (init(¬ w)))) = (¬ (di (#True ∧ fin (init (¬ w)))))
     using 4 by fastforce
  have 6: ⊢ □ (init w) = (¬ (di (#True ∧ fin (init (¬ w)))))
     using 5 always-d-def Initprop(2) by (metis int-eq)
  have 7: ⊢ □ (init w) = bi (¬ (fin (init (¬ 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 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))))
      by (metis 3 DiEqvNotBiNot DiState Initprop(2) StateEqvBi int-eq)
  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 OrDiEqvDi:
⊢ (f ∨ di f) = di f
proof −
  have 1: ⊢ f ⟶ di f using DiIntro by blast
 from 1 show ?thesis by auto
qed

lemma AndDiEqv:
⊢ (f ∧ di f) = f
proof −
  have 1: ⊢ f ⟶ di f using DiIntro by blast
 from 1 show ?thesis by auto
qed

lemma BiEmptyEqvEmpty:
⊢ bi empty = empty
proof −
 have 1: ⊢ bi empty = (¬ (di (¬ empty))) by (simp add: bi-d-def)
 have 2: ⊢ (¬ (di (¬ empty))) = (¬ ((¬ empty);#True)) by (simp add: di-d-def)
 have 3: ⊢ (¬ ((¬ empty);#True)) = (¬ (more;#True)) by (simp add: empty-d-def)
 have 4: ⊢ more;#True =  more using MoreEqvMoreChopTrue by auto
```

hence *5*: $\vdash (\neg(more;\#True)) = (\neg\ more)$ **by** *fastforce*
from *1 2 3 5* **show** *?thesis* **using** *NotEmptyEqvMore* **by** *fastforce*
qed

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

lemma *MoreImpImpChopSkipEqv*:
$\vdash more \longrightarrow (\ (f \longrightarrow g);skip = ((f;skip) \longrightarrow (g;skip))\ )$
proof −
have *01*: $\vdash (f \longrightarrow g) = (\neg\ f \vee g)$ **by** *auto*
hence *02*: $\vdash (f \longrightarrow g);skip = (\neg\ f \vee g);skip$ **by** (*simp add*: *LeftChopEqvChop*)
hence *1*: $\vdash (more \wedge (f \longrightarrow g);skip) = (more \wedge\ (\neg\ f \vee g);skip)$ **by** *fastforce*
have *2*: $\vdash (\neg\ f \vee g);skip = ((\neg\ f);skip \vee g;skip)$
using *OrChopEqv* **by** *auto*
hence *3*: $\vdash (more \wedge\ (\neg\ f \vee g);skip) = (more \wedge ((\neg\ f);skip \vee g;skip))$
by *auto*
have *4*: $\vdash (\neg((\neg\ f);skip)) = (empty \vee (f;skip))$
using *NotNotChopSkip* **by** *blast*
hence *5*: $\vdash ((\neg\ f);skip) = (\neg(empty \vee (f;skip)))$
by *fastforce*
have *6*: $\vdash \neg(empty \vee (f;skip)) = (more \wedge \neg(f;skip))$
using *5 NotChopSkipEqvMoreAndNotChopSkip* **by** *fastforce*
have *7*: $\vdash ((\neg\ f);skip \vee g;skip) = (\ (more \wedge \neg(f;skip)) \vee g;skip)$
using *5 6* **by** *fastforce*
hence *8*: $\vdash (more \wedge(\neg\ f;skip \vee g;skip)) = (more \wedge (\ (more \wedge \neg(f;skip)) \vee g;skip))$
by *auto*
have *9*: $\vdash (more \wedge (\ (more \wedge \neg(f;skip)) \vee g;skip)) = (more \wedge (\neg(f;skip) \vee g;skip))$
by *auto*
have *10*: $\vdash (more \wedge (\neg(f;skip) \vee g;skip)) = (more \wedge ((f;skip) \longrightarrow (g;skip)))$
by *auto*
have *11*: $\vdash (more \wedge (f \longrightarrow g);skip) = (more \wedge ((f;skip) \longrightarrow (g;skip)))$
using *1 2 3 8 9 10 7* **by** *fastforce*
from *11* **show** *?thesis* **using** *MP* **by** *fastforce*
qed

lemma *MoreImpImpPrevEqv*:

$\vdash$ *more* $\longrightarrow$ (*prev*($f \longrightarrow g$) = (*prev f* $\longrightarrow$ *prev g*))
by (*simp add*: *MoreImpImpChopSkipEqv prev-d-def*)

lemma *BiBoxNotEqvNotTrueChopChopTrue*:
$\vdash$ *bi*($\Box$ ($\neg$ *f*)) = ($\neg$((#*True*;*f*);#*True*))
by (*simp add*: *bi-d-def always-d-def di-d-def sometimes-d-def*)

lemma *DiAndEmptyEqvAndEmpty*:
$\vdash$ (*di f* $\wedge$ *empty*) = (*f* $\wedge$ *empty*)
proof −
have *1* : $\vdash$ *di f* = (*f* $\vee$ *di f*;*skip*)
using *DiEqvOrDiChopSkipB* **by** *blast*
hence *2*: $\vdash$ (*di f* $\wedge$ *empty*) = ((*f* $\vee$ *di f*;*skip*) $\wedge$ *empty*)
by *fastforce*
have *3* : $\vdash$ ((*f* $\vee$ *di f*;*skip*) $\wedge$ *empty*) = ((*f* $\wedge$ *empty*) $\vee$ (*di f*;*skip* $\wedge$ *empty*))
by *auto*
have *4*: $\vdash$ $\neg$(*di f*;*skip* $\wedge$ *empty*)
by (*metis AndChopB AndDiEqv ChopAndEmptyEqvEmptyChopEmpty DiEmpty MoreEqvSkipChopTrue TrueChopSkipEqvSkipChopTrue empty-d-def int-eq int-eq-true int-simps*(*14*) *int-simps*(*21*) *lift-and-com*)
hence *5* : $\vdash$ ((*f* $\wedge$ *empty*) $\vee$ (*di f*;*skip* $\wedge$ *empty*)) = (*f* $\wedge$ *empty*)
by *auto*
from *2 3 5* **show** *?thesis* **by** *fastforce*
qed

6.4.3 Strict initial intervals

lemma *DsMoreDi*:
$\vdash$ *ds f* = (*more* $\wedge$ (*di f*);*skip*)
proof −
have *1*: $\vdash$ *ds f* = ($\neg$(*bs* ($\neg$ *f*)))
by (*simp add*: *ds-d-def*)
have *2*: $\vdash$ ($\neg$(*bs* ($\neg$ *f*))) = ($\neg$(*empty* $\vee$ (*bi* ($\neg$ *f*));*skip*))
by (*simp add*: *bs-d-def*)
have *3*: $\vdash$ ($\neg$(*empty* $\vee$ (*bi* ($\neg$ *f*));*skip*)) = ($\neg$*empty* $\wedge$ $\neg$((*bi* ($\neg$*f*));*skip*))
by *auto*
have *4*: $\vdash$ ($\neg$*empty* $\wedge$ $\neg$((*bi* ($\neg$*f*));*skip*)) = (*more* $\wedge$ $\neg$((*bi* ($\neg$*f*));*skip*))
using *NotEmptyEqvMore* **by** *auto*
have *5*: $\vdash$ (*more* $\wedge$ $\neg$((*bi* ($\neg$*f*));*skip*)) = (*more* $\wedge$ $\neg$($\neg$(*di f*);*skip*))
by (*metis DiEqvNotBiNot DiIntro DiSkipEqvMore NotChopSkipEqvMoreAndNotChopSkip Prop10 RightChopImpMoreRule int-simps*(*4*) *inteq-reflection lift-and-com*)
have *6*: $\vdash$ (*more* $\wedge$ $\neg$(($\neg$(*di f*));*skip*)) = (*more* $\wedge$ (*empty* $\vee$ (*di f*);*skip*))
using *NotNotChopSkip* **by** *fastforce*
have *7*: $\vdash$ (*more* $\wedge$ (*empty* $\vee$ (*di f*);*skip*)) = (*more* $\wedge$ (*di f*);*skip*)
using *NotEmptyEqvMore* **by** *auto*
from *1 2 3 4 5 6 7* **show** *?thesis* **by** *fastforce*
qed

lemma *DsDi*:

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

lemma *BsEqvNotDsNot*:
$\vdash$ *bs f* = (¬(*ds* (¬ *f*)))
proof −
have *1*: $\vdash$ *ds* (¬ *f*) = (*more* ∧ (*di* (¬ *f*));*skip*)
by (*rule DsMoreDi*)
hence *2*: $\vdash$ (¬(*ds* (¬ *f*))) = (¬(*more* ∧ (*di* (¬ *f*));*skip*))
by *auto*
have *3*: $\vdash$ (¬(*more* ∧ (*di* (¬ *f*));*skip*)) = (*empty* ∨ ¬((*di* (¬ *f*));*skip*))
using *NotEmptyEqvMore* **by** *auto*
have *4*: $\vdash$ (*empty* ∨ ¬((*di* (¬ *f*));*skip*)) = (*empty* ∨ ¬((¬(*bi f*));*skip*))
using *DiNotEqvNotBi* **by** (*metis 3 inteq-reflection*)
have *5*: $\vdash$ (¬((¬(*bi f*));*skip*)) = (*empty* ∨ (*bi f*);*skip*)
by (*rule NotNotChopSkip*)
hence *6*: $\vdash$ (*empty* ∨ ¬((¬(*bi f*));*skip*)) = (*empty* ∨ (*bi f*);*skip*)
by *auto*
from *2 3 4 6* **show** *?thesis* **by** (*metis bs-d-def inteq-reflection*)
qed

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

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

lemma *NotDsAndEmpty*:
$\vdash$ ¬(*ds f* ∧ *empty*)
proof −
have *1*: $\vdash$ *ds f* = (*more* ∧ (*di f*);*skip*) **by** (*rule DsMoreDi*)
have *2*: $\vdash$ *more* ∧ (*di f*);*skip* ∧ *empty* ⟶ #*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* ∨ (*bi more*);*skip*) **by** (*simp add*: *bs-d-def*)
have *2*: $\vdash$ *bi more* ⟶ #*False* **using** *DiEmpty NotEmptyEqvMore* **by** (*simp add*: *bi-d-def empty-d-def*)
hence *3*: $\vdash$ (*bi more*);*skip* ⟶ #*False*;*skip* **using** *LeftChopImpChop* **by** *blast*
have *31*: $\vdash$ #*False*;*skip* ⟶ #*False* **by** (*simp add*: *Valid-def skip-defs chop-defs*)
have *32*: $\vdash$ (*bi more*);*skip* ⟶ #*False* **using** *3 31* **by** *fastforce*
hence *4*: $\vdash$ (*empty* ∨ ((*bi more*);*skip*)) = *empty* **by** *fastforce*
from *1 4* **show** *?thesis* **by** *fastforce*
qed

lemma *BsAndEqv*:
$\vdash$ (*bs f* ∧ *bs g*) = *bs*(*f* ∧ *g*)
proof −
have *1*: $\vdash$ *bs f* = (*empty* ∨ (*bi f*) ;*skip*)
by (*simp add*: *bs-d-def*)
have *2*: $\vdash$ *bs g* = (*empty* ∨ (*bi g*) ;*skip*)
by (*simp add*: *bs-d-def*)
have *3*: $\vdash$ (*bs f* ∧ *bs g*) = ((*empty* ∨ (*bi f*) ;*skip*) ∧ (*empty* ∨ (*bi g*) ;*skip*))
using *1 2* **by** *fastforce*
have *4*: $\vdash$ ((*empty* ∨ (*bi f*) ;*skip*) ∧ (*empty* ∨ (*bi g*) ;*skip*)) =
(*empty* ∨ ((*bi f*) ;*skip* ∧ (*bi g*) ;*skip*))
by *auto*
have *5*: $\vdash$ (((*bi f*) ;*skip* ∧ (*bi g*) ;*skip*)) = *bi*(*f* ∧ *g*);*skip*
using *BiAndChopSkipEqv* **by** *fastforce*
hence *6*: $\vdash$ (*empty* ∨ ((*bi f*) ;*skip* ∧ (*bi g*) ;*skip*)) = (*empty* ∨ *bi*(*f* ∧ *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* ∨ *ds g*) = *ds* (*f* ∨ *g*)
proof −
have *1*: $\vdash$ *ds f* = (¬(*bs* (¬ *f*))) **by** (*simp add*: *ds-d-def*)
have *2*: $\vdash$ *ds g* = (¬(*bs* (¬ *g*))) **by** (*simp add*: *ds-d-def*)
have *3*: $\vdash$ (*ds f* ∨ *ds g*) = (¬(*bs* (¬ *f*)) ∨ ¬(*bs* (¬ *g*))) **using** *1 2* **by** *fastforce*
have *4*: $\vdash$ (¬(*bs* (¬ *f*)) ∨ ¬(*bs* (¬ *g*))) = (¬(*bs* (¬ *f*) ∧ *bs* (¬ *g*))) **by** *auto*
have *5*: $\vdash$ (*bs* (¬ *f*) ∧ *bs* (¬ *g*)) = *bs*(¬ *f* ∧ ¬*g*) **by** (*rule BsAndEqv*)
hence *6*: $\vdash$ (¬(*bs* (¬ *f*) ∧ *bs* (¬ *g*))) = (¬(*bs*(¬ *f* ∧ ¬*g*))) **by** *auto*
have *7*: $\vdash$ (¬(*bs*(¬ *f* ∧ ¬*g*))) = *ds* (¬(¬ *f* ∧ ¬*g*)) **by** (*rule NotBsEqvDsNot*)
have *8*: $\vdash$ (¬(¬ *f* ∧ ¬*g*)) = (*f* ∨ *g*) **by** *auto*
hence *9*: $\vdash$ *ds*(¬(¬ *f* ∧ ¬*g*)) = *ds* (*f* ∨ *g*) **by** (*rule DsEqvRule*)
from *3 4 6 7 9* **show** *?thesis* **by** *fastforce*
qed

lemma *BsOrImp*:
$\vdash$ *bs f* $\vee$ *bs g* $\longrightarrow$ *bs*(*f* $\vee$ *g*)
proof −
have *1*: $\vdash$ *bi f* $\vee$ *bi g* $\longrightarrow$ *bi*(*f* $\vee$ *g*)
by (*rule BiOrBiImpBiOr*)
hence *2*: $\vdash$ (*bi f* $\vee$ *bi g*);*skip* $\longrightarrow$ (*bi*(*f* $\vee$ *g*));*skip*
by (*rule LeftChopImpChop*)
have *3*: $\vdash$ (*bi f*);*skip* $\vee$ (*bi g*);*skip* $\longrightarrow$ (*bi*(*f* $\vee$ *g*));*skip*
using *1 OrChopEqv 2* **by** *fastforce*
hence *4*: $\vdash$ *empty* $\vee$ (*bi f*);*skip* $\vee$ (*bi g*);*skip* $\longrightarrow$ *empty* $\vee$ (*bi*(*f* $\vee$ *g*));*skip*
by *auto*
hence *5*: $\vdash$ (*empty* $\vee$ (*bi f*);*skip*) $\vee$ (*empty* $\vee$ (*bi g*);*skip*) $\longrightarrow$ *empty* $\vee$ (*bi*(*f* $\vee$ *g*));*skip*
by *auto*
from *5* **show** *?thesis* **by** (*simp add*: *bs-d-def*)
qed

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

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

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

lemma *BiImpBs*:
$\vdash$ *bi f* $\longrightarrow$ *bs f*
proof −
have *1*: $\vdash$ *empty* $\longrightarrow$ *empty* $\vee$ (*bi f*);*skip* **by** *auto*
hence *2*: $\vdash$ *empty* $\wedge$ *bi f* $\longrightarrow$ *empty* $\vee$ (*bi f*);*skip* **by** *auto*
have *2*: $\vdash$ *more* $\wedge$ *bi f* $\longrightarrow$ (*bi f*);*skip* **by** (*rule MoreAndBiImpBiChopSkip*)
hence *3*: $\vdash$ *more* $\wedge$ *bi f* $\longrightarrow$ *empty* $\vee$ (*bi f*);*skip* **by** *auto*
have *4*: $\vdash$ *bi f* = ((*bi f* $\wedge$ *empty*) $\vee$ (*bi f* $\wedge$ *more*)) **by** (*simp add*: *empty-d-def*, *auto*)
have *5*: $\vdash$ (*empty* $\vee$ (*bi f*);*skip*) = *bs f* **by** (*simp add*: *bs-d-def*)
from *2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *BsImpBsBs*:
$\vdash$ *bs f* $\longrightarrow$ *bs* (*bs f*)
proof −
have *1*: $\vdash$ *bi f* $\longrightarrow$ *bs f* **by** (*rule BiImpBs*)
hence *2*: $\vdash$ *bi* (*bi f*) $\longrightarrow$ *bi*(*bs f*) **by** (*rule BiImpBiRule*)
hence *3*: $\vdash$ (*bi f*) $\longrightarrow$ *bi*(*bs f*) **using** *BiEqvBiBi* **by** *fastforce*
hence *4*: $\vdash$ (*bi f*);*skip* $\longrightarrow$ (*bi*(*bs f*));*skip* **by** (*rule LeftChopImpChop*)
hence *5*: $\vdash$ *empty* $\vee$ (*bi f*);*skip* $\longrightarrow$ *empty* $\vee$ (*bi*(*bs f*));*skip* **by** *auto*
from *5* **show** *?thesis* **by** (*simp add*: *bs-d-def*)
qed

lemma *DsImpDi*:
$\vdash$ *ds f* $\longrightarrow$ *di f*
proof −
have *1*: $\vdash$ *bi* ($\neg$ *f*) $\longrightarrow$ *bs* ($\neg$ *f*) **by** (*rule BiImpBs*)
hence *2*: $\vdash$ $\neg$(*bs* ($\neg$ *f*)) $\longrightarrow$ $\neg$(*bi* ($\neg$ *f*)) **by** *fastforce*
from *2* **show** *?thesis* **using** *NotBsEqvDsNot DiEqvNotBiNot* **by** *fastforce*
qed

lemma *BsImpBsRule*:
assumes $\vdash$ *f* $\longrightarrow$ *g*
shows $\vdash$ *bs f* $\longrightarrow$ *bs g*
proof −
have *1*: $\vdash$ *f* $\longrightarrow$ *g* **using** *assms* **by** *auto*
hence *2*: $\vdash$ *bi f* $\longrightarrow$ *bi g* **by** (*rule BiImpBiRule*)
hence *3*: $\vdash$ (*bi f*);*skip* $\longrightarrow$ (*bi g*);*skip* **by** (*rule LeftChopImpChop*)
hence *4*: $\vdash$ *empty* $\vee$ (*bi f*);*skip* $\longrightarrow$ *empty* $\vee$ (*bi g*);*skip* **by** *auto*
from *4* **show** *?thesis* **by** (*simp add*: *bs-d-def*)
qed

lemma *DsChopImpDsB*:
$\vdash$ *ds* (*f*;*g*) $\longrightarrow$ *ds f*
proof −
have *1*: $\vdash$ *di*(*f*;*g*) $\longrightarrow$ *di f* **by** (*rule DiChopImpDiB*)
hence *2*: $\vdash$ (*di*(*f*;*g*));*skip* $\longrightarrow$ (*di f*);*skip* **by** (*rule LeftChopImpChop*)
from *2* **show** *?thesis* **using** *DsDi* **by** *fastforce*
qed

lemma *NotBsImpBsNotChop*:
$\vdash$ *bs* ($\neg$ *f*) $\longrightarrow$ *bs* ($\neg$(*f*;*g*))
proof −
have *1*: $\vdash$ *ds* (*f*;*g*) $\longrightarrow$ *ds f* **by** (*rule DsChopImpDsB*)
hence *2*: $\vdash$ $\neg$(*ds f*) $\longrightarrow$ $\neg$(*ds* (*f*;*g*)) **by** *fastforce*
from *2* **show** *?thesis* **using** *NotDsEqvBsNot* **by** *fastforce*
qed

lemma *BsOrBsEqvBsBiOrBi*:
$\vdash$ (*bs f* $\vee$ *bs g*) = *bs*(*bi f* $\vee$ *bi g*)

proof −
have *1*: ⊢ (*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 (*meson BiBiOrImpBi BiImpBiBiOr BiImpBiBiOrB Prop02 int-iffI*)
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 *OrDsEqvDi*:
⊢ (*f* ∨ *ds f*) = *di f*
proof −
have *1*: ⊢ *ds f* = (*di f*);*skip* **by** (*rule DsDi*)
hence *2*: ⊢ (*f* ∨ *ds f*) = (*f* ∨ (*di f*);*skip*) **by** *auto*
from *2* **show** *?thesis* **using** *DiEqvOrDiChopSkipB* **by** *fastforce*
qed

lemma *AndBsEqvBi*:
⊢ (*f* ∧ *bs f*) = *bi f*

proof −
have *1*: ⊢ (*f* ∧ *bs f*) = (*f* ∧ (*empty* ∨ (*bi f*);*skip*)) **by** (*simp add*: *bs-d-def*)
from *1* **show** *?thesis* **using** *BiEqvAndEmptyOrBiChopSkip* **by** *fastforce*
qed

lemma *BsEqvBsBi*:
⊢ *bs f* = *bs* (*bi f*)
proof −
have *1*: ⊢ *bs f* = (*empty* ∨ (*bi f*);*skip*) **by** (*simp add*: *bs-d-def*)
have *2*: ⊢ *bi f* = *bi* (*bi f*) **by** (*rule BiEqvBiBi*)
hence *3*: ⊢ (*bi f*);*skip* = *bi* (*bi f*);*skip* **using** *LeftChopEqvChop* **by** *blast*
hence *4*: ⊢ (*empty* ∨ (*bi f*);*skip*) = (*empty* ∨ *bi* (*bi f*);*skip*) **by** *auto*
from *1 4* **show** *?thesis* **by** (*simp add*: *bs-d-def*)
qed

lemma *StateImpBs*:
⊢ *init w* ⟶ *bs* (*init w*)
proof −
have *1*: ⊢ *init w* = *bi* (*init w*) **by** (*rule StateEqvBi*)
have *2*: ⊢ *bi* (*init w*) ⟶ *bs* (*init w*) **by** (*rule BiImpBs*)
from *1 2* **show** *?thesis* **using** *StateImpBi* **by** *fastforce*
qed

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

lemma *BsEqvBiMoreImpChop*:
$\vdash$ *bs f* = *bi*(*more* $\longrightarrow$ *f*;*skip*)
proof −
have *1*: $\vdash$ *bs f* = (*empty* $\vee$ (*bi f*;*skip*))
by (*simp add*: *bs-d-def*)
have *2*: $\vdash$ (*empty* $\vee$ (*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* $\vee$ (*f*;*skip*))
using *NotNotChopSkip* **using** *BiEqvBi* **by** *blast*
have *11*: $\vdash$ *bi*(*empty* $\vee$ (*f*;*skip*)) = *bi*(¬ *more* $\vee$ (*f*;*skip*))
by (*simp add*: *empty-d-def*)
have *12*: $\vdash$ (¬ *more* $\vee$ (*f*;*skip*)) = (*more* $\longrightarrow$ *f*;*skip*) **by** *auto*
have *13*: $\vdash$ *bi*(¬ *more* $\vee$ (*f*;*skip*)) = *bi*(*more* $\longrightarrow$ *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* $\longrightarrow$ *f*;*skip*) **using** *8 9 10 11 13* **by** *fastforce*
from *14 15* **show** *?thesis* **by** *fastforce*
qed

lemma *BoxMoreStateEqvBsFinState*:
$\vdash$ □(*more* $\longrightarrow$ ¬ (*init w*)) = *bs*(¬(*fin*(*init w*)))
proof −
have *1*: $\vdash$ □(*more* $\longrightarrow$ ¬ (*init w*)) = (¬(◇(¬(*more* $\longrightarrow$ ¬ (*init w*)))))
by (*simp add*: *always-d-def*)
have *01*: $\vdash$ (¬(*more* $\longrightarrow$ ¬ (*init w*))) = (*init w* ∧ *more*) **by** *auto*
hence *2*: $\vdash$ ¬(◇(¬(*more* $\longrightarrow$ ¬ (*init w*)))) = (¬(#*True*;(*init w* ∧ *more*)))
by (*metis int-eq int-iffD1 int-simps*(*14*) *int-simps*(*6*) *sometimes-d-def*)
have *3*: $\vdash$ *more* = #*True*; *skip*
using *MoreEqvSkipChopTrue SkipTrueEqvTrueSkip* **by** *fastforce*
have *4*: $\vdash$ (*init w* ∧ *more*) = (*init w* ∧ (#*True*; *skip*))
using *3* **by** *auto*
have *5*: $\vdash$ (*init w* ∧ (#*True*; *skip*)) = ((*init w* ∧ *empty*);(#*True*;*skip*))
using *StateAndEmptyChop* **by** *fastforce*
have *6*: $\vdash$ (*init w* ∧ *more*) = ((*init w* ∧ *empty*);(#*True*;*skip*))
using *4 5* **by** *fastforce*
have *7*: $\vdash$ (#*True*;(*init w* ∧ *more*)) = (#*True*;((*init w* ∧ *empty*);(#*True*;*skip*)))

using *6 RightChopEqvChop* **by** *blast*
have *8*: $\vdash (\#True;((init\ w \wedge empty);(\#True;skip))\ ) =$
$(((\#True;(init\ w \wedge empty));(\#True;skip))\ )$
using *ChopAssoc* **by** *blast*
have *9*: $\vdash (((\#True;(init\ w \wedge empty));(\#True;skip))\ ) =$
$((((\#True;(init\ w \wedge empty));\#True);skip)\ )$
using *ChopAssoc* **by** *blast*
have *10*: $\vdash (\#True;(init\ w \wedge\ more)) =$
$((((\#True;(init\ w \wedge empty));\#True);skip)\ )$
using *7 8 9* **by** *fastforce*
hence *11*: $\vdash (\neg(\#True;(init\ w \wedge\ more))) =$
$(\neg((((\#True;(init\ w \wedge empty));\#True);skip)\ ))$
by *auto*
have *12*: $\vdash \neg((((\#True;(init\ w \wedge empty));\#True);skip)\ ) =$
$empty \vee (\neg(((\#True;(init\ w \wedge empty));\#True);skip)$
using *NotChopNotSkip* **by** *fastforce*
have *13*: $\vdash (\neg(((\#True;(init\ w \wedge empty));\#True)) = bi(\Box\ (\neg(init\ w \wedge empty)))$
using *BiBoxNotEqvNotTrueChopChopTrue* **by** *fastforce*
hence *14*: $\vdash (\neg(((\#True;(init\ w \wedge empty));\#True));skip =$
$(bi(\Box\ (\neg(init\ w \wedge empty))));skip$
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 *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 BiEqvAndWprevBi MoreEqvSkipChopTrue NotChopSkipEqvMoreAndNotChopSkip SkipTrueEqvTrueSkip int-eq int-iffD1 int-simps*(*14*) *int-simps*(*19*) *int-simps*(*2*) *int-simps*(*21*))
from *1 2* **show** *?thesis* **by** *fastforce*
qed

6.4.4 First occurrence

lemma *FstWithAndImp*:
$\vdash$ $\rhd$*f* $\wedge$ *g* $\longrightarrow$ $\rhd$ (*f* $\wedge$ *g*)
proof −
have *1*: $\vdash$ ($\rhd$*f* $\wedge$ *g*) = ((*f* $\wedge$ (*bs* ($\neg$*f*))) $\wedge$ *g*)
by (*simp add*: *first-d-def*)
have *2*: $\vdash$ ((*f* $\wedge$ (*bs* ($\neg$*f*))) $\wedge$ *g*) = (*f* $\wedge$ $\neg$(*ds f*) $\wedge$ *g*)
using *NotDsEqvBsNot* **by** *fastforce*
have *3*: $\vdash$ $\neg$(*ds f*) $\longrightarrow$ $\neg$(*ds*(*f* $\wedge$ *g*))
using *DsAndImpElimL* **by** *fastforce*
hence *4*: $\vdash$ *f* $\wedge$ $\neg$(*ds f*) $\wedge$ *g* $\longrightarrow$ *f* $\wedge$ *g* $\wedge$ $\neg$(*ds*(*f* $\wedge$ *g*))
by *auto*
have *5*: $\vdash$ (*f* $\wedge$ *g* $\wedge$ $\neg$(*ds*(*f* $\wedge$ *g*))) = ((*f* $\wedge$ *g*) $\wedge$ (*bs* ($\neg$(*f* $\wedge$ *g*))))
using *NotDsEqvBsNot* **by** *fastforce*
have *6*: $\vdash$ ((*f* $\wedge$ *g*) $\wedge$ (*bs* ($\neg$(*f* $\wedge$ *g*)))) = $\rhd$(*f* $\wedge$ *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* $(\neg\ g)) = (\triangleright f \wedge$ *bs* $(\neg\ g))$
by (*simp add*: *first-d-def*)
have *8*: $\vdash$ $(g \wedge$ *bs* $(\neg\ f) \wedge$ *bs* $(\neg\ g)) = ((g \wedge$ *bs* $(\neg\ g)) \wedge$ *bs* $(\neg\ f))$
by *auto*
have *9*: $\vdash$ $((g \wedge$ *bs* $(\neg\ g)) \wedge$ *bs* $(\neg\ f)) = (\triangleright g \wedge$ *bs* $(\neg\ f))$
by (*simp add*: *first-d-def*)
have *10*: $\vdash$ $((f \wedge$ *bs* $(\neg\ f)) \wedge$ *bs* $(\neg\ g)) \vee (g \wedge$ *bs* $(\neg\ f) \wedge$ *bs* $(\neg\ g)) =$
$(\triangleright f \wedge$ *bs* $(\neg\ g)) \vee (\triangleright g \wedge$ *bs* $(\neg\ f))$
using *7 8 9* **by** *fastforce*
from *1 5 6 10* **show** *?thesis* **by** (*metis 7 8 9 int-eq*)
qed

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

lemma *FstTrue*:
$\vdash \triangleright\ \#True = empty$
proof −
have *1*: $\vdash \triangleright\ \#True = (\#True \wedge$ *bs* $(\neg\ \#True))$
by (*simp add*: *first-d-def*)
have *2*: $\vdash$ *bs* $(\neg\ \#True) =$ (*empty* $\vee$ (*bi* $(\neg\ \#True))$;*skip*)
by (*simp add*: *bs-d-def*)
have *3*: $\vdash \neg$(*bi* $(\neg\ \#True))$

```
    using BiElim by fastforce
 have 4: ⊢ ¬((bi (¬ #True));skip)
    by (metis AndChopA BiEqvAndEmptyOrBiChopSkip MoreEqvSkipChopTrue
       NotChopSkipEqvMoreAndNotChopSkip SkipTrueEqvTrueSkip int-eq int-simps(14) int-simps(21))
 have 5: ⊢ bs (¬ #True) = empty
    using 2 4 by fastforce
 from 1 5 show ?thesis by fastforce
qed

lemma FstFalse:
  ⊢ ¬(▷ #False)
proof −
 have 1: ⊢ ▷ #False = (#False ∧ bs #True) by (simp add: first-d-def)
 from 1 show ?thesis by auto
qed

lemma FstChopFalseEqvFalse:
⊢ ¬(▷f ; #False)
by (simp add:Valid-def chop-defs)

lemma FstEmpty:
⊢ ▷ empty = empty
proof −
 have 1: ⊢ ▷ empty = (empty ∧ bs (¬ empty)) by (simp add: first-d-def)
 have 2: ⊢ bs (¬ empty) = (empty ∨ bi (¬ empty);skip) by (simp add: bs-d-def)
 from 1 2 show ?thesis by fastforce
qed

lemma FstAndEmptyEqvAndEmpty:
⊢ (▷f ∧ empty) = (f ∧ empty)
proof −
 have 1: ⊢ (▷f ∧ empty) = ((f ∧ bs (¬ f)) ∧ empty) by (simp add: first-d-def)
 have 2: ⊢ bs (¬ f) = (empty ∨ bi (¬f);skip) by (simp add: bs-d-def)
 from 1 2 show ?thesis by fastforce
qed

lemma FstEmptyOrEqvEmpty:
⊢ ▷(empty ∨ f) = empty
proof −
 have 1: ⊢ ▷(empty ∨ f) = ((▷empty ∧ bs (¬ f)) ∨ (▷f ∧ bs (¬ empty))) using FstWithOrEqv by blast
 have 2: ⊢  (¬ empty) = more  by (simp add: empty-d-def)
 hence 3: ⊢ bs (¬ empty) = bs more using BsEqvRule by blast
 have 4: ⊢ bs more = empty using BsMoreEqvEmpty by blast
 have 5: ⊢ (▷f ∧ bs (¬ empty)) = (▷f ∧ empty) using 3 4 by fastforce
 have  6: ⊢ ▷empty = empty using FstEmpty by blast
 hence 7: ⊢ (▷empty ∧ bs (¬ f)) = (empty ∧ bs (¬ f)) by auto
 have 8: ⊢ (empty ∧  bs (¬ f)) = (empty ∧(empty ∨ bi (¬ f);skip)) by (simp add:bs-d-def)
 have 9: ⊢ (empty ∧(empty ∨ bi (¬ f);skip)) = empty by auto
 have 10: ⊢ (empty ∧  bs (¬ f)) = empty  using 8 9 by auto
 have 11: ⊢ ((▷empty ∧ bs (¬ f)) ∨ (▷f ∧ bs (¬ empty))) =
```

```
        (empty ∨ (▷f ∧ empty)) using 7 10 5 by fastforce
 have 12: ⊢ (empty ∨ (▷f ∧ empty)) = empty by auto
 from 1 11 12 show ?thesis by fastforce
qed

lemma FstChopEmptyEqvFstChopFstEmpty:
 ⊢ (▷f;g ∧ empty) = (▷f;▷g ∧ empty)
proof −
 have 1: ⊢ (▷f;g ∧ empty) = (▷f ∧ g ∧ empty) using ChopEmptyAndEmpty by blast
 have 2: ⊢ (▷g ∧ empty) = (g ∧ empty)   using FstAndEmptyEqvAndEmpty by blast
 hence 3: ⊢ (▷f ∧ g ∧ empty) = (▷f ∧ ▷g ∧ empty) by auto
 have 4: ⊢ (▷f;▷g ∧ empty) = (▷f ∧ ▷g ∧ empty) using ChopEmptyAndEmpty by blast
 from 1 3 4 show ?thesis by fastforce
qed

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

lemma FstEqvBsNotAndDi:
 ⊢ ▷f = (bs (¬ f) ∧ di f)
proof −
 have 1: ⊢ bs (¬ f)  = (¬(ds f))  by (simp add: ds-d-def)
 hence 2: ⊢ (bs (¬ f)  ∧ di f) = (¬(ds f) ∧ di f) by auto
 have 3: ⊢  di f = (ds f ∨ f) using OrDsEqvDi by fastforce
 hence 4 : ⊢ (¬(ds f)  ∧ di f) = (¬(ds f)  ∧ (ds f ∨ f)) by auto
 have 5 : ⊢ (¬(ds f)  ∧ (ds f ∨ f)) = (¬(ds f)  ∧ f) by auto
 have 6 : ⊢ (¬(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:
 ⊢ (▷f ∨ di f) = di f
proof −
 have 1: ⊢ (▷f ∨ di f) = ((f ∧ bs (¬f)) ∨ di f) by (simp add: first-d-def)
 have 2: ⊢ ((f ∧ bs (¬f)) ∨ di f) = ((f ∨ di f) ∧ (bs (¬ f) ∨ di f)) by auto
 have 3: ⊢ (f ∨ di f) = di f
 by (metis 2 DiIntro RRDiamondEqvDi int-eq Prop02 Prop03 Prop11 Prop12)
 hence 4: ⊢ ((f ∨ di f) ∧ (bs (¬ f) ∨ di f)) = (di f ∧ (bs (¬ f) ∨ di f)) by auto
```

have *5*: $\vdash (di\ f \wedge (bs\ (\neg\ f) \vee di\ f)) = di\ f$ **by** *auto*
from *1 2 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndDiEqvFst*:
$\vdash (\rhd f \wedge di\ f) = \rhd f$
proof −
have *1*: $\vdash (\rhd f \wedge di\ f) = ((f \wedge bs\ (\neg f)) \wedge di\ f)$ **by** (*simp add*: *first-d-def*)
have *2*: $\vdash (f \wedge di\ f) = f$ **by** (*meson DiIntro Prop10 Prop11*)
hence *3*: $\vdash (f \wedge bs\ (\neg f) \wedge di\ f) = (f \wedge bs\ (\neg f))$ **by** *auto*
from *1 3* **show** *?thesis* **by** (*metis first-d-def int-iffD2 int-iffI Prop12*)
qed

lemma *DiEqvDiFst*:
$\vdash di\ f = di\ (\rhd\ f)$
proof −
have *1*: $\vdash\ \ di\ (\rhd\ f) = di\ (f \wedge bs\ (\neg f))$
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*: $\vdash di\ (\rhd\ f) \longrightarrow di\ f$ **using** *1 3*
by *fastforce*
have *5*: $\vdash (di\ f \wedge empty) = (f \wedge empty)$
using *DiAndEmptyEqvAndEmpty* **by** *blast*
have *6*: $\vdash (\rhd\ f \wedge empty) = (f \wedge empty)$
using *FstAndEmptyEqvAndEmpty* **by** *auto*
have *7*: $\vdash di\ f \wedge empty \longrightarrow \rhd f$
using *5 6* **by** *fastforce*
have *8*: $\vdash \rhd f \longrightarrow di\ (\rhd\ f)$
using *DiIntro* **by** *auto*
have *9*: $\vdash\ \ di\ f \wedge empty \longrightarrow di\ (\rhd\ f)$
using *7 8* **using** *lift-imp-trans* **by** *blast*
hence *10*: $\vdash empty \longrightarrow (di\ f \longrightarrow di\ (\rhd\ f))$
by *auto*
have *11*: $\vdash prev\ (\ di\ f \longrightarrow di\ (\rhd\ f)) \longrightarrow more$
by (*simp add*: *ChopSkipImpMore prev-d-def*)
have *12*: $\vdash more \longrightarrow (\ prev\ (\ di\ f \longrightarrow di\ (\rhd\ f)) = (prev(di\ f) \longrightarrow prev(di\ (\rhd f))))$
using *MoreImpImpPrevEqv* **by** *auto*
have *13*: $\vdash (more \wedge prev\ (\ di\ f \longrightarrow di\ (\rhd\ f))) = (more \wedge (prev(di\ f) \longrightarrow prev(di\ (\rhd f))))$
using *12* **by** *fastforce*
have *14*: $\vdash prev\ (\ di\ f \longrightarrow di\ (\rhd\ f)) = (more \wedge (prev(di\ f) \longrightarrow prev(di\ (\rhd f))))$
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*: $\vdash (\neg(bs\ (\neg\ f)) \vee bs\ (\neg\ f)) \wedge (prev\ (di\ f \longrightarrow di\ (\triangleright\ f))) \longrightarrow (di\ f \longrightarrow di\ (\triangleright f))$
using *40 41* **by** *fastforce*
have *43*: $\vdash (\neg(bs\ (\neg\ f)) \vee bs\ (\neg\ f))$
by *auto*
have *44*: $\vdash (prev\ (di\ f \longrightarrow di\ (\triangleright\ f))) \longrightarrow (di\ f \longrightarrow di\ (\triangleright f))$
using *42 43* **by** *fastforce*
have *45*: $\vdash di\ f \longrightarrow di\ (\triangleright f)$
using *10 44 EmptyChopSkipInduct* **by** *blast*
from *4 45* **show** *?thesis* **by** *fastforce*
qed

lemma *FstDiEqvFst*:
$\vdash \triangleright(di\ f) = \triangleright f$
proof −
have *1*: $\vdash \triangleright(di\ f) = (di\ f \wedge bs\ (\neg\ (di\ f)))$ **by** (*simp add*: *first-d-def*)
have *2*: $\vdash\ (\neg\ (di\ f)) = bi\ (\neg\ f)$ **by** (*simp add*: *NotDiEqvBiNot*)
hence *3*: $\vdash bs\ (\neg\ (di\ f)) = bs\ (bi\ (\neg\ f))$ **using** *BsEqvRule* **by** *blast*
have *4*: $\vdash bs\ (bi\ (\neg\ f)) = bs\ (\neg\ f)$ **using** *BsEqvBsBi* **by** *fastforce*
hence *5*: $\vdash (di\ f \wedge bs\ (\neg\ (di\ f))) = (di\ f \wedge bs\ (\neg\ f))$ **using** *3* **by** *fastforce*
have *6* : $\vdash di\ f = (f \vee ds\ f)$ **using** *OrDsEqvDi* **by** *fastforce*
hence *7*: $\vdash (di\ f \wedge bs\ (\neg\ f)) = ((f \vee ds\ f) \wedge bs\ (\neg\ f))$ **by** *auto*
have *8*: $\vdash ((f \vee ds\ f) \wedge bs\ (\neg\ f)) = ((f \wedge bs\ (\neg\ f)) \vee (ds\ f \wedge bs\ (\neg\ f)))$ **by** *auto*
have *9*: $\vdash \neg(ds\ f \wedge bs\ (\neg\ f))$ **by** (*simp add*: *ds-d-def*)
have *10*: $\vdash (f \wedge bs\ (\neg\ f)) = \triangleright f$ **by** (*simp add*: *first-d-def*)
have *11*: $\vdash ((f \wedge bs\ (\neg\ f)) \vee (ds\ f \wedge bs\ (\neg\ f))) = \triangleright f$ **using** *9 10* **by** *fastforce*
from *1 5 7 8 11* **show** *?thesis* **by** (*metis int-eq*)
qed

lemma *DiAndFstOrEqvFstOrDiAnd*:
$\vdash (di\ f \wedge (\triangleright f \vee g)) = (\triangleright f \vee (di\ f \wedge g))$
proof −
have *1*: $\vdash (di\ f \wedge (\triangleright f \vee g)) = (\triangleright f \wedge di\ f) \vee (di\ f \wedge g)$ **by** *auto*
have *2*: $\vdash (\triangleright f \wedge di\ f) = \triangleright f$ **using** *FstAndDiEqvFst* **by** *blast*
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *DiOrFstAndEqvDi*:
$\vdash di\ f \vee (\triangleright f \wedge g) = di\ f$
proof −
have *1*: $\vdash (di\ f \vee (\triangleright f \wedge g)) = ((\triangleright f \vee di\ f) \wedge (di\ f \vee g))$ **by** *auto*
have *2*: $\vdash (\triangleright f \vee di\ f) = di\ f$ **using** *FstOrDiEqvDi* **by** *blast*
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *FstDiAndDiEqv*:
$\vdash \triangleright(di\ f \wedge di\ g) = ((\triangleright f \wedge di\ g) \vee (\triangleright g \wedge di\ f))$
proof −
have *1*: $\vdash \triangleright(di\ f \wedge di\ g) = ((di\ f \wedge di\ g) \wedge bs\ (\neg(di\ f \wedge di\ g)))$ **by** (*simp add*: *first-d-def*)
have *2*: $\vdash (\neg(di\ f \wedge di\ g)) = (bi\ (\neg f) \vee bi\ (\neg g))$ **by** (*simp add*: *bi-d-def*, *auto*)

hence *3*: $\vdash$ *bs* $(\neg(di\ f \wedge di\ g)) = bs(bi\ (\neg f) \vee bi\ (\neg g))$ **using** *BsEqvRule* **by** *blast*
hence *4*: $\vdash$ $((di\ f \wedge di\ g) \wedge bs\ (\neg(di\ f \wedge di\ g))) =$
$(di\ f \wedge di\ g \wedge bs(bi\ (\neg f) \vee bi\ (\neg g)))$ **by** *auto*
have *5*: $\vdash$ $(bs\ (\neg f) \vee bs\ (\neg g)) = bs(bi\ (\neg f) \vee bi\ (\neg g))$ **using** *BsOrBsEqvBsBiOrBi* **by** *blast*
hence *6*: $\vdash$ $(di\ f \wedge di\ g \wedge bs(bi\ (\neg f) \vee bi\ (\neg g))) =$
$(di\ f \wedge di\ g \wedge (bs\ (\neg f) \vee bs\ (\neg g)))$ **by** *auto*
have *7*: $\vdash$ $(di\ f \wedge di\ g \wedge (bs\ (\neg f) \vee bs\ (\neg g))) =$
$((\ bs\ (\neg f) \wedge di\ f \wedge di\ g) \vee (di\ f \wedge bs\ (\neg g) \wedge di\ g\ ))$ **by** *auto*
have *8*: $\vdash \rhd f =\ (bs\ (\neg f)\ \wedge di\ f)$ **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 *FstState*:
$\vdash \rhd\ (init\ w) = (empty \wedge init\ w)$
proof $-$
have *1*: $\vdash \rhd\ (init\ w) = (init\ w \wedge bs\ (\neg(init\ w)))$ **by** (*simp add*: *first-d-def*)
hence *2*: $\vdash \rhd\ (init\ w) \longrightarrow init\ w$ **by** *auto*
have *3*: $\vdash init\ w \longrightarrow bs\ (init\ w)$ **using** *StateImpBs* **by** *auto*
have *4*: $\vdash \rhd\ (init\ w) \longrightarrow bs\ (init\ w)$ **using** *2 3* **by** *fastforce*
have *5*: $\vdash \rhd\ (init\ w) \longrightarrow bs\ (\neg(init\ w))$ **using** *1* **by** *auto*
have *6*: $\vdash \rhd\ (init\ w) \longrightarrow bs\ (init\ w) \wedge bs\ (\neg(init\ w))$ **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*: $\vdash \rhd\ (init\ w) \longrightarrow empty$ **using** *10 9 7 6* **by** *fastforce*
have *12*: $\vdash \rhd\ (init\ w) \longrightarrow empty \wedge init\ w$ **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* $(\neg(\mathit{init}\ w))$;*skip* **by** *auto*
hence *15*: $\vdash$ *empty* $\wedge$ *init w* $\longrightarrow$ *bs* $(\neg(\mathit{init}\ w))$ **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* $(\neg(\mathit{init}\ w))$ **using** *16 15* **by** *auto*
hence *18*: $\vdash$ *empty* $\wedge$ *init w* $\longrightarrow$ $\rhd(\mathit{init}\ w)$ **by** (*simp add*: *first-d-def*)
from *12 18* **show** *?thesis* **by** *fastforce*
qed

lemma *FstStateAndBsNotEmpty*:
$\vdash$ $(\rhd\ (\mathit{init}\ w) \wedge \mathit{bs}\ (\neg\ \mathit{empty})) = \rhd\ (\mathit{init}\ w)$
proof −
have *1*: $\vdash$ $(\rhd\ (\mathit{init}\ w) \wedge \mathit{bs}\ (\neg\ \mathit{empty})) = (\rhd\ (\mathit{init}\ w) \wedge \mathit{bs\ more})$
using *BsEqvRule NotEmptyEqvMore* **by** (*simp add*: *empty-d-def*)
have *2*: $\vdash$ $(\rhd\ (\mathit{init}\ w) \wedge \mathit{bs\ more}) = (\rhd\ (\mathit{init}\ w) \wedge \mathit{empty})$
using *BsMoreEqvEmpty* **by** *fastforce*
have *3*: $\vdash$ $\rhd\ (\mathit{init}\ w)\ =\ (\mathit{empty} \wedge (\mathit{init}\ w))$
using *FstState* **by** *blast*
hence *4*: $\vdash$ $(\rhd\ (\mathit{init}\ w)\ \wedge \mathit{empty}) =\ (\mathit{empty} \wedge (\mathit{init}\ w)\ \wedge \mathit{empty})$
by *auto*
have *5*: $\vdash$ $(\mathit{empty} \wedge (\mathit{init}\ w)\ \wedge \mathit{empty}) = (\mathit{empty} \wedge (\mathit{init}\ w))$
by *auto*
have *6*: $\vdash$ $(\mathit{empty} \wedge (\mathit{init}\ w)) = \rhd(\mathit{init}\ w)$
using *FstState* **by** *fastforce*
from *1 2 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *FstStateImpFstStateOr*:
$\vdash$ $\rhd(\mathit{init}\ w) \longrightarrow \rhd(\mathit{init}\ w \vee f)$
proof −
have *1*: $\vdash$ $\rhd(\mathit{init}\ w) = (\mathit{empty} \wedge \mathit{init}\ w)$
using *FstState* **by** *blast*
have *2*: $\vdash$ $(\mathit{empty} \wedge \mathit{init}\ w) =\ (\mathit{empty} \wedge (\mathit{empty} \vee \mathit{bi}\ (\neg\ f);\mathit{skip}) \wedge \mathit{init}\ w)$
by *auto*
have *3*: $\vdash$ $(\mathit{empty} \wedge (\mathit{empty} \vee \mathit{bi}\ (\neg\ f);\mathit{skip}) \wedge \mathit{init}\ w) =$
$(\mathit{empty} \wedge \mathit{bs}\ (\neg\ f) \wedge \mathit{init}\ w)$
by (*simp add*: *bs-d-def*)
have *4*: $\vdash$ $(\mathit{empty} \wedge \mathit{bs}\ (\neg\ f) \wedge \mathit{init}\ w) = (\mathit{empty} \wedge \mathit{init}\ w \wedge \mathit{bs}\ (\neg\ f))$
by *auto*
have *5*: $\vdash$ $(\mathit{empty} \wedge \mathit{init}\ w)\ = \rhd\ (\mathit{init}\ w)$
using *FstState* **by** *fastforce*
hence *6*: $\vdash$ $(\mathit{empty} \wedge \mathit{init}\ w \wedge \mathit{bs}\ (\neg\ f)) = (\rhd\ (\mathit{init}\ w) \wedge \mathit{bs}\ (\neg\ f))$
by *auto*
have *7*: $\vdash$ $\rhd\ (\mathit{init}\ w) \wedge \mathit{bs}\ (\neg\ f) \longrightarrow (\rhd\ (\mathit{init}\ w) \wedge \mathit{bs}\ (\neg\ f)) \vee (\rhd f \wedge \mathit{bs}\ (\neg(\mathit{init}\ w)))$
by *auto*
have *8*: $\vdash$ $\rhd(\mathit{init}\ w \vee f) = ((\rhd\ (\mathit{init}\ w) \wedge \mathit{bs}\ (\neg\ f)) \vee (\rhd f \wedge \mathit{bs}\ (\neg(\mathit{init}\ w))))$
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*: $(\sigma \models \rhd\ f) = (\ (\sigma \models \rhd\ f) \wedge (\exists\, k.\ (\sigma \models \mathit{len}(k))))$
using *AndExistsLen*[*of TEMP* $\rhd\ f$] **by** (*simp add*: *Valid-def*)
have *3*: $((\sigma \models \rhd\ f) \wedge (\exists\, k.\ (\sigma \models \mathit{len}(k)))) =$
$(\exists\, k.\ \ (\sigma \models \rhd\ f) \wedge\ \ (\sigma \models \mathit{len}(k)))$
by *auto*
have *4*: $(\sigma \models \mathit{di}(\rhd\ f)) = (\exists\, k.\ (\sigma \models \mathit{di}(\rhd f \wedge \mathit{len}(k))))$
using *2 3* **by** (*metis 1 DiLensem di-defs*)
have *5*: $(\exists\, k.\ (\sigma \models \mathit{di}(\rhd f \wedge \mathit{len}(k))))$
using *1* **using** *4* **by** *auto*
then obtain *i* **where** *6*: $(\sigma \models \mathit{di}(\rhd f \wedge \mathit{len}(i)))$ **by** *blast*
from *5* **obtain** *j* **where** *7*: $(\sigma \models \mathit{di}(\rhd f \wedge \mathit{len}(j)))$ **by** *blast*
have *8*: $(\sigma \models \mathit{di}(\rhd f \wedge \mathit{len}(i))) \wedge (\sigma \models \mathit{di}(\rhd f \wedge \mathit{len}(j)))$
using *6 7* **by** *auto*
hence *9*: $(\sigma \models \mathit{di}(\rhd f \wedge \mathit{len}(i)) \wedge \mathit{di}(\rhd f \wedge \mathit{len}(j)))$
by *simp*
hence *10*: $i{=}j$
using *FstLenSame* **by** *blast*
have *11*: $\bigwedge j.\ (\sigma \models \mathit{di}(\rhd f \wedge \mathit{len}(j))) \longrightarrow (j{=}i)$
using *9 10* **using** *FstLenSame* **by** *auto*
thus $(\exists!\ k.\ (\sigma \models \mathit{di}(\ \ \rhd f \wedge \mathit{len}(k)\ )))$
using *11 5* **by** *blast*
qed
qed

lemma *DiImpExistsOneDiLenAndFst-1*:
$\vdash \mathit{di}\ f \longrightarrow (\exists!\ k.\ (\ \mathit{di}(\ \rhd f \wedge \mathit{len}(k))))$
using *Valid-def DiImpExistsOneDiLenAndFst* **by** *fastforce*

lemma *LFstAndDist-help*:
$(\sigma \models((\rhd f \wedge g1) \wedge \mathit{len}(k));h1\ \wedge ((\rhd f \wedge g2) \wedge \mathit{len}(k));h2) =$
$(\sigma \models(((\rhd f \wedge g1) \wedge (\rhd f \wedge g2)) \wedge \mathit{len}(k));(h1 \wedge h2)\ )$
using *LFixedAndDistr* **by** *fastforce*

lemma *LFstAndDist-help-1*:
$(\exists\ \ k.\ (\sigma \models((\rhd f \wedge g1) \wedge \mathit{len}(k));h1\ \wedge ((\rhd f \wedge g2) \wedge \mathit{len}(k));h2)) =$
$(\exists\ \ k.\ (\sigma \models(((\rhd f \wedge g1) \wedge (\rhd f \wedge g2)) \wedge \mathit{len}(k));(h1 \wedge h2)\ ))$
proof
assume *0*: $\exists\, k.\ \sigma \models\ \ ((\rhd\ f \wedge g1) \wedge \mathit{len}\ k)\ ;\ h1\ \ \wedge\ \ ((\rhd\ f \wedge g2) \wedge \mathit{len}\ k)\ ;\ h2$
obtain *k* **where** *1*: $\ \sigma \models\ \ ((\rhd\ f \wedge g1) \wedge \mathit{len}\ k)\ ;\ h1\ \ \wedge\ \ ((\rhd\ f \wedge g2) \wedge \mathit{len}\ k)\ ;\ h2$
using *0* **by** *auto*
hence *2*: $(\sigma \models(((\rhd f \wedge g1) \wedge (\rhd f \wedge g2)) \wedge \mathit{len}(k));(h1 \wedge h2))$
using *LFstAndDist-help* **by** *blast*
show $(\exists\ \ k.\ (\sigma \models(((\rhd f \wedge g1) \wedge (\rhd f \wedge g2)) \wedge \mathit{len}(k));(h1 \wedge h2)\ ))$
using *2* **by** *auto*
next
assume *3*: $(\exists\ \ k.\ (\sigma \models(((\rhd f \wedge g1) \wedge (\rhd f \wedge g2)) \wedge \mathit{len}(k));(h1 \wedge h2)\ ))$
obtain *k* **where** *4*: $(\sigma \models(((\rhd f \wedge g1) \wedge (\rhd f \wedge g2)) \wedge \mathit{len}(k));(h1 \wedge h2)\ )$
using *3* **by** *auto*
hence *5*: $(\sigma \models((\rhd f \wedge g1) \wedge \mathit{len}(k));h1\ \wedge ((\rhd f \wedge g2) \wedge \mathit{len}(k));h2)$

```
  using  LFstAndDist-help by blast
  show (∃ k. (σ ⊨((▷f ∧ g1) ∧ len(k));h1  ∧ ((▷f ∧ g2) ∧ len(k));h2))
  using 5 by auto
qed

lemma LFstAndDistrsem:
 (∀ σ. (σ ⊨((▷f ∧ g1);h1 ∧ (▷f ∧ g2);h2) = (▷f ∧ g1 ∧ g2);(h1 ∧ h2)))
proof
 fix σ
 show (σ ⊨((▷f ∧ g1);h1 ∧ (▷f ∧ g2);h2) = (▷f ∧ g1 ∧ g2);(h1 ∧ h2))
 proof −
  have 1: (σ ⊨(▷f ∧ g1);h1) = (∃ i. (σ ⊨((▷f ∧ g1) ∧ len(i));h1) )
  using AndExistsLenChop[of TEMP (▷ f ∧ g1)] by fastforce
  have 2: (σ ⊨(▷f ∧ g2);h2) = (∃ j. (σ ⊨((▷f ∧ g2) ∧ len(j));h2) )
  using AndExistsLenChop[of TEMP (▷f ∧ g2)] by fastforce
  have 3: (σ ⊨(▷f ∧ g1);h1 ∧ (▷f ∧ g2);h2) =
         ( (∃ i j. (σ ⊨((▷f ∧ g1) ∧ len(i));h1  ∧
                         ((▷f ∧ g2) ∧ len(j));h2) )
         )
  using 1 2 by auto
  have 4: ( (∃ i j. (σ ⊨((▷f ∧ g1) ∧ len(i));h1  ∧
                         ((▷f ∧ g2) ∧ len(j));h2) )
         ) =
         ( (∃ k. (σ ⊨((▷f ∧ g1) ∧ len(k));h1  ∧
                         ((▷f ∧ g2) ∧ len(k));h2) )
         )
  using FstLenSameChop by blast
  have 5: (∃ k. (σ ⊨((▷f ∧ g1) ∧ len(k));h1  ∧ ((▷f ∧ g2) ∧ len(k));h2)) =
           (∃ k. (σ ⊨(((▷f ∧ g1) ∧ (▷f ∧ g2)) ∧ len(k));(h1 ∧ h2) ))
  using LFstAndDist-help-1 by blast
  have 6 :  (∃ k. (σ ⊨(((▷f ∧ g1) ∧ (▷f ∧ g2)) ∧ len(k));(h1 ∧ h2) )) =
            (σ ⊨((▷f ∧ g1) ∧ (▷f ∧ g2));(h1 ∧ h2))
  using AndExistsLenChop[of TEMP ((▷ f ∧ g1) ∧ ▷ f ∧ g2)] by fastforce
  have 7 : (σ ⊨((▷f ∧ g1) ∧ (▷f ∧ g2));(h1 ∧ h2)) =
           (σ ⊨(▷f ∧ g1 ∧ g2);(h1 ∧ h2))
  by (simp add: chop-defs, auto)
  from 3 4 5 6 7 show ?thesis by auto
  qed
qed

lemma LFstAndDistr:
⊢ ((▷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*:
$\vdash ((\triangleright f \wedge g);h1 \wedge (\triangleright f \wedge g);h2) = (\triangleright f \wedge g);(h1 \wedge h2)$
proof −
have *1*: $\vdash ((\triangleright f \wedge g);h1 \wedge (\triangleright f \wedge g);h2) = (\triangleright f \wedge g \wedge g);(h1 \wedge h2)$ **using** *LFstAndDistr* **by** *blast*
have *2*: $\vdash (\triangleright f \wedge g \wedge g);(h1 \wedge h2) = (\triangleright f \wedge g);(h1 \wedge h2)$ **by** *auto*
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *LFstAndDistrC*:
$\vdash ((\triangleright f\ );h1 \wedge (\triangleright f\ );h2) = (\triangleright f);(h1 \wedge h2)$
proof −
have *1*: $\vdash ((\triangleright f \wedge \#True);h1 \wedge (\triangleright f \wedge \#True);h2) = (\triangleright f \wedge \#True \wedge \#True);(h1 \wedge h2)$
using *LFstAndDistr* **by** *blast*
have *2*: $\vdash (\triangleright f \wedge \#True);h1 = (\triangleright f\ );h1$
by *auto*
have *3*: $\vdash (\triangleright f \wedge \#True);h2 = (\triangleright f\ );h2$
by *auto*
have *4*: $\vdash (\triangleright f \wedge \#True \wedge \#True);(h1 \wedge h2) = (\triangleright f);(h1 \wedge h2)$
by *auto*
from *1 2 3 4* **show** *?thesis* **by** *auto*
qed

lemma *LFstAndDistrD*:
$\vdash (di(\triangleright f \wedge g1) \wedge di(\triangleright f \wedge g2)) = di(\triangleright f \wedge g1 \wedge g2)$
proof −
have *1*: $\vdash ((\triangleright f \wedge g1);\#True \wedge (\triangleright f \wedge g2);\#True) = (\triangleright f \wedge g1 \wedge g2);(\#True \wedge \#True)$
using *LFstAndDistr* **by** *blast*
have *2*: $\vdash (\triangleright f \wedge g1);\#True = di(\triangleright f \wedge g1)$
by (*simp add*: *di-d-def*)
have *3*: $\vdash (\triangleright f \wedge g2);\#True = di(\triangleright f \wedge g2)$
by (*simp add*: *di-d-def*)
have *4*: $\vdash (\triangleright f \wedge g1 \wedge g2);(\#True \wedge \#True) = di(\triangleright f \wedge g1 \wedge g2)$
by (*simp add*: *di-d-def*)
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *LstAndDistr*:
$\vdash (h1;(\triangleleft\ f \wedge g1) \wedge h2;(\triangleleft\ f \wedge g2)) = (h1 \wedge h2);(\triangleleft\ f \wedge g1 \wedge g2)$
proof −
have *1*: $\vdash ((\triangleright(f^r) \wedge g1^r);(h1^r) \wedge (\triangleright(f^r) \wedge (g2^r));(h2^r)) =$
$(\triangleright(f^r) \wedge (g1^r) \wedge (g2^r));((h1^r) \wedge (h2^r))$
using *LFstAndDistr* **by** *blast*
hence *2*: $\vdash ((\triangleright(f^r) \wedge g1^r);(h1^r) \wedge (\triangleright(f^r) \wedge (g2^r));(h2^r))^r =$
$((\triangleright(f^r) \wedge (g1^r) \wedge (g2^r));((h1^r) \wedge (h2^r)))^r$
using *1 REqvRule* **by** *blast*
have *3*: $\vdash\ (((\triangleright(f^r) \wedge g1^r);(h1^r))^r \wedge ((\triangleright(f^r) \wedge (g2^r));(h2^r))^r) =$
$((\triangleright(f^r) \wedge g1^r);(h1^r) \wedge (\triangleright(f^r) \wedge (g2^r));(h2^r))^r$

using *RAnd* **by** *fastforce*
have *4*: $\vdash ((h1^r);(\rhd(f^r) \wedge g1^r)^r \wedge (h2^r);(\rhd(f^r) \wedge (g2^r))^r) =$
$(((\rhd(f^r) \wedge g1^r);(h1^r))^r \wedge ((\rhd(f^r) \wedge (g2^r));(h2^r))^r)$

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*: $\vdash (\rhd(f^r) \wedge g1^r)^r = ((\rhd(f^r))^r \wedge (g1^r)^r)$
using *RAnd* **by** *blast*
have *11*: $\vdash (\rhd(f^r) \wedge g2^r)^r = ((\rhd(f^r))^r \wedge (g2^r)^r)$
using *RAnd* **by** *blast*
have *12*: $\vdash (\rhd(f^r))^r = \lhd(f)$
using *RRFirstEqvLast* **by** *blast*
have *13*: $\vdash ((\rhd(f^r))^r \wedge (g1^r)^r) = (\lhd f \wedge g1)$
using *12 7* **by** *fastforce*
have *14*: $\vdash ((\rhd(f^r))^r \wedge (g2^r)^r) = (\lhd f \wedge g2)$
using *12 8* **by** *fastforce*
have *15*: $\vdash (h1;(\lhd f \wedge g1) \wedge h2;(\lhd f \wedge g2)) =$
$((h1^r);(\rhd(f^r) \wedge g1^r)^r \wedge (h2^r);(\rhd(f^r) \wedge (g2^r))^r)$

using *14 13 10 11 5 6* **by** (*metis 4 int-eq*)
have *16*: $\vdash (((\rhd(f^r) \wedge (g1^r) \wedge (g2^r));((h1^r) \wedge (h2^r)))^r) =$
$((h1^r) \wedge (h2^r))^r;((\rhd(f^r)) \wedge (g1^r) \wedge (g2^r))^r$
by (*simp add*: *RevChop*)
have *17*: $\vdash ((\rhd(f^r)) \wedge (g1^r) \wedge (g2)^r)^r = ((\rhd(f^r))^r \wedge (g1^r)^r \wedge (g2^r)^r)$
by (*metis inteq-reflection rev-fun2*)
have *18*: $\vdash ((\rhd(f^r))^r \wedge (g1^r)^r \wedge (g2^r)^r) = (\lhd f \wedge g1 \wedge g2)$
using *12 7 8* **by** *fastforce*
have *19*: $\vdash ((h1^r) \wedge (h2^r))^r = (h1 \wedge h2)$
using *RRAnd* **by** *auto*
have *20*: $\vdash ((h1^r) \wedge (h2^r))^r;((\rhd(f^r)) \wedge (g1^r) \wedge (g2^r))^r =$
$(h1 \wedge h2);(\lhd f \wedge g1 \wedge g2)$
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*:
$\vdash (h;(\lhd f \wedge g1) \wedge h;(\lhd f \wedge g2)) = h;(\lhd f \wedge g1 \wedge g2)$
proof −
have *1*: $\vdash (h;(\lhd f \wedge g1) \wedge h;(\lhd f \wedge g2)) = (h \wedge h);(\lhd f \wedge g1 \wedge g2)$
using *LstAndDistr* **by** *blast*
have *2*: $\vdash (h \wedge h);(\lhd f \wedge g1 \wedge g2) = h;(\lhd f \wedge g1 \wedge g2)$

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

lemma *LstAndDistrB*:
$\vdash\ (h1;(\lhd\ f \wedge g) \wedge h2;(\lhd\ f \wedge g)) = (h1 \wedge h2);(\lhd\ f \wedge g)$
proof −
have *1*: $\vdash (h1;(\lhd\ f \wedge g) \wedge h2;(\lhd\ f \wedge g)) = (h1 \wedge h2);(\lhd\ f \wedge g \wedge g)$
using *LstAndDistr* **by** *blast*
have *2*: $\vdash (h1 \wedge h2);(\lhd\ f \wedge g \wedge g) = (h1 \wedge h2);(\lhd\ f \wedge g)$
by *auto*
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *LstAndDistrC*:
$\vdash\ (h1;(\lhd\ f\ ) \wedge h2;(\lhd\ f\ )) = (h1 \wedge h2);(\lhd\ f\ )$
proof −
have *1*: $\vdash (h1;(\lhd\ f \wedge \#True) \wedge h2;(\lhd\ f \wedge \#True)) = (h1 \wedge h2);(\lhd\ f \wedge \#True \wedge \#True\ \ )$
using *LstAndDistr* **by** *blast*
have *2*: $\vdash (h1 \wedge h2);(\lhd\ f \wedge \#True \wedge \#True\ \ ) = (h1 \wedge h2);(\lhd\ f\ )$
by *auto*
have *3*: $\vdash h1;(\lhd\ f \wedge \#True) = h1;(\lhd\ f\ )$
by *auto*
have *4*: $\vdash h2;(\lhd\ f \wedge \#True) = h2;(\lhd\ f\ )$
by *auto*
from *1 2 3 4* **show** *?thesis* **by** *auto*
qed

lemma *LstAndDistrD*:
$\vdash\ (\Diamond(\lhd\ f \wedge g1) \wedge \Diamond(\lhd\ f \wedge g2)) = \Diamond(\lhd\ f \wedge g1 \wedge g2)$
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*: $\vdash ((\triangleright f;\ (\neg g)) \wedge (\triangleright f;g)) = \triangleright f;(\neg g \wedge g)$ **using** *LFstAndDistrC* **by** *blast*
have *7*: $\vdash \neg(\triangleright f;(\neg g \wedge g))$ **by** (*simp add*: *FstChopFalseEqvFalse*)
have *8*: $\vdash \triangleright f;\ (\neg g) \longrightarrow \neg(\triangleright f;g)$ **using** *5 6 7* **by** *fastforce*
have *9*: $\vdash \neg(di(\triangleright f)) \vee (\triangleright f;\ (\neg g)) \longrightarrow \neg(\triangleright f;g)$ **using** *4 8* **by** *fastforce*
have *10*: $\vdash di(\triangleright f) \vee \neg(di(\triangleright f))$ **by** *auto*
hence *11*: $\vdash (\triangleright f;\#True) \vee \neg(di(\triangleright f))$ **by** (*simp add*: *di-d-def*)
hence *12*: $\vdash (\triangleright f;(g \vee \neg g)) \vee \neg(di(\triangleright f))$ **by** *auto*
have *13*: $\vdash (\triangleright f;(g \vee \neg g)) = ((\triangleright f;g) \vee (\triangleright f;(\neg g)))$ **using** *ChopOrEqv* **by** *fastforce*
have *14*: $\vdash ((\triangleright f;g) \vee (\triangleright f;(\neg g))) \vee \neg(di(\triangleright f))$ **using** *12 13* **by** *fastforce*
hence *15*: $\vdash \neg(\triangleright f;g) \longrightarrow \neg(di(\triangleright f)) \vee (\triangleright f;\ (\neg g))$ **by** *auto*
from *9 15* **show** *?thesis* **by** *fastforce*
qed

lemma *BsNotFstChop*:
$\vdash bs(\neg(\triangleright f;g)) = (empty \vee \neg(di(\triangleright f)) \vee (\triangleright f;bs(\neg g)))$
proof −
have *1*: $\vdash bs(\neg(\triangleright f;g)) = (empty \vee bi(\neg(\triangleright f;g));skip)$
by (*simp add*:*bs-d-def*)
have *2*: $\vdash (empty \vee bi(\neg(\triangleright f;g));skip) = (empty \vee (\neg(di(\triangleright f;g)));skip)$
by (*metis 1 NotDiEqvBiNot int-eq*)
have *3*: $\vdash (empty \vee (\neg(di(\triangleright f;g)));skip) = (empty \vee (\neg((\triangleright f;g);\#True));skip)$
by (*simp add*: *di-d-def*)
have *4*: $\vdash\ (\neg((\triangleright f;g);\#True));skip =\ (\neg(\triangleright f;(g;\#True)));skip$
by (*metis ChopAssocB LeftChopEqvChop int-simps*(*15*) *inteq-reflection*)
hence *5*: $\vdash (empty \vee (\neg((\triangleright f;g);\#True));skip) =\ (empty \vee (\neg(\triangleright f;(g;\#True)));skip)$
by *auto*
have *6*: $\vdash (empty \vee (\neg(\triangleright f;(g;\#True)));skip) = (empty \vee (\neg(\triangleright f;di(g)));skip)$
by (*simp add*: *di-d-def*)
have *7*: $\vdash (empty \vee (\neg(\triangleright f;di(g)));skip) = (empty \vee \neg(\neg((\neg(\triangleright f;di(g)));skip)))$
by *auto*
have *8*: $\vdash \neg(\neg((\neg(\triangleright f;di(g)));skip)) =\ (\neg(empty \vee (\triangleright f;di(g));skip\ ))$
using *NotNotChopSkip* **by** *fastforce*
hence *9*: $\vdash (empty \vee \neg(\neg((\neg(\triangleright f;di(g)));skip))) =\ (empty \vee \neg(empty \vee (\triangleright f;di(g));skip\ ))$
by *auto*
have *10*: $\vdash (empty \vee \neg(empty \vee (\triangleright f;di(g));skip\ )) = (empty \vee (more \wedge \neg((\triangleright f;di(g));skip\ )))$
by (*meson 6 7 9 NotChopSkipEqvMoreAndNotChopSkip Prop04 Prop06*)
have *11*: $\vdash (empty \vee (more \wedge \neg((\triangleright f;di(g));skip\ ))) = (empty \vee \neg((\triangleright f;di(g));skip\ ))$
by (*simp add*: *empty-d-def*, *auto*)
have *12*: $\vdash\ (empty \vee \neg((\triangleright f;di(g));skip\ )) =\ (empty \vee \neg(\triangleright f;(di(g);skip)\ ))$
using *ChopAssocB 11* **by** *fastforce*
have *13*: $\vdash\ (\neg(\triangleright f;(di(g);skip)\ )) =\ (\neg(\triangleright f;(ds(g))\ ))$
using *DsDi* **using** *RightChopEqvChop* **by** *fastforce*
hence *14*: $\vdash (empty \vee \neg(\triangleright f;(di(g);skip)\ )) =\ (empty \vee \neg(\triangleright f;(ds(g))\ ))$
by *auto*
have *15*: $\vdash (empty \vee \neg(\triangleright f;(ds(g))\ )) = (empty \vee \neg(di\ (\triangleright f)) \vee (\triangleright f;(\neg(ds\ g))))$
using *NotFstChop* **by** *fastforce*
have *16*: $\vdash\ (\triangleright f;(\neg(ds\ g)))=\ \ (\triangleright f;(bs\ (\neg g)))$
using *NotDsEqvBsNot RightChopEqvChop* **by** *blast*
hence *17*: $\vdash ((empty \vee \neg(di\ (\triangleright f))) \vee (\triangleright f;(\neg(ds\ g)))) = ((empty \vee \neg(di\ (\triangleright f))) \vee\ (\triangleright f;(bs\ (\neg g))))$
by *auto*

from *1 2 3 5 6 7 9 10 11 12 14 15 17* **show** *?thesis* **by** *fastforce*
qed

lemma *FstFstChopEqvFstChopFst*:
$\vdash \rhd(\rhd f;g) = \rhd f;\rhd g$
proof −
have *1*: $\vdash \rhd(\rhd f;g) = ((\rhd f;g) \wedge bs\ (\neg(\rhd f;g)))$
by (*simp add*: *first-d-def*)
have *2*: $\vdash bs\ (\neg(\rhd f;g)) = (empty \vee \neg(di(\rhd f)) \vee (\rhd f;bs(\neg g)))$
using *BsNotFstChop* **by** *auto*
hence *3*: $\vdash ((\rhd f;g) \wedge bs\ (\neg(\rhd f;g))) = ((\rhd f;g) \wedge (empty \vee \neg(di(\rhd f)) \vee (\rhd f;bs(\neg g))))$
by *auto*
have *4*: $\vdash ((\rhd f;g) \wedge (empty \vee \neg(di(\rhd f)) \vee (\rhd f;bs(\neg g)))) =$
$(((\rhd f;g) \wedge empty) \vee ((\rhd f;g) \wedge \neg(di(\rhd f))) \vee ((\rhd f;g) \wedge (\rhd f;bs(\neg g))))$
by *auto*
have *5*: $\vdash \neg((\rhd f;g) \wedge \neg(di(\rhd f)))$
using *ChopImpDi* **by** *fastforce*
hence *6*: $\vdash (((\rhd f;g) \wedge empty) \vee ((\rhd f;g) \wedge \neg(di(\rhd f))) \vee ((\rhd f;g) \wedge (\rhd f;bs(\neg g)))) =$
$(((\rhd f;g) \wedge empty) \vee ((\rhd f;g) \wedge (\rhd f;bs(\neg g))))$
by *auto*
have *7*: $\vdash\ ((\rhd f;g) \wedge (\rhd f;(bs(\neg g)))) = ((\rhd f;(g \wedge (bs(\neg g)))))$
using *LFstAndDistrC* **by** *blast*
hence *8*: $\vdash (((\rhd f;g) \wedge empty) \vee ((\rhd f;g) \wedge (\rhd f;(bs(\neg g))))) =$
$(((\rhd f;g) \wedge empty) \vee ((\rhd f;(g \wedge (bs(\neg g))))))$
by *auto*
have *9*: $\vdash (((\rhd f;g) \wedge empty) \vee ((\rhd f;(g \wedge (bs(\neg g)))))) = (((\rhd f;g) \wedge empty) \vee \rhd f;\rhd g)$
by (*simp add*: *first-d-def*)
have *10*: $\vdash ((\rhd f;g) \wedge empty) = ((\rhd f;\rhd g) \wedge empty)$
using *FstChopEmptyEqvFstChopFstEmpty* **by** *blast*
hence *11*: $\vdash (((\rhd f;g) \wedge empty) \vee \rhd f;\rhd g) = (((\rhd f;\rhd g) \wedge empty) \vee \rhd f;\rhd g)$
by *auto*
have *12*: $\vdash (((\rhd f;\rhd g) \wedge empty) \vee \rhd f;\rhd g) = \rhd f;\rhd g$
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*: $\vdash \rhd f = (\rhd f);empty$ **using** *ChopEmpty* **by** (*metis int-eq*)
hence *2*: $\vdash \rhd(\ \rhd f\ ) = \rhd((\rhd f);empty)$ **using** *FstEqvRule* **by** *blast*
have *3*: $\vdash \rhd((\rhd f);empty) = \rhd f;\rhd empty$ **using** *FstFstChopEqvFstChopFst* **by** *auto*
have *4*: $\vdash \rhd f;\rhd empty = \rhd f;empty$ **using** *FstEmpty* **using** *RightChopEqvChop* **by** *blast*
have *5*: $\vdash \rhd f;empty = \rhd f$ **using** *ChopEmpty* **by** *blast*
from *2 3 4 5* **show** *?thesis* **by** *fastforce*
qed

lemma *FstCSEqvEmpty*:
$\vdash \rhd(f^\star) = empty$
proof −

have *1*: $\vdash \rhd(f^\star) = \rhd(empty \vee ((f \wedge more);f^\star))$ **using** *ChopstarEqv FstEqvRule* **by** *blast*
from *1* **show** *?thesis* **using** *FstEmptyOrEqvEmpty* **by** *fastforce*
qed

lemma *FstIterFixFst*:
$\vdash power\ (\rhd\ f)\ n = \rhd(power\ (\rhd\ f)\ n)$
proof
(*induct n*)
case *0*
then show *?case*
proof −
have *1*: $\vdash power\ (\rhd\ f)\ 0 = empty$ **by** *auto*
have *2*: $\vdash empty = \rhd\ empty$ **using** *FstEmpty* **by** *auto*
have *3*: $\vdash \rhd\ empty = \rhd(power\ (\rhd f)\ 0)$ **by** *auto*
from *1 2 3* **show** *?thesis* **by** *auto*
qed
next
case (*Suc n*)
then show *?case*
proof −
have *4*: $\vdash (power\ (\rhd\ f)\ (Suc\ n)) = (\rhd\ f)\ ;(power\ (\rhd f)\ n)$
by (*simp*)
have *5*: $\vdash (\rhd\ f)\ ;(power\ (\rhd f)\ n) = (\rhd\ f)\ ;\ \rhd\ (power\ (\rhd\ f)\ n)$
using *RightChopEqvChop Suc.hyps* **by** *blast*
have *6*: $\vdash (\rhd\ f)\ ;\ \rhd\ (power\ (\rhd\ f)\ n) = \rhd(\rhd f;(power\ (\rhd\ f)\ n))$
using *FstFstChopEqvFstChopFst* **by** *fastforce*
have *7*: $\vdash \rhd(\rhd f;(power\ (\rhd\ f)\ n)) = \rhd(power\ (\rhd\ f)\ (Suc\ n))$
by *simp*
from *4 5 6 7* **show** *?thesis* **by** *fastforce*
qed
qed

lemma *DsImpNotFst*:
$\vdash ds\ f \longrightarrow (\neg(\rhd f))$
proof −
have *1*: $\vdash (ds\ f \wedge \rhd f) = (ds\ f \wedge (f \wedge bs\ (\neg\ f)))$ **by** (*simp add*: *first-d-def*)
have *2*: $\vdash (ds\ f \wedge (f \wedge bs\ (\neg\ f))) = (ds\ f \wedge f \wedge \neg(ds\ f))$ **using** *NotDsEqvBsNot* **by** *fastforce*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FstLenAndEqvLenAnd*:
$\vdash \rhd(len(k) \wedge f) = (len(k) \wedge f)$
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*:
$\vdash \rhd f \longrightarrow (di\ f = f)$
proof −
have *1*: $\vdash \rhd f \longrightarrow \neg(di\ f;skip)$ **using** *FstImpNotDiChopSkip* **by** *blast*
have *2*: $\vdash di\ f \longrightarrow f \vee (di\ f;skip)$ **using** *DiEqvOrDiChopSkipB* **by** *fastforce*
have *3*: $\vdash \rhd f \wedge di\ f \longrightarrow (f \vee (di\ f;skip)) \wedge \neg(di\ f;skip)$ **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*: $\vdash \ \rhd f \wedge di\ f \longrightarrow f \wedge \neg(di\ f;skip)$ **using** *3 4* **by** *fastforce*
hence *6*: $\vdash \ \rhd f \wedge di\ f \longrightarrow f$ **by** *fastforce*
hence *7*: $\vdash \rhd f \longrightarrow (di\ f \longrightarrow f)$ **using** *FstAndElimL* **by** *fastforce*
have *8*: $\vdash f \longrightarrow di\ f$ **using** *DiIntro* **by** *auto*
hence *9*: $\vdash \rhd f \longrightarrow (f \longrightarrow (di\ f))$ **by** *auto*
from *7 9* **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndDiFstAndEqvFstAnd*:
$\vdash (\rhd f \wedge \ di(\rhd f \wedge g)) = (\rhd f \wedge g)$
proof −
have *1*: $\vdash \rhd f \wedge \ di(\rhd f \wedge g) \longrightarrow \rhd f$
by *auto*
have *2*: $\vdash \rhd f \wedge \ di(\rhd f \wedge g) \longrightarrow di(\rhd f \wedge g)$
by *auto*
have *3*: $\vdash di(\rhd f \wedge g) = ((\rhd f \wedge g) \vee di((\rhd f \wedge g);skip))$
using *DiEqvOrDiChopSkipA* **by** *blast*
have *4*: $\vdash di((\rhd f \wedge g);skip) = ((\rhd f \wedge g);skip);\#True$
by (*simp add*: *di-d-def*)
have *5*: $\vdash \rhd f \wedge \ di(\rhd f \wedge g) \longrightarrow (\rhd f \wedge g) \vee ((\rhd f \wedge g);skip);\#True$
using *2 3 4* **by** *fastforce*
have *6*: $\vdash \rhd f \wedge g \longrightarrow f$
using *FstAndElimL* **by** *fastforce*
hence *7*: $\vdash ((\rhd f \wedge g);skip);\#True \ \longrightarrow (f;skip);\#True$
by (*simp add*: *LeftChopImpChop*)
hence *8*: $\vdash ((\rhd f \wedge g);skip);\#True \ \longrightarrow di(f;skip)$
by (*simp add*: *di-d-def*)
have *9*: $\vdash \rhd f \longrightarrow \neg(di(f;skip))$
using *FstImpNotDiChopSkipB* **by** *blast*
have *10*: $\vdash \rhd f \wedge \ di(\rhd f \wedge g) \longrightarrow ((\rhd f \wedge g) \vee di(f;skip))$
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*: $\vdash ((\triangleright f \vee g) \wedge bs(\neg(\triangleright f)) \wedge bs\ (\neg\ g)) =$
$((\triangleright f \vee g) \wedge bs\ (\neg(\triangleright f) \wedge \neg\ g))$
using *BsAndEqv* **by** *fastforce*
have *13*: $\vdash\ (\neg(\triangleright f) \wedge \neg\ g) = (\neg(\triangleright f \vee g))$
by *auto*
hence *14*: $\vdash bs\ (\neg(\triangleright f) \wedge \neg\ g) = bs\ (\neg(\triangleright f \vee g))$
using *BsEqvRule* **by** *blast*
hence *15*: $\vdash ((\triangleright f \vee g) \wedge bs\ (\neg(\triangleright f) \wedge \neg\ g)) = ((\triangleright f \vee g) \wedge bs\ (\neg(\triangleright f \vee g)))$
by *auto*
have *16*: $\vdash ((\triangleright f \vee g) \wedge bs\ (\neg(\triangleright f \vee g))) = \triangleright(\triangleright f \vee g)$
by (*simp add*: *first-d-def*)
from *16 15 12 11 10 9 8 7 6 5 1* **show** *?thesis* **by** (*metis int-eq*)
qed

lemma *FstFstOrEqvFstOrR*:
$\vdash \triangleright(f \vee \triangleright g) = \triangleright(f \vee g)$
proof −
have *1*: $\vdash (f \vee \triangleright g) = (\triangleright g \vee f)$ **by** *auto*
hence *2*: $\vdash \triangleright\ (f \vee \triangleright g) = \triangleright(\triangleright g \vee f)$ **using** *FstEqvRule* **by** *blast*
have *3*: $\vdash \triangleright(\triangleright g \vee f) = \triangleright(g \vee f)$ **using** *FstFstOrEqvFstOrL* **by** *blast*
have *4*: $\vdash (g \vee f) = (f \vee g)$ **by** *auto*
hence *5*: $\vdash \triangleright(g \vee f) = \triangleright(f \vee g)$ **using** *FstEqvRule* **by** *blast*
from *2 3 5* **show** *?thesis* **by** *fastforce*
qed

lemma *FstFstOrEqvFstOr*:
$\vdash \triangleright(\triangleright f \vee \triangleright g) = \triangleright(f \vee g)$
proof −
have *1*: $\vdash\ \triangleright(\triangleright f \vee \triangleright g) = \triangleright(f \vee \triangleright g)$ **using** *FstFstOrEqvFstOrL* **by** *blast*
have *2*: $\vdash \triangleright(f \vee \triangleright g) = \triangleright(f \vee g)$ **using** *FstFstOrEqvFstOrR* **by** *blast*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *FstLenEqvLen*:
$\vdash \triangleright(\ len(k)) = len(k)$
proof −

have *1*: $\vdash \rhd(\ len(k) \wedge \#True) = (len(k) \wedge \#True)$ **using** *FstLenAndEqvLenAnd* **by** *blast*
have *2*: $\vdash (len(k) \wedge \#True) = len(k)$ **by** *auto*
hence *3*: $\vdash \rhd(\ len(k) \wedge \#True) = \rhd(len(k))$ **using** *FstEqvRule* **by** *blast*
from *1 2 3* **show** *?thesis* **by** *auto*
qed

lemma *FstSkip*:
$\vdash \rhd\ skip = skip$
proof −
have *1*: $\vdash skip = len(1)$ **using** *LenOneEqvSkip* **by** *fastforce*
hence *2*: $\vdash \rhd skip = \rhd\ (len(1))$ **using** *FstEqvRule* **by** *blast*
have *3*: $\vdash \rhd(\ len(1)) = len(1)$ **using** *FstLenEqvLen* **by** *blast*
from *1 2 3* **show** *?thesis* **using** *LenOneEqvSkip* **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* = $\triangleright$*skip* ; *f* **using** *FstSkip* **using** *LeftChopEqvChop* **by** *fastforce*
have *3*: $\vdash$ $\triangleright$(*skip* ; *f*) = $\triangleright$ ($\triangleright$*skip* ; *f*) **using** *2 FstEqvRule LeftChopEqvChop* **by** *blast*
have *4*: $\vdash$ $\triangleright$ ($\triangleright$*skip* ; *f*) = $\triangleright$*skip* ; $\triangleright$*f* **using** *3 FstFstChopEqvFstChopFst* **by** *blast*
have *5*: $\vdash$ $\triangleright$*skip* ; $\triangleright$*f* = *skip* ; $\triangleright$*f* **using** *4 FstSkip LeftChopEqvChop* **by** *blast*
have *6*: $\vdash$ *skip* ; $\triangleright$*f* = $\bigcirc$($\triangleright$ *f*) **by** (*simp add*: *next-d-def*)
from *1 2 3 4 5 6* **show** *?thesis* **by** *fastforce*
qed

lemma *FstDiamondStateEqvHalt*:
$\vdash$ $\triangleright$ ($\Diamond$ (*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*: $\vdash$ $\triangleright$ ($\Diamond$ (*init w*)) = $\triangleright$ (*di* (*fin* (*init w*))) **using** *4 FstEqvRule* **by** *blast*
hence *6*: $\vdash$ $\triangleright$ ($\Diamond$ (*init w*)) = $\triangleright$ (*fin* (*init w*)) **using** *FstDiEqvFst* **by** *fastforce*
hence *7*: $\vdash$ $\triangleright$ ($\Diamond$ (*init w*)) = *halt* (*init w*) **using** *HaltStateEqvFstFinState* **by** *fastforce*
from *7* **show** *?thesis* **by** *simp*
qed

lemma *FstBoxStateEqvStateAndEmpty*:
$\vdash$ $\triangleright$ ($\Box$ (*init w*)) = ((*init w*) $\wedge$ *empty*)
proof −
have *1*: $\vdash$ ((*init w*) $\wedge$ ($\Box$ (*init w*))*) = $\Box$ (*init w*)
using *BoxCSEqvBox* **by** *blast*
have *2*: $\vdash$ $\Box$ (*init w*) = ((*init w*) $\wedge$ ($\Box$ (*init w*))*)
using *1* **by** *auto*
hence *3*: $\vdash$ $\Box$ (*init w*) = ((*init w*) $\wedge$ ($\Box$ (*init w*))*)
by *blast*
have *4*: $\vdash$ ((*init w*) $\wedge$ *empty*) ; ($\Box$ (*init w*))* = ((*init w*) $\wedge$ ($\Box$ (*init w*))*)
using *StateAndEmptyChop* **by** *blast*
have *5*: $\vdash$ ((*init w*) $\wedge$ ($\Box$ (*init w*))*) = ((*init w*) $\wedge$ *empty*) ; ($\Box$ (*init w*))*
using *4* **by** *fastforce*
have *6*: $\vdash$ $\Box$ (*init w*) = ((*init w*) $\wedge$ *empty*) ; ($\Box$ (*init w*))*
using *3 5* **by** *fastforce*
have *7*: $\vdash$ ((*init w*) $\wedge$ *empty*) ; ($\Box$ (*init w*))* = $\triangleright$ (*init w*) ; ($\Box$ (*init w*))*
using *FstState* **by** (*metis AndChopCommute int-eq*)
have *8*: $\vdash$ $\Box$ (*init w*) = $\triangleright$ (*init w*) ; ($\Box$ (*init w*))*
using *6 7* **by** *fastforce*
have *9*: $\vdash$ $\triangleright$ ($\Box$ (*init w*)) = $\triangleright$ ($\triangleright$ (*init w*) ; ($\Box$ (*init w*))*)
using *8 FstEqvRule* **by** *blast*
have *10*: $\vdash$ $\triangleright$ ($\triangleright$ (*init w*) ; ($\Box$ (*init w*))*) = $\triangleright$ (*init w*) ; $\triangleright$ (($\Box$ (*init w*))*)
using *FstFstChopEqvFstChopFst* **by** *blast*
have *11*: $\vdash$ $\triangleright$ (*init w*) ; $\triangleright$ (($\Box$ (*init w*))*) = $\triangleright$ (*init w*) ; *empty*
using *RightChopEqvChop FstCSEqvEmpty* **by** *blast*
have *12*: $\vdash$ $\triangleright$ (*init w*) ; *empty* = $\triangleright$ (*init w*)
using *RightChopEqvChop ChopEmpty* **by** *blast*
have *13*: $\vdash$ $\triangleright$ (*init w*) = ((*init w*) $\wedge$ *empty*)
using *FstState* **by** *fastforce*

from *9 10 11 12 13* **show** *?thesis* **by** *fastforce*
qed

lemma *FstAndFstStarEqvFst*:
$\vdash (\triangleright f \wedge (\triangleright f)^\star) = \triangleright f$
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 *HaltStateEqvFstHaltState*:
$\vdash halt(init(w)) = \triangleright(halt(init(w)))$
proof −
have *1*: $\vdash halt\ (init\ w) = \triangleright\ (fin\ (init\ w))$
by (*simp add*: *HaltStateEqvFstFinState*)
have *2*: $\vdash \triangleright\ (fin\ (init\ w)) = \triangleright\ (\triangleright\ (fin\ (init\ w)))$
using *FstEqvRule FstFixFst* **by** *fastforce*
have *3*: $\vdash \triangleright\ (\triangleright\ (fin\ (init\ w))) = \triangleright(halt(init(w)))$
using *FstEqvRule HaltStateEqvFstFinState* **by** *fastforce*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *DiHaltAndDiHaltAndEqvDiHaltAndAnd*:
$\vdash (di(halt\ (init\ w) \wedge f) \wedge di(halt\ (init\ w) \wedge g)) = di(halt\ (init\ w) \wedge f \wedge g)$
proof −
have *1*: $\vdash (di(halt\ (init\ w) \wedge f) \wedge di(halt\ (init\ w) \wedge g)) =$

$(di(\triangleright(fin\ (init\ w)) \wedge f) \wedge di\ (\triangleright(fin\ (init\ w)) \wedge g))$
using *HaltStateEqvFstFinState* **by** (*metis LFstAndDistrD inteq-reflection*)
have *2*: $\vdash (di(\triangleright(fin\ (init\ w)) \wedge f) \wedge di(\triangleright(fin\ (init\ w)) \wedge g)) =$
$di(\triangleright(fin\ (init\ w)) \wedge f \wedge g)$
using *LFstAndDistrD* **by** *fastforce*
have *3*: $\vdash di(\triangleright(fin\ (init\ w)) \wedge f \wedge g) = di(halt\ (init\ w) \wedge f \wedge g)$
using *HaltStateEqvFstFinState* **by** (*metis DiEqvDi int-eq lift-and-com*)
from *1 2 3* **show** *?thesis* **using** *int-eq* **by** *metis*
qed

lemma *counter-ex-lhs*:
$\vdash ((\triangleright(len(5)) \wedge \triangleright(len(2)))\ ;\ (len(5) \vee len(2))) = \#False$
proof −
have *1*: $\vdash ((\triangleright(len(5)) \wedge \triangleright(len(2)))\ ;\ (len(5) \vee len(2))) =$
$(len(5) \wedge len(2));\ (len(5) \vee len(2))$
by (*metis FstLenAndEqvLenAnd FstLenEqvLen LeftChopEqvChop inteq-reflection*)
have *2*: $\vdash (len(5) \wedge len(2)) = \#False$
by (*simp add*: *Valid-def len-defs*)
have *3*: $\vdash ((len(5) \wedge len(2));\ (len(5) \vee len(2))) = (\#False;(len(5) \vee len(2)))$
by (*simp add*: *2 LeftChopEqvChop*)
have *4*: $\vdash (\#False;(len(5) \vee len(2))) = \#False$
by (*simp add*: *Valid-def chop-defs*)
from *1 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *counter-ex-rhs*:
$\vdash ((\triangleright\ (len(5))\ ;\ (len(5) \vee len(2))) \wedge (\triangleright\ (len(2))\ ;\ (len(5) \vee len(2)))) = len(7)$
proof −
have *1*: $\vdash (\triangleright\ (len(5))\ ;\ (len(5) \vee len(2))) =$
$len(5);(len(5) \vee len(2))$
using *FstLenEqvLen LeftChopEqvChop* **by** *blast*
have *2*: $\vdash (\triangleright\ (len(2))\ ;\ (len(5) \vee len(2))) =$
$len(2)\ ;(len(5) \vee len(2))$
using *FstLenEqvLen LeftChopEqvChop* **by** *blast*
have *3*: $\vdash len(5);(len(5) \vee len(2)) =$
$((len(5);len(5)) \vee (len(5);len(2)))$
by (*simp add*: *ChopOrEqv*)
have *4*: $\vdash ((len(5);len(5)) \vee (len(5);len(2))) =$
$(len(10) \vee len(7))$
using *LenEqvLenChopLen inteq-reflection* **by** *fastforce*
have *5*: $\vdash len(2)\ ;(len(5) \vee len(2)) =$
$((len(2);len(5)) \vee (len(2);len(2)))$
by (*simp add*: *ChopOrEqv*)
have *6*: $\vdash ((len(2);len(5)) \vee (len(2);len(2))) =$
$(len(7) \vee len(4))$
using *LenEqvLenChopLen inteq-reflection* **by** *fastforce*
have *7*: $\vdash ((len(10) \vee len(7)) \wedge (len(7) \vee len(4))) =$
$((len(7) \vee len(10)) \wedge (len(7) \vee len(4)))$

by *fastforce*
have *8*: $\vdash ((len(7) \vee len(10)) \wedge (len(7) \vee len(4))) = (len(7) \vee (len(10) \wedge len(4)))$
by *fastforce*
have *9*: $\vdash (len(10) \wedge len(4)) = \#False$
by (*simp add*: *Valid-def len-defs*)
have *10* : $\vdash (len(7) \vee (len(10) \wedge len(4))) = len(7)$
using *9* **by** *auto*
have *11*: $\vdash ((\triangleright (len(5)) ; (len(5) \vee len(2))) \wedge (\triangleright (len(2)) ; (len(5) \vee len(2)))) = (len(5);(len(5) \vee len(2)) \wedge len(2) ;(len(5) \vee len(2)))$
using *1 2* **by** *fastforce*
have *12*: $\vdash (len(5);(len(5) \vee len(2)) \wedge len(2) ;(len(5) \vee len(2))) = len(7)$
using *10 3 4 5 6*
by *fastforce*
from *11 12* **show** *?thesis* **by** *fastforce*
qed

end

theory *Monitor*
imports *First*

begin

7 Monitors

The RV monitors language is introduced plus the algebraic properties of the monitor operators.

7.1 Syntax

datatype ('*a* ::*world*) *monitor* =
mFIRST-d '*a formula* ((*FIRST* -) [*84*] *83*)
| *mUPTO-d* '*a monitor* '*a monitor* ((- *UPTO* -) [*84*,*84*] *83*)
| *mTHRU-d* '*a monitor* '*a monitor* ((- *THRU* -) [*84*,*84*] *83*)
| *mTHEN-d* '*a monitor* '*a monitor* ((- *THEN* -) [*84*,*84*] *83*)
| *mWITH-d* '*a monitor* '*a formula* ((- *WITH* -) [*84*,*84*] *83*)

fun *MON* :: ('*a*::*world*) *monitor* ⇒ '*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* ((*M* -) [*80*] *80*)

translations
-MON == *CONST MON*

definition *eq-d* :: (*'a*:: *world*) *monitor* ⇒ *'a monitor* ⇒ *bool* ((- ≃ -) [*84*,*84*] *83*)
where
eq-d a b ≡ (⊢ (*M a*) = (*M b*))

lemma *MonEqRefl*:
a ≃ *a*
by (*simp add*: *eq-d-def*)

lemma *MonEqSym*:
assumes *a* ≃ *b*
shows *b* ≃ *a*
using *assms* **by** (*metis eq-d-def inteq-reflection*)

lemma *MonEqTrans*:
assumes *a* ≃ *b*
b ≃ *c*
shows *a* ≃ *c*
using *assms*(*1*) *assms*(*2*) **by** (*metis eq-d-def inteq-reflection*)

lemma *MonEq*:
(*a* ≃ *b*) = (⊢ (*M a*) = (*M b*))
by (*simp add*: *eq-d-def*)

lemma *MonEqSubstWith*:
assumes *a* ≃ *b*
shows (*a WITH f*) ≃ (*b WITH f*)
using *assms* **by** (*metis MON.simps*(*5*) *eq-d-def inteq-reflection lift-and-com*)

lemma *MonEqSubstThen*:
assumes *a1* ≃ *b1*
a2 ≃ *b2*
shows (*a1 THEN a2*) ≃ (*b1 THEN b2*)
using *assms*(*1*) *assms*(*2*) **by** (*simp add*: *ChopEqvChop eq-d-def*)

lemma *MonEqSubstUpto*:
assumes *a1* ≃ *b1*
a2 ≃ *b2*
shows (*a1 UPTO a2*) ≃ (*b1 UPTO b2*)
using *assms*(*1*) *assms*(*2*) **by** (*metis* (*mono-tags*, *lifting*) *MON.simps*(*2*) *eq-d-def int-eq MonEqRefl*)

lemma *MonEqSubstThru*:
assumes *a1* ≃ *b1*
a2 ≃ *b2*
shows (*a1 THRU a2*) ≃ (*b1 THRU b2*)
using *assms*(*1*) *assms*(*2*) **by** (*metis* (*mono-tags*, *lifting*) *MON.simps*(*3*) *eq-d-def int-eq MonEqRefl*)

7.2 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*(*M b*))

definition *ITERATE-d* :: (*'a* :: *world*) *monitor* ⇒ *'a monitor* ⇒ *'a monitor*
where
ITERATE-d a b ≡ (*a WITH* (*LIFT* (*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*)))))

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*

7.3 Monitor Laws

lemma *MFixFst*:
⊢ (ℳ *a*) = ▷ (ℳ *a*)
proof
(*induct a*)
case (*mFIRST-d x*)
then show *?case*
proof −
have *1*: ⊢ (ℳ (*FIRST x*)) = ▷ *x* **by** *simp*
have *2*: ⊢ ▷ *x* = ▷ (▷ *x*) **using** *FstFixFst* **by** *fastforce*
have *3*: ⊢ ▷ (▷ *x*) = ▷(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*: ⊢ (ℳ (*a1 UPTO a2*)) = ▷((ℳ *a1*) ∨ (ℳ *a2*))
by (*simp*)
have *2*: ⊢ ▷((ℳ *a1*) ∨ (ℳ *a2*)) = ▷(▷((ℳ *a1*) ∨ (ℳ *a2*)))
using *FstFixFst* **by** *fastforce*
have *3*: ⊢ ▷(▷((ℳ *a1*) ∨ (ℳ *a2*))) = ▷(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*: ⊢ (ℳ (*a1 THRU a2*)) = ▷(*di* (ℳ *a1*) ∧ *di*(ℳ *a2*))
by (*simp*)
have *2*: ⊢ ▷(*di* (ℳ *a1*) ∧ *di*(ℳ *a2*)) = ▷(▷(*di*(ℳ *a1*) ∧ *di*(ℳ *a2*)))
using *FstFixFst* **by** *fastforce*
have *3*: ⊢ ▷(▷(*di* (ℳ *a1*) ∧ *di*(ℳ *a2*))) = ▷(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*: ⊢ (ℳ (*a1 THEN a2*)) = (ℳ *a1*) ; (ℳ *a2*)
by (*simp*)

```
  have 2: ⊢ (M a1) ; (M a2) = ▷(M a1) ; ▷(M a2)
      using ChopEqvChop mTHEN-d.hyps(1) mTHEN-d.hyps(2) by blast
  have 3: ⊢ ▷(M a1) ; ▷(M a2) = ▷(▷(M a1) ; (M a2))
      using FstFstChopEqvFstChopFst by fastforce
  have 4: ⊢ ▷(▷(M a1) ; (M a2)) = ▷((M a1) ; (M a2))
      using FstEqvRule LeftChopEqvChop mTHEN-d.hyps(1) by (metis inteq-reflection)
  have 5: ⊢ ▷((M a1) ; (M a2)) = ▷(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: ⊢ (M (a WITH x2)) = ((M a) ∧ ( x2))
      by (simp )
  have 2: ⊢ ((M a) ∧ ( x2)) = (▷(M a) ∧ ( x2))
      using mWITH-d.hyps by fastforce
  have 3: ⊢ (▷(M a) ∧ ( x2)) = ▷(▷(M a) ∧ ( x2))
      using FstFstAndEqvFstAnd by fastforce
  have 4: ⊢ ▷(▷(M a) ∧ ( x2)) = ▷((M a) ∧ ( x2))
      using 2 FstEqvRule by fastforce
  have 5: ⊢ ▷((M a) ∧ ( x2)) = ▷(M (a WITH x2))
      using 4 by simp
  from 1 2 3 4 5 show ?thesis by (metis inteq-reflection)
 qed
qed

lemma MGuardFalseEqvFalse:
⊢ M(GUARD #False) = #False
proof −
 have 1: ⊢ M(GUARD #False) = M(EMPTY WITH LIFT(init #False)) by (simp add: GUARD-d-def)
 have 2: ⊢ M(EMPTY WITH LIFT(init #False)) = (M(EMPTY) ∧ (init #False)) by (simp )
 have 3: ⊢ #False = (init #False) by (simp add:init-defs Valid-def)
 have 4: ⊢ (M(EMPTY) ∧ (init #False)) = (M(EMPTY) ∧ #False) using 3 by auto
 have 5: ⊢ (M(EMPTY) ∧ #False) = #False by simp
 have 6: ⊢ (M(EMPTY) ∧ (init #False)) = #False using 4 5 by simp
 have 7: ⊢ M(EMPTY WITH LIFT(init #False)) = #False using 2 6 by fastforce
 have 8: ⊢ M(GUARD #False) = #False using 1 7 by fastforce
 from 8 show ?thesis by auto
qed

lemma MFirstFalseEqvFalse:
⊢ M(FIRST LIFT #False) = #False
proof −
 have 1: ⊢ M(FIRST LIFT #False) = ▷ #False by (simp )
 have 2: ⊢ M(FIRST LIFT #False) = #False using FstFalse by fastforce
from 2 show ?thesis by auto
qed
```

lemma *MFailAlt*:
$\vdash \mathcal{M}\ FAIL = \#False$
proof −
have *1*: $\vdash \mathcal{M}\ FAIL = \mathcal{M}\ (GUARD\ (\#False))$ **by** (*simp add*: *FAIL-d-def*)
have *2*: $\vdash \mathcal{M}(GUARD\ (\#False)) = \#False$ **using** *MGuardFalseEqvFalse* **by** *auto*
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *MFailEqvFirstFalseWithinEmpty*:
$FAIL \simeq ((FIRST\ LIFT\ \#False)\ WITHIN\ empty)$
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* **using** *MonEq* **by** (*metis int-eq*)
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*: $\vdash$ $(\mathit{empty} \wedge (\ \mathit{init}\ w)) = (\mathit{empty} \wedge \mathit{init}\ w)$ **by** *simp*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed

lemma *MLengthAlt*:
$\vdash \mathcal{M}\ (\mathit{LEN}(k)) = \mathit{len}(k)$
proof −
have *1*: $\vdash \mathcal{M}(\mathit{LEN}(k)) = \mathcal{M}(\mathit{FIRST\ LIFT}(\mathit{len}(k)))$ **by** (*simp add*:*LEN-d-def*)
have *2*: $\vdash \mathcal{M}(\mathit{FIRST\ LIFT}(\mathit{len}(k))) = \triangleright(\mathit{len}(k))$ **by** (*simp*)
have *3*: $\vdash \triangleright(\mathit{len}(k)) = \mathit{len}(k)$ **using** *FstLenEqvLen* **by** *blast*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MAlwaysAlt*:
$\vdash \mathcal{M}(a\ \mathit{ALWAYS}\ w) = (\mathcal{M}(a) \wedge \Box\ (\mathit{init}\ w))$
proof −
have *1*: $\vdash \mathcal{M}(a\ \mathit{ALWAYS}\ w) = \mathcal{M}(a\ \mathit{WITH\ LIFT}(\mathit{bi}\ (\mathit{fin}\ (\mathit{init}\ w))))$
by (*simp add*: *ALWAYS-d-def*)
have *2*: $\vdash \mathcal{M}(a\ \mathit{WITH\ LIFT}(\mathit{bi}\ (\mathit{fin}\ (\mathit{init}\ w)))) = (\mathcal{M}(a) \wedge (\mathit{bi}\ (\mathit{fin}\ (\mathit{init}\ w))))$
by (*simp*)
have *3*: $\vdash (\mathcal{M}(a) \wedge (\mathit{bi}\ (\mathit{fin}\ (\mathit{init}\ w)))) = (\mathcal{M}(a) \wedge \Box\ (\mathit{init}\ w))$
using *BoxStateEqvBiFinState* **by** *fastforce*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MSometimeAlt*:
$\vdash \mathcal{M}(a\ \mathit{SOMETIME}\ w) = (\mathcal{M}(a) \wedge \Diamond\ (\mathit{init}\ w))$
proof −
have *1*: $\vdash \mathcal{M}(a\ \mathit{SOMETIME}\ w) = \mathcal{M}(a\ \mathit{WITH\ LIFT}(\mathit{di}\ (\mathit{fin}\ (\mathit{init}\ w))))$
by (*simp add*: *SOMETIME-d-def*)
have *2*: $\vdash \mathcal{M}(a\ \mathit{WITH\ LIFT}(\mathit{di}\ (\mathit{fin}\ (\mathit{init}\ w)))) = (\mathcal{M}(a) \wedge (\mathit{di}\ (\mathit{fin}\ (\mathit{init}\ w))))$
by (*simp*)
have *3*: $\vdash \mathcal{M}(a\ \mathit{WITH\ LIFT}(\mathit{di}\ (\mathit{fin}\ (\mathit{init}\ w)))) = (\mathcal{M}(a) \wedge\ \Diamond\ (\mathit{init}\ w))$
using *DiamondStateEqvDiFinState* **by** *fastforce*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MWithinAlt*:
$\vdash \mathcal{M}(a\ \mathit{WITHIN}\ f) = (\mathcal{M}(a) \wedge (\mathit{bs}\ (\neg\ f)))$
proof −
have *1*: $\vdash \mathcal{M}(a\ \mathit{WITHIN}\ f) = \mathcal{M}(a\ \mathit{WITH\ LIFT}(\mathit{bs}\ (\neg\ f)))$
by (*simp add*: *WITHIN-d-def LIMIT-d-def*)
have *2*: $\vdash \mathcal{M}(a\ \mathit{WITH\ LIFT}(\mathit{bs}\ (\neg\ f))) = (\mathcal{M}(a) \wedge (\mathit{bs}\ (\neg\ f)))$
by (*simp*)
from *1 2* **show** *?thesis* **by** *fastforce*
qed

lemma *MTimesAlt*:
$\vdash \mathcal{M}(a\ \mathit{TIMES}\ k) = \mathit{power}\ (\mathcal{M}(a))\ k$

proof
(*induct k*)
case *0*
then show *?case*
proof −
have *1*: ⊢ $\mathcal{M}$ (*a TIMES 0*) = $\mathcal{M}$ *EMPTY* **by** *simp*
have *2*: ⊢ $\mathcal{M}$ *EMPTY* = *empty* **using** *MEmptyAlt* **by** *simp*
have *3*: ⊢ *empty* = *power* ($\mathcal{M}$ *a*) *0* **by** *simp*
from *1 2 3* **show** *?thesis* **by** *auto*
qed
next
case (*Suc k*)
then show *?case*
proof −
have *1*: ⊢ $\mathcal{M}$(*a TIMES Suc k*) = $\mathcal{M}$(*a THEN* (*a TIMES k*))
by *simp*
have *2*: ⊢ $\mathcal{M}$(*a THEN* (*a TIMES k*)) = ($\mathcal{M}$ *a*);($\mathcal{M}$ (*a TIMES k*))
by (*simp*)
have *3*: ⊢ ($\mathcal{M}$ *a*);($\mathcal{M}$(*a TIMES k*)) = ($\mathcal{M}$ *a*);(*power* ($\mathcal{M}$ *a*) *k*)
using *RightChopEqvChop Suc.hyps* **by** *blast*
have *4*: ⊢ ($\mathcal{M}$ *a*);(*power* ($\mathcal{M}$ *a*) *k*) = *power* ($\mathcal{M}$ *a*) (*Suc k*)
by *simp*
from *1 2 3 4* **show** *?thesis* **by** *fastforce*
qed
qed

lemma *MUptoAlt*:
⊢ $\mathcal{M}$(*a UPTO b*) = ((($\mathcal{M}$ *a*) ∧ *bi* (¬($\mathcal{M}$ *b*))) ∨ (($\mathcal{M}$ *b*) ∧ *bi* (¬($\mathcal{M}$ *a*))) ∨ (($\mathcal{M}$ *a*) ∧ ($\mathcal{M}$ *b*)))
proof −
have *1*: ⊢ $\mathcal{M}$ (*a UPTO b*) = ▷(($\mathcal{M}$ *a*) ∨ ($\mathcal{M}$ *b*))
by (*simp*)
have *2*: ⊢ ▷(($\mathcal{M}$ *a*) ∨ ($\mathcal{M}$ *b*)) = ((▷($\mathcal{M}$ *a*) ∧ (*bs* (¬($\mathcal{M}$ *b*)))) ∨ (▷($\mathcal{M}$ *b*) ∧ (*bs* (¬($\mathcal{M}$ *a*)))))
using *FstWithOrEqv* **by** *blast*
have *3*: ⊢ ((▷($\mathcal{M}$ *a*) ∧ (*bs* (¬($\mathcal{M}$ *b*)))) ∨ (▷($\mathcal{M}$ *b*) ∧ (*bs* (¬($\mathcal{M}$ *a*))))) =
((($\mathcal{M}$ *a*) ∧ (($\mathcal{M}$ *b*) ∨ ¬($\mathcal{M}$ *b*)) ∧ (*bs* (¬($\mathcal{M}$ *b*)))) ∨
(($\mathcal{M}$ *b*) ∧ (($\mathcal{M}$ *a*) ∨ ¬($\mathcal{M}$ *a*)) ∧ (*bs* (¬($\mathcal{M}$ *a*)))))
using *MFixFst* **by** *fastforce*
have *4*: ⊢ ((($\mathcal{M}$ *a*) ∧ (($\mathcal{M}$ *b*) ∨ ¬($\mathcal{M}$ *b*)) ∧ (*bs* (¬($\mathcal{M}$ *b*)))) ∨
(($\mathcal{M}$ *b*) ∧ (($\mathcal{M}$ *a*) ∨ ¬($\mathcal{M}$ *a*)) ∧ (*bs* (¬($\mathcal{M}$ *a*))))) =
((($\mathcal{M}$ *a*) ∧ ((($\mathcal{M}$ *b*) ∧ *bs* (¬($\mathcal{M}$ *b*))) ∨ (¬($\mathcal{M}$ *b*) ∧ *bs* (¬($\mathcal{M}$ *b*))))) ∨
(($\mathcal{M}$ *b*) ∧ ((($\mathcal{M}$ *a*) ∧ *bs* (¬($\mathcal{M}$ *a*))) ∨ (¬($\mathcal{M}$ *a*) ∧ *bs* (¬($\mathcal{M}$ *a*))))))
by *auto*
have *5*: ⊢ ((($\mathcal{M}$ *a*) ∧ ((($\mathcal{M}$ *b*) ∧ *bs* (¬($\mathcal{M}$ *b*))) ∨ (¬($\mathcal{M}$ *b*) ∧ *bs* (¬($\mathcal{M}$ *b*))))) ∨
(($\mathcal{M}$ *b*) ∧ ((($\mathcal{M}$ *a*) ∧ *bs* (¬($\mathcal{M}$ *a*))) ∨ (¬($\mathcal{M}$ *a*) ∧ *bs* (¬($\mathcal{M}$ *a*)))))) =
((($\mathcal{M}$ *a*) ∧ ((▷($\mathcal{M}$ *b*)) ∨ (¬($\mathcal{M}$ *b*) ∧ *bs* (¬($\mathcal{M}$ *b*))))) ∨
(($\mathcal{M}$ *b*) ∧ ((▷($\mathcal{M}$ *a*)) ∨ (¬($\mathcal{M}$ *a*) ∧ *bs* (¬($\mathcal{M}$ *a*))))))
by (*simp add*: *first-d-def*)
have *6*: ⊢ ((($\mathcal{M}$ *a*) ∧ ((▷($\mathcal{M}$ *b*)) ∨ (¬($\mathcal{M}$ *b*) ∧ *bs* (¬($\mathcal{M}$ *b*))))) ∨
(($\mathcal{M}$ *b*) ∧ ((▷($\mathcal{M}$ *a*)) ∨ (¬($\mathcal{M}$ *a*) ∧ *bs* (¬($\mathcal{M}$ *a*)))))) =
((($\mathcal{M}$ *a*) ∧ ((($\mathcal{M}$ *b*)) ∨ (¬($\mathcal{M}$ *b*) ∧ *bs* (¬($\mathcal{M}$ *b*))))) ∨

$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a)) \vee (\neg(\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))))\ ))$
using *MFixFst* **by** *fastforce*
have *7*: $\vdash (\neg(\mathcal{M}\ b) \wedge bs\ (\neg(\mathcal{M}\ b))) = bi(\neg(\mathcal{M}\ b))$
using *AndBsEqvBi* **by** *blast*
have *8*: $\vdash (\neg(\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))) = bi(\neg(\mathcal{M}\ a))$
using *AndBsEqvBi* **by** *blast*
have *9*: $\vdash (((\mathcal{M}\ a) \wedge (\ ((\mathcal{M}\ b)) \vee ((\neg(\mathcal{M}\ b)) \wedge bs(\neg(\mathcal{M}\ b))))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a)) \vee ((\neg(\mathcal{M}\ a)) \wedge bs(\neg(\mathcal{M}\ a))))\ )) =$
$(((\mathcal{M}\ a) \wedge (\ ((\mathcal{M}\ b)) \vee (\ bi(\neg(\mathcal{M}\ b)))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a)) \vee (\ bi(\neg(\mathcal{M}\ a))))\ ))$
using *7 8* **by** *fastforce*
have *10*: $\vdash (((\mathcal{M}\ a) \wedge (\ ((\mathcal{M}\ b)) \vee (\ bi(\neg(\mathcal{M}\ b)))\ ) \vee$
$((\mathcal{M}\ b) \wedge (\ ((\mathcal{M}\ a)) \vee (\ bi(\neg(\mathcal{M}\ a))))\ )) =$
$(((\ (\mathcal{M}\ a) \wedge (\mathcal{M}\ b)) \vee (\ (\mathcal{M}\ a) \wedge bi(\neg(\mathcal{M}\ b)))) \vee$
$((\ (\mathcal{M}\ b) \wedge (\mathcal{M}\ a)) \vee (\ (\mathcal{M}\ b) \wedge bi(\neg(\mathcal{M}\ a)))))$
by *auto*
have *11*: $\vdash (((\ (\mathcal{M}\ a) \wedge (\mathcal{M}\ b)) \vee (\ (\mathcal{M}\ a) \wedge bi(\neg(\mathcal{M}\ b)))) \vee$
$((\ (\mathcal{M}\ b) \wedge (\mathcal{M}\ a)) \vee (\ (\mathcal{M}\ b) \wedge bi(\neg(\mathcal{M}\ a))))) =$
$((\ (\mathcal{M}\ a) \wedge bi\ (\neg(\mathcal{M}\ b))) \vee ((\mathcal{M}\ b) \wedge bi\ (\neg(\mathcal{M}\ a))) \vee ((\mathcal{M}\ a) \wedge (\mathcal{M}\ b)))$
by *auto*
from *1 2 3 4 5 6 9 10 11* **show** *?thesis* **by** (*metis int-eq*)
qed

lemma *MThruAlt*:
$\vdash \mathcal{M}(a\ THRU\ b) = (((\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee ((\mathcal{M}\ b) \wedge di(\mathcal{M}\ a)))$
proof −
have *1*: $\vdash \mathcal{M}(a\ THRU\ b) = \triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))$
by (*simp*)
have *2*: $\vdash \triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) = ((\triangleright(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee (\triangleright(\mathcal{M}\ b) \wedge di(\mathcal{M}\ a)))$
using *FstDiAndDiEqv* **by** *auto*
have *3*: $\vdash ((\triangleright(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee (\triangleright(\mathcal{M}\ b) \wedge di(\mathcal{M}\ a))) =$
$(((\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee ((\mathcal{M}\ b) \wedge di(\mathcal{M}\ a)))$
using *MFixFst* **by** *fastforce*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MHaltAlt*:
$\vdash \mathcal{M}(HALT\ w) = halt(init\ w)$
proof −
have *1*: $\vdash \mathcal{M}(HALT\ w) = \mathcal{M}(FIRST\ LIFT(fin\ (init\ \ w)))$ **by** (*simp add*: *HALT-d-def*)
have *2*: $\vdash \mathcal{M}(FIRST\ LIFT(fin\ (init\ \ w))) = \triangleright\ (fin\ (init\ \ w))$ **by** (*simp*)
have *3*: $\vdash \triangleright\ (fin\ (init\ \ w)) = halt(init\ w)$ **using** *HaltStateEqvFstFinState* **by** *fastforce*
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma *MFailUpto*:
$(FAIL\ UPTO\ a) \simeq (\ a)$
proof −
have *1*: $\vdash\ \mathcal{M}(FAIL\ UPTO\ a) = \triangleright(\ (\mathcal{M}\ FAIL) \vee (\mathcal{M}\ a))$ **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* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MFailThru*:
$(FAIL\ THRU\ (\ a)) \simeq \ FAIL$
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*: $\vdash \rhd(di\ (\#False) \wedge di(\mathcal{M}\ a)) = \rhd(\ (\#False) \wedge di(\mathcal{M}\ a))$
by (*metis 2 inteq-reflection*)
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* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MFailAnd*:
$(FAIL\ AND\ a) \simeq \ FAIL$
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* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThenFail*:
$(a\ THEN\ FAIL) \simeq FAIL$
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* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MFailThen*:
$(\ FAIL\ THEN\ a) \simeq \ FAIL$

proof −
have *1*: ⊢ $\mathcal{M}$(*FAIL THEN a*) = ($\mathcal{M}$ *FAIL*);($\mathcal{M}$ *a*) **by** (*simp*)
have *2*: ⊢ ($\mathcal{M}$ *FAIL*);($\mathcal{M}$ *a*) = #*False*;($\mathcal{M}$ *a*) **using** *MFailAlt* **using** *LeftChopEqvChop* **by** *blast*
have *3*: ⊢ #*False*;($\mathcal{M}$ *a*) = #*False* **by** (*simp add*: *chop-d-def Valid-def*)
have *4*: ⊢ #*False* = $\mathcal{M}$ *FAIL* **using** *MFailAlt* **by** *auto*
from *1 2 3 4* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MFailWith*:
(*FAIL WITH f*) ≃ *FAIL*
proof −
have *1*: ⊢ $\mathcal{M}$ (*FAIL WITH f*) = (($\mathcal{M}$ *FAIL*) ∧ *f*) **by** (*simp*)
have *2*: ⊢ (($\mathcal{M}$ *FAIL*) ∧ *f*) = (#*False* ∧ *f*) **using** *MFailAlt* **by** *auto*
have *3*: ⊢ (#*False* ∧ *f*) = #*False* **by** *simp*
have *4*: ⊢ #*False* = $\mathcal{M}$ *FAIL* **using** *MFailAlt* **by** *auto*
from *1 2 3 4* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MWithFalse*:
(*a WITH* (*LIFT*(#*False*))) ≃ *FAIL*
proof −
have *1*: ⊢ $\mathcal{M}$ (*a WITH LIFT*(#*False*)) = (($\mathcal{M}$ *a*) ∧ #*False*) **by** (*simp*)
have *2*: ⊢ (($\mathcal{M}$ *a*) ∧ #*False*) = $\mathcal{M}$ *FAIL* **using** *MFailAlt* **by** *auto*
from *1 2* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MWithTrue*:
(*a WITH* (*LIFT*(#*True*))) ≃ *a*
proof −
have *1*: ⊢ $\mathcal{M}$ (*a WITH LIFT*(#*True*)) = (($\mathcal{M}$ *a*) ∧ #*True*) **by** (*simp*)
have *2*: ⊢ (($\mathcal{M}$ *a*) ∧ #*True*) = $\mathcal{M}$ *a* **by** *simp*
from *1 2* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MEmptyUpto*:
(*EMPTY UPTO a*) ≃ *EMPTY*
proof −
have *1*: ⊢ $\mathcal{M}$ (*EMPTY UPTO a*) = ▷($\mathcal{M}$ *EMPTY* ∨ ($\mathcal{M}$ *a*)) **by** (*simp*)
have *2*: ⊢ $\mathcal{M}$ *EMPTY* = *empty* **using** *MEmptyAlt* **by** *auto*
hence *3*: ⊢ ($\mathcal{M}$ *EMPTY* ∨ ($\mathcal{M}$ *a*)) = (*empty* ∨ ($\mathcal{M}$ *a*)) **by** *auto*
hence *4*: ⊢ ▷($\mathcal{M}$ *EMPTY* ∨ $\mathcal{M}$ *a*) = ▷(*empty* ∨ $\mathcal{M}$ *a*) **using** *FstEqvRule* **by** *blast*
have *5*: ⊢ ▷(*empty* ∨ $\mathcal{M}$ *a*) = *empty* **using** *FstEmptyOrEqvEmpty* **by** *blast*
have *6*: ⊢ *empty* = $\mathcal{M}$ *EMPTY* **using** *MEmptyAlt* **by** *auto*
from *1 4 5 6* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MEmptyThru*:
(*EMPTY THRU a*) ≃ (*a*)
proof −
have *1*: ⊢ $\mathcal{M}$(*EMPTY THRU a*) = ▷(*di*($\mathcal{M}$ *EMPTY*) ∧ *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* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThenEmpty*:
$(\ a\ THEN\ EMPTY) \simeq (a)$
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* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MEmptyThen*:
$(\ EMPTY\ THEN\ a) \simeq a$
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* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MEmptyIterate*:
$(\ EMPTY\ ITERATE\ b) \simeq\ EMPTY$
proof −
have *1*: $\vdash$ $\mathcal{M}(\ EMPTY\ ITERATE\ b) = \mathcal{M}(\ EMPTY\ WITH\ LIFT(\mathcal{M}\ b)^\star)$
by (*simp add*: *ITERATE-d-def*)
have *2*: $\vdash$ $\mathcal{M}\ (EMPTY\ WITH\ LIFT(\mathcal{M}\ b)^\star) = (\mathcal{M}\ EMPTY \wedge (\mathcal{M}\ b)^\star)$
by (*simp*)
have *3*: $\vdash$ $(\mathcal{M}\ EMPTY \wedge (\mathcal{M}\ b)^\star)\ = (empty \wedge (\mathcal{M}\ b)^\star)$
using *MEmptyAlt* **by** *auto*
have *4*: $\vdash$ $(empty \wedge (\mathcal{M}\ b)^\star)\ = (empty \wedge (empty \vee (((\mathcal{M}\ b) \wedge more);(\mathcal{M}\ b)^\star)))$
using *ChopstarEqv* **by** *fastforce*
have *5*: $\vdash$ $(empty \wedge (empty \vee (((\mathcal{M}\ b) \wedge more);(\mathcal{M}\ b)^\star))) = empty$
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* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MIterateIdemp*:
$(a\ ITERATE\ a) \simeq\ (a)$
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*: $\vdash\ \mathcal{M}(a\ \mathit{WITH}\ \mathit{LIFT}(\mathcal{M}\ a)^\star) = ((\mathcal{M}\ a) \wedge (\mathcal{M}\ a)^\star)$ **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* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MUptoIdemp*:
$(a\ \mathit{UPTO}\ a) \simeq\ (a)$
proof −
have *1*: $\vdash\ \mathcal{M}(a\ \mathit{UPTO}\ a) = \triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ a))$ **by** *auto*
have *2*: $\vdash \triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ a)) = \triangleright(\mathcal{M}\ a)$ **using** *FstEqvRule* **by** *fastforce*
have *3*: $\vdash \triangleright(\mathcal{M}\ a) = (\mathcal{M}\ a)$ **using** *MFixFst* **by** *fastforce*
from *1 2 3* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThruIdemp*:
$(\ a\ \mathit{THRU}\ a) \simeq (a)$
proof −
have *1*: $\vdash\ \mathcal{M}(\ a\ \mathit{THRU}\ a) = \triangleright(\ \mathit{di}(\mathcal{M}\ a) \wedge \mathit{di}(\mathcal{M}\ a))$ **by** *auto*
have *2*: $\vdash \triangleright(\ \mathit{di}(\mathcal{M}\ a) \wedge \mathit{di}(\mathcal{M}\ a)) = \triangleright(\mathit{di}\ (\mathcal{M}\ a))$ **using** *FstEqvRule* **by** *fastforce*
have *3*: $\vdash \triangleright(\mathit{di}\ (\mathcal{M}\ a)) = \triangleright(\mathcal{M}\ a)$ **using** *FstDiEqvFst* **by** *blast*
have *4*: $\vdash \triangleright(\mathcal{M}\ a) = (\mathcal{M}\ a)$ **using** *MFixFst* **by** *fastforce*
from *1 2 3 4* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MAndIdemp*:
$(a\ \mathit{AND}\ a) \simeq (a)$
proof −
have *1*: $\vdash \mathcal{M}(a\ \mathit{AND}\ a) = ((\mathcal{M}\ a) \wedge (\mathcal{M}\ a))$ **by** (*simp add*: *AND-d-def*)
have *2*: $\vdash ((\mathcal{M}\ a) \wedge (\mathcal{M}\ a)) = (\mathcal{M}\ a)$ **by** *fastforce*
from *1 2* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MWithIdemp*:
$(\ (a\ \mathit{WITH}\ f)\ \ \mathit{WITH}\ f) \simeq (a\ \mathit{WITH}\ f)$
proof −
have *1*: $\vdash \mathcal{M}(\ (a\ \mathit{WITH}\ f)\ \ \mathit{WITH}\ f) = (((\mathcal{M}\ a) \wedge (\ f)) \wedge (\ f))$ **by** *auto*
have *2*: $\vdash (((\mathcal{M}\ a) \wedge (\ f)) \wedge (\ f)) = ((\mathcal{M}\ a) \wedge (\ f))$ **by** *fastforce*
have *3*: $\vdash ((\mathcal{M}\ a) \wedge (\ f)) = \mathcal{M}(a\ \mathit{WITH}\ f)$ **by** *auto*
from *1 2 3* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MUptoCommut*:
$(a\ \mathit{UPTO}\ b) \simeq (b\ \mathit{UPTO}\ a)$
proof −
have *1*: $\vdash \mathcal{M}(a\ \mathit{UPTO}\ b) = \triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))$ **by** (*simp*)
have *2*: $\vdash ((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) = ((\mathcal{M}\ b) \vee (\mathcal{M}\ a))$ **by** *auto*
hence *3*: $\vdash \triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) = \triangleright((\mathcal{M}\ b) \vee (\mathcal{M}\ a))$ **using** *FstEqvRule* **by** *blast*

have *4*: $\vdash \rhd((\mathcal{M}\ b) \vee (\mathcal{M}\ a)) = \mathcal{M}(b\ UPTO\ a)$ **by** *auto*
from *1 3 4* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThruCommut*:
$(a\ THRU\ b) \simeq (b\ THRU\ a)$
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* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MAndCommut*:
$(a\ AND\ b) \simeq (b\ AND\ a)$
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* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MWithCommut*:
$((a\ WITH\ f)\ WITH\ g) \simeq ((a\ WITH\ g)\ WITH\ f)$
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 *3*: $\vdash (((\mathcal{M}\ a) \wedge (g)) \wedge (f)) = \mathcal{M}((a\ WITH\ g)\ WITH\ f)$ **by** *auto*
from *1 2 3* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MWithAbsorp*:
$((a\ WITH\ f)\ WITH\ g) \simeq (a\ WITH\ LIFT(f \wedge g))$
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** (*simp add*: *MonEq*)
qed

lemma *MUptoAssoc*:
$((a\ UPTO\ b)\ UPTO\ c) \simeq (a\ UPTO\ (b\ UPTO\ c))$
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*: $\vdash \triangleright(((\mathcal{M}\ a) \vee (\mathcal{M}\ b)) \vee (\mathcal{M}\ c)) = \triangleright((\mathcal{M}\ a) \vee ((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))$
using *FstEqvRule* **by** *blast*
have *6*: $\vdash \triangleright((\mathcal{M}\ a) \vee ((\mathcal{M}\ b) \vee (\mathcal{M}\ c))) = \triangleright((\mathcal{M}\ a) \vee \triangleright((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))$
using *FstFstOrEqvFstOrR* **by** *fastforce*
have *7*: $\vdash \triangleright((\mathcal{M}\ a) \vee \triangleright((\mathcal{M}\ b) \vee (\mathcal{M}\ c))) = \triangleright((\mathcal{M}\ a) \vee \mathcal{M}(b\ UPTO\ c))$
by *auto*
have *8*: $\vdash \triangleright((\mathcal{M}\ a) \vee \mathcal{M}(b\ UPTO\ c)) = \mathcal{M}(a\ UPTO\ (b\ UPTO\ c))$
by *auto*
from *1 2 3 5 6 7 8* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThruAssoc*:
$((a\ THRU\ b)\ THRU\ c) \simeq (a\ THRU\ (b\ THRU\ c))$
proof −
have *1*: $\vdash \mathcal{M}((a\ THRU\ b)\ THRU\ c) = \triangleright(di(\triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) \wedge di(\mathcal{M}\ c))$
by *auto*
have *2*: $\vdash di(\triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) = di((di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))$
using *DiEqvDiFst* **by** *fastforce*
have *3*: $\vdash di((di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) = (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))$
using *DiDiAndEqvDi* **by** *blast*
have *4*: $\vdash di(\triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) = (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))$
using *2 3* **by** *fastforce*
hence *5*: $\vdash (di(\triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))\ \wedge di(\mathcal{M}\ c)) = (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ \wedge di(\mathcal{M}\ c))$
by *auto*
have *6*: $\vdash (di(\mathcal{M}\ b)\ \wedge di(\mathcal{M}\ c)) = di\ (di(\mathcal{M}\ b)\ \wedge di(\mathcal{M}\ c))$
using *DiDiAndEqvDi* **by** *fastforce*
have *7*: $\vdash di\ (di(\mathcal{M}\ b)\ \wedge di(\mathcal{M}\ c)) = di\ (\triangleright(di(\mathcal{M}\ b)\ \wedge di(\mathcal{M}\ c)))$
using *DiEqvDiFst* **by** *blast*
have *8*: $\vdash (di(\mathcal{M}\ b)\ \wedge di(\mathcal{M}\ c)) = di\ (\triangleright(di(\mathcal{M}\ b)\ \wedge di(\mathcal{M}\ c)))$
using *6 7* **by** *fastforce*
hence *9*: $\vdash (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ \wedge di(\mathcal{M}\ c)) = (di(\mathcal{M}\ a) \wedge di\ (\triangleright(di(\mathcal{M}\ b)\ \wedge di(\mathcal{M}\ c))))$
by *auto*
have *10*: $\vdash (di(\triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))\ \wedge di(\mathcal{M}\ c)) =$
$(di(\mathcal{M}\ a) \wedge di\ (\triangleright(di(\mathcal{M}\ b)\ \wedge di(\mathcal{M}\ c))))$
using *5 9* **by** *fastforce*
hence *11*: $\vdash \triangleright(di(\triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))\ \wedge di(\mathcal{M}\ c)) =$
$\triangleright(di(\mathcal{M}\ a) \wedge di\ (\triangleright(di(\mathcal{M}\ b)\ \wedge di(\mathcal{M}\ c))))$
using *FstEqvRule* **by** *fastforce*
have *12*: $\vdash \triangleright(di(\mathcal{M}\ a) \wedge di\ (\triangleright(di(\mathcal{M}\ b)\ \wedge di(\mathcal{M}\ c)))) = \mathcal{M}(a\ THRU\ (b\ THRU\ c))$
by *auto*
from *1 11 12* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MAndAssoc*:
$((a\ AND\ b)\ AND\ c) \simeq (a\ AND\ (b\ AND\ c))$
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 eq-d-def*)
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* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThenAssoc*:
$((a\ THEN\ b)\ THEN\ c) \simeq (a\ THEN\ (b\ THEN\ c))$
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* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MUptoThruAbsorp*:
$(a\ UPTO\ (a\ THRU\ b)) \simeq\ a$
proof −
have *1*: $\vdash \mathcal{M}(a\ UPTO\ (a\ THRU\ b)) = \rhd((\mathcal{M}\ a) \vee \rhd(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ ))$
by *simp*
have *2*: $\vdash \rhd((\mathcal{M}\ a) \vee \rhd(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ )) =$
$\rhd((\mathcal{M}\ a) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ ))$
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*: $\vdash \rhd ((\mathcal{M}\ a) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)\ )) =$
$\rhd\ ((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))$
using *FstEqvRule* **by** *blast*
have *7*: $\vdash \rhd\ ((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b))) =$
$((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs\ (\neg(\ (di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))))$
by (*simp add*: *first-d-def*, *auto*)
have *8*: $\vdash ((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b))) =$
$((di(\mathcal{M}\ a) \wedge (\mathcal{M}\ a)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))$
by *auto*
hence *9*: $\vdash (\neg((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))) =$
$(\neg((di(\mathcal{M}\ a) \wedge (\mathcal{M}\ a)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
by *fastforce*
have *10*: $\vdash (\neg((di(\mathcal{M}\ a) \wedge (\mathcal{M}\ a)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$(\neg(((\mathcal{M}\ a)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *AndDiEqv* **using** *5* **by** *auto*
have *11*: $\vdash (\neg(((\mathcal{M}\ a)) \vee (di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$(\neg(\mathcal{M}\ a) \wedge \neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))$
by *auto*
have *12*: $\vdash (\neg((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))) =$

$(\neg(\mathcal{M}\ a) \wedge \neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))$
using *9 10 11* **by** *auto*
hence *13*: $\vdash$ $bs\ (\neg((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))) =$
$bs\ (\neg(\mathcal{M}\ a) \wedge \neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))$
using *BsEqvRule* **by** *blast*
have *14*: $\vdash$ $bs\ ((\neg(\mathcal{M}\ a)) \wedge \neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) =$
$(bs\ ((\neg(\mathcal{M}\ a))) \wedge bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *BsAndEqv* **by** *fastforce*
have *141*: $\vdash$ $bs\ (\neg((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))) =$
$(bs\ ((\neg(\mathcal{M}\ a))) \wedge bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *13 14* **by** *fastforce*
hence *15*: $\vdash$ $((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs\ (\neg((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b))))) =$
$((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs\ ((\neg(\mathcal{M}\ a))) \wedge bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
by *auto*
have *16*: $\vdash$ $((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs\ ((\neg(\mathcal{M}\ a))) \wedge bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$((bs\ ((\neg(\mathcal{M}\ a))) \wedge di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
by *auto*
have *17*: $\vdash$ $((bs\ ((\neg(\mathcal{M}\ a))) \wedge di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$((\triangleright(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *FstEqvBsNotAndDi* **by** *fastforce*
have *18*: $\vdash$ $((\triangleright(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$(((\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *MFixFst* **by** *fastforce*
have *19*: $\vdash$ $(((\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$(((\mathcal{M}\ a)) \wedge\ \ bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
by *auto*
have *20*: $\vdash$ $\ \ (\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) = (\neg(di(\mathcal{M}\ a)) \vee \neg(di(\mathcal{M}\ b))\ )$
by *auto*
have *21*: $\vdash$ $(\neg(di(\mathcal{M}\ a)) \vee \neg(di(\mathcal{M}\ b))\ ) = ((bi\ (\neg(\mathcal{M}\ a))) \vee (bi\ (\neg(\mathcal{M}\ b)))\ )$
by (*simp add*: *bi-d-def*)
have *22*: $\vdash$ $(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) = ((bi\ (\neg(\mathcal{M}\ a))) \vee (bi\ (\neg(\mathcal{M}\ b))))$
using *20 21* **by** *auto*
hence *23*: $\vdash$ $bs\ (\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) = bs\ ((bi\ (\neg(\mathcal{M}\ a))) \vee (bi(\neg(\mathcal{M}\ b))))$
using *BsEqvRule* **by** *blast*
have *24*: $\vdash$ $bs\ ((bi\ (\neg(\mathcal{M}\ a))) \vee (bi\ (\neg(\mathcal{M}\ b)))) = bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b))$
using *BsOrBsEqvBsBiOrBi* **by** *fastforce*
have *25*: $\vdash$ $bs\ (\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))) = (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b)))$
using *23 24* **using** *BsOrBsEqvBsBiOrBi* **by** *fastforce*
hence *26*: $\vdash$ $((\mathcal{M}\ a) \wedge bs\ (\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) =$
$((\mathcal{M}\ a) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b))))$
by *auto*

have *27*: $\vdash ((\mathcal{M}\ a) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b)))) =$
$(\triangleright(\mathcal{M}\ a) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b))))$
using *MFixFst* **by** *fastforce*
have *28*: $\vdash (\triangleright(\mathcal{M}\ a) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b)))) =$
$((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a)) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b))))$
by (*simp add*: *first-d-def*, *auto*)
have *29*: $\vdash ((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a)) \wedge (bs\ (\neg(\mathcal{M}\ a)) \vee bs\ (\neg(\mathcal{M}\ b)))) =$
$((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a)))$
by *auto*
have *30*: $\vdash ((\mathcal{M}\ a) \wedge bs\ (\neg(\mathcal{M}\ a))) = \triangleright(\mathcal{M}\ a)$
by (*simp add*: *first-d-def*)
have *31*: $\vdash \triangleright(\mathcal{M}\ a) = (\mathcal{M}\ a)$
using *MFixFst* **by** *fastforce*
have *32*: $\vdash \mathcal{M}(a\ UPTO\ (a\ THRU\ b)) =$
$((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs\ (\neg(\ (di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))))$
using *1 2 6 7* **by** *fastforce*
have *33*: $\vdash ((di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge$
$bs\ (\neg(\ (di(\mathcal{M}\ a)) \wedge ((\mathcal{M}\ a) \vee di(\mathcal{M}\ b))))) =$
$(((\mathcal{M}\ a)) \wedge\ \ bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b))))$
using *15 16 17 18 19* **by** (*metis int-eq*)
have *34*: $\vdash (((\mathcal{M}\ a)) \wedge\ \ bs(\neg(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)))) = (\mathcal{M}\ a)$
using *26 27 28 29 30 31* **by** (*metis int-eq*)
from *32 33 34* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThruUptoAbsorp*:
$(a\ THRU\ (a\ UPTO\ b)) \simeq (a)$
proof −
have *1*: $\vdash \mathcal{M}(a\ THRU\ (a\ UPTO\ b)) = \triangleright(di(\mathcal{M}\ a) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))))$
by *simp*
have *2*: $\vdash\ \ \triangleright(di(\mathcal{M}\ a) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b)))) =$
$\triangleright(di(\mathcal{M}\ a) \wedge di(((\mathcal{M}\ a) \vee (\mathcal{M}\ b))))$
by (*metis DiEqvDiFst FstEqvRule inteq-reflection lift-and-com*)
have *3*: $\vdash \triangleright(di(\mathcal{M}\ a) \wedge di(((\mathcal{M}\ a) \vee (\mathcal{M}\ b)))) =$
$\triangleright(di(\mathcal{M}\ a) \wedge (di(\mathcal{M}\ a) \vee di(\mathcal{M}\ b)))$
by (*metis DiOrEqv FstEqvRule inteq-reflection lift-and-com*)
have *4*: $\vdash (di(\mathcal{M}\ a) \wedge (di(\mathcal{M}\ a) \vee di(\mathcal{M}\ b))) = (di(\mathcal{M}\ a))$
by *auto*
hence *5*: $\vdash \triangleright(di(\mathcal{M}\ a) \wedge (di(\mathcal{M}\ a) \vee di(\mathcal{M}\ b))) = \triangleright(di(\mathcal{M}\ a))$
using *FstEqvRule* **by** *blast*
have *6*: $\vdash \triangleright(di(\mathcal{M}\ a)) = \triangleright(\mathcal{M}\ a)$
using *FstDiEqvFst* **by** *blast*
have *7*: $\vdash \triangleright(\mathcal{M}\ a) = (\mathcal{M}\ a)$
using *MFixFst* **by** *fastforce*
from *1 2 3 5 6 7* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MUptoThruDistrib*:
$(a\ UPTO\ (b\ THRU\ c)) \simeq ((a\ UPTO\ b)\ THRU\ (a\ UPTO\ c))$

proof −
have *1*: $\vdash \mathcal{M}((a\ \mathit{UPTO}\ b)\ \mathit{THRU}\ (a\ \mathit{UPTO}\ c)) =$
$\triangleright(\ di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ )$
by *simp*
have *2*: $\vdash (\ di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ ) =$
$(\ di(((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ )$
using *DiEqvDiFst* **by** *fastforce*
have *3*: $\vdash (\ di(((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ ) =$
$((\ di(\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge (di(\mathcal{M}\ a) \vee di(\mathcal{M}\ c)))$
using *DiOrEqv* **by** *fastforce*
have *4*: $\vdash ((\ di(\mathcal{M}\ a) \vee di(\mathcal{M}\ b)) \wedge (di(\mathcal{M}\ a) \vee di(\mathcal{M}\ c))) =$
$(di(\mathcal{M}\ a) \vee (di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)))$
by *auto*
have *5*: $\vdash (\ di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ ) =$
$(di(\mathcal{M}\ a) \vee (di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)))$
using *2 3 4* **by** *fastforce*
hence *6*: $\vdash \triangleright(\ di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ b))) \wedge di(\triangleright((\mathcal{M}\ a) \vee (\mathcal{M}\ c)))\ ) =$
$\triangleright(\ di(\mathcal{M}\ a) \vee (di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))\ )$
using *FstEqvRule* **by** *blast*
have *7*: $\vdash \triangleright(\ di(\mathcal{M}\ a) \vee (di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))\ ) =$
$\triangleright(\ \triangleright(di(\mathcal{M}\ a)) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))\ )$
using *FstFstOrEqvFstOr* **by** *fastforce*
have *8*: $\vdash \triangleright(di(\mathcal{M}\ a)) = \triangleright((\mathcal{M}\ a))$
using *FstDiEqvFst* **by** *blast*
have *9*: $\vdash \triangleright((\mathcal{M}\ a)) = (\mathcal{M}\ a)$
using *MFixFst* **by** *fastforce*
have *10*: $\vdash \triangleright(di(\mathcal{M}\ a)) = (\mathcal{M}\ a)$
using *8 9* **by** *fastforce*
hence *11*: $\vdash (\triangleright(di(\mathcal{M}\ a)) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))) =$
$((\mathcal{M}\ a) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)))$
by *auto*
hence *12*: $\vdash \triangleright(\triangleright(di(\mathcal{M}\ a)) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))) =$
$\triangleright((\mathcal{M}\ a) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)))$
using *FstEqvRule* **by** *blast*
have *13*: $\vdash \triangleright((\mathcal{M}\ a) \vee \triangleright(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))) = \mathcal{M}(a\ \mathit{UPTO}\ (b\ \mathit{THRU}\ c))$
by *simp*
from *1 6 7 12 13* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThruUptoDistrib*:
$(a\ \mathit{THRU}\ (b\ \mathit{UPTO}\ c)) \simeq ((a\ \mathit{THRU}\ b)\ \mathit{UPTO}\ (a\ \mathit{THRU}\ c))$
proof −
have *1*: $\vdash \mathcal{M}((a\ \mathit{THRU}\ b)\ \mathit{UPTO}\ (a\ \mathit{THRU}\ c)) =$
$\triangleright(\triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ b)) \vee \triangleright(di(\mathcal{M}\ a) \wedge di(\mathcal{M}\ c)))$
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* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThruUptoRDistrib*:
$((a\ THRU\ b)\ UPTO\ c) \simeq ((a\ UPTO\ c)\ THRU\ (b\ UPTO\ c))$
proof −
have *1*: $((a\ THRU\ b)\ UPTO\ c) \simeq (c\ UPTO\ (a\ THRU\ b))$
using *MUptoCommut* **by** *auto*
have *2*: $(c\ UPTO\ (a\ THRU\ b)) \simeq ((c\ UPTO\ a)\ THRU\ (c\ UPTO\ b))$
using *MUptoThruDistrib* **by** *auto*
have *3*: $(c\ UPTO\ a) \simeq (a\ UPTO\ c)$
using *MUptoCommut* **by** *auto*
have *4*: $(c\ UPTO\ b) \simeq (b\ UPTO\ c)$
using *MUptoCommut* **by** *auto*
have *5*: $((c\ UPTO\ a)\ THRU\ (c\ UPTO\ b)) \simeq ((a\ UPTO\ c)\ THRU\ (c\ UPTO\ b))$
using *3* **by** (*simp add*: *MonEqRefl MonEqSubstThru*)
have *6*: $((a\ UPTO\ c)\ THRU\ (c\ UPTO\ b)) \simeq ((c\ UPTO\ b)\ THRU\ (a\ UPTO\ c))$
using *MThruCommut* **by** *auto*
have *7*: $((c\ UPTO\ b)\ THRU\ (a\ UPTO\ c)) \simeq ((b\ UPTO\ c)\ THRU\ (a\ UPTO\ c))$
using *4* **by** (*simp add*: *MonEqRefl MonEqSubstThru*)
from *1 2 5 6 7* **show** *?thesis* **using** *MThruCommut MonEq* **by** (*metis int-eq*)
qed

lemma *MUptoThruRDistrib*:
$((a\ UPTO\ b)\ THRU\ c) \simeq ((a\ THRU\ c)\ UPTO\ (b\ THRU\ c))$
proof −
have *1*: $((a\ UPTO\ b)\ THRU\ c) \simeq (c\ THRU\ (a\ UPTO\ b))$
using *MThruCommut* **by** *auto*
have *2*: $(c\ THRU\ (a\ UPTO\ b)) \simeq ((c\ THRU\ a)\ UPTO\ (c\ THRU\ b))$
using *MThruUptoDistrib* **by** *auto*
have *3*: $(c\ THRU\ a) \simeq (a\ THRU\ c)$
using *MThruCommut* **by** *auto*
have *4*: $(c\ THRU\ b) \simeq (b\ THRU\ c)$
using *MThruCommut* **by** *auto*
have *5*: $((c\ THRU\ a)\ UPTO\ (c\ THRU\ b)) \simeq ((a\ THRU\ c)\ UPTO\ (c\ THRU\ b))$
using *3* **by** (*simp add*: *MonEqRefl MonEqSubstUpto*)
have *6*: $((a\ THRU\ c)\ UPTO\ (c\ THRU\ b)) \simeq ((c\ THRU\ b)\ UPTO\ (a\ THRU\ c))$
using *MUptoCommut* **by** *auto*
have *7*: $((c\ THRU\ b)\ UPTO\ (a\ THRU\ c)) \simeq ((b\ THRU\ c)\ UPTO\ (a\ THRU\ c))$
using *4* **by** (*simp add*: *MonEqRefl MonEqSubstUpto*)

from *1 2 5 6 7* **show** *?thesis* **using** *MUptoCommut MonEq* **by** (*metis int-eq*)
qed

lemma *MWithAndDistrib*:
$((a\ AND\ b)\ WITH\ f) \simeq ((a\ WITH\ f)\ AND\ (b\ WITH\ f))$
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* **using** *MonEq* **by** (*metis AND-d-def MWithAbsorp int-eq*)
qed

lemma *MHaltWithAndDistrib*:
$(((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g)) \simeq ((HALT\ w)\ WITH\ LIFT(f \wedge g))$
proof −
have *1*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g)) =$
$\mathcal{M}(((HALT\ w)\ WITH\ f)\ WITH\ LIFT(\mathcal{M}((HALT\ w)\ WITH\ g)))$
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* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MHaltWithUptoHaltWithEqvHaltWithOr*:
$(((HALT\ w)\ WITH\ f)\ UPTO\ ((HALT\ w)\ WITH\ g)) \simeq ((HALT\ w)\ WITH\ LIFT(f \vee g))$
proof −
have *1*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ UPTO\ ((HALT\ w)\ WITH\ g)) =$
$\triangleright(\mathcal{M}((HALT\ w)\ WITH\ f) \vee \mathcal{M}((HALT\ w)\ WITH\ g))$
by (*simp*)
have *2*: $\vdash \triangleright(\mathcal{M}((HALT\ w)\ WITH\ f) \vee \mathcal{M}((HALT\ w)\ WITH\ g)) =$
$\triangleright((\mathcal{M}(HALT\ w) \wedge f) \vee (\mathcal{M}(HALT\ w) \wedge g))$
by *auto*

have *3*: $\vdash ((\mathcal{M}(HALT\ w) \wedge f) \vee (\mathcal{M}(HALT\ w) \wedge g)) = (\mathcal{M}(HALT\ w) \wedge (f \vee g))$
by *auto*
have *4*: $\vdash \triangleright((\mathcal{M}(HALT\ w) \wedge f) \vee (\mathcal{M}(HALT\ w) \wedge g)) = \triangleright(\mathcal{M}(HALT\ w) \wedge (f \vee g))$
using *3 FstEqvRule* **by** *fastforce*
have *5*: $\vdash \triangleright(\mathcal{M}(HALT\ w) \wedge (f \vee g)) = \triangleright(\mathcal{M}((HALT\ w)\ WITH\ LIFT(f \vee g)))$
by *simp*
have *6*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ LIFT(f \vee g))) = \triangleright(\mathcal{M}((HALT\ w)\ WITH\ LIFT(f \vee g)))$
using *MFixFst* **by** *blast*
from *1 2 3 4 5 6* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MHaltWithThruHaltWithEqvHaltWithAndHaltWith*:
$(((HALT\ w)\ WITH\ f)\ THRU\ ((HALT\ w)\ WITH\ g)) \simeq (((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g))$
proof −
have *1*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ THRU\ ((HALT\ w)\ WITH\ g)) =$
$\triangleright(\ di(\mathcal{M}(HALT\ w) \wedge f) \wedge di(\mathcal{M}(HALT\ w) \wedge g)\ )$
by *simp*
have *2*: $\vdash (di(\mathcal{M}(HALT\ w) \wedge f) \wedge di(\mathcal{M}(HALT\ w) \wedge g)) =$
$(di(halt(init\ w) \wedge f) \wedge di(halt(init\ w) \wedge g))$
using *MHaltAlt DiEqvDi*
by (*metis* (*no-types*, *lifting*) *inteq-reflection lift-and-com*)
have *3*: $\vdash (di(halt(init\ w) \wedge f) \wedge di(halt(init\ w) \wedge g)) =$
$di(halt(init\ w) \wedge f \wedge g)$
using *DiHaltAndDiHaltAndEqvDiHaltAndAnd* **by** *fastforce*
have *4*: $\vdash di(\ halt(init\ w) \wedge f \wedge g) = di(\mathcal{M}(HALT\ w) \wedge f \wedge g)$
by (*metis DiEqvDi MHaltAlt inteq-reflection lift-and-com*)
have *5*: $\vdash (di(\mathcal{M}(HALT\ w) \wedge f) \wedge di(\mathcal{M}(HALT\ w) \wedge g)) = di(\mathcal{M}(HALT\ w) \wedge f \wedge g)$
using *2 3 4* **by** *fastforce*
have *6*: $\vdash \triangleright(di(\mathcal{M}(HALT\ w) \wedge f) \wedge di(\mathcal{M}(HALT\ w) \wedge g)) = \triangleright(di(\mathcal{M}(HALT\ w) \wedge f \wedge g))$
using *5 FstEqvRule* **by** *blast*
have *7*: $\vdash \triangleright(di(\mathcal{M}(HALT\ w) \wedge f \wedge g)) = \triangleright(\mathcal{M}(HALT\ w) \wedge f \wedge g)$
using *FstDiEqvFst* **by** *fastforce*
have *8*: $\vdash \triangleright(\mathcal{M}(HALT\ w) \wedge f \wedge g) = \triangleright(\mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)))$
by *simp*
have *9*: $\vdash \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)) = \triangleright(\mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)))$
using *MFixFst* **by** *blast*
have *10*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ THRU\ ((HALT\ w)\ WITH\ g)) = \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g))$
using *1 2 3 4 5 6 7 8 9 int-eq* **by** *metis*
have *11*: $\vdash \mathcal{M}(((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g)) = \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g))$
using *MHaltWithAndDistrib* **using** *eq-d-def* **by** *blast*
have *12*: $\vdash \mathcal{M}((HALT\ w)\ WITH\ LIFT(f \wedge g)) = \mathcal{M}(((HALT\ w)\ WITH\ f)\ AND\ ((HALT\ w)\ WITH\ g))$
using *11* **by** *fastforce*
from *10 12* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThenAndDistrib*:
$(a\ THEN\ (b\ AND\ c)) \simeq ((a\ THEN\ b)\ AND\ (a\ THEN\ c))$
proof −
have *1*: $\vdash \mathcal{M}(a\ THEN\ (b\ AND\ c)) = (\mathcal{M}(a))\ ;\ (\mathcal{M}(b\ AND\ c))$
by *simp*

have *2*: $\vdash (\mathcal{M}(a))\ ;\ (\mathcal{M}(b\ AND\ c)) = (\mathcal{M}(a))\ ;\ (\mathcal{M}(b) \wedge \mathcal{M}(c))$
by (*simp add*: *AND-d-def*)
have *3*: $\vdash (\mathcal{M}(a))\ ;\ (\mathcal{M}(b) \wedge \mathcal{M}(c)) = \ \rhd\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(b) \wedge \mathcal{M}(c))$
using *MFixFst LeftChopEqvChop* **by** *blast*
have *4*: $\vdash \rhd\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(b) \wedge \mathcal{M}(c)) = \ ((\rhd\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(b))) \wedge (\rhd\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(c))))$
using *LFstAndDistrC* **by** *fastforce*
have *5*: $\vdash((\rhd\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(b))) \wedge (\rhd\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(c)))) =$
$((\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(b))) \wedge (\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(c)))\ )$ **using** *MFixFst*
by (*metis 4 inteq-reflection*)
have *6*: $\vdash ((\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(b))) \wedge (\ (\mathcal{M}(a))\ ;\ (\mathcal{M}(c)))\ ) =$
$(\mathcal{M}(a\ THEN\ b) \wedge \mathcal{M}(a\ THEN\ c))$
by *simp*
have *7*: $\vdash (\mathcal{M}(a\ THEN\ b) \wedge \mathcal{M}(a\ THEN\ c)) = \mathcal{M}((a\ THEN\ b)\ AND\ (a\ THEN\ c))$
by (*simp add*: *AND-d-def*)
from *1 2 3 4 5 6 7* **show** *?thesis* **using** *MonEq* **by** (*metis int-eq*)
qed

lemma *MThenUptoDistrib*:
$(a\ THEN\ (b\ UPTO\ c)) \simeq ((a\ THEN\ b)\ UPTO\ (a\ THEN\ c))$
proof −
have *1*: $\vdash (\mathcal{M}\ (a\ THEN\ (b\ UPTO\ c))) = ((\mathcal{M}\ a);(\rhd((\mathcal{M}\ b) \vee (\mathcal{M}\ c))))$
by *simp*
have *2*: $\vdash ((\mathcal{M}\ a);(\rhd((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))) = (\rhd(\mathcal{M}\ a);(\rhd((\mathcal{M}\ b) \vee (\mathcal{M}\ c))))$
by (*simp add*: *MFixFst LeftChopEqvChop*)
have *3*: $\vdash (\rhd(\mathcal{M}\ a);(\rhd((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))) = ((\rhd(\rhd(\mathcal{M}\ a);((\mathcal{M}\ b) \vee (\mathcal{M}\ c)))))$
using *FstFstChopEqvFstChopFst* **by** *fastforce*
have *4*: $\vdash \rhd(\mathcal{M}\ a);((\mathcal{M}\ b) \vee (\mathcal{M}\ c)) = (\mathcal{M}\ a);((\mathcal{M}\ b) \vee (\mathcal{M}\ c))$
using *MFixFst* **by** (*metis LeftChopEqvChop inteq-reflection*)
have *5*: $\vdash (\mathcal{M}\ a);((\mathcal{M}\ b) \vee (\mathcal{M}\ c)) = ((\mathcal{M}\ a);(\mathcal{M}\ b) \vee (\mathcal{M}\ a);(\mathcal{M}\ c))$
by (*simp add*: *ChopOrEqv*)
have *6*: $\vdash ((\mathcal{M}\ a);(\mathcal{M}\ b) \vee (\mathcal{M}\ a);(\mathcal{M}\ c)) = (\mathcal{M}(a\ THEN\ b) \vee \mathcal{M}(a\ THEN\ c))$
by *simp*
have *7*: $\vdash \rhd(\mathcal{M}\ a);((\mathcal{M}\ b) \vee (\mathcal{M}\ c)) = (\mathcal{M}(a\ THEN\ b) \vee \mathcal{M}(a\ THEN\ c))$
using *6 5 4* **by** *fastforce*
have *8*: $\vdash \rhd(\rhd(\mathcal{M}\ a);((\mathcal{M}\ b) \vee (\mathcal{M}\ c))) = \rhd(\mathcal{M}(a\ THEN\ b) \vee \mathcal{M}(a\ THEN\ c))$
using *7* **by** (*simp add*: *FstEqvRule*)
have *9*: $\vdash \rhd(\mathcal{M}(a\ THEN\ b) \vee \mathcal{M}(a\ THEN\ c)) = \mathcal{M}((a\ THEN\ b)\ UPTO\ (a\ THEN\ c))$
by *simp*
from *9 7 1 2 3* **show** *?thesis* **by** (*metis eq-d-def inteq-reflection*)
qed

lemma *MThenThruDistrib*:
$(a\ THEN\ (b\ THRU\ c)) \simeq ((a\ THEN\ b)\ THRU\ (a\ THEN\ c))$
proof −
have *1*: $\vdash \mathcal{M}(a\ THEN\ (b\ THRU\ c)) = (\mathcal{M}\ a);\rhd(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))$
by *simp*
have *2*: $\vdash (\mathcal{M}\ a);\rhd(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)) = \rhd(\mathcal{M}\ a);\rhd(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))$
by (*simp add*: *MFixFst LeftChopEqvChop*)
have *3*: $\vdash \rhd(\mathcal{M}\ a);\rhd(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)) = \rhd(\rhd(\mathcal{M}\ a);(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c))\ ))$
using *FstFstChopEqvFstChopFst* **by** *fastforce*

have *4*: $\vdash \triangleright(\mathcal{M}\ a);(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)\ ) = (\triangleright(\mathcal{M}\ a);di(\mathcal{M}\ b) \wedge \triangleright(\mathcal{M}\ a);di(\mathcal{M}\ c))$
by (*meson LFstAndDistrC Prop11*)
have *5*: $\vdash (\triangleright(\mathcal{M}\ a);di(\mathcal{M}\ b) \wedge \triangleright(\mathcal{M}\ a);di(\mathcal{M}\ c)) = ((\mathcal{M}\ a);di(\mathcal{M}\ b) \wedge (\mathcal{M}\ a);di(\mathcal{M}\ c))$
using *MFixFst* **by** (*metis 4 int-eq*)
have *6*: $\vdash (\mathcal{M}\ a);di(\mathcal{M}\ b) = (\mathcal{M}\ a);((\mathcal{M}\ b);\#True)$
by (*simp add*: *di-d-def*)
have *7*: $\vdash (\mathcal{M}\ a);((\mathcal{M}\ b);\#True) = ((\mathcal{M}\ a);(\mathcal{M}\ b));\#True$
by (*simp add*: *ChopAssoc*)
have *8*: $\vdash ((\mathcal{M}\ a);(\mathcal{M}\ b));\#True = di((\mathcal{M}\ a);(\mathcal{M}\ b))$
by (*simp add*: *di-d-def*)
have *9*: $\vdash (\mathcal{M}\ a);di(\mathcal{M}\ b) = di((\mathcal{M}\ a);(\mathcal{M}\ b))$
using *8 7 6* **by** *fastforce*
have *10*: $\vdash (\mathcal{M}\ a);di(\mathcal{M}\ c) = (\mathcal{M}\ a);((\mathcal{M}\ c);\#True)$
by (*simp add*: *di-d-def*)
have *11*: $\vdash (\mathcal{M}\ a);((\mathcal{M}\ c);\#True) = ((\mathcal{M}\ a);(\mathcal{M}\ c));\#True$
by (*simp add*: *ChopAssoc*)
have *12*: $\vdash ((\mathcal{M}\ a);(\mathcal{M}\ c));\#True = di((\mathcal{M}\ a);(\mathcal{M}\ c))$
by (*simp add*: *di-d-def*)
have *13*: $\vdash (\mathcal{M}\ a);di(\mathcal{M}\ c) = di((\mathcal{M}\ a);(\mathcal{M}\ c))$
using *12 11 10* **by** *fastforce*
have *14*: $\vdash ((\mathcal{M}\ a);di(\mathcal{M}\ b) \wedge (\mathcal{M}\ a);di(\mathcal{M}\ c)) = (di((\mathcal{M}\ a);(\mathcal{M}\ b)) \wedge di((\mathcal{M}\ a);(\mathcal{M}\ c)))$
using *13 9* **by** *fastforce*
have *15*: $\vdash (di((\mathcal{M}\ a);(\mathcal{M}\ b)) \wedge di((\mathcal{M}\ a);(\mathcal{M}\ c))) = (di(\mathcal{M}(a\ THEN\ b)) \wedge di(\mathcal{M}(a\ THEN\ c)))$
by *simp*
have *16*: $\vdash \triangleright(\mathcal{M}\ a);(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)\ ) = (di(\mathcal{M}(a\ THEN\ b)) \wedge di(\mathcal{M}(a\ THEN\ c)))$
using *15 14 4 5* **by** *fastforce*
have *17*: $\vdash \triangleright(\triangleright(\mathcal{M}\ a);(di(\mathcal{M}\ b) \wedge di(\mathcal{M}\ c)\ )) = \triangleright(di(\mathcal{M}(a\ THEN\ b)) \wedge di(\mathcal{M}(a\ THEN\ c)))$
using *16* **by** (*simp add*: *FstEqvRule*)
have *18*: $\vdash \triangleright(di(\mathcal{M}(a\ THEN\ b)) \wedge di(\mathcal{M}(a\ THEN\ c))) = \mathcal{M}((a\ THEN\ b)\ THRU\ (a\ THEN\ c))$
by *simp*
from *18 16 1 2 3* **show** *?thesis* **by** (*metis eq-d-def int-eq*)
qed

end

theory *Example*
imports
FOTheorems
begin

8 Examples

8.1 Example 1

definition *F1* :: *nat statefun* ⇒ *temporal*
where *F1 w* ≡ *TEMP* □ (*#0* ≤ $*w*)

definition *Init1* :: *nat statefun* ⇒ *temporal*
where *Init1 w* ≡ *TEMP* $*w* = #*0*

lemma *init1*:
(⟨*s0*,*s1*,*s2*⟩ ⊨ *len*(*2*) ∧ *Init1 w*) = ((*w s0*) = *0*)
by (*simp add*: *Init1-def current-val-d-def len-defs*)

lemma *exist-test-F1* :
⊢ ∃∃ *w*. *F1 w*
proof −
have *1*: ⋀ *w*. ⊢ *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

8.2 Example 2

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* ≡ (λ *v*. *TEMP* □ (#*0* ≤ $*v*))
defines *F3* ≡ (λ *p*. *TEMP* □ ($*p* ∨ ¬ $*p*))
defines *F4* ≡ (λ *z*. *TEMP* □ (#*0*≤ $*z* ∨ $*z* < #*0*))
defines *F5* ≡ (λ *v*. *TEMP* $*v*=#*0* ∧ *v gets* $*v*+#*1*)
defines *Init2* ≡ (λ *v*. *TEMP* $*v* = #*0*)
defines *Init3* ≡ (λ *p*. *TEMP* $*p*)

lemma (**in** *Test*) *currentval-test* :
(*s* ⊨ ($*v* = #*0*)) = ((*v* (*nth s 0*)) = *0*)
by (*simp add*: *current-val-d-def*)

lemma (**in** *Test*) *nextempty-test* :
(⟨*s0*⟩ ⊨ *v*$) = (ϵ *x*. *x*=*x*)
by (*simp add*: *next-val-d-def*)

lemma (**in** *Test*) *nextempty-test-1* :
(⟨*s0*⟩ ⊨ *v*$ = *v*$)
by *simp*

lemma (**in** *Test*) *nextempty-test-2* :
(⟨*s0*⟩ ⊨ *v*$ = *v1*$)

by (*simp add*: *Test.nextempty-test*)

lemma (**in** *Test*) *nextcurrent-test*:
(⟨*s0*,*s1*⟩ ⊨ *skip* ∧ ($*v*=#*0*) ∧ (*v*$=$*v*+#*1*)) = (((*v s0*) = *0*) ∧ ((*v s1*) = *1*))
unfolding *current-val-d-def next-val-d-def skip-defs* **by** *auto*

lemma (**in** *Test*) *nextcurrentfinpenult-test*:
(⟨*s0*,*s1*,*s2*,*s3*⟩ ⊨ *len*(*3*) ∧ *v* =: !*v*−#*1* ∧ *v* ← #*3* ∧ $*v*=#*0* ∧ *v* := $*v*+#*1*) =
(((*v s0*) = *0*) ∧ ((*v s1*) = *1* ∧ (*v* (*s2*)) = *2* ∧ ((*v s3*) = *3*)))
unfolding *current-val-d-def next-val-d-def fin-val-d-def penult-val-d-def*
next-assign-d-def prev-assign-d-def temporal-assign-d-def len-defs **by** *auto*

lemma (**in** *Test*) *stable-test*:
(⟨*s0*,*s1*,*s2*,*s3*⟩ ⊨ *len*(*3*) ∧ *stable v* ∧ $*v*=#*0*) =
((*v s0*) = *0* ∧ (*v s1*) = *0* ∧ (*v s2*) = *0* ∧ (*v s3*) = *0*)
by (*auto simp*: *stable-defs len-defs*
current-val-d-def next-val-d-def Nitpick.case-nat-unfold)

lemma (**in** *Test*) *revnextcurrentfinpenult-test*:
(⟨*s0*,*s1*,*s2*,*s3*⟩ ⊨ (*len 3* ∧ *v*! = !*v*−#*1* ∧ !*v* = #*3* ∧ $*v*=#*0* ∧ *v*$=$*v*+#*1*)^r) =
(((*v s3*) = *0*) ∧ ((*v s2*) = *1* ∧ (*v* (*s1*)) = *2* ∧ ((*v s0*) = *3*)))
unfolding *reverse-d-def len-defs current-val-d-def next-val-d-def*
penult-val-d-def fin-val-d-def **by** *auto*

lemma (**in** *Test*) *exist-test-F2* :
⊢ ∃∃ *v*. *F2 v*
proof −
have *1*: ⊢ *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* :
⊢ ∃∃ *y*. *F3 y*
proof −
have *1*: ⊢ *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

8.3 Example 3

locale *Test1* =
fixes *v* :: *state* ⇒ *nat*
fixes *F5* :: *nat statefun* ⇒ *nat* ⇒ *temporal*
defines *F5* ≡ (λ *v n*. *TEMP* $*v*=#*0* ∧ *v gets* $*v*+#*1* ∧ *fin*($*v*=#*n*))

lemma (**in** *Test1*) *test-E-F5-1*:
(
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)
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

8.4 Example 4

locale *Testrev* =
 fixes *x* :: *state* ⇒ *nat*
 fixes *F1* :: *nat statefun* ⇒ *temporal*
 defines *F1* ≡ (λ *v*. *TEMP* $*v*=#*0* ∧ *skip* ∧ *v*:= $*v*+#*1*)

lemma (**in** *Testrev*) *testrev1*:

$(\sigma \models F1\ (x)) = (intlen\ \sigma = 1 \wedge (x\ (nth\ \sigma\ 0)) = 0 \wedge (x\ (nth\ \sigma\ 1)) = 1)$
by (*simp add*: *F1-def skip-defs next-assign-d-def next-val-d-def current-val-d-def* , *auto*)

lemma (**in** *Testrev*) *testrev2*:
$(\sigma \models (F1\ (x))^r) = (intlen\ \sigma = 1 \wedge (x\ (nth\ \sigma\ 0)) = 1 \wedge (x\ (nth\ \sigma\ 1)) = 0)$
proof −
have $(\sigma \models (F1\ (x))^r) = (\sigma \models (\$x{=}\#0 \wedge skip \wedge x := \$x{+}\#1)^r)$
by (*simp add*: *F1-def*)
also have ... =
$(\sigma \models ((\$x{=}\#0)^r \wedge skip^r \wedge (x := \$x{+}\#1)^r))$
by (*simp add*: *all-rev-eq*)
also have ... =
$(\sigma \models ((!x{=}\#0) \wedge skip \wedge (x! = !x{+}\#1)))$
by (*smt RRAnd all-rev-eq*(*1*) *all-rev-eq*(*10*) *all-rev-eq*(*11*) *all-rev-eq*(*12*)
all-rev-eq(*3*) *int-eq next-assign-d-def*)
also have ... =
$(\sigma \models ((x\${=}\#0) \wedge skip \wedge (\$x = x\${+}\#1)))$
by (*simp add*: *skip-defs next-val-d-def finval-defs penultval-defs current-val-d-def* , *auto*)
also have ... =
$(intlen\ \sigma = 1 \wedge (x\ (nth\ \sigma\ 0)) = 1 \wedge (x\ (nth\ \sigma\ 1)) = 0)$
by (*simp add*: *skip-defs next-val-d-def current-val-d-def* , *auto*)
finally show $(\sigma \models (F1\ (x))^r) = (intlen\ \sigma = 1 \wedge (x\ (nth\ \sigma\ 0)) = 1 \wedge (x\ (nth\ \sigma\ 1)) = 0)$.
qed

8.5 Example 5

lemma *revnextcurrentfinpenult*:
$\vdash\ (v\$ = \$v)^r = (v!{=}!v)$
proof −
have *1*: $\vdash (v\$ = \$v)^r = (\ (v\$)^r = (\$v)^r)$ **by** (*simp add*: *rev-fun2*)
have *2*: $\vdash ((v\$)^r = \ (v!))$ **by** (*simp add*: *rev-next*)
have *3*: $\vdash ((\$v)^r = \ (!v))$ **by** (*simp add*: *rev-current*)
have *4*: $\vdash\ ((\ (v\$)^r = (\$v)^r) = (\ (v!) = (!v)\ ))$ **by** (*metis 1 2 3 inteq-reflection*)
from *1 4* **show** *?thesis* **by** *fastforce*
qed

end

theory *MonitorExample*
imports
FOTheorems Monitor
begin

9 Example

locale *Test* =
fixes *v* :: *state* ⇒ *nat*

fixes $y :: state \Rightarrow bool$
fixes $z :: state \Rightarrow nat$
fixes $F2 :: nat\ statefun \Rightarrow temporal$
fixes $F3 :: bool\ statefun \Rightarrow temporal$
fixes $F4 :: nat\ statefun \Rightarrow temporal$
fixes $F5 :: nat\ statefun \Rightarrow temporal$
fixes $Init2 :: nat\ statefun \Rightarrow temporal$
fixes $Init3 :: bool\ statefun \Rightarrow temporal$
fixes $Mon1 :: state\ \ monitor$
fixes $Mon2 :: state\ monitor$
fixes $Mon3 :: state\ monitor$
fixes $Mon4 :: state\ monitor$
fixes $Mon5 :: state\ monitor$
fixes $Mon6 :: state\ monitor$
defines $F2 \equiv (\lambda\ v.\ TEMP\ \Box\ (\ \#0 \leq \$v\ ))$
defines $F3 \equiv (\lambda\ p.\ TEMP\ \Box\ (\ \$p \vee \neg\ \$p))$
defines $F4 \equiv (\lambda\ z.\ TEMP\ \$z{=}\#0 \wedge z\ gets\ \$z{+}\#1)$
defines $F5 \equiv (\lambda\ z.\ TEMP\ fin(\$z{=}\#4))$
defines $Init2 \equiv (\lambda\ v.\ TEMP\ \$v = \#0)$
defines $Init3 \equiv (\lambda\ p.\ TEMP\ \$p)$
defines $Mon1 \equiv FIRST(\ F2\ v\ )$
defines $Mon2 \equiv EMPTY\ UPTO\ Mon1$
defines $Mon3 \equiv Mon1\ WITH\ (F2\ v)$
defines $Mon4 \equiv Mon2\ THEN\ Mon1$
defines $Mon5 \equiv Mon3\ THRU\ Mon4$
defines $Mon6 \equiv (FIRST\ F4\ z)\ WITH\ (F5\ z)$

lemma (**in** *Test*) *test*:
$\vdash \mathcal{M}(Mon1) =\ empty$
proof −
have *1*: $\vdash \mathcal{M}(Mon1) = \triangleright(\Box\ (\ \#0 \leq \$v\ ))$
using *F2-def Mon1-def* **by** *fastforce*
have *2*: $\vdash \Box\ (\ \#0 \leq \$v\ )$
by (*simp add*: *Valid-def always-defs current-val-d-def*)
have *3*: $\vdash \triangleright(\Box\ (\ \#0 \leq \$v\ )) = empty$
using *2* **by** (*metis FstTrue int-eq int-eq-true*)
from *1 2 3* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test1*:
$\vdash \mathcal{M}(Mon2) = empty$
proof −
have *1*: $\vdash \mathcal{M}(Mon2) = \mathcal{M}(EMPTY\ UPTO\ Mon1)$
using *Mon2-def* **by** *fastforce*
have *2*: $\vdash \mathcal{M}(EMPTY\ UPTO\ Mon1) = \triangleright(\mathcal{M}(EMPTY) \vee \mathcal{M}(Mon1))$
by *fastforce*
have *3*: $\vdash \triangleright(\mathcal{M}(EMPTY) \vee \mathcal{M}(Mon1)) = \triangleright(empty \vee empty)$
using *test* **by** (*metis 2 MEmptyAlt int-eq*)
have *4*: $\vdash \triangleright(empty \vee empty) = empty$
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*: $\vdash \mathcal{M}(Mon3\ THRU\ Mon4) = \triangleright(di(\mathcal{M}(Mon3)) \wedge di(\mathcal{M}(Mon4)))$
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*: $\vdash \triangleright(di(\mathcal{M}(Mon3)) \wedge di(\mathcal{M}(Mon4))) = \triangleright(di(empty) \wedge di(empty))$
by (*simp add*: *FstEqvRule*)
have *5*: $\vdash \triangleright(di(empty) \wedge di(empty)) = \triangleright(di(empty))$
by *simp*
have *6*: $\vdash \triangleright(di(empty)) = empty$
using *FstDiEqvFst FstEmpty* **by** *fastforce*
from *6 5 4 2 1* **show** *?thesis* **by** *fastforce*
qed

lemma (**in** *Test*) *test5*:
$\vdash \mathcal{M}\ (Mon6) = (\triangleright(\$z{=}\#0\ \wedge\ z\ gets\ \$z{+}\#1) \wedge fin(\$z{=}\#4)\ )$
proof −
have *1*: $\vdash \mathcal{M}(Mon6) = (\mathcal{M}(FIRST\ F4\ z) \wedge (F5\ z))$

```
    using Mon6-def by fastforce
 have 2: ⊢ (M(FIRST F4 z) ∧ (F5 z)) = (▷(F4 z) ∧ 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* ($z=\#0 \wedge z$ *gets* $\$z+\#1) \wedge$ *fin*($\$z=\#4$)]
by *simp*

lemma (**in** *Test*) *test5-9* :
$\neg($ $(s \models (\$z=\#0 \wedge z$ *gets* $\$z+\#1) \wedge$ *fin*($\$z=\#4$)) $\wedge$ *intlen* $s = 0)$
using *test5-7* **by** *simp*

lemma (**in** *Test*) *test5-10*:
$(s \models (\$z=\#0 \wedge z$ *gets* $\$z+\#1) \wedge$ *fin*($\$z=\#4$))
$\Longrightarrow$
$0 <$ *intlen* $s \wedge$
$(\forall\ ia <$ *intlen* $s.$ (*prefix* $ia\ s \models \neg((\$z=\#0 \wedge z$ *gets* $\$z+\#1) \wedge$ *fin*($\$z=\#4$))))

proof −
assume *0*: $s \models (\$z=\#0 \wedge z$ *gets* $\$z+\#1) \wedge$ *fin*($\$z=\#4$)
show $0 <$ *intlen* $s \wedge$
$(\forall\ ia <$ *intlen* $s.$ (*prefix* $ia\ s \models \neg((\$z=\#0 \wedge z$ *gets* $\$z+\#1) \wedge$ *fin*($\$z=\#4$))))
proof −
have *1*: $0 <$ *intlen* s **using** *test5-7 0* **by** *simp*
have *2*: $(\forall\ ia <$ *intlen* $s.$ (*prefix* $ia\ s \models \neg((\$z=\#0 \wedge z$ *gets* $\$z+\#1) \wedge$ *fin*($\$z=\#4$))))
proof
fix *ia*
show $ia <$ *intlen* $s \longrightarrow$
(*prefix* $ia\ s \models \neg\ ((\$z = \#0 \wedge z$ *gets* $\$z + \#1) \wedge$ *fin* $(\$z = \#4)))$
proof −
have *1*: (*prefix* $ia\ s \models \neg\ ((\$z = \#0 \wedge z$ *gets* $\$z + \#1) \wedge$ *fin* $(\$z = \#4))) =$
$(\neg((prefix\ ia\ s \models\ ((\$z = \#0 \wedge z$ *gets* $\$z + \#1) \wedge$ *fin* $(\$z = \#4)))))$
by *auto*
have *2*: (*prefix* $ia\ s \models\ ((\$z = \#0 \wedge z$ *gets* $\$z + \#1) \wedge$ *fin* $(\$z = \#4))) =$
$(intlen\ (prefix\ ia\ s) = 4 \wedge (\forall\ i \leq intlen(prefix\ ia\ s)\ .\ z\ (nth\ (prefix\ ia\ s)\ i) = i))$
using *test5-7* **by** *simp*
have *3*: $ia <$ *intlen* $s \longrightarrow \neg(intlen\ (prefix\ ia\ s) = 4 \wedge$
$(\forall\ i \leq intlen(prefix\ ia\ s)\ .\ z\ (nth\ (prefix\ ia\ s)\ i) = i))$
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* :
$(s \models \triangleright((\$z=\#0 \wedge z$ *gets* $\$z+\#1) \wedge$ *fin*($\$z=\#4$))) =
$(s \models (\$z=\#0 \wedge z$ *gets* $\$z+\#1) \wedge$ *fin*($\$z=\#4$))
using *test5-8 test5-9 test5-10* **by** *fastforce*

lemma (**in** *Test*) *test5-12* :
$\vdash \triangleright((\$z=\#0 \wedge z$ *gets* $\$z+\#1) \wedge$ *fin*($\$z=\#4$)) $= ((\$z=\#0 \wedge z$ *gets* $\$z+\#1) \wedge$ *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] S. Merz. An Encoding of TLA in Isabelle. http://www.pst.informatik.uni-muenchen.de/~merz/isabelle/. Part of the Isabelle distribution., 1998.

[4] B. Moszkowski. A Hierarchical Completeness Proof for Propositional Interval Temporal Logic with Finite Time. *Journal of Applied Non-Classical Logics*, 14(1–2):55–104, 2004.

[5] B. C. Moszkowski. Imperative reasoning in interval temporal logic. Technical report, University of Newcastle upon Tyne, 1996.

[6] 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.